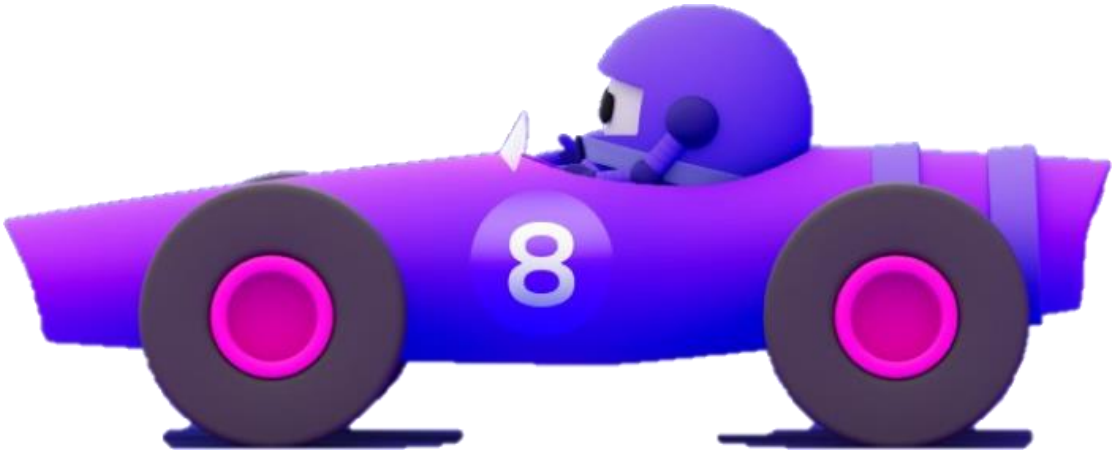
C Sharp بالعربي

أساسيات ومفاهيم البرمجة

لغة C#

1

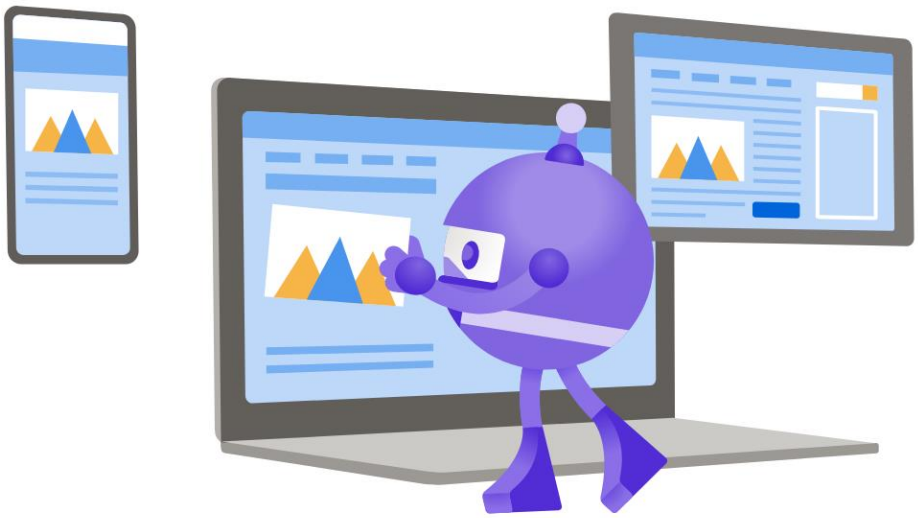


مجتمع المبرمجين العرب

هل أنت مهتم بتعلم لغة برمجة C# ولكنك غير متأكد من أين تبدأ؟ أبدأ هنا! تعرف على عمليات بناء الجملة البرمجية، والأفكار الأساسية اللازمة لإنشاء تطبيقات صغيرة، باستخدام لغة C#

ابدأ مع C#

ابدأ رحلتك التعليمية لـ C# من خلال هذا المسار التعليمي المكون من ستة فصول، يأخذك في رحلة من البداية إلى النهاية في تعلم لغة البرمجة C# حتى تتمكن من البدء في إنشاء التطبيقات، باستخدام C# و .NET.



الفصل الأول

اكتب التعليمات البرمجية الأولى باستخدام C#

تعلم بناء الجملة الأساسية، وعمليات التفكير اللازمة لإنشاء تطبيقات بسيطة باستخدام C#

المحتويات

الوحدة الأولي

اكتب التعليمات البرمجية الأولى باستخدام C#

ابدأ بكتابة أمثلة التعليمات البرمجية، لتتعلم أساسيات بناء جملة C#

الأهداف التعليمية

- كتابة الأسطر الأولى من التعليمات البرمجية بلغة C#
- استخدم تقنيتين مختلفتين لطباعة رسالة.
- تشخيص الأخطاء عندما تكون التعليمات البرمجية مكتوبة بشكل غير صحيح.
- تعريف العناصر المختلفة لبناء جملة syntax elements لغة C# مثل عوامل التشغيل operators والفئات classes والأساليب methods

محتويات الوحدة:

١ المقدمة

لغة البرمجة C# تسمح لك بإنشاء العديد من أنواع التطبيقات، مثل:

- تطبيقات الأعمال Business applications لالتقاط البيانات وتحليلها ومعالجتها.
- تطبيقات الويب التفاعلية Dynamic web applications التي يمكن الوصول إليها من مستعرض ويب.
- الألعاب Games بنوعيها ثنائية الأبعاد وثلاثية الأبعاد.
- التطبيقات المالية والعلمية Financial and scientific applications
- التطبيقات المعتمدة على السحابة Cloud-based applications
- تطبيقات الجوال Mobile applications

كيف تبدأ في إنشاء تطبيق؟

تتكون جميع التطبيقات من العديد من أسطر التعليمات أو الرموز البرمجية lines of code التي تعمل معاً لتحقيق مهمة محددة، حتى الآن، أفضل طريقة لفهم التعليمات البرمجية كتابتها بيدك، بمعني اخر ممارسة البرمجة عملياً، من المشجع أن تكتب التعليمات البرمجية أثناء التدريبات في هذه الوحدة، وغيرها في مسار التعلم هذا، كتابة التعليمات بنفسك في كل تمرين سيسرع تعلمك، وحل تحديات الترميز الصغيرة، كل هذا سوف يشجعك على الاستمرار، وتعلم المزيد عن اللغة.

سوف تبدأ أيضاً في تعلم المفاهيم الأساسية الصغيرة، والبناء عليها من خلال الممارسة والاستكشاف المستمرين.

في نهاية هذه الوحدة، ستتمكن من كتابة التعليمات البرمجية بلغة C# لطباعة رسالة إلى جهاز الإخراج، مثل شاشة أو طابعة، وهذه الأسطر من التعليمات البرمجية سوف تعطيك أول نظرة على بناء جملة لغة C# لتقديم رؤى لا تقدر بثمن.

٢ تمرين ـ كتابة التعليمات البرمجية الأولى

في هذا التمرين العملي الأول، ستستخدم C# لطباعة العبارة الشهيرة للمبرمجين على الشاشة.

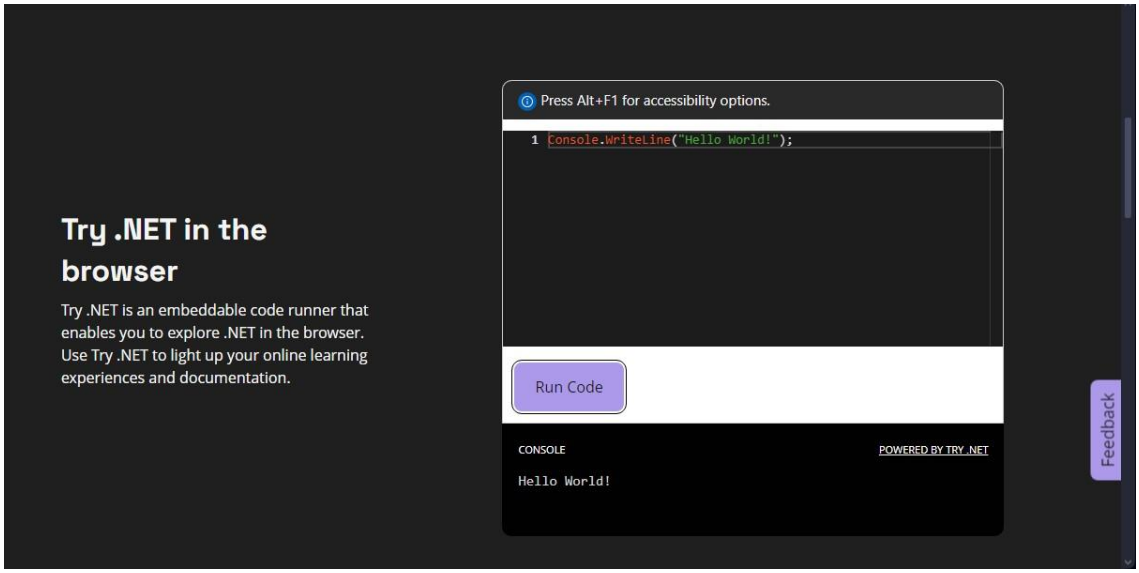
اكتب السطر الأول من التعليمات البرمجية

هناك تقليد قديم بين مطوري البرامج لطباعة العبارة **"مرحبًا بالعالم!** Hello World!" إلى نافذة الإخراج، كما سترى، يمكنك معرفة الكثير عن البرمجة ولغة C# من هذا التمرين البسيط.

ملاحظة

إذا لم تقم بتنصب برنامج Visual Studio Code (سوف نشرح تثبيت البرنامج ونتعرف على الواجهة في الفصل الثاني) يمكنك الكتابة في أحد محرري المستعرض، مثل العنوان التالي TrydotNet يعتبر محرر .NET. ووحدة تحكم الإخراج في المستعرض تجربة رائعة، مثالية لهذا النهج التعليمي، حيث يقع محرر .NET. داخل صفحة الويب، ووحدة تحكم الإخراج أسفلها، كي تري مباشرة نتيجة عملك.

أمسح جميع التعليمات الموجودة داخل المحرر.



أدخل التعليمات البرمجية في محرر .NET.

١ـ أدخل هذه التعليمة البرمجية تماما كما تظهر في المحرر.

```
Console.WriteLine("Hello World!");
```

سترى شرحاً وافياً لكيفية عملها، وسبب عملها قريباً، ولكن أولاً، يجب أن تجرب تشغيلها، وتأكد من إدخالها بالشكل الصحيح، للتأكد من ذلك، عليك تشغيل التعليمة في المحرر.

ملاحظة

قد تميل إلى تحديد Copy أو Run وتخطي كل ضغطات المفاتيح، ومع ذلك، هناك فوائد لكتابة التعليمات البرمجية بنفسك، إدخال التعليمات البرمجية بنفسك يعزز الذاكرة، والفهم الذي سيساعدك على الحصول على رؤى لن تحصل عليها بخلاف ذلك.

تشغيل التعليمات البرمجية الأولى

١. الضغط على الزر تشغيل Run

يقوم الزر «Run» بمهمتين:

- تحويل التعليمات البرمجية إلى تنسيق قابل للتنفيذ، يمكن أن يفهمه جهاز الكمبيوتر.
- يقوم بتشغيل التطبيق المحول برمجياً، وعند كتابته بشكل صحيح، سيتم إخراج "Hello World!"

مراقبة النتائج

في وحدة تحكم الإخراج، لاحظ نتيجة التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

Hello World!

ما يجب تنفيذه في حالة رؤية رسالة خطأ

تُعد كتابة التعليمات البرمجية بلغة C# تمرينًا غاية في الدقة، إذا كتبت حرفاً واحدًا فقط بشكل غير صحيح، فستتلقى رسالة خطأ في منطقة الإخراج عند تشغيل التعليمات البرمجية.

على سبيل المثال، إذا قمت بإدخال أحرف صغيرة بشكل غير صحيح "c" في الكلمة Console مثل ما يلي:

```
console.WriteLine("Hello World!");
```

تظهر رسالة الخطأ التالية:

```
(1,1): error CS0103: The name 'console' does not exist in the current context
```

يشير الجزء الأول (1,1) إلى السطر والعمود حيثما حدث الخطأ، ولكن ماذا تعني رسالة الخطأ هذه؟

إن لغة C# حساسة لحالة الأحرف، بمعنى أن المحول البرمجي "المترجم" للغة C# يعتبر الكلمات console و Console مختلفة مثل الكلمات cat و dog في بعض الأحيان يمكن أن تكون رسالة الخطأ مضللة بعض الشيء، ستحتاج إلى فهم السبب الحقيقي لوجود الخطأ، وهذا يأتي من خلال معرفة المزيد عن بناء جملة لغة C# وهذا ما سنشرحه، فلا تقلق.

وبالمثل، إذا استخدمت علامات اقتباس مفردة (') لإحاطة السلسلة Hello World! الحرفية كما يلي:

```
Console.WriteLine('Hello World!');
```

قد تظهر رسالة الخطأ التالية:

```
(1,19): error CS1012: Too many characters in character literal
```

مرة أخرى، في السطر ١ يشير الحرف ١٩ إلى المذنب، يمكنك استخدام الرسالة كدليل بينما تحقق في المشكلة، ولكن ماذا تعني رسالة الخطأ؟ ما هو بالضبط **"حرف زائد في الجملة الحرفية؟** `Too many characters in character literal`" لاحقاً سوف تتعرف على المزيد حول القيم الحرفية، لمختلف أنواع البيانات (بما في ذلك القيمة الحرفية للأحرف نفسها) في الوقت الحالي، كن حذراً عند إدخال التعليمات البرمجية.

لحسن الحظ، فإن الأخطاء ليست دائمة الحدوث، يمكنك فقط اكتشاف الخطأ وإصلاحه، ثم إعادة تشغيل تعليماتك البرمجية.

إذا تلقيت خطأ مرة أخرى عند تشغيل التعليمات البرمجية، فخذ لحظة للنظر إليها عن كثب، افحص كل حرف، تأكد من إدخال هذا السطر البرمجي بالطريقة الصحيحة.

ملاحظة

يقوم المحرر بمراقبة التعليمات البرمجية التي تكتبها باستمرار، عن طريق إجراء عملية التحويل البرمجي المسبق للبحث عن الأخطاء المحتملة، سيحاول مساعدتك عن طريق إضافة خطوط متعرجة حمراء أسفل التعليمات البرمجية التي ستنتج خطأ.

الأخطاء الشائعة للمبرمجين الجدد:

- إدخال أحرف صغيرة بدلاً من الأحرف الكبيرة "**C**" في Console أو الأحرف W أو L في WriteLine
- إدخال فاصلة بدلاً من نقطة بين كلمات Console,WriteLine
- نسيان استخدام علامات اقتباس مزدوجة "" أو استخدام علامات اقتباس مفردة لإحاطة العبارة 'Hello World!'
- نسيان فاصلة منقوطة (;) في نهاية السطر البرمجي

كل من هذه الأخطاء يمنع التعليمات البرمجية من التحويل البرمجي بنجاح.

يسلط محرر التعليمات البرمجية The code editor الضوء على أخطاء ما قبل التحويل البرمجي، لمساعدتك في تحديد الأخطاء، وتصحيحها بسهولة أثناء كتابة التعليمات، يمكنك التفكير في الأمر مثل المدقق الإملائي الذي يساعدك على إصلاح الأخطاء النحوية أو الإملائية في مستند.

على افتراض أنك كنت ناجحاً في الخطوات السابقة، دعنا نواصل.

عرض رسالة جديدة

في هذه المهمة، ستقوم بتعليق تنفيذ السطر السابق من التعليمات البرمجية، ثم إضافة أسطر برمجية جديدة في محرر .NET. لطباعة رسالة.

١- تعديل السطر البرمجي الذي كتبته، بحيث يكون مسبوق بخطين // مائلين للأمام:

```
//Console.WriteLine("Hello World!");
```

يمكنك إنشاء تعليق سطر برمجي عن طريق إضافة شرطتن مائلتين // في بداية السطر، ترشد هذه البادئة المحول البرمجي أو المترجم the compiler إلى تجاهل جميع الإرشادات الموجودة على هذا السطر.

التعليقات البرمجية مفيدة، عندما لا تكون مستعدًا لحذف التعليمات البرمجية بعد، ولكنك تريد تجاهلها مؤقتاً، يمكنك أيضاً استخدام التعليقات البرمجية لإضافة رسائل إلى نفسك لتذكيرك بما تقوم به التعليمات أو إلى الآخرين، الذين قد يقرأون تعليماتك البرمجية لاحقاً.

٢- أضف أسطر جديدة من التعليمات البرمجية، تطابق التعليمات البرمجية التالية:

```
Console.Write("Congratulations!");
Console.Write(" ");
Console.Write("You wrote your first lines of code.");
```

٣- اضغط على زر «Run» تشغيل مرة أخرى، هذه المرة، سيظهر الإخراج التالي.

```
Congratulations!

You wrote your first lines of code.
```

الفرق بين Console.Write and Console.WriteLine

أظهرت الأسطر الثلاثة الجديدة من التعليمات البرمجية التي أضفتها الفرق بين الأسلوبين Console.WriteLine() و Console.Write()

لطباعة رسالة متعددة الأسطر إلى وحدة الإخراج، استخدمت التقنية الأولى، Console.WriteLine() إذا لاحظت في نهاية الجملة البرمجية كلمة Line أضافت سطر جديد في الرسالة، مشابه لكيفية إنشاء سطر جديد داخل مستند، عن طريق الضغط على «Enter»

لطباعة رسالة مكونة من سطر واحد إلى وحدة الإخراج، أي دون إضافة سطر جديد، استخدمت التقنية الثانية Console.Write() بدون إضافة كلمة Line في نهاية الجملة، لذلك الاستدعاء التالي ل Console.Write() لطباعة رسالة أخرى، ستضيفها إلى نفس السطر.

تهانينا على كتابة الأسطر الأولى من التعليمات البرمجية!

٣ معرفة كيفية عمل التعليمات البرمجية

لفهم كيفية عمل التعليمات البرمجية، تحتاج إلى التراجع والتفكير في ماهية لغة البرمجة، ضع في اعتبارك كيفية توصيل التعليمات البرمجية الأوامر للكمبيوتر، لذلك يجب معرفة المفاهيم، التي توضح طريقة سير عمل البرنامج النهائي الذي أعدته.

ما المقصود بلغة البرمجة؟ a programming language

تتيح لك لغات البرمجة مثل C# كتابة التعليمات التي تريد أن ينفذها الكمبيوتر، كل لغة برمجة لها بناء جملة خاص بها، لكن بعد تعلم لغة البرمجة الأولى، وحاولت الاطلاع على لغة أخرى، ستدرك بسرعة أنها تشترك جميعها في العديد من المفاهيم المتشابهة. وظيفة لغة البرمجة هي السماح للإنسان بالتعبير عن امره للكمبيوتر بطريقة يمكن للبشر قراءتها وفهمها، تسمى هذه الأسطر التي يكتبها مطورو البرامج بلغة برمجة "التعليمات البرمجية المصدر source code" أو فقط "التعليمات أو الرموز البرمجية code"

في مرحلة الإنشاء هذه، يمكن للمطور تحديث التعليمات البرمجية وتغييرها، ولكن لا يمكن للكمبيوتر فهم التعليمات البرمجية، إلى أن يتم التحويل البرمجي أولاً للتعليمة البرمجية، إلى تنسيق يمكن أن يفهمه جهاز الكمبيوتر.

ما المقصود بالتحويل البرمجي؟ compilation

يقوم برنامج خاص يسمى المحول البرمجي "المترجم" **a compiler** بتحويل التعليمات البرمجية المصدر إلى تنسيق مختلف، يمكن لوحدة المعالجة المركزية (CPU) في الكمبيوتر تنفيذه، عند استخدام الزر تشغيل (Run) يتم أولاً تحويل التعليمات البرمجية التي كتبتها، ثم تنفيذها.

لماذا تحتاج التعليمات البرمجية إلى التحويل البرمجي؟ على الرغم من أن معظم لغات البرمجة تبدو غامضة في البداية، إلا أنه يمكن فهمها بسهولة عن

اللغة المفضلة للكمبيوتر، تفهم وحدة المعالجة المركزية التعليمات التي يتم التعبير عنها أما عن طريق تشغيل الآلاف أو الملايين من المفاتيح الصغيرة أو إيقاف تشغيلها، مهمة برامج التحويل البرمجي تواصل هذين العالمين، عالم الأنسان وعالم الكمبيوتر، من خلال ترجمة التعليمات البرمجية التي يمكن قراءتها بواسطة الإنسان، إلى مجموعة أوامر برمجية يمكن فهمها بواسطة جهاز الكمبيوتر.

ما المقصود بالبنية؟ syntax

تسمى قواعد كتابة التعليمات البرمجية بناء الجملة **syntax** تماماً مثل اللغات البشرية، لديها قواعد لغوية، فيما يتعلق بعلامات الترقيم وبنية الجملة، إن لغات برمجة الكمبيوتر لديها أيضاً قواعد، تحدد هذه القواعد الكلمات الأساسية keywords وعوامل تشغيل operators وكيفية تجميعها معاً لتشكيل البرامج.

عند كتابة التعليمات البرمجية في محرر NET. لاحظت تغييرات دقيقة في لون الكلمات، والتعليمات المختلفة، يُعد تمييز بناء الجملة بالألوان ميزة مفيدة، ستبدأ في استخدامها لاكتشاف الأخطاء بسهولة، في التعليمات البرمجية التي تكتبها، ولا تتوافق مع قواعد بناء جملة لغة C#

الآن كيف عملت التعليمات البرمجية؟

دعنا نركز على السطر التالي من التعليمات البرمجية التي كتبتها:

```
Console.WriteLine("Hello World!");
```

عند تشغيل التعليمات البرمجية، رأيت أن الرسالة Hello World! تمت طباعتها، عندما تكون العبارة محاطة بعلامات اقتباس مزدوجة ("") في C# فإنها تسمى سلسلة حرفية literal string أو جملة حرفية، بمعنى آخر، أردت حرفياً إرسال الأحرف "H,e,l,l,o" هكذا _كما هي مكتوبة_ إلى وحدة الإخراج.

شرح بناء الجملة

تسمى الكلمة `Console` من الجملة، فئة `class` والفئة تحوي مجموعة من الأدوات أو الخصائص التي تسمى أساليب أو دوال methods مثل `WriteLine` يمكنك أن تقول إن الخصائص أو الأساليب methods تعيش داخل الفئة class للاستخدام أداة أو أسلوب محدد، يجب أن تعرف أي فئة موجود فيها الأسلوب لاستدعائه، فكر في الفئة كأنها مدينة سكنية، والأساليب هي الشارع الذي تقصده.

هناك أيضاً نقطة "`.`" تفصل اسم الفئة Console واسم الأسلوب WriteLine() أما النقطة فهي عامل تشغيل الأسلوب، بمعنى آخر، النقطة هي كيفية «التنقل» من الفئة إلى أحد أساليبها.

يمكنك دائماً تمييز الأسلوب عن غيره، لأنه يحتوي على مجموعة من الأقواس بعده، وكل أسلوب له وظيفة واحدة، مثلاً وظيفة WriteLine() هي كتابة سطر من البيانات إلى وحدة الإخراج، يتم إرسال البيانات المطبوعة بين قوسي الفتح والإغلاق () كمعلمة إدخال input parameter سواء كانت جمل نصية أو بيانات رقمية كعلامات إدخال، تحتاج بعض الأساليب إلى علامات إدخال اجبارياً، بينما لا تحتاج أساليب أخرى ذلك، ولكن في كل مرة تستدعي أسلوب، يجب عليك استخدام الأقواس بعد اسم الأسلوب، تُعرف الأقواس بعامل استدعاء الأسلوب the method invocation operator

وأخيراً، فإن الفاصلة المنقوطة "`;`" هي نهاية عامل تشغيل العبارة البرمجية، وتُعد هذه العبارة رمز كامل في C# تخبر الفاصلة المنقطة المحول البرمجي بأنك انتهيت من إدخال الأمر.

لا تقلق إذا كانت كل هذه الأفكار والمصطلحات لا معنى لها، في الوقت الحالي، أو غير مفهومة بالنسبة لك، كل ما تحتاج إلى تذكره الآن هو إذا كنت تريد طباعة رسالة:

- استخدام `Console.WriteLine("Your message here");`
- اكتب الحرف الأول من كل كلمة كبيراً Console، وWrite، وLine
- استخدم علامات الترقيم الصحيحة لأن لها دوراً خاصاً في C#

- إذا ارتكبت خطأ، فما عليك سوى التحلي بالصبر، واكتشاف الخطأ وإصلاحه، سوف يساعدك المحرر في ذلك، ثم حاول إعادة تشغيل التعليمات.

تلميح

قم بإنشاء ورقة ملاحظات لنفسك، دون بها الأشياء الهامة، حتى تحفظ أوامر وكلمات رئيسية معينة.

فهم تدفق التنفيذ

من المهم فهم تدفق التنفيذ، بمعنى آخر، لقد تم تنفيذ جمل تعليماتك البرمجية بالترتيب، سطر واحد في كل مرة، ثم الذي يليه، حتى الانتهاء من تنفيذها. ستتطلب بعض التعليمات من المعالج الانتظار قبل أن تتمكن من المتابعة، يمكن استخدام تعليمات أخرى لتعديل مسار التنفيذ.

والآن، دعنا نختبر ما تعلمته، تتميز كل وحدة بتحد صغير، إذا واجهتك مشكلة فلا تقلق، فسيتم تزويدك بحل، وفي الدرس التالي، ستحصل على فرصة كتابة بعض لغة C# بنفسك.

اختبر معلوماتك

ما الفرق بين Console.Write و Console.WriteLine

- Console.Write طباعة الإخراج على سطر جديد
- Console.WriteLine طباعة الإخراج على سطر جديد
- Console.WriteLine إلحاق سطر جديد بعد الإخراج

الإجابة الصحيحة:

Console.WriteLine يطبع الإخراج على السطر الحالي، ويلحق سطراً جديداً بعده

٤ تمرين صغير على ما سبق

إكمال التحدي، سوف تعزز تحديات التعليمات البرمجية في جميع هذه الدروس ما تعلمته، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

التحدي: كتابة تعليمات برمجية تعرض رسالتين

كتابة التعليمات البرمجية التي تنتج الإخراج التالي:

```
This is the first line.
This is the second line.
```

في الدرس السابق، تعلمت كيفية استخدام التعليمات البرمجية لعرض رسالة في سطر واحد فقط، وتعلمت أيضاً كيفية عرض رسالة باستخدام أسطر متعددة، استخدم كلتا التقنيتين لهذا التحدي، لا يهم أي تقنية تطبقها على أي سطر، ولا يهم عدد الطرق التي تستخدم بها التعليمات البرمجية، كي تقسم بها إحدى الرسائل إلى أسطر متعددة، إنه اختيارك.

بغض النظر عن كيفية القيام بذلك، يجب أن تنتج التعليمات البرمجية الإخراج المحدد.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو أتممت عملك بنجاح، استمر إلى الدرس التالي لعرض حل لهذا التحدي.

٥ مراجعة حل التمرين

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة، للتحدي من الدرس السابق.

```
Console.WriteLine("This is the first line.");
Console.Write("This is ");
Console.Write("the second ");
Console.Write("line.");
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، من بين العديد من الطرق الممكنة لتحقيق نفس النتيجة، ومع ذلك، يجب أن تستخدم كلاً من الأساليب `Console.Write` و `Console.WriteLine` لإنتاج الإخراج المطلوب.

```
This is the first line.
This is the second line.
```

هام

إذا واجهتك مشكلة في إكمال هذا التحدي، فراجع الدروس السابقة قبل المتابعة.

إذا نجحت، فتهانينا! تابع إلى الدرس التالي لتختبر معلوماتك.

6 اختبر معلوماتك

١- ما هي المهمة الأساسية للمحول البرمجي؟ the compiler

- يحدد المحول البرمجي بشكل أساسي الأخطاء الإملائية في التعليمات البرمجية.
- يقوم المحول البرمجي في بتنفيذ التعليمات البرمجية.
- يقوم المحول البرمجي بتحويل التعليمات البرمجية إلى تنسيق قابل للتنفيذ يمكن لجهاز الكمبيوتر فهمه.

٢- أي من العبارات التالية صحيحة حول C#

- لغة C# غير حساسة لحالة الأحرف
- Console عبارة عن أسلوب، أما WriteLine() فعبارة عن فئة
- يمكنك استخدام علامات اقتباس مزدوجة لإنشاء جملة حرفية

٣- ما هو الخطأ في هذا السطر من التعليمات البرمجية؟

```
Console.WriteLine("What is wrong with me?")
```

- L في WriteLine ينبغي أن يكون مكتوباً بحروف صغيرة.
- تفتقد الفاصلة المنقوطة في نهاية السطر.
- يجب أن تستخدم الجمل الحرفية علامات اقتباس مفردة.

راجع إجابتك

١ يقوم المحول البرمجي بتحويل التعليمات البرمجية إلى تنسيق قابل للتنفيذ يمكن لجهاز الكمبيوتر فهمه.

صحيح المهمة الأساسية للمحول البرمجي هي تحويل التعليمات البرمجية إلى تنسيق يمكن للكمبيوتر فهمه

٢ يمكنك استخدام علامات اقتباس مزدوجة لإنشاء جملة حرفية

صحيح يمكنك إنشاء جملة حرفية في C# باستخدام علامات اقتباس مزدوجة.

٣ تفتقد الفاصلة المنقوطة في نهاية السطر.

صحيح يجب استخدام الفاصلة المنقوطة في نهاية عبارة التعليمات البرمجية

٧ الملخص

كان هدفك هو كتابة التعليمات البرمجية التي تعرض رسائل قصيرة إلى وحدة الإخراج، أثناء التعرف على بناء الجملة، لقد كتبت الأسطر الأولى من التعليمات البرمجية باستخدام بناء جملة C# syntax الأساسي، وتعلمت تقنيتين لعرض بيانات الجملة الحرفية على الشاشة، تعلمت أيضاً ما تبحث عنه عندما تصادف خطأ في تعليماتك البرمجية.

وأخيراً، حددت عناصر بناء جملة C# مثل الفئات والأساليب classes and methods والغرض من عدة رموز خاصة تعرف باسم عوامل التشغيل operators لقد اتخذت خطواتك الأولى نحو بناء تطبيقات أكثر تطوراً.

الوحدة الثانية

تخزين البيانات واستردادها باستخدام القيم الحرفية والمتغيرة في C#

استخدم البيانات في تطبيقاتك، عن طريق إنشاء قيم حرفية، وقيم متغيرة لأنواع بيانات مختلفة.

الأهداف التعليمية

- إنشاء قيم حرفية لخمسة أنواع بيانات أساسية.
- الإعلان عن المتغيرّات واستخدامها.
- استرداد القيم وتعينها داخل المتغيرات.
- السماح للمحول البرمجي بتحديد نوع البيانات الخاص بالمتغير عند البدء.

محتويات الوحدة: -

١ مقدمة

٢ طباعة القيم الحرفية

٣ إعلان المتغيرات

٤ تعيين القيم وإحضارها من المتغيرات

٥ الإعلان عن المتغيرات المحلية المكتوبة ضمنياً

٦ إكمال التحدي

٧ مراجعة الحل

٨ اختبر معلوماتك

٩ الملخص

١ المقدمة

تتطلب منك العديد من التطبيقات التي ستقوم بإنشائها في C# العمل مع البيانات، في بعض الأحيان تكون البيانات تعليمات برمجية مضمّنة في تطبيقك، القيم المضمنة هي قيم ثابتة، وغير متغيرة طوال تنفيذ البرنامج، على سبيل المثال، قد تحتاج إلى طباعة رسالة إلى المستخدم عند نجاح بعض العمليات، رسالة "النجاح" ستكون على الأرجح هي نفسها في كل مرة يُنفذ فيها التطبيق، يمكن أيضاً تسمية هذه القيمة ذات التعليمات البرمجية المضمنة بالقيمة الثابتة a constant أو القيمة الحرفية a literal value

افترض أنك تريد عرض رسالة منسقة للمستخدم النهائي، تتضمن أنواعاً مختلفة من البيانات، ستتضمن الرسالة "سلاسل strings" مجموعة عبارات نصية، ذات تعليمات برمجية مضمنة، جنباً إلى جنب مع المعلومات التي يجمعها تطبيقك من المستخدم، لعرض رسالة منسقة، ستحتاج إلى إنشاء قيم مشفرة hard-coded وتعريف المتغيرات التي يمكنها تخزين بيانات من نوع معين، سواءً كانت رقمية أو أبجدية رقمية وهكذا.

في هذه الوحدة، ستقوم بإنشاء قيم حرفية تحتوي على أنواع بيانات مختلفة، ستُنشئ متغيرات يمكنها الاحتفاظ بأنواع بيانات معينة، وتعيين قيمة لتلك المتغيرات، ثم استرداد هذه القيم لاحقاً في التعليمات البرمجية، وأخيراً، سوف تتعلم كيف يمكنك تبسيط التعليمات البرمجية، عن طريق السماح للمحول البرمجي the compiler بتحمل بعض العمل.

في نهاية هذه الوحدة، ستتمكن من إنشاء قيم حرفية literal values وتخزين البيانات في المتغيرات، واستردادها.

٢ تمرين ـ طباعة القيم الحرفية

في هذا التمرين، ستقوم بطباعة الرسائل التي تحتوي على أنواع مختلفة من البيانات، ومعرفة سبب أهمية أنواع البيانات في C#

ما هي القيمة الحرفية؟ literal values

القيمة الحرفية هي قيمة ثابتة لا تتغير أبدًا، في السابق، قمت بعرض رسالة حرفية إلى وحدة التحكم Console بمعنى آخر، تريد فعلياً عرض سلسلة أو مجموعة الأحرف الأبجدية الرقمية `H, e, l, l, o` وهكذا في وحدة التحكم.

استخدم نوع البيانات الحرفية أو السلسلة `string` كلما كان لديك كلمات أو عبارات أو بيانات أبجدية رقمية للعرض فقط، وليس حساباً رياضياً، ما هي الأنواع الأخرى من البيانات الحرفية التي يمكنك طباعتها للإخراج؟

تمرين ـ طباعة أنواع بيانات حرفية مختلفة

هناك العديد من أنواع البيانات في C# ولكن مع بدء العمل، لا تحتاج إلا معرفة خمسة أو ستة أنواع بيانات، لأنها تغطي معظم السيناريوهات، دعونا نعرض مثيلاً حرفياً لنوع البيانات للإخراج.

ملاحظة

قد تلاحظ عندما تبدأ العمل في نافذة محرر التعليمات أنه يُلون بناء جملة معينة بألوان مختلفة، للإشارة إلى الكلمات الأساسية، وعوامل التشغيل، وأنواع البيانات وغير ذلك، ابدأ في ملاحظة الألوان، يمكن أن تساعدك على اكتشاف أخطاء بناء الجملة في أثناء إدخال الأحرف، ويمكن أن تساعدك على فهم التعليمات البرمجية بشكل أكثر فعالية.

استخدام القيم الحرفية للأحرف character literals

إذا كنت تريد طباعة حرف أبجدي رقمي، واحد فقط على الشاشة، يمكنك إنشاء **حرف حرفي char literal** عن طريق إحاطة حرف أبجدي رقمي واحد بعلامات اقتباس مفردة، المصطلح char اختصار character بمعني حرف، في C# يسمى نوع البيانات هذا رسمياً "char" ولكن يشار إليه بشكل متكرر باسم "character" حرف.

١. أضف السطر التالي من التعليمات البرمجية في المحرر:

```
Console.WriteLine('b');
```

٢. لاحظ التعليمات البرمجية التي أدخلتها.

لاحظ أن الحرف b محاط بعلامات 'b' اقتباس مفردة، تنشئ علامات الاقتباس المفردة الحرف نفسه، فعلياً، تذكر أن استخدام علامات الاقتباس المزدوجة "" ينشئ نوع بيانات نصي string

٣. اضغط على الزر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن تشاهد النتيجة التالية في نافذة الإخراج:

```
b
```

إذا قمت بإدخال العليمات التالية:

```
Console.WriteLine('Hello World!');
```

قد تحصل على الخطأ التالي:

```
(1,19): error CS1012: Too many characters in character literal
```

لاحظ علامات الاقتباس المفردة المحيطة بكلمات 'Hello World!' عند استخدام علامات اقتباس مفردة، يتوقع المحول البرمجي حرفاً واحدًا فقط، لكن في هذه الحالة، تم استخدام بناء عامل جملة الحرف الواحد، ولكن تم كتابة ١٢ حرفاً، تماماً مثل نوع البيانات string يمكنك استخدام char كلما كان لديك حرف أبجدي رقمي واحد فقط، للعرض التقديمي (ليس حساب رياضي)

استخدام القيم الحرفية للأعداد الصحيحة integer literals

إذا كنت تريد عرض قيمة عدد صحيح رقمي (بدون كسور) في إخراج وحدة التحكم Console يمكنك استخدام قيمة int literal عدد صحيح حرفي، المصطلح "int" اختصار "integer" بمعني الأعداد الصحيحة، والتي قد تتعرف عليها من دراسة الرياضيات في C# يسمى نوع البيانات هذا رسمياً "int" ولكن يشار إليه بشكل متكرر باسم "integer" عدد صحيح، لا تتطلب القيمة الحرفية للأرقام أي عوامل تشغيل أخرى مثل string or char

١. أضف السطر التالي من التعليمات البرمجية في المحرر:

```
Console.WriteLine(123);
```

٢. اضغط على الزر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن ترى النتيجة التالية في إخراج وحدة التحكم:

```
123
```

استخدم القيم الحرفية ذات الفاصلة العائمة floating-point literals

رقم الفاصلة العائمة هو رقم يحتوي على علامة أو رقم عشري، على سبيل المثال ٣,١٤١٥٩ تدعم C# ثلاثة أنواع من البيانات لتمثيل الأرقام العشرية float, double, decimal يدعم كل نوع درجات متفاوتة من الدقة.

```
Float Type    Precision
---------------------------
float         ~6-9 digits
double        ~15-17 digits
decimal       28-29 digits
```

هنا، تعكس الدقة Precision عدد الأرقام التي تجاوزت الفاصلة العشرية الدقيقة.

١. أضف السطر التالي من التعليمات البرمجية في المحرر:

```
Console.WriteLine(0.25F);
```

لإنشاء عدد عشري حرفي (float) قم بإلحاق الحرف "**F**" بعد الرقم مباشرة، في هذا السياق، يسمى F لاحقة قيمة حرفية، تخبر اللاحقة الحرفية المحول البرمجي أنك ترغب في العمل بقيمة من النوع العشري float يمكنك استخدام إما حرف f صغير أو كبير F كلاحقة حرفية ل float

٢. اضغط على زر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن ترى النتيجة التالية في إخراج وحدة التحكم:

```
0.25
```

لاحظ أن نوع البيانات float الأقل دقة من بين الأنواع الأخرى، لذلك من الأفضل استخدام هذا النوع من البيانات للقيم الكسرية الثابتة، لتجنب أخطاء الحساب غير المتوقعة.

٣. أضف السطر التالي من التعليمات البرمجية في المحرر:

```
Console.WriteLine(2.625);
```

لإنشاء رقم حرفي عشري double ما عليك سوى إدخال رقم عشري فقط، يتم تعيين المحول البرمجي افتراضياً على double عند إدخال رقم عشري دون لاحقة حرفية.

٤. اضغط على زر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن تشاهد النتيجة التالية في نافذة الإخراج:

```
2.625
```

٥. أضف السطر التالي من التعليمات البرمجية في المحرر:

```
Console.WriteLine(12.39816m);
```

لإنشاء قيمة حرفية عشرية decimal ألحق الحرف m بعد الرقم مباشرة، في هذا السياق، يسمى m لاحقة قيمة حرفية، تخبر اللاحقة الحرفية m

المحول البرمجي أنك ترغب في العمل بقيمة من النوع decimal يمكنك استخدام إما حرف m صغير أو M كبير كلاحقة حرفية ل decimal

٦. أضغط على زر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن تشاهد النتيجة التالية في نافذة الإخراج:

```
12.39816
```

استخدام القيم الحرفية المنطقية Boolean literals

إذا كنت تريد طباعة قيمة تمثل إما true or false بمعنى صح أو خطأ، يمكنك استخدام قيمة منطقية حرفية.

المصطلح bool اختصار للقيم المنطقية Boolean في C# يشار إليها رسمياً باسم "bool" ولكن غالباً ما يستخدم المطورون مصطلح منطقي "Boolean"

١. أضف الأسطر التالية من التعليمات البرمجية في المحرر:

```
Console.WriteLine(true);
Console.WriteLine(false);
```

٢. أضغط على زر تشغيل Run لتشغيل التعليمات البرمجية، يجب أن تشاهد النتيجة التالية في نافذة الإخراج:

```
True
False
```

تمثل القيم الحرفية bool فكرة الخطأ والصواب، ستستخدم القيم المنطقية bool على نطاق واسع عندما تبدأ في إضافة منطق القرار إلى تطبيقاتك، ستقوم بتقييم التعبيرات لمعرفة ما إذا كان التعبير صحيحاً أم خاطئاً.

لماذا التأكيد على أنواع البيانات؟

تلعب أنواع البيانات دوراً مركزياً في C# يعد التركيز على أنواع البيانات إحدى الميزات المميزة الرئيسية ل C# مقارنة باللغات الأخرى مثل JavaScript يعتقد مصممو C# أنهم يمكن أن يساعدوا المطورين على تجنب أخطاء البرامج الشائعة، عن طريق فرض أنواع البيانات، سوف يتكشف هذا المفهوم أمامك بينما تتعلم المزيد عن C#

تُحدد أنواع البيانات القدرات

في وقت سابق، رأيت أن الجمل أو السلاسل النصية strings والأحرف chars تستخدم في العرض التقديمي "وليس الحساب" إذا كنت بحاجة إلى إجراء عملية حسابية على قيم رقمية، فيجب عليك استخدام int أو العلامة العشرية decimal إذا كانت لديك بيانات يتم استخدامها للعرض التقديمي أو معالجة النص، فيجب عليك استخدام نوع بيانات السلسلة string أو حرف char

لنفترض أنك بحاجة إلى جمع البيانات من مستخدم، مثل رقم هاتف أو رمز بريدي، اعتمادًا على البلد/المنطقة التي تعيش فيها، قد تتكون هذه البيانات من أحرف رقمية، ومع ذلك، بما أنك نادراً ما تقوم بإجراء حسابات رياضية على أرقام الهواتف والرموز البريدية، يجب أن تفضل استخدام أنواع بيانات string عند العمل بها.

يمكن قول الشيء نفسه عن bool إذا كنت بحاجة إلى التعامل مع الكلمات صح وخطأ فقط true and false في تطبيقك، فيمكنك استخدام string وإذا كنت بحاجة إلى العمل بمفهوم الصواب والخطأ true and false عند إجراء تقييم، يمكنك استخدام bool

من المهم أن تعرف أن هذه القيم قد تبدو مثل معادلاتها الحرفية، بمعنى آخر، قد تعتقد أن هذه العبارات هي نفسها:

```
Console.WriteLine("123");
Console.WriteLine(123);

Console.WriteLine("true");
Console.WriteLine(true);
```

ومع ذلك، فإن الإخراج المعروض هو فقط الذي يبدو مشابها، الحقيقة هي أن أنواع الأشياء التي يمكنك القيام بها مع الكلمات الأساسية int أو bool ستكون مختلفة عن مكافئها string

الخلاصة

الميزة الرئيسية هي أن هناك العديد من أنواع البيانات، ولكنك ستركز على عدد قليل فقط في الوقت الحالي:

- string للكلمات أو العبارات أو أي بيانات أبجدية رقمية للعرض التقديمي، وليس العمليات الحسابية.
- char لحرف أبجدي أو رقمي واحد.
- int لعدد صحيح.
- decimal لرقم مع مكون كسري.
- bool لقيمة true/false

٣ إعلان المتغيرات Declare variables

القيمة الحرفية هي فعلياً قيمة ذات تعليمات برمجية "قيم مشفرة" مضمّنة داخل البرنامج، القيم المضمنة هي قيم ثابتة وغير متغيرة طوال تنفيذ البرنامج، تتطلب منك معظم التطبيقات، العمل مع القيم التي لا تعرف الكثير عنها مسبقاً، بمعنى آخر، ستحتاج إلى العمل مع البيانات التي تأتي من المستخدمين، أو من الملفات أو عبر الشبكة، بعد بناء التطبيق.

عندما تحتاج إلى العمل مع البيانات غير المضمنة في التعليمات البرمجية، ستحتاج حاوية لهذه البيانات، هنا، ستعلن عن متغير.

ما هو المتغير؟ variable

المتغير هو مساحة تخزين، نوع من القيم، المتغيرات مهمة لأن قيمها يمكن أن تتغير، أو تختلف، طوال تنفيذ البرنامج، يتم تعيين المتغيرات وقراءتها وتغييرها، يمكنك استخدام المتغيرات لتخزين القيم التي تنوي استخدامها في التعليمات البرمجية.

اسم المتغير هو تسمية مألوفة للإنسان يعينها المحول البرمجي إلى عنوان ذاكرة، عندما تريد تخزين قيمة أو تغييرها في عنوان الذاكرة هذا، أو عندما تريد استرداد القيمة المخزنة، ما عليك سوى استخدام اسم المتغير الذي أنشأته.

الإعلان عن متغير

لإنشاء متغير جديد، يجب أولاً تعريف نوع بيانات المتغير، ثم منحه اسماً.

```
string firstName;
```

في هذا المثال، تقوم بإنشاء متغير جديد من النوع النصي string يسمى firstName من الآن فصاعدًا، يمكن لهذا المتغير أن يحمل قيم حرفية-السلسلة فقط.

يمكنك اختيار أي اسم طالما أنه يتمسك بقواعد بناء الجملة في C# لتسمية المتغيرات.

قواعد واصطلاحات أسماء المتغيرات

ذات مرة، قال أحد مطوري البرامج: "أصعب جزء في تطوير البرامج هو تسمية الأشياء" لا يجب أن يشبه اسم المتغير قواعد بناء جملة معينة فحسب، بل يجب أيضاً استخدامه لجعل التعليمة البرمجية سهلة القراءة من قِبل الإنسان ومفهومه. هذا كثير لطلبه من سطر واحد من التعليمات البرمجية!

فيما يلي بعض الاعتبارات الهامة حول أسماء المتغيرات:

- يجب أن تبدأ أسماء المتغيرات بحرف أبجدي أو رمز مثل شرطة "_" أسفل السطر underscore character وليس برقم.
- يمكن أن تحتوي أسماء المتغيرات على أحرف أبجدية رقمية وشرطة أسفل السطر، لا يُسمح باستخدام الأحرف الخاصة مثل رمز التجزئة # (المعروف أيضاً برمز الرقم) أو رمز الدولار $
- أسماء المتغيرات حساسة لحالة الأحرف، وهذا يعني أن قيمة `string Value;` وقيمة `string value;` هما متغيران مختلفان.
- لا يجب أن تكون أسماء المتغيرات كلمة أساسية في C# على سبيل المثال، لا يمكنك استخدام إعلانات المتغير التالية:

```
decimal decimal; or string string;
```

هناك اصطلاحات ترميز تساعد على إبقاء المتغيرات قابلة للقراءة وسهل التعرف عليها، أثناء تطوير تطبيقات أكبر، يمكن أن تساعدك اصطلاحات الترميز هذه في تعقب المتغيرات من بين نص آخر.

فيما يلي بعض اصطلاحات الترميز للمتغيرات:

- يجب أن تستخدم الأسماء المتغيرة حروفًا كبيرة بالوسط، وهي أسلوب كتابة يستخدم حرفًا صغيراً في بداية الكلمة الأولى، وحرفًا كبيراً في بداية كل كلمة لاحقة، على سبيل المثال:

```
string thisIsCamelCase;
```

- يجب أن تبدأ أسماء المتغيرات بحرف أبجدي، يستخدم المطورين رمز التسطير "_" underscore أول الكلمة لغرض خاص، لذا حاول عدم استخدام ذلك في الوقت الراهن.
- يجب أن تكون أسماء المتغيرات وصفية وذات معنى في تطبيقك، اختر اسماً للمتغير يمثل نوع البيانات التي سيحتفظ بها.
- يجب أن تكون أسماء المتغيرات كلمة كاملة واحدة أو أكثر ملحقين معاً، لا تستخدم كلمات منقوصة أو الاختصارات لأن اسم المتغير (وبالتالي الغرض منه) قد يكون غير واضح للآخرين الذين يقرأون تعليماتك البرمجية.
- لا يجب أن تتضمن أسماء المتغير نوع بيانات المتغير، قد ترى بعض النصائح لاستخدام نمط مثل string strValue; لم تعد هذه النصيحة سائدة.

يتبع المثال `string firstName;` هذه القواعد والاصطلاحات كافة، على افتراض أنك تريد استخدام هذا المتغير لتخزين البيانات التي تمثل الاسم الأول لشخص ما.

أمثلة أسماء المتغيرات

فيما يلي بعض الأمثلة على الإعلان عن المتغيرات باستخدام أنواع البيانات التي تعلمتها حتى الآن:

```
char userOption;

int gameScore;

decimal particlesPerMillion;

bool processedCustomer;
```

الخلاصة

إليك ما تعلمته حتى الآن عن المتغيرات:

- المتغيرات هي قيم مؤقتة تُخزنها في ذاكرة الكمبيوتر.
- قبل أن تتمكن من استخدام متغير، يجب أن تُعرفه.
- لتعريف متغير تحدد أولاً نوع البيانات مثلاً حرفية أو رقمية، حسب نوع البيانات التي تريد تخزينها، ثم إعطاء المتغير اسماً يتبع القواعد.

الآن بعد أن عرفت كيفية الإعلان عن متغير، دعنا نتعلم كيفية تعيين قيمة متغير واستردادها وتهيئتها.

٤ تعيين القيم وإحضارها من المتغيرات

نظراً إلى أن المتغيرات عبارة عن حاويات أو مساحة تخزين مؤقتة للبيانات، فهي مجهّزة للكتابة إليها، والقراءة منها، ستحصل على فرصة للقيام بالأمرين كليهما في التمرين التالي.

التمرين ـ العمل مع المتغيرات

في هذا التمرين، ستقوم بتعريف متغير، وتعيين قيمة له، واسترداد قيمته، والمزيد.

إنشاء المتغير الأول

١. أدخل التعليمات البرمجية التالية في محرر التعليمات البرمجية:

```
string firstName;
firstName = "Bob";
```

للإعلان عن متغير، قم بإدخال نوع البيانات الذي تريد استخدامه، متبوعاً باسم المتغير، ولتعيين قيمة لمتغير، استخدام عامل تشغيل التعيين، وهو الرمز يساوي =

ملاحظة

يشار أيضاً إلى تعيين قيمة بإعداد المتغير (setting the variable) أو ببساطة عملية ضبط ("set" operation)

تعيين قيمة إلى متغير بشكل غير صحيح

من المهم ملاحظة أن مهمة التعيين تتم من اليمين إلى اليسار، بمعنى آخر، يجب على مترجم C# أولاً أن يفهم القيمة الموجودة على الجانب الأيمن من عامل التعيين "=" ثم يمكنه إجراء التعيين للمتغير الموجود على الجانب

الأيسر من عامل التعيين، إذا قمت بعكس الترتيب، فسوف تربك المترجم أو التحويل البرمجي للغة C#

١. عدل التعليمات البرمجية التي كتبتها لمطابقة التعليمات البرمجية التالية:

```
string firstName;
"Bob" = firstName;
```

٢. الآن، شغّل التعليمات البرمجية. سترى الخطأ التالي في إخراج وحدة التحكم:

```
(2,1): error CS0131: The left-hand side of an assignment must be a variable, property or indexer
```

تعيين قيمة نوع البيانات غير الصحيح إلى المتغير بشكل غير صحيح

لقد تعلمت أن C# تم تصميمها لفرض أنواع البيانات، عند العمل مع المتغيرات، يعني فرض الأنواع أنه لا يمكنك تعيين قيمة من نوع بيانات معين إلى متغير تم الإعلان عنه للاحتفاظ بنوع بيانات مختلف، أي لا يمكن الاعلان عن متغير عدد صحيح int أو عشري double ثم تعيين له قيمة نصية string

١. عدل التعليمات البرمجية التي كتبتها لمطابقة التعليمات البرمجية التالية:

```
int firstName;
firstName = "Bob";
```

٢. الآن، شغّل التعليمات البرمجية. سترى الخطأ التالي في إخراج وحدة التحكم:

```
(2,9): error CS0029: Cannot implicitly convert type 'string' to 'int'
```

رسالة الخطأ تُلمح إلى ما يحاول المحول البرمجي لـ C# القيام به خلف الكواليس، حاولت "ضمنياً تحويل" السلسلة الحرفية "Bob" لتصبح قيمة

int ولكن هذا مستحيل، ومع ذلك، حاول C# إجراء التحويل ولكنه فشل نظراً لعدم وجود مكافئ رقمي لكلمة "Bob"

ستتعرف على المزيد حول التحويل الضمني والصريح للنوع لاحقاً، في الوقت الحالي، تذكر أن المتغير يمكنه فقط الاحتفاظ بالقيم المطابقة لنوع البيانات المحدد.

استرداد قيمة مخزنه في المتغير

لاسترداد قيمة من متغير، استخدم فقط اسم المتغير، سيقوم هذا المثال بتعيين قيمة لمتغير، ثم استرداد تلك القيمة وعرضها في وحدة التحكم.

١. عدل التعليمات البرمجية التي كتبتها لمطابقة التعليمات البرمجية التالية:

```
string firstName;
firstName = "Bob";
Console.WriteLine(firstName);
```

٢. شغّل التعليمات البرمجية. ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Bob
```

يُشار أيضًا إلى استرداد قيمة من متغير باسم الحصول على المتغير (getting the variable) أو ببساطة، عملية الحصول على ("get" operation)

أثناء كتابة أسطر التعليمات البرمجية، سترى أن المحول البرمجي يتحقق من التعليمات، ويكتشف الأخطاء المحتملة، المحول البرمجي هو أداة رائعة لمساعدتك في الحصول على التعليمات البرمجية الصحيحة في أسرع وقت، الآن بعد أن أصبحت على دراية بأنواع مختلفة من الأخطاء، يمكنك إصلاح الأخطاء بسرعة بمساعدة رسائل خطأ المحول البرمجي.

إعادة تعيين قيمة متغير

يمكنك إعادة استخدام المتغير، وإعادة تعيينه عدة مرات كما تريد، يوضح هذا المثال هذه الفكرة.

١. عدل التعليمات البرمجية التي كتبتها لمطابقة التعليمات التالية:

```
string firstName;
firstName = "Bob";
Console.WriteLine(firstName);
firstName = "Ahmed";
Console.WriteLine(firstName);
firstName = "Mohamed";
Console.WriteLine(firstName);
firstName = "Eman";
Console.WriteLine(firstName);
```

٢. الآن، شغّل التعليمات البرمجية. ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Bob
Ahmed
Mohamed
Eman
```

تهيئة المتغير

من المنطقي، تعيين قيمة إلى متغير، قبل أن تتمكن من الحصول على القيمة من المتغير، وإلا سترى خطأ.

١. عدل التعليمات البرمجية التي كتبتها في الستة خطوات لمطابقة التعليمات التالية:

```
string firstName;
Console.WriteLine(firstName);
```

٢. الآن، شغّل التعليمات البرمجية. ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
(2,19): error CS0165: Use of unassigned local variable 'firstName'
```

لتجنب إمكانية وجود متغير داخلي غير معين، يوصى بتعيين القيمة في أقرب وقت ممكن بعد الإعلان عن المتغير.

في الواقع، يمكنك تنفيذ كل من التعريف وتعيين قيمة المتغير في سطر واحد من التعليمات، يسمى هذا الأسلوب تهيئة المتغير initializing the variable

١. عدل التعليمات التي كتبتها لمطابقة التعليمات البرمجية التالية:

```
string firstName = "Bob";
Console.WriteLine(firstName);
```

٢. الآن، شغّل التعليمات البرمجية. يجب أن تشاهد الإخراج التالي:

```
Bob
```

الخلاصة

إليك ما تعلمته حول العمل مع المتغيرات:

- يجب تعيين قيمة لمتغير(set) قبل أن تتمكن من استرداد القيمة من المتغير (get)
- يمكنك تهيئة المتغير مباشرة، عن طريق تعيين قيمة إلى المتغير عند الإعلان عنه.
- يحدث التعيين من اليسار إلى اليمين.
- تستخدم رمزاً واحداً "=" single equals كعامل التعيين.
- لاسترداد القيمة من المتغير، استخدام اسم المتغير فقط.

٥ الإعلان عن المتغيرات الداخلية المكتوبة ضمنيًا

يعمل المحول البرمجي The compiler خلف الكواليس لمساعدتك أثناء كتابة التعليمات البرمجية، يمكنه استنتاج نوع بيانات المتغير الخاص بك بواسطة قيمته المُهيأة، في هذا الدرس، ستتعرف على هذه الميزة، التي تسمى الكتابة الضمنية للمتغيرات المحلية implicitly typed local variables

ما هي الكتابة الضمنية للمتغيرات المحلية أو الداخلية؟

يتم إنشاء متغير داخلي مكتوب ضمنيًا باستخدام var الكلمة الأساسية متبوعة بتهيئة المتغير. على سبيل المثال:

```
var message = "Hello world!";
```

في هذا المثال، تم إنشاء متغير سلسلة باستخدام الكلمة الأساسية the var keyword بدلاً من الكلمة الأساسية the string keyword

تخبر الكلمة الأساسية var المحول البرمجي بأن نوع البيانات ضمني، بواسطة القيمة المعينة، أي يخبره أن القيمة التي يتم تمريرها إلى المتغير هي من تحدد نوع بيانات المتغير، إذا كانت رقم يصبح النوع int وإذا كانت نصية يصبح النوع string وهكذا.

بعد تضمين النوع، يعمل المتغير كما لو كان نوع البيانات الفعلي قد تم استخدامه للإعلان عنه، يتم استخدام الكلمة الأساسية var لتوفير ضغطات المفاتيح عندما تكون الأنواع طويلة، أو عندما يكون النوع واضحاً من السياق، في المثال:

```
var message = "Hello world!";
```

نظراً لتعيين المتغير message على الفور، وإعطاءه القيمة الحرفية "Hello world!" يفهم المحول البرمجي القصد ضمنياً، ويعامل كل مثيل كمثيل message على أنه من النوع string

في الواقع يتم كتابة المتغير message ليكون string ولا يمكن تغييره أبدًا، على سبيل المثال، ضع في اعتبارك التعليمات البرمجية التالية:

```
var message = "Hello World!";
message = 10.703m;
```

إذا قمت بتشغيل هذه التعليمات البرمجية، فسترى رسالة الخطأ التالية:

```
(2,11): error CS0029: Cannot implicitly convert type 'decimal' to 'string'
```

ملاحظة

تستخدم لغات البرمجة الأخرى الكلمة الأساسية var بشكل مختلف، في C# يتم تحديد نوع البيانات للمتغيرات وقت الإعلان، بواسطة المترجم، بغض النظر عما إذا كنت تستخدم اسم نوع البيانات الفعلي، أو تسمح للمترجم بتحديد نوع البيانات، بمعنى آخر، يتم تأمين النوع في وقت الإعلان، وبالتالي لن يتمكن أبدًا من الاحتفاظ بقيم من نوع بيانات مختلف.

يجب تهيئة المتغيرات التي تستخدم الكلمة الأساسية var

من المهم أن نفهم أن الكلمة الأساسية var تعتمد على القيمة التي تستخدمها لتهيئة المتغير، إذا حاولت استخدام الكلمة الأساسية var دون تهيئة المتغير، فستتلقى خطأ عند محاولة ترجمة التعليمات البرمجية.

```
var message;
```

إذا حاولت تشغيل هذه التعليمة البرمجية، أثناء التحويل البرمجي، فسترى الإخراج التالي:

```
(1,5): error CS0818: Implicitly-typed variables must be initialized
```

لماذا تستخدم الكلمة الأساسية var

تم اعتماد الكلمة الأساسية var على نطاق واسع في مجتمع C# من المحتمل أنه إذا نظرت إلى مثال تعليمة برمجية في كتاب أو عبر الإنترنت، فسترى الكلمة الأساسية var مستخدمه بدلاً من اسم نوع البيانات الفعلي، لذلك من المهم فهم استخدامها.

الكلمة var لها استخدام مهم في C# في كثير من الأحيان، يكون نوع المتغير واضحاً من تهيئته، في هذه الحالات، من الأسهل استخدام var يمكن أن تكون الكلمة الأساسية مفيدة أيضاً عند تخطيط التعليمات البرمجية لتطبيق ما، عند البدء في تطوير التعليمات البرمجية لمشروع، لا تعرف حينها نوع البيانات الذي يجب استخدامه، يمكن أن يساعدك استخدام var في تطوير الحل بشكل أكثر حيوية.

أثناء البدء، يوصى باستخدام اسم نوع البيانات الفعلي، عند الإعلان عن المتغيرات، حتى تصبح أكثر راحة في العمل مع التعليمات البرمجية، يساعد استخدام نوع البيانات عند الإعلان عن المتغيرات أن تكون هادفة أثناء كتابة تعليماتك البرمجية.

الخلاصة

إليك ما تعلمته عن الكلمة الأساسية var حتى الآن:

- تُعلم var المحول البرمجي لاستنتاج نوع البيانات من المتغير ضمنياً، استنادًا إلى القيمة التي هُيئت.
- من المحتمل أن ترى var أثناء قراءتك التعليمات البرمجية الخاصة بالأشخاص الآخرين؛ ومع ذلك، يجب عليك تحديد نوع البيانات عندما يكون ذلك ممكناً.

٦ إكمال التحدي

في هذا التحدي، ستكتب تعليمات تجمع بين القيم الحرفية والمتغيرة في رسالة واحدة.

التحدي: عرض القيم الحرفية والمتغيرة

١. تخزين القيم التالية في المتغيرات:

- Bob
- 3
- 34.4

ينبغي إعطاء هذه المتغيرات أسماءً تعكس الغرض منها.

تأكد من تحديد نوع البيانات الصحيح لكل من المتغيرات استنادًا إلى نوع البيانات التي سيُحتفظ بها.

وأخيراً، ستقوم بدمج المتغيرات في جملة حرفية، تم تحويلها إلى قيمة نصية string باستخدام Console.Write() لتشكيل رسالة كاملة.

٢. اكتب التعليمات البرمجية في محرر .NET. لعرض الرسالة التالية:

```
Hello, Bob! You have 3 messages in your inbox.
The temperature is 34.4 celsius.
```

بغض النظر عن كيفية القيام بذلك، يجب أن تنتج تعليماتك البرمجية الإخراج المحدد.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل، أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي في الدرس القادم.

٧ مراجعة الحل

التعليمات التالية هو أحد الحلول الممكنة للتحدي من الدرس السابق:

```
string name = "Bob";
int messages = 3;
decimal temperature = 34.4m;

Console.Write("Hello, ");
Console.Write(name);
Console.Write("! You have ");
Console.Write(messages);
Console.Write(" messages in your inbox. The temperature is ");
Console.Write(temperature);
Console.Write(" celsius.");
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، لكيفية إنشاء هذا الإخراج، على سبيل المثال، من الممكن أن تستخدم المزيد من عبارات `Console.Write()` ومع ذلك، يجب أن تكون قد قمت بتهيئة ثلاث متغيرات لتخزين القيم الثلاث، وفقاً للتعليمات في التحدي.

وعلاوة على ذلك، يجب أن تستخدم:

- متغيراً من النوع `string` للاحتفاظ بالاسم "Bob"
- متغيراً من النوع `int` لتخزين عدد الرسائل.
- متغيراً من النوع `decimal, float, or double` لتخزين درجة الحرارة.

إذا نجحت، فتهانينا! تابع إلى اختبار المعلومات في الدرس التالي.

إذا واجهتك مشكلة في إكمال هذا التحدي، فربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٨ اختبر معلوماتك

١- أي من أسماء المتغيرات التالية يجب أو يلزم تجنبها؟

- $DATA
- registrationComplete
- flag

٢- ما هي المشكلة في هذا السطر من التعليمة البرمجية `var message;`

- Var ليس نوع بيانات
- استخدم الكلمة الأساسية var دون تهيئة المتغير
- يجب أن يكون المتغير المسمى message حرفي دوماً

٣- أي مما يلي هو إخراج `Console.WriteLine(34.40M);`

- 34.40M
- 34.4
- 34.40

٤- أي من الأسطر التالية من التعليمات البرمجية ينشئ متغيراً بشكل صحيح؟

- int x = 12.3m;
- decimal x = 12.3m;
- bool x = 'False';

راجع إجابتك

١ $DATA

صحيح لا يمكن استخدام الرمز $ في اسم متغير، ويجب عدم استخدام كافة الأحرف الكبيرة لاسم متغير

٢ استخدم الكلمة الأساسية var دون تهيئة المتغير

صحيح يجب أن يكون لنوع البيانات var قيمة تهيئة

٣ 34.40

صحيح استخدام القيمة العشرية الحرفية M أو m سيشير إلى نوع بيانات عشري، تتضمن البيانات العشرية القيم بعد الفاصلة العشرية عند عرضها.

٤ decimal x = 12.3m;

صحيح يتطابق التعيين بشكل صحيح مع نوع البيانات المُعلن عنه

٩ الملخص

كان هدفك هو عرض رسالة منسقة باستخدام مجموعة من القيم الحرفية والمتغيرة.

باستخدام بناء جملة C# الأساسي، قمت بإنشاء قيم حرفية للعديد من أنواع البيانات المختلفة، لقد أعلنت المتغيرات، وقمت أيضاً بتعيين القيم واستردادها من تلك المتغيرات، قمت أيضاً بتهيئة المتغيرات، وتعلمت كيفية استخدام الكلمة الأساسية `var` لكتابة متغير عن طريق استنتاج النوع ضمنياً من التهيئة.

الوحدة الثالثة

نفّذ تنسيق الجمل الحرفية الأساسية (صياغة الجمل) في لغة C#

أدمج البيانات النصية الحرفية والمتغيرة التي تحتوي على أحرف خاصة، وتنسيقات Unicode في رسائل ذات معنى للمستخدم النهائي.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- إنشاء بيانات نصية متسلسلة (صياغة الجمل) تحتوي على علامات تبويب، وخطوط جديدة، وأحرف خاصة أخرى.
- إنشاء بيانات سلسلة تحتوي على أحرف Unicode
- دمج البيانات المتسلسلة في قيمة سلسلة جديدة بالربط بينهم.
- دمج البيانات المتسلسلة في قيمة سلسلة جديدة عبر استنتاج.

محتويات الوحدة: -

١ مقدمة

٢ دمج النصوص (صياغة الجمل) باستخدام ترابط أحرف الهروب و Unicode

٣ دمج النصوص (صياغة الجمل) باستخدام ترابط السلسلة

٤ دمج النصوص (صياغة الجمل) باستخدام استنتاج السلسلة

٥ إكمال التحدي

٦ مراجعة الحل

٧ اختبر معلوماتك

٨ الملخص

١ المقدمة

بصفتك مطوراً للبرامج، ستحتاج إلى كتابة تعليمة برمجية لدمج بيانات حرفية، ومتغيرة التنسيق لإنشاء قيمة جديدة، ربما تعرض هذه القيمة أو تحفظ في ملف أو ترسلها عبر الشبكة، لحسن الحظ، توفر C# العديد من الطرق لدمج البيانات وتنسيقها.

لنفترض أنك تريد عرض مخرجات سطر الأوامر الذي تكتبه، تريد عرض القيم بما في ذلك النص الحرفي، والنص في المتغيرات، والبيانات الرقمية والبيانات النصية بلغات أخرى، كيف تتمكن من التنسيق بشكل صحيح بحيث يمكن للمستخدم فهم ماهية تطبيقك؟

في هذه الوحدة، ستستخدم تسلسلات إلغاء الأحرف أو أحرف الهروب (character escape) لتنسيق جمل النصوص الحرفية، لإضافة أحرف خاصة بما في ذلك علامات التبويب وموجزات الأسطر، حتى الأحرف من لغات مختلفة مثل اليابانية أو الروسية! ستتعلم كيفية ربط "نصين" سلسلتين معاً، وستستخدم استنتاج السلسلة لإنشاء قالب جمل حرفية، بأجزاء قابلة للاستبدال.

في نهاية هذه الوحدة، ستكون قادراً على التحكم في طريقة عرض البيانات للمستخدمين النهائيين في تطبيقاتك.

٢ دمج النصوص (صياغة الجمل) باستخدام ترابط أحرف الإلغاء أو الهروب character escape sequences

لنفترض مطالبتك بإنشاء نسخة من أداة سطر الأوامر (command-line) التي ستنشئ فواتير باللغتين الإنجليزية واليابانية، لا ينبغي لك تصميم الوظيفة الفعلية التي تقوم بإنشاء الفواتير النهائية، تحتاج فقط إلى توفير واجهة سطر الأوامر للعملاء الداخليين في قسم الفوترة للتعديل عليها، طلب منك مديرك التأكد من إضافة تنسيق لجعل التقدم الحالي للأداة واضحاً، كمّا طلب منك مديرك تقديم إرشادات للمستخدمين اليابانيين حول كيفية إنشاء الفواتير باللغة اليابانية.

ملاحظة

المقصود بالسلسلة هنا، مجموعة نصوص، حرفية أو رقمية أو أي بيانات أخرى، يتم ربطها معًا، لتكوين قطعة نصية كاملة، ومنسقة.

تمرين ـ تنسيق السلاسل الحرفية (صياغة الجمل) في C#

في هذا التمرين، ستتعلم تقنيات مختلفة لعرض أحرف خاصة، وإضافة أنواع مختلفة من التنسيق إلى الإخراج.

تسلسلات أحرف الإلغاء أو الهروب character escape sequences

تسلسل أو تتابع أحرف الهروب character escape sequences هي إرشادات لوقت التشغيل لإدراج حرف خاص سيؤثر على إخراج التسلسل في C# يبدأ ترابط حرف الهروب بعلامة \ مائلة عكسية backslash متبوعة بالحرف الذي تريد ربطه، على سبيل المثال **\n** سيضيف التسلسل سطراً جديدًا، وسيضيف التسلسل **\t** مسافة مزدوجة فارغة على السطر (علامة التبويب)

Escape Sequence	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null, not the same as the C# *null* value
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

تستخدم التعليمات البرمجية التالية تسلسلات أحرف الهروب، لإضافة خطوط ومسافة مزدوجة فارغة علامات تبويب جديدة:

```
Console.WriteLine("Hello\nWorld!");
Console.WriteLine("Hello\tWorld!");
```

إذا قمت بتشغيل التعليمة البرمجية، فسترى المخرجات التالية:

```
Hello
World!
Hello   World!
```

ماذا لو كنت بحاجة إلى إدراج علامة اقتباس مزدوجة في تسلسل لقيمة حرفية؟ إذا لم تستخدم تسلسل أحرف الهروب، فسوف تربك المحول البرمجي لأنه سيعتقد أنك تريد إنهاء السلسلة قبل الأوان، لن يفهم المحول البرمجي الغرض من الأحرف بعد علامة الاقتباس المزدوجة الثانية.

```
Console.WriteLine("Hello "World"!");
```

سيقوم محرر .NET Editor. بوضع خط أحمر متعرج تحت World ولكن إذا حاولت تشغيل التعليمات البرمجية على أي حال، فسترى الخطأ التالي:

```
(1,27): error CS1003: Syntax error, ',' expected
```

```
(1,27): error CS0103: The name 'World' does not exist in the current context
(1,32): error CS1003: Syntax error, ',' expected
```

للتعامل مع هذا الموقف، استخدم تسلسل الهروب \"

```
Console.WriteLine("Hello \"World\"!");
```

إذا قمت بتشغيل التعليمات البرمجية أعلاه، فسترى الإخراج التالي:

```
Hello "World"!
```

ماذا لو كنت بحاجة إلى استخدام شرطة مائلة \ backslash لأغراض أخرى، مثل عرض مسار ملف؟

```
Console.WriteLine("c:\source\repos");
```

لسوء الحظ، تحجز C# الشرطة المائلة العكسية \ backslash لتنسيق تسلسل أحرف الهروب، فإذا قمت بتشغيل التعليمات البرمجية، سيعرض المحول البرمجي الخطأ التالي:

```
(1,22): error CS1009: Unrecognized escape sequence
```

تكمن المشكلة في تسلسل **\r \s** لا ينتج عن خطأ لأنه تسلسل إلغاء صالح لإرجاع حرف، ومع ذلك، لا تريد استخدام إرجاع حرف في هذا السياق. لحل هذه المشكلة، يمكنك استخدام علامتين للشرطة المائلة \\ لعرض علامة مائلة نصية واحدة.

```
Console.WriteLine("c:\\source\\repos");
```

يؤدي الهروب من علامة الشرطة المائلة للخلف إلى الإخراج الذي قصدته:

```
c:\source\repos
```

تنسيق الإخراج باستخدام تسلسلات أحرف الهروب

١. لإنشاء نموذج لأداة سطر الأوامر، أدخل التعليمات البرمجية التالية في المحرر:

```
Console.WriteLine("Generating invoices for
customer \"Contoso Corp\" ...\n");
Console.WriteLine("Invoice:
1021\t\tComplete!");
Console.WriteLine("Invoice:
1022\t\tComplete!");
Console.WriteLine("\nOutput Directory:\t");
```

٢. الآن، شغل التعليمات البرمجية، ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Generating invoices for customer "Contoso Corp" ...

Invoice: 1021        Complete!
Invoice: 1022        Complete!

Output Directory:
```

سلسلة حرفية فعلياً

ستحتفظ السلسلة الحرفية بكل المسافات البيضاء والأحرف المحجوزة، دون الحاجة إلى الهروب منها، مثلاً من الشرطة المائلة العكسية، لإنشاء سلسلة حرفية، استخدم علامة @ قبل السلسلة الحرفية.

```
Console.WriteLine(@"    c:\source\repos
  (this is where your code goes)");
```

لاحظ أن الجملة تُمدد سطرين ثم تظهر المسافة البيضاء التي تم إنشاؤها باستخدام هذا التعليمة، في الإخراج التالي.

```
c:\source\repos
    (this is where your code goes)
```

تنسيق الإخراج باستخدام سلسلة حرفية (صياغة الجمل)

١. أضف السطر التالي من التعليمات أسفل التعليمات البرمجية التي قمت بإنشائها مسبقا:

```
Console.Write(@"c:\invoices");
```

٢. الآن، تشغيل التعليمات البرمجية. سترى النتيجة التالية التي تتضمن "دليل الإخراج":

```
Generating invoices for customer "Contoso Corp" ...

Invoice: 1021       Complete!
Invoice: 1022       Complete!

Output Directory:
c:\invoices
```

أحرف الهروب Unicode escape characters

يمكنك أيضاً إضافة أحرف مرمزة، في تسلسل حرفي باستخدام سلسلة الهروب **\u** ثم رمز مكون من أربعة أحرف يمثل بعض الأحرف في Unicode (UTF-16)

```
// Kon'nichiwa World
Console.WriteLine("\u3053\u3093\u306B\u3061\
u306F World!");
```

ملاحظة

توجد العديد من المحاذير هنا، أولاً، لن تدعم بعض وحدات التحكم مثل موجه الأوامر لـ Windows كل أحرف Unicode سوف تستبدل تلك الأحرف الغير مدعومة بعلامات الاستفهام، من الأمثلة المستخدمة هنا UTF-16 يتطلب بعض الأحرف مثل UTF-32 وبالتالي تتطلب تسلسل إلغاء مختلف، هذا موضوع مُعقد، وهذا الدرس يهدف فقط إلى إظهار ما هو ممكن، اعتمادًا على حاجتك، قد تحتاج إلى قضاء قدر كبير من الوقت في التعلّم والعمل مع أحرف Unicode في تطبيقاتك.

تنسيق الإخراج باستخدام أحرف Unicode escape characters

لإكمال نموذج أداة سطر الأوامر، ستضيف عبارة باللغة اليابانية تترجم لعبارة: "لإنشاء فواتير يابانية" ثم ستعرض تسلسل تنسيق حرفي يمثل شيء يمكن للمستخدم إدخاله، ستضيف أيضاً بعض تسلسلات Unicode للتنسيق.

١. أضف التعليمة البرمجية التالية إلى تطبيقك:

```
// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c
42\u66f8\u3092\u751f\u6210\u3059\u308b\u306b\u3
06f\uff1a\n\t");
// User command to run an application
Console.WriteLine(@"c:\invoices\app.exe -j");
```

٢. للتأكد من صحة تلك التعليمات البرمجية، قارنها بما يلي:

```
Console.WriteLine("Generating invoices for
customer \"Contoso Corp\" ...\n");
Console.WriteLine("Invoice:
1021\t\tComplete!");
Console.WriteLine("Invoice:
1022\t\tComplete!");
Console.WriteLine("\nOutput Directory:\t");
```

```
Console.Write(@"c:\invoices");

// To generate Japanese invoices:
// Nihon no seikyū-sho o seisei suru ni wa:
Console.Write("\n\n\u65e5\u672c\u306e\u8acb\u6c
42\u66f8\u3092\u751f\u6210\u3059\u308b\u306b\u3
06f\uff1a\n\t");
// User command to run an application
Console.WriteLine(@"c:\invoices\app.exe -j");
```

٣. والآن، شغل التعليمات البرمجية. ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Generating invoices for customer "Contoso Corp" ...

Invoice: 1021           Complete!
Invoice: 1022           Complete!

Output Directory:
c:\invoices

日本の請求書を生成するには：
      c:\invoices\app.exe -j
```

خلاصة

إليك ما تعلمته حتى الآن حول تنسيق التسلسلات الحرفية literal strings:

- استخدم تنسيق أحرف الهروب عندما تحتاج إلى إدراج حرف خاص في سلسلة حرفية، مثل مسافة مزدوجة **\t** أو سطر جديد **\n** أو علامة اقتباس مزدوجة **\"**
- استخدم حرف الشرطة المائلة المقررة \\ عند الحاجة إلى استخدام علامة مائلة في النصوص المختلفة.
- استخدم حرف التوجيه ***@*** لإنشاء سلسلة حرفية تحافظ على تنسيق المسافات البيضاء، وأحرف الشرط المائلة في النص.
- استخدم **\u** مع رمز من أربعة أحرف لتمثيل أحرف Unicode (UTF-16) في تسلسل نصي.
- قد لا يدعم نظام التشغيل كل رموز Unicode أو تتم طباعة أحرف Unicode بشكل صحيح استنادًا إلى التطبيق.

٣ دمج النصوص (صياغة الجمل) باستخدام ترابط السلسلة

ستحتاج أحياناً إلى دمج البيانات من العديد من المصادر المختلفة، بما في ذلك السلاسل، والمتغيرات الحرفية التي تحتوي على كلٍّ من البيانات النصية والرقمية، في هذا الدرس، ستستخدم سلسلة لمزج قيمتين أو أكثر في سلسلة جديدة.

ما هو ترابط السلسلة؟ string concatenation

ترابط السلسلة هو "مبرمج يتحدث" ببساطة لدمج قيمتين string أو أكثر في قيمة string جديدة، على عكس الإضافة، يتم إلحاق القيمة الثانية بنهاية القيمة الأولى، وهكذا، في التمرين التالي، ستقوم بكتابة التعليمات البرمجية لربط قيم string معاً.

ترابط سلسلة حرفية ومتغير

لترابط سلسلتين معاً، يمكنك استخدام عامل ترابط السلسلة the string concatenation operator وهو رمز الجمع +

١. أدخل التعليمات البرمجية التالية في محرر التعليمات البرمجية:

```
string firstName = "Bob";
string message = "Hello " + firstName;
Console.WriteLine(message);
```

٢. الآن، شغل التعليمات البرمجية. ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Hello Bob
```

لاحظ تسلسل الترتيب "Hello " هي الأولى في القيمة الجديدة، ويتم إضافة قيمة المتغير firstName في النهاية.

ترابط متغيرات متعددة وقيم حرفية

يمكنك تنفيذ عدة عمليات ترابط في السطر ذاته من التعليمات البرمجية.

١. تعديل التعليمات البرمجية التي كتبتها سابقاً إلى ما يلي:

```
string firstName = "Bob";
string greeting = "Hello";
string message = greeting + " " + firstName + "!";
Console.WriteLine(message);
```

هنا يمكنك إنشاء رسالة أكثر تعقيدًا عن طريق الجمع بين العديد من المتغيرات والقيم الحرفية.

٢. شغل التعليمات البرمجية، ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Hello Bob!
```

تجنب المتغيرات الوسيطة

في الخطوات السابقة، استخدمت متغيراً إضافيًا، للاحتفاظ بالقيمة الجديدة الناتجة عن عملية التسلسل أو الترابط، إذا لم يكن لديك سبب وجيه للقيام بذلك، يمكنك (وينبغي) تجنب استخدام طريقة المتغيرات الوسيطة، لتنفيذ عملية ترابط، الا إذا كنت في حاجة ضرورية إليها.

١. تعديل التعليمات البرمجية التي كتبتها سابقاً إلى ما يلي:

```
string firstName = "Bob";
string greeting = "Hello";
Console.WriteLine(greeting + " " + firstName + "!");
```

٢. شغل التعليمات البرمجية. يجب أن تكون النتيجة في إخراج وحدة التحكم هي نفسها، حتى إذا قمت بتبسيط التعليمات البرمجية:

```
Hello Bob!
```

خلاصة

إليك ما تعلمته عن ترابط النصوص string concatenation حتى الآن:

- يتيح لك ترابط النصوص ضم أصغر قيمة حرفية وقيم المتغيرات في نص مترابط واحد.
- تجنب إنشاء متغيرات وسيطة إذا لم تزيد إضافتها من سهولة القراءة.

٤ دمج النصوص باستخدام استنتاج السلسلة string interpolation

على الرغم من أن ترابط النصوص بسيط ومريح، إلا أن استنتاج القيم string interpolation يزداد شيوعًا في المواقف التي تحتاج فيها إلى دمج العديد من النصوص، والمتغيرات الحرفية في رسالة واحدة منسقة.

ما هو استنتاج أو استيفاء السلسلة "صياغة الجمل النصية"؟

يجمع استنتاج السلسلة قيمًا متعددة في قيمة حرفية واحدة، باستخدام قالب (template) وتعبير استيفاء واحد أو أكثر، يشار إلى تعبير الاستنتاج برمز قوسي الفتح والإغلاق المتعرج { } يمكنك وضع أي تعبير يُرجع قيمة داخل الأقواس، تصبح السلسلة الحرفية قالبًا عندما تكون مسبوقة بالرمز $

بمعني أنه بدلاً من كتابة السطر التالي من التعليمات البرمجية:

```
string message = greeting + " " + firstName + "!";
```

يمكنك كتابة هذا السطر بإيجاز أكثر:

```
string message = $"{greeting} {firstName}!";
```

في هذا المثال البسيط، يمكنك توفير القليل من ضغطات المفاتيح، يمكنك تخيل مدى دقة وإيجاز توليد أو استنتاج النصوص في العمليات الأكثر تعقيدًا، ويجد الكثيرون هذه الطريقة أكثر وضوحاً، وأسهل في القراءة.

في التمرين التالي، ستقوم بإعادة كتابة الرسائل السابقة باستخدام استنتاج النصوص

استخدام الاستنتاج لدمج قيم حرفية وقيمة متغيرة

لصياغة ودمج سلسلتين معاً، عليك بإنشاء سلسلة حرفية، تكون بادئة السلسلة الرمز $ يجب أن تحتوي السلسلة الحرفية على مجموعة واحدة من الأقواس المتعرجة {} واستخدام داخل هذه الأقواس اسم متغير واحد على الأقل أو أكثر.

١. أدخل التعليمات البرمجية التالية في محرر.NET

```
string firstName = "Bob";
string message = $"Hello {firstName}!";
Console.WriteLine(message);
```

٢. شغل التعليمات، يجب أن تكون النتيجة في إخراج وحدة التحكم:

```
Hello Bob!
```

استخدام استنتاج السلسلة مع متغيرات متعددة وقيم حرفية

يمكنك تنفيذ عدة عمليات إقحام في السطر ذاته من التعليمات البرمجية.

١. تعديل التعليمات السابقة إلى ما يلي:

```
int version = 11;
string updateText = "Update to Windows";
string message = $"{updateText} {version}";
Console.WriteLine(message);
```

٢. شغل التعليمات البرمجية، ستشاهد النتيجة التالية في إخراج وحدة التحكم:

```
Update to Windows 11
```

تجنب المتغيرات الوسيطة

تماماً كما فعلت في التمرين السابق، يمكنك إزالة المتغير المؤقت لتخزين الرسالة.

١. تعديل التعليمات السابقة إلى ما يلي:

```
int version = 11;
string updateText = "Update to Windows";
Console.WriteLine($"{updateText} {version}!");
```

٢. شغل التعليمات، يجب أن تكون النتيجة هي نفسها حتى إذا قمت بتبسيط التعليمات البرمجية:

```
Update to Windows 11!
```

الجمع بين القيم الحرفية واستنتاج السلسلة

افترض أنك تحتاج إلى استخدام قيمة حرفية في قالب خاص، يمكنك استخدام كل من رمز بادئة القيمة الحرفية @ ورمز $ معاً.

١. اكتب التعليمات البرمجية التالية في محرر .NET.

```
string projectName = "First-Project";
Console.WriteLine($@"C:\Output\{projectName}
\Data");
```

٢. شغّل الآن التعليمات، ينبغي أن تشاهد النتيجة التالية.

```
C:\Output\First-Project\Data
```

في هذا المثال، يسمح لك الرمز $ بالإشارة إلى المتغير projectName داخل الأقواس، بينما يسمح لك الرمز @ باستخدام الرمز \ الذي لا يمكن الهروب منه، لأنه من الحروف المحجوزة في C#

خلاصة

إليك ما تعلمته عن استنتاج السلسلة أو صياغة الجمل حتى الآن:

- يوفر تنسيق استنتاج السلسلة تحسينًا عبر ترابط الجمل من خلال تقليل عدد الأحرف المطلوبة في بعض الحالات.
- يمكنك ربط استنتاج السلسلة والقيمة الحرفية، عن طريق إضافة الرموز $@ لكل منهم، واستخدام ذلك كبادئة لقالب السلسلة.

٥ إكمال التحدي

ستعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

في هذا التحدي، ستقوم بطباعة ارشادات للمستخدم النهائي، لإعلامه بالمكان الذي سيقوم فيه تطبيقك بحفظ ملفات البيانات، لن تقوم فعلياً بإنشاء أي ملفات، أنت مهتم فقط بعرض الارشادات المنسقة على نافذة وحدة التحكم.

سوف تستخدم ما تعلمته عن تسلسلات أحرف Unicode والسلاسل الحرفية، واستنتاج/صياغة الجمل، لتوفير الإرشادات باللغتين الإنجليزية والروسية.

التحدي: إرشادات التنسيق والعرض

١. ابدأ حل التحدي بالسطرين التاليين من التعليمات البرمجية.

```
string projectName = "ACME";

string russianMessage =
"\u041f\u043e\u0441\u043c\u043e\u0442\u0440\u04
35\u0442\u044c\u0440\u0443\u0441\u0441\u043a\u0
438\u0439\u0432\u044b\u0432\u043e\u0434";
```

سيُستخدم المتغير projectName مرتين في الإخراج المطلوب.

يحتوي المتغير russianMessage على الرسالة "عرض الإخراج الروسي View Russian out-put" باللغة الروسية، ينبغي استخدام هذا المتغير في التعليمات البرمجية التي تطبع الرسالة.

لا يجوز تغيير هذين السطرين من التعليمات البرمجية، وينبغي استخدامهم لتكوين الإخراج المطلوب، ولكن يمكنك إضافة التعليمات أعلى كل سطر وأسفله.

١. يمكنك استخدام الأسلوب `Console.WriteLine()` أو الأسلوب `Console.Write()` مرتين فقط.

بمعنى آخر، لإكمال هذا التحدي، استخدم سطرين فقط من التعليمات البرمجية التي تطبع الإخراج إلى وحدة التحكم، إذا كنت بحاجة إلى طباعة أسطر جديدة إضافية أو إضافة أي تنسيق، ينبغي استخدام ما تعلمته في هذه الوحدة لإنجاز ذلك.

١. استخدم تسلسلات أحرف Unicode والسلاسل الحرفية واستنتاج السلسلة لإنشاء الإخراج.

لإكمال هذا التحدي، ينبغي أن تنتج تعليماتك الإخراج التالي.

```
View English output:
  c:\Exercise\ACME\data.txt

Посмотреть русский вывод:
  c:\Exercise\ACME\ru-RU\data.txt
```

انتبه للأسطر الجديدة، والمسافات المزدوجة، وكيفية استخدام السطرين الإلزاميين من التعليمات البرمجية في الإخراج.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل، أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٦ مراجعة الحل

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
string projectName = "ACME";
string englishLocation =
$@"c:\Exercise\{projectName}\data.txt";
Console.WriteLine($"View English
output:\n\t\t{englishLocation}\n");

string russianMessage =
"\u041f\u043e\u0441\u043c\u043e\u0442\u0440\u04
35\u0442\u044c
\u0440\u0443\u0441\u0441\u043a\u0438\u0439
\u0432\u044b\u0432\u043e\u0434";
string russianLocation =
$@"c:\Exercise\{projectName}\ru-RU\data.txt";
Console.WriteLine($"{russianMessage}:\n\t\t{rus
sianLocation}\n");
```

حل آخر:

```
string projectName = "ACME";
string russianMessage =
"\u041f\u043e\u0441\u043c\u043e\u0442\u0440\u04
35\u0442\u044c\u0440\u0443\u0441\u0441\u043a\u0
438\u0439\u0432\u044b\u0432\u043e\u0434";

Console.WriteLine($"View English
output:\n\t\tc:\\Exercise\\{projectName}\\data.
txt\n");
Console.Write($"{russianMessage}:\n\t\tc:\\Exer
cise\\{projectName}\\data.txt");
```

تمثل هذه التعليمة البرمجية حلاً واحدًا ممكناً، قد يكون لديك بعض الاختلاف في تسمية المتغيرات أو في تسلسلات أحرف Unicode التي استخدمتها، ربما استخدمت Console.Write بدلاً من Console.WriteLine ربما حاولت دمج كل شيء دون استخدام عدة متغيرات.

ومع ذلك، طالما أن تعليماتك البرمجية تتبع الإرشادات الواردة في التحدي، وتنتج المخرجات المطلوبة، فتهانينا! تابع التحقق من المعلومات في الدرس التالي.

هام: إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ اختبر معلوماتك

١- أي من تسلسلات Unicode التالية يجب استخدامها لإضافة علامات اقتباس مزدوجة إلى سلسلة حرفية في التعليمات البرمجية؟

- `\q`
- `\'`
- `\"`

٢- أيّ من الأسطر التالية من التعليمات يستخدم بشكل صحيح استنتاج سلسلة بافتراض أن المتغير value يُعد سلسلة؟

- `Console.WriteLine(@"My value: {value}");`
- `Console.WriteLine($"My value: {value}");`
- `Console.WriteLine(@"My value: [value]");`

٣- تم تضمين أحرف Unicode في سلاسل تطبيق وحدة تحكم، لتقديم رسالة ترحيب باللغة التايلاندية، ومع ذلك، يتم عرض الرسالة فقط كأحرف علامة استفهام، ما هو السبب المحتمل؟

- تسلسل الهروب \u مفقود لأحرف Unicode
- تم استخدام أحرف Unicode الخاطئة
- لا تعتمد وحدة تحكم المستخدم أحرف Unicode

٤- أي من الأسطر التالية من التعليمات يضيف رمز \ واحد للسلسلة directory؟

- `directory = directory + "\";`
- `directory = directory + '\';`
- `directory = directory + @"\";`

راجع إجابتك

١ `\"`

صحيح \" سيعرض علامات اقتباس مزدوجة

٢
```
Console.WriteLine($"My value: {value}");
```
صحيح

٣ لا تعتمد وحدة تحكم المستخدم أحرف Unicode

صحيح

٤

٤
```
directory = directory + @"\";
```

صحيح

٨ الملخص

كان هدفك هو كتابة التعليمات البرمجية التي تنسق الجمل بأحرف خاصة، مثل علامات الاقتباس المزدوجة، والاسطر الجديدة، ومساحة فارغة مزدوجة، والمساحة البيضاء الأخرى، بالإضافة إلى أحرف Unicode كما قمت بدمج واستنتاج/صياغة النصوص باستخدام تقنيتين مختلفتين.
باستخدام تسلسلات أحرف Unicode أضفت أحرفاً خاصة، في سلاسل القيمة الحرفية، إما باستخدام تسلسلات هروب خاصة أو باستخدام سلاسل حرفية، لقد أضفت أحرف Unicode من مجموعة لغات أخرى مثل اليابانية والروسية في الجمل الحرفية.
لقد استخدمت دمج سلسلة بسيطة باستخدام الرمز + وقمت بالترقية إلى الاستنتاج لدمج القيم في قالب سلسلة واحد.
بدون القدرة على تنسيق الإخراج الخاص بك، سيتم تقييدك بشدة في أنواع المعلومات التي يمكنك تقديمها للمستخدم، ومع ذلك، يمكنك الآن تزويد المستخدمين بتعليمات وملاحظات متطورة، بمجموعة متنوعة من التنسيقات والرموز واللغات.

الوحدة الرابعة

تنفيذ العمليات الأساسية على الأرقام في C#

التعرف على عوامل التشغيل والتقنيات المستخدمة في تنفيذ العمليات الرياضية الأساسية على البيانات الرقمية.

الأهداف التعليمية

بعد إكمال هذه الوحدة، ستتمكن مما يلي:

- تنفيذ العمليات الحسابية على القيم الرقمية.
- ملاحظة تحويل النوع الضمني بين الجمل النصية والقيم الرقمية.
- تحويل نوع بيانات إلى آخر مؤقتًا.

محتويات الوحدة:

١ المقدمة

ستتطلب التطبيقات التي ستقوم ببنائها، العمل على بيانات رقمية حرفية ومتغيرة، ويمكن أن تشمل الأمثلة ما يلي:

- تنفيذ عمليات رياضية بسيطة بما في ذلك الجمع والطرح والضرب والقسمة
- تنفيذ عمليات متعددة الخطوات يجب إكمالها بترتيب معين
- تحديد الباقي بعد أداء القسمة
- زيادة قيمة أو إنقاصها، وما إلى ذلك

افترض أنك تريد إجراء عملية حسابية تقوم بتحويل قيمة من وحدة قياس إلى أخرى، على سبيل المثال، ماذا لو كنت بحاجة إلى تحويل درجة الحرارة الحالية من فهرنهايت إلى درجة مئوية؟ بعد حساب درجة الحرارة بالدرجة مئوية، يجب عليك عرض هذه المعلومات في رسالة منسقة للمستخدم، لتحقيق ذلك، ستحتاج إلى تعلم استخدام عوامل التشغيل للعمل على المعاملات مثل القيم الحرفية والمتغيرة.
في هذه الوحدة، ستقوم بإجراء عمليات النصوص الأساسية والرقمية على بياناتك، سيقوم المحول البرمجي/المترجم بتنفيذ مهام مختلفة حسب أنواع بيانات القيم، حول المشغل المحدد، والأهم من ذلك، سوف تفهم كيف يقوم المشغلون بتنفيذ الإجراءات على المعاملات، سيساعدك تعلم كيفية العمل مع المشغلين والمعاملات بشكل صحيح على صياغة تعليمات برمجية مفيدة.
في نهاية هذه الوحدة، ستكون قادراً على كتابة التعليمات البرمجية التي تقوم بتنفيذ العمليات الأساسية على القيم الحرفية والمتغيرة.

٢ إجراء عملية الجمع مع تحويل البيانات الضمني

في كثير من الأحيان، ستحتاج إلى إجراء عمليات رياضية على البيانات الرقمية، ستبدأ بالإضافة في هذا الدرس، وتتوسع إلى عمليات أخرى في الدروس التالية، لأن هناك درسًا مهماً للتعرف على كيفية تحليل المحول البرمجي C# وتفسير التعليمات البرمجية.

إضافة قيمتين رقميتين

لإضافة رقمين معاً، ستستخدم عامل الإضافة the addition operator وهو رمز الجمع + يتم استخدام رمز الجمع + نفسه الذي تستخدمه لدمج النصوص، وللإضافة أيضاً، تسمى أحياناً إعادة استخدام رمز واحد لأغراض متعددة عامل تشغيل التحميل الزائد overloading the operator ويحدث بشكل متكرر في C#
في هذه الحالة، يفهم المترجم the C# compiler ما تحاول القيام به، يقوم المترجم بتحليل التعليمات البرمجية ويرى أن + (المشغل) محاط بقيمتين رقميتين (المعاملات) بالنظر إلى أنواع بيانات المتغيرات (كلاهما int) يفهم أنك تنوي إضافة هاتين القيمتين.

١. أدخل التعليمات التالية في محرر.NET.

```
int firstNumber = 12;
int secondNumber = 7;
Console.WriteLine(firstNumber + secondNumber);
```

٢. شغل التعليمات، وسترى النتيجة التالية في إخراج وحدة التحكم:

```
19
```

مزج أنواع البيانات لفرض تحويلات النوع الضمنية

ماذا يحدث إذا حاولت استخدام الرمز + مع كل من قيم نصية ورقمية string and int

١. تعديل التعليمات التي كتبتها لتطابق التعليمات البرمجية التالية:

```
string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " +
widgetsSold + " widgets.");
```

٢. شغل التعليمات، وسترى النتيجة التالية:

```
Bob sold 7 widgets.
```

في هذه الحالة، يفهم مترجم C# أنك تريد استخدام الرمز + لدمج معاملين. يستنتج ذلك لأن الرمز + محاط بالمعاملات الخاصة بأنواع بيانات string and int لذلك، فهو يحاول ضمنيًا تحويل المتغير int إلى نص مؤقتًا، حتى يمكن دمجه ببقية الجملة.

يحاول مترجم C# مساعدتك بشكل مثالي كلما أمكن، ولكن من الأفضل أن تكون واضحاً بشأن نواياك.

تجربة حالة أكثر تعقيدًا تتمثل في إضافة الأرقام ودمج النصوص

١. عدل التعليمات التي كتبتها لتطابق التعليمات التالية:

```
string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " +
widgetsSold + 7 + " widgets.");
```

٢. شغل التعليمات، وسترى النتيجة التالية:

```
Bob sold 77 widgets.
```

إضافة قوسين لتوضيح نيتك إلى المحول البرمجي

١. عدل التعليمات التي كتبتها لتطابق التعليمات التالية:

```
string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine(firstName + " sold " +
(widgetsSold + 7) + " widgets.");
```

٢. شغل التعليمات، وسترى النتيجة التالية في إخراج وحدة التحكم:

```
Bob sold 14 widgets.
```

يصبح رمز الأقواس () عامل تشغيل زائدًا آخر another overloaded operator في هذه الحالة، يشكل القوسين ترتيب عمليات التشغيل، تماماً كما قد تستخدم في صيغة رياضية، أنت تشير إلى أنك تريد حل ما بداخل الأقواس أولاً، مما يؤدي إلى إضافة رقم 7 إلى قيمة widgetsSold بمجرد الانتهاء من الحل، سيتم تحويل النتيجة ضمنياً إلى جملة بحيث يمكن ربطها مع بقية الرسالة.

ملاحظة

ربما ينبغي عليك تجنب إجراء كل من الحساب ودمج النصوص في سطر واحد من التعليمات البرمجية، المثال هنا لمساعدتك على فهم كيفية عرض عوامل التشغيل والمعاملات، بالطريقة التي يعمل بها المترجم.

الخلاصة

إليك ما تعلمته حتى الآن حول العمليات الرياضية في C#

- يمكنك إجراء عمليات جمع شبيهة بالرياضيات على الأرقام.
- يستخدم رمز الزائد + لكلاً من دمج النصوص وجمع الاعداد، وهذا ما يسمى عامل تشغيل التحميل الزائد، ويستنتج المترجم ضمنياً الاستخدام الصحيح، بناءً على أنواع البيانات التي يعمل عليها.
- عندما يتمكن من ذلك، سيقوم مترجم C# ضمنياً بتحويل الأرقام int إلى نصوص string إذا كان من الواضح أن المطور يحاول دمج جملة نصية برقم، لأغراض العرض.
- استخدم الأقواس لتحديد ترتيب العمليات، لإخبار المترجم بوضوح أنك تريد تنفيذ عمليات محدده قبل العمليات الأخرى.

٣ تنفيذ العمليات الرياضية

الآن بعد أن فهمت أساسيات الإضافة والجمع، والأهم من ذلك، تحويل النوع الضمني بين أنواع البيانات الرقمية والنصية، دعنا نلقي نظرة على العديد من العمليات الرياضية الشائعة الأخرى على البيانات الرقمية.

تنفيذ العمليات الرياضية الأساسية

إجراء الجمع والطرح والضرب والقسمة، باستخدام أعداد صحيحة

١. أدخل التعليمات البرمجية التالية في محرر NET.

```
int sum = 7 + 5;
int difference = 7 - 5;
int product = 7 * 5;
int quotient = 7 / 5;

Console.WriteLine("Sum: " + sum);
Console.WriteLine("Difference: " + difference);
Console.WriteLine("Product: " + product);
Console.WriteLine("Quotient: " + quotient);
```

٢. شغل التعليمات، يجب أن ترى الإخراج التالي:

```
Sum: 12
Difference: 2
Product: 35
Quotient: 1
```

كما تعلم وترى:

- + هو عامل تشغيل الجمع
- - هو عامل تشغيل الطرح
- * هو عامل تشغيل الضرب
- / هو عامل تشغيل القسمة

ومع ذلك، قد لا يكون حاصل القسمة الناتج الذي كنت تتوقعه، يتم اقتطاع القيم بعد الفاصلة العشرية من المتغير quotient لأنه يتم تعريفها على أنها رقم صحيح int ولا يمكن أن يحتوي int على قيم بعد الفاصلة العشرية.

إضافة تعليمة برمجية لإجراء القسمة باستخدام البيانات العشرية الفعلية

لرؤية عملية القسمة تعمل بشكل صحيح، تحتاج إلى استخدام نوع بيانات يدعم الأرقام الكسرية بعد الفاصلة/العلامة العشرية مثل نوع البيانات decimal

١. احذف التعليمات البرمجية من الخطوات السابقة، وأدخل التعليمات التالية في محرر .NET.

```
decimal decimalQuotient = 7.0m / 5;
Console.WriteLine($"Decimal quotient:
{decimalQuotient}");
```

٢. شغل التعليمات، يجب أن تشاهد الإخراج التالي:

```
Decimal quotient: 1.4
```

لكي ينجح هذا، يجب أن تكون قيمة حاصل القسمة الموجودة (يسار عامل التعيين) من النوع العشري decimal أو على الأقل، يجب أن تكون قيمة أحد الأرقام التي يتم تقسيمها من النوع العشري (يمكن أن يكون كلا الرقمين أيضاً من النوع العشري)
فيما يلي مثالان إضافيان يعملان بشكل جيد على قدم المساواة:

```
decimal decimalQuotient = 7 / 5.0m;
decimal decimalQuotient = 7.0m / 5.0m;
```

ومع ذلك، لن تعمل الأسطر التالية من التعليمات البرمجية (أو تعطي نتائج غير دقيقة):

```
int decimalQuotient = 7 / 5.0m;
int decimalQuotient = 7.0m / 5;
int decimalQuotient = 7.0m / 5.0m;
decimal decimalQuotient = 7 / 5;
```

إضافة تعليمة برمجية لإجراء القسمة باستخدام البيانات العشرية الفعلية

ماذا لو كنت لا تعمل مع القيم الحرفية؟ وبعبارة أخرى، ماذا لو كنت بحاجة إلى تقسيم اثنين من المتغيرات من نوع int ولكن لا تريد أن يتم اقتطاع النتيجة؟ في هذه الحالة، يجب عليك إجراء تحويل نوع البيانات من int إلى decimal
يعد التحويل المرسل Casting نوعاً واحدًا من تحويل البيانات، الذي يرشد المترجم إلى التعامل مع قيمة بشكل مؤقت، كما لو كانت نوع بيانات مختلف. لتحويل int إلى decimal يمكنك إضافة عامل التحويل the cast operator قبل القيمة، ويجب استخدام اسم نوع البيانات محاطا بأقواس أمام القيمة لتحويلها، في هذه الحالة، يمكنك إضافة (decimal) قبل المتغيرات `first and second`

١. أدخل التعليمات البرمجية التالية في محرر .NET.

```
int first = 7;
int second = 5;
decimal quotient = (decimal)first / (decimal)second;
Console.WriteLine(quotient);
```

٢. شغل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
1.4
```

ملاحظة

لقد رأيت حتى الآن ثلاث استخدامات لمشغل الأقواس (the parenthesis operator): استدعاء الأسلوب، وترتيب العمليات، والتحويل.

تحديد الباقي بعد قسمة العدد الصحيح

يخبرك عامل التشغيل الرمز `%` بالباقي من قسمة رقم صحيح int ما تتعلمه حقاً من هذا هو ما إذا كان أحد الأرقام قابلاً للقسمة على رقم آخر أم لا، ويمكن أن يكون هذا مفيدًا في أثناء عمليات المعالجة الطويلة، عند المرور عبر مئات أو آلاف سجلات البيانات، وتريد تقديم ملاحظات إلى المستخدم النهائي بعد معالجة كل ١٠٠ سجل بيانات.

١. أدخل التعليمات التالية في محرر .NET.

```
Console.WriteLine($"Modulus of 200 / 5 : {200 % 5}");
Console.WriteLine($"Modulus of 7 / 5 : {7 % 5}");
```

٢. شغل التعليمات، يجب أن ترى الإخراج التالي:

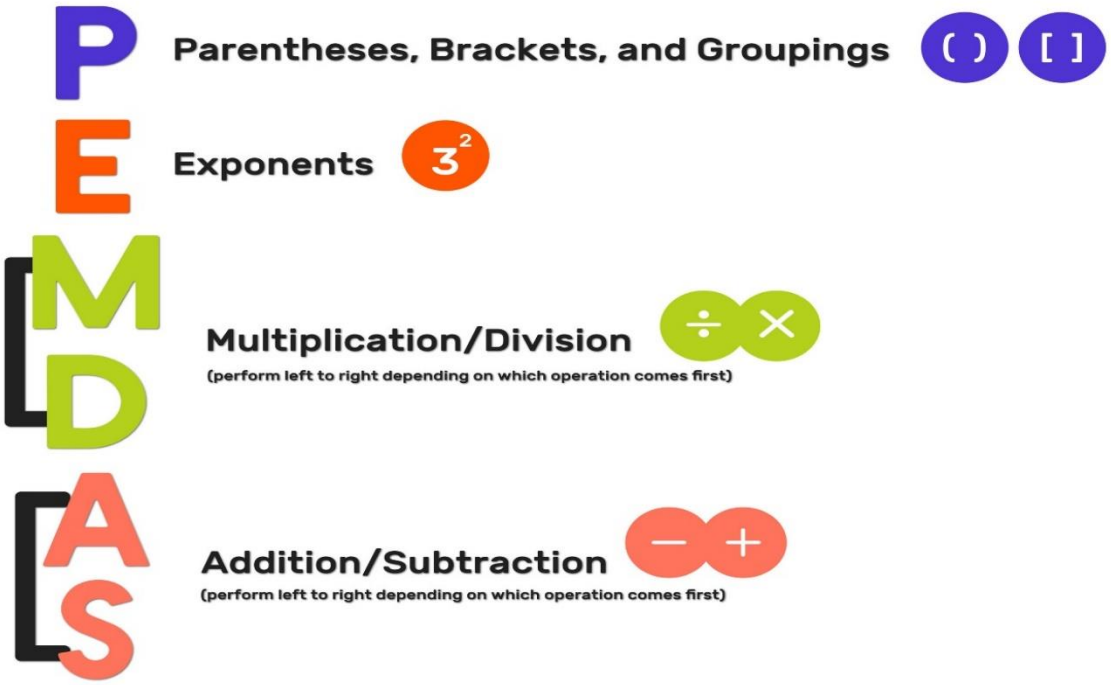
```
Modulus of 200 / 5 : 0
Modulus of 7 / 5 : 2
```

عندما يكون المعامل `0` فهذا يعني أن المقسوم يقبل القسمة على المقسوم عليه. بمعني لا ينتج من القسمة أي كسور عشرية.

ترتيب العمليات

كما تعلمت في التمرين السابق، يمكنك استخدام رمز القوسين () كترتيب لمشغلي العمليات الرياضية، ومع ذلك، هذه ليست الطريقة الوحيدة لتحديد ترتيب العمليات.
في الرياضيات PEMDAS هو اختصار يساعد الطلاب على تذكر ترتيب العمليات. الترتيب هو:

١. Parentheses الأقواس: قم بإجراء العمليات داخل الأقواس أو المجموعات قبل القيام بأي شيء آخر (إذا لم تكن هناك مجموعات أو أقواس، يمكنك تخطي هذه الخطوة)
٢. Exponents الأسس: بعد إجراء العمليات داخل الأقواس والتجميعات (إن وجدت) قم بتطبيق أي أسس (إذا لم تكن هناك أسس، يمكنك تخطي هذه الخطوة)
٣. Multiplication and Division إجراء الضرب والقسمة (من اليسار إلى اليمين بناءً على أي عملية تتم أولاً) هذا لا يعني أنك ستجري دائماً الضرب قبل القسمة.
٤. Addition and Subtraction إجراء الجمع والطرح (من اليسار إلى اليمين بناءً على أي عملية تتم أولاً) هذا لا يعني أنك ستجري دائماً عملية الجمع قبل الطرح.



تتبع C# نفس ترتيب PEMDAS باستثناء الأسس، على الرغم من عدم وجود عامل تشغيل أسي في لغة C# يمكنك استخدام الأسلوب System.Math.Pow ستعرض الوحدة استدعاء الأساليب من مكتبة فئات .NET Framework. باستخدام C# هذا الأسلوب وغيره.

أمثلة على ترتيب العمليات

27 ÷ (8-5)^2

داخل الأقواس: 8-5 = 3

الخطوة التالية تقييم الأسس: 3^2=9

الخطوة الأخيرة هي القسمة: 27÷9 = 3

النتيجة النهائية: **3**

مثال آخر:

10 x 6 + 1

لاحظ أن هذا المثال لا يتضمن مجموعات أو أسس، لذلك، يمكنك تخطي PE في قاعدة PEMDAS والبدء بـ MD

بما أن الضرب والقسمة يأتي قبل الجمع والطرح، فيمكنك حل هذه المشكلة بالانتقال من اليسار إلى اليمين كما يلي:

10x6 = 60

60 + 1 = 61

النتيجة النهائية: **61**

مثال آخر:

$$42 \div 7 \times 3$$

لنبدأ بالقول إن العديد من الأشخاص سيخطئون في فهم هذه المشكلة البسيطة، لأنهم ينسون القواعد الفرعية الأساسية لقاعدة PEMDAS سوف يرتكبون خطأ اتباع قاعدة PEMDAS بدقة، وإجراء الضرب قبل القسمة، نظراً لأن M تأتي قبل D في PEMDAS

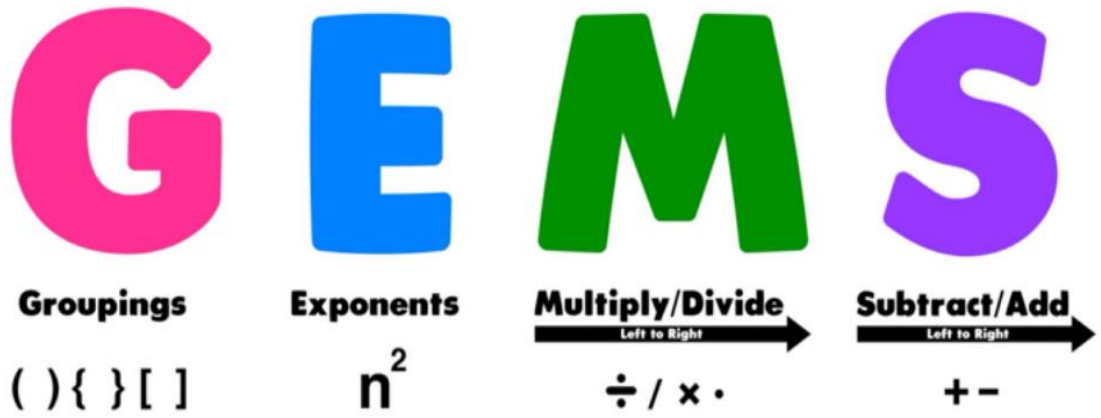
تذكر أن مجرد أن M يأتي قبل D في قاعدة PEMDAS لا يعني أنك ستجري دائماً الضرب قبل القسمة.

في هذه الحالة، العمليات الوحيدة هي الضرب والقسمة، هذه المرة، **القسمة تأتي أولاً**، وهذا أمر جيد، ومازلت تحل المشكلة بالانتقال من اليسار إلى اليمين كما يلي:

$$42 \div 7 = 6$$

$$6 \times 3 = 18$$

النتيجة النهائية: **18**



كتابة التعليمات البرمجية لممارسة ترتيب العمليات في C#

١. دخل التعليمات التالية في محرر .NET.

```
int value1 = 3 + 4 * 5;
int value2 = (3 + 4) * 5;
Console.WriteLine(value1);
Console.WriteLine(value2);
```

هنا ترى الفرق عند تنفيذ نفس العمليات بترتيب مختلف.

٢. شغل التعليمات، يجب أن تشاهد الإخراج التالي:

```
23
35
```

الخلاصة

إليك ما تعلمته حتى الآن عن العمليات الرياضية في C#

- استخدم عوامل التشغيل مثل `+` و `–` و `*` و `/` لتنفيذ العمليات الحسابية الأساسية.
- ستؤدي قسمة قيمتين من الأرقام الصحيحة `int` إلى اقتطاع أي قيم بعد الفاصلة العشرية، للاحتفاظ بالقيم بعد الفاصلة العشرية، تحتاج أولاً إلى تحويل القاسم أو المقسوم (أو كليهما) من رقم صحيح `int` إلى رقم عشري مثل `decimal` يجب أيضاً أن يكون ناتج حاصل القسمة من نفس نوع العلامة العشرية، لتجنب الاقتطاع.
- تنفيذ عملية تحويل لمعالجة قيمة بشكل مؤقت، كما لو كانت نوع بيانات مختلفاً.
- استخدم عامل التشغيل `%` لالتقاط الباقي بعد القسمة.
- سيتبع ترتيب العمليات قواعد الاختصار PEMDAS

٤ زيادة القيم وإنقاصها

العمليات الأساسية النهائية التي ستتعلمها في هذا الدرس عبارة عن كيفية زيادة القيم وإنقاصها، باستخدام عوامل تشغيل مخصصة، عبارة عن مجموعات من الرموز.

الزيادة والتناقص

في كثير من الأحيان، ستحتاج إلى زيادة أو إنقاص القيم، خاصة عند كتابة منطق التكرار، أو التعليمات البرمجية التي تتفاعل مع بنية البيانات.

يقوم عامل التشغيل `+=` بإضافة القيمة الموجودة على يمين عامل التشغيل وتعيينها إلى القيمة الموجودة على يسار عامل التشغيل، لذا، فإن السطرين الثاني والثالث من التعليمات التالية متماثلان:

```
int value = 0;      // value is now 0.
value = value + 5; // value is now 5.
value += 5;         // value is now 10.
```

يقوم عامل التشغيل `++` بزيادة قيمة المتغير بمقدار `١` إذن، السطران الثاني والثالث من التعليمات التالية متماثلان:

```
int value = 0;      // value is now 0.
value = value + 1; // value is now 1.
value++;            // value is now 2.
```

يمكن استخدام هذه التقنيات نفسها للطرح والضرب، والمزيد، وستسلط خطوات التمرين التالية الضوء على عدد منها.

ملاحظة

تعرف عوامل التشغيل مثل =+ و =- و =* و++ و -- على أنها عوامل تشغيل التعيين المركبة، لأنها تقوم بتركيب بعض العمليات بالإضافة إلى تعيين النتيجة إلى المتغير، ويطلق على عامل التشغيل =+ عامل تشغيل تعيين الإضافة على وجه التحديد.

كتابة التعليمات البرمجية لزيادة قيمة وإنقاصها

١. أدخل التعليمات البرمجية التالية في محرر NET.

```
int value = 1;
value = value + 1;
Console.WriteLine("First increment: " + value);
value += 1;
Console.WriteLine("Second increment: " + value);
value++;
Console.WriteLine("Third increment: " + value);
value = value - 1;
Console.WriteLine("First decrement: " + value);
value -= 1;
Console.WriteLine("Second decrement: " + value);
value--;
Console.WriteLine("Third decrement: " + value);
```

٢. شغل التعليمات البرمجية. يجب أن تشاهد الإخراج التالي:

```
First increment: 2
Second increment: 3
Third increment: 4
First decrement: 3
Second decrement: 2
Third decrement: 1
```

ملاحظة

في الزيادة الثانية استخدمت `value += 1;` ومع ذلك، كان بإمكانك استخدام أي قيمة حرفية int (أو متغير) لزيادة هذا الرقم، وينطبق نفس الشيء على الإنقاص الثاني `value -= 1;`

وضع عوامل تشغيل الزيادة والإنقاص

تتمتع كل من عوامل الزيادة والإنقاص بجودة مثيرة للاهتمام، اعتمادًا على موقعها، فإنها تؤدي عملياتها قبل أو بعد استرداد قيمتها، بمعنى آخر، إذا استخدمت عامل التشغيل قبل القيمة كما في `++value` فستحدث الزيادة قبل استرداد _استدعاء_ القيمة، وبالمثل، فإن `value++` ستزيد القيمة بعد استردادها.

استخدام عامل تشغيل الزيادة قبل القيمة وبعدها

١. أدخل التعليمات البرمجية التالية في محرر .NET.

```
int value = 1;
value++;
Console.WriteLine("First: " + value);
Console.WriteLine($"Second: {value++}");
Console.WriteLine("Third: " + value);
Console.WriteLine("Fourth: " + (++value));
```

٢. شغل التعليمات، يجب أن تشاهد الإخراج التالي:

```
First: 2
Second: 2
Third: 3
Fourth: 4
```

لاحظ هذا السطر من التعليمات البرمجية:

```
Console.WriteLine($"Second: {value++}");
```

هناك خطوتان في هذا السطر:

١. تم استرداد القيمة الحالية للمتغير `value` واستخدم ذلك في عملية دمج النص.
٢. ثم زيادة القيمة.

يؤكد السطر التالي من التعليمات أن القيمة في الواقع، تمت زيادتها:

```
Console.WriteLine("Third: " + value);
```

في المقابل، خذ في الاعتبار السطر الأخير من التعليمات:

```
Console.WriteLine("Fourth: " + (++value));
```

هنا، يتم تبديل ترتيب العمليات لأن عامل التشغيل `++` موضوع قبل المتغير `value`

١. زيادة القيمة أولاً.
٢. ثم استرداد القيمة الجديدة الزائدة للمتغير `value` واستخدامها في عملية دمج النص.

على الرغم من أن ذلك ليس ضرورياً تماماً، قمت بإضافة قوسين حول التعبير `(++value)` لسهولة القراءة، يبدو أن رؤية العديد من عوامل التشغيل `+` بجانب بعضها البعض قد يساء فهمها، من قبل المطورين الآخرين، القرارات الأسلوبية هي قرارات شخصية، ومع ذلك، نظراً لأنك ستكتب الرمز أو التعليمة البرمجية مرة واحدة، ولكنك ستقرأها عدة مرات، يجب عليك إعطاء الأولوية لسهولة القراءة.

الخلاصة

إليك ما تعلمته حتى الآن عن العمليات الرياضية أو الحسابية في C#

- استخدم عوامل التعيين المركبة مثل `--` ,`++` ,`*=` ,`-=` ,`+=` لإجراء عملية حسابية مثل الزيادة أو الإنقاص، ثم قم بتعيين النتيجة في المتغير الأصلي.
- يختلف أداء عوامل الزيادة والنقصان، اعتمادًا على ما إذا كان العامل قبل المتغير أم بعده.

٥ تحدي تحويل فهرنهايت إلى درجة مئوية

في هذا التحدي، سوف تكتب تعليمات برمجية تستخدم صيغة، لتحويل درجة حرارة من درجة فهرنهايت إلى درجة مئوية، وطباعة النتيجة في رسالة منسقة للمستخدم.

التحدي: احسب الدرجة المئوية المحولة من درجة الحرارة الحالية بالفهرنهايت

١. أدخل التعليمات التالية في محرر .NET.

```
int fahrenheit = 94;
```

٢. لتحويل درجات الحرارة من درجة فهرنهايت إلى درجة مئوية، اطرح 32 أولاً، ثم اضرب في خمسة على تسعة (5 / 9)
٣. عرض نتيجة تحويل درجة الحرارة في رسالة منسقة.
٤. قم بدمج المتغيرات مع الجملة الحرفية التي تم تمريرها إلى Console.WriteLine() لتكوين الرسالة الكاملة.
٥. عند الانتهاء، يجب أن تكون الرسالة مشابهة للإخراج التالي:

```
The temperature is 34.444444444444444444444444447 Celsius.
```

ملاحظة

من المسلم به أنه يفضل عدم رؤية العديد من القيم بعد الفاصلة العشرية، وبشكل مثالي، سيتم تنسيق القيمة إلى قيمة مفردة بعد الفاصلة العشرية 34.4

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٦ مراجعة الحل تحويل فهرنهايت إلى درجة مئوية

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
int fahrenheit = 94;
decimal celsius = (fahrenheit - 32m) * (5m / 9m);
Console.WriteLine("The temperature is " +
celsius + " Celsius.");
```

هذه التعليمة البرمجية هي حل واحد ممكن من بين العديد من الحلول الممكن، إذا نجحت، فتهانينا! تابع إلى اختبارات المعلومات في الدرس التالي.

إذا كنت تواجه مشكلة في إكمال هذا التحدي، فربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ اختبر معلوماتك

١-أي من الأسطر التالية من التعليمات البرمجية سوف يفشل في إخراج النص Windows 11

- `Console.WriteLine("Windows " + 7 + 4);`
- `Console.WriteLine("Windows " + 11);`
- `Console.Write("Windows " + 1 + 1);`

٢-ما هي قيمة النتيجة التالية؟ `int result = 3 + 1 * 5 / 2;`

- 10
- 5
- 6

٣-ما هي نتيجة السطر التالي من التعليمات البرمجية؟

```
Console.WriteLine(5 / 10);
```

- 0.5
- 0
- 1

٤-أي من الأسطر التالية من التعليمات البرمجية يستخدم عامل تشغيل تعيين الإضافة؟

- `value += 5;`
- `value++;`
- `value = value + 5;`

راجع إجاباتك

١

```
Console.WriteLine("Windows " + 7 + 4);
```

صحيح

٢ 5

صحيح

٣ 0

صحيح

٤

```
value += 5;
```

صحيح

٨ الملخص

كان هدفك هو إجراء العمليات الأساسية على البيانات التسلسلية النصية والرقمية، كأحد تحديات الترميز، قمت بتحويل قيمة من وحدة قياس (فهرنهايت) إلى أخرى (مئوية) وعرض النتيجة في رسالة منسقة.

لقد استخدمت عوامل تشغيل مختلفة لتنفيذ العمليات الحسابية، وصياغة النصوص الأساسية، تعلمت كيفية إعادة استخدام بعض الرموز (تحميلها بشكل زائد) كعوامل تشغيل مختلفة، اعتمادًا على السياق، تعلمت أيضاً كيف تؤثر أنواع بيانات المعاملات على مضمون عوامل التشغيل.

وأخيراً، تعلمت كيفية تغيير نوع بيانات لقيمة ما، باستخدام عامل تشغيل التحويل المرسل the cast operator

الوحدة الخامسة

مشروع إرشادي - حساب درجات الطلاب وطباعتها

تعلم كيفية تطوير تطبيق يقوم بحساب النتائج، وعرضها استنادًا إلى بيانات محددة.

الأهداف التعليمية

بعد إكمال هذه الوحدة، ستتمكن مما يلي:

- تعلم كيفية تطوير التطبيقات الأساسية عن طريق تقسيم العمل إلى أجزاء أصغر.
- تحديد أنواع البيانات المناسبة لتخزين المتغيرات والعمل معها.
- تنفيذ العمليات الرياضية لتحديد النتائج.
- تطبيق تقنيات التنسيق لعرض الإخراج.

محتويات الوحدة:

١ المقدمة

يقوم المطورون بتنفيذ بعض المهام كل يوم تقريباً، بما في ذلك مهام الإعلان عن المتغيرات الرقمية والنصية، وتعيين واستخراج القيم، وإجراء العمليات الحسابية، هي مهام روتينية، ولكنها ضرورية، بنفس القدر من الأهمية لمهمة توصيل النتائج إلى مستخدم التطبيق، إتقان القدرة على تطبيق هذه المهارات لحل مشكلة معينة، هو أمر يجب على كل مطور أن يتعلم القيام به.

لنفترض أنك مساعد معلم في المدرسة، وتم تكليفك بتطوير تطبيق يعمل على تصنيف تقدير الطلاب، يستخدم التطبيق جميع الواجبات المسجلة لكل طالب لحساب درجاته الإجمالية للصف، قام المعلم أيضاً بتوفير التنسيق المطلوب لعرض درجات الطلاب.

سترشدك هذه الوحدة خلال الخطوات المطلوبة لتطوير تطبيق "تصنيف الطالب" ستقوم بتعريف القيم وتعيينها للمتغيرات، حسب أسماء الطلاب، وإجراء حسابات رقمية مختلفة، وعرض النتائج، تتضمن العمليات الحسابية تحديد مجموع درجات المواد، وحساب الدرجات الإجمالية لكل طالب في الصف، لعرض النتائج بالتنسيق المطلوب، ستستخدم الأسلوب `Console.WriteLine()` بالإضافة إلى تسلسلات الأحرف التي ستساعد في تنسيق نتائجك.

في نهاية هذه الوحدة، ستتمكن من كتابة التعليمات البرمجية التي تستخدم أنواع متغيرات مختلفة، وتنفذ حسابات رقمية، وتعرض البيانات المُنسقة للمستخدم.

ملاحظة

هذا نموذج مشروع موجهة، حيث يمكنك إكمال المشروع بالكامل من خلال اتباع الإرشادات خطوة بخطوة

٢ الاستعداد للمشروع الإرشادي

ستستخدم محرر NET. كبيئة تطوير التعليمات البرمجية، ستقوم بكتابة التعليمات التي تستخدم متغيرات نصية ورقمية، وتنفذ العمليات الحسابية، وتعرض النتائج إلى وحدة تحكم.

نظرة عامة على المشروع

أنت تقوم بتطوير تطبيق تقدير الطلاب، الذي يقوم بحساب التقديرات الحالية لكل طالب في الفصل الدراسي، معطيات التطبيق هي:

- يتم إعطاؤك قائمة قصيرة تضم أربعة طلاب ودرجات موادهم الخمسة.
- يتم التعبير عن كل درجة كقيمة عددية، من 0 إلى 100 حيث يمثل 100 نسبة تقديرية صحيحة 100%
- يتم حساب الدرجات النهائية كمتوسط مجموع درجات المواد الخمسة.
- يحتاج تطبيقك إلى إجراء عمليات حسابية أساسية لحساب الدرجات النهائية لكل طالب.
- يحتاج تطبيقك إلى إخراج وعرض اسم كل طالب والنتيجة النهائية.

حالياً، يعرض دفتر درجات المعلمين درجات المواد لكل طالب على النحو التالي:

محمد 100, 95, 98, 87, 93

أحمد 85, 88, 82, 83, 80

خديجة 79, 85, 73, 96, 84

إيمان 97, 100, 98, 92, 90

يطلب المعلم عرض المجموع الكلي لكل طالب على النحو التالي:

Student	Grade	
Mohamed	94.6	A
Ahmed	83.6	B
Khadija	83.4	B
Eman	95.4	A

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي:

انسخ التعليمات البرمجية التالية وألصقها في محرر NET. تمثل هذه القيم درجات المواد لكل طالب.

```
// initialize variables - graded assignments
int currentAssignments = 5;

int mohamed1 = 93;
int mohamed2 = 87;
int mohamed3 = 98;
int mohamed4 = 95;
int mohamed5 = 100;

int ahmed1 = 80;
int ahmed2 = 83;
int ahmed3 = 82;
int ahmed4 = 88;
int ahmed5 = 85;

int khadija1 = 84;
int khadija2 = 96;
int khadija3 = 73;
int khadija4 = 85;
int khadija5 = 79;

int eman1 = 90;
int eman2 = 92;
int eman3 = 98;
int eman4 = 100;
int eman5 = 97;
```

٣ تمرين ـ حساب مجموع درجات مواد الطالب

في هذا التمرين، ستستخدم درجات المواد لكل طالب لحساب تقديره الحالي في الفصل الدراسي، لإجراء هذه العملية الحسابية، عليك أولاً جمع قيم درجات المواد الخاصة بهم، ثم حساب متوسط درجاتهم (الدرجة الحالية). هيا بنا نبدأ.

هام: يجب أن تكون قد أكملت إرشادات الإعداد في الدرس السابق، قبل بدء هذا التمرين

إنشاء متغيرات لتخزين المجموع

في هذه المهمة، ستقوم بإنشاء متغير لكل طالب يمثل مجموع درجات المواد، كما ستعرض مجموع الطالب مع اسمه في إخراج وحدة التحكم، لأن درجات التقدير يتم تمثيلها كأعداد صحيحة، فستنشئ متغيرات عدد صحيح لتخزين المجموع.

١. تأكد من فتح محرر NET. وأن لديك المتغيرات التي تم إنشاء مثيل لها مع درجات مواد كل طالب.

في الدرس السابق إعداد لهذا المشروع الإرشادي، تحتوي إرشادات الإعداد على نسخ درجات مواد الطلاب إلى المحرر، إذا لزم الأمر، فارجع وأكمل إرشادات الإعداد.

٢. مرر إلى أسفل التعليمات البرمجية، وأنشئ سطر فارغ جديد.
٣. للإعلان عن متغير عدد صحيح لكل طالب يمكنك استخدامه لجمع درجاته، أدخل التعليمات البرمجية التالية:

```
int mohamedSum = 0;
int ahmedSum = 0;
int khadijaSum = 0;
int emanSum = 0;
```

لاحظ أنه تم تعيين 0 إلى المتغيرات كجزء من بيان الإعلان، بمعنى آخر، تتم تهيئة المتغيرات إلى 0 على الرغم من أن تعيين القيمة غير مطلوب عند الإعلان عن المتغيرات، إلا أن ذلك يجعل التعليمات أكثر كفاءة، الخطوة التالية هي عرض الإخراج، وبما أن هذا الإخراج سيتضمن مرجعاً إلى هذه المتغيرات، يجب تهيئتها.

٤. لإنشاء عبارات Console.WriteLine() تعرض اسم الطالب وقيمة درجات المواد التي تم جمعها، أدخل التعليمات التالية في الأسفل:

```
Console.WriteLine("Mohamed: " + mohamedSum);
Console.WriteLine("Ahmed: " + ahmedSum);
Console.WriteLine("Khadija: " + khadijaSum);
Console.WriteLine("Eman: " + emanSum);
```

في النهاية، تريد عرض التقدير الإجمالي للطالب، ولكن في الوقت الحالي، دعنا نستخدم هذه العبارات Console.WriteLine() لعرض قيمة حسابات المجموع، وبهذه الطريقة، يمكنك التحقق لمعرفة ما إذا كانت التعليمات البرمجية، تعمل بشكل صحيح في كل مرحلة من مراحل عملية التطوير.

ملاحظة

توفر بيئات التطوير الأكثر تقدماً، مثل Visual Studio Code أدوات تمكن المطورين من مراقبة قيم متغيرات التطبيق، أثناء تشغيل التعليمات البرمجية الخاصة بهم، يوفر تعلم كيفية استخدام هذه الأدوات للمطورين المحترفين الكثير من الوقت أثناء التطوير، في الوقت الحالي، يمكنك استخدام أسلوب Console.WriteLine() للمساعدة في التحقق من أن التعليمات البرمجية الخاصة بك تعمل كما هو متوقع.

٥. في محرر .NET. لتشغيل التعليمات البرمجية، حدد الزر تشغيل Run

لاحظ أنه ليس لديك مشكلة في عرض قيم الأعداد الصحيحة، كلها 0 في الوقت الحالي، باستخدام نفس الأسلوب WriteLine() الذي يعرض القيم الحرفية لسلسلة (أسماء الطلاب)

يتم استرداد القيمة الرقمية الحالية تلقائيًا عن طريق الرجوع إلى اسم المتغير.

الآن بعد أن أصبحت العبارات Console.WriteLine() جاهزة لعرض نتائجك، دعنا نبدأ في إضافة التعليمات البرمجية التي تقوم بإجراء العمليات الحسابية.

هام: تحتاج إلى كتابة التعليمات التي تقوم بالحسابات أعلى التعليمات البرمجية التي تعرض الإخراج.

٦. حدد سطر التعليمة التالية `int mohamedSum = 0;`

ستكتب التعليمات التي تحسب قيمة المجموع لكل طالب، أولاً، ستضيف درجات مواد/واجبات الطلاب، ثم تعين القيمة إلى متغيرات "المجموع" لنبدأ ب (محمد)

٧. تحديث سطر التعليمات البرمجية إلى ما يلي:

```
int mohamedSum = mohamed1 + mohamed2 + mohamed3 + mohamed4 + mohamed5;
```

٨. في محرر .NET. حدد تشغيل **Run**

يجب أن يظهر الإخراج مجموع محمد يساوي 473 سيظل الآخرون 0 ستضيف حسابات مجموع مماثلة لبقية الطلاب.

١. بدءًا من سطر التعليمة السابقة، عدل التعليمات البرمجية كالتالي:

```
int ahmedSum = ahmed1 + ahmed2 + ahmed3 + ahmed4 + ahmed5;
```

```
int khadijaSum = khadija1 + khadija2 + khadija3 + khadija4 + khadija5;

int emanSum = eman1 + eman2 + eman3 + eman4 + eman5;
```

راجع عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية والتحقق من صحة الإخراج.

١. قارن التعليمات البرمجية الخاصة بك بالآتي:

```
int currentAssignments = 5;

int mohamed1 = 93;
int mohamed2 = 87;
int mohamed3 = 98;
int mohamed4 = 95;
int mohamed5 = 100;

int ahmed1 = 80;
int ahmed2 = 83;
int ahmed3 = 82;
int ahmed4 = 88;
int ahmed5 = 85;

int khadija1 = 84;
int khadija2 = 96;
int khadija3 = 73;
int khadija4 = 85;
int khadija5 = 79;
```

```
int eman1 = 90;
int eman2 = 92;
int eman3 = 98;
int eman4 = 100;
int eman5 = 97;

int mohamedSum = mohamed1 + mohamed2 + mohamed3
+ mohamed4 + mohamed5;
int ahmedSum = ahmed1 + ahmed2 + ahmed3 + ahmed4
+ ahmed5;
int khadijaSum = khadija1 + khadija2 + khadija3
+ khadija4 + khadija5;
int emanSum = eman1 + eman2 + eman3 + eman4 +
eman5;

//int mohamedSum = 0;
//int ahmedSum = 0;
//int khadijaSum = 0;
//int emanSum = 0;

Console.WriteLine("Mohamed: " + mohamedSum);
Console.WriteLine("Ahmed: " + ahmedSum);
Console.WriteLine("Khadija: " + khadijaSum);
Console.WriteLine("Eman: " + emanSum);
```

٢. في محرر .NET. حدد تشغيل **Run**

٣. راجع الإخراج وتحقق من صحة مجموع الدرجات:

```
Mohamed: 473
Ahmed: 418
Khadija: 417
Eman: 477
```

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، ثم قم بتشغيلها مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تعرض النتائج المتوقعة.

هام:

تأكد من عدم حذف أي من التعليمات البرمجية التي كتبتها حتى الآن، سوف تبني على هذه التعليمات البرمجية في التمرين التالي.

٤ تمرين ـ حساب متوسط درجات المواد الطالب

في هذا التمرين، ستقوم بحساب وتخزين متوسط درجات المواد لكل طالب، نظراً لأنك تعرف عدد المواد المسجلة لكل طالب، يتم حساب المتوسط عن طريق تقسيم مجموع الدرجات على عدد المواد، لتخزين المتوسطات، ستستخدم نوع البيانات العشرية.

إنشاء متغيرات لتخزين المتوسط

في هذه المهمة، ستقوم بإنشاء متغير لكل طالب، لاستخدامه لتخزين متوسط درجة المواد.

١. في محرر .NET. حدد موقع العبارات Console.WriteLine() المستخدمة لعرض مجموع درجات كل طالب.
٢. إنشاء سطر فارغ أعلى العبارات Console.WriteLine()
٣. في السطر الفارغ الذي قمت بإنشائه، للإعلان عن المتغيرات العشرية التي سيتم استخدامها للدرجات الحالية للطلاب، أدخل التعليمات البرمجية التالية:

```
decimal mohamedScore;
decimal ahmedScore;
decimal khadijaScore;
decimal emanScore;
```

لاحظ أنك تعلن فقط عن المتغيرات decimal ولا تقوم بتهيئتها، لقد اخترت النوع العشري decimal لأنك تخزن متوسط الدرجات، وتريد تضمين الكسور العشرية، ولن تكون متوفرة إذا استخدمت عددًا صحيحاً، بهذه الطريقة، يمكنك معرفة ما إذا كان الطالب قد حقق درجة 89.9 وتقديرها من A إلى B

في التمرين السابق، قمت بتهيئة متغيرات عدد صحيح، بحيث يمكنك استخدامها على الفور في إخراج وحدة التحكم، في هذه الحالة، سيتم تهيئة

هذه المتغيرات decimal في الخطوة التالية باستخدام العمليات الحسابية مع بياناتك الحالية، بدءًا من درجة محمد.

٤. لتعيين درجة محمد الحالية إلى الرقم العشري mohamedScore قم بتحديث المتغير بالتعليمات التالية:

```
decimal mohamedScore = mohamedSum / currentAssignments;
```

لحساب الدرجة الحالية للطالب في الفصل الدراسي، يمكنك قسمة مجموع درجات المواد، على عدد المواد، لدى كل طالب في الفصل خمس مواد، ممثلة بالمهام الحالية التي قمت بتهيئتها أثناء الإعداد.

٥. لتعيين بقية درجات الطلاب، أدخل التعليمات البرمجية التالية:

```
decimal ahmedScore = ahmedSum / currentAssignments;
decimal khadijaScore = khadijaSum / currentAssignments;
decimal emanScore = emanSum / currentAssignments;
```

في نهاية المطاف، تريد عرض درجات كل طالب في هذا التطبيق، في الخطوة التالية، ستقوم بتعديل التعليمات لعرض درجة كل طالب بدلاً من مجموع المواد الخاص به.

٦. لعرض النتيجة الحالية لكل طالب، استبدل متغيرات المجموع في عبارات العرض بمتغيرات النتيجة:

```
Console.WriteLine("Mohamed: " + mohamedScore);
Console.WriteLine("Ahmed: " + ahmedScore);
Console.WriteLine("Khadija: " + khadijaScore);
Console.WriteLine("Eman: " + emanScore);
```

٧. خذ دقيقة للنظر في النهج المتزايد الذي تستخدمه لتطوير هذا التطبيق.

يعد تقسيم مشكلة إلى أجزاء أصغر، مهارة مهمة للمطورين، يتيح لك إنشاء التعليمات البرمجية بشكل متزايد، والتحقق من عملك بشكل متكرر، تطوير

تطبيقات موثوقة بسرعة، في هذه الحالة، يمكنك إعادة استخدام `Console.WriteLine()` للتحقق من صحة حساباتك، أثناء إكمال كل مرحلة من مراحل العملية.

٨. لعرض قيم الصف الحالي لكل طالب، حدد تشغيل Run

يجب أن تشاهد الناتج التالي:

```
Mohamed: 94
Ahmed: 83
Khadija: 83
Eman: 95
```

٩. لاحظ أن الدرجات يتم عرضها كأعداد صحيحة بدلاً من أرقام عشرية.

عندما تريد أن تكون نتيجة حساب القسمة قيمة عشرية، يجب أن يكون المقسوم أو المقسوم عليه من النوع العشري (أو كليهما) عند استخدام متغيرات عدد صحيح في الحساب، تحتاج إلى تطبيق تقنية تعرف باسم التحويل `convert` لتحويل عدد صحيح إلى رقم عشري.

لحساب النتيجة، يمكنك الحصول على نتيجة عشرية عن طريق تحويل متغير المجموع كنوع عشري، في هذه الحالة، ستجد المجموع عدد عشري.

١٠. في عمليات القسمة، لتحويل متغير عدد صحيح إلى عدد عشري، عدل التعليمات بإضافة عامل التحويل `(decimal)` على النحو التالي:

```
decimal mohamedScore = (decimal) mohamedSum
/ currentAssignments;
decimal ahmedScore = (decimal) ahmedSum /
currentAssignments;
decimal khadijaScore = (decimal)
khadijaSum/currentAssignments;
decimal emanScore = (decimal) emanSum /
currentAssignments;
```

ما عليك سوى أن يكون المجموع أو المقسوم عليه من النوع العشري، لتنتج القسمة قيمة عشرية، هنا تقوم فقط بإدخال متغير المجموع الذي يتم استخدامه لتوزيع التقديرات.

١١. راجع مقياس الدرجات التالي الذي يستخدمه المعلم لتعيين درجات التقدير:

```
97 - 100    A+
93 - 96     A
90 - 92     A-
87 - 89     B+
83 - 86     B
```

الخطوة التالية هي تضمين درجة التقدير الحرفي، لكل طالب بناءً على مجموع درجاته، ستكون إضافة التقدير إلى المخرجات المعروضة عملية يدوية.

١٢. لتحديد قيمة الصف الحالي لكل طالب، حدد تشغيل Run

استخدم الصف الحالي لكل طالب لتحديد درجة التقدير المناسبة، والتقريب للأكثر أو الأقل حسب الضرورة.

١٣. لإلحاق درجة التقدير بعد النتيجة الرقمية لكل طالب، قم بتحديث التعليمات البرمجية على النحو التالي:

```
Console.WriteLine("Mohamed: " + mohamedScore + " A");
Console.WriteLine("Ahmed: " + ahmedScore + " B");
Console.WriteLine("Khadija: " + khadijaScore + " B");
Console.WriteLine("Eman: " + emanScore + " A");
```

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية، والتحقق من صحة الإخراج.

١. قارن التعليمات البرمجية الخاصة بك بالآتي:

```
int currentAssignments = 5;

int mohamed1 = 93;
int mohamed2 = 87;
int mohamed3 = 98;
int mohamed4 = 95;
int mohamed5 = 100;

int ahmed1 = 80;
int ahmed2 = 83;
int ahmed3 = 82;
int ahmed4 = 88;
int ahmed5 = 85;

int khadija1 = 84;
int khadija2 = 96;
int khadija3 = 73;
int khadija4 = 85;
int khadija5 = 79;

int eman1 = 90;
int eman2 = 92;
int eman3 = 98;
int eman4 = 100;
int eman5 = 97;

int mohamedSum = mohamed1 + mohamed2 + mohamed3
+ mohamed4 + mohamed5;
int ahmedSum = ahmed1 + ahmed2 + ahmed3 + ahmed4
+ ahmed5;
int khadijaSum = khadija1 + khadija2 + khadija3
+ khadija4 + khadija5;
int emanSum = eman1 + eman2 + eman3 + eman4 +
eman5;
```

```
decimal mohamedScore = (decimal) mohamedSum / currentAssignments;
decimal ahmedScore = (decimal) ahmedSum / currentAssignments;
decimal khadijaScore = (decimal) khadijaSum/currentAssignments;
decimal emanScore = (decimal) emanSum / currentAssignments;

Console.WriteLine("Mohamed: " + mohamedScore + " A");
Console.WriteLine("Ahmed: " + ahmedScore + " B");
Console.WriteLine("Khadija: " + khadijaScore + " B");
Console.WriteLine("Eman: " + emanScore + " A");
```

٢. لعرض درجات الطلاب بحروف التقدير، حدد تشغيل Run
٣. للتحقق من أن التعليمات البرمجية، تعمل كما هو متوقع، قارن إخراج التطبيق الخاص بك مع الإخراج التالي:

```
Mohamed: 94.6 A
Ahmed: 83.6 B
Khadija: 83.4 B
Eman: 95.4 A
```

يجب أن تعرض تعليماتك البرمجية، درجات الطلاب كقيمة عشرية، ويجب أن ترى درجة التقدير الحرفية التي قمت بتعيينها.

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، ثم قم بتشغيل التعليمات البرمجية مرة أخرى لمعرفة ما إذا كنت قد قمت بإصلاح المشكلة، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تنتج النتائج المتوقعة.

٥ تمرين ـ تنسيق الإخراج باستخدام تسلسلات الأحرف

في هذا التمرين، ستقوم بتعديل إخراج وحدة التحكم من التمرين السابق، لتحقيق تنسيق التقارير المحدد من قبل المعلم.

تنسيق إخراج وحدة التحكم

في هذه المهمة، ستقوم بتحديث التعليمات البرمجية لتطبيق اللمسات الأخيرة على الإخراج المطلوب، ستبدأ بإضافة صف رأس يتضمن تسميات الأعمدة المحددة من قبل المعلم، بعد ذلك، ستستخدم تسلسلات أحرف الهروب لإضافة مساحة إضافية بين أعمدة المعلومات.

١. في محرر .NET. حدد موقع عبارات Console.WriteLine() المستخدمة لعرض درجة الصف الحالية لكل طالب، وحرف التقدير.
٢. إنشاء سطر فارغ أعلى العبارات Console.WriteLine()
٣. في السطر الفارغ الذي قمت بإنشائه، لإضافة عنوان لدرجات الطلاب، أدخل التعليمات البرمجية التالية:

```
Console.WriteLine("Student Grade\n");
```

لاحظ أنك تقوم بإضافة **\n** في نهاية النص، في الوحدة " تنفيذ تنسيق النصوص الأساسية باستخدام C#" تعلمت أن تسلسل أحرف الهروب **\n** سيؤدي إلى إنشاء سطر جديد، موقع الحرف مهم، في هذه الحالة **\n** يكون في نهاية المعلومات التي تريد كتابتها إلى وحدة التحكم، لذلك ستتم إضافة السطر الجديد بعد عرض "درجة الطالب"

٤. لتنسيق الإخراج كأعمدة محاذية للنص، استبدل المسافات بين الكلمات بتسلسل الهروب **\t** كما يلي:

```
Console.WriteLine("Student Grade\n");

Console.WriteLine("Mohamed:\t" +
mohamedScore + "\tA");
```

```
Console.WriteLine("Ahmed:\t" + ahmedScore +
"\tB");
Console.WriteLine("Khadija:\t" +
khadijaScore + "\tB");
Console.WriteLine("Eman:\t" + emanScore +
"\tA");
```

سيقوم تسلسل الهروب \t بإدراج علامة تبويب "مسافة مزدوجة" بين عناصر النص، يجب أن تؤدي إضافة المسافة المزدوجة إلى أعمدة معلومات محاذاة لليسار.

٥. لعرض نتائج التحديثات، حدد تشغيل.
٦. قارن إخراج التطبيق الخاص بك مع الإخراج التالي:

```
Student Grade

Mohamed:    94.6    A
Ahmed:  83.6    B
Khadija:    83.4    B
Eman:   95.4    A
```

٧. لاحظ أنه على الرغم من استخدام علامة تبويب مسافة مزدوجة، بدلاً من حرف مسافة، لا تزال بعض الأسطر لا تحتوي على مسافة بيضاء كبيرة بين اسم الطالب ودرجاته الرقمية.

يرجع هذا الاختلاف إلى الطريقة التي يتم بها تطبيق طول مساحة علامة التبويب "المسافة المزدوجة" يتم تعيين المسافة بحيز يشغل أربعة أحرف.

٨. لجعل المسافة البيضاء أكثر وضوحاً بين العمودين الأولين، أضف عمودًا آخر \t بعد اسمي الطلاب الأقصر كما يلي:

```
Console.WriteLine("Mohamed:\t" + mohamedScore + "\tA");
Console.WriteLine("Ahmed:\t\t" + ahmedScore + "\tB");
Console.WriteLine("Khadija:\t" + khadijaScore + "\tB");
Console.WriteLine("Eman:\t\t" + emanScore + "\tA");
```

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية والتحقق من صحة الإخراج.

١. قارن التعليمات البرمجية الخاصة بك إلى ما يلي:

```
int currentAssignments = 5;

int mohamed1 = 93;
int mohamed2 = 87;
int mohamed3 = 98;
int mohamed4 = 95;
int mohamed5 = 100;

int ahmed1 = 80;
int ahmed2 = 83;
int ahmed3 = 82;
int ahmed4 = 88;
int ahmed5 = 85;

int khadija1 = 84;
int khadija2 = 96;
int khadija3 = 73;
int khadija4 = 85;
int khadija5 = 79;

int eman1 = 90;
int eman2 = 92;
int eman3 = 98;
int eman4 = 100;
int eman5 = 97;
```

```
int mohamedSum = mohamed1 + mohamed2 +
mohamed3 + mohamed4 + mohamed5;
int ahmedSum = ahmed1 + ahmed2 + ahmed3 +
ahmed4 + ahmed5;
int khadijaSum = khadija1 + khadija2 +
khadija3 + khadija4 + khadija5;
int emanSum = eman1 + eman2 + eman3 + eman4
+ eman5;

decimal mohamedScore = (decimal) mohamedSum
/ currentAssignments;
decimal ahmedScore = (decimal) ahmedSum /
currentAssignments;
decimal khadijaScore = (decimal)
khadijaSum/currentAssignments;
decimal emanScore = (decimal) emanSum /
currentAssignments;

Console.WriteLine("Student Grade\n");
Console.WriteLine("Mohamed:\t" +
mohamedScore + "\tA");
Console.WriteLine("Ahmed:\t\t" + ahmedScore
+ "\tB");
Console.WriteLine("Khadija:\t" +
khadijaScore + "\tB");
Console.WriteLine("Eman:\t\t" + emanScore +
"\tA");
```

٢. للتحقق من أن التعليمات البرمجية تعمل كما هو متوقع، قارن إخراج التطبيق الخاص بك مع الإخراج التالي:

```
Mohamed:   94.6 A
Ahmed:     83.6 B
Khadija:   83.4 B
Eman:      95.4 A
```

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تنتج النتائج المتوقعة.

٦ اختبر معلوماتك

١- ماذا تنجز التعليمات البرمجية التالية؟

```
var value = (int) dividend / (int) divisor;
```

- تحويل المعاملات لاقتطاع النتيجة
- يعلن عن متغير int مكتوب بشكل صريح
- تحويل المعاملات لمنع اقتطاع النتيجة

٢- لماذا تبدو علامة التبويب "المسافة مزدوجة" مسافة واحدة في الإخراج التالي `Console.WriteLine("Student\tGrade");`

- تسلسل علامة التبويب غير صحيح
- علامة التبويب التالية بعد `Student` ببساطة مسافة واحدة فقط
- يجب مضاعفة تسلسل علامة التبويب (\t\t) لتوضيح المسافة

٣- ما الخطأ في التعليمات البرمجية التالية؟

```
int KhadijaSum; Console.WriteLine("Khadija: "
+ KhadijaSum);
```

- `KhadijaSum` لم تتم تهيئته قبل الاستخدام
- `KhadijaSum` غير مكتوب ضمنيًا إلى سلسلة
- `KhadijaSum` يجب استخدام الكلمة الأساسية var

راجع إجابتك

١ تحويل المعاملات لاقتطاع النتيجة

صحيح يتم تحويل المعاملات إلى نوع بيانات رقمي int الذي سينشئ نتيجة int مكتوبة ضمنيًا.

٢ علامة التبويب التالية بعد Student تكون ببساطة بعرض مسافة واحدة

صحيح طباعة علامة تبويب تملأ مسافة بيضاء لطول المتغير، حتى يتم الوصول إلى الموضع الصحيح

٣ لم تتم تهيئة `khadijaSum` قبل الاستخدام

صحيح يجب تعيين قيمة إلى المتغير، قبل أن تتمكن من استدعاء القيمة من المتغير

٧ الملخص

كان هدفك هو إنشاء تطبيق يسجل درجات المواد للطلاب في فصل دراسي، ويحسب درجاتهم، ويعرض النتائج.

لتحقيق ذلك، قمت بالإعلان عن القيم، وتعيينها إلى متغيرات من أنواع البيانات المختلفة، وتنفيذ عمليات رقمية وحسابية، واستخدام تحويل النوع لتحقيق نتائج دقيقة، كما استخدمت الأسلوب `Console.WriteLine()` وتسلسلات الأحرف، وتنسيق الإخراج.

من خلال حل المشكلة، تمكنت من إنشاء حل باستخدام المهارات التي تعلمتها في الوحدات السابقة.

تهانينا على تطوير تطبيق منظم!

الوحدة السادسة

المشروع الإرشادي – حساب المعدل التراكمي "GPA" النهائي

اكتساب خبرة في تطوير تطبيق يقوم بحساب النتائج وعرضها، استنادًا إلى بيانات معينة.

الأهداف التعليمية:

بعد إكمال هذه الوحدة، ستتمكن من:

- تعلم كيفية تطوير التطبيقات الأساسية عن طريق تقسيم العمل إلى أجزاء أصغر.
- تحديد أنواع البيانات المناسبة لتخزين المتغيرات والعمل معها.
- إجراء العمليات الحسابية لتحديد النتائج.
- تطبيق تقنيات تنسيق النصوص لعرض النتائج.

المتطلبات الأساسية:

- خبرة مع قواعد بناء جملة C# الأساسية `basic C# syntax rules`
- تجربة عرض رسالة إلى وحدة التحكم باستخدام أساليب `Console.WriteLine() and Console.Write()`
- خبرة في إنشاء القيم الحرفية وإعلان متغيرات بأنواع البيانات الأساسية مثل النصية string والرقمية int والعشرية decimal
- خبرة في دمج السلسلة وصياغتها.
- خبرة في إجراء العمليات الأساسية على الأرقام.

محتويات الوحدة:

١- مقدمة

٢- الاستعداد للمشروع الإرشادي

٣- تمرين - تخزين قيم الدرجات الرقمية لكل دورة تدريبية

٤- تمرين - حساب مجموع ساعات الحضور المعتمدة ونقاط الدرجات

٥- تمرين - تنسيق الإخراج العشري

٦- تمرين - تنسيق الإخراج باستخدام تنسيق الأحرف

٧- اختبر معلوماتك

٨- الملخص

١ المقدمة

يقوم المطورون بتنفيذ بعض المهام كل يوم تقريباً، بما في ذلك الإعلان عن متغيرات النصوص والأرقام، وتعيين القيم واستخراجها، وإجراء العمليات الحسابية، هذه المهام ليست روتينية فحسب، بل ضرورية أيضاً، بنفس القدر من الأهمية لمهمة توصيل النتائج إلى مستخدم التطبيق، إن إتقان القدرة على تطبيق هذه المهارات لحل مشكلة معينة، أمر يجب على كل مطور أن يتعلم القيام به.

لنفترض أنك مساعد مدرس في جامعة، وكلفت بتطوير تطبيق يساعد في حساب متوسط درجات الطلاب، يستخدم التطبيق درجات الطلاب، وساعات الحضور المستغرقة، لحساب إجمالي المعدل التراكمي "GPA" الخاص بهم، كما يتم توفير تنسيق مطلوب لعرض "GPA" للطلاب.

سترشدك هذه الوحدة خلال الخطوات المطلوبة لتطوير تطبيق حاسبة المعدل التراكمي "GPA" ستقوم التعليمات البرمجية بالإعلان عن القيم وتعيينها للمتغيرات، استنادًا إلى معلومات الدورة التدريبية، وإجراء حسابات رقمية مختلفة، وتنسيق النتائج وعرضها. تشمل العمليات الحسابية تحديد مجموع نقاط الدرجات المكتسبة، وإجمالي ساعات الحضور، لعرض النتائج بالتنسيق المطلوب، ستحتاج إلى معالجة قيمة عشرية لعرض ما مجموعه ثلاثة أرقام، ستستخدم أيضاً أساليب Console.WriteLine() بالإضافة إلى تسلسلات الأحرف التي تساعد على تنسيق نتائجك.

في نهاية هذه الوحدة، ستتمكن من كتابة التعليمات البرمجية التي تستخدم أنواع متغيرات مختلفة، وتنفذ حسابات رقمية، وتعرض البيانات المنسقة للمستخدم.

٢ الاستعداد للمشروع الإرشادي

ستستخدم محرر NET. كبيئة تطوير التعليمات البرمجية، ستقوم بكتابة التعليمات التي تستخدم المتغيرات النصية والرقمية، وإجراء العمليات الحسابية، ثم تنسيق النتائج وعرضها على وحدة تحكم.

نظرة عامة على المشروع

تقوم بتطوير حاسبة المعدل التراكمي GPA للطلاب التي ستساعد في حساب المتوسط الإجمالي لنقاط الدرجات للطلاب، معلمات التطبيق هي:

- يتم منحك اسم الطالب ومعلومات الصف.
- لكل فصل اسم، ودرجة الطالب، وعدد الساعات المعتمدة لذلك الفصل.
- يحتاج تطبيقك إلى إجراء عمليات رياضية أساسية لحساب GPA للطالب الواحد.
- يحتاج تطبيقك إلى إخراج/عرض اسم الطالب ومعلومات الفصل الدراسي والمعدل التراكمي

لحساب المعدل التراكمي GPA

- ضرب قيمة الدرجات للدورة التدريبية في عدد ساعات الاعتماد لتلك الدورة التدريبية.
- نفذ ذلك لكل دورة تدريبية، ثم أضف هذه النتائج معاً.
- قسم المجموع الناتج على إجمالي عدد ساعات الحضور.

يتم تزويدك بالعينة التالية من معلومات الدورة التدريبية للطالب وGPA

Student: Khadija Ahmed

Course	Grade	Credit Hours
English 101	4	3
Algebra 101	3	3
Biology 101	3	4
Computer Science I	3	4
Psychology 101	4	3
Final GPA:	3.35	

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي:

١. افتح بيئة ترميز محرر NET.
٢. انسخ التعليمات التالية وألصقها في محرر NET. تمثل هذه القيم اسم الطالب وتفاصيل الدورة التدريبية

```
string studentName = "Khadija Ahmed";
string course1Name = "English 101";
string course2Name = "Algebra 101";
string course3Name = "Biology 101";
string course4Name = "Computer Science I";
string course5Name = "Psychology 101";
int course1Credit = 3;
int course2Credit = 3;
int course3Credit = 4;
int course4Credit = 4;
int course5Credit = 3;
```

أنت الآن جاهز لبدء تمارين المشروع الإرشادي. حظ سعيد!

٣ تمرين ـ تخزين قيم الدرجات الرقمية لكل دورة تدريبية

في هذا التمرين، ستبدأ في إعداد المتغيرات اللازمة لحساب GPA الخاص بالطالب. لنشرع في العمل!

هام: يجب أن تكون قد أكملت إرشادات الإعداد في الدرس السابق، الإعداد، قبل بدء هذا التمرين.

إنشاء متغيرات لتخزين قيم الدرجات

في هذه المهمة، ستحدد المعادلات الرقمية للدرجة الحرفية التي حصل عليها الطالب، بعد ذلك ستقوم بإعلان المتغيرات لتخزين قيمة الدرجة الرقمية لكل فصل، يتم تمثيل المعادلات الرقمية كأرقام صحيحة، لذلك ستستخدم نوع البيانات عدد صحيح لتخزين القيم.

١. تأكد من فتح محرر NET. وأن لديك المتغيرات التي تم تجهيزها باسم الطالب وأسماء المواد/المقررات الدراسية والساعات المعتمدة.

في وحدة التحضير للمشروع الإرشادي هذا، تطلب منك تعليمات الإعداد نسخ معلومات المقرر الدراسي للطالب إلى المحرر، إذا لزم الأمر، قم بالرجوع وإكمال تعليمات الإعداد.

٢. راجع قيم المعادلات الرقمية لدرجات الحرفية التالية A = 4 نقاط درجات
B = 3 نقاط درجات
٣. مرر إلى أسفل التعليمات البرمجية، وأنشئ سطراً فارغًا.
٤. للإعلان عن متغير عدد صحيح لكل قيمة درجة رقمية، أدخل التعليمات التالية:

```
int gradeA = 4;
int gradeB = 3;
```

لاحظ استخدام القيم الثابتة لتمثيل الدرجات الرقمية، تساعد هذه التقنية على تسهيل فهم تعليماتك البرمجية، والمساعدة في منع الأخطاء المطبعية إذا كنت

بحاجة إلى إدخال درجات مختلفة بشكل متكرر، يتم حذف قيم الدرجات C, D, F في الوقت الحالي، لأنها غير مستخدمة.

٥. راجع درجات الطالب لكل دورة:

Course	Grade
English 101	A
Algebra 101	B
Biology 101	B
Computer Science I	B
Psychology 101	A

ستستخدم هذه المعلومات لإنشاء متغيرات تخزن قيم الدرجات الرقمية لكل دورة تدريبية.

٦. لإنشاء متغيرات ستخزن الدرجات لكل دورة تدريبية، أدخل التعليمات التالية في الأسفل:

```
int course1Grade = gradeA;
int course2Grade = gradeB;
int course3Grade = gradeB;
int course4Grade = gradeB;
int course5Grade = gradeA;
```

٧. لعرض أسماء الدورات التدريبية جنبًا إلى جنب مع الدرجة الرقمية، أدخل التعليمات البرمجية التالية في الأسفل:

```
Console.WriteLine($"{course1Name} {course1Grade}");
Console.WriteLine($"{course2Name} {course2Grade}");
Console.WriteLine($"{course3Name} {course3Grade}");
Console.WriteLine($"{course4Name} {course4Grade}");
Console.WriteLine($"{course5Name} {course5Grade}");
```

٨. في محرر .NET. لتشغيل تعليماتك البرمجية، حدد الزر تشغيل Run

يجب أن يتطابق إخراج تطبيقك مع الإخراج التالي:

```
English 101 4
Algebra 101 3
Biology 101 3
Computer Science I 3
Psychology 101 4
```

إذا لم يتطابق الإخراج الخاص بك، فتأكد من التحقق من أسماء المتغيرات.

٩. خذ لحظة للنظر في الإخراج الحالي، والإخراج النهائي للتطبيق.

في الإخراج النهائي لتطبيقك، تريد عرض اسم الفصل الدراسي، والدرجة، وساعات الحضور المعتمدة، هذا هو الوقت المناسب لإضافة ساعات الحضور إلى كشوف الطباعة.

١٠. لإضافة ساعات الحضور لكل فئة إلى عبارات الطباعة، قم بتحديث التعليمات البرمجية على النحو التالي:

```
Console.WriteLine($"{course1Name}    {course1Grade}    {course1Credit}");
Console.WriteLine($"{course2Name}    {course2Grade}    {course2Credit}");
Console.WriteLine($"{course3Name}    {course3Grade}    {course3Credit}");
Console.WriteLine($"{course4Name}    {course4Grade}    {course4Credit}");
Console.WriteLine($"{course5Name}    {course5Grade}    {course5Credit}");
```

راجع عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية، والتحقق من صحة الإخراج.

١. في محرر NET. حدد تشغيل
٢. راجع مخرجاتك وتحقق من صحة أسماء المقررات الدراسية والدرجات وساعات الحضور:

English 101 4 3
Algebra 101 3 3
Biology 101 3 4
Computer Science I 3 4
Psychology 101 4 3

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، ثم قم بتشغيلها مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تعرض النتائج المتوقعة.

هام

تأكد من عدم حذف أي من التعليمات البرمجية التي كتبتها حتى الآن، ستبني على هذه التعليمات البرمجية في التمرين التالي.

٤ تمرين - حساب مجموع ساعات الحضور المعتمدة ونقاط الدرجات

في هذا التمرين، ستقوم بحساب وتخزين العدد الإجمالي لساعات الحضور، وإجمالي نقاط الدرجات المكتسبة لكل فصل أو مادة، سيتم استخدام هذه القيم لاحقًا لحساب المعدل التراكمي GPA لأن كل من ساعات الحضور، وقيم الدرجات يتم تمثيلها كأرقام كاملة، فستخزن المجاميع باستخدام نوع بيانات عدد صحيح.

إنشاء متغيرات لتخزين المتوسط

تذكر لحساب GPA الخاص بالطالب، تحتاج إلى إجمالي عدد ساعات الحضور، وإجمالي عدد نقاط الدرجات المكتسبة، نقاط الدرجات المكتسبة للدورة، تساوي ناتج عدد ساعات الحضور لتلك الدورة وقيمة الدرجات الرقمية المكتسبة، على سبيل المثال:

Course	Credit	Credit Hours	Grade Points
English 101	4	3	12

في هذه المهمة، ستقوم بإنشاء المتغيرات لتخزين القيم المطلوبة لحساب المعدل التراكمي، ستقوم بإنشاء متغير لتخزين مجموع إجمالي ساعات الحضور لكل دورة، ومتغير آخر لتخزين مجموع نقاط الدرجات التي حصل عليها الطالب لكل دورة/مادة.

١. في محرر.NET. حدد موقع عبارات Console.WriteLine() المستخدمة لعرض معلومات المواد.
٢. إنشاء سطر فارغ أعلى هذه العبارات.
٣. في السطر الفارغ، لإنشاء متغير يقوم بتخزين العدد الإجمالي لساعات الحضور، أدخل التعليمات التالية:

```
int totalCreditHours = 0;
```

لاحظ أنه تمت تهيئة الإجمالي إلى **0** تسمح لك هذه التهيئة بزيادة المجموع مع الحفاظ على تنظيم تعليماتك البرمجية.

٤. لزيادة المجموع لتمثيل العدد الإجمالي للساعات، أدخل التعليمات التالية أسفل السطر الذي أضفته:

```
totalCreditHours += course1Credit;
totalCreditHours += course2Credit;
totalCreditHours += course3Credit;
totalCreditHours += course4Credit;
totalCreditHours += course5Credit;
```

تذكر أن عامل التشغيل `+=` هو رمز مختصر لإضافة قيمة إلى متغير، هذه الأسطر من التعليمات لها نفس النتيجة مثل إضافة كل متغير courseCredit على سطر واحد، على سبيل المثال:

```
totalCreditHours = course1Credit + course2Credit + course3Credit + course4Credit + course5Credit;
```

٥. لإنشاء متغير يقوم بتخزين العدد الإجمالي لنقاط الدرجات المكتسبة لكل مادة، أدخل التعليمات التالية:

```
int totalGradePoints = 0;
```

٦. لزيادة المجموع حسب نقاط الدرجات المكتسبة للمادة الأولى، أدخل التعليمات التالية:

```
totalGradePoints += course1Credit * course1Grade;
```

تذكر أن نقاط درجات التقدير للمادة تساوي الساعات المعتمدة للمادة مضروبة في الدرجة المكتسبة، في هذا السطر من التعليمات، يمكنك استخدام عامل تشغيل التعيين المركب لإضافة منتج

```
course1Credit * course1Grade to totalGradePoints
```

٧. لزيادة المجموع حسب نقاط الدرجات المكتسبة لبقية المقررات، أدخل التعليمات التالية:

```
totalGradePoints += course2Credit * course2Grade;
totalGradePoints += course3Credit * course3Grade;
totalGradePoints += course4Credit * course4Grade;
totalGradePoints += course5Credit * course5Grade;
```

٨. خذ دقيقة لمراجعة تعليماتك البرمجية.

لاحظ أن التعليمات البرمجية التي كتبتها تقسم المشكلة إلى أجزاء قابلة للإدارة بدلاً من محاولة حساب GPA في عملية كبيرة واحدة، أولاً، قمت بتهيئة وحساب قيمة totalCreditHours ثم قمت بتهيئة وحساب قيمة totalGradePoints بعد ذلك، ستستخدم هذه القيم في العملية الحسابية النهائية.

الآن بعد أن أصبحت تعليماتك البرمجية تحسب قيمة ل totalGradePoints دعنا نتحقق من صحة حساباتك قبل المتابعة، من المهم إيقاف عملك والتحقق منه بشكل دوري، سيؤدي التحقق من عملك في وقت مبكر من عملية التطوير إلى تسهيل تحديد موقع أي أخطاء في التعليمات البرمجية وإصلاحها.

٩. لعرض قيم totalGradePoints ، totalCreditHours أدخل التعليمات التالية:

```
Console.WriteLine($"{totalGradePoints} {totalCreditHours}");
```

ستقوم بإزالة عبارة WriteLine() لاحقًا لأنها غير مطلوبة في الإخراج النهائي.

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية والتحقق من صحة الإخراج.

١. تحقق من أن التعليمات البرمجية مشابهة للآتي:

```
string studentName = "Khadija Ahmed";

string course1Name = "English 101";
string course2Name = "Algebra 101";
string course3Name = "Biology 101";
string course4Name = "Computer Science I";
string course5Name = "Psychology 101";

int course1Credit = 3;
int course2Credit = 3;
int course3Credit = 4;
int course4Credit = 4;
int course5Credit = 3;

int gradeA = 4;
int gradeB = 3;

int course1Grade = gradeA;
int course2Grade = gradeB;
int course3Grade = gradeB;
int course4Grade = gradeB;
int course5Grade = gradeA;
int totalCreditHours = 0;

totalCreditHours += course1Credit;
totalCreditHours += course2Credit;
totalCreditHours += course3Credit;
totalCreditHours += course4Credit;
totalCreditHours += course5Credit;

int totalGradePoints = 0;
```

```
totalGradePoints += course1Credit * course1Grade;
totalGradePoints += course2Credit * course2Grade;
totalGradePoints += course3Credit * course3Grade;
totalGradePoints += course4Credit * course4Grade;
totalGradePoints += course5Credit * course5Grade;

Console.WriteLine($"{totalGradePoints}
{totalCreditHours}");
Console.WriteLine($"{course1Name}    {course1Grade}
{course1Credit}");
Console.WriteLine($"{course2Name}    {course2Grade}
{course2Credit}");
Console.WriteLine($"{course3Name}    {course3Grade}
{course3Credit}");
Console.WriteLine($"{course4Name}    {course4Grade}
{course4Credit}");
Console.WriteLine($"{course5Name}    {course5Grade}
{course5Credit}");
```

٢. تحقق من أن إخراج التطبيق الخاص بك يطابق الإخراج التالي:

```
57 17
English 101 4 3
Algebra 101 3 3
Biology 101 3 4
Computer Science I 3 4
Psychology 101 4 3
```

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تعرض النتائج المتوقعة.

هام: تأكد من عدم حذف أي من التعليمات البرمجية التي كتبتها حتى الآن، ستبني على هذه التعليمة البرمجية في التمرين التالي.

٥ تمرين - تنسيق الإخراج العشري

في هذا التمرين، ستقوم بحساب GPA النهائي، وتعديل إخراج وحدة التحكم لتحقيق تنسيق التقارير المطلوب، يساوي GPA مجموع نقاط الدرجات مقسوما على مجموع إجمالي ساعات الحضور.

حساب GPA النهائي

١. في محرر.NET. حدد موقع عبارات Console.WriteLine() المستخدمة لعرض معلومات الدورة التدريبية.
٢. قم بإزالة التعليمات التالية من التمرين السابق:

```
Console.WriteLine($"{totalGradePoints} {totalCreditHours}");
```

نظرا لأنك تحققت من صحة قيمك، لم تعد هناك حاجة إلى هذا السطر.

٣. إنشاء سطر فارغ أعلى عبارات Console.WriteLine()
٤. في السطر الفارغ الذي قمت بإنشائه، لتهيئة متغير سيقوم بتخزين GPA النهائي، أدخل التعليمات التالية:

```
decimal gradePointAverage = totalGradePoints / totalCreditHours;
```

٥. خذ لحظة للنظر في أنواع البيانات التي تقوم بتقسيمها.

عندما تريد أن تكون نتيجة حساب القسمة قيمة عشرية، يجب أن يكون المقسوم أو المقسوم عليه من النوع العشري (أو كليهما) عند استخدام متغيرات عدد صحيح في الحساب، تحتاج إلى استخدام عامل تشغيل التحويل، لتحويل عدد صحيح مؤقتًا إلى رقم عشري.

٦. لاسترداد قيمة عشرية من القسمة، قم بتحديث التعليمات كما يلي:

```
decimal gradePointAverage = (decimal)totalGradePoints / totalCreditHours;
```

٧. انتقل إلى عبارة Console.WriteLine() الأخيرة وأنشئ سطر فارغ جديد بعد العبارة الأخيرة.
٨. لعرض GPA النهائي، أدخل التعليمات التالية:

```
Console.WriteLine($"Final GPA: {gradePointAverage}");
```

٩. لعرض النتائج، حدد تشغيل.

قارن إخراج تطبيقك مع الإخراج التالي:

```
English 101 4 3
Algebra 101 3 3
Biology 101 3 4
Computer Science I 3 4
Psychology 101 4 3
Final GPA: 3.352941176470588235294117647
```

تنسيق الإخراج العشري

ربما لاحظت أن النتيجة العشرية تحتوي على العديد من الأرقام أكثر من GPA القياسي، في هذه المهمة، ستعالج قيمة GPA العشرية بحيث يتم عرض ثلاثة أرقام فقط.

في النهاية، تريد عرض الرقم الأول من GPA وهي نقطة عشرية، متبوعة بالرقمين الأولين بعد الفاصلة العشرية، يمكنك تحقيق هذا التنسيق باستخدام المتغيرات لتخزين الأرقام البادئة واللاحقة على التوالي، ثم طباعتها معاً باستخدام Console.WriteLine() يمكنك استخدام العمليات الرياضية التي تعلمتها لاستخراج الأرقام البادئة واللاحقة.

ملاحظة

أثناء متابعة رحلة التطوير، ستكتشف العمليات المضمنة التي يمكنها تطبيق التنسيق تلقائياً على بياناتك، في الوقت الحالي، هذه فرصة رائعة لترسيخ ما تعلمته حتى الآن، انتبه جيداً ووسع مداركك للخطوات القادمة.

١. انتقل إلى أعلى عبارات Console.WriteLine()
٢. إنشاء سطر فارغ أعلى عبارات Console.WriteLine()
٣. في السطر الفارغ الذي قمت بإنشائه، لتهيئة متغير سيقوم بتخزين الرقم البادئ من GPA أدخل التعليمات التالية:

```
int leadingDigit = (int) gradePointAverage;
```

لاحظ أنه لاستخراج الرقم البادي أو الكسور بعد الفاصلة العشرية، فإنك تقوم بإرجاعها إلى قيمة عدد صحيح، هذه طريقة بسيطة وموثوقة، لأن تحويل قيمة كسرية لن يؤدي أبدًا إلى تقريب النتيجة، بمعنى إذا كان GPA هو 2.99 فإن تحويل القيمة العشرية إلى int سيؤدي إلى 2

٤. لتهيئة متغير سيقوم بتخزين الرقمين الأولين بعد الفاصلة العشرية، أدخل التعليمات البرمجية التالية:

```
int firstDigit = (int) (gradePointAverage * 10) % 10;
```

في النصف الأول من هذه العملية، تقوم بتحريك الفاصلة العشرية إلى اليمين رقم واحد، وتحويلها إلى عدد صحيح، في النصف الثاني، يمكنك استخدام عامل التشغيل % للحصول على باقي القسمة بمقدار 10 ما يعزل الرقم الأخير في العدد الصحيح، وفيما يلي مثال على ذلك:

لنفترض gradePointAverage = 2.994573 أن تنفيذ العملية على هذه القيم سيؤدي إلى الخطوات التالية:

```
int firstDigit = (int) (2.994573 * 10) % 10;
int firstDigit = 29 % 10;
int firstDigit = 9;
```

والقيمة الناتجة ل firstDigit هي ٩ (firstDigit is 9)

بعد ذلك، ستقوم بتطبيق نفس العملية لاسترداد الرقم الثاني.

٥. أدخل التعليمات البرمجية التالية:

```
int secondDigit = (int)(gradePointAverage*100) % 10;
```

في هذا السطر، يمكنك تحريك العلامة العشرية رقمين، واستخدام عامل التشغيل % لاسترداد الرقم الأخير.

٦. لتصحيح إخراج GPA النهائي، عدل عبارة Console.WriteLine() الأخيرة كما يلي:

```
Console.WriteLine($"Final                    GPA:
{leadingDigit}.{firstDigit}{secondDigit}");
```

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التعليمات البرمجية والتحقق من صحة الإخراج، لاحظ أن سطر التعليمات ربما يكون مقسم فوق سطرين لضيق عرض الصفحة، لكن في المحرر هو سطراً واحدًا.

```
string studentName = "Khadija Ahmed";
string course1Name = "English 101";
string course2Name = "Algebra 101";
string course3Name = "Biology 101";
string course4Name = "Computer Science I";
string course5Name = "Psychology 101";

int course1Credit = 3;
int course2Credit = 3;
int course3Credit = 4;
int course4Credit = 4;
int course5Credit = 3;
```

```
int gradeA = 4;
int gradeB = 3;

int course1Grade = gradeA;
int course2Grade = gradeB;
int course3Grade = gradeB;
int course4Grade = gradeB;
int course5Grade = gradeA;

int totalCreditHours = 0;
totalCreditHours += course1Credit;
totalCreditHours += course2Credit;
totalCreditHours += course3Credit;
totalCreditHours += course4Credit;
totalCreditHours += course5Credit;

int totalGradePoints = 0;
totalGradePoints += course1Credit * course1Grade;
totalGradePoints += course2Credit * course2Grade;
totalGradePoints += course3Credit * course3Grade;
totalGradePoints += course4Credit * course4Grade;
totalGradePoints += course5Credit * course5Grade;

decimal gradePointAverage = (decimal) totalGradePoints/totalCreditHours;

int leadingDigit = (int) gradePointAverage;
int firstDigit = (int) (gradePointAverage * 10) % 10;
int secondDigit = (int) (gradePointAverage * 100 ) % 10;
```

```
Console.WriteLine($"{course1Name}        {course1Grade}
{course1Credit}");
Console.WriteLine($"{course2Name}        {course2Grade}
{course2Credit}");
Console.WriteLine($"{course3Name}        {course3Grade}
{course3Credit}");
Console.WriteLine($"{course4Name}        {course4Grade}
{course4Credit}");
Console.WriteLine($"{course5Name}        {course5Grade}
{course5Credit}");
Console.WriteLine($"Final                GPA:
{leadingDigit}.{firstDigit}{secondDigit}");
```

١. شغل التعليمات البرمجية لعرض الإخراج المنسق حدد تشغيل.
٢. للتحقق من أن تعليماتك تعمل كما هو متوقع، قارن إخراج التطبيق مع الإخراج التالي:

```
English 101 4 3
Algebra 101 3 3
Biology 101 3 4
Computer Science I 3 4
Psychology 101 4 3
Final GPA: 3.35
```

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، استمر في تعديل التعليمات البرمجية وتشغيلها حتى تعرض النتائج المتوقعة.

هام

تأكد من عدم حذف أي من التعليمات البرمجية التي كتبتها حتى الآن. ستبني على هذه التعليمة البرمجية في التمرين التالي.

٦ تمرين ـ تنسيق الإخراج باستخدام تنسيق الأحرف

في هذا التمرين، ستقوم بتعديل إخراج وحدة التحكم من التمرين السابق لتحقيق تنسيق التقارير المحدد.

تنسيق إخراج وحدة التحكم

في هذه المهمة، ستقوم بتحديث التعليمات البرمجية لتطبيق اللمسات الأخيرة على الإخراج المطلوب، ستبدأ بتضمين اسم الطالب، وإضافة صف رأسي، يتضمن تسميات الأعمدة، بعد ذلك، ستستخدم تنسيق الأحرف لإضافة مساحة إضافية بين أعمدة المعلومات.

١. في محرر.NET. حدد موقع عبارات Console.WriteLine() المستخدمة لعرض معلومات المواد للطالب.
٢. إنشاء سطر فارغ أعلى العبارات.
٣. في السطر الفارغ الذي قمت بإنشائه لإضافة اسم الطالب، أدخل التعليمات التالية:

```
Console.WriteLine($"Student: {studentName}\n");
```

لاحظ أنك تقوم بإضافة تنسيق سطر **\n** في نهاية النص، في الوحدة تنفيذ تنسيق الجمل الأساسية باستخدام C# تعلمت أن تنسيق الحرف **\n** سيؤدي إلى إنشاء سطر جديد، موقع تسلسل التنسيق مهم، في هذه الحالة **\n** يكون في نهاية المعلومات التي تريد كتابتها إلى وحدة التحكم، لذلك ستتم إضافة السطر الجديد بعد عرض "اسم الطالب"

٤. إنشاء سطر فارغ بعد عبارة Console.WriteLine() السابقة.
٥. في السطر الفارغ الذي قمت بإنشائه، لإضافة عنوان لمعلومات المادة أو الدورة التدريبية، أدخل التعليمات التالية:

```
Console.WriteLine("Course\t\t\t\tGrade\tCredit Hours");
```

لاحظ أنك تضيف أربع علامات تبويب بعد اسم المادة، ستسمح العلامات الإضافية بمسافة إضافية لأسماء المواد الأطول.

٦. لتنسيق الإخراج كأعمدة محاذاة للنص، استبدل المسافات بين الكلمات بتسلسل الحرف **\t** كما يلي:

```
Console.WriteLine($"{course1Name}\t\t\t{cour
se1Grade}\t\t{course1Credit}");
Console.WriteLine($"{course2Name}\t\t\t{cour
se2Grade}\t\t{course2Credit}");
Console.WriteLine($"{course3Name}\t\t\t{cour
se3Grade}\t\t{course3Credit}");
Console.WriteLine($"{course4Name}\t{course4G
rade}\t\t{course4Credit}");
Console.WriteLine($"{course5Name}\t\t{course
5Grade}\t\t{course5Credit}");
```

سيدرج تنسيق الأحرف **\t** مسافة مزدوجة "علامة تبويب" بين عناصر النص، يجب أن تؤدي علامات الجدولة هذه إلى أعمدة معلومات محاذاتها لليسار.

لاحظ أنك تقلل من عدد علامات التبويب ل course4Name course5Name ويرجع هذا الاختلاف إلى أن أسماء المواد هذه أطول من غيرها، في الخطوة السابقة، أضفت علامات تبويب إضافية بعد عمود اسم المواد للحفاظ على تناسق التباعد لكل من أسماء المواد الطويلة والقصيرة.

٧. لإضافة مساحة إلى إخراج نتيجة GPA النهائي، قم بتحديث التعليمات كما يلي:

```
Console.WriteLine($"\nFinal       GPA:\t\t\t
{leadingDigit}.{firstDigit}{secondDigit}");
```

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل تعليماتك البرمجية والتحقق من صحة الإخراج. تحقق من أن التعليمات البرمجية مشابهة للآتي:

```
string studentName = "Khadija Ahmed";
string course1Name = "English 101";
string course2Name = "Algebra 101";
string course3Name = "Biology 101";
string course4Name = "Computer Science I";
string course5Name = "Psychology 101";

int course1Credit = 3;
int course2Credit = 3;
int course3Credit = 4;
int course4Credit = 4;
int course5Credit = 3;

int gradeA = 4;
int gradeB = 3;

int course1Grade = gradeA;
int course2Grade = gradeB;
int course3Grade = gradeB;
int course4Grade = gradeB;
int course5Grade = gradeA;

int totalCreditHours = 0;
totalCreditHours += course1Credit;
totalCreditHours += course2Credit;
totalCreditHours += course3Credit;
```

```
totalCreditHours += course4Credit;
totalCreditHours += course5Credit;

int totalGradePoints = 0;
totalGradePoints    +=    course1Credit    *
course1Grade;
totalGradePoints    +=    course2Credit    *
course2Grade;
totalGradePoints    +=    course3Credit    *
course3Grade;
totalGradePoints    +=    course4Credit    *
course4Grade;
totalGradePoints    +=    course5Credit    *
course5Grade;

decimal    gradePointAverage    =    (decimal)
totalGradePoints/totalCreditHours;

int leadingDigit = (int) gradePointAverage;
int firstDigit = (int) (gradePointAverage *
10 ) % 10;
int secondDigit = (int) (gradePointAverage *
100 ) % 10;

Console.WriteLine($"Student: {studentName}\n");
Console.WriteLine("Course\t\t\tGrade\tCred
it Hours");

Console.WriteLine($"{course1Name}\t\t\t{cour
se1Grade}\t\t{course1Credit}");
```

```
Console.WriteLine($"{course2Name}\t\t\t{cour
se2Grade}\t\t{course2Credit}");
Console.WriteLine($"{course3Name}\t\t\t{cour
se3Grade}\t\t{course3Credit}");
Console.WriteLine($"{course4Name}\t\t{course
4Grade}\t\t{course4Credit}");
Console.WriteLine($"{course5Name}\t\t\t{cour
se5Grade}\t\t{course5Credit}");
Console.WriteLine($"\nFinal        GPA:\t\t\t
{leadingDigit}.{firstDigit}{secondDigit}");
```

١. للتحقق من أن تعليماتك البرمجية تعمل كما هو متوقع، قارن إخراج تطبيقك مع الإخراج التالي:

```
Student: Khadija Ahmed

Course                  Grade  Credit Hours
English 101             4              3
Algebra 101             3              3
Biology 101             3              4
Computer Science I      3              4
Psychology 101          4              3

Final GPA:              3.35
```

إذا كانت تعليماتك البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها، للعثور على الخطأ، وإجراء التعديلات، استمر في تعديل التعليمات وتشغيلها حتى تعرض النتائج المتوقعة.

٧ اختبر معلوماتك

١- لنفترض ان `decimal gradePointAverage = 3.99872831;` ما هي قيمة (int) gradePointAverage

- ٣,٩٩
- ٤
- ٣

٢-في التعليمات البرمجية التالية `decimal x = 7 / 5;` لماذا x = 1

- يؤدي استخدام نوع البيانات decimal إلى اقتطاع الكسر
- يؤدي تقسيم عددين صحيحين إلى تقسيم عدد صحيح
- يحتوي هذا السطر من التعليمات البرمجية على خطأ

راجع إجابتك

٣

صحيح يتم حذف باقي القسمة

يؤدي تقسيم عددين صحيحين إلى تقسيم عدد صحيح

صحيح سيؤدي إجراء قسمة عدد صحيح إلى نتيجة عدد صحيح

٨ الملخص

كان هدفك هو إنشاء تطبيق يأخذ معلومات المواد للطالب، ويحسب إجمالي GPA ويعرض النتائج.

لتحقيق ذلك، قمت بالإعلان عن القيم، وتعيينها إلى متغيرات من أنواع البيانات المختلفة، وتنفيذ عمليات رقمية، واستخدام تحويل النوع لتحقيق نتائج دقيقة، كما استخدمت أسلوب WriteLine() وتسلسلات الأحرف لتنسيق الإخراج.

الفصل الثاني

إنشاء وتشغيل تطبيقات وحدة تحكم بسيطة باستخدام C#

استخدم Visual Studio Code لتطوير تطبيقات وحدة تحكم C# التي تنفذ المصفوفات، وحلقات `foreach` وعبارات `if`

المحتويات

الوحدة الأولى:

تثبيت وتكوين Visual Studio Code

الوحدة الثانية:

أساليب الاستدعاء من مكتبة فئاتNET. باستخدام لغة C#

الوحدة الثالثة:

إضافة منطق القرار إلى التعليمات البرمجية باستخدام عبارات "if" و"else" و"else if"

الوحدة الرابعة:

التخزين والتكرار من خلال تسلسل البيانات باستخدام المصفوفات وعبارة foreach

الوحدة الخامسة:

تنسيق التعليمات البرمجية، إنشاء اصطلاحات ومسافات بيضاء وتعليقات في C#

الوحدة السادسة:

مشروع التحدي - تطوير هياكل foreach, if-elseif-else لمعالجة بيانات المصفوفة في C#

الوحدة الأولى

تثبيت وتكوين Visual Studio Code

تعرف على كيفية تكوين Visual Studio Code ل C# وكيفية استخدام بيئة التطوير المتكاملة (IDE) الاحترافي لإنشاء تطبيقات وحدة التحكم وتشغيلها.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- تنزيل Visual Studio Code وتثبيته
- استكشف واجهة مستخدم Visual Studio Code
- تكوين Visual Studio Code لاستخدام ملحق C#
- تثبيت مكتبة وقت التشغيل .NET runtime.
- إنشاء تطبيق وحدة تحكم (New Console) وتحريره وبنائه وتشغيله باستخدام Visual Studio Code

محتويات الوحدة:

١ المقدمة

تطوير البرامج هو أكثر من مجرد كتابة تعليمات برمجية، إنها عملية، يمكن أن تستغرق مشاريع الترميز الأكبر شهورًا أو حتى سنوات لإكمالها، ويمكن أن تكون معقدة، تعد عملية كتابة التعليمات البرمجية، وتصحيح الأخطاء واختبارها، وتحديثها، وإصدارها مهمة كبيرة، للمساعدة في هذه العملية، يستخدم المطورون أداة متخصصة تعرف باسم بيئة التطوير المتكاملة (IDE)

يتضمن IDE عادة مجموعة من الأدوات التي تدعم عملية تطوير البرامج من البداية إلى النهاية، وهي عملية تعرف باسم دورة حياة التطوير، تمكن أدوات IDE المطور من العمل بكفاءة أكبر، ويمكن أن تساعد المطور أو فريق المطورين على كتابة التعليمات البرمجية الخاصة بهم، وتصحيحها واختبارها، وإصدارها بسهولة أكبر، إن IDE جيد، وهو أفضل صديق للمبرمج.

لنفترض أنك مهتم بالبدء في تطوير تطبيقات C# خطوتك الأولى هي تحديد بيئة برمجة _محرر الأكواد الذي سوف تبني من خلاله برنامجك_ بعد البحث في الخيارات المتوفرة عبر الإنترنت، يمكنك تحديد أن Visual Studio Code هو أحد محرري التعليمات البرمجية الأكثر شيوعًا بين مطوري C# يسرك أن ترى أن Visual Studio Code سريع وسهل التثبيت، وأنه يدعم العديد من الملحقات لتحسين إنتاجية المطورين.

يمكنك أيضًا إلقاء نظرة على منتج Visual Studio الكامل، والذي يوفر المزيد من الميزات للمطورين المحترفين، يتضمن كلا المنتجين خيارًا مجانيًا، لنعتبر وبعد استشارة صديق مطور، تقرر أن Visual Studio Code سوف يوفر حاليًا جميع الأدوات التي تحتاجها للبدء.

في هذه الوحدة، يمكنك تثبيت وتكوين Visual Studio Code لتلبية متطلبات تعليماتك البرمجية، يمكنك التعرف على اللوحات والقوائم المختلفة، التي تتضمن واجهة مستخدم Visual Studio Code بالإضافة إلى كيفية استخدامها ومتى تستخدمها، والأهم من ذلك، ستتعلم كيفية تكوين Visual

Studio Code لتطوير C# وكيفية إنشاء تطبيق C# وبنائه وتشغيله وتحديثه في Visual Studio Code

في نهاية هذه الوحدة، ستتمكن من استخدام Visual Studio Code لإنشاء تطبيقات تحكم C# الخاصة بك وتشغيلها.

---

٢ تنزيل Visual Studio Code وتثبيته

Visual Studio Code هو محرر تعليمات برمجية مفتوح المصدر، خفيف الوزن لكنه قوي وفعال، يعمل على أجهزة الكمبيوتر، متاح ل Windows، macOS، Linux يأتي مع دعم مضمن ل JavaScript، TypeScript، Node.js يحتوي على نظام بيئي غني من الامتدادات للغات وأوقات التشغيل runtimes الأخرى (مثل C++، C#، Java، Python، PHP، Go، .NET).

تنزيل Visual Studio Code

يعد بدء استخدام Visual Studio Code وتشغيله أمرًا سريعًا وسهلًا، تنزيل صغير وتتمكن من تثبيته في غضون دقائق.

١. افتح نافذة مستعرض جديدة، ثم انتقل إلى العنوان:

https://code.visualstudio.com

يمكنك استخدام أي مستعرض للتنزيل.

٢. في نافذة المستعرض، حدد **Download for Windows**

ملاحظة

تكشف صفحة تنزيل Visual Studio Code تلقائيًا عن نظام التشغيل الخاص بك. يعرض الإصدار المراد تنزيله لنظام التشغيل، مثل Linux أو macOS أو Windows

٣. انتظر حتى ينتهي ملف المثبت من التنزيل.
سيكون اسم ملف المثبت (Windows) مشابها لما يلي:

```
VSCodeUserSetup-x64-1.81.0.exe
```

ملاحظة

يعتمد اسم الملف على الإصدار الحالي من Visual Studio Code ونظام تشغيل الكمبيوتر الخاص بك.

تثبيت التطبيق

Visual Studio Code خفيف الوزن، يعمل على معظم إصدارات الأجهزة والنظم الأساسية المتوفرة. يمكنك مراجعة متطلبات النظام للتحقق مما إذا كان تكوين الكمبيوتر مدعومًا أم لا.

١. على جهاز الكمبيوتر الخاص بك، افتح تطبيق مستكشف الملفات، ثم انتقل إلى مجلد التنزيلات بالكمبيوتر.

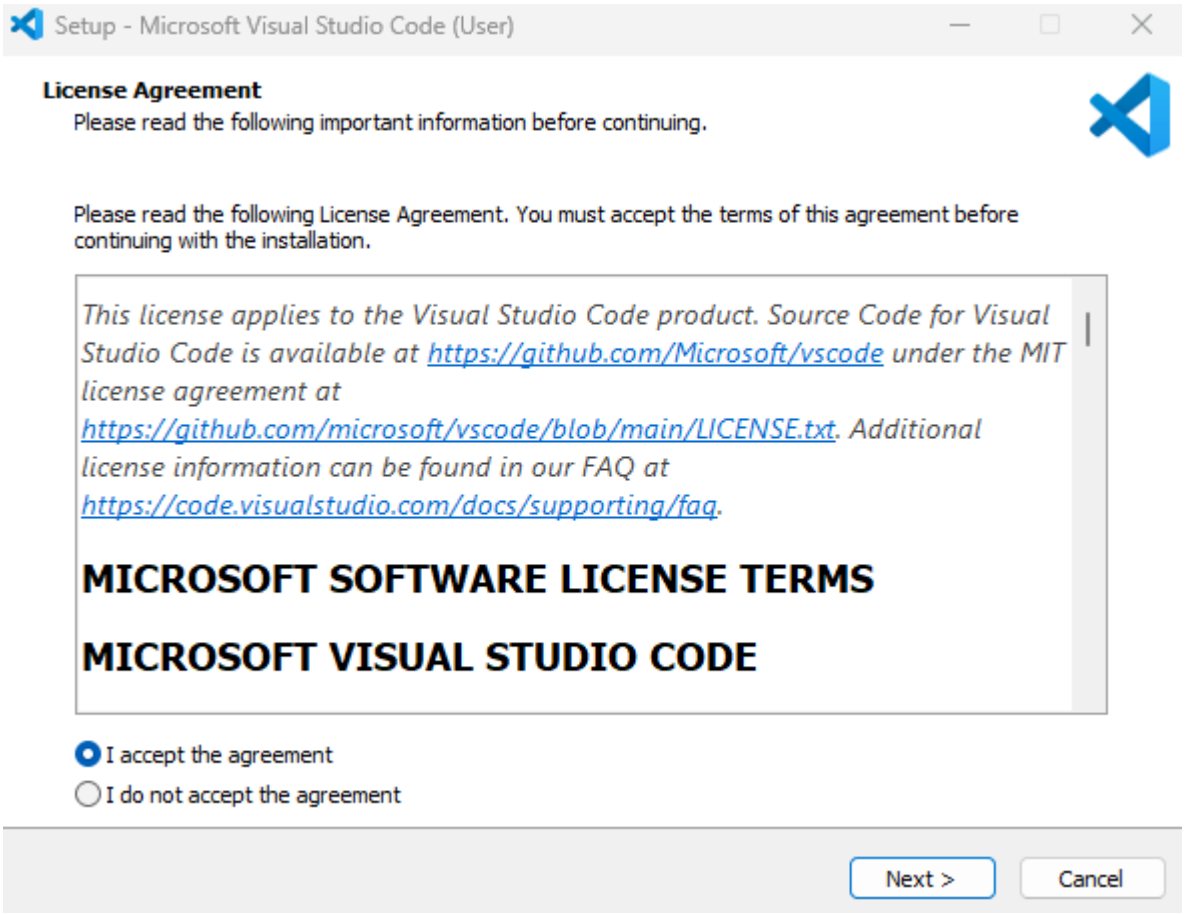
٢. في تطبيق مستكشف الملفات، حدد ملف مثبت Visual Studio Code وقم بتشغيله.

على نظام Windows يمكنك النقر نقراً مزدوجاً بالفأرة، فوق ملف البرنامج، لبدء عملية التثبيت. على سبيل المثال، انقر نقراً مزدوجاً فوق **VSCodeUserSetup-x64-1.81.0**

ملاحظة

يمكنك تثبيت Visual Studio Code للمستخدم الحالي فقط أو للنظام لجميع المستخدمين. مثبت المستخدم هو الخيار الموصي به لمعظم المستخدمين.

بعد لحظة، ستظهر نافذة الحوار Setup - Microsoft Visual Studio Code



١. حدد أوافق على اتفاقية الترخيص، ثم اتبع الإرشادات لإكمال التثبيت.

اقبل الخيارات الافتراضية من الجزء المتبقي من التثبيت.

للحصول على إرشادات التثبيت التفصيلية، راجع صفحة وثائق Visual Studio https://code.visualstudio.com/docs/setup/windows

٢. إذا تم فتح Visual Studio Code في نهاية التثبيت، قم بإغلاقه.

ستقوم بفحص Visual Studio Code لاحقًا في هذه الوحدة

تهانينا، لقد قمت بتثبيت Visual Studio Code بيئة التطوير المتكاملة التي ستستخدمها لكتابة تطبيقات C#. خطوتك التالية هي تثبيت أداة .NET. مكتبة التعليمات البرمجية المطلوبة لتشغيل تطبيقات C#

٣ تثبيت .NET SDK

.NET. هو نظام أساسي للمطورين من خلال محرر مفتوح المصدر، عبر أنظمة التشغيل المعروفة مثل Windows يمكن استخدامه لتطوير أنواع مختلفة من التطبيقات، يتضمن لغات البرمجة، ومكتبات التعليمات البرمجية المستخدمة لتطوير تطبيقات .NET. يمكنك كتابة تطبيقات .NET. في C# أو F# أو Visual Basic

أداة وقت التشغيل .NET runtime. هي مكتبة تعليمات برمجية مطلوبة لتشغيل تطبيقات C# يشار إليها أيضًا باسم أثناء تشغيل اللغة أو CLR

أداة وقت تشغيل .NET runtime. غير مطلوبة أثناء كتابة التعليمات البرمجية، لكنها مطلوبة لتشغيل التطبيقات فعليًا بعد بنائها.

تحقق لمعرفة ما إذا كان .NET runtime. مثبتا بالفعل

١. افتح تطبيق موجه **أوامر Windows** أو أي تطبيق آخر يعرض موجه أوامر terminal

يمكن العثور على تطبيقات موجه الأوامر وWindows PowerShell على قائمة Windows يمكنك أيضًا فتح أي من هذين التطبيقين من شريط مهام Windows عن طريق إدخال cmd أو powershell على التوالي.

يجب أن يكون السطر المعروض داخل موجه الأوامر مثل:

```
C:\Users\AccountName>
```

٢. في موجه الأوامر اكتب: `dotnet --version` ثم اضغط على المفتاح Enter

يعرض هذا الأمر رقم الإصدار إذا تم تثبيت .NET. على جهاز الكمبيوتر الخاص بك.

٣. خذ لحظة لعرض الاستجابة لأمرك `dotnet --version`

إذا تلقيت رسالة خطأ تخبرك بعدم التعرف على المصطلح dotnet فلن يكون لديك .NET SDK مثبتا.

إذا تم تثبيت إصدار من .NET. يجب أن تشاهد رسالة استجابة تسرد رقم الإصدار مشابها للآتي:

```
8.0.100
```

٤. أغلق نافذة موجه الأوامر.

إذا لم تكن حزمة تطوير البرامج .NET مثبتة سابقاً، فقم بتثبيت أحدث نسخة صادرة حتى الآن .NET 8 Software Development Kit. سأشرح لك الطريقة.

تثبيت .NET 8 Software Development Kit.

يوصى بتثبيت أحدث إصدار مدعوم .NET. على الرغم من أن .NET 6 و8 مدعومان أيضًا، فإن .NET 8 يتضمن تحسينات قد تجدها مفيدة.

١. افتح نافذة مستعرض إنترنت.

٢. افتح صفحة التنزيل .NET SDK. انتقل إلى العنوان الالكتروني التالي:

https://dotnet.microsoft.com/download

٣. في **صفحة Download .NET** حدد إصدار .NET 8. من .NET SDK

يقوم المستعرض بتنزيل ملف مثبت .NET SDK.

٤. انتظر حتى ينتهي ملف التثبت من التنزيل.

٥. قم بتشغيل ملف التثبت .NET SDK.

على جهاز Windows يمكنك العثور على مجلد التنزيلات باستخدام مستكشف الملفات، انقر نقرًا مزدوجًا فوق ملف التثبيت لبدء عملية التثبيت.

٦. في نافذة .NET SDK Installer. حدد **Install**
٧. انتظر حتى تكتمل عملية التثبيت.

يجب أن يستغرق التثبيت حوالي دقيقة. بمجرد اكتمال التثبيت، تظهر رسالة تؤكد نجاح التثبيت.

٨. حدد إغلاق، لإغلاق نافذة المثبت.

التحقق من التثبيت كما شرحنا سابقاً

١. افتح تطبيقا يعرض موجه أوامر terminal
تحتاج إلى فتح نافذة موجه أوامر جديدة بعد تثبيت .NET SDK للتأكد من اكتشافه.

٢. في موجه الأوامر، اكتب `dotnet --version` ثم اضغط على المفتاح Enter
٣. تحقق من إدراج رقم إصدار .NET 8.
يجب أن يكون رقم الإصدار المعروض مشابها ل: `8.0.100`

من المحتمل أن ترى رقم إصدار أحدث، لكن يجب أن تبدأ `8.` إذا قمت بتثبيت .NET 8 SDK.

تهانينا على تثبيت .NET. أنت الآن جاهز لبدء العمل في Visual Studio Code

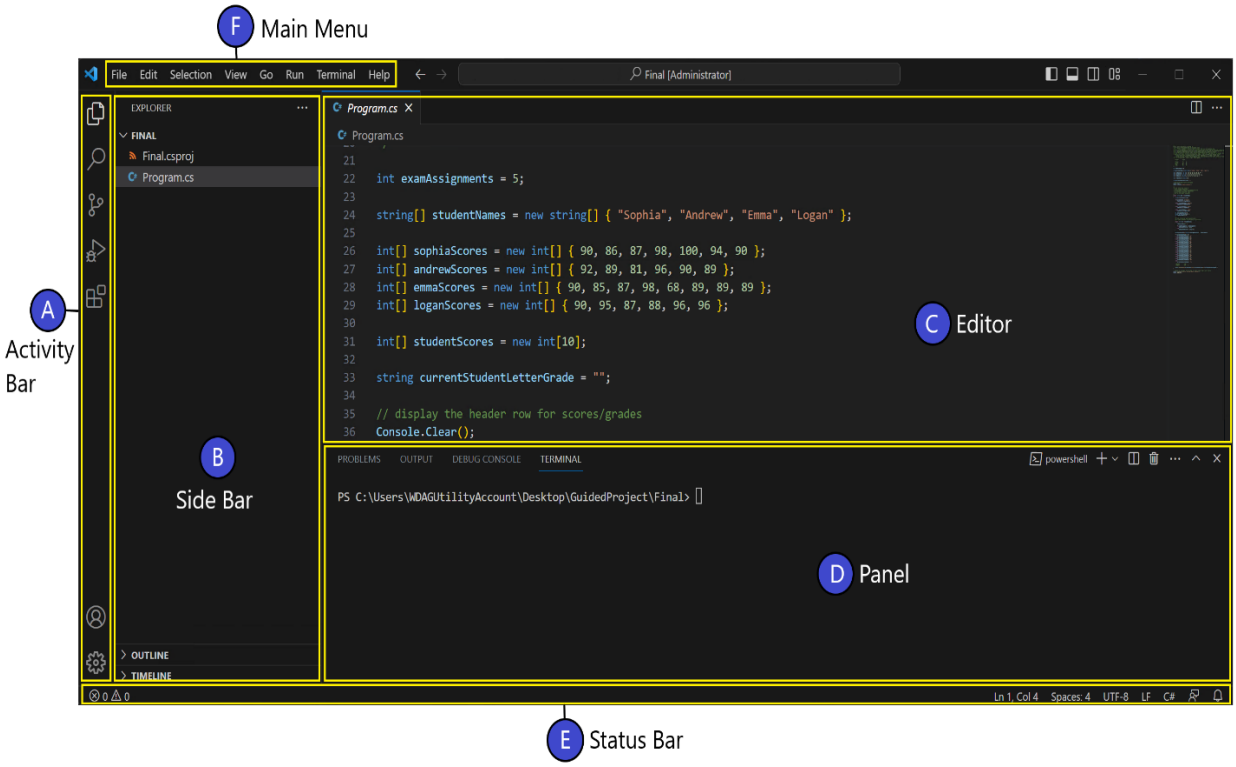
٤ فحص واجهة مستخدم Visual Studio Code

قد يتساءل البعض لماذا أجلنا شرح برنامج **Visual Studio Code** للفصل الثاني، من الأفضل للمبتدئين العمل على محرر المستعرض لأنه يغنيهم عن تفاصيل كثيرة.

Visual Studio Code هو محرر تعليمات برمجية، مع واجهة مستخدم وتخطيط مشابه للعديد من محرري التعليمات البرمجية الاخرى، على الجانب الأيمن من الواجهة يوجد شريط جانبي ستستخدمه للوصول إلى الملفات والمجلدات في مشروعك البرمجي، على الجانب الأيمن توجد منطقة محرر تعرض محتويات ملفات التطبيق.

فحص المناطق الرئيسية لواجهة المستخدم

يوضح الرسم التخطيطي التالي الأقسام الرئيسية الستة المضمنة في واجهة مستخدمVisual Studio Code



A – **قائمة النشاط** هو الشريط العمودي الضيق الموجود في أقصى الجانب الأيسر من النافذة، يتيح لك شريط النشاط التبديل بين طرق العرض (مثل المستكشف أو الملحقات) ويمنحك مؤشرات أخرى خاصة بالسياق.

B - **الشريط الجانبي** يحوي على طرق عرض، توفر الأدوات والخيارات، وأسماء ومسارات المشاريع، تكون طرق عرض الشريط الجانبي مثل المستعرض EXPLORER مفيدة جداً عند العمل على المشاريع البرمجية.

C - **المحرر** هو المنطقة المستخدمة لكتابة التعليمات البرمجية، وتحرير ملفاتك، يمكنك فتح أكبر عدد تريده من المحررات جنباً إلى جنب عمودياً وأفقياً.

D - **تستخدم منطقة اللوحة أو قائمة موجه الأوامر** لعرض لوحات مختلفة أسفل منطقة المحرر للحصول على معلومات الإخراج، عرض نتيجة التعليمات البرمجية، وإظهار رسائل الأخطاء والتحذيرات، تعتبر وحدة إخراج متكاملة.

E - **قائمة المعلومات** هو الشريط الأفقي الموجود أسفل النافذة الذي يعرض معلومات حول المشروع المفتوح والملفات التي تقوم بتحريرها.

F - **القائمة الرئيسية** هي واجهة القائمة في أعلى نافذة التطبيق، تمكنك خيارات القائمة من إنشاء مشروع جديد، وفتح الملفات البرمجية، وتحريرها، وتشغيلها، وحفظها، وأكثر من ذلك بكثير، سوف نتعرف عليه لاحقاً.

عندما تصبح أكثر كفاءة مع Visual Studio Code سوف تستعمل للبدء كل نوافذ الشاشة الموضحة، ومع ذلك، فإن أهم النوافذ التي يجب تذكرها **هي قائمة النشاط والشريط الجانبي والمحرر**، قائمة النشاط هي المكان الذي تحدد فيه النشاط الذي تريد تنفيذه، يعرض الشريط الجانبي الأدوات والخيارات، ويعرض المحرر المعلومات التي تقوم بتحريرها، عادة تعليماتك البرمجية.

٥ تمرين - استكشاف واجهة مستخدم Visual Studio Code

يوفر Visual Studio Code أدوات للمطورين الذين بدأوا للتو، ولكنه أيضاً قابل للتوسع، ومتقدم بما يكفي للمطورين المحترفين.

في هذا التمرين، يمكنك فتح Visual Studio Code وإكمال جولة سريعة في واجهة المستخدم.

افتح Visual Studio Code وافحص صفحة الترحيب

١. استخدم القائمة Windows لفتح Visual Studio Code

إذا فتحت القائمة Windows يجب أن تشاهد Visual Studio Code مدرجاً كتطبيق تمت إضافته مؤخراً، يمكنك أيضاً التمرير لأسفل للعثور عليه.

خيار آخر هو كتابة Visual Studio Code في مربع Search أسفل الشاشة.

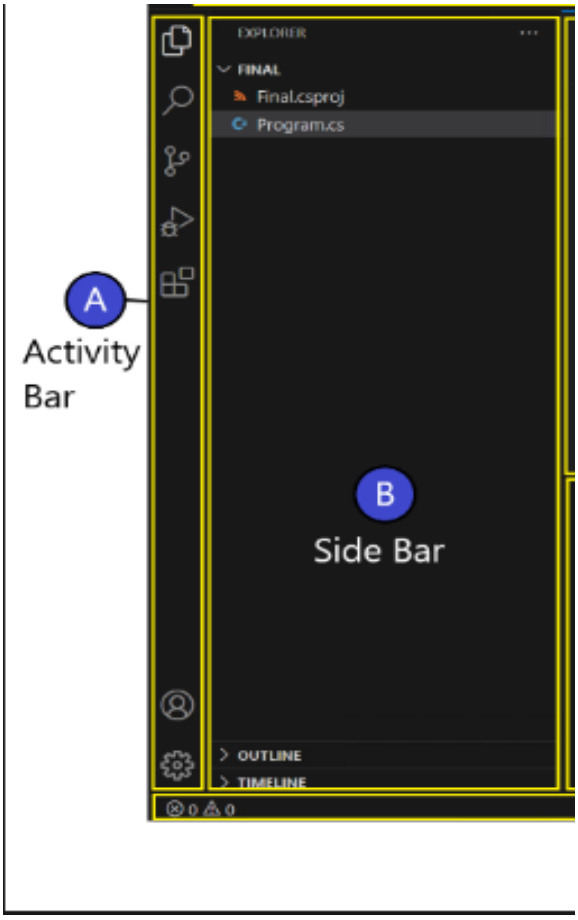
٢. لاحظ أن Visual Studio Code يفتح على صفحة "Welcome" التي تتضمن بعض الخيارات ومعلومات أخرى.

في المرة الأولى التي تفتح فيها Visual Studio Code تعرض صفحة Welcome بعض الإرشادات التفصيلية المفيدة، مثل محتوى **Get Started with VS Code** يمكنك فحص هذه المعلومات لاحقاً في وقت فراغك.

٣. لإغلاق صفحة الترحيب، حدد الزر إغلاق (يظهر X في الواجهة)

تتضمن كل صفحة مفتوحة في المحرر زر إغلاق (**X**) الموجود على يمين عنوان الصفحة، تظهر علامة تبويب صفحة الترحيب، في الجزء العلوي الأيسر من نافذة Visual Studio Code أسفل القائمة الرئيسية، إذا قمت بتمرير مؤشر الماوس فوق **X** فستظهر الكلمة **إغلاق**.

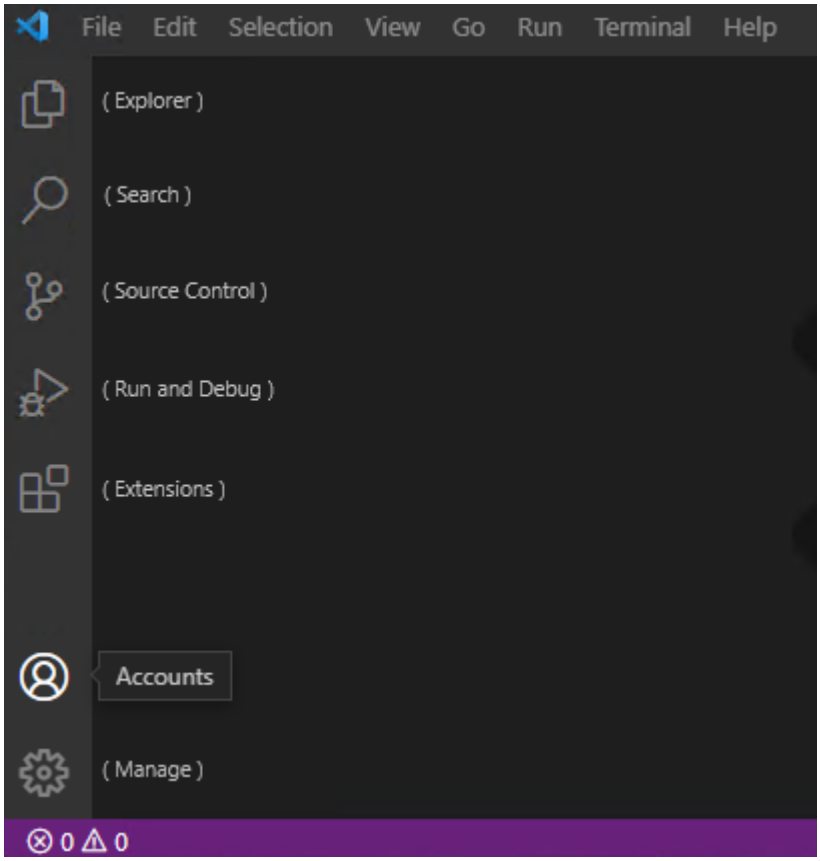
فحص قائمة النشاط والشريط الجانبي



تذكر أن **قائمة النشاط** هي الشريط العمودي للأيقونات في أقصى يسار نافذة Visual Studio Code تعتمد محتويات **الشريط الجانبي** على ما تم تحديده حاليا من **قائمة النشاط**.

١. لاحظ في الصورة التالية أنه تم طي الشريط الجانبي إلى يمين قائمة النشاط.
٢. ضع مؤشر الماوس فوق قائمة النشاط، ثم مرر مؤشر الماوس فوق كل أيقونة لعرض التسميات.

يظهر الاسم عند تمرير الماوس فوق الأيقونات، يجب أن تشاهد أزرار قائمة النشاط الموضحة في الصورة التالية:



من أعلى إلى أسفل، أيقونات قائمة النشاط هي: **Explorer، Search، Source Control، Run and Debug، Extensions، Accounts، Manage**

٣. في قائمة النشاط، حدد مستكشف.
يجب أن يفتح الشريط الجانبي المعلومات السياقية ويعرضها.

٤. لاحظ أن الشريط الجانبي يسمى الآن المستعرض أو المستكشف **EXPLORER**
يتم استخدام طريقة عرض EXPLORER للوصول إلى استكشاف مجلدات المشروع وملفات التعليمات البرمجية.

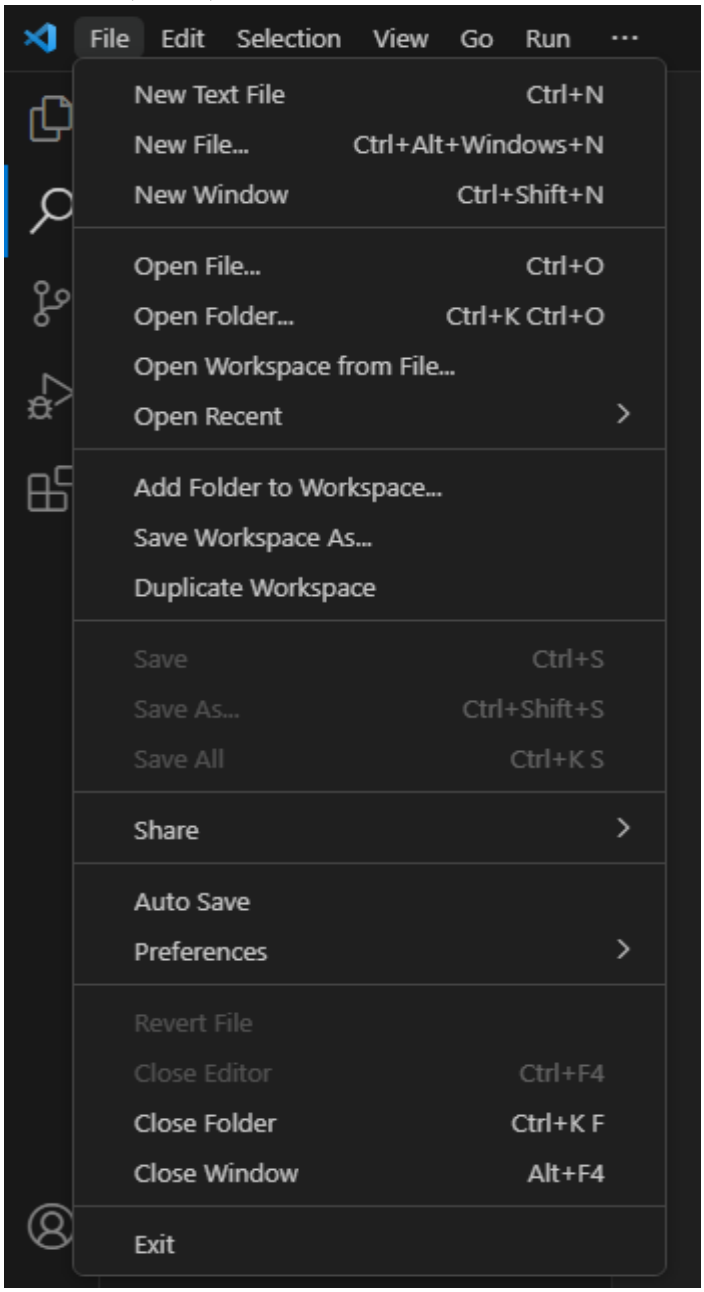
سيتذكر Visual Studio Code محفوظات عملك، ويفتح أحدث ملفات المشروع عند فتحها، لأن هذه المرة الأولى التي تفتح فيها Visual Studio Code فلن يتم فتح أي مجلد مشروع.

٥. في شريط النشاط، حدد **Extensions**
لاحظ أن الشريط الجانبي يسمى الآن EXTENSIONS خذ وقتك لفحص المعلومات المعروضة في طريقة عرض EXTENSIONS

تمكنك ملحقات Visual Studio Code من إضافة لغات البرمجة، ومصححي الأخطاء، والأدوات الأخرى إلى بيئة المحرر، لدعم سير عمل تطوير البرامج، سوف تقوم بتثبيت ملحق C# لاحقاً في هذه الوحدة.

فحص خيارات القائمة الرئيسية العلوية

١. لعرض خيارات القائمة ملف (File) حدد ملف.



لاحظ الخيارات جديد New وفتح Open وحفظ Save وإغلاق Close المدرجة في القائمة ملف File

خذ وقتك لفحص خيارات القائمة تحرير Edit ثم كل عنصر من عناصر القائمة الرئيسية العلوية الأخرى. لاحظ أن العديد من القوائم تتضمن خيارات للتفاعل مع التعليمات البرمجية الخاصة بك.

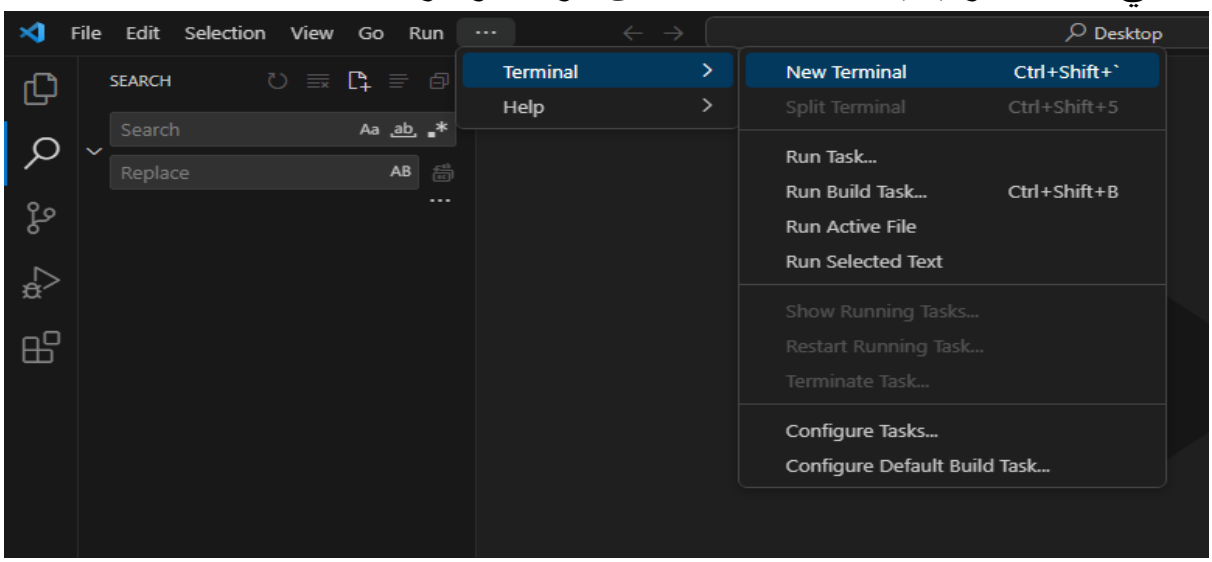
على سبيل المثال:

تتضمن القائمة تحرير خيارات تعليقات التعليمات البرمجية، واستبدالها، وتبديلها، بالإضافة إلى القص والنسخ واللصق، والتراجع والإعادة.

تتضمن قائمة التحديد Selection خيارات لتحديد خطوط التعليمات البرمجية ومعالجتها.

تتضمن قائمة تشغيل Run خيارات لتشغيل التطبيق وتصحيحه.

٢. في القائمة الرئيسية لاحظ أنها تتضمن موجه أوامر Terminal

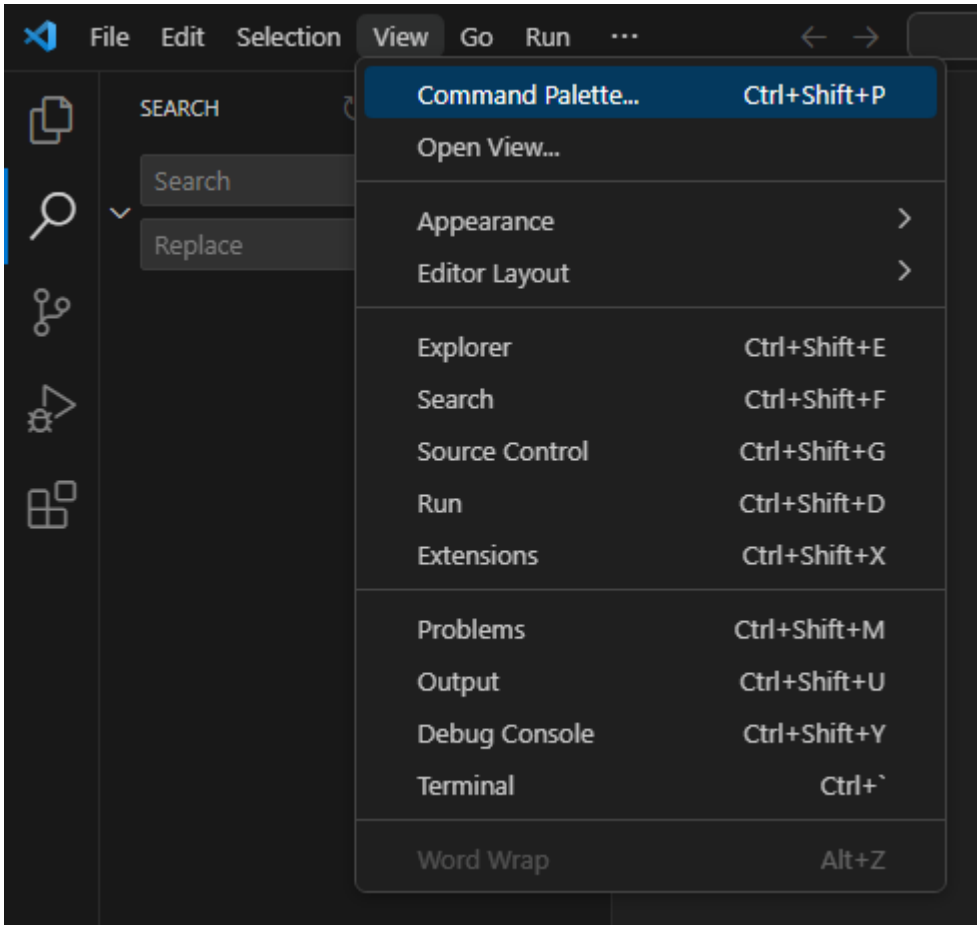


يمكن استخدام قائمة Terminal لتشغيل أوامر واجهة، سطر الأوامر (CLI) ستستخدم أوامر .NET CLI. لاحقاً في هذه الوحدة. خذ وقتك لفحص واجهة Terminal

يمكنك التبديل بين علامات التبويب (المشاكل، الإخراج، تتبع الأخطاء، قائمة Terminal) مرر مؤشر الماوس فوق الأزرار (أعلى اليمين) لعرض تسميات الأزرار.

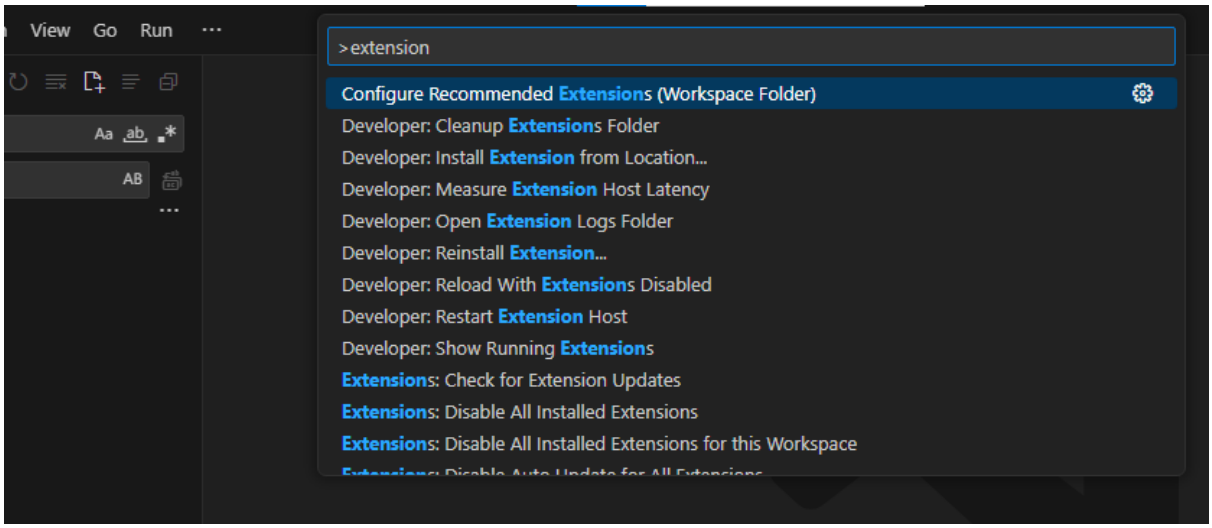
في الزاوية العلوية اليسرى من قائمة Terminal حدد (Close Panel) **X**

٣. في القائمة عرض View حدد لوحة الأوامر (**Command Palette**)

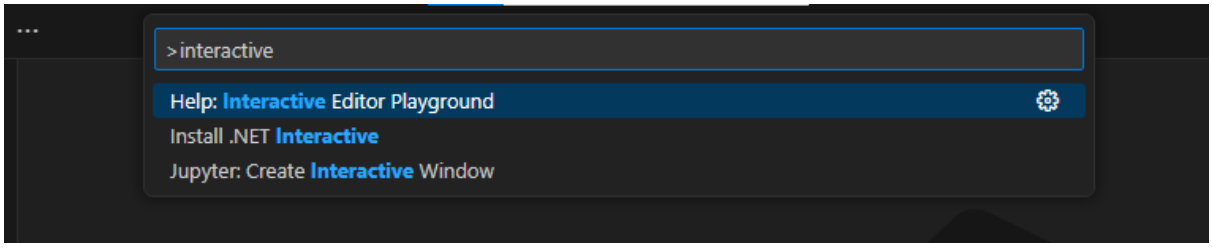


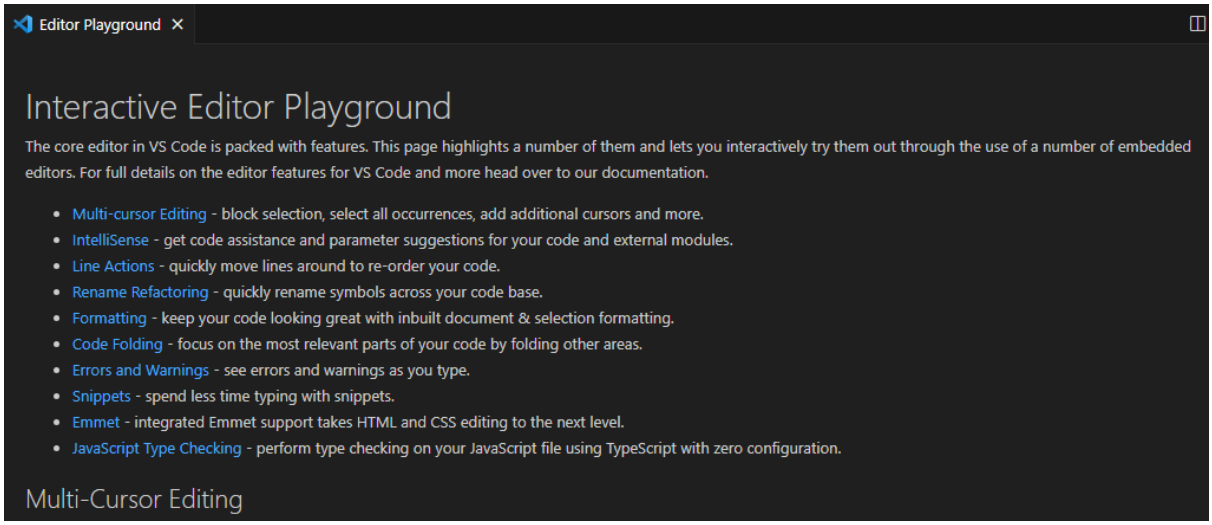
يمكن استخدام لوحة الأوامر (**Command Palette**) للبحث عن جميع أنواع الأوامر المفيدة وتشغيلها. ليس لديك الوقت أو الحاجة الآن لفحصها بالتفصيل، ولكن من الجيد معرفة مكان العثور على لوحة الأوامر.

في واجهة لوحة الأوامر، اكتب (**extensions**) **الملحقات** لاحظ أن قائمة الأوامر تتم تصفيتها استناداً إلى إدخالك.

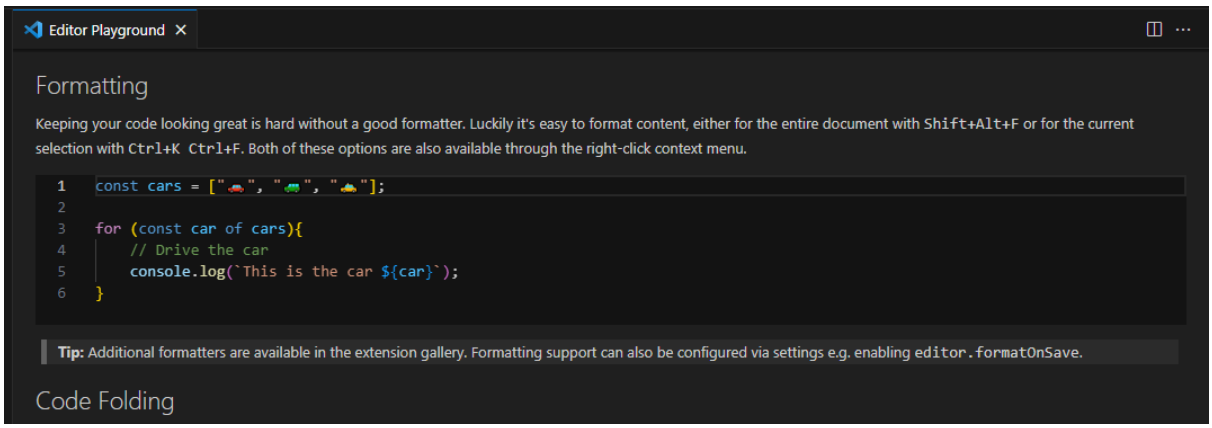


من قائمة خيارات الأوامر، حدد التعليمات: **Interactive Editor Playground** (ساحة التحرير التفاعلية) يتضمن مستند Editor Playground الذي يتم فتحه داخل المحرر قائمة بأنشطة تفاعلية





من قائمة الأنشطة التفاعلية، حدد **Formatting (تنسيق)** خذ دقيقة لقراءة خيارات التنسيق.



ستستخدم أوامر تنسيق التعليمات البرمجية أثناء معرفة المزيد حول برمجة C#

أغلق مستند Editor Playground

اكتملت جولتك في واجهة مستخدم Visual Studio Code بمجرد البدء في الترميز، تستمر مواد التدريب والدروس في الإشارة إلى الطرق التي يمكن أن يساعد بها Visual Studio Code في تعزيز إنتاجيتك.

٦ تكوين ملحقات Visual Studio Code

يمكنك توسيع قدرات Visual Studio Code باستخدام الملحقات، قام مطوري مجتمع Visual Studio Code ببناء مئات الملحقات المفيدة المتوفرة على Visual Studio Code Marketplace

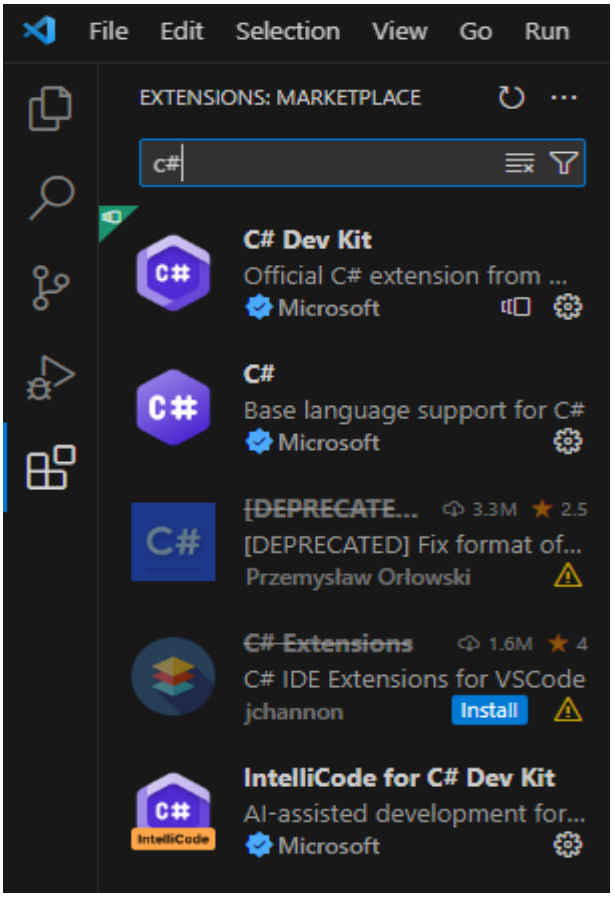
تكوين ملحقات Visual Studio Code

الآن بعد أن أصبحت على دراية بواجهة مستخدم Visual Studio Code فإن خطوتك التالية هي تكوين الملحقات الضرورية التي تسهل عليك مهمة ترميز أو كتابة التعليمات البرمجية C#

١. تأكد من فتح Visual Studio Code

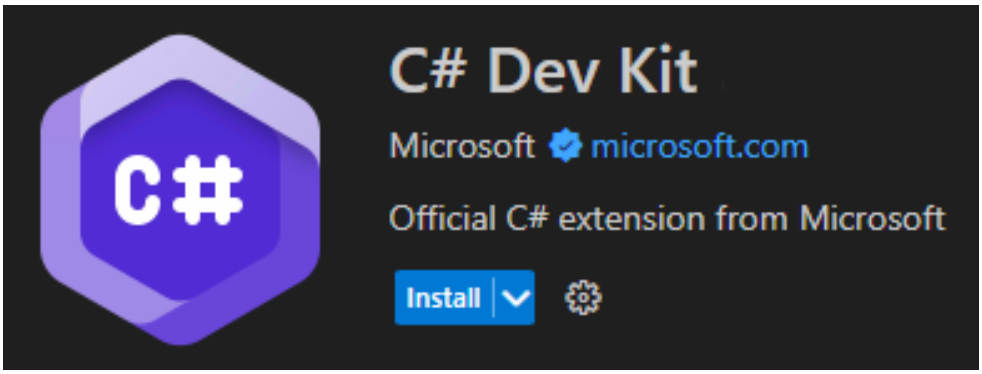
٢. في شريط النشاط حدد Extensions

٣. في مربع النص Search Extensions in Marketplace أدخل C#



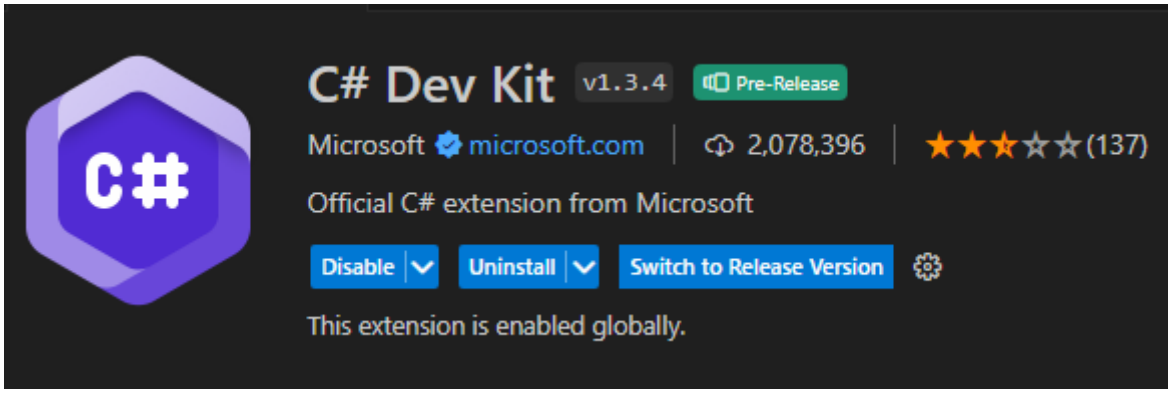
يؤدي إدخال "C#" إلى تصفية قائمة الملحقات التي لها علاقة فقط بلغة C#

٤. في قائمة الملحقات المتوفرة التي تمت تصفيتها، حدد الملحق المسمى: **C# Dev Kit** ملحق C# الرسمي من Microsoft



هناك الكثير من المعلومات حول ملحق C# الذي يمكنك معرفتها، في الوقت الحالي، الشيء المهم الذي يجب معرفته هو أن هذا الملحق يساعدك على تطوير التعليمات البرمجية C# وتحريرها وتصحيحها في Visual Studio Code يمكنك العودة لاحقاً للقراءة حول الملحق إذا كنت تريد ذلك.

٥. لتثبيت الملحق حدد **تثبيت**.
يتغير الزر تثبيت إلى مثبت، يتم تثبيت هذا الملحق سريعاً.



ملاحظة

ضع في اعتبارك أنه من الأفضل الانتظار قبل تثبيت أي ملحق متوفر على Marketplace تؤدي العديد من العناصر وظائف مماثلة، وبعض العناصر متخصصة، تركز على جانب واحد في اللغة البرمجية، ستحتاج إلى اختيار الملحقات التي توفر أكبر فائدة للمهام التي تحتاج إلى إكمالها، بدلاً من تثبيت أي ملحق يبدو مثيراً للاهتمام.

٦. أغلق صفحة C# Dev Kit في المحرر.
٧. في طريقة العرض EXTENSIONS قم بإلغاء تحديد النص #C من مربع البحث.

سيتم تحديث طريقة عرض EXTENSIONS لإظهار كافة الملحقات المثبتة في أعلى القائمة، يتم سرد الملحقات الشائعة، والموصي بها أسفل الملحقات المثبتة.

٨. لاحظ أن تثبيت حزمة تطوير #C يثبت الملحقات التالية تلقائياً:

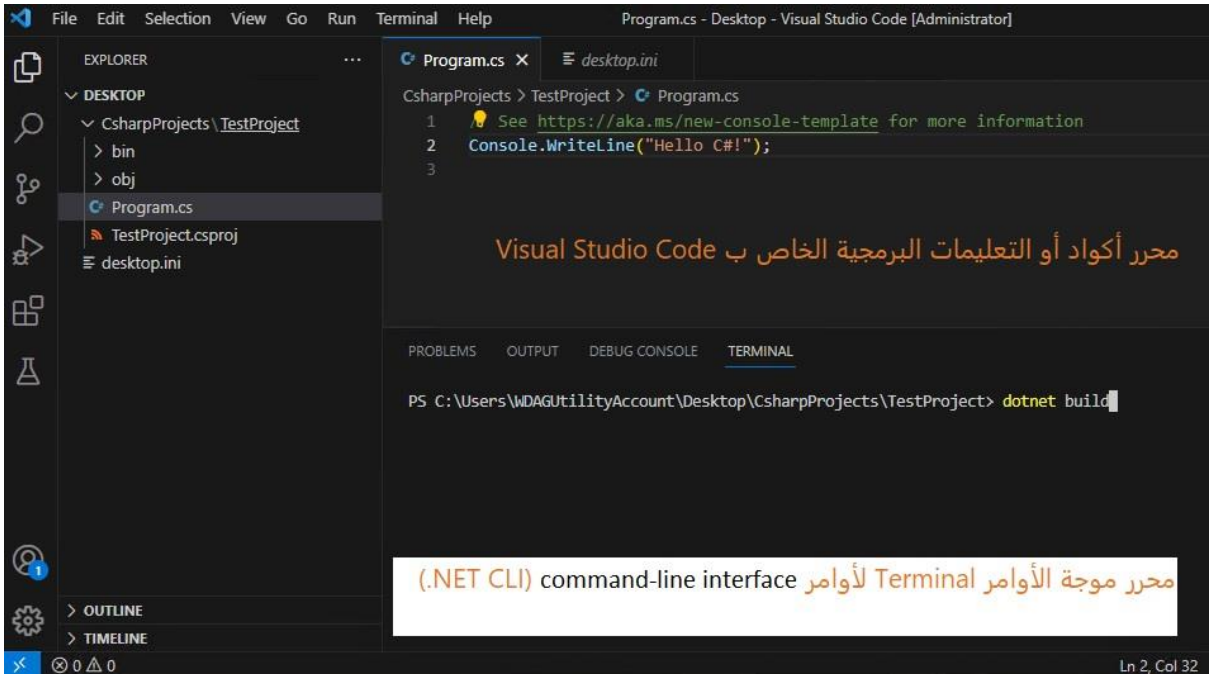
- C# Dev Kit ملحق #C الرسمي منMicrosoft
- #C داعم اللغة الأساسية
- IntelliCode ل #C التطوير بمساعدة الذكاء الاصطناعي ل C# Dev Kit
- أداة تثبيت NET. لمؤلفي الملحقات

هذه الملحقات الأربعة هي الملحقات الوحيدة التي تحتاج إليها الآن.

في الدرس التالي، يمكنك إنشاء مشروع برمجي #C في Visual Studio Code

٧ تمرين ـ إنشاء تطبيقك وبنائه وتشغيله

تتضمن مجموعة تطوير برامج .NET (SDK). واجهة أسطر الأوامر (command-line interface) تعرف باختصار CLI التي يمكن الوصول إليها من integrated Terminal (المحطة الطرفية المتكاملة) في Visual Studio Code أثناء هذا التدريب يمكنك استخدام (.NET CLI commands) لإنشاء new console applications (تطبيقات وحدة تحكم جديدة) وكتابة التعليمات البرمجية لمشروعك، واختبار التطبيقات الخاصة بك.



على سبيل المثال، سيقوم أمر .NET CLI التالي بإنشاء new console تطبيق لوحة وحدة تحكم جديدة في موقع المجلد المحدد:

```
dotnet new console -o ./CsharpProjects/TestProject
```

تتكون بنية أمر CLI من الأجزاء الثلاثة التالية:

- برنامج التشغيل: dotnet كما في المثال.
- الأمر: new console كما في المثال.

- مسارات الأمر: -o ./CsharpProjects/TestProject كما في المثال.

ملاحظة

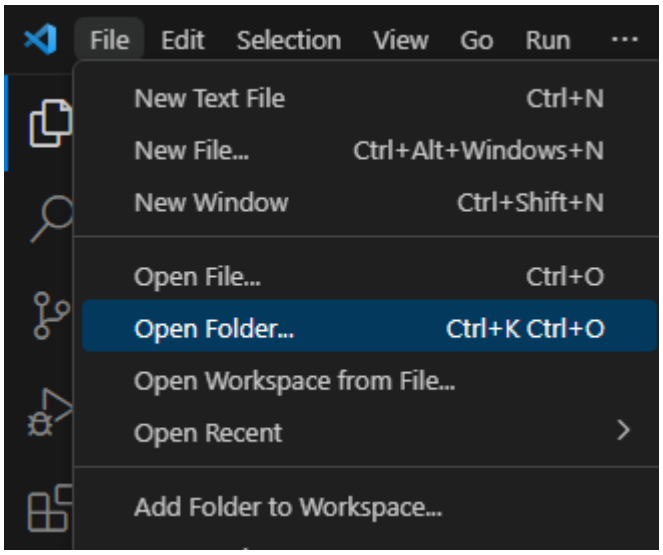
مسارات الأوامر هي معلمات اختيارية يمكن الاستغناء عنها، تستخدم لتوفير معلومات إضافية، إذا تم تشغيل الأمر السابق دون تحديد موقع المجلد الاختياري، على سبيل المثال: dotnet new console فقط، في هذه الحالة، سيتم إنشاء new console تطبيق وحدة التحكم الجديد في موقع مجلد البرنامج الحالي.

في هذا التمرين، يمكنك استخدام Visual Studio Code لإنشاء مجلد مشروع جديد، وإنشاء new console تطبيق وحدة تحكم جديد باستخدام أمر CLI وتخصيص التطبيق في محررVisual Studio Code ثم إنشاء تطبيقك واختباره.

إنشاء تطبيق new console وحدة تحكم C# في مجلد محدد

للبدء، يمكنك إنشاء تطبيق وحدة تحكم في موقع مجلد يسهل العثور عليه، لإعادة استخدامه.

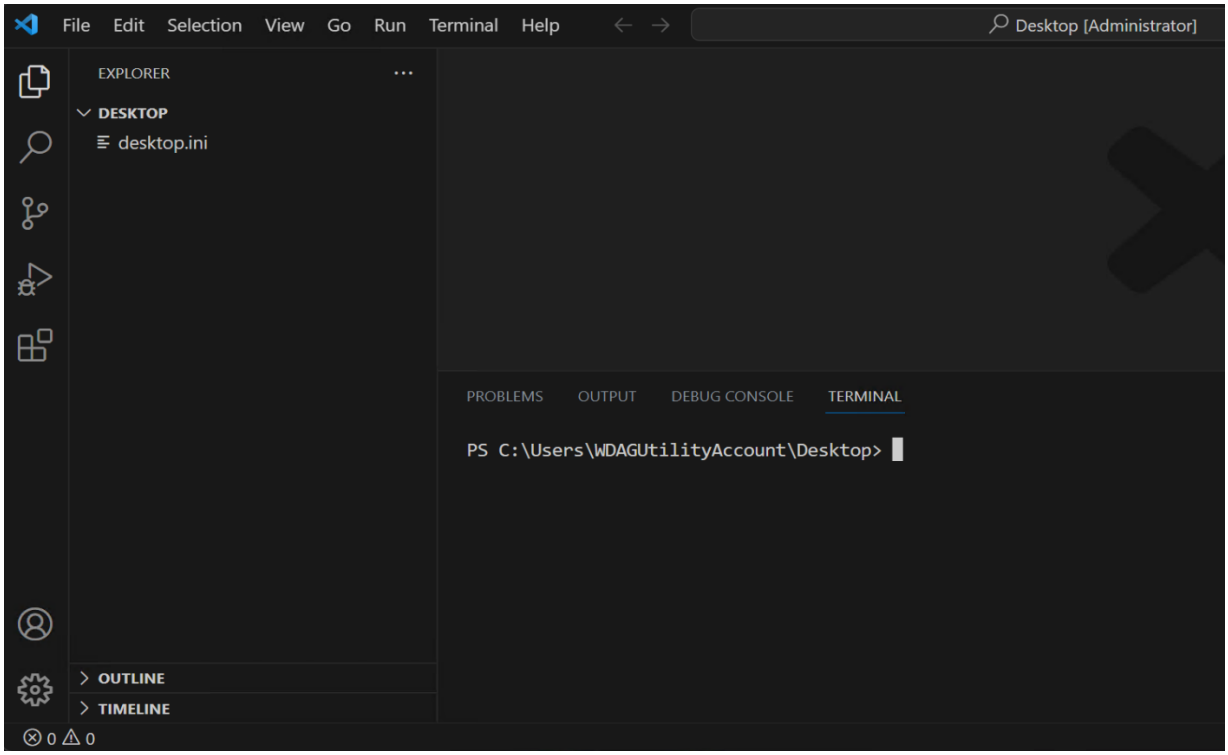
١. يمكنك استخدام القائمة Windows لتحديد موقع Visual Studio Code وفتحه.
٢. في قائمة ملف (**File**) حدد فتح مجلد **Open Folder**
سيتم عرض قائمة فتح مجلد (**Open Folder**) يمكنك استخدامه لإنشاء مجلد جديد لمشروع C# الخاص بك.



إذا كان لديك مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام موقع هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره، يمكنك استخدام قائمة فتح مجلد للانتقال إلى موقع المجلد المطلوب.

٣. في قائمة فتح مجلد، انتقل إلى مجلد سطح مكتب Windows
٤. في قائمة فتح مجلد، حدد **Select Folder** تحديد مجلد.
إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد نعم.

٥. في قائمة Terminal حدد New Terminal



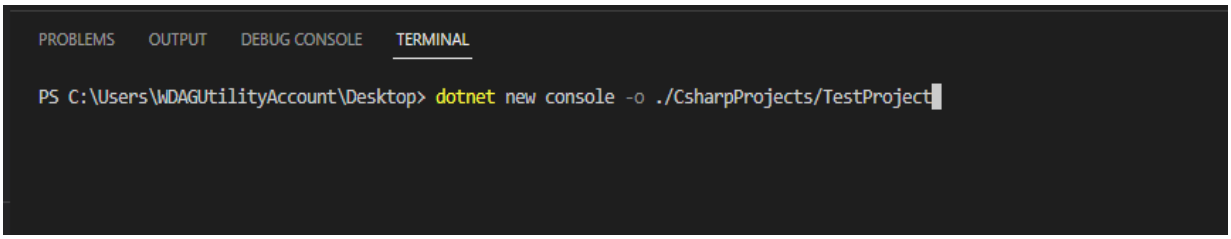
يجب أن تتأكد أن موجه الأوامر في لوحة Terminal يعرض المسار للمجلد الحالي، على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. في موجه الأوامر Terminal لإنشاء new console (تطبيق وحدة تحكم جديد) في مجلد محدد، أدخل الأمر التالي:

```
dotnet new console -o ./CsharpProjects/TestProject
```

لاحظ ان كلمات الأمر `dotnet new console` مكتوبة بحروف صغيرة.



يستخدم The CLI command قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد، ينشئ الأمر مجلدات CsharpProject، TestProject نيابة عنك، ويستخدم TestProject كامتداد لاسم لملفك. csproj

٧. في قائمة المستعرض أو استكشاف EXPLORER قم بتوسيع المجلد CsharpProjects

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# يسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

يستخدم The CLI command اسم المجلد عند إنشاء ملفات المشروع، التي تحتوي التعليمات البرمجية C#

٨. في قائمة EXPLORER حدد ملف **Program.cs**

كما تري، new console (تطبيق وحدة التحكم) التعليمات البرمجية الافتراضية تظهر الرسالة الأيقونة "مرحبًا بالعالم!"

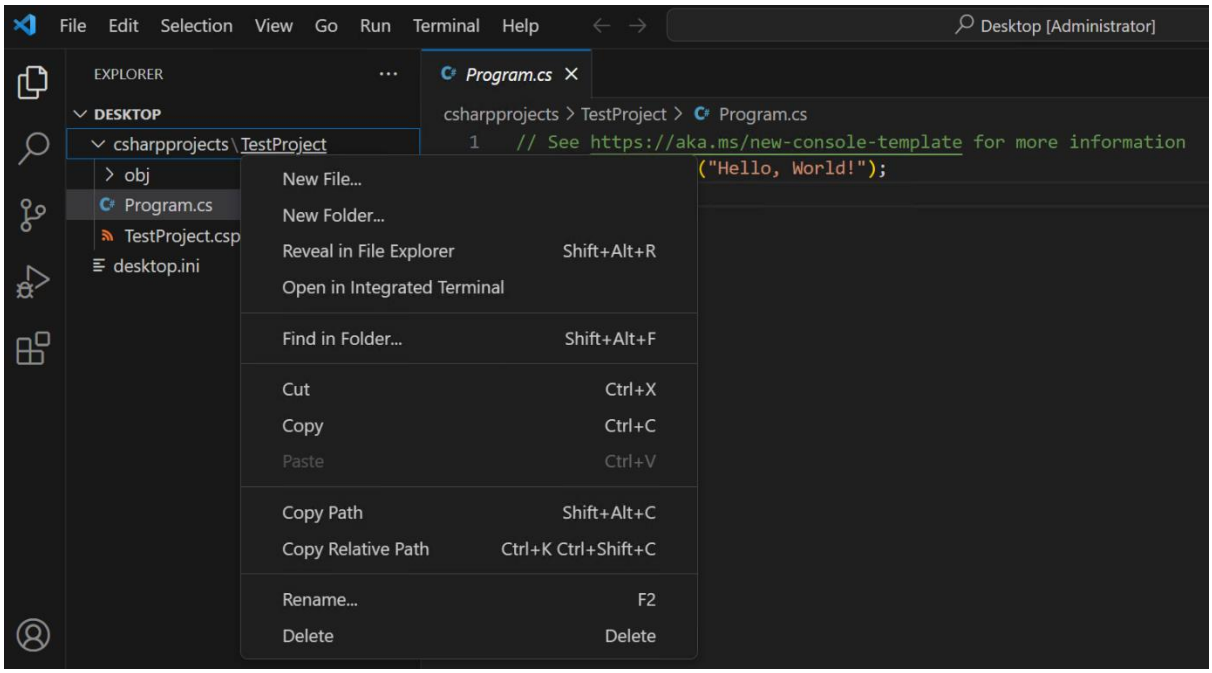
يستخدم Console.WriteLine() لعرض رسالة " Hello, World" في نافذة الإخراج.

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

تحديث، وبناء، وتشغيل تطبيقك

في هذه المهمة، يمكنك استخدام طريقة عرض EXPLORER لفتح مجلد مشروع التعليمات البرمجية وتخصيص رسالة "Hello" ثم تشغيل تطبيقك.

١. في قائمة الشريط الجانبي "EXPLORER" انقر بزر الماوس الأيمن **فوق المجلد (TestProject) ثم حدد Open in integrated Terminal**



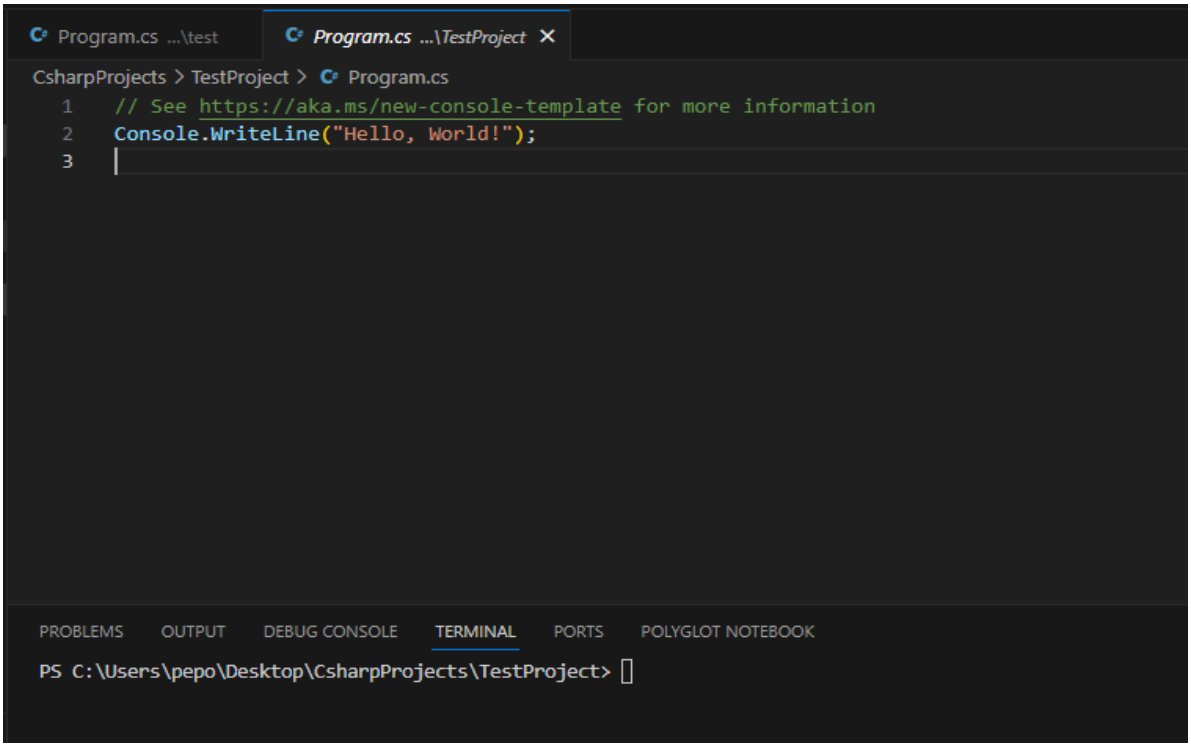
هام

يظهر موجه الأوامر Terminal في المساحة المخصصة لكتابة الأوامر موقع مجلد المشروع الحالي. قبل تشغيل أمر (build, run) يجب التأكد من أن Terminal مفتوح في مسار مجلد المشروع الحالي، كي لا يحدث خطأ.

لا تنس هذا التنبيه

٢. تحقق من أن موجه الأوامر في Terminal يعرض مسار المجلد التالي:

```
C:\Users\someuser\Desktop\CsharpProjects\TestProject>
```



٣. في محرر (Visual Studio Code) قم بتحديث الكود أو التعليمة البرمجية Console.WriteLine() كما يلي:

```
Console.WriteLine("Hello C#!");
```

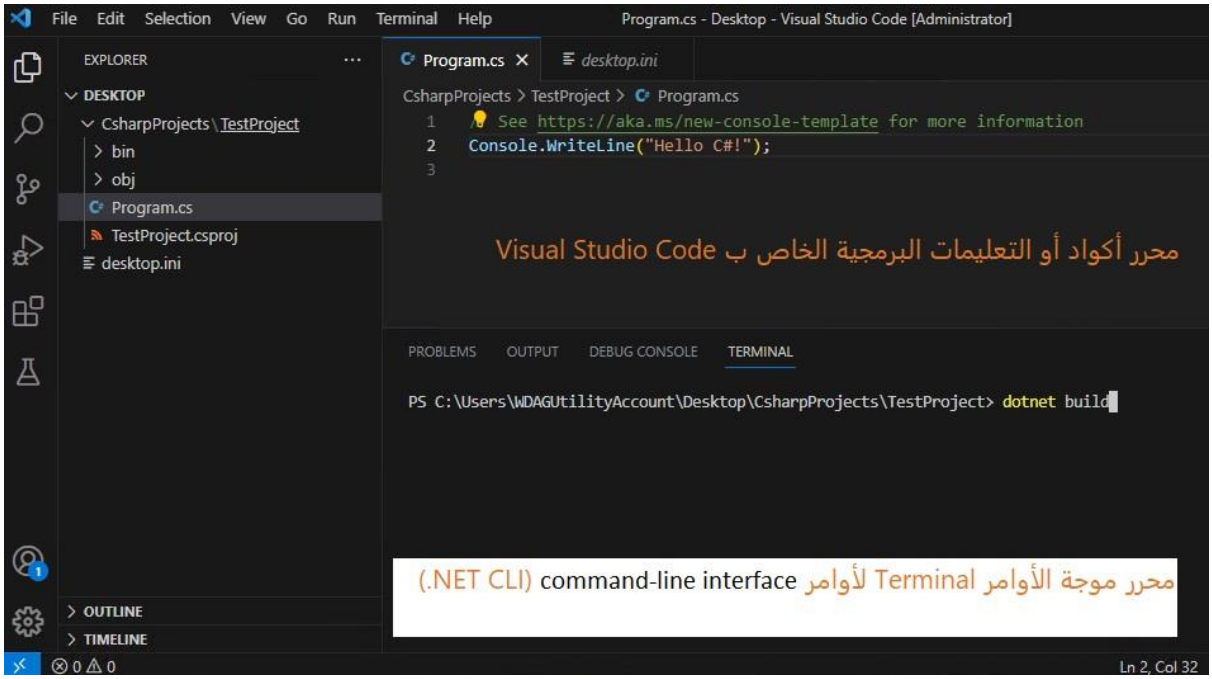
في المرة الأولى التي تقوم فيها بتحرير ملف cs. قد يطالبك Visual Studio Code بإضافة الأصول المفقودة لإنشاء تطبيقك وتصحيحه، إذا رأيت المطالبة، يمكنك تحديد **نعم**.

٤. في القائمة **ملف**، حدد **حفظ**.

يجب دائما حفظ التعليمات البرمجية إلى ملف تطبيقك بعد كل تغيير أو تعديل. لن يتعرف المحول البرمجي على تغييرات التعليمات البرمجية التي أجريتها في المحرر، قبل أن يتم حفظها.

٥. لتحويل بنية تطبيقك برمجياً، أدخل الأمر التالي في موجه الأوامر Terminal

```
dotnet build
```



يبني الأمر dotnet build المشروع وتبعياته في مجموعة من الثنائيات (اللغة التي يفهمها جهاز الكمبيوتر)، تبادر الثنائيات بتضمين التعليمات البرمجية للمشروع في ملفات اللغة الوسيطة Intermediate Language تعرف اختصاراً (IL) مع ملحقات .dll (dll ملفات يتم حفظ مجموعة التعليمات البرمجية داخلها) اعتماداً على نوع المشروع وإعداداته، قد يتم أيضاً تضمين ملفات أخرى، يمكنك العثور على ملف TestProject.dll من قائمة EXPLORER تجده محفوظ في موقع مجلد تطبيقك، الاسم مشابه للمسار التالي:

```
C:\Users\someuser\Desktop\CsharpProjects\TestProject\bin\Debug\net7.0\
```

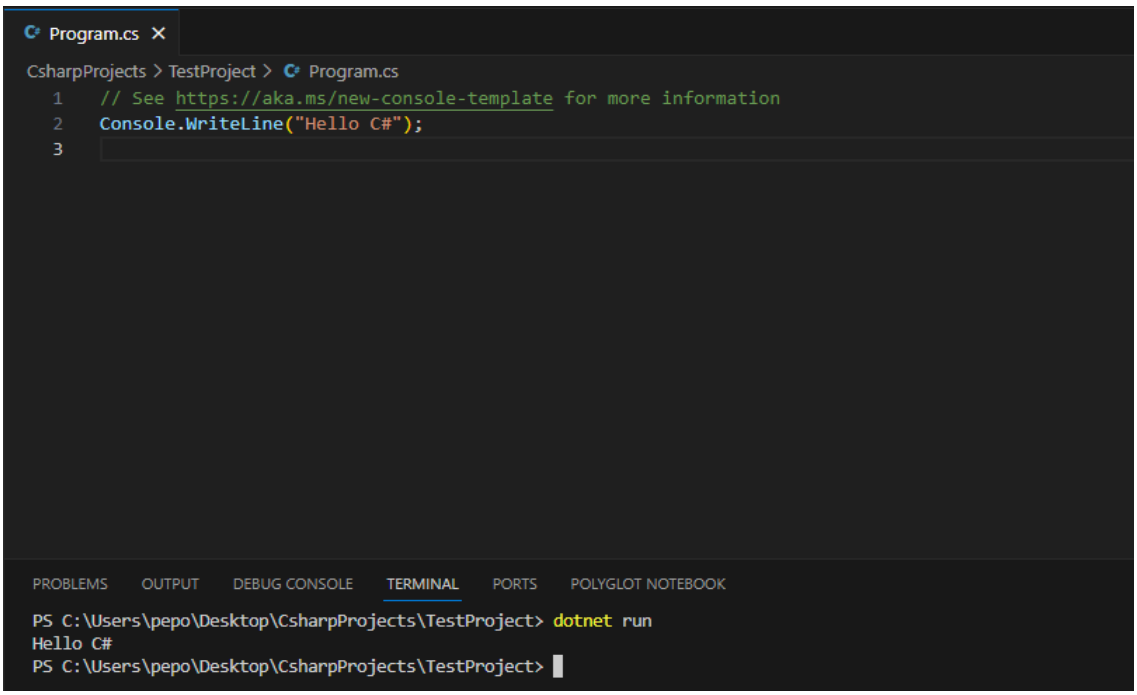
لاحظ أن أمر dotnet build لا يشغل تطبيقك، تحتاج إلى أمر آخر لتشغيله.

٦. لتشغيل تطبيقك، أدخل الأمر التالي في موجه الأوامر Terminal

```
dotnet run
```

يقوم أمر `dotnet run` بتشغيل التعليمات البرمجية دون الحاجة لأي أوامر تحويل برمجي أو تشغيل صريحة، يوفر خياراً مناسباً لتشغيل تطبيقك من أمر واحد، مفيد للتطوير التكراري السريع من سطر الأوامر، يعتمد الأمر على أمر إنشاء dotnet لإنشاء التعليمات البرمجية.

٧. لاحظ أنه يتم عرض **Hello C#** في قائمة أوامر Terminal على السطر أسفل أمر dotnet run



إذا رأيت "Hello, World!" معروضة أو حدث خطأ، تأكد من حفظ تغييرات التعليمات البرمجية.

تهانينا، لقد أكملت إعداد Visual Studio Code وإنشاء وتشغيل سطر صغير من التعليمات البرمجية!

٨ اختبر معلوماتك

اختبر معلوماتك

١. ما هي منطقة واجهة مستخدم Visual Studio Code المستخدمة لكتابة التعليمات البرمجية؟

- Explorer
- الملحقات
- المحرر

٢. أي من الأدوات التالية مطلوبة لتشغيل أمر (command line interface) واجهة أسطر الأوامر مثل dotnet run؟

- تعليمة Visual Studio البرمجية
- C# Dev Kit
- The .NET SDK

راجع إجابتك

المحرر

صحيح المحرر هو المساحة الرئيسية حيث يكتب المطورون التعليمات البرمجية

The .NET SDK

صحيح .NET SDK. مطلوبه لتشغيل أمر (command line interface) أوامر واجهة أسطر الأوامر مثل dotnet run

٩ الملخص

هدفك كان تثبيت وتكوين بيئة تطوير تفي بمتطلبات الترميز الخاصة بك للغة C#

لقد قمت بتثبيت Visual Studio Code وقمت بجولة سريعة في واجهة المستخدم، تمكنت من تكوين الملحقات التي تجعل Visual Studio Code أداة أكثر قوة لتطوير التعليمات البرمجية بلغة C# قمت أيضاً بتثبيت .NET SDK لتوفير دعم المحول البرمجي `compiler` ووقت التشغيل `runtime` ودعم `command line interface commands` أوامر واجهة أسطر الأوامر.NET.

أنت الآن مستعد تماماً لإنشاء تطبيقات C# رائعة في بيئة ترميز يستخدمها المطورون المحترفون في جميع أنحاء العالم.

الوحدة الثانية

استدعاء الأساليب من مكتبة فئات .NET Class. باستخدام لغة C#

استخدم الوظائف في مكتبة فئات .NET Class. عن طريق استدعاء الأساليب التي ترجع القيم، وتقبل معلمات الإدخال والمزيد.

الأهداف التعليمية

- كتابة التعليمات البرمجية التي تستدعي الأساليب عديمة الحالة في مكتبة فئات .NET.
- إنشاء نسخة جديدة لفئات مكتبة .NET. لاستدعاء الأساليب التي لها حالة
- استخدم IntelliSense لمعرفة المزيد حول الأسلوب، وإصداراته ذات التحميل الزائد، ونوع بيانات القيمة المرجعة، وأنواع بيانات معلمات الإدخال الخاصة به (نوع قيم البيانات المدخلة التي تمرر داخل الأسلوب)
- استخدم وثائق Microsoft Learn للبحث عما يفعله الأسلوب، وإصداراته ذات التحميل الزائد، ونوع القيمة المرجعة، ومعلمات الإدخال الخاصة به، وما تمثله كل معلمة

محتويات الوحدة:

١ مقدمة

٢ بدء استخدام مكتبات NET.

٣ استدعاء أساليب فئة NET.

٤ إرجاع القيم ومعلمات الإدخال للأساليب

٥ إكمال نشاط تحدي لاكتشاف وتنفيذ استدعاء أسلوب

٦ راجع الحل لاكتشاف نشاط تحدي استدعاء الأسلوب وتنفيذه

٧ اختبر معلوماتك

٨ الملخص

١ المقدمة

تم دعم لغة البرمجة C# بمكتبة كبيرة تحوي الكثير من الوظائف والخصائص التي تمكنك من إنشاء التطبيقات، والوصول إلى البيانات في الملفات أو على الإنترنت، وتنفيذ العمليات الرياضية المتقدمة، وأكثر من ذلك بكثير. يعد فهم كيفية التنقل في مكتبة الوظائف library of functionality هذه مهارة مهمة، تساعدك على إنشاء تطبيقات غنية بالميزات بشكل أسرع.

لنفترض أنك بحاجة إلى إنشاء تقرير يومي، يستند إلى آلاف من ملفات البيانات، توجد الكثير من الفئات وأساليبها لدعم العمليات الرياضية والحسابية لتوفير الوقت، ويمكن استخدام التعليمات البرمجية لإخراج عينة عشوائية من ملفات البيانات لحساب نتيجة تقريبية، دون تحليل جميع البيانات.

كيف يمكنك إنشاء رقم عشوائي؟ هل هذا مدعوم من C# إذا كان الأمر كذلك، كيف يمكنك كتابة التعليمات البرمجية لتعيين نطاق القيمة لرقم عشوائي، ثم إنشاء النتيجة واستردادها؟

في هذه الوحدة، يمكنك التعرف على بعض فئات المكتبة `class libraries` وكتابة التعليمات البرمجية التي تستدعي الأساليب `methods` من فئات مكتبة .NET. كما يمكنك التعرف على خصائص الأساليب، ولماذا يتم تصميم `methods` أساليب فئات مكتبة .NET. وتنفيذها بشكل مختلف عبر مكتبة .NET. والأهم من ذلك، يمكنك كتابة التعليمات البرمجية التي تستخدم أساليب مكتبة .NET. لتحقيق مهمة برمجة معينة، وأخيراً، ستستخدم الأدوات المضمنة، والوثائق عبر الإنترنت، لمساعدتك في جمع المعلومات حول الأساليب التي تريد العمل بها.

في نهاية هذه الوحدة، ستعرف كيفية العثور على الفئات `class` والأساليب `methods` في مكتبة .NET. وكيفية استخدامها لتنفيذ مهام البرمجة الشائعة.

هام

تتضمن هذه الوحدة التدريب على خصائص الترميز التي تتطلب Visual Studio Code ستحتاج إلى تثبيت Visual Studio Code وتكوين ملحقاته.

٢ بدء استخدام مكتبات .NET.

هناك المزيد لبناء تطبيقات C# بربط العديد من أسطر التعليمات البرمجية معاً، ستحتاج أيضاً إلى وقت تشغيل **.NET Runtime**. هو أداة أو تطبيق يستضيف ويدير التعليمات البرمجية، أثناء تنفيذه على كمبيوتر المستخدم النهائي، ستعتمد أيضاً على مكتبة **.NET Class Library**. وهي عبارة عن مجموعة تعليمات برمجية جاهزة، مكتوبة مسبقاً من أسطر برمجية، تماماً مثل التي باستطاعتك استخدامها في تطبيقاتك، توفر كل الموارد البرمجية التي تحتاجها، ولولا هذه المكتبة لاحتجت لكتابة التعليمات التي تحويها الأساليب والفئات بنفسك، تشرح هذه الوحدة ماهية مكتبة .NET وكيفية تكميلها للغة البرمجة C#

ما المقصود بمكتبة فئات .NET Class Library.

عندما تحتاج إلى العثور على كتاب، فإن المكتبة العامة هي مكان جيد للبحث، بعد كل شيء، المكتبات تحتوي على آلاف وآلاف الكتب، يتم تنظيمها في أقسام تساعدك على العثور على ما تبحث عنه، عندما تحتاج إلى تنفيذ مهمة برمجة، تعد مكتبة فئات .NET مكاناً جيداً للبحث، لأنها مجموعة منظمة من موارد البرمجة.

تعد مكتبة فئات .NET. مجموعة من آلاف الفئات `Class` التي تحتوي داخلها على عشرات الآلاف من الأساليب `methods` على سبيل المثال، تتضمن مكتبة فئات .NET. على Console فئة للمطورين الذين يعملون على تطبيقات وحدة التحكم Console تتضمن الفئة أساليب لعمليات الإدخال والإخراج مثل `ReadLine()` `Read()` `WriteLine()` `Write()` وغيرها الكثير، على سبيل المثال، أنت على دراية بالتعليمات البرمجية التالية:

```
Console.WriteLine("Hello, World!");
```

يمكنك التفكير في فئة كحاوية للأساليب، مثل أقسام المكتبة العامة، عادةً ما يحتفظ المطورون بالأساليب المتشابهة وذات الصلة معاً، في فئة واحدة. كما رأيت في المثال السابق، يتم تجميع أي أساليب يمكنها إرسال معلومات أو تلقيها من نافذة وحدة التحكم في فئة System.Console في مكتبة فئات .NET.

في الكثير من الحالات، تمكنك هذه الفئات والأساليب من إنشاء نوع تطبيقات مخصصة، على سبيل المثال، تمكنك إحدى المجموعات الفرعية الكبيرة من الفئات والأساليب من إنشاء تطبيقات **ويب تفاعلية**، هناك أيضاً العديد من الفئات التي تمكنك من إنشاء تطبيقات **سطح المكتب والهواتف**، وتمكّنك مجموعة فرعية أخرى من الفئات والأساليب من الوصول إلى **قواعد البيانات**. هناك الكثير من الفئات في مكتبة NET. التي تدعم أنواعاً معينة من التطبيقات.

هناك فئات أخرى مع أساليب توفر الدعم بطريقة عامة، أكثر شمولية، بعبارة أخرى، تمتد أدواتها المساعدة على مجموعة واسعة من الأنظمة الأساسية للأجهزة وأطر عمل التطبيقات ومجالات التكنولوجيا، على سبيل المثال، إذا كنت تريد قراءة معلومات الملف أو كتابتها، أو تنفيذ العمليات الرياضية أو المثلثات أو حساب التفاضل والتكامل، فهناك فئات وأساليب للأغراض العامة، يمكنك استخدامها في تعليماتك البرمجية.

لا يهم ما إذا كنت تقوم بإنشاء تطبيقات للويب أو سطح المكتب أو الهواتف أو السحابة، فهناك فئات وأساليب للأغراض العامة جاهزة للمساعدة.

وكما يمكنك أن تتخيل، فإن امتلاك مكتبة ضخمة من الوظائف المتاحة لتطبيقاتك، يعد توفيراً كبيراً للوقت والجهد، بالنسبة لك كمطور برامج، يتم إنشاء وتطوير الفئات والأساليب في مكتبة NET. بواسطة Microsoft وتوفرها للاستخدام في تطبيقاتك.

أنواع البيانات جزء من مكتبة NET.

يتم توفير أنواع بيانات C# مثل string and int وغيرها، يتم إتاحتها بالفعل من خلال الفئات في مكتبة NET. تخفي لغة C# الاتصال بين أنواع البيانات والفئات لتبسيط عملك، ومع ذلك، يتم تنفيذ أنواع البيانات تماماً مثل أي فئة أخرى في مكتبة NET. خلف الكواليس، يزود هذا الاتصال مهامك اليومية التطويرية بأساليب مدمجة يمكن أن تكون مفيدة للغاية.

كيفية العثور على ما تحتاجه في .NET Class Library.

مع وجود العديد من الفئات والأساليب، كيف يمكنك العثور على ما تحتاجه لتطبيقك؟

أولا وقبل كل شيء، تذكر أن العثور على كل فئة وأسلوب في مكتبة فئات NET. يشبه العثور على كل الكتب في مكتبة عامة كبيرة! **أنت لا تحتاج إلى كل كتاب في المكتبة**. ولن تستخدم كل فئة وأسلوب في مكتبة فئات NET.

استناداً إلى أنواع المشاريع التي تعمل عليها، ستصبح أكثر دراية ببعض أجزاء مكتبة فئات NET. وأقل دراية بالبعض الآخر، هذا طبيعي فلا تقلق، مرة أخرى، يشبه قضاء بعض الوقت في قسم من المكتبة العامة، مع مرور الوقت تصبح على دراية بما هو متاح، لا أحد يعرف كل مكتبة فئات NET. **ولا حتى الأشخاص الذين يعملون في Microsoft**

ثانياً، سوف تدفعك الضرورة إلى ما تحتاجه، يذهب معظم الأشخاص إلى المكتبة عندما يحتاجون إلى العثور على كتاب، وليس لمعرفة عدد الكتب ومصنفاتها المختلفة التي يمكنهم العثور عليها، لا تحتاج إلى الكتب والأبحاث العلمية دون سبب، عندما تواجه مشكلة في معرفة مهمة برمجية، يمكنك استخدام محرك البحث المفضل لديك، للعثور على منشورات المواقع أو المقالات أو المنتديات التي تعامل فيها مطورون آخرون مع مشكلات مماثلة، يمكن أن تعطيك المصادر المتاحة على الانترنت، معلومات وافية حول فئات NET. والأساليب التي قد ترغب في استخدامها، ستجد أيضاً نماذج تعليمات برمجية يمكنك تجربتها.

ثالثا، توفر Microsoft مرجعا للغة C# عبر الإنترنت ودليل برمجة، يمكنك الاطلاع عليه، من المحتمل أن تقضي بعض الوقت في قراءة وثائق Microsoft عندما تحتاج إلى فهم الأساليب التي تحتاجها، وكيفية عملها وقيودها، وثائق Microsoft مصدر معلومات موثوقة، لأن البرنامج وملحقاته والمكتبة صناعتهم، ويعمل فريق وثائق Microsoft بشكل مستمر مع مطوري برامج مكتبة NET. لضمان دقتها.

وأخيراً، عندما تبدأ في تجربة مشاريع البرمجة الصغيرة، سيتعمق فهمك لكيفية عمل الفئات والأساليب.

يتبع جميع مطوري البرامج طريق مماثل، عند الدخول في مجال غير مألوف، تعد عملية الاكتشاف ممتعة ولكن تمثل تحدياً، فتحلي بالصبر والمثابرة.

خلاصة

- توفر لك مكتبة NET Class Library. ثروة من الوظائف التي يمكنك استخدامها عن طريق الإشارة فقط إلى الفئات والأساليب التي تحتاجها.
- حتى أنواع البيانات التي تستخدمها هي جزء من مكتبة فئات NET. توفر C# مجرد اسم مستعار لأنواع البيانات هذه.

٣ استدعاء أساليب فئة .NET.

سواء أدركت ذلك أم لا، كنت تتصل بأساليب C# منذ أول تطبيق " Hello World" استخدمت الأسلوب WriteLine() والفئة Console لعرض رسالة أهلا بالعالم "Hello, World"

مع ذلك، لا يتم تنفيذ جميع الفئات والأساليب بنفس الطريقة السابقة، تغطي هذه الوحدة بعض المفاهيم الأكثر شيوعاً التي ستحتاج إلى معرفتها عند استخدام أساليب من مكتبة فئات .NET. والأهم من ذلك سوف تتعلم كيفية العثور على المعلومات، واستخدامها لفهم المزيد حول كل أسلوب بشكل أفضل.

كيفية استدعاء الأساليب في مكتبة فئات .NET.

من تجربتك السابقة مع الأسلوب `Console.WriteLine()` يجب أن تكون بالفعل على دراية بالأساسيات التالية:

- البدء بكتابة اسم الفئة، في هذه الحالة، يكون اسم الفئة هو `Console`
- أضف عامل وصول الأسلوب الرمز '.'
- أضافة اسم الأسلوب، في هذه الحالة، يكون اسم الأسلوب هو `WriteLine`
- أضافة عامل استدعاء الأسلوب، وهو مجموعة من الأقواس **()**
- وأخيرا، حدد الوسائط (القيم) إن وجدت، التي يتم تمريرها إلى الأسلوب، بين أقواس عامل تشغيل الأسلوب، في هذه الحالة، يمكنك تحديد النص الذي تريد أن يظهره الأسلوب `Console.WriteLine()` إلى وحدة التحكم على سبيل المثال "أهلا بالعالم"

واختيارياً، بناءً على كيفية تصميم المطورين، وتنفيذهم للأسلوب المحدد، قد تحتاج أيضاً إلى ما يلي:

- تمرير قيم إضافية كمعلمات إدخال.

- قبول قيمة إرجاع.

في الدرس التالي، سوف تفحص كيفية تمرير قيم الإدخال إلى أسلوب، وكيف يمكن استخدام أسلوب لإرجاع قيمة على نمط الاستدعاء.

بينما يمكن استدعاء بعض الأساليب بنفس الطريقة التي قمت باستدعائها مثل Console.WriteLine() هناك أساليب أخرى في مكتبة فئات .NET تتطلب نهجاً مختلفاً.

إعداد بيئة الترميز الخاصة بك

تتضمن هذه الوحدة أنشطة الترميز، ترشدك خلال عملية إنشاء نموذج التعليمات البرمجية وتشغيله، وتشجيعك على إكمال هذه الأنشطة باستخدام (Visual Studio Code) كبيئة تطوير، سوف يساعدك استخدام (Visual Studio Code) من خلال هذه الأنشطة على أن تصبح أكثر راحة في كتابة التعليمات البرمجية، وتشغيلها في بيئة تطوير يستخدمها المحترفون في جميع أنحاء العالم.

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو القائمة المماثلة لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف (File) حدد فتح مجلد (**Open Folder**)

٣. في قائمة (**مربع الحوار**) **فتح مجلد** (**Open Folder**)، انتقل إلى **مجلد سطح المكتب**

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام موقع هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع حوار فتح مجلد**، حدد **تحديد مجلد** (**Select Folder**)

إذا رأيت رسالة أمان تسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** حدد **New Terminal**

تأكد أن موجه الأوامر في قائمة محرر Terminal يعرض مسار مجلد المشروع الحالي.

إذا كنت تعمل على جهاز الكمبيوتر الخاص بك، وأكملت الدروس السابقة في سلسلة C# هذه، فربما تكون قد أنشأت بالفعل مجلد مشروع لاختبار التعليمات البرمجية، إذا كان الأمر كذلك، يمكنك تخطي الخطوة التالية، والتي تستخدم لإنشاء تطبيق (a console app) وحدة تحكم في مجلد TestProject

٦. في موجه الأوامر Terminal لإنشاء تطبيق (new console) وحدة تحكم جديد في مجلد محدد، اكتب:

dotnet new console -o ./CsharpProjects/TestProject ثم اضغط Enter

يستخدم .NET CLI command. البرنامج لإنشاء مشروع تطبيق `new console application` C# وحدة تحكم جديد، في موقع المجلد المحدد، ينشئ الأمر مجلدات CsharpProjects and TestProject folders نيابة عنك، ويستخدم لملف TestProject امتداد .csproj **يمكنك القول إن البرنامج يستخدم الملف كاسم للامتداد باعتبار الامتداد ملف C#**

٧. في قائم استكشاف EXPLORER اختار المجلد **CsharpProjects** وقم بتوسعته، يجب أن تشاهد مجلد TestProject وملفين، ملف C# يسمى Program.cs وملف مشروع يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في قائمة المحرر، حدد **Program.cs**

٩. أحذف أسطر التعليمات البرمجية الموجودة مسبقاً

أغلق قائمة محرر Terminal

ستستخدم مشروع new console وحدة تحكم هذا لإنشاء نماذج التعليمات البرمجية، وبنائها وتشغيلها أثناء هذه الوحدة.

استدعاء أنواع مختلفة من الأساليب من مكتبة فئات .NET.

١. في محرر التعليمات البرمجية لإنشاء نموذج مشروع برمجي ينفذ أساليب الفئات System.Random and System.Console classes أدخل التعليمات البرمجية التالية:

```
Random dice = new Random();
int roll = dice.Next(1,7);
Console.WriteLine(roll);
```

تحاكي هذه التعليمة البرمجية لفة حجر الزهر باستخدام الأسلوب `Random.Next()` لإنشاء رقم عشوائي، والأسلوب `Console.WriteLine()` لعرض القيمة.

ملاحظة

سوف تفحص هذه التعليمات البرمجية بالتفصيل لاحقاً في هذه الوحدة، فلا تقلق إذا اختلطت عليك الأمور

٢. في قائمة ملف **(File)** انقر فوق حفظ **Save**

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

عند استخدام Terminal لتشغيل أوامر .NET CLI. يكون موقع مجلد المشروع الحالي هو المكان الذي سيتم فيه تشغيل الأوامر، لذلك تأكد من أن مجلد المشروع الحالي يطابق مسار المجلد المعروض موجه الأوامر Terminal قبل إنشاء التعليمات البرمجية أو تشغيلها، كي لا يحدث خطأ.

٤. في محرر موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب dotnet run ثم اضغط على Enter

لاحظ أنه يتم عرض رقم من ١ إلى ٦ في مخرجات وحدة التحكم (عدد النقاط على حجر الزهر) إذا قمت بتشغيل التعليمات مرات كافية، فسوف ترى كل مرة رقم مختلف من ١ إلى ٦ وفي النهاية ستري كل الأرقام المعروضة.

٥. خذ دقيقة لفحص بناء الجملة المستخدم لمعرفة صلاحية الأسلوبين Next() ، WriteLine()

لاحظ أنك استخدم هذه المرة تقنيات مختلفة للوصول إلى الأساليب.

```
Random dice = new Random();
int roll = dice.Next(1,7);
Console.WriteLine(roll);
```

في سطر التعليمات البرمجية الثالث، يمكنك تضمين مرجع (reference) إلى الفئة Console واستدعاء الأسلوب Console.WriteLine() مباشرة ومع ذلك، يمكنك استخدام تقنية مختلفة لاستدعاء الأسلوب Random.Next()

سبب استخدام تقنيتين مختلفتين، هو أن بعض الأساليب ذات حالة (تقبل تمرير قيم لها على اختلاف أنواعها بين الأقواس) والبعض الآخر عديم الحالة (لا تتطلب إدخال أي قيم بين الأقواس) خطوتك التالية هي فحص الفرق بين الأساليب ذات الحالة والأساليب عديمة الحالة.

الأساليب ذات الحالة مقابل الأساليب عديمة الحالة

Stateful versus stateless methods

في مشاريع تطوير البرمجيات، يتم استخدام مصطلح الحالة لوصف حالة بيئة التنفيذ في لحظة محددة من الزمن، أثناء تنفيذ التعليمات البرمجية سطرًا تلو الآخر، يتم تخزين القيم في المتغيرات، في أي لحظة أثناء التنفيذ، تكون الحالة الحالية للتطبيق هي مجموعة كافة القيم المخزنة في الذاكرة.

لا تعتمد بعض الأساليب على الحالة الحالية للتطبيق لتعمل بشكل صحيح، بمعنى آخر، يتم تنفيذ **الأساليب عديمة الحالة stateless methods** بحيث يمكنها العمل دون الرجوع إلى أي قيم مخزنة بالفعل في الذاكرة أو تغييرها، وتُعرف الأساليب عديمة الحالة أيضاً **بالأساليب الثابتة static methods**

على سبيل المثال، لا يعتمد الأسلوب `Console.WriteLine()` على أي قيم مخزنة في الذاكرة، يؤدي وظيفته وينتهي دون التأثير على حالة التطبيق بأي شكل من الأشكال.

ومع ذلك، يجب أن يكون للأساليب الأخرى حق الوصول إلى حالة التطبيق لتعمل بشكل صحيح، بمعنى آخر، **يتم إنشاء أساليب ذات حالة stateful methods** بطريقة تعتمد على القيم المخزنة في الذاكرة بواسطة أسطر التعليمات البرمجية المكتوبة، التي تم تنفيذها بالفعل، أو تقوم بتعديل حالة التطبيق عن طريق تحديث القيم أو تخزين قيم جديدة في الذاكرة، وتُعرف أيضاً باسم **أساليب المثيل instance methods**

تتعقب الأساليب ذات الحالة (مثيل `instance`) حالتها في الحقول `fields` وهي متغيرات محددة داخل الفئة class ويحصل كل مثيل جديد من الفئة على نسخته الخاصة من تلك الحقول التي يتم تخزين الحالة فيها.

يمكن لفئة واحدة أن تدعم كلاً من الأساليب ذات الحالة، والأساليب عديمة الحالة، ومع ذلك، عندما تحتاج إلى استدعاء أساليب ذات حالة، يجب أولاً إنشاء مثيل للفئة class بحيث يمكن للأسلوب الوصول إلى الحالة.

إنشاء نسخة أو مثيل لفئة instance of a class

يسمى مثيل فئة كائن أو عنصراً object لإنشاء مثيل أو نسخة جديدة لفئة، يمكنك استخدام عامل التشغيل new اطلع على السطر التالي من التعليمات البرمجية الذي ينشئ نسخة أو مثيلاً جديداً للفئة Random لنطلق عليه dice مثلاً:

```
Random dice = new Random();
```

يقوم عامل التشغيل new بعدة أشياء مهمة:

- يطلب أولاً عنواناً في ذاكرة الكمبيوتر، بما يكفي لتخزين نسخة عنصر جديد بناءً على حجم الفئة Random
- يقوم بإنشاء نسخة الكائن الجديد، وتخزينه في عنوان الذاكرة.
- يقوم بإرجاع عنوان الذاكرة بحيث يمكن حفظه في المتغير dice

من تلك النقطة فصاعداً، عند الإشارة إلى المتغير dice يقوم وقت التشغيلNET Runtime. بإجراء بحث خلف الكواليس للوصول إلى عنوان النسخة، لإعطاء انطباع أنك تعمل مباشرةً مع العنصر نفسه.

يمكّنك الإصدار الأحدث من NET Runtime. من إنشاء كائن object نسخة أو مثيل دون الحاجة إلى تكرار اسم الفئة (استدعاء الفئة المستهدفة التي تم كتابة التعبير new من أجلها) على سبيل المثال، التعليمة المختصرة التالية سوف يقوم بإنشاء كائن للفئة Random

```
Random dice = new();
```

القصد من ذلك هو تبسيط إمكانية قراءة التعليمات البرمجية. لكن تذكر يجب استخدام الأقواس دائمًا عند كتابة تعبير new بهدف إنشاء كائن (نسخة جديدة من الفئة) المستهدفة.

لماذا يعتبر الأسلوب Next() ذا حالة؟

قد تتساءل عن سبب تنفيذ أسلوب Next() كأسلوب ذا حالة؟ ألا يستطيع مصممو مكتبة فئات .NET. اكتشاف طريقة لإنشاء عدد عشوائي دون الحاجة إلى متغيرات؟ وما الذي يتم تخزينه أو الإشارة إليه بالضبط بواسطة أسلوب Next()

هذه أسئلة منطقية. تعد أجهزة الكمبيوتر جيدة في اتباع تعليمات محددة على مستوى عالٍ، لإنشاء نتيجة موثوقة وقابلة للتكرار، لإيحاء انطباع بالعشوائية قرر مطورو أسلوب Next() التقاط التاريخ والوقت حتى الجزء من الملي ثانية، واستخدام ذلك في إنشاء قيمة أولية للتعليمات البرمجية تنتج عدداً مختلفاً في كل مرة، على الرغم من أنه ليس عشوائياً تماماً، إلا أنه يعد كافياً لمعظم التطبيقات، الحالة التي يتم التقاطها وصيانتها خلال عمر الكائن dice هي القيمة الأولية، يعيد كل استدعاء لاحق لأسلوب Next() تشغيل التعليمات البرمجية، لكنه يضمن أن القيمة الأولية تتغير بحيث لا يتم إرجاع نفس القيمة بالضرورة.

لاستخدام الأسلوب Random.Next() لا يجب عليك فهم كيفية عمله، الشيء المهم الذي يجب معرفته هو أن بعض الأساليب تتطلب منك إنشاء نسخة لفئة قبل استدعائها، والبعض الآخر لا يطلب ذلك.

كيف يمكنك تحديد ما إذا كنت بحاجة إلى إنشاء مثيل لفئة قبل استدعاء أساليبها؟

أحد النهج لتحديد ما إذا كان الأسلوب عديم الحالة هو الرجوع إلى وثائق Microsoft learn والمعلومات المتوفرة عنه، تتضمن الوثائق أمثلة توضح ما إذا كان يجب استدعاء الأسلوب من نسخة الكائن أو مباشرة من الفئة.

ملاحظة

قد تحتاج إلى التمرير لأسفل في صفحة الوثائق للعثور على أمثلة التعليمات البرمجية.

أو يمكنك محاولة الوصول إلى الأسلوب مباشرة من الفئة نفسها، كبديل للبحث من خلال الوثائق، إذا عمل، فأنت تعلم أنه أسلوب عديم الحالة، أسوأ ما يمكن أن يحدث هو الحصول على خطأ في التحويل البرمجي.

حاول الوصول إلى الأسلوب Random.Next() مباشرة دون نسخة، وشاهد ما يحدث.

١. أدخل السطر التالي من التعليمات البرمجية في محرر التعليمات Visual Studio

```
int result = Random.Next();
```

أنت تعرف أن Next() أسلوب ذا حالة، ومع ذلك يوضح هذا المثال كيفية تفاعل محرر التعليمات البرمجية Visual Studio عند محاولة الوصول إلى أسلوب ما بشكل غير صحيح.

٢. لاحظ أن خط أحمر متعرج يظهر تحت Random.Next مما يشير إلى أن لديك خطأ في التحويل البرمجي.
إذا كان الأسلوب الذي تهم باستخدامه عديم الحالة، فلن يظهر خط متعرج أحمر.

٣. مرر مؤشر الماوس فوق الخط الأحمر المتعرج، يجب أن تظهر نافذة منبثقة مع الرسالة التالية:

```
(1,14): error CS0120: An object reference is required for the
non-static field, method, or property 'Random.Next()'
```

كما رأيت التعليمات البرمجية في بداية الدرس، يمكنك إصلاح هذا الخطأ عن طريق إنشاء مثيل للفئة Random قبل الوصول إلى الأسلوب Next() على سبيل التذكير:

```
Random dice = new Random();
int roll = dice.Next();
```

في هذه الطريقة تم استدعاء الأسلوب Next() دون معلمات الإدخال.

خلاصة

- لاستدعاء أساليب الفئات من مكتبة .NET. يمكنك استخدام التنسيق اسم الفئة + اسم الأسلوب **ClassName.MethodName()** حيث يكون الرمز "." هو عامل تشغيل الوصول إلى أسلوب محدد في الفئة، وتكون الأقواس () هي عامل مشغلي استدعاء الأسلوب.
- عند استدعاء أسلوب عديم الحالة، لا تحتاج إلى إنشاء مثيل جديد من فئته أولاً.
- عند استدعاء أسلوب ذا حالة، تحتاج إلى إنشاء مثيل للفئة والوصول إلى الأسلوب من المثيل الكائن.
- استخدم عامل التشغيل new لإنشاء مثيل جديد للفئة.
- يسمى مثيل الفئة عنصراً أو كائن

اختبر معلوماتك

١. أي مما يلي يمثل الطريقة الصحيحة لإنشاء مثيل فئة؟

- `Random dice = new Random.Next();`
- `Random dice = new Random();`
- `String dice = new Random();`

٢. قام مطور بإنشاء مثيل للفئة Random وأطلق عليها أسم coins أي من أسطر التعليمات البرمجية التالية يمكن استخدامها لاستدعاء الأسلوب Next()

- `int money = new coins.Next();`
- `int money = Random.Next();`
- `int money = coins.Next();`

راجع إجابتك

١

```
Random dice = new Random();
```

صحيح تعريف النوع وتضمين عامل التشغيل new هو الطريقة الصحيحة لإنشاء كائن مثيل الفئة.

٢

```
int money = coins.Next();
```

صحيح تستخدم هذه العبارة لاستدعاء مثيل الفئة Random المسمى coins لإرجاع رقم عشوائي.

٤ إرجاع القيم ومعلمات الإدخال للأساليب

في الدرس السابق، استخدمت سيناريو ترميز "roll dice" لتوضيح الفرق بين الأساليب ذات الحالة (المثيل أو الكائن) والأساليب عديمة الحالة (الثابتة) يمكن أن يساعدك هذا السيناريو نفسه على فهم المفاهيم المهمة الأخرى، حول أساليب الاستدعاء، على سبيل المثال:

- معالجة القيمة المرجعة من أسلوب.
- تمرير معلمات الإدخال إلى أسلوب.
- اختيار إصدار محمل تحميلاً زائداً من أسلوب.

القيم المرجعة Return values

تم تصميم بعض الأساليب لإكمال وظيفتها وإنهاءها "بهدوء" بمعنى آخر، لا ترجع قيمة عند الانتهاء، ويشار إليها ب **أساليب باطلة** void methods وصممت أساليب أخرى لإرجاع قيمة عند الانتهاء، عادة ما تكون القيمة المرجعة نتيجة عملية ما.

قيمة الإرجاع هي الطريق الأساسي لأسلوب ما، للتواصل مرة أخرى مع التعليمات التي تستدعي الأسلوب.

لقد رأيت أن الأسلوب Random.Next() يقوم بإرجاع قيمة رقمية int تحتوي على العدد الذي تم إنشاؤه عشوائياً، ومع ذلك، يمكن تصميم أسلوب لإرجاع أي نوع بيانات أخرى، حتى يمكن إرجاع فئة، على سبيل المثال، تحتوي الفئة String على بعض الأساليب التي ترجع حروف، وبعضها يرجع عدداً صحيحاً integer وبعضها يرجع قيمة منطقية Boolean true or false

عند استدعاء أسلوب يرجع قيمة، غالباً ما تقوم بتعيين القيمة المرجعة إلى متغير، بهذه الطريقة، يمكنك استخدام القيمة لاحقاً في التعليمات البرمجية، في سيناريو dice قمت بتعيين القيمة المرجعة Random.Next() إلى متغير roll

```
int roll = dice.Next(1, 7);
```

في بعض الحالات، قد تحتاج إلى استخدام القيمة المرجعة مباشرة، دون تعيينها إلى متغير، على سبيل المثال، قد تحتاج إلى طباعة القيمة المرجعة إلى وحدة التحكم console كما يلي:

```
Console.WriteLine(dice.Next(1, 7));
```

على الرغم من أن الأسلوب يرجع قيمة، فمن الممكن استدعاء الأسلوب دون استخدام القيمة المرجعة، على سبيل المثال، يمكنك تجاهل القيمة المرجعة، عن طريق استدعاء الأسلوب كما يلي:

```
dice.Next(1, 7);
```

ومع ذلك، فإن تجاهل القيمة المرجعة سيكون بلا فائدة، لأن السبب في استدعاء الأسلوب `Next()` إمكانية استرداد القيمة العشوائية التالية.

معلمات الإدخال Input parameters

تسمى المعلومات التي يستقبلها أسلوب معلمة parameter يمكن للأسلوب استخدام معلمة واحدة أو أكثر parameters لإنجاز مهمته، أو لا شيء على الإطلاق.

في كثير من الأحيان، يتم استخدام مصطلحي "المعلمة parameter" و"الوسيطة argument" بالتبادل، ومع ذلك تشير **المعلمة parameter إلى المتغير الذي يتم استخدامه داخل الأسلوب** أما **الوسيطة argument هي القيمة التي يتم تمريرها عند استدعاء الأسلوب.**

تم تصميم معظم الأساليب لقبول معلمة إدخال واحدة أو أكثر، يمكن استخدام معلمات الإدخال لتحديد كيفية أداء الأسلوب لعمله، أو قد يتم تشغيلها مباشرة، على سبيل المثال الأسلوب `Random.Next()` يستخدم معلمات الإدخال لتكوين الحد الأدنى والأعلى للقيمة المرجعة، ومع ذلك، يستخدم الأسلوب `Console.WriteLine()` معلمة الإدخال مباشرة عن طريق طباعة القيمة إلى وحدة التحكم.

تستخدم الأساليب **توقيع أسلوب method signature** لتحديد عدد معلمات الإدخال التي سيقبلها الأسلوب، بالإضافة إلى نوع بيانات كل معلمة، يجب أن تلتزم عبارة الترميز التي تستدعي الأسلوب بالمتطلبات المحددة بواسطة توقيع الأسلوب، توفر بعض الأساليب خيارات، لعدد ونوع المعلمات التي يقبلها الأسلوب.

عندما يستدعي الأسلوب، فإنه يوفر قيمًا ملموسة، تسمى الوسائط، لكل معلمة، يجب أن تكون الوسائط متوافقة مع نوع المعلمة، ومع ذلك، لا يجب أن يكون اسم الوسيطة، إذا تم استخدامه في تعليمات الاستدعاء، هو نفس المعلمة المسماة، المحددة في الأسلوب.

راجع التعليمة البرمجية التالية:

```
Random dice = new Random();
int roll = dice.Next(1, 7);
Console.WriteLine(roll);
```

أنت حالياً تفهم كيف تعمل هذه التعليمات، ينشئ سطر التعليمات البرمجية الأول مثيلاً للفئة المسماة `Random dice` يستخدم سطر التعليمات الثاني الأسلوب `dice.Next(1, 7)` لتعيين قيمة عشوائية إلى متغير عدد صحيح يسمى `roll` لاحظ أن عبارة الاستدعاء توفر وسيطتين ل `Next()` مفصولتين برمز الفاصلة `,` يتضمن الأسلوب توقيع يقبل معلمتين إدخال من النوع الرقمي `int` تستخدم هذه المعلمات parameters لتكوين القيمة الأعلى والأدنى للرقم العشوائي، الذي تم إرجاعه، يستخدم سطر التعليمات النهائي الأسلوب `Console.WriteLine()` لطباعة قيمة المتغير `roll` إلى وحدة التحكم.

يجب أن تكون الوسيطات arguments التي تم تمريرها إلى أسلوب ما، تتوافق مع نوع البيانات، مثل نوع معلمات الإدخال المعرفة بواسطة الأسلوب، إذا حاولت تمرير وسيطة مكتوبة بشكل غير صحيح إلى أسلوب، فسيلتقط المحول البرمجي C# خطأك ويجبرك على تحديث عبارة الاستدعاء، قبل أن تقوم التعليمات البرمجية بالتحويل البرمجي والتشغيل.

يعد التحقق من نوع البيانات، إحدى الطرق التي تستخدمها C# and .NET لمنع المستخدمين النهائيين من مواجهة أخطاء في وقت التشغيل runtime

ملاحظة

على الرغم من أن معلمات الإدخال input parameters غالباً ما تستخدم، لا تتطلب جميع الأساليب معلمات الإدخال لإكمال مهمتها، على سبيل المثال، تتضمن الفئة Console أسلوب **Console.Clear()** لا يستخدم معلمات

الإدخال، نظراً لاستخدام هذا الأسلوب لمسح أي معلومات معروضة في وحدة التحكم، فإنه لا يحتاج إلى معلمات الإدخال لإكمال مهمته.

الأساليب ذات التحميل الزائد Overloaded methods

تحتوي العديد من الأساليب في مكتبة فئة .NET على تحميل زائد لتوقيعات الأساليب. من بين أمور أخرى، يمكّنك هذا من استدعاء الأسلوب مع أو بدون الوسائط المحددة arguments في بيان الاستدعاء. **يستطيع أسلوب واحد القيام بأكثر من مهمة والتعامل مع أنواع مختلفة من البيانات بتعليمه استدعاء واحدة.**

يتم تحديد **الأسلوب ذا التحميل الزائد** بتوقيعات أساليب متعددة، توفر الأساليب ذات التحميل الزائد طرقاً مختلفة لاستدعاء الأسلوب، أو توفير أنواع مختلفة من البيانات.

في بعض الحالات، يتم استخدام الإصدارات ذات التحميل الزائد "المثقلة" لتعريف معلمة إدخال input parameter (تحديد نوع البيانات) باستخدام أنواع بيانات مختلفة، على سبيل المثال، يحتوي الأسلوب `Console.WriteLine()` على ١٩ إصدار مختلف ذا تحميل زائد. تسمح معظم هذه التحميلات الزائدة للأسلوب بقبول أنواع مختلفة، ثم كتابة المعلومات المحددة إلى وحدة التحكم. راجع التعليمة البرمجية التالية:

```
int number = 7;
string text = "seven";

Console.WriteLine(number);
Console.WriteLine();
Console.WriteLine(text);
```

في هذا المثال، ستقوم باستدعاء ثلاثة إصدارات مختلفة، ذات تحميل زائد من أسلوب `WriteLine()`

- يستخدم الأسلوب الأول WriteLine() توقيع أسلوب يعرف معلمة parameter رقمية int
- يستخدم الأسلوب الثاني WriteLine() توقيع أسلوب يعرف معلمات parameter الإدخال الصفرية أو فارغة
- يستخدم الأسلوب الثالث WriteLine() توقيع أسلوب يعرف معلمة parameter حرفية string

في حالات أخرى، تحدد الإصدارات ذات التحميل الزائد لأسلوب عدداً مختلفاً من معلمات الإدخال input parameters تستخدم معلمات الإدخال البديلة لتوفير مزيد من التحكم في النتيجة المطلوبة، على سبيل المثال، يحتوي الأسلوب Random.Next() على إصدارات محملة بشكل زائد تمكنك من تعيين مستويات مختلفة من القيود أو الشروط على الرقم الذي تم إنشاؤه عشوائياً، وهكذا.

يستدعي التمرين التالي الأسلوب Random.Next() لإنشاء قيم عدد صحيح عشوائي مع مستويات مختلفة من القيود:

١. افتح ملف Program.cs فارغا في Visual Studio Code

٢. لفحص الإصدارات المحملة بشكل زائد من الأسلوب Random.Next() أدخل التعليمات البرمجية التالية:

```
Random dice = new Random();
int roll1 = dice.Next();
int roll2 = dice.Next(101);
int roll3 = dice.Next(50, 101);

Console.WriteLine($"First roll: {roll1}");
Console.WriteLine($"Second roll: {roll2}");
Console.WriteLine($"Third roll: {roll3}");
```

٣. في قائمة ملف **File** Visual Studio Code انقر فوق حفظ **Save**

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

تأكد من أن مسار المجلد المعروض في موجه الأوامر يشير إلى المجلد الحالي الذي يحتوي على ملف Program.cs

٥. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

لاحظ أن النتيجة مشابهة للإخراج التالي:

```
First roll: 342585470
Second roll: 43
Third roll: 89
```

الأرقام تم إنشاؤها عشوائية، لذلك ستكون نتائجك وفي كل مرة مختلفة، ومع ذلك، يوضح هذا المثال نطاق النتائج التي قد تراها.

خذ دقيقة لفحص التعليمات البرمجية.

لا يعيّن التحميل الأول من الأسلوب Next() حداً أعلى وأدنى، لذلك سيُرجع الأسلوب قيماً تتراوح من 0 إلى 2,147,483,647 وهي أقصى قيمة يمكن لـ int تخزينها.

يحدد التحميل الثاني من الأسلوب Next() الحد الأقصى للقيمة كحد أعلى، لذلك في هذه الحالة، يمكنك توقع قيمة عشوائية بين 0 و 100

يحدد التحميل الثالث من الأسلوب Next() كلاً من القيمة الأدنى والحد الأقصى، لذلك في هذه الحالة، يمكنك توقع قيمة عشوائية بين 50 و 100

٦. أغلق قائمة Terminal

لقد درست بالفعل العديد من الموضوعات في هذا الدرس، فيما يلي قائمة سريعة بما قمت بتغطيته:

- لقد درست كيفية استخدام method's return value القيمة المرجعة للأسلوب، عندما يوفر الأسلوب قيمة إرجاع return value
- لقد فحصت كيف يمكن لأسلوب استخدام معلمات الإدخال input parameters التي يتم تعريفها على أنها أنواع بيانات محددة.
- لقد فحصت الإصدارات ذات التحميل الزائد overloaded versions لبعض الأساليب التي تتضمن معلمات إدخال input parameters مختلفة أو أنواع معلمات parameter types

استخدام المقترحات الذكية IntelliSense

يتضمن Visual Studio Code ميزات المقترحات الذكية IntelliSense التي يتم تشغيلها بواسطة خدمة لغة ما، تشبه المقترحات التلقائية للكلمات، على سبيل المثال، توفر خدمة لغة C# تكملة مقترحة للتعليمات البرمجية، استناداً إلى دلالات اللغة، وتحليل سياق تعليماتك البرمجية، في هذا القسم، ستستخدمIntelliSense لمساعدتك في تنفيذ الأسلوب Random.Next()

نظرا لأن IntelliSense يتم كشفه داخل محرر التعليمات البرمجية، يمكنك معرفة الكثير عن أسلوب دون مغادرة بيئة الترميز، يوفر IntelliSense تلميحات ومعلومات مرجعية، في نافذة منبثقة أسفل موقع المؤشر أثناء إدخال تعليماتك البرمجية، عند كتابة التعليمات ستقوم النافذة المنبثقة IntelliSense بتغيير محتوياتها اعتماداً على السياق.

على سبيل المثال، أثناء إدخال الكلمة dice ببطء، سيعرض تحسس المقترحات الذكي IntelliSense جميع الكلمات الأساسية C# والمعرفات (أو بالأحرى، أسماء المتغيرات في التعليمات البرمجية) والفئات في مكتبة فئات NET. التي تطابق الأحرف التي يتم إدخالها، يمكن استخدام ميزات الإكمال التلقائي لمحرر التعليمات البرمجية، لإنهاء كتابة الكلمة المطابقة للتي في النافذة المنبثقة IntelliSense

لنختبر IntelliSense

١. تأكد من فتح ملف Program.cs

يجب أن يحتوي تطبيقك على التعليمات البرمجية التالية:

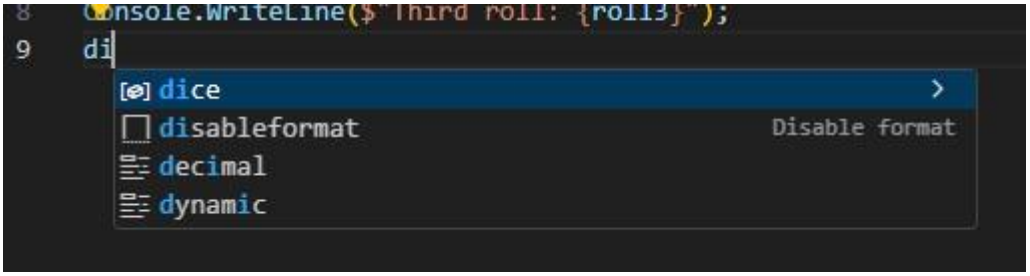
```
Random dice = new Random();
int roll1 = dice.Next();
int roll2 = dice.Next(101);
int roll3 = dice.Next(50, 101);

Console.WriteLine($"First roll: {roll1}");
Console.WriteLine($"Second roll: {roll2}");
Console.WriteLine($"Third roll: {roll3}");
```

٢. في أسفل ملف التعليمات البرمجية، لتجربة IntelliSense أدخل الأحرف `d, i, c` ببطء

٣. لاحظ النافذة المنبثقة IntelliSense التي تظهر عند بدء الكتابة.

عندما ينبثق IntelliSense يجب أن تظهر قائمة الاقتراحات، بحلول الوقت الذي قمت بإدخال الحروف `dic` يجب أن يكون المعرف `dice` في أعلى القائمة.



٤. اضغط على المفتاح Tab على لوحة المفاتيح.

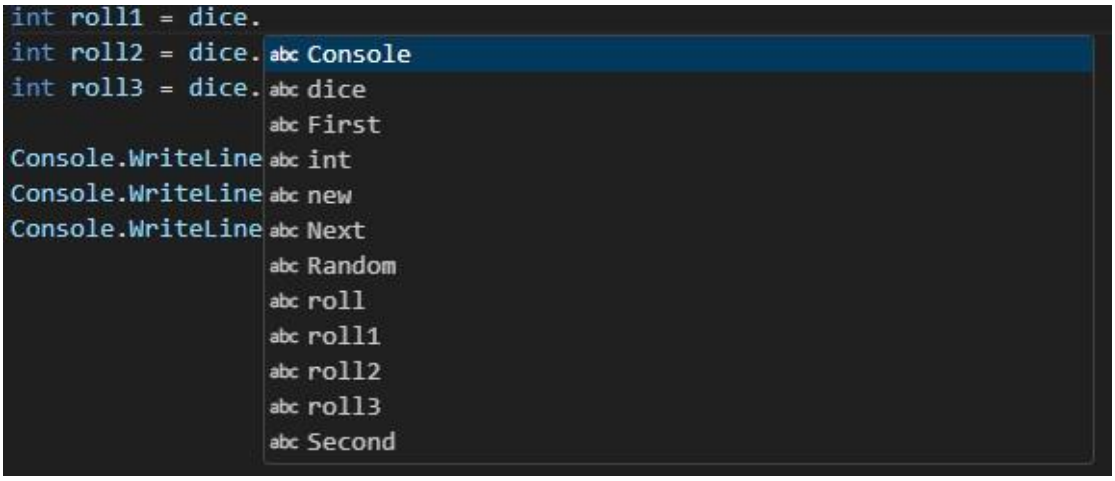
لاحظ اكتمال الكلمة `dice` بأكملها في المحرر، يمكنك استخدام مفتاحي السهم لأعلى ولأسفل لتغيير التحديد قبل الضغط على المفتاح Tab

ملاحظة

إذا اختفت نافذة IntelliSense يمكن تحديدها باستخدام المفتاح backspace على لوحة المفاتيح لحذف الحرف الأخير، ثم إعادة كتابة الحرف الأخير لفتح IntelliSense مرة أخرى.

٥. لتحديد عامل تشغيل وصول العضو، أدخل رمز .

لاحظ أن النافذة المنبثقة IntelliSense تظهر مرة أخرى عند إدخال . وتعرض قائمة غير مفلترة بجميع الأساليب المتوفرة (وأعضاء الفئات الآخرين)



٦. أدخل **N**

ستتم تصفية القائمة، ويجب أن تكون الكلمة Next هي التحديد الأول

٧. للإكمال التلقائي للكلمة بأكملها، اضغط على المفتاح Tab

٨. لتحديد عامل استدعاء الأسلوب، أدخل القوس)

لاحظ أنه تتم إضافة قوس الإغلاق تلقائياً لك.

عامل تشغيل استدعاء الأسلوب هو مجموعة الأقواس الموجودة على يمين اسم الأسلوب، هذا الجزء من عبارة الاستدعاء هو المكان الذي تحدد فيه

الوسائط the arguments التي سيتم تمريرها إلى الأسلوب، عامل تشغيل استدعاء الأسلوب مطلوب عند استدعاء الأسلوب.

٩. لاحظ أن النافذة المنبثقة IntelliSense تعرض الآن معلومات مفصلة حول `Random.Next()`

١٠. خذ دقيقة لفحص المعلومات المنبثقة IntelliSense للأسلوب `Random.Next()`

ملاحظة

إذا أغلقت النافذة المنبثقة IntelliSense قبل أن تسنح لك الفرصة لقراءتها، فاحذف الأقواس `()` ثم أدخل القوس `)` مرة أخرى لعرض النافذة المنبثقة.

لاحظ أن النافذة المنبثقة تتضمن ثلاثة أقسام، واحد على اليسار، واثنين على اليمين.

على الجانب الأيمن، يجب أن تشاهد في القسم العلوي

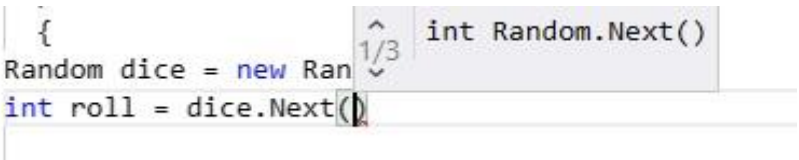
```
int Random.Next(int minValue, int maxValue)
```

وفي القسم السفلي

```
Returns a non-negative random integer
```

يعرف نوع الإرجاع للأسلوب `int` بمعنى آخر، عند تنفيذ هذا الإصدار من الأسلوب، فإنه سيرجع قيمة رقمية من النوع int

على الجانب الأيسر من النافذة المنبثقة IntelliSense فإنه يعرض `1/3` يشير 1/3 إلى أنك تبحث في التوقيعات الثلاثة الأولى للأسلوب `Next()` هذا الإصدار من توقيع الأسلوب يمكن الأسلوب من العمل مع أي معلمات إدخال (لا توجد وسائط تم تمريرها إلى الأسلوب في عبارة الاستدعاء)



لاحظ أن هناك أيضا سهم صغير أعلى وأسفل `1/3`

١١. لفحص الإصدار الثاني المحمل تحميلاً زائداً من الأسلوب، اضغط على مفتاح السهم لأسفل على لوحة المفاتيح.

يمكنك استخدام مفتاحي الأسهم لأعلى ولأسفل للتنقل بين الإصدارات المختلفة المحملة تحميلاً زائداً، عند القيام بذلك، سترى 1/3، 2/3، وتظهر 3/3 على الجانب الأيسر من النافذة المنبثقة IntelliSense وتفسيرات مفيدة على اليمين.

١٢. خذ دقيقة لفحص كل إصدار من إصدارات الأسلوب `Random.Next()`

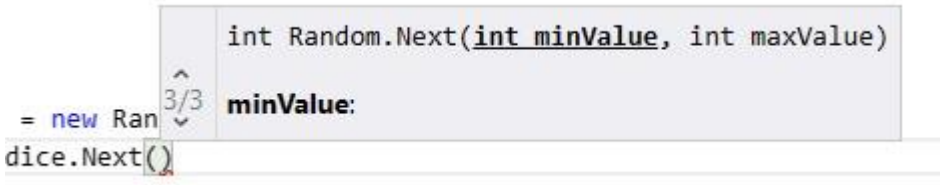
يبلغك الإصدار الثاني من الأسلوب `2/3` بأن `Next()` يمكن أن يقبل معلمة إدخال `int maxValue`

يخبرك الوصف أن `maxValue` هذا هو الحد الأعلى المتاح للرقم الذي تريد أن ينشئه `Next()`

لتكون دقيقاً يشير إلى أن رقم الإرجاع سيكون أقل من القيمة الأعلى maxValue لذلك عند تحديد `dice.Next(1,7);` الحد الأعلى للرقم العشوائي هو٦ لاحظ أنه تم تحديث الرسالة في أسفل القسم إلى:

`Returns a non-negative random integer that is less than the specified maximum`

يعلمك الإصدار الثالث `3/3` من `Next()` بقبول كلاً من القيمة الرقمية الأدنى والأعلى `int maxValue-int minValue` كمعلمات إدخال.



المعلمة الجديدة minValue هي الرقم الأقل للرقم الذي تريد أن ينشئه Next() أي أنه يشمل الرقم الأدنى المدخل نفسه، لذلك يمكن أن تكون القيمة المرجعة مساوية للقيمة الأدنى minValue توضح الرسالة الموجودة في الأسفل الآن ما يلي:

Returns a random integer that is within a specified range

في هذه الحالة، يوفر IntelliSense جميع المعلومات التي تحتاجها لتحديد التحميل الزائد المناسب، بما في ذلك شرح مفصل للقيمة الأقل والأعلى minValue - maxValue ومع ذلك، قد تواجه مواقف تحتاج فيها إلى الاطلاع على معلومات وثائق الأسلوب.

استخدام learn.microsoft.com للحصول على معلومات حول الأساليب المحملة بشكل زائد "المثقلة"

إن الطريقة الثانية للتعرف على الإصدارات ذات التحميل الزائد من الأساليب هي الرجوع إلى الوثائق الخاصة بالأسلوب، تساعدك الوثائق أيضاً على فهم الغرض من كل معلمة إدخال بالضبط.

١. للبدء، افتح مستعرض الويب المفضل لديك ومحرك البحث.

٢. إجراء بحث عن **C# Random.Next()**

يجب أن يتضمن البحث اسم الفئة واسم الأسلوب. قد تحتاج أيضاً إلى تضمين المصطلح C# للتأكد من عدم الحصول على نتائج عن طريق الخطأ للغات البرمجة الأخرى.

٣. حدد أعلى نتيجة بحث تستخدم عنوان يبدأ ب https://learn.microsoft.com

يجب أن تؤدي إحدى أفضل نتائج البحث إلى عنوان يبدأ learn.microsoft.com يجب أن يظهر عنوان الارتباط Random.Next Method

إليك الارتباط في حالة واجهت مشكلة في العثور عليه باستخدام محرك البحث:

Random.Next Method

٤. افتح عنوان C# Random.Next()

٥. افحص الوثائق.

قم بالتمرير لأسفل عبر محتويات الصفحة لمشاهدة نماذج التعليمات البرمجية المختلفة، لاحظ أنه يمكنك تشغيل العينات في نافذة المستعرض.

تتبع وثائق learn.microsoft.com تنسيقاً قياسياً لكل فئة وأسلوب في مكتبة فئات .NET.

١. بالقرب من أعلى صفحة الويب، حدد موقع القسم المسمى **Overloads**

لاحظ أن هناك ثلاث إصدارات محملة تحميلاً زائداً من الأسلوب المدرج، يتضمن كل إصدار مدرجاً ارتباطاً تشعبياً إلى موقع آخر أسفل الصفحة.

٢. للانتقال "أعلى الصفحة" إلى وصف للإصدار الثاني الذي تم تحميله بشكل زائد، حدد **Next(Int32)**

تتضمن الوثائق الخاصة بكل إصدار من الأسلوب ما يلي:

- وصف موجز لوظائف الأسلوب
- تعريف الأسلوب
- معلمات الإدخال التي يقبلها الأسلوب
- القيم المرجعة
- الاستثناءات التي يمكن تنفيذها
- أمثلة على الأسلوب المستخدم
- ملاحظات أخرى حول الأسلوب

٣. خذ دقيقة لمراجعة **قسم المعلمات** Parameters section

في قسم Parameters يمكنك قراءة أن معلمة الإدخال maxValue هي الحد الأعلى المحدد للرقم العشوائي الذي سيتم إنشاؤه، يعني الحد الأعلى أنه إذا كنت تريد أرقاماً لا تزيد عن 10 فيجب عليك تمرير القيمة 11

يمكنك أيضًا أن تقرأ في السطر التالي يجب أن تكون maxValue أكبر من أو تساوي 0 ماذا يحدث إذا تجاهلت هذا البيان؟ يمكنك أن ترى في قسم الاستثناءات أن الطريقة ستعيد ArgumentOutOfRangeException عندما تكون maxValue أقل من 0

ملاحظة

المحتوى في learn.microsoft.com هو مصدر الحقيقة، لمكتبة فئات .NET.
من المهم أن تأخذ الوقت الكافي لقراءة الوثائق لفهم كيفية عمل أسلوب معين.

خلاصة

- قد لا تقبل الأساليب أي معلمات، أو تقبل معلمات متعددة، اعتماداً على كيفية تصميمها وتنفيذها، لكن عند تمرير معامِلات إدخال متعددة، افصل بينها بعلامة الفاصلة ,
- قد ترجع الأساليب قيمة عند إكمالها لمهمتها، أو قد لا ترجع أي شيء (فارغ)
- تدعم الأساليب ذات التحميل الزائد العديد من عمليات تنفيذ الأسلوب، لكل منها توقيع فريد للأسلوب (عدد معلمات الإدخال ونوع البيانات لكل معلمة إدخال)
- يمكن أن يساعد IntelliSense في كتابة التعليمات بسرعة أكبر، يوفر مرجعاً سريعاً للأساليب، والقيم المرجعة، وإصداراتها ذات التحميل الزائد، وأنواع معلمات الإدخال الخاصة بها.

- learn.microsoft.com هو مصدر الحقيقة، عندما تريد معرفة كيفية عمل الأساليب في مكتبة فئات .NET.

اختبر معلوماتك

١ ما هي القيمة المرجعة؟

- نوع قيمة يتم إرجاعه بواسطة أسلوب.
- يشار إلى وسيطة في استدعاء الأسلوب كقيمة إرجاع داخل الأسلوب.
- قيمة سلسلة.

٢ ما هي معلمات الإدخال؟

- أنواع القيم (أو المتغيرات) داخل أسلوب.
- القيم التي تم إرجاعها بواسطة أسلوب.
- القيم التي تم تمريرها إلى أسلوب في عبارة الاستدعاء.

٣ ما هي طريقة التحميل الزائد؟

- أسلوب يرجع نوع قيمة.
- أسلوب مع أكثر من خمس معلمات.
- طريقة تدعم العديد من عمليات التنفيذ للأسلوب، ولكل منها توقيع أسلوب فريد

٤ كيف يساعد IntelliSense المطورين؟

- يمكن أن يساعد IntelliSense المطورين على كتابة التعليمات البرمجية بسرعة أكبر.
- يساعد IntelliSense المطورين على إعادة بناء التعليمات البرمجية الخاصة بهم.
- يمكن ل IntelliSense تغيير "نسق.IDE "

راجع إجاباتك

١- نوع قيمة يتم إرجاعه بواسطة أسلوب

صحيح هذا هو التعريف الصحيح لقيمة الإرجاع

٢-أنواع القيم (أو المتغيرات) داخل أسلوب

صحيح هذا هو التعريف الصحيح لمعلمة الإدخال

٣-طريقة تدعم العديد من عمليات التنفيذ للأسلوب، ولكل منها توقيع أسلوب فريد

صحيح هذا هو التعريف الصحيح لأسلوب التحميل الزائد

٤-يمكن أن يساعد IntelliSense المطورين على كتابة التعليمات بسرعة أكبر

صحيح يمكن ل IntelliSense التقاط الأخطاء الإملائية، والاستخدام غير الصحيح، واقتراح التصحيحات، هذه الأشياء تسرع عملية الترميز

٥ إكمال نشاط تحدي لاكتشاف وتنفيذ استدعاء أسلوب

سوف تعزز تحديات التعليمات البرمجية ما تعلمته، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

تحدي أساليب فئة الرياضيات

في هذا التحدي، ستستخدم إما IntelliSense أو learn.microsoft.com للعثور على أسلوب، يقوم بإرجاع الرقم الأكبر من بين رقمين، واستدعائه.

تحدي التعليمات البرمجية: تنفيذ أسلوب من فئة الرياضيات التي ترجع الرقم الأكبر من بين رقمين

فيما يلي متطلبات التحدي الخاصة بك:

١. تأكد من أن لديك ملف Program.cs فارغ ومفتوح في Visual Studio Code

٢. أدخل التعليمات البرمجية التالية كنقطة بداية للتحدي:

```
int firstValue = 500;
int secondValue = 600;
int largerValue;
Console.WriteLine(largerValue);
```

يجب أن يستخدم الحل النهائي لهذا التحدي هذه التعليمة البرمجية، ويجب استخدام العبارة `Console.WriteLine(largerValue);` لإنشاء الإخراج

٣. ابحث عن أسلوب للفئة System.Math التي ترجع الرقم الأكبر من رقمين.

يمكنك استخدام إما IntelliSense أو learn.microsoft.com للعثور على الأسلوب، وتحديد كيفية استدعائه بشكل صحيح.

٤. استخدم الأسلوب الذي وجدته لتعيين قيمة للمتغير المسمى largerValue يجب أن تكون قادراً على تمرير قيمتي العدد الصحيح `firstValue, secondValue` إلى الأسلوب الذي وجدته، ويرجع قيمة من النوع `int` الذي يمثل أكبر وسيطتين قمت بتمريرهما.

يمكنك تعيين القيمة المرجعة إلى largerValue على سطر تعليمة برمجية جديد أو على سطر التعليمات الموجودة لتعريف largerValue

٥. تحقق من أن تطبيقك يكتب الإخراج التالي:

```
600
```

سواء واجهتك مشكلة، أو تحتاج إلقاء نظرة خاطفة على الحل، أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٦ راجع الحل لاكتشاف نشاط تحدي استدعاء الأسلوب وتنفيذه

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق

```
int firstValue = 500;
int secondValue = 600;
int largerValue;
largerValue = Math.Max(firstValue,
secondValue);
Console.WriteLine(largerValue);
```

يدعم الأسلوب `Math.Max()` 13 إصدار محمل بشكل زائد، لقبول أنواع بيانات مختلفة، الإصدار من `Math.Max()` الذي تستدعيه يقبل `int` كمعلمتي إدخال، ويعيد أكبر قيمة من القيمتين `int`

عند تشغيل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
600
```

نهنئك في حال نجاحك! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ اختبر معلوماتك

١- ما هو العنصر أو الكائن object؟

- عنوان ذاكرة
- فئة عديمة الحالة
- مثيل فئة

٢- أي من العبارات التالية صحيحة؟

- تحدد مكتبة فئات .NET دائماً الأساليب ذات التحميل الزائد لكل أسلوب
- تحتوي مكتبة فئات .NET على تعريفات لأنواع البيانات المستخدمة في C#
- تتضمن مكتبة فئات .NET العديد من الفئات بحيث تزيد بالفعل من وقت التطوير

٣- أي من الرموز التالية يمثل عامل استدعاء الطريقة المطلوبة لتنفيذ أسلوب؟

- .
- {}
- ()

راجع إجاباتك

١ الكائن object مثيل فئة

صحيح يمثل الكائن إنشاء مثيل لفئة

٢ تحتوي مكتبة فئات .NET. على تعريفات لأنواع البيانات المستخدمة في C#.

صحيح مكتبة فئات .NET. هي المكان الذي يمكن فيه العثور على معلومات عن أنواع بيانات C#

٣ ()

صحيح الأقواس المفتوحة والإغلاق مطلوبة لتنفيذ أسلوب، يتم أيضاً استخدام الأقواس عند تمرير الوسائط إلى أسلوب

٨ الملخص

كان هدفك هو استدعاء أساليب محددة من مكتبة فئات .NET. لتنفيذ مهام مثل إنشاء رقم عشوائي، أو إجراء عملية رياضية.

باستخدام بناء جملة C# قمت باستدعاء أنواع مختلفة من الأساليب في مكتبة فئات .NET. الأساليب التي أرجعت القيم، والأساليب التي قبلت معلمات إدخال متعددة، والأساليب التي حافظت على الحالة الثابتة، لقد استخدمت IntelliSense وموقع learn.microsoft.com للبحث عن الأساليب، ولتحسين فهمك لما يفعله الأسلوب وكيفية عمله.

تخيل مقدار الجهد الذي سيبذل لتطوير البرامج إذا لم يكن لديك مكتبة واسعة، من الفئات والأساليب (التي لم تقوم ببنائها شخصياً) تعد مكتبة فئات .NET. نعمة لملايين المطورين، تعتمد عشرات الآلاف من الشركات على تطبيقات مبنية على الأساليب المطبقة في مكتبة فئات .NET.

خلال تعرفك على كيفية الاستفادة من مكتبة فئات .NET. اتخذت خطوة كبيرة نحو فهم كيفية قيام مطوري البرامج بإنشاء تطبيقات، لها استخدام في العالم الحقيقي.

الوحدة الثالثة

إضافة منطق القرار إلى التعليمات البرمجية باستخدام عبارات statements if, else, and else if

تعلم تفريع مسار تنفيذ التعليمات البرمجية، عن طريق تقييم التعبيرات المنطقية.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- كتابة التعليمات البرمجية التي تقيم الشروط باستخدام عبارات the statements if, else, and else if
- إنشاء تعبيرات منطقية لتقييم شرط.
- دمج التعبيرات المنطقية باستخدام عوامل التشغيل المنطقية.
- قم بمداخلة كتل التعليمات البرمجية ضمن كتل تعليمات برمجية أخرى.

محتويات الوحدة:

١- مقدمة

٢- إنشاء منطق قرار باستخدام عبارات if

٣- إنشاء منطق قرار متداخل مع if و else if و else

٤- إكمال نشاط تحدي لتطبيق قواعد العمل

٥- مراجعة الحل لتطبيق نشاط تحدي قواعد العمل

٦- التحقق من المعرفة

٧- الملخص

١ المقدمة

تتيح لك لغة البرمجة C# بناء التطبيقات التي تستخدم منطق صنع القرار، افترض أنك تريد عرض معلومات مختلفة للمستخدم النهائي استنادًا إلى بعض قواعد العمل، على سبيل المثال، إذا كنت تريد عرض رسالة خاصة على فاتورة عميل طبقًا لمنطقته الجغرافية؟ ماذا لو كنت تريد إعطاء عميل خصمًا طبقًا لحجم طلبيته؟ أو ماذا لو أردت عرض وظيفة موظف طبقًا لمستواه في الشركة، في كل حالة، ستحتاج إلى إضافة منطق القرار.

في نهاية هذه الوحدة، ستكون قادرًا على كتابة تعليمات برمجية يمكنها تغيير تدفق تنفيذ التعليمات البرمجية، استنادًا إلى بعض المعايير.

هام

تتضمن هذه الوحدة أنشطة الترميز التي تتطلب Visual Studio Code ستحتاج إلى الوصول إلى بيئة تطوير Visual Studio Code وتكوينها لتطوير تطبيقات C#

٢ تمرين - إنشاء منطق قرار باستخدام عبارات if statements

تتضمن معظم التطبيقات عددا كبيرا من مسارات التنفيذ، على سبيل المثال، يمكن للتطبيق تنفيذ مسارات مختلفة، استناداً إلى قائمة الخيارات التي يحددها المستخدم، يشير المطورون إلى التعليمات البرمجية التي تنفذ مسارات تنفيذ مختلفة باسم **فروع التعليمات** البرمجية **code branches**

عبارة تفريع التعليمات البرمجية الأكثر استخداماً هي العبارة `if` تعتمد العبارة `if` على تعبير منطقي محاط بمجموعة من الأقواس، إذا كان التعبير صحيحاً، فسيتم تنفيذ التعليمة البرمجية بعد العبارة `if` إذا لم يكن الأمر كذلك، يتجاهل وقت تشغيل .NET runtime التعليمات البرمجية ولا ينفذها.

```
if (شرط معين لتنفيذ التعليمات بين الأقواس المتعرجة)
{
التعليمات البرمجية حال تحقق الشرط
}
```

في هذا التمرين، ستتدرب على كتابة عبارات `if` من خلال إنشاء لعبة، أولاً عليك تحديد قواعد اللعبة، ثم عليك تنفيذها برمجياً.

ستستخدم الأسلوب `Random.Next()` لمحاكاة رمي ثلاثة أحجار زهر سداسية الجوانب، ستقوم بتقييم القيم المحتسبة لحساب النتيجة، إذا كانت النتيجة أكبر من إجمالي العدد العشوائي، فستعرض رسالة رابحة للمستخدم، إذا كانت النتيجة أقل من القطع، فستعرض رسالة خاسرة للمستخدم.

- إذا كان ناتج إلقائك لأي حجرين هو نفس القيمة، فستحصل على نقطتين مكافأة لحصولك على زوج مماثل.
- إذا كان ناتج إلقائك لثلاثة أحجار هو نفس القيمة، فستحصل على ست نقاط مكافأة لحصولك على ثلاثة مماثلين.

- إذا كان مجموع إلقاء الأحجار الثلاثة، بالإضافة إلى أي نقاط مكافأة، هو ١٥ أو أكثر، فستفوز في المباراة، وإذا لم يكن، فسوف تخسر.

ستقوم بتحسين القواعد أثناء معرفة المزيد حول العبارة if

هام

يستخدم هذا التمرين على نطاق واسع الفئة System.Random يمكنك الرجوع إلى الوحدة بعنوان أساليب الاستدعاء من مكتبة فئات .NET. إذا كنت بحاجة إلى تذكر كيفية عمل Random.Next()

إعداد بيئة الترميز الخاصة بك

تتضمن هذه الوحدة أنشطة ترشدك خلال عملية إنشاء نموذج التعليمات البرمجية وتشغيله، يتم تشجيعك على إكمال هذه الأنشطة باستخدام Visual Studio Code كبيئة تطوير، سيساعدك استخدامه على أن تصبح أكثر راحة في كتابة التعليمات البرمجية، وتشغيلها في بيئة تطوير يستخدمها المحترفون في جميع أنحاء العالم.

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف (File) حدد فتح مجلد Open Folder

٣. في مربع الحوار فتح مجلد، انتقل إلى مجلد سطح مكتب

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام موقع هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في مربع الحوار فتح مجلد، حدد تحديد مجلد Select Folder

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة Terminal حدد New Terminal

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

ملاحظة

إذا كنت تعمل على جهاز الكمبيوتر الخاص بك، وأكملت وحدات Microsoft Learn الأخرى في سلسلة C# هذه، فربما تكون قد أنشأت بالفعل مجلد مشروع لاختبار التعليمات البرمجية، إذا كان الأمر كذلك، يمكنك تخطي الخطوة التالية، والتي تستخدم لإنشاء تطبيق وحدة تحكم في مجلد TestProject

٦. في موجه الأوامر Terminal لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب `dotnet new console -o ./CsharpProjects/TestProject` ثم اضغط على Enter

يستخدم هذا الأمر .NET CLI. قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد، ينشئ الأمر مجلدات CsharpProjects و TestProject نيابة عنك، ويستخدم TestProject `.csproj` امتداد لملفك

٧. في قائمة الاستكشاف EXPLORER قم بتوسيع المجلد CsharpProjects

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# يسمى `Program.cs` وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة الاستكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر حدد Program.cs

٩. حذف أسطر التعليمات البرمجية الموجودة.

ستستخدم مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق قائمة موجه الأوامر Terminal

كتابة التعليمات البرمجية التي تنشئ ثلاثة أرقام عشوائية وتعرضها في الإخراج

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

٢. لإنشاء التعليمات البرمجية الأولية لهذا التمرين، أدخل ما يلي:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");
```

٣. خذ دقيقة لمراجعة التعليمات البرمجية التي أدخلتها.

للبدء، يمكنك إنشاء مثيل جديد أو نسخة كائن للفئة `System.Random` وتخزين مرجع الكائن `dice` في متغير يسمى بعد ذلك، يمكنك استدعاء الأسلوب `Random.Next()` من خلال الكائن `dice` ثلاث مرات، مع توفير كل من الحد الادنى والأعلى لتقييد القيم المحتملة بين ١و٦ (الحد العلوي خاص) يمكنك حفظ الأرقام العشوائية الثلاثة في المتغيرات `roll1`, `roll2`, `roll3` على التوالي.

بعد ذلك، يمكنك جمع لفات الحجر الثلاثة، وحفظ القيمة في متغير عدد صحيح يسمى `total`

أخيراً، يمكنك استخدام الأسلوب WriteLine() لعرض القيم الثلاث باستخدام دمج التسلسل النصي.

عند تشغيلك للتعليمات البرمجية، يجب أن تشاهد الرسالة التالية (ستكون الأرقام مختلفة كل مرة).

```
Dice roll: 4 + 5 + 2 = 11
```

كانت هذه المهمة الأولى، مهمة إعداد، الآن، يمكنك إضافة منطق القرار إلى تعليماتك البرمجية، لجعل اللعبة أكثر إثارة للاهتمام.

إضافة عبارة if لعرض رسائل مختلفة استناداً إلى قيمة المتغير الإجمالي

١. في محرر التعليمات Visual Studio أسفل ملف التعليمات البرمجية، ثم قم بإنشاء سطر فارغ.

٢. لإنشاء ميزة لعبتك الأولى، أدخل العبارات التالية **if**

```
if (total > 14)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

يتم استخدام عبارتي if للتعامل مع السيناريوهات الربح والخسارة. خذ دقيقة لفحص العبارة الأولى if

لاحظ أن العبارة if تتكون من ثلاثة أجزاء:

- الكلمة الأساسية `if`
- تعبير منطقي بين أقواس `()`
- كتلة تعليمات برمجية معرفة بواسطة أقواس متعرجة `{ }`

```
if ( 20 > 14 )
  {
          Console.WriteLine("العدد ٢٠ أكبر من ١٤");
  }
```

في وقت التشغيل، يتم تقييم التعبير المنطقي `total > 14` إذا كانت هذه عبارة حقيقية إذا كانت قيمة `total` أكبر من 14 فسوف يستمر تدفق التنفيذ في التعليمات البرمجية، المعرفة داخل كتلة تعليمات العبارة `if` بمعنى آخر، سيتم تنفيذ التعليمات البرمجية داخل الأقواس المتعرجة.

ومع ذلك، إذا كان التعبير المنطقي خطأ قيمة `total` ليست أكبر من 14 فسوف يتخطى تدفق التنفيذ كتلة تعليمات العبارة `if` بمعنى آخر، لن يتم تنفيذ التعليمات البرمجية داخل الأقواس المتعرجة.

أخيراً، تتحكم عبارة `if` الثانية في الرسالة في حالة خسارة المستخدم. في الدرس التالي، سوف تستخدم تبايناً على العبارة `if` لاختصار هاتين العبارتين في عبارة واحدة تعبر بوضوح أكبر عن الهدف.

ما التعبير المنطقي Boolean expression

التعبير المنطقي أو "الشرط" هو جميع التعليمات البرمجية التي ترجع قيمة منطقية، إما `true` or `false` هذه التعبيرات هي من أبسط التعبيرات المنطقية، بدلاً من ذلك، يمكن أن يكون التعبير المنطقي نتيجة أسلوب يرجع القيمة `true` or `false` فيما يلي مثال بسيط باستخدام الأسلوب `string.Contains()` لتقييم ما إذا كانت جملة واحدة تحتوي على كلمة معينة.

```
string message = "الثعلب البني يقفز فوق الكلب الكسول";
bool result = message.Contains("الكلب");
Console.WriteLine(result);

if (message.Contains("الثعلب"))
{
    Console.WriteLine("ماذا فعل الثعلب؟");
}
```

نظراً لأن `message.Contains()` تعرض قيمة `true` or `false` فإنها مؤهلة كتعبير منطقي، ويمكن استخدامها في عبارة `if`

يمكن إنشاء تعبيرات منطقية بسيطة أخرى، باستخدام عوامل تشغيل مقارنة قيمتين، ويشمل المشغلون:

- `==` عامل التشغيل "يساوي" لاختبار المساواة
- `>` عامل التشغيل "أكبر من" لاختبار أن القيمة الموجودة على اليسار أكبر من القيمة الموجودة على اليمين
- `<` عامل التشغيل "أقل من" لاختبار أن القيمة على اليسار أقل من القيمة الموجودة على اليمين
- `>=` عامل التشغيل "أكبر من أو يساوي"
- `<=` عامل التشغيل "أقل من أو يساوي"
- وما إلى ذلك...

ملاحظة

يكرس هذا المسار التعليمي C# وحدة كاملة للتعبيرات المنطقية، هناك العديد من عوامل التشغيل التي يمكنك استخدامها لإنشاء تعبير منطقي، سنغطي فقط

بعض الأساسيات هنا، في هذه الوحدة. لمزيد من المعلومات حول التعبيرات المنطقية، راجع الوحدة بعنوان تقييم التعبيرات المنطقية لاتخاذ القرارات في "Evaluate Boolean expressions to make decisions in C# C#"

في هذا المثال، قمت بتقييم التعبير المنطقي total > 14 ومع ذلك، كان بإمكانك اختيار التعبير المنطقي total >= 15 لأنه في هذه الحالة، يكونان متشابهين، نظراً لأن قواعد اللعبة تحدد "إذا كان مجموع الأحجار الثلاثة، بالإضافة إلى أي مكافآت، هو ١٥ أو أكثر، فإنك تفوز باللعبة" فمن المحتمل أن تقوم بتنفيذ التعبير total >= 15 ستقوم بإجراء هذا التغيير في الخطوة التالية من التمرين.

ما هي كتلة التعليمات البرمجية؟ code block

كتلة التعليمات البرمجية هي مجموعة من سطر واحد أو أكثر من التعليمات البرمجية التي يتم تعريفها بواسطة رمز الأقواس المتعرجة {} وهو يمثل وحدة كاملة من التعليمات البرمجية التي لها غرض واحد في نظام البرامج، وفي هذه الحالة، وفي وقت التشغيل، يتم تنفيذ كافة أسطر التعليمات البرمجية داخل الكتلة إذا كان التعبير المنطقي صحيحاً، على العكس من ذلك، إذا كان التعبير المنطقي خاطئاً، فسيتم تجاهل جميع أسطر التعليمة البرمجية داخل الكتلة.

يجب أن تعرف أيضاً أن كتل التعليمات البرمجية يمكن أن تحتوي داخلها على كتل تعليمات أخرى، في الواقع، من الشائع أن تكون كتلة التعليمات البرمجية "متداخلة" داخل كتلة تعليمات برمجية أخرى في تطبيقاتك، ستبدأ في كتابة الكتل المتداخلة، لاحقاً في هذه الوحدة، عند إنشاء عبارة if واحدة داخل كتلة تعليمات أخرى.

ملاحظة

يكرس هذا المسار التعليمي وحدة كاملة لفهم كتل التعليمات البرمجية. تعتبر كتل التعليمة البرمجية أساسية لفهم تنظيم وهيكل التعليمة البرمجية، وهي تحدد حدود نطاق المتغير. راجع الوحدة التحكم في نطاق المتغير والمنطق باستخدام كتل التعليمات البرمجية [Control variable scope and logic using code blocks in C#]

إضافة عبارة if أخرى لتنفيذ مكافأة doubles

بعد ذلك، يمكنك تنفيذ القاعدة: إذا كان أي من حجري الزهر تقوم بتدحرجهما ينتج عنهما نفس القيمة، فستحصل على نقطتي مكافأة لتدحرج مزدوج، عدل التعليمات البرمجية من الخطوة السابقة لمطابقة التعليمات التالية:

١. في محرر Visual Studio Code حدد سطر فارغ أعلى العبارة if الأولى

٢. لإنشاء ميزة اللعبة "doubles" أدخل عبارة if التالية

```
if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    Console.WriteLine("You rolled doubles! +2
bonus to total!");
    total += 2;
}
```

هنا يمكنك دمج ثلاث تعبيرات منطقية، لإنشاء تعبير منطقي مركب واحد، فوق سطر واحد من التعليمات البرمجية، ويسمى هذا أحياناً حالة مركبة compound condition لديك مجموعة خارجية واحدة من الأقواس، تجمع بين ثلاث مجموعات داخلية من الأقواس، مفصولة برمزين من الشرطة العمودية || يمكنك إطلاق عليها رمز الأنبوبة.

الشُرط العمودية المزدوجة || هي مشغل OR المنطقي، والتي تقول في الأساس **"إما أن يكون التعبير على اليسار أو التعبير على اليمين صحيحاً حتى يكون التعبير المنطقي بأكمله صحيحاً"** وإذا كان كلا التعبيرين المنطقيين خطأ، فإن التعبير المنطقي بأكمله غير صحيح، يمكنك استخدام عاملي OR منطقيين حتى تتمكن من توسيع التقييم ليشمل تعبيرًا منطقيًا ثالثًا.

أولاً، يمكنك تقييم (roll1 == roll2) إذا كان هذا صحيحًا، فإن التعبير بأكمله صحيح، إذا كان خطأ يمكنك تقييم (roll2 == roll3) إذا كان هذا صحيحًا، فإن التعبير بأكمله صحيح، إذا كان خطأ، يمكنك تقييم (roll1 == roll3) إذا كان هذا صحيحًا، فإن التعبير بأكمله صحيح، إذا كان هذا خطأ، فإن التعبير بأكمله غير صحيح.

إذا كان التعبير المنطقي المركب صحيحاً، فنفذ كتلة التعليمات البرمجية. هذه المرة، هناك سطران من التعليمات البرمجية، يقوم السطر الأول من التعليمات بطباعة رسالة للمستخدم، السطر الثاني من التعليمات يزيد قيمة total بمقدار 2

٣. لتحسين قابلية قراءة التعليمات البرمجية، قم بتحديث عبارة if الثانية كما يلي:

```
if (total >= 15)
```

لاحظ أنك تستخدم عامل التشغيل في التعبير >= يعني **"أكبر من أو يساوي"** ونتيجة لذلك، يمكنك مقارنة total بقيمة 15 بدلاً من 14 من المفترض أن يساعد هذا في تسهيل فهم التعليمات البرمجية (قراءة أكثر سهولة) نظراً لأنك تتعامل مع قيم عدد صحيح، سيعمل التعبير (total >= 15) الجديد بشكل مماثل لما كتبته سابقاً (total > 14)

٤. خذ دقيقة لمراجعة التعليمات البرمجية. يجب أن تتطابق التعليمة البرمجية مع ما يلي:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
```

```
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    Console.WriteLine("You rolled doubles! +2
bonus to total!");
    total += 2;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}
if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

لاحظ المحاذاة المحسنة بين التعبيرات المستخدمة لتقييم القوائم الفائزة والخاسرة.

إضافة عبارة if أخرى لتنفيذ مكافأة ثلاثية

بعد ذلك، يمكنك تنفيذ القاعدة: إذا كانت أحجار الزهر الثلاثة التي تقوم بتدحرجها تؤدي إلى نفس القيمة، فستحصل على ست نقاط مكافأة، لذلك،

عدل التعليمات البرمجية من الخطوات السابقة لمطابقة قائمة التعليمات البرمجية التالية:

١. في محرر التعليمات البرمجية Visual Studio قم بإنشاء سطر فارغ أسفل كتلة التعليمات البرمجية لعبارة if الزوجية doubles

٢. لإنشاء ميزة اللعبة الثلاثية triples أدخل عبارة if التالية:

```
if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples! +6
bonus to total!");
    total += 6;
}
```

هنا يمكنك دمج شرطيين أو تعبيرين منطقيين لإنشاء تعبير منطقي مركب، فوق سطر واحد من التعليمات البرمجية. لديك مجموعة خارجية واحدة من الأقواس، تجمع بين مجموعتين داخليتين من الأقواس مفصولة بحرفين من **&&**

علامات العطف المزدوجة **&&** هي مشغل AND المنطقي، الذي يقول بشكل أساسي إذا كان كلا التعبيرين صحيحين، فإن التعبير بأكمله صحيحاً.

في هذه الحالة، إذا كان roll1 يساوي roll2 و roll2 يساوي roll3 وثم بالاستنتاج، يجب أن يكون roll1 يساوي roll3 وقام المستخدم بدحرجة الأحجار ثلاث مرات.

٣. في قائمة ملف **File** Visual Studio Code انقر فوق حفظ **Save**

٤. خذ دقيقة لمراجعة تعليماتك البرمجية. تأكد من أن التعليمات تطابق ما يلي:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
```

```
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    Console.WriteLine("You rolled doubles! +2
bonus to total!");
    total += 2;
}

if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples! +6
bonus to total!");
    total += 6;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

٥. في قائمة استكشاف EXPLORER لفتح موجة الأوامر Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح قائمة Terminal ويظهر أن Terminal مفتوحة في موقع مجلد TestProject

٦. في قائمة موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ملاحظة

إذا رأيت رسالة تقول "تعذر العثور على المشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج الذي يشبه إحدى النتائج التالية:

```
Dice roll: 3 + 6 + 1 = 10
Sorry, you lose.
```

أو هكذا:

```
Dice roll: 1 + 4 + 4 = 9
You rolled doubles! +2 bonus to total!
Sorry, you lose.
```

أو هكذا:

```
Dice roll: 5 + 6 + 4 = 15
You win!
```

أو إذا كنت محظوظًا، فسترى:

```
Dice roll: 6 + 6 + 6 = 18
```

```
You rolled doubles! +2 bonus to total!
You rolled triples! +6 bonus to total!
You win!
```

ولكن انتظر، هل يجب أن تكافئ اللاعب بكل من المكافأة الثلاثية والمكافأة المزدوجة مرة واحدة؟ لفة الثلاثي يعني أنها أيضاً تدحرجت مزدوجة، من الناحية المثالية، لا ينبغي أن تتكدس المكافآت، يجب أن يكون هناك شرطان منفصلان للمكافأة، هذا خطأ منطقي يجب تصحيحه، وليس خطأ برمجي.

المشاكل في المنطق وفرص تحسين التعليمات البرمجية

على الرغم من أن هذه بداية جيدة، وتعلمت الكثير عن عبارة if والتعبيرات المنطقية أو الشروط، والكتل البرمجية، وعوامل التشغيل المنطقية OR وAND وما إلى ذلك، هناك الكثير الذي يمكن تحسينه. وستقوم بذلك في الدرس التالي.

خلاصة

- استخدم عبارة if لتفريع منطق التعليمات البرمجية، سوف تتخذ عبارة if القرار بتنفيذ التعليمات البرمجية داخل الكتلة البرمجية الخاصة بها، إذا كان التعبير المنطقي الخاص بها يساوي true بمعني آخر إذا تحقق الشرط داخل الأقواس، وإلا، سيقوم وقت التشغيل بتخطي كتلة if البرمجية والمتابعة للسطر التالي من التعليمات البرمجية بعد الكتلة.
- التعبير المنطقي "الشرط" كل تعبير يُرجع قيمة منطقية.
- ستقوم عوامل التشغيل المنطقية مثل == أو =< بمقارنة القيمتين جهة اليسار واليمين للمساواة، والمقارنة، وأكثر من ذلك.
- تُعرف كتلة التعليمات البرمجية بواسطة أقواس متعرجة { }وتجمع أسطر من التعليمات البرمجية التي يجب أن تعامل كوحدة واحدة.
- يجمع عامل التشغيل المنطقي && "AND" تعبيرين بحيث يجب أن يكون كلا التعبيران الفرعيان صحيحين لكي يكون التعبير بأكمله صحيحاً.
- يجمع عامل التشغيل المنطقي || "OR" تعبيرين بحيث إذا كان تعبير فرعي واحد صحيحاً، يكون التعبير بأكمله صحيحاً.

اختبر معلوماتك

١- ما هي كتلة التعليمات البرمجية؟

- مكتبة فئات .NET.
- أسطر التعليمات البرمجية التي يجب التعامل معها كوحدة واحدة.
- كتلة من التعليمات البرمجية التي تم حظر الوصول إليها.

١- ما هو الشرط أو التعبير المنطقي؟

- تعبير معامل
- مصطلح ترتيبي
- التعليمات البرمجية التي ترجع إما true أو false

راجع إجابتك

١ أسطر التعليمات البرمجية التي يجب التعامل معها كوحدة واحدة.

صحيح يتم تعريف كتلة التعليمات البرمجية بواسطة أقواس متعرجة { } وتجمع أسطر من التعليمات البرمجية التي يجب أن تعامل كوحدة واحدة.

٢ التعليمات البرمجية التي ترجع إما true أو false

صحيح التعبير المنطقي هو أي تعليمات برمجية تقوم بإرجاع قيمة منطقية، إما true أو false

٣ إنشاء منطق قرار متداخل مع if, else if, else

في الدرس السابق، استخدمت عبارات if متعددة لتنفيذ قواعد اللعبة، ومع ذلك، لاحظت أن هناك حاجة إلى عبارات if أكثر تعبيراً لإصلاح خطأ منطقي في تعليمات البرمجية.

في هذا التمرين، سوف تستخدم عبارات if, else if, else لتحسين خيارات التفريع في التعليمات، وإصلاح الخطأ منطقي.

استخدام عبارات if, else بدلا من عبارات if منفردة

بدلا من إجراء فحصين لعرض الرسالة "You win!" أو "Sorry, you lose" ستستخدم الكلمة الأساسية else

١. تأكد من تطابق التعليمات البرمجية Program.cs مع ما يلي:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    Console.WriteLine("You rolled doubles! +2
bonus to total!");
    total += 2;
}
```

```
if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples! +6
bonus to total!");
    total += 6;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

هذه هي التعليمات البرمجية التي أكملتها في الدرس السابق.

٢. خذ دقيقة لفحص العبارات if في نهاية الملف:

```
if (total >= 15)
{
    Console.WriteLine("You win!");
}

if (total < 15)
{
    Console.WriteLine("Sorry, you lose.");
}
```

لاحظ أن كلا العبارتين if تقارن total بنفس القيمة الرقمية، هذه هي الفرصة المثالية لاستخدام عبارة else

٣. تحديث العبارتين if كما يلي:

```
if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
```

هنا، إذا كانت total >= 15 غير مطابقة للشرط، فسوف يتم الانتقال لتنفيذ كتلة التعليمات البرمجية داخل else التي تتبع الكلمة الأساسية if نظراً لأن الحصيلتين مرتبطتان، فهذا سيناريو مثالي للكلمة الأساسية else

١. يجب أن يحتوي ملف Program.cs المحدث على التعليمات التالية:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
```

```
    Console.WriteLine("You rolled doubles!  +2
bonus to total!");
    total += 2;
}

if ((roll1 == roll2) && (roll2 == roll3))
{
    Console.WriteLine("You rolled triples!  +6
bonus to total!");
    total += 6;
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
}
```

تعديل التعليمات البرمجية لمنع الحصول على مكافأة زوجية وثلاثية باستخدام التداخل أو التفريع

في الدرس السابق، رأيت أنه تم إدخال خطأ منطقي في تطبيقك، يمكنك إصلاح هذه المشكلة عن طريق إضافة عباراتك if

يسمح لك التداخل بوضع كتل تعليمات برمجية داخل كتل تعليمات برمجية أخرى، في هذه الحالة، ستدمج if and else (للتحقق المزدوج) داخل عبارة if أخرى (للتحقق الثلاثي) لمنع منح كلا العلاوات مرة واحدة.

١. قم بتعديل التعليمات البرمجية، لمطابقة التعليمات التالية:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples!
+6 bonus to total!");
        total += 6;
    }
    else
    {
        Console.WriteLine("You rolled doubles!
+2 bonus to total!");
        total += 2;
    }
}

if (total >= 15)
{
    Console.WriteLine("You win!");
}
else
{
    Console.WriteLine("Sorry, you lose.");
}
```

٢. خذ دقيقة لمراجعة العبارات المتداخلة `if`

الهدف هو إنشاء بناء داخلي `if-else` حيث تكون النتيجتان متضادتان مرتبطتان، ثم استخدام النتائج المتعارضة (إذا/صحيح وإلا/خطأ) لمنح نقاط المكافأة الثلاثية والزوجية.

لتحقيق هذا الهدف، عليك التحقق من وجود مكافئة مضاعفة في عبارة `if` الخارجية، ومن ثم التحقق من وجود مكافئة مضاعفة في عبارة `if` الداخلية. يضمن هذا النمط أنه عندما يُرجع الفحص الداخلي للمكافئة الثلاثية `false` بمعنى لا تتحقق المكافئة الثلاثية، تمنح كتلة `else` نقاط التضاعف.

بعد ذلك، سوف تقوم بكتابة "تعليمات برمجية صعبة" لنتائج اللفات الثلاث، من أجل اختبار منطق تعليماتك البرمجية.

٣. قم بإنشاء سطر فارغ أعلى السطر الذي تم فيه الإعلان عن المتغير `total` وتهيئته.
٤. لاختبار لفة مزدوجة، أدخل التعليمات البرمجية التالية:

```
roll1 = 6;
roll2 = 6;
roll3 = 5;
```

يمكنك الترميز الثابت للمتغيرات الثلاثة `roll` من اختبار التعليمات البرمجية دون الحاجة إلى تشغيل التطبيق عشرات المرات.

٥. في قائمة ملف Visual Studio Code **File** انقر فوق حفظ **Save**

٦. في قائمة استكشاف EXPLORER لفتح موجة الأوامر Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح لوحة Terminal ويظهر Terminal أن موجه الأوامر مفتوح في موقع مجلد TestProject

٧. في موجه أوامر Terminal لتشغيل التعليمات، اكتب **dotnet run** ثم اضغط على Enter

عند تشغيل تعليماتك البرمجية، يجب أن ترى:

```
Dice roll: 6 + 6 + 5 = 17
You rolled doubles!  +2 bonus to total!
You win!
```

٨. لاختبار لفة من الثلاثي، قم بتحديث متغيرات الفة، كما يلي:

```
roll1 = 6;
roll2 = 6;
roll3 = 6;
```

٩. في قائمة ملف **File** Visual Studio Code انقر فوق حفظ **Save**

١٠. في قائمة استكشاف EXPLORER لفتح موجة أوامر Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

١١. في موجه أوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

عند تشغيل التعليمات البرمجية، يجب أن ترى:

```
Dice roll: 6 + 6 + 6 = 18
You rolled triples!  +6 bonus to total!
You win!
```

استخدم عبارات if, else, and else if لمنح جائزة بدلاً من رسالة win-lost

لجعل اللعبة أكثر متعة، يمكنك تغيير اللعبة من "الفوز أو الخسارة" إلى منح جوائز وهمية لكل نتيجة، يمكنك تقديم أربع جوائز، ومع ذلك، يجب أن يفوز اللاعب بجائزة واحدة فقط:

- إذا سجل اللاعب ما يساوي ١٦ أو أكثر، فسيربح سيارة جديدة.

- إذا سجل اللاعب ما يساوي ١٠ أو أكثر، فسيربح جهاز كمبيوتر جديدًا.
- إذا سجل اللاعب ٧ بالضبط، فسيربح رحلة.
- بخلاف ذلك، سيربح اللاعب قطة صغيرة.

١. قم بتعديل التعليمات البرمجية من الخطوات السابقة إلى التالية:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;

Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples!
+6 bonus to total!");
        total += 6;
    }
    else
    {
        Console.WriteLine("You rolled doubles!
+2 bonus to total!");
        total += 2;
    }

    Console.WriteLine($"Your total including
the bonus: {total}");
}
```

```
if (total >= 16)
{
    Console.WriteLine("You win a new car!");
}
else if (total >= 10)
{
    Console.WriteLine("You win a new laptop!");
}
else if (total == 7)
{
    Console.WriteLine("You win a trip for
two!");
}
else
{
    Console.WriteLine("You win a kitten!");
}
```

٢. خذ دقيقة لمراجعة البنية المحدثة `if-elseif-else`

تسمح لك العبارة if, else if, and else بإنشاء شروط خاصة متعددة كتعبيرات منطقية، بمعنى آخر، عندما تريد حدوث نتيجة واحدة فقط، ولكن لديك العديد من الشروط والنتائج المحتملة، استخدم أكبر عدد تريده من العبارات `else if` إذا لم تنطبق أي من عبارات if and else if فسيتم تنفيذ كتلة التعليمة البرمجية else الأخيرة. else اختيارية، ولكن يجب أن تأتي أخيرًا إذا اخترت إضافتها.

٣. استخدم تقنية الترميز الثابت للمتغيرات `roll` مؤقتا لاختبار كل رسالة.

خلاصة

- يتيح لك الجمع بين عبارات `else if` اختبار شرط واحد، ثم تنفيذ إحدى نتيجتين بناءً على هذه الشرط.

- سيتم تشغيل كتلة التعليمات البرمجية if عندما يكون التعبير المنطقي true وسيتم تشغيل كتلة التعليمات البرمجية else عندما يكون التعبير المنطقي false
- يمكنك إضافة المزيد من عبارات if لتحديد شرط محتمل، ومع ذلك، فكر في استخدام العبارات if, else if, and else بدلاً من ذلك.
- استخدم عبارات else if لإنشاء شروط حصرية متعددة.
- else الأخيرة اختيارية، ولكن يجب إضافتها دائماً في النهاية.

٤ إكمال نشاط تحدي لتطبيق قواعد العمل

ستعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

التحدي: تحسين معدل تجديد الاشتراكات

لقد طُلب منك إضافة ميزة إلى برنامج شركتك، وتهدف هذه الميزة إلى تحسين معدل تجديد الاشتراكات في البرنامج، مهمتك هي عرض رسالة لتجديد الاشتراك عندما يسجل مستخدم الدخول إلى نظام البرنامج، ويتم إعلامه بانتهاء اشتراكه قريباً، ستحتاج إلى إضافة عدة عبارات القرار لإضافة منطق التفريع بشكل صحيح إلى التطبيق لتلبية المتطلبات.

إعداد بيئة الترميز الخاصة بك

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في **القائمة ملف** حدد **فتح مجلد**.
- استخدم مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects وفتحه
- في قائمة استكشاف EXPLORER حدد **Program.cs**
- في قائمة حدد **Selection** حدد اختيار الكل **Select All** ثم اضغط على مفتاح Delete لحذف كل التعليمات البرمجية الموجودة.

٢. لإنشاء التعليمات البرمجية الأولية لهذا التحدي، أدخل التعليمات التالية:

```
Random random = new Random();
int daysUntilExpiration = random.Next(12);
int discountPercentage = 0;
```

```
// Your code goes here
```

لاحظ أن هذه التعليمة البرمجية ستنشئ رقماً عشوائياً قيمته من 0 إلى 11 يتم تعيين الرقم العشوائي إلى متغير عددي يسمى `daysUntilExpiration` لديك متغير عدد صحيح آخر يسمى `discountPercentage` تمت تهيئته إلى 0

هام

في هذا التحدي، يمكنك فقط إزالة سطر التعليمات البرمجية المعلقة، بمعنى آخر، يمكنك إزالة السطر الذي يبدأ ب `//` ولكن لا يمكنك إزالة أي تعليمات برمجية أخرى، ويجب استخدام كل متغير من المتغيرات في حلك.

مراجعة قواعد العمل لهذا التحدي

١. القاعدة الأولى: يجب أن تعرض تعليماتك البرمجية رسالة واحدة فقط.

تعتمد الرسالة التي تعرضها على القواعد الخمس الأخرى، بالنسبة للقواعد من ٢ إلى ٦ تكون القواعد بالترتيب.

٢. القاعدة الثانية: إذا كانت صلاحية اشتراك المستخدم ستنتهي في غضون عشرة أيام أو أقل، فأعرض الرسالة التالية:

```
Your subscription will expire soon. Renew now!
```

٣. القاعدة الثالثة: إذا كانت صلاحية اشتراك المستخدم ستنتهي في غضون خمسة أيام أو أقل، فأعرض الرسائل:

```
Your subscription expires in 5 days.
Renew now and save 10%!
```

ملاحظة

تأكد من استبدال الرقم ٥ المعروض في الرسالة أعلاه بالقيمة المخزنة في المتغير `daysUntilExpiration` عند إنشاء إخراج الرسالة.

٤. القاعدة الرابعة: إذا كانت صلاحية اشتراك المستخدم ستنتهي في غضون يوم واحد، فأعرض الرسائل:

```
Your subscription expires within a day!
Renew now and save 20%!
```

٥. القاعدة الخامسة: إذا انتهت صلاحية اشتراك المستخدم، فعرض الرسالة:

```
Your subscription has expired.
```

٦. القاعدة السادسة: إذا لم تنتهي صلاحية اشتراك المستخدم خلال عشرة أيام أو أقل، فلا تعرض أي شيء.

تنفيذ التعليمات البرمجية للحل باستخدام عبارات if

يجب أن يستخدم الحل عبارات if منفصلة وعبارات if-else لتنفيذ قواعد العمل، يمكن أن تتضمن عبارة if-else عدة أجزاء أخرى.

١. إنشاء عبارة if-else تعرض رسالة حول موعد انتهاء صلاحية الاشتراك.

تلميح: استخدم else if لضمان حساب كل قواعد انتهاء صلاحية.

٢. إنشاء عبارة if منفصلة تعرض عرض خصم.

تشير قواعد العمل إلى متى يجب تقديم خصم.

سواء واجهتك مشكلة واحتجت إلى إلقاء نظرة على الحل أو كنت أنهيت الحل بنجاح، فتابع لعرض حل هذا التحدي.

٥ مراجعة الحل لتطبيق نشاط تحدي قواعد العمل

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الوحدة السابقة.

```
Random random = new Random();

int daysUntilExpiration = random.Next(12);
int discountPercentage = 0;

if (daysUntilExpiration == 0)
{
    Console.WriteLine("Your subscription has
expired.");
}
else if (daysUntilExpiration == 1)
{
    Console.WriteLine("Your subscription
expires within a day!");
    discountPercentage = 20;
}
else if (daysUntilExpiration <= 5)
{
    Console.WriteLine($"Your subscription
expires in {daysUntilExpiration} days.");
    discountPercentage = 10;
}
else if (daysUntilExpiration <= 10)
{
    Console.WriteLine("Your subscription will
expire soon. Renew now!");
}

if (discountPercentage > 0)
{
    Console.WriteLine($"Renew now and save
{discountPercentage}%.");
}
```

إن هذه التعليمة البرمجية هي مجرد حل واحد محتمل، لأن كثيراً من الحلول تعتمد على الطريقة التي قررت بها تنفيذ المنطق، طالما حصلت على النتائج الصحيحة، حسب قواعد التحدي، واستخدمت اثنتين من عبارات if فقد أديت عملاً رائعاً

إذا نجحت، فتهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٦ أختبر معلوماتك

١- بالنظر إلى `int x = 5` أي من التعبيرات المنطقية التالية صالح ويتم تقييمه إلى صحيح؟

- X < 5
- X > 5
- X == 5

٢- أي من الخيارات أدناه ليس عامل تشغيل صالحا في C#؟

- %%
- &&
- ||

٣- تصف خيارات الإجابة التالية بناء العبارة if أي من الأوصاف صحيح؟

- التعبير المنطقي الذي يتم تقييمه بواسطة عبارة if اختياري
- لا يمكن وضع عبارة else قبل عبارة else if
- العبارة else مطلوبة عندما تتضمن if عبارة if else

راجع إجابتك

١ x == 5

صحيح X تساوي ٥ فإن استخدام عامل تشغيل المساواة == في التعبير، يتم تقييمه إلى صحيح

٢ %%

صحيح رمز %% علامة النسبة المئوية المزدوجة ليس عامل تشغيل C# صالحاً

٣ لا يمكن وضع عبارة else قبل عبارة else if

صحيح لا يمكن وضع else قبل عبارة else if

٧ الملخص

كان هدفك في هذه الوحدة هو إضافة منطق تفريع إلى التعليمات البرمجية الخاصة بك باستخدام عبارات القرار if

باستخدام عبارات if, else if, and else قمت بتقييم التعبيرات المنطقية لإنشاء مسارات تنفيذ بديلة من خلال تطبيقك، وهذا يسمح لك بتنفيذ بعض التعليمات البرمجية، وتجاهل تعليمات أخرى، اعتماداً على بعض الشروط.

بدون بيانات القرار، ستفتقر تطبيقاتك إلى القدرة على إتمام الأعمال المشتركة، والألعاب، والمهام العلمية المطلوبة في التطبيقات الحديثة.

سوف تعتمد على التقنيات التي تعلمناها في هذه الوحدة، في كل تطبيق ننشئه تقريباً.

الوحدة الرابعة

التخزين والتكرار من خلال تتابع البيانات، باستخدام المصفوفات arrays وجملة foreach

العمل مع تتابع البيانات ذات الصلة sequences of related data في هياكل البيانات المعروفة باسم المصفوفات، ثم تعلم كيفية تكرار كل عنصر في التسلسل.

الأهداف التعليمية:

خلال هذه الوحدة، سوف تتمكن مما يلي:

- إنشاء مصفوفة array جديدة وتهيئتها.
- تعيين القيم والحصول عليها من المصفوفات.
- التكرار من خلال كل عنصر من عناصر المصفوفة، باستخدام عبارة foreach

محتويات الوحدة:

١- مقدمة

٢- بدء استخدام أساسيات المصفوفات arrays

٣- تنفيذ عبارة foreach

٤- إكمال نشاط تحدي التكرار المتداخل nested iteration وعبارات التحديد selection statements

٥- مراجعة الحل لنشاط تحدي التكرار المتداخل وعبارات التحديد

٦- اختبر معلوماتك

٧- الملخص

١ المقدمة

تسمح لك المصفوفات `arrays` بتخزين صفيف من القيم في هيكل بيانات فردية، بعبارة أخرى، تخيل متغيراً واحداً يمكنه الاحتفاظ بالعديد من القيم، بمجرد أن يكون لديك متغير واحد يخزن جميع القيم، يمكنك فرز القيم، وعكس ترتيبها، وتكرار كل قيمة، وفحصها على حدة، وما إلى ذلك.

لنفترض أنك تعمل في قسم الأمان في شركة تطابق البائعين عبر الإنترنت، مع المعلنين الموثقين، وطلب منك كتابة التعليمات البرمجية C# لمعرفة الطلبات الواردة التي سيتم تكرارها من خلال معرفات الطلبات، تحتاج إلى فحص كل معرف طلب، لتحديد الطلبات التي قد تكون احتيالية، ستحتاج إلى تنفيذ مصفوفات لإنجاز مهمة البرمجة هذه.

في هذه الوحدة، ستقوم بإنشاء المصفوفات arrays وتشغيلها، ستعين القيم وتستردها من العناصر الموجودة في المصفوفة، لتصل إلى كل عنصر باستخدام فهرسها، ستنشئ منطقاً تكرارياً يسمح لك بالعمل مع كل عنصر في مصفوفة ما.

في نهاية الوحدة، ستكون قد استخدمت البنية الأولى الخاصة بك، للاحتفاظ بقيم بيانات متعددة، لاحقاً، وفي دروس أخرى، ستتعلم كيفية الفرز والتصفية والاستعلام والتجميع وتنفيذ عمليات أخرى في بياناتك.

هام

تتضمن هذه الوحدة أنشطة الترميز التي تتطلب Visual Studio Code سوف تحتاج إلى الوصول إلى بيئة التطوير، وتكوينها.

٢ بدء استخدام أساسيات المصفوفات arrays

يمكن استخدام المصفوفات arrays لتخزين قيم متعددة، من نفس النوع في متغير واحد، ترتبط القيم المخزنة في مصفوفة بشكل عام، على سبيل المثال، يمكن تخزين قائمة بأسماء الطلاب في مصفوفة نصية تسمى students

يركز عملك في قسم الأمان على العثور على نمط للطلبات الاحتيالية، تريد أن تقوم تعليماتك البرمجية بمراجعة طلبات العملاء السابقة، وتحديد العلامات المرتبطة بطلبات احتيالية، تأمل شركتك في إمكانية استخدام العلامات لتحديد أوامر الشراء الاحتيالية المحتملة قبل معالجتها، نظراً لأنك لا تعرف مسبقاً عدد الطلبات التي تحتاج إلى مراجعتها، فلا يمكنك إنشاء متغيرات فردية للاحتفاظ بكل معرف طلب، كيف يمكنك إنشاء بنية بيانات للاحتفاظ بقيم متعددة ذات صلة؟

في هذا التمرين، يمكنك استخدام المصفوفات، لتخزين وتحليل تسلسل معرفات الطلبات.

ما هي المصفوفة array

المصفوفة هي صفيف أو مجموعة من عناصر البيانات الفردية، يمكن الوصول إليها من خلال اسم متغير واحد، يمكنك استخدام فهرس رقمي يبدأ بالصفر للوصول إلى كل عنصر من عناصر المصفوفة، كما ترى، تسمح لك المصفوفات بجمع بيانات مشابهة، تشترك في غرض مشترك أو خصائص مشتركة في هيكل بيانات واحد لتسهيل المعالجة.

الإعلان عن المصفوفات والوصول إلى عناصر المصفوفة

المصفوفة هي نوع خاص من المتغيرات يمكن أن تحتوي على قيم متعددة من نفس نوع البيانات، يختلف بناء الإعلان قليلاً، لأنه يجب عليك تحديد كل من نوع البيانات، وحجم المصفوفة.

إعداد بيئة الترميز

تتضمن هذه الوحدة أنشطة ترشدك خلال عملية إنشاء نموذج التعليمات البرمجية وتشغيله، يتم تشجيعك على إكمال هذه الأنشطة باستخدام Visual Studio Code كبيئة تطوير.

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** حدد فتح مجلد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدامه بدلا من ذلك الموقع لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. في موجه أوامر Terminal لإنشاء تطبيق وحدة تحكم جديدة في مجلد محدد، اكتب:

```
dotnet new console -o ./CsharpProjects/TestProject
```

ثم اضغط على Enter

يستخدم أمر .NET CLI. قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد، ينشئ الأمر مجلدات CsharpProjects, TestProject نيابة عنك، ويستخدم TestProject كاسم لملف `.csproj` أو امتداد له.

٧. في قائمة استكشاف EXPLORER قم بتوسيع المجلد CsharpProjects

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# يسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في قائمة المحرر، حدد **Program.cs**

٩. حذف أسطر التعليمات البرمجية الموجودة.

يمكنك استخدام مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق قائمة Terminal

الإعلان عن مصفوفة جديدة array

١. لتعريف مصفوفة جديدة نصية، تحتوي على ثلاثة عناصر، أدخل التعليمات البرمجية التالية:

```
string[] fraudulentOrderIDs = new string[3];
```

٣. خذ دقيقة لفحص التعليمات البرمجية.

يقوم العامل new بإنشاء نسخة كائن لمصفوفة في ذاكرة الكمبيوتر، تحتوي على ثلاث قيم نصية. لمزيد من المعلومات حول الكلمة الأساسية new راجع الوحدة السابقة **أساليب الاستدعاء من مكتبة فئات .NET.**

لاحظ أن المجموعة الأولى من الأقواس المربعة **[]** تخبر المحول البرمجي أن المتغير المسمى fraudulentOrderIDs عبارة عن مصفوفة، ولكن المجموعة الثانية من الأقواس المربعة **[3]** تشير إلى عدد العناصر التي يمكن للمصفوفة الاحتفاظ بها.

يوضح هذا المثال كيفية تعريف مصفوفة نصية، ومع ذلك، يمكنك إنشاء مصفوفة من كل أنواع البيانات، مثل الرقمية int والمنطقية bool وأنواع

البيانات الأكثر تعقيداً، مثل الفئات، يستخدم هذا المثال بساطة النصوص لتقليل عدد الأفكار الجديدة التي تستوعبها عندما تبدأ.

تعيين قيم لعناصر مصفوفة array

عند هذه النقطة، قمت بالإعلان عن مصفوفة نصية، ولكن كل عنصر من عناصر المصفوفة فارغ، للوصول إلى عنصر مصفوفة، يمكنك استخدام فهرس رقمي يبدأ بالصفر، داخل الأقواس المربعة `[0]` يمكنك تعيين قيمة لعنصر مصفوفة باستخدام `=` كما لو كان متغيراً عادياً.

١. لتعيين قيم إلى مصفوفتك `fraudulentOrderIDs` قم بتحديث تعليماتك البرمجية كما يلي:

```
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
```

٢. خذ دقيقة لفحص التعليمات البرمجية.

لاحظ أنك تستخدم اسم المصفوفة للوصول إلى عناصرها، يتم الوصول إلى كل عنصر بشكل منفرد، عن طريق تحديد رقم الفهرس داخل الأقواس المربعة.

نظراً لأن مصفوفتك تم تعريفها كنص، يجب أن تكون القيم التي تقوم بتعيينها أيضاً نصية، في هذا السيناريو، تقوم بتعيين معرفات الطلبات لعناصر المصفوفة.

محاولة استخدام فهرس خارج حدود المصفوفة array

قد لا يبدو بديهياً في البداية، ولكن من المهم أن تتذكر أنك تعلن عن عدد العناصر في المصفوفة، ومع ذلك، يمكنك الوصول إلى كل عنصر من

عناصر المصفوفة بدءاً من الصفر [0] لذلك للوصول إلى العنصر الثاني في الصفيف، استخدام الفهرس [1]

من الشائع نسيان المبتدئون أن المصفوفات تستند إلى الصفر، ويحاولون الوصول إلى عنصر زائد غير موجود، إذا ارتكبت هذا الخطأ، يحدث استثناء وقت التشغيل، لإعلامك أنك حاولت الوصول إلى عنصر خارج حدود المصفوفة.

لقطع تطبيقك عن قصد، حاول الوصول إلى عنصر رابع من مصفوفتك باستخدام قيمة الفهرس 3

١. في أسفل ملف التعليمات البرمجية، أدخل سطر التعليمات التالي:

```
fraudulentOrderIDs[3] = "D000";
```

٢. تأكد من أن التعليمات البرمجية تطابق هذا المثال:

```
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
fraudulentOrderIDs[3] = "D000";
```

٣. في قائمة ملف file حدد حفظ **Save**

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح قائمة Terminal وأن يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٥. في موجه الأوامر Terminal لبناء التعليمات البرمجية، اكتب dotnet build ثم اضغط على Enter

يجب أن تشاهد الرسالة التالية:

```
Build succeeded.
    0 Warning(s)
    0 Error(s)
```

٦. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب dotnet run ثم اضغط على Enter

عند تشغيل التطبيق، تحصل على رسالة الخطأ التالية:

```
Unhandled exception. System.IndexOutOfRangeException:
Index was outside the bounds of the array.
at Program.<Main>$(String[] args) in
C:\Users\someuser\Desktop\CsharpProjects\TestProject\Progra
m.cs:line 6
```

لاحظ الأجزاء التالية من الخطأ:

- رسالة الخطأ Error message: `System.IndexOutOfRangeException: Index was outside the bounds of the array.`
- موقع الخطأ Error location: `Program.cs:line 6`

٧. تعليق السطر الذي أنشأ الخطأ وقت التشغيل.

```
// fraudulentOrderIDs[3] = "D000";
```

لقد رأيت كيفية تعيين قيمة لعنصر مصفوفة، انظر الآن إلى كيفية الوصول إلى قيمة تم تخزينها في عنصر مصفوفة.

استرداد القيم من عناصر مصفوفة array

طريقة الوصول إلى قيمة عنصر مصفوفة، هي نفس طريقة تعيين قيمة لعنصر المصفوفة، ما عليك سوى تحديد فهرس العنصر الذي تريد استرداد قيمته.

١. لكتابة قيمة كل معرف طلب احتيالي، قم بتحديث تعليماتك البرمجية على النحو التالي:

```
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";

Console.Write($"First:{fraudulentOrderIDs[0]}\n");
Console.Write($"Second:{fraudulentOrderIDs[1]}\n");
Console.Write($"Third:{fraudulentOrderIDs[2]}\n");
```

٢. في قائمة ملف حدد حفظ

٣. في قائمة استكشاف لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٤. في موجه الأوامر Terminal اكتب `dotnet run` ثم اضغط على Enter

يجب أن تشاهد الرسالة التالية:

```
First: A123
Second: B456
Third: C789
```

إعادة تعيين قيمة مصفوفة array

عناصر المصفوفة هي تماماً مثل أي قيمة متغيرة أخرى، يمكنك تعيين قيمة واستردادها، وإعادة تعيينها، لكل عنصر من عناصر المصفوفة.

١. في نهاية ملف التعليمات البرمجية، لإعادة تعيين ثم طباعة قيمة عنصر المصفوفة الأول، أدخل التعليمات التالية:

```
fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First:
{fraudulentOrderIDs[0]}");
```

٢. تأكد من أن تعليماتك البرمجية تطابق المثال التالي:

```
string[] fraudulentOrderIDs = new string[3];

fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");
```

٣. في قائمة ملف حدد حفظ

٤. في قائمة استكشاف لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٥. في موجه الأوامر Terminal اكتب `dotnet run` ثم اضغط على Enter

يجب أن تشاهد الرسالة التالية:

```
First: A123
Second: B456
Third: C789
Reassign First: F000
```

تهيئة مصفوفة array

يمكنك تهيئة مصفوفة أثناء الإعلان تماماً كما تفعل مع متغير عادي. ومع ذلك، لتهيئة عناصر المصفوفة، يمكنك استخدام بناء جملة خاص يضم أقواس متعرجة.

١. تعليق الأسطر حيث تقوم بتعريف المتغير fraudulentOrderIDs

يمكنك استخدام تعليق متعدد الأسطر (/* ... */) للتعليق إعلان fraudulentOrderIDs والأسطر المستخدمة لتعيين قيم لعناصر المصفوفة.

٢. للإعلان عن تهيئة مصفوفة، وتخزين القيم في عبارة واحدة، أدخل التعليمات البرمجية التالية:

```
string[] fraudulentOrderIDs = { "A123", "B456", "C789" };
```

٣. تأكد من أن تعليماتك البرمجية تطابق المثال التالي:

```
string[] fraudulentOrderIDs = { "A123", "B456", "C789" };

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First: {fraudulentOrderIDs[0]}");
```

٤. خذ دقيقة لفحص بيان الإعلان.

لاحظ أن بناء الجملة مضغوط، وسهل القراءة، عند تشغيل التطبيق، يجب ألا يكون هناك أي تغيير في الإخراج.

٥. في قائمة ملف حدد حفظ

٦. في قائمة استكشاف لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٧. في موجه الأوامر Terminal اكتب dotnet run ثم اضغط على Enter

يجب أن تشاهد نفس الرسالة كما كان من قبل:

```
First: A123
Second: B456
Third: C789
Reassign First: F000
```

استخدام الخاصية الطول Length لمصفوفة

بناءً على كيفية إنشاء المصفوفة، قد لا تعرف مسبقاً عدد العناصر التي تحتوي عليها المصفوفة، لتحديد حجم مصفوفة، يمكنك استخدام خاصية Length انتبه، خاصية الطول Length للمصفوفة ليست مبنية على الصفر.

١. في نهاية ملف التعليمات البرمجية، للإبلاغ عن عدد الطلبات الاحتيالية، أدخل التعليمات التالية:

```
Console.WriteLine($"There are {fraudulentOrderIDs.Length} fraudulent orders to process.");
```

تستخدم هذه التعليمة البرمجية خاصية الطول للمصفوفة Length وهي عدد صحيح، لإرجاع عدد العناصر في مصفوفتك fraudulentOrderIDs

٢. تأكد من أن تعليماتك البرمجية تطابق هذا المثال:

```
/*
string[] fraudulentOrderIDs = new string[3];
```

```
fraudulentOrderIDs[0] = "A123";
fraudulentOrderIDs[1] = "B456";
fraudulentOrderIDs[2] = "C789";
// fraudulentOrderIDs[3] = "D000";
*/

string[] fraudulentOrderIDs = { "A123", "B456", "C789" };

Console.WriteLine($"First: {fraudulentOrderIDs[0]}");
Console.WriteLine($"Second: {fraudulentOrderIDs[1]}");
Console.WriteLine($"Third: {fraudulentOrderIDs[2]}");

fraudulentOrderIDs[0] = "F000";

Console.WriteLine($"Reassign First:
{fraudulentOrderIDs[0]}");

Console.WriteLine($"There are {fraudulentOrderIDs.Length}
fraudulent orders to process.");
```

احفظ التغييرات في ملف **Program.cs** ثم قم بتشغيل التطبيق.

ينبغي أن تشاهد الإخراج التالي:

```
First: A123
Second: B456
Third: C789
Reassign First: F000
There are 3 fraudulent orders to process.
```

خلاصة

فيما يلي أهم الأشياء التي يجب تذكرها عند العمل مع المصفوفات arrays

- المصفوفة عبارة عن متغير خاص يحتوي على سلسلة من عناصر بيانات ذات صلة.
- يجب عليك حفظ التنسيق الأساسي لإعلان متغير مصفوفة.

```
string[] firstArrays = new string[3];
string[] firstArrays = { "A13", "B46", "C89" };
```

- قم بالوصول إلى كل عنصر من عناصر المصفوفة لتعيين قيمه، أو الحصول عليها باستخدام فهرس يبدأ بالصفر داخل أقواس مربعة [0]
- إذا حاولت الوصول إلى فهرس خارج حدود المصفوفة، فستحصل على استثناء خطأ وقت التشغيل.
- تمنحك الخاصية الطول Length طريقة برمجية لتحديد عدد العناصر في المصفوفة.

اختبر معلوماتك

١- ما هي المصفوفة؟

- متغير نصي
- تسلسل من عناصر البيانات الفردية التي يمكن الوصول إليها من خلال اسم متغير واحد
- مكتبة فئات .NET.

٢- أي من هذه الأمثلة يعد مثالاً صحيحاً لإنشاء مصفوفة وتهيئتها؟

- `string[] myarray = new string[3]; myarray = "test1"; myarray = "test2"; myarray = "test3";`
- `string[] myarray = string[3]; myarray[0] = test1; myarray[1] = test2; myarray[2] = test3;`
- `int[] myarray = new int[3]; myarray[0] = 1; myarray[1] = 2; myarray[2] =3;`

راجع إجابتك

١ تسلسل من عناصر البيانات الفردية التي يمكن الوصول إليها من خلال اسم متغير واحد.

صحيح بنية بيانات مصفوفة باسم واحد ومواقع بيانات متعددة.

٢

```
int[] myarray = new int[3]; myarray[0] = 1;
myarray[1] = 2; myarray[2] = 3;
```

صحيح يتم تعريف المصفوفة myarray بشكل صحيح وتحدد عبارات التعليمات البرمجية المستخدمة لتعيين قيم لعناصر المصفوفة مواقع فهرس المصفوفة

٣ تنفيذ عبارة foreach

لنفترض أنك تعمل في شركة تصنيع، تحتاج الشركة منك إكمال جرد المخزن، لتحديد عدد المنتجات الجاهزة للشحن، بالإضافة إلى العدد الإجمالي للمنتجات، تحتاج إلى الإبلاغ عن عدد المنتجات النهائي الذي يستوعبه المخزن، إلى جانب العدد الإجمالي المتوفر حالياً، سيتم استخدام هذه المعلومات لإنشاء سجل تدقيق، حتى تتمكن من التحقق مرات متعددة من عملك وتحديد الانكماش.

التكرار من خلال مصفوفة باستخدام foreach

توفر العبارة `foreach` طريقة بسيطة ونظيفة للتكرار، من خلال عناصر المصفوفة، تعالج العبارة `foreach` عناصر المصفوفة في زيادة ترتيب الفهرس، بدءاً من الفهرس `0` وتنتهي بطول الفهرس `-1` تستخدم `foreach` متغيراً مؤقتاً للاحتفاظ بقيمة عنصر المصفوفة الخاص بالتكرار الحالي. سيقوم كل تكرار بتشغيل كتلة التعليمات البرمجية الموجودة أسفل الإعلان عن `foreach`

إليك مثال بسيط:

```
string[] names = {"Rawan ", "Ahmed", "Bilal"};
foreach (string name in names)
{
    Console.WriteLine(name);
}
```

أسفل الكلمة الرئيسية `foreach` ستقوم كتلة التعليمة البرمجية التي تحوي `Console.WriteLine(name);` بعرض مرة واحدة كل عنصر من عناصر المصفوفة `names` نظراً لأن وقت تشغيل .NET. يتكرر عبر كل عنصر من عناصر المصفوفة، يتم تعيين القيمة المخزنة في العنصر الحالي لمصفوفة `names` إلى المتغير المؤقت `name` لسهولة الوصول إليه داخل كتلة التعليمة البرمجية.

إذا قمت بتشغيل التعليمات البرمجية، فسترى النتيجة التالية.

```
Rawan
Ahmed
Bilal
```

استخدم العبارة foreach لإنشاء مجموع جميع العناصر الموجودة في كل سلة من مخزنك.

إنشاء مصفوفة int وتهيئتها

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في **القائمة ملف**، حدد **فتح مجلد**
- استخدم قائمة فتح مجلد، للانتقال إلى المجلد CsharpProjects ثم فتحه
- في قائمة استكشاف EXPLORER حدد **Program.cs**
- في قائمة حدد **Selection** حدد **Select All** ثم اضغط على مفتاح Delete

٢. لإنشاء مصفوفة من النوع int الذي يخزن عدد المنتجات النهائية في كل سلة، أدخل التعليمات البرمجية التالية:

```
int[] inventory = { 200, 450, 700, 175, 250 };
```

إضافة عبارة foreach للتكرار من خلال المصفوفة array

١. لإنشاء عبارة تكرار foreach من خلال كل عنصر من عناصر المصفوفة inventory أدخل التعليمات التالية:

```
foreach (int items in inventory)
{

}
```

لاحظ أن العبارة foreach تعين مؤقتاً قيمة عنصر المصفوفة الحالي إلى متغير int يسمى items

٢. تأكد من أن تعليماتك البرمجية تطابق ما يلي:

```
int[] inventory = { 200, 450, 700, 175, 250 };

foreach (int items in inventory)
{

}
```

إضافة متغير لجمع قيمة كل عنصر في المصفوفة

١. ضع المؤشر على السطر الفارغ أعلى عبارة foreach

٢. للإعلان عن متغير جديد، يمثل مجموع جميع المنتجات النهائية في المخزن، أدخل التعليمات التالية:

```
int sum = 0;
```

تأكد من إعلان المتغير خارج عبارة foreach

٣. ضع المؤشر داخل الكتلة البرمجية للعبارة foreach لإضافة القيمة الحالية المخزنة items إلى المتغير sum أدخل التعليمات التالية:

```
sum += items;
```

٤. تأكد من تطابق تعليماتك البرمجية مع ما يلي:

```
int[] inventory = { 200, 450, 700, 175, 250 };
int sum = 0;
foreach (int items in inventory)
{
    sum += items;
}
```

عرض القيمة النهائية للمجموع

١. إنشاء سطر فارغ أسفل الكتلة البرمجية للعبارة foreach

٢. للإبلاغ عن المجموع النهائي للعناصر في المخزنة، أدخل التعليمات التالية:

```
Console.WriteLine($"We have {sum} items in inventory.");
```

٣. تأكد من تطابق تعليماتك البرمجية مع ما يلي:

```
int[] inventory = { 200, 450, 700, 175, 250 };
int sum = 0;
foreach (int items in inventory)
{
    sum += items;
}

Console.WriteLine($"We have {sum} items in
inventory.");
```

٤. في قائمة ملف **File** Visual Studio Code انقر فوق حفظ **Save**

٥. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٦. في موجه الأوامر Terminal اكتب **dotnet run** ثم اضغط على Enter

```
We have 1775 items in inventory.
```

إنشاء متغير للاحتفاظ برقم المخزون الحالي وعرض الإجمالي الفعلي

للوفاء بالمتطلبات النهائية لمشروع إعداد تقارير عن المخزون، ستحتاج إلى إنشاء متغير يحتفظ بالنتيجة الحالية لعبارة foreach حتى تتمكن من عرض

سلة الأصناف، وعدد العناصر النهائية في تلك السلة، إلى جانب الإجمالي الحالي لجميع عناصر السلال التي تم حسابها حتى الآن.

١. إنشاء سطر فارغ أعلى عبارة foreach

٢. للإعلان عن المتغير int المسمى bin الذي تمت تهيئته إلى 0 أدخل التعليمات التالية:

```
int bin = 0;
```

يستخدم bin لتخزين عدد السلال التي تتم معالجة مخزونها الحالي.

٣. داخل الكتلة البرمجية foreach لزيادة bin في كل مرة يتم فيها تنفيذ كتلة التعليمات البرمجية، أدخل التعليمات التالية:

```
bin++;
```

لاحظ أنك تستخدم عامل التشغيل ++ لزيادة قيمة المتغير بمقدار ١ هذا اختصار ل bin = bin + 1

٤. للإبلاغ عن رقم السلة وعدد المنتجات النهائية في السلة، والإجمالي الحالي للمنتجات، أدخل التعليمات التالية داخل الكتلة البرمجية foreach بعد bin++;

```
Console.WriteLine($"Bin {bin} = {items} items
(Running total: {sum})");
```

تستخدم هذه التعليمة البرمجية عداد المتغير bin وعناصر متغير items الخاصة بعبارة foreach والمتغير sum للإبلاغ عن الحالة الحالية للمخزون، في رسالة منسقة بشكل جيد.

٥. تأكد من أن التعليمات البرمجية الخاصة بك تطابق ما يلي:

```
int[] inventory = { 200, 450, 700, 175, 250 };
int sum = 0;
int bin = 0;
foreach (int items in inventory)
{
    sum += items;
```

```
    bin++;
    Console.WriteLine($"Bin {bin} = {items}
items (Running total: {sum})");
}
Console.WriteLine($"We have {sum} items in
inventory.");
```

٦. احفظ التغييرات في ملف Program.cs ثم قم بتشغيل التطبيق.

ينبغي أن تشاهد الإخراج التالي:

```
Bin 1 = 200 items (Running total: 200)
Bin 2 = 450 items (Running total: 650)
Bin 3 = 700 items (Running total: 1350)
Bin 4 = 175 items (Running total: 1525)
Bin 5 = 250 items (Running total: 1775)
We have 1775 items in inventory.
```

خلاصة

فيما يلي بعض الأشياء التي يجب تذكرها حول عبارات `foreach` والقيم الإضافية التي تعلمتها في هذا الدرس:

- استخدم جملة `foreach` للتكرار خلال كل عنصر داخل المصفوفة، وتنفيذ كتلتها البرمجية مرة واحدة، لكل عنصر في المصفوفة.
- تُعيّن عبارة `foreach` قيمة العنصر الحالي في المصفوفة لمتغير مؤقت، لاستخدامه داخل محتوى كتلتها البرمجية.
- استخدم عامل الزيادة `++` لإضافة 1 إلى قيمة المتغير الحالية.

٤ إكمال نشاط تحدي تكرار متداخل nested iteration وعبارات التحديد selection statements

تعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

تحدي أمر احتيالي

قمت في هذه الوحدة بكتابة تعليمات برمجية من شأنها تخزين معرفات الطلبات، التي تنتمي إلى طلبات يحتمل أن تكون احتيالية، هدفك هو العثور على طلبات احتيالية سريعاً، والإبلاغ عنها، لتحليل أعمق.

تحدي التعليمات البرمجية ـ الإبلاغ عن معرفات الطلبات التي تحتاج إلى مزيد من التحليل

لقد وجد فريقك نمطاً، الطلبات التي تبدأ بالحرف "B" احتيالية بمعدل ٢٥ ضعف المعدل العادي، يمكنك كتابة تعليمات جديدة تقوم بإخراج الطلبات الجديدة، حيث يبدأ معرف الطلب بالحرف "B" وسيتم استخدام هذا من قبل فريق الاحتيال لمزيد من التحقيق.

استخدم الخطوات التالية لإكمال هذا التحدي.

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في القائمة ملف file حدد فتح مجلد open folder
- استخدم قائمة فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه
- في قائمة استكشاف EXPLORER حدد Program.cs.

٢. قم بتعريف مصفوفة وتهيئتها لتحوي العناصر التالية:

```
B123
C234
A345
C15
B177
G3003
C235
B179
```

تمثل هذه القيم بيانات معرف الطلب الاحتيالية التي يستخدمها تطبيقك.

٣. إنشاء عبارة foreach للتكرار خلال كل عنصر من عناصر المصفوفة، والإبلاغ عن معرفات الطلب التي تبدأ بالحرف "B"

تحتاج إلى تقييم كل عنصر من عناصر المصفوفة، والإبلاغ عن معرفات الطلبات التي يحتمل أن تكون احتيالية، عن طريق الكشف عن الطلبات التي تبدأ بالحرف "B" ولتحديد ما إذا كان العنصر يبدأ بالحرف "B" أو لا، استخدم الأسلوب String.StartsWith() فيما يلي مثال بسيط على كيفية استخدام الأسلوب String.StartsWith() الذي يمكنك تكييفه لتعليماتك البرمجية.

```
string name = "Bob";
if (name.StartsWith("B"))
{
    Console.WriteLine("The name starts with
'B'!");
}
```

يجب أن يتطابق الإخراج مع ما يلي:

```
B123
B177
B179
```

تلميح

إليك هذا التلميح: أثناء قيامك بالتكرار خلال كل عنصر في مصفوفتك، ستحتاج إلى عبارة if تتطلب عبارة if استخدام أسلوب في فئة string لتحديد ما إذا كانت الكلمة تبدأ بحرف معين، إذا لم تكن متأكداً من كيفية استخدام عبارة if فالرجاء الاطلاع على الوحدة "إضافة منطق القرار إلى تعليمتك البرمجية باستخدام جملة if-elseif-else بلغة #C"

٥ مراجعة الحل لنشاط تحدي عبارات التحديد والتكرار المتداخل

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
string[] orderIDs = { "B123", "C234", "A345",
"C15", "B177", "G3003", "C235", "B179" };

foreach (string orderID in orderIDs)
{
    if (orderID.StartsWith("B"))
    {
        Console.WriteLine(orderID);
    }
}
```

إن هذه التعليمة البرمجية هي مجرد حل واحد محتمل، لأن كثيراً من الحلول تعتمد على الطريقة التي قررت بها تنفيذ المنطق، طالما حصلت على النتائج الصحيحة، وفقاً لقواعد التحدي، واستخدمت مصفوفة وعبارة foreach فحينئذٍ يكون أداؤك رائعاً.

نهنئك في حال نجاحك! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٦ اختبر معلوماتك

١- أي مما يلي يمكن استخدامه للوصول إلى العنصر الثالث في المصفوفة؟

- myArray[3]
- myArray[2]
- myArray{3}

١- أي من الخيارات التالية يصف الطريقة الصحيحة لإنشاء مصفوفة جديدة عدد صحيح بثلاثة عناصر؟

- int[] myArray = new int[3];
- int[] myArray = new int[2];
- int myArray = new int[3];

١- أي من العبارات foreach التالية صحيحة من الناحية التركيبية؟

- `foreach (int value in values)`
- `foreach (int value of values)`
- `foreach (int value with values)`

راجع إجابتك

١

```
myArray[2]
```

صحيح! الفهارس مستندة إلى الصفر، لذلك يقوم فهرس من اثنين بإرجاع العنصر الثالث في مصفوفة

٢

```
int[] myArray = new int[3];
```

صحيح هذه هي الطريقة الصحيحة لإنشاء مصفوفة بثلاثة عناصر

٣

```
foreach (int value in values)
```

صحيح يتم تعريف العبارة foreach باستخدام بناء جملة صالح

٧ الملخص

كان هدفك في هذه الوحدة هو العمل مع سلسلة من معرفات الطلبات برمجياً، لتحديد الطلبات التي يحتمل أن تكون احتيالية، قمت بإنشاء مصفوفة من معرفات الطلبات، ثم تكرارها من خلال كل عنصر من عناصر التسلسل، بحثاً عن سمة احتيال شائعة.

مكنتك المصفوفات من تخزين كل معرف طلب، كعنصر تابع لمتغير واحد، تعاملت مع عنصراً معيناً من المصفوفة باستخدام فهرس، قيمة رقمية قائمة على الصفر، كنت قادراً على استرداد وتعيين قيمة كل عنصر، لقد كررت من خلال عناصر المصفوفة لفحص قيمة كل عنصر أو إخراجها.

تخيل مدى صعوبة التعامل مع البيانات ذات الصلة في التعليمات البرمجية، إذا لم يكن لديك بنية مثل المصفوفة؟ ستحتاج إلى معرفة عدد عناصر البيانات التي تتوقع العمل معها مسبقاً، وتحديد متغير منفصل لكل قيمة، هذا من شأنه أن يخلق حلاً هشاً.

يمكن تغيير حجم المصفوفات عند إنشائها، وفقاً لكمية البيانات التي تحتاج إلى العمل معها، عندما تصبح أكثر دراية بالمصفوفات (وهياكل البيانات المماثلة) غالباً ما تستخدمها في تطبيقاتك لمعالجة البيانات وإدارتها.

الوحدة الخامسة

تنسيق التعليمات البرمجية، إنشاء اصطلاحات، ومسافات بيضاء، وتعليقات في C#

كتابة تعليمات برمجية، تعد أسهل في القراءة، والتحديث والدعم؛ باستخدام إصلاحات تسمية، وتعليقات، ومسافات بيضاء.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- اختيار أسماء وصفية، للمتغيرات تصف الغرض منها وهدفها.
- استخدام تعليقات التعليمة البرمجية؛ كي يتجاهلها المحول البرمجي بشكل مؤقت.
- استخدام تعليقات التعليمة البرمجية لوصف متطلبات البرنامج، أو الغرض من كتابة التعليمة البرمجية.
- كتابة تعليمة برمجية تستخدم بشكل فعال مسافات بيضاء، وتنسيق خطوط التعليمة البرمجية.

محتويات الوحدة:

١- مقدمة

٢- اختيار أسماء المتغيرات التي تتبع القواعد والاصطلاحات

٣- إنشاء تعليقات تعليمات برمجية فعالة

٤- استخدام المسافة البيضاء لتسهيل قراءة التعليمات البرمجية

٥- تمرين - إكمال نشاط التحدي لتحسين سهولة قراءة التعليمات البرمجية

٦- مراجعة الحل لتحسين نشاط تحدي سهولة قراءة التعليمات البرمجية

٧- اختبر معلوماتك

٨- الملخص

١ المقدمة

يجب أن تنقل التعليمات البرمجية التي تكتبها هدفك، إلى كل من المحول البرمجي، وأيضاً المطورين الآخرين الذين قد يحتاجون إلى قراءة تعليماتك البرمجية. وبما أنك المطور الذي سيقرأ تعليماتك البرمجية في معظم الأحيان، وأحيانا بعد أشهر من كتابتك لها في الأصل، **فمن مصلحتك كتابة تعليمة برمجية واضحة وسهلة الفهم**.

تذكر أنه يمكنك كتابة التعليمات البرمجية مرة واحدة، ولكن ستحتاج إلى قراءتها عدة مرات.

لنفترض أنه تمت مطالبتك بكتابة بعض التعليمات البرمجية لمجموعة أخرى من المطورين، يمكنك الاجتماع معهم لمناقشة المواصفات والمتطلبات، بعد الاجتماع، يخبرونك أنك ستعمل بشكل مستقل أثناء التطوير، بمجرد الانتهاء، ستقوم بتسليم التعليمات البرمجية إلى المجموعة الأخرى، مهمة الترميز ليست خارج مستوى مهاراتك، ولكن لم يكن لديك نية لكتابة التعليمات البرمجية التي سيطورها شخص آخر، أخبرك الفريق أنه طالما عليك اتباع اصطلاحات الترميز القياسية لC# فلا داعي للقلق، يمكنك وضع خطط لمراجعة اصطلاحات ترميز C# التي تتعلق بالتعليمات البرمجية التي ستعمل عليها.

في هذه الوحدة، ستتعلم كيفية اختيار أسماء للمتغيرات، تصف الغرض منها وهدفها، ستتعلم كيفية إضافة تعليقات التعليمات البرمجية التي توثق متطلبات البرنامج، ونهجك في التعليمات البرمجية، بالإضافة إلى إرشاد المحول البرمجي مؤقتاً لتجاهل أسطر التعليمات البرمجية المعلقة، أخيرًا، ستتعلم كيف يمكن استخدام مسافات بيضاء للمساعدة في فهم العلاقة بين الخطوط الفردية للتعليمة البرمجية.

في نهاية هذه الوحدة، سوف تكتب التعليمة البرمجية بشكل هادف أكثر، مع التركيز على قابلية القراءة، وجودة التعليمة، للاتصال بكل من المحول البرمجي والمطورين الآخرين.

٢ اختيار أسماء المتغيرات التي تتبع القواعد والاصطلاحات

قال مطور برامج ذات مرة: "أصعب جزء في تطوير البرامج هو تسمية الأشياء" فاسم أي متغير لا يجب أن يتبع فقط قواعد بناء جملة معينة؛ بل يجب أيضاً أن يُستخدم لجعل التعليمة البرمجية أكثر سهولة في القراءة، والاستيعاب للإنسان. هذا كثير لنطلبه من سطر واحد من التعليمات البرمجية!

قواعد اسم المتغير

هناك بعض قواعد تسمية المتغير، يتم فرضها من قبل المحول البرمجي C#

- يمكن أن تحتوي أسماء المتغيرات على أرقام، وحرف شرطة سفلية underscore _ لا يسمح بأحرف خاصة مثل رموز العملة $ والشرطة dash - وعلامة #
- يجب أن تبدأ أسماء المتغيرات بحرف أبجدي، أو شرطة سفلية underscore _ وليس برقم، يستخدم المطورون شرطة سفلية لغرض خاص، لذلك حاول ألا تستخدمها في الوقت الراهن.
- يجب ألا تكون أسماء المتغيرات كلمة أساسية في C# على سبيل المثال، لن يسمح بإعلان أسماء المتغيرات هذه float float; أو string string;
- أسماء المتغيرات حساسة لحالة الأحرف، وهذا يعني أن `string myValue;` , `string MyValue;` هما متغيرات مختلفة.

اصطلاحات اسم المتغير

الاصطلاحات هي اقتراحات يتم الاتفاق عليها من قبل مجتمع تطوير البرمجيات، بينما أنت حر في أن تقرر عدم اتباع هذه الاصطلاحات، فهي شائعة لدرجة أنه قد يصعب على المطورين الآخرين فهم تعليماتك البرمجية أن لم تتبع هذه القواعد، يجب عليك ممارسة اعتماد هذه الاصطلاحات، وجعلها جزءاً من عادات الترميز الخاصة بك.

- يجب أن تستخدم في أسماء المتغيرات **حروفًا كبيرة وسطية**، وهي أسلوب كتابة يستخدم حرفًا صغيرًا في بداية الكلمة الأولى وحرفًا كبيرًا في بداية كل كلمة لاحقة، مثل `string thisIsCamelCase;`
- يجب أن تكون أسماء المتغيرات وصفية، ذات معنى في تطبيقك، اختيار اسم المتغير يمثل نوعية البيانات التي سيحتفظ بها (وليس نوع بيانات) مثل `bool orderComplete;` وليس `bool isComplete;`
- يجب أن تكون أسماء المتغيرات كلمة كاملة واحدة، أو أكثر ملحقة معًا، لا تستخدم الاختصارات؛ لأن اسم المتغير قد يكون غير واضح للآخرين الذين يقرؤون تعليماتك، مثل `decimal orderAmount;` وليس `decimal odrAmt;`
- لا يجب أن تتضمن أسماء المتغيرات نوع بيانات المتغير، قد ترى بعض النصائح لاستخدام نمط مثل `string strMyValue;` لقد كان نمطًا شائعًا منذ سنوات، ومع ذلك، فإن معظم المطورين لا يتبعون هذه النصيحة اليوم، وهناك أسباب وجيهة لعدم استخدامها.

يتبع المثال `string firstName;` هذه القواعد والاصطلاحات كافة، على افتراض أنني أريد استخدام هذا المتغير لتخزين البيانات التي تمثل الاسم الأول لشخص ما.

أمثلة اسم المتغير

فيما يلي بعض الأمثلة على إعلان المتغيرات (باستخدام أنواع البيانات الشائعة):

```
char userOption;

int gameScore;

float particlesPerMillion;

bool processedCustomer;
```

إصلاحات تسمية أخرى

القواعد والاصطلاحات الموضحة أعلاه مخصصة للمتغيرات الداخلية أو المحلية local variables المتغير المحلي هو متغير يتم تحديد نطاقه داخل نص أسلوب method أو متغير في تطبيق وحدة تحكم، يستخدم عبارات المستوى الأعلى (مثل التعليمات البرمجية في هذه الوحدة).

هناك أنواع أخرى من البنيات التي يمكنك استخدامها في تطبيقاتك، والعديد منها له اصطلاحات خاصة به، على سبيل المثال، غالباً ما تستخدم الفئات classes في برمجة C# وتكون لها اصطلاحات مقترنة، على الرغم من أنك لن تقوم بإنشاء فئات في هذه الوحدة، فمن المهم أن تعرف أن اصطلاحات التسمية التي تعلمتها للتو تناسب إطار عمل تسمية أوسع.

اختبر معلوماتك

أي مما يلي هو مثال على التنسيق الصحيح لاسم متغير نصي؟

- UserOption
- user Option
- userOption

userOption

صحيح هذه هي الطريقة الصحيحة لإعلان متغير نصي.

٣ إنشاء تعليقات تعليمات برمجية فعالة code comments

في هذا التمرين، ستضيف ملاحظات إلى تعليماتك البرمجية، وتعطيل أسطر معينة من التعليمات البرمجية مؤقتاً من التحويل البرمجي، ثم ستنظر في كيفية فهم المحول البرمجي C# للمسافة البيضاء، وكيفية استخدام المسافة البيضاء لزيادة سهولة قراءة التعليمات البرمجية.

ما هو تعليق التعليمة البرمجية؟ code comment

تعليق التعليمة البرمجية هو إصدار تنبيه للمحول البرمجي بتجاهل كل شيء بعد رموز تعليق التعليمة في السطر الحالي "بمعني آخر هي ملاحظة تكتبها لنفسك، يتجاهلها المحرر لأنها ليست تعليمة برمجية"

```
// This is a code comment!
```

قد لا يبدو هذا مفيدًا في البداية، ومع ذلك فهو مفيد في ثلاث حالات:

- عندما تريد ترك ملاحظة حول هدف التعليمة البرمجية، قد يكون من المفيد تضمين تعليقات التعليمات التي تصف الغرض منها، أو تسجيل أفكارك، عند كتابة مجموعة صعبة بشكل خاص من التعليمات، سوف تشكرك نفسك في المستقبل.
- عندما تريد إزالة التعليمة البرمجية مؤقتًا من التطبيق، لمحاولة اتباع طريقة أخرى، أو نهج مختلف؛ ولكنك لم تقتنع بعد بأن فكرتك الجديدة سوف تنجح، يمكنك تعليق التعليمة البرمجية، وكتابة التعليمة الجديدة، بمجرد اقتناعك، أعمل بالطريقة التي تريدها، يمكنك حذف التعليمة القديمة بأمان (التعليمة البرمجية المعلق عليها)
- إضافة رسالة مثل TODO لتذكيرك بالنظر إلى مقطع معين من التعليمات البرمجية لاحقاً، استخدامه بحكمة، فهو نهج مفيد. ربما تعمل على ميزة أخرى عندما تقرأ سطراً من التعليمات البرمجية يثير القلق، بدلاً من تجاهل المشكلة الجديدة، يمكنك وضع علامة عليها للتحقيق فيها لاحقاً.

ملاحظة

يجب استخدام التعليقات لنطق ما لا يمكن للتعليمة البرمجية النطق به. في كثير من الأحيان، يعدل المطورون تعليماتهم البرمجية؛ ولكنهم ينسون تحديث تعليقات التعليمة البرمجية، من الأفضل استخدام التعليقات للأفكار العامة للمشروع، وعدم إضافة تعليقات حول كيفية عمل سطر فردي للتعليمة البرمجية.

إعداد بيئة الترميز الخاصة بك

يتضمن هذا الدرس تمارين ترشدك خلال عملية إنشاء نماذج التعليمات البرمجية وتشغيلها، يتم تشجيعك على إكمال هذه الأنشطة باستخدام Visual Studio Code كبيئة تطوير. سيساعدك Visual Studio Code على أن تصبح أكثر راحة في كتابة التعليمات البرمجية، وتشغيلها في بيئة مطور يستخدمها المحترفون في جميع أنحاء العالم.

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** حدد فتح مجلد Visual Studio Code **Open Folder**

٣. في مربع حوار **فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدامه لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في مربع حوار **فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في **Terminal** حدد **New Terminal**

لاحظ أن موجه الأوامر في قائمة Terminal يعرض مسار المجلد للمجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

ملاحظة

إذا كنت تعمل على جهازك بدلا من بيئة أخرى، وأكملت الوحدات الأخرى في سلسلة C# هذه، فربما تكون قد أنشأت بالفعل مجلد مشروع لاختبار التعليمات البرمجية. إذا كان الأمر كذلك، يمكنك تخطي الخطوة التالية، والتي تستخدم لإنشاء تطبيق وحدة تحكم في مجلد TestProject

٦. في موجه الأوامر Terminal لإنشاء تطبيق وحدة تحكم في مجلد محدد، اكتب **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

٧. في قائمة استكشاف قم بتوسيع المجلد CsharpProjects

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# يسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**

٩. احذف أسطر التعليمات البرمجية الموجودة.

سوف تستخدم هذا المشروع، وحدة تحكم C# لإنشاء نماذج التعليمات البرمجية، وبنائها، وتشغيلها أثناء هذا الدرس.

١٠. أغلق لوحة Terminal

إنشاء تعليقات التعليمات البرمجية code comments واستخدامها

في هذه المهمة، ستحاول إنشاء أنواع مختلفة من تعليقات التعليمات البرمجية وإزالتها، في محرر Visual Studio Code، أدخل التعليمات البرمجية التالية:

```
string firstName = "Bob";
int widgetsSold = 7;
Console.WriteLine($"{firstName} sold
{widgetsSold} widgets.");
```

١. لتعديل تعليماتك مع تعليقها والمراجعات، قم بتحديث التعليمات كما يلي:

```
string firstName = "Bob";
int widgetsPurchased = 7;
// Testing a change to the message.
// int widgetsSold = 7;
// Console.WriteLine($"{firstName} sold
{widgetsSold} widgets.");
Console.WriteLine($"{firstName} purchased
{widgetsPurchased} widgets.");
```

٢. خذ دقيقة لمراجعة تعليقاتك وتحديثات التعليمات البرمجية.

لاحظ أنه يتم استخدام تعليقات التعليمات البرمجية لتوثيق التغيير المحتمل الذي يتم إجراؤه، وتعطيل الرسالة القديمة مؤقتاً أثناء اختبار الرسالة الجديدة. ستكون خطوتك التالية اختبار تحديثك، إذا كنت راضياً عن التعليمات البرمجية الجديدة، يمكنك حذف التعليمات القديمة التي تم التعليق عليها بأمان، هذا نهج أكثر أماناً، ومنهجية لتعديل التعليمات البرمجية للعمل حتى تقتنع بأنك مستعد لإزالته نهائياً.

٣. في قائمة ملف **File** انقر فوق حفظ **Save**

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٥. في موجه الأوامر Terminal اكتب **dotnet run** ثم اضغط على Enter

ينبغي أن تشاهد الإخراج التالي:

```
Bob purchased 7 widgets.
```

مرة أخرى، إذا كنت راضياً عن التحديثات، فاحذف التعليمات القديمة التي تم التعليق عليها.

٦. حذف تعليقات التعليمات البرمجية.

يجب أن تتطابق التعليمة البرمجية مع ما يلي:

```
string firstName = "Bob";
int widgetsPurchased = 7;
Console.WriteLine($"{firstName} purchased
{widgetsPurchased} widgets.");
```

٧. تطبيق تعليق كتلة block comment لتعليق أسطر متعددة، قم بتحديث تعليماتك كما يلي:

```
/*
string firstName = "Bob";
int widgetsPurchased = 7;
Console.WriteLine($"{firstName} purchased
{widgetsPurchased} widgets.");
*/
```

تعليقات الكتلة Block comments رائعة إذا كنت بحاجة إلى كتابة تعليق طويل أو إزالة العديد من أسطر التعليمات البرمجية. لتعليقات الكتلة استخدام `/*` في بداية الأسطر `*/` وفي النهاية. استخدام تعليق كتلة هو أسرع وأسهل طريقة لتعطيل ثلاثة أسطر أو أكثر من التعليمات البرمجية.

٨. استبدل التعليمات البرمجية الموجودة لديك بالآتي:

```
Random random = new Random();
string[] orderIDs = new string[5];
// Loop through each blank orderID
for (int i = 0; i < orderIDs.Length; i++)
{
```

```
    // Get a random value that equates to ASCII
letters A through E
    int prefixValue = random.Next(65, 70);
    // Convert the random value into a char,
then a string
    string prefix =
Convert.ToChar(prefixValue).ToString();
    // Create a random number, pad with zeroes
    string suffix = random.Next(1,
1000).ToString("000");
    // Combine the prefix and suffix together,
then assign to current OrderID
    orderIDs[i] = prefix + suffix;
}
// Print out each orderID
foreach (var orderID in orderIDs)
{
    Console.WriteLine(orderID);
}
```

ملاحظة

هناك العديد من مفاهيم C# في هذا المثال من التعليمات البرمجية، ربما تكون جديدة عليك. ليس من الضروري فهم ما تقوم به، المثال لتوضيح كيف تساعدك التعليقات على فهم الغرض من التعليمة البرمجية.

٩. خذ دقيقة لمعرفة ما إذا كان يمكنك معرفة الغرض من التعليمات البرمجية.

بالنظر إلى التعليقات، قد تتمكن من معرفة ما تفعله التعليمة (بافتراض أن التعليقات تصف الحالة الحالية بدقة، وتم تعديلها مع تعديل التعليمات) ولكن هل يمكنك تخمين سبب وجود هذه التعليمة البرمجية؟ ألن يكون من المفيد أن يكون هناك بعض الشرح في الجزء العلوي، من ملف التعليمات البرمجية، الذي يوفر بعض السياق، ويصف الغرض منه؟

١٠. فكر في كيفية تحسين التعليقات

لاحظ أن هناك مشكلتين رئيسيتين مع هذه التعليقات:

- تفسر تعليقات التعليمة البرمجية بشكل غير ضروري الوظيفة الواضحة للخطوط الفردية للتعليمة البرمجية. تعتبر هذه التعليقات منخفضة الجودة؛ لأنها مجرد شرح لكيفية عمل C# أو أساليب مكتبة فئات «Microsoft .NET» إذا لم يكن القارئ على دراية بهذه الأفكار، فيمكنه البحث عنها باستخدام learn.microsoft.com أو IntelliSense
- لا توفر تعليقات التعليمة البرمجية أي سياق للمشكلة التي يتم حلها بواسطة التعليمة البرمجية. تعتبر هذه التعليقات منخفضة الجودة؛ لأن القارئ لا يحصل على أي نتيجة بخصوص الغرض من هذه التعليمة البرمجية، خاصةً فيما يتعلق بالنظام العام.

١١. إزالة التعليقات الموجودة. يجب أن تتطابق التعليمة البرمجية مع ما يلي:

```
Random random = new Random();
string[] orderIDs = new string[5];

for (int i = 0; i < orderIDs.Length; i++)
{
    int prefixValue = random.Next(65, 70);
    string prefix =
Convert.ToChar(prefixValue).ToString();
    string suffix = random.Next(1,
1000).ToString("000");

    orderIDs[i] = prefix + suffix;
}

foreach (var orderID in orderIDs)
{
```

```
    Console.WriteLine(orderID);
}
```

لاحظ أن التعليمات البرمجية أقل ازدحاماً بالفعل.

١٢. لإضافة تعليق يشرح الغرض العام من تعليماتك البرمجية، قم بتعديل التعليمات كما يلي:

```
/*
  The following code creates five random
OrderIDs
  to test the fraud detection process.
OrderIDs
  consist of a letter from A to E, and a three
  digit number. Ex. A123.
*/
Random random = new Random();
string[] orderIDs = new string[5];

for (int i = 0; i < orderIDs.Length; i++)
{
    int prefixValue = random.Next(65, 70);
    string prefix =
Convert.ToChar(prefixValue).ToString();
    string suffix = random.Next(1,
1000).ToString("000");

    orderIDs[i] = prefix + suffix;
}

foreach (var orderID in orderIDs)
{
    Console.WriteLine(orderID);
}
```

فائدة التعليق قرار شخصي، في جميع المسائل المتعلقة بسهولة قراءة التعليمة البرمجية، يجب عليك استخدام أفضل حكم لديك، قم بما تعتقد أنه الأفضل لتحسين وضوح تعليماتك البرمجية.

خلاصة

الفوائد الأساسية من هذا التمرين:

- استخدام تعليقات التعليمة البرمجية لترك ملحوظات ذات معنى، لنفسك حول المشكلة التي تحلها تعليمتك البرمجية.
- لا تستخدم تعليقات التعليمة البرمجية التي تشرح كيفية عمل «C#» أو مكتبة فئات «Microsoft .NET»
- استخدام تعليقات التعليمة البرمجية عند تجربة الحلول البديلة، مؤقتًا حتى تصبح جاهزًا لتثبيت حل التعليمة البرمجية الجديد، وعند هذه النقطة يمكنك حذف التعليمة القديمة.
- لا تثق أبدًا في تعليقات الغير، قد لا تعكس التعليقات الحالة الحالية للتعليمة البرمجية بعد العديد من التغييرات والتحديثات.

٤ استخدام المسافة البيضاء whitespace لتسهيل قراءة التعليمات البرمجية

يفهم مصممو الطباعة والويب أن وضع الكثير من المعلومات في مساحة صغيرة يربك المشاهد، لذلك فإنهم يستخدمون المسافات البيضاء أو مساحة فارغة بشكل تنظيمي، لتقسيم المعلومات، وتعظيم قدرة المشاهد على استهلاك الرسالة الأساسية لعملهم.

يمكن للمطورين استخدام استراتيجية مشابهة عند كتابة التعليمة البرمجية في المحرر، استخدام مسافة بيضاء لنقل المعنى، يمكن للمطورين زيادة وضوح هدف تعليماتهم البرمجية.

ما هي المسافة البيضاء؟ whitespace

يشير مصطلح "المسافة البيضاء" إلى المسافات التي تنتجها space bar وعلامات التبويب، المسافة المزدوجة التي ينتجها مفتاح tab والأسطر الجديدة التي ينتجها مفتاح enter

يتجاهل المحول البرمجي «C#» المسافة البيضاء. لفهم كيفية تجاهل المسافة البيضاء، وكيفية زيادة الوضوح باستخدام المساحة البيضاء، اعمل من خلال التمرين التالي.

إضافة تعليمة برمجية لتوضيح كيفية تجاهل المسافة البيضاء بواسطة المحول البرمجي C#

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر افتح Visual Studio Code ثم أكمل خطوات إعداد ملف Program.cs

٢. أدخل التعليمات التالية في المحرر:

```
// Example 1:
Console
.
WriteLine
(
"Hello Example 1!"
)
;

// Example 2:
string firstWord="Hello";string
lastWord="Example
2";Console.WriteLine(firstWord+"
"+lastWord+"!");
```

٣. في موجه الأوامر Terminal اكتب dotnet run ثم اضغط على Enter

ينبغي أن تشاهد الإخراج التالي:

```
Hello Example 1!
Hello Example 2!
```

٤. خذ دقيقة للنظر فيما تخبرك به هذه النتيجة، حول كيفية استخدام المسافة البيضاء في تعليماتك البرمجية.

يوضح هذان المثالان للتعليمة البرمجية اثنين من الأفكار الحيوية:

- لا تهم المسافة البيضاء المحول البرمجي. لكن...
- يمكن للمسافة البيضاء، عند استخدامها بشكل صحيح، زيادة قدرتك على قراءة وفهم التعليمة البرمجية.

من المحتمل أن تكتب التعليمات البرمجية مرة واحدة، ولكن تحتاج إلى قراءتها عدة مرات. لذلك، يجب التركيز على سهولة قراءة التعليمات التي تكتبها، بمرور الوقت، ستتعرف على مكان، وكيفية استخدام المسافة البيضاء، مثل حرف المسافة، وعلامات التبويب، والأسطر الجديدة.

التوجيه المبكر:

- ينتمي كل أمر كامل (عبارة) إلى سطر منفصل.
- إذا أصبح سطر واحد من التعليمات البرمجية طويلًا، فيمكنك تقسيمه. ومع ذلك، يجب عليك تجنب تقسيم عبارة واحدة بشكل تعسفي إلى عدة أسطر، الا إذا كان لديك سبب وجيه للقيام بذلك.
- استخدم مسافة على يسار ويمين عامل التشغيل.

٥. استبدل التعليمات البرمجية الموجودة بالتعليمات التالية:

```
Random dice = new Random();
int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);
int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");
if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3)) {
    if ((roll1 == roll2) && (roll2 == roll3)) {
        Console.WriteLine("You rolled triples!
+6 bonus to total!");
        total += 6;
    } else {
        Console.WriteLine("You rolled doubles!
+2 bonus to total!");
        total += 2;
    }
}
```

لاحظ أن هذه التعليمة البرمجية لا تتضمن الكثير من المسافة البيضاء. سيتم استخدام هذه التعليمة لتوضيح نهج إضافة مسافة بيضاء إلى تطبيقاتك. يجب أن تسهل المسافة البيضاء الفعالة فهم ما تقوم به تعليماتك البرمجية.

ملاحظة

تستخدم التعليمات البرمجية الفئة Random للمساعدة في تطوير لعبة محاكاة أحجار الزهر، حيث يتم استخدام القيمة الإجمالية من ثلاث لفات لتقييم درجة "الفوز" تمنح التعليمات البرمجية نقاطاً إضافية لتدحرج الزوجي أو الثلاثي. لا تحتاج إلى فهم هذه التعليمات البرمجية بشكل كامل من أجل رؤية فائدة تضمين المسافة البيضاء.

٦. خذ دقيقة للنظر في كيفية استخدام المسافة البيضاء لتحسين قابلية قراءة هذه التعليمات البرمجية.

توجد سمتين لهذه التعليمة البرمجية يجب العلم بهما:

- لا توجد مسافة بيضاء رأسية في مثال التعليمات البرمجية هذا، بمعنى آخر، لا توجد أسطر فارغة تفصل بين أسطر التعليمات. يتم تشغيل كل ذلك معًا في قائم واحدة كثيفة.
- يتم ضغط كتل التعليمات البرمجية كما تم تعريفها بواسطة رمزي الأقواس { } المتعرجين معًا، مما يجعل من الصعب تمييز حدودها بصريًا.

بشكل عام، لتحسين قابلية القراءة، يمكنك تقديم سطر فارغ بين سطرين أو ثلاثة أو أربعة أسطر من التعليمات التي تفعل أشياء مماثلة أو ذات صلة.

صياغة التعليمة باستخدام مسافة بيضاء رأسية قرار شخصي، ربما لا يتفق اثنين من المطورين على ما هو أكثر سهولة للقراءة، أو متى يتم إضافة مسافة بيضاء، استخدام أفضل حكم لديك.

٧. لإضافة مسافة بيضاء رأسية تحسن من سهولة القراءة، قم بتعديل التعليمات كما يلي:

```
Random dice = new Random();
```

```
int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3)) {
    if ((roll1 == roll2) && (roll2 == roll3)) {
        Console.WriteLine("You rolled triples!
+6 bonus to total!");
        total += 6;
    } else {
        Console.WriteLine("You rolled doubles!
+2 bonus to total!");
        total += 2;
    }
}
```

يتم استخدام السطر الأول من المسافة البيضاء لفصل إعلان المتغير dice عن أسطر التعليمات المستخدمة لتعيين القيم إلى متغيرات لفتك. يسهل هذا الفصل معرفة كيفية استخدام dice في تعليماتك البرمجية.

يفصل السطر التالي من المسافة البيضاء إعلان متغيرات لفتك عن إعلان total يعد تجميع إعلان متغيرات لفة الثلاثي مفيداً بطريقتين، أولاً، يقوم بإنشاء مجموعة من أسطر التعليمات التي تتضمن المتغيرات ذات الصلة، ثانيا، أسماء المتغيرات متشابهة جداً ويتبع الإعلان نفس النمط، لذا، فإن تجميعهما معاً يجذب عينك إلى أوجه التشابه، ويساعد على كشف الاختلافات.

أخيرًا، يفصل السطر الثالث من المسافة البيضاء مجموعة أخرى من العبارات ذات الصلة عن عبارات if المتداخلة، ترتبط مجموعة العبارات التي تتضمن الإعلان عن total وأسلوب Console.WriteLine() بالغرض وليس بالمظهر. تركز تعليماتك على القيمة الإجمالية التي تحققها أحجار الزهر الثلاثة، وما إذا كانت اللفة زوجية أم ثلاثية. ترتبط هذه الأسطر لأنك تحتاج إلى حساب الإجمالي، وإبلاغ المستخدم بنتائج اللفة.

قد يجادل بعض المطورين أنه يجب عليك إضافة سطر فارغ بين الإعلان عن total, Console.WriteLine() اختيار المسافة البيضاء يتوقف على أفضل حكم لك. يجب أن تقرر ما هو أكثر قابلية للقراءة بالنسبة لك، واستخدام هذا النمط باستمرار.

كل ما تبقى لديك هو عبارة if يمكنك فحص ذلك الآن.

٧. مع التركيز على أسطر التعليمات البرمجية أسفل الكلمة الأساسية if قم بتعديل التعليمات البرمجية كما يلي:

```
Random dice = new Random();

int roll1 = dice.Next(1, 7);
int roll2 = dice.Next(1, 7);
int roll3 = dice.Next(1, 7);

int total = roll1 + roll2 + roll3;
Console.WriteLine($"Dice roll: {roll1} +
{roll2} + {roll3} = {total}");

if ((roll1 == roll2) || (roll2 == roll3) ||
(roll1 == roll3))
{
    if ((roll1 == roll2) && (roll2 == roll3))
    {
        Console.WriteLine("You rolled triples!
+6 bonus to total!");
        total += 6;
    }
```

```
    else
    {
        Console.WriteLine("You rolled doubles!
+2 bonus to total!");
        total += 2;
    }
}
```

٨. لاحظ أنك نقلت الأقواس المتعرجة {} إلى خطها الخاص لتحسين التباعد.

يقوم الرمزان {} بإنشاء كتل تعليمات برمجية. تتطلب العديد من بنيات C# الكتل البرمجية. يجب وضع هذه الرموز على سطر منفصل بحيث تكون حدودها مرئية، وقابلة للقراءة بوضوح.

علاوة على ذلك، من المهم استخدام مفتاح tab لمحاذاة رموز كتلة التعليمات البرمجية أسفل الكلمة الأساسية التي تنتمي إليها. على سبيل المثال، لاحظ سطر التعليمات البرمجية الذي يبدأ بالكلمة الأساسية if أسفل هذا السطر يوجد الرمز { تسهل هذه المحاذاة فهم أن الرمز { ينتمي إلى عبارة if علاوة على ذلك، فإن الرمز } الأخير يتوافق مع عبارة if أيضًا.

إن الجمع بين المحاذاة والمسافات البادئة، يجعل من السهل فهم أين تبدأ كتلة التعليمات البرمجية وتنتهي.

إضافة مسافة بادئة لخطوط التعليمات البرمجية الفرعية، داخل الكتلة البرمجية هذه، مما يشير إلى أنها "تنتمي" إلى هذه الكتلة البرمجية.

تتبع نمطًا مشابهًا مع عبارة if الداخلية وعبارة else والتعليمات الموجود داخل كتل التعليمات البرمجية هذه.

لا يتفق الجميع مع إرشادات النمط هذه، لتضمين المسافة البيضاء. ومع ذلك، يجب أن تفكر في استخدام هذا التوجيه، كنقطة بداية عند كتابة التعليمات البرمجية، في المستقبل، يمكن أن تكون هادفاً عند اتخاذ قرار، الانحراف عن هذا التوجيه.

خلاصة

الفوائد الأساسية من هذا التمرين:

- استخدام مسافة بيضاء بحكمة؛ لتحسين سهولة قراءة التعليمة البرمجية.
- استخدام فراغ الأسطر، إنشاء أسطر فارغة لفصل عبارات التعليمات البرمجية، والعبارات التي تضم أسطر التعليمات المتشابهة أو التي تعمل معاً.
- استخدام الأسطر الفارغة لفصل رموز كتلة التعليمات البرمجية؛ بحيث تكون التعليمة البرمجية على الخط الخاص بها.
- استخدم المفتاح tab لمحاذاة كتلة التعليمات البرمجية مع الكلمة الأساسية المقترنة بها.
- وضع مسافة بادئة للتعليمات الفرعية، داخل كتلة التعليمات البرمجية لإظهار التبعية.

٥ تمرين ـ إكمال نشاط التحدي لتحسين قابلية قراءة التعليمات البرمجية

ستعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

تحدي تحسين قابلية قراءة التعليمات البرمجية

في هذا التحدي، سوف تستخدم التقنيات التي تعلمتها في هذه الوحدة، لتحسين قابلية قراءة نموذج التعليمات البرمجية، يتم تزويدك بعينة تعليمات برمجية سيئة النمط، والتعليق عليها، هدفك هو تعديل التعليمات البرمجية، باستخدام إرشادات النمط، لأسماء المتغيرات، وتعليقات التعليمات البرمجية، ومسافات بيضاء، لتحسين سهولة قراءة التعليمات البرمجية.

تحدي التعليمات البرمجية ـ تطبيق إرشادات النمط لتحسين قابلية القراءة

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code إذا لزم الأمر افتح Visual Studio Code ثم أكمل خطوات إعداد ملف Program.cs في المحرر.

٢. أكتب الرمز التالي:

```
string str = "The quick brown fox jumps over
the lazy dog.";
// convert the message into a char array
char[] charMessage = str.ToCharArray();
// Reverse the chars
Array.Reverse(charMessage);
int x = 0;
// count the o's
foreach (char i in charMessage) { if (i == 'o')
{ x++; } }
// convert it back to a string
```

```
string new_message = new String(charMessage);
// print it out
Console.WriteLine(new_message);
Console.WriteLine($"'o' appears {x} times.");
```

ملاحظة

ربما يتضمن نموذج التعليمات البرمجية هذا، أساليب مكتبة فئات .NET غير معروفة لك. على سبيل المثال، قد لا تكون على دراية بأسلوب ToCharArray() الفئة string أو أسلوب Reverse الفئة Array لا تحتاج إلى فهم نموذج التعليمات البرمجية بشكل كامل، من أجل أن تكون ناجحا في هذا التحدي.

الغرض العام من هذا النموذج هو عكس سلسلة نصية، وحساب عدد المرات التي يظهر فيها حرف معين.

٣. لتحسين قابلية القراءة، قم بتعديل التعليمات البرمجية باستخدام إرشادات النمط.

استخدم التقنيات التي تعلمتها في هذه الوحدة لإدخال تحسينات على التعليمات البرمجية، وزيادة إمكانية قراءتها.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٦ مراجعة حل تحدي تحسين قابلية قراءة التعليمات البرمجية

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
/*
   This code reverses a message, counts the
number of times
   a particular character appears, then prints
the results
   to the console window.
 */

string originalMessage = "The quick brown fox
jumps over the lazy dog.";

char[] message = originalMessage.ToCharArray();
Array.Reverse(message);

int letterCount = 0;

foreach (char letter in message)
{
    if (letter == 'o')
    {
        letterCount++;
    }
}

string newMessage = new String(message);

Console.WriteLine(newMessage);
Console.WriteLine($"'o' appears {letterCount}
times.");
}
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، قد تكون أتيت ببعض أسماء متغيرات مختلفة، ومسافات عمودية، ومسافات بادئة لعلامات التبويب، فيما يلي قائمة بالتغييرات التي تم إجراؤها.

- تتضمن التعليمات البرمجية وصفاً عاماً، لما تحاول قائمة التعليمات بأكملها إنجازه، في تعليق متعدد الأسطر في الأعلى، يمكنك القول بأن هذا تحسن صغير على تعليقات التعليمات الأصلية، ومع ذلك، نظراً لوصف التحدي للتعليمات، لم يكن هناك المزيد من السياق المتاح.
- تمت إزالة التعليقات الفردية لأنها لم تكن توفر أي نظرة ثاقبة حقيقية، حول الغرض من التعليمات البرمجية أو دالتها.
- تمت إضافة عدة أسطر فارغة لتحسين صياغة سرد التعليمات البرمجية، احتفظ بخطوط التعليمات البرمجية معاً عندما تظهر متشابهة، أو عندما تعمل مع بعضها لإنجاز مهمة صغيرة.
- تمت إضافة فراغات أسطر وعلامات تبويب، لتحسين مظهر العبارة foreach والعبارة if
- تم تطبيق اصطلاحات تسمية المتغيرات الداخلية لتوصيل الغرض من كل قيمة بشكل أفضل.

إذا حددت نفس المشكلات، وعالجتها بطريقة مماثلة، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

٧ اختبر معلوماتك

١- أي مما يلي يوضح الإرشادات الموصي بها لتسمية متغير؟

- `my-string`
- `initialMessage$`
- `initialMessage`

٢- أي من التالي هو سبب سيء لاستخدام تعليق التعليمة البرمجية؟

- لوصف الهدف العام للتعليمات البرمجية.
- لشرح كيفية عمل كلمة أساسية C# جديدة.
- للتعليق مؤقتاً على سطر من التعليمات البرمجية أثناء تقييم ميزة بطريقة مختلفة

٣- أي من العبارات التالية حول استخدام المسافة البيضاء في التعليمات البرمجية صحيح؟

- سيؤدي تقليل مقدار المسافة البيضاء إلى تحسين أداء التعليمات البرمجية (سرعة التنفيذ)
- يجب استخدام المسافة البيضاء كل سطرين أو ثلاثة لفصل أسطر التعليمات البرمجية
- يجب استخدام المسافة البيضاء بحكمة، لتحسين قابلية قراءة التعليمات البرمجية

راجع إجابتك

١ `initialMessage`

صحيح هذا النهج المناسب لاسم متغير والاسم وصفي.

٢ لشرح كيفية عمل كلمة أساسية C# جديدة

لا ينبغي استخدام التعليقات لهذا السبب

٣ يجب استخدام المسافة البيضاء بحكمة، لتحسين قابلية قراءة التعليمات البرمجية.

صحيح يجب استخدام المسافة البيضاء لتحسين قابلية القراءة ومعرفة الغرض العام

٨ الملخص

كان هدفك هو كتابة التعليمات البرمجية التي يسهل قراءتها وفهمها.

لقد استخدمت اصطلاحات تسمية المتغيرات لتحسين سهولة قراءة التعليمات البرمجية. تساعدك حالة Camel case على تحديد متغير داخلي بدلاً من المعرفات الأخرى في التعليمات البرمجية.

كما أنه يمنحك طريقة لاستخدام كلمات متعددة تصف الغرض من متغير معين للقارئ، لقد استخدمت تعليقات التعليمات البرمجية لإزالة التعليمات البرمجية التي تريد إعادة كتابتها مؤقتاً قبل الحذف، كما استخدمت تعليقات التعليمات البرمجية لإضافة توضيحات أكثر وضوحاً، للغرض العام للتعليمات البرمجية في النظام، وأخيراً، رأيت كيف لا يهتم المحول البرمجي بالمسافة البيضاء في التعليمات، لذلك استخدمت المسافة البيضاء بحكمة، للمساعدة في تحسين قراءة تعليماتك البرمجية بسهولة.

لقد فهمت أنك تكتب التعليمات البرمجية مرة واحدة، ولكنك تقرأها عدة مرات. يساعدك تحسين قابلية القراءة، أنت والمطورين الآخرين على الحفاظ على التعليمات البرمجية الخاصة بك، ودعمها على المدى الطويل.

الوحدة السادسة

مشروع التحدي - تطوير هياكل foreach, if-elseif-else لمعالجة بيانات المصفوفة

أظهر قدرتك على تحديث تطبيق وحدة تحكم يقوم بتنفيذ المصفوفات، وحلقات التكرار foreach وما إذا كانت العبارات استجابة لتحديث الميزات المطلوب من قبل المستخدم.

الأهداف التعليمية

في هذه الوحدة، ستثبت قدرتك على:

- مراجعة على تطبيق وحدة تحكم C# الذي يستخدم حلقات foreach المتداخلة، للوصول إلى بيانات المصفوفة استناداً إلى مواصفات ميزة محدثة.
- تعديل المتغيرات وعبارات if في تطبيق وحدة تحكم C# لإنشاء فروع التعليمات البرمجية، والحسابات المطلوبة لتلبية مواصفات ميزة محدثة.

محتويات الوحدة:

١- مقدمة

٢- الإعداد

٣- تمرين - تحديث الإخراج المنسق

٤- تمرين - تحديث القيم المحسوبة

٥- التحقق من المعرفة

٦ ملخص

١ المقدمة

غالبا ما يطلب المستخدمون النهائيون للتطبيقات من المطورين إضافة ميزات جديدة إلى أحد التطبيقات، تشير طلبات المستخدم إلى استخدام تطبيقك، والأهم من ذلك، أن العميل يخطط لمواصلة استخدام التطبيق، يريد فقط تحديث بعض الميزات.

القدرة على تحديث تطبيق موجود استناداً إلى طلبات المستخدم مهمة جداً، يحافظ التحديث الناجح على تكامل التطبيق الأصلي، مع تزويد المستخدم بالتجربة المحسنة التي طلبها.

لنفترض أنك مساعد معلم في مدرسة، وطورت تطبيقاً لإتمام عملية الدرجات، يستخدم التطبيق مصفوفات لتخزين أسماء الطلاب، والمواد التي تم تقديرها، ينفذ التطبيق أيضاً مجموعة من عبارات التكرار، والاختيار، لحساب التقدير النهائي لكل طالب وعرضه، حتى أن تطبيقك يفرق بين مواد الاختبار، ودرجات التقدير الإضافية عند حساب درجات الطلاب، على الرغم من أن التطبيق يقوم بكل ما طلبه المعلم، إلا أنك تلقيت طلباً للحصول على ميزات جديدة، طلب منك المعلم تحديث تقرير الدرجات لإظهار درجات الامتحان، ودرجات التقدير الإضافية، بشكل منفصل عن النتيجة الرقمية النهائية، ودرجة التقدير الحرفية.

تتحداك هذه الوحدة لإضافة قدرات جديدة إلى تطبيق Student Grading

باختصار، تحتاج إلى استخدام مواصفات تقرير الدرجات الجديد للمعلم، من أجل:

- تحديث التعليمات البرمجية للتكرار foreach والاختيار، لحساب درجات نهائية منفصلة للامتحانات، ودرجات التقدير الإضافية، والدرجة الإجمالية.
- تحديث التعليمات البرمجية التي تكتب تقرير الدرجات إلى وحدة التحكم.

في نهاية هذه الوحدة، سيكون لديك إصدار محدث من تطبيق تقييم الطالب Student Grading قادر على حساب درجات الطلاب، وعرضها وفقاً المتطلبات الجديدة للمعلم.

ملاحظة

هذه وحدة مشروع تحدي حيث ستكمل مشروعاً شاملاً وفق المواصفات. تهدف هذه الوحدة أن تكون اختباراً لمهاراتك؛ هناك القليل من التوجيهات، ولا توجد تعليمات خطوة بخطوة.

٢ الإعداد

في هذا المشروع الإرشادي، ستستخدم Visual Studio Code لتطوير أجزاء من تطبيق وحدة تحكم C# ستبدأ بكتابة التعليمات البرمجية التي تجري حسابات رقمية مختلفة، يجب إكمال كافة العمليات الحسابية ضمن بنيات التكرار، والاختيار الموجودة، توفر لك وحدة الإعداد هذه الأهداف العامة للمشروع، ومتطلبات التطبيق. يصف درس الإعداد كيفية تجهيز بيئة التطوير، بما في ذلك مشروع التعليمات البرمجية "Starter"

هام

تتضمن هذه الوحدة أنشطة الترميز التي تتطلب Visual Studio Code ستحتاج إلى الوصول إلى بيئة تطوير Visual Studio Code وتكوينها لتطوير تطبيق C#

مواصفات المشروع

مشروع التعليمات البرمجية Starter هو تطبيق وحدة تحكم C# الذي ينفذ ميزات التعليمات البرمجية التالية:

- يستخدم المصفوفات لتخزين أسماء الطلاب، ودرجات المواد.
- يستخدم عبارة `foreach` للتكرار عبر أسماء الطلاب في حلقة خارجية.
- يستخدم عبارة `if` داخل الحلقة الخارجية لتحديد اسم الطالب الحالي، والوصول إلى درجات المواد الخاصة بالطالب.
- يستخدم عبارة `foreach` داخل الحلقة الخارجية للتكرار من خلال مصفوفة درجات المواد، وإجمالي التقدير.
- يستخدم تعليمة داخل الحلقة الخارجية لحساب متوسط درجة الامتحان لكل طالب.

- استخدم بنية if-elseif-else داخل حلقة foreach الخارجية لتقييم متوسط درجة الاختبار، وتعيين درجة التقدير تلقائياً.
- يدمج الدرجات الإضافية عند حساب درجة الطالب النهائية ودرجة التقدير الحرفية كما يلي:
  - يحسب درجات المواد الإضافية استناداً إلى عدد العناصر في مصفوفة درجات الطالب.
  - يطبق عامل ترجيح بنسبة ١٠٪ على درجات التقدير الإضافية قبل إضافة درجات تقدير إضافية إلى مجموع درجات الاختبار.

هدفك في هذا التحدي هو تنفيذ التعديلات، لإنتاج تقرير النتيجة المطلوب للمعلم.

يسرد تقرير النتيجة الحالي اسم الطالب، متبوعاً بالدرجات الإجمالية المحسوبة، ودرجة التقدير الحرفية. فيما يلي تنسيق التقرير الحالي:

```
Student         Grade   Letter Grade

Sophia          95.6    A
Andrew          91.6    A-
Emma            89.2    B+
Logan           93      A
```

بالإضافة إلى درجة الطالب الرقمية النهائية، ودرجة التقدير الحرفية، يريد المعلم أن يتضمن التقرير المحدث درجة الامتحان، وتأثير العمل الإضافي على الصف النهائي للطالب. يجب أن يظهر تنسيق تقرير النتيجة المحدث كما يلي:

```
Student         Exam Score      Overall Grade   Extra Credit

Sophia          92.2            95.88   A       92 (3.68 pts)
Andrew          89.6            91.38   A-      89 (1.78 pts)
Emma            85.6            90.94   A-      89 (5.34 pts)
Logan           91.2            93.12   A       96 (1.92 pts)
```

الإعداد

استخدم الخطوات التالية للتحضير لتمارين مشروع التحدي:

١. اضغط على Lab Files لتنزيل ملف مضغوط يحتوي على مصدر مشروع Starter

٢. فك ضغط ملفات التنزيل.

ضع في اعتبارك استخدام الكمبيوتر الخاص بك، كبيئة تطوير حتى تتمكن من الوصول إلى تعليماتك البرمجية، بعد إكمال هذه الوحدة.

* على جهازك المحلي، انتقل إلى مجلد التنزيلات.

* انقر بزر الماوس الأيمن فوق Challenge-project-foreach-if-array-CSharp-main.zip ثم حدد Extract all

* حدد إظهار الملفات المستخرجة عند اكتمالها، ثم حدد استخراج.

* دون موقع المجلد المستخرج.

٣. انسخ مجلد ChallengeProject المستخرج إلى مجلد سطح مكتب Windows

ملاحظة

إذا كان هناك مجلد يسمى ChallengeProject موجود بالفعل، يمكنك تحديد استبدال الملفات في الوجهة لإكمال عملية النسخ.

٤. فتح مجلد ChallengeProject الجديد في Visual Studio Code

- افتح Visual Studio Code في بيئة التطوير الخاصة بك.
- في Visual Studio Code في القائمة ملف File حدد فتح مجلد Open Folder

- انتقل إلى مجلد Windows Desktop وحدد موقع مجلد "ChallengeProject"
- حدد ChallengeProject ثم حدد اختيار مجلد Select Folder

يجب أن تظهر قائمة استكشاف أو عرض EXPLORER مجلد ChallengeProject ومجلدين فرعيين باسم Final, Starter

إذا اردت شرح كيفية بناء التطبيق من البداية، أدخل على رابط مشروع موجة تطوير هياكل foreach, if

أنت الآن جاهز لبدء تمارين مشروع التحدي. حظ سعيد!

٣ تمرين ـ تحديث الإخراج المنسق

يتم استخدام تطبيق تقدير الطالب لحساب درجات الطلاب، وعرض درجاتهم استناداً إلى المواد، وواجبات التقدير الإضافية. هدفك في هذا التحدي هو تحديث التعليمات البرمجية التي تنشئ تقرير نقاط، وفقاً لمتطلبات المعلم الجديدة.

المواصفات

في تمرين التحدي الأول هذا، تحتاج إلى إنشاء مثيل للمتغيرات المطلوبة لتقرير النتيجة الجديدة، ثم تحديث العبارة Console.WriteLine() التي تعرض درجات الطلاب في وحدة التحكم.

يجب أن يشمل تطبيقك المحدث على:

- استخدم المصفوفات الموجودة وقيم المصفوفة لكافة حسابات درجات الطلاب.
- استخدم البنية المتداخلة التي توفرها العبارات الموجودة foreach, if
- قم بتعريف وتهيئة أي متغيرات عدد صحيح أخرى مطلوبة لحساب إجمالي الدرجات.
- قم بتعريف وتهيئة أي متغيرات عشرية أخرى مطلوبة للحسابات و/أو قيم تقرير التقدير.

تنسيق التقرير الجديد كما يلي:

```
Student         Exam Score    Overall Grade   Extra Credit

Sophia          0               95.8    A       0 (0 pts)
Andrew          0               91.2    A-      0 (0 pts)
Emma            0               90.4    A-      0 (0 pts)
Logan           0               93      A       0 (0 pts)
```

ملحوظة

بما أن هذا التمرين لا يتضمن تحديث الحسابات، فسيظهر تقرير النتيجة 0 في الحقول الجديدة كما هو موضح أعلاه.

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة لهذا التمرين، أكمل الخطوات التالية:

١. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله.

٢. تحقق من أن التطبيق الخاص بك ينشئ الإخراج التالي:

```
Student          Exam Score      Overall Grade    Extra Credit

Sophia            0                 95.8    A       0 (0 pts)
Andrew            0                 91.2    A-      0 (0 pts)
Emma              0                 90.4    A-      0 (0 pts)
Logan             0                 93      A       0 (0 pts)
```

٤ تمرين ـ تحديث القيم المحسوبة

يتم استخدام تطبيق تقدير الطالب لحساب درجات الطلاب وإعلان درجاتهم، استناداً إلى اختبار المواد، وواجبات التقدير الإضافية. هدفك في هذا التحدي هو تحديث التعليمات البرمجية التي تحسب درجات الطلاب وفقاً لمتطلبات المعلم الجديدة.

المواصفات

في تمرين التحدي هذا، تحتاج إلى إنشاء مثيل للمتغيرات المطلوبة لتقرير النتيجة الجديدة، وإكمال العمليات الحسابية المطلوبة، ثم تحديث العبارة Console.WriteLine() التي تعرض درجات الطلاب إلى وحدة التحكم.

يجب أن يشمل تطبيقك المحدث على:

- استخدم المصفوفات الموجودة وقيم المصفوفة لكافة حسابات درجات الطلاب.
- استخدم البنية المتداخلة التي توفرها العبارات الموجودة foreach, if
- حساب متوسط درجات اختبار المواد، ودرجات التقدير الإضافية باستخدام متغيرات من التعليمات البرمجية الأصلية.
- حساب النتيجة الرقمية النهائية كما يلي: إضافة ١٠٪ من مجموع نقاط التقدير الإضافية إلى مجموع نقاط الاختبار، ثم قسم هذه القيمة على عدد الاختبارات.
- حساب نقاط التقدير الإضافية المكتسبة على النحو التالي: قسمة ١٠٪ من مجموع نقاط التقدير الإضافية على عدد الاختبارات

تلميح

ستحتاج إلى استخدام نوع بيانات (decimal) في المعادلات للحفاظ على المكون الكسري أثناء العمليات الحسابية

تنسيق تقرير النتيجة المطلوب هو:

```
Student      Exam Score    Overall Grade   Extra Credit

Sophia         92.2          95.88    A     92 (3.68 pts)
Andrew         89.6          91.38    A-    89 (1.78 pts)
Emma           85.6          90.94    A-    89 (5.34 pts)
Logan          91.2          93.12    A     96 (1.92 pts)
```

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله.

٢. تحقق من أن التطبيق الخاص بك ينشئ الإخراج المطلوب.

تهانينا إذا نجحت في هذا التحدي

٥ اختبر معلوماتك

١- يعمل مطور على بنية متداخلة foreach تتكرر من خلال بيانات مصفوفات التطبيق، تحتوي المصفوفة الأولى على أسماء ١٠ مناطق جغرافية، وتستند المصفوفات المتبقية على المناطق العشرة، تحتوي كل مصفوفة إقليمية على عدد سكان المدن، التي يزيد عدد سكانها عن مليون نسمة، تنتقل قيم المحتوي في المصفوفات الإقليمية من الأكبر إلى الأصغر، يجمع التطبيق ٢٥ مدينة مأهولة بالسكان في كل منطقة. كيف يجب على المطور التأكد من إضافة أكبر ٢٥ مجموعة سكانية فقط إلى المجموع؟

- يجب على المطور إدراج marker value "قيمة علامة" في مصفوفات السكان. يجب إضافة "قيمة العلامة" في موضع الفهرس ٢٥ عند اكتشاف قيمة العلامة، يجب أن يتوقف التطبيق عن إضافة القيم إلى المجموع.
- يجب على المطور التحقق من رقم الفهرس لعنصر المصفوفة الحالي داخل الكتلة البرمجية foreach يجب أن يتوقف التطبيق عن إضافة قيم إلى المجموع عندما يصل رقم الفهرس إلى ٢٥
- يجب على المطور زيادة العداد داخل كتلة التعليمات البرمجية foreach يجب أن يتوقف التطبيق عن إضافة قيم إلى المجموع عندما يصل العداد إلى ٢٥

٢- يعمل مطور مع مطورين آخرين لتحديث مجموعة من التطبيقات، سيستخدم المطورون تعليقات التعليمات أثناء عملية التحديث. أي مما يلي يصف الاستخدام المناسب لتعليقات؟

- عند إجراء التحديثات، يستخدم المطورون تعليقات السطر لتحديد كل تحديث فردي للتعليمات البرمجية.
- عند إجراء التحديثات، يترك المطورون جميع تعليقات التعليمات البرمجية الموجودة سليمة، تتم إضافة تعليقات جديدة للإشارة إلى متى لم تعد التعليقات القديمة صالحة.
- عند إجراء التحديثات، يلخص المطورون التغييرات باستخدام التعليقات.

راجع إجابتك

١ يجب على المطور زيادة العداد داخل كتلة التعليمات البرمجية foreach يجب أن يتوقف التطبيق عن إضافة قيم إلى المجموع عندما يصل العداد إلى ٢٥

صحيح يجب أن يستخدم المطور عدادا يتزايد داخل الحلقة foreach

٢ عند إجراء التحديثات، يلخص المطورون التغييرات باستخدام التعليقات

صحيح يعد استخدام تعليق كتلة لتلخيص التغييرات التي تم تنفيذها أثناء التحديث استخداماً جيداً للتعليقات. غالباً ما يكون تعليق كتلة واحدة في أعلى الملفات التي تحتوي على تحديثات كافياً.

٦ الملخص

كان التحدي الذي تواجهه هو تحديث تطبيق بميزات جديدة استناداً إلى ملاحظات المستخدم.

من خلال تنفيذ التحديثات المطلوبة من المعلم بطريقة تحافظ على تكامل التطبيق الأصلي، تمكنت من تلبية طلب المستخدم بأفضل طريقة ممكنة.

يعد امتلاك القدرة على تفسير ملاحظات المستخدم، وتطبيقها بطريقة تتجنب عمليات إعادة التصميم الهامة، جزءاً مهماً من صيانة التطبيقات وتحديثها.

الفصل الثالث

إضافة منطق إلى تطبيقات وحدة تحكم C#

افحص العلاقة بين كتل التعليمات البرمجية ونطاق المتغير، وعمق تجربة ترميز C# باستخدام التعبيرات المنطقية، وعبارات التحديد، وعبارات التكرار.

المحتويات

الوحدة الأولي

تقييم التعبيرات الشرطية لاتخاذ القرارات في C#

Evaluate Boolean expressions

تعرف على العوامل والتقنيات المطلوبة لتقييم ومقارنة القيم في بيانات قرارك.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- استخدم عوامل التشغيل operators لإنشاء تعبيرات شرطية Boolean expressions تختبر المقارنة والمساواة.
- استخدم أساليب مضمنة built-in methods لفئة string لإجراء تقييمات أفضل على النصوص.
- استخدام عامل تشغيل النفي negation operator لاختبار نظير شرط معين.
- استخدم عامل التشغيل الشرطي conditional operator لإجراء تقييم مضمن.

محتويات الوحدة:

١- مقدمة

٢- تقييم تعبير

٣- تنفيذ عامل التشغيل الشرطي

٤- تمرين - إكمال نشاط التحدي باستخدام عوامل التشغيل الشرطية

٥- مراجعة الحل لنشاط تحدي عامل التشغيل الشرطي

٦- تمرين - إكمال نشاط التحدي باستخدام التعبيرات المنطقية

٧- مراجعة الحل لنشاط تحدي التعبيرات المنطقية

٨- اختبر معلوماتك

٩- الملخص

١ المقدمة

يعتمد منطق القرار على التعبيرات، المعروفة Boolean expressions بالتعبيرات المنطقية التي يتم تقييمها إلى "صحيح" أو "خطأ" يستخدم المطورون أنواعًا مختلفة من عوامل التشغيل لإنشاء تعبيرات منطقية تلبي متطلبات الترميز الخاصة بهم. عندما يتم تقييم التعبيرات، يتفرع تنفيذ التعليمات البرمجية بناءً على النتيجة، تدعم لغة C# مجموعة واسعة من العوامل (مثل المساواة والمقارنة والعوامل المنطقية)، يخدم كل منها غرضًا محددًا عند تنفيذ منطق القرار.

لنفترض أنه تم اختيارك للعمل على سلسلة من تطبيقات وحدة التحكم C# المستخدمة لمعالجة بيانات العميل، والمدخلات التي يوفرها المستخدم، يتطلب كل تطبيق تنفيذ منطق القرار الذي يحقق متطلبات معالجة البيانات، وقواعد العمل المرتبطة بها، تختلف متطلبات معالجة البيانات وقواعد العمل لكل تطبيق، على سبيل المثال، قد تحتاج التطبيقات التي تعالج طلبات العملاء إلى تقييم حالة العميل، قبل اتخاذ أي إجراء، للتحضير لهذه المهمة القادمة، سوف تكمل بعض الأنشطة التدريبية التي تطبق التعبيرات المنطقية، وعوامل تشغيل C#

في هذه الوحدة، ستتعرف على التعبيرات المنطقية، وتستخدم أنواعاً مختلفة من عوامل التشغيل، لتقييم التعبيرات من أجل المساواة، وعدم المساواة، والمقارنة، سوف تتعلم أيضاً استخدام إصدار مضمن خاص من عبارة if (عامل تشغيل شرطي) ينتج نتيجة إما / أو

في نهاية هذه الوحدة، يمكنك كتابة التعليمات البرمجية باستخدام أي مجموعة من عوامل تشغيل C# لتنفيذ منطق القرار في تطبيقاتك.

٢ تقييم تعبير Evaluate an expression

تمرين - تقييم تعبير

يستخدم منطق القرار Decision logic لإنشاء مسارات بديلة من خلال تعليماتك البرمجية، حيث يعتمد القرار حول المسار الذي يجب اتباعه على تقييم التعبير، على سبيل المثال، قد تكتب بعض التعليمات البرمجية التي تنفذ أحد المسارين بناءً على إدخال المستخدم، إذا أدخل المستخدم الحرف "a" فستنفذ التعليمات كتلة برمجية واحدة، إذا أدخل الحرف "b" فستنفذ التعليمات كتلة برمجية مختلفة.

في هذا المثال، أنت تتحكم في مسار التنفيذ، بناءً على القيمة المعينة لسلسلة ما. تحدد تعليماتك البرمجية مسار تنفيذ استنادًا إلى التعبير، وكيفية تقييم هذا التعبير، والمنطق الأساسي المستخدم لتحديد المسارات.

يعد فحص كيفية إنشاء تعبير وتقييمه مكاناً جيدا للبدء.

ما هو التعبير؟ expression

يقصد بالتعبير expression هو أي مجموعة من القيم (حرفية أو متغيرة) وعوامل وأساليب تُرجع قيمة واحدة. العبارة statements هي تعليمات كاملة في C# وتتكون العبارات من تعبير واحد أو أكثر، على سبيل المثال، تحتوي عبارة if التالية على تعبير واحد يرجع قيمة واحدة:

```
if (myName == "Luiz")
```

ربما كنت تعتقد أن القيمة التي يُرجعها التعبير ستكون رقمًا أو ربما كلمات، صحيح أن مطوري التطبيقات يستخدمون أنواعًا مختلفة من التعبيرات، لأغراض مختلفة، لكن في هذه الحالة، عندما تقوم بتطوير عبارة تحديد if ستستخدم تعبيرًا يُرجع إما صحيحًا أو خطأ `true` or `false` يشير المطورون إلى هذا النوع من التعبير على أنه تعبير منطقي، عندما تتضمن التعليمات البرمجية تعبيرًا منطقيًا، تكون قيمة الإرجاع دائمًا قيمة واحدة صحيحة أو خاطئة true or false

تعد التعبيرات المنطقية مهمة لأن تعليماتك البرمجية، يمكنها استخدام هذه التعبيرات لتحديد الكتلة البرمجية المراد تنفيذها.

هناك العديد من أنواع عوامل التشغيل المختلفة التي يمكنك استخدامها داخل تعبير منطقي، على سبيل المثال، تستخدم العبارة if أعلاه عامل تشغيل المساواة == للتحقق مما إذا تم تعيين قيمة معينة لمتغير نصي، سيعتمد عامل التشغيل الذي تختاره على مسارات التعليمات البرمجية المتوفرة، والشروط المقترنة بالمسارات، ومنطق التطبيق الأساسي.

تقييم المساواة وعدم المساواة Evaluating equality and inequality

أحد تقييمات التعليمات البرمجية الأكثر شيوعاً هو التحقق لمعرفة ما إذا كانت قيمتان متساويتين، عند التحقق من المساواة، ستحدد موقع عامل تشغيل المساواة == بين القيمتين اللتين يتم التحقق منهما، إذا كانت القيم الموجودة على أي من جانبي عامل تشغيل المساواة مكافئة، فسيرجع التعبير true وإلا فإنه سيعود false

وعلى العكس من ذلك، قد تحتاج أيضاً إلى التحقق مما إذا كانت قيمتان غير متساويتين، للتحقق من عدم المساواة، ستستخدم عامل عدم المساواة =! بين القيمتين.

قد تتساءل لماذا تحتاج إلى عوامل المساواة وعدم المساواة على حد سواء، سيصبح السبب أكثر وضوحًا عندما تتعلم كيفية إنشاء عبارات متفرعة، والبدء في كتابة تعليمة برمجية للعالم الحقيقي، يتيح لك العاملان اللذان يؤديان مهام متعارضة أن تكون أكثر تعبيرًا وإحكامًا.

حان الوقت الآن لإعداد بيئة الترميز، والبدء في كتابة التعليمات البرمجية التي تقيم التعبيرات المنطقية.

استخدام عامل تشغيل المساواة equality operator

١. تأكد من فتح Visual Studio Code وعرض Program.cs في قائمة المحرر.

ملاحظة

يجب أن يكون Program.cs فارغ، إذا لم يكن كذلك، فحدد جميع أسطر التعليمات البرمجية واحذفها.

٢. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
Console.WriteLine("a" == "a");
Console.WriteLine("a" == "A");
Console.WriteLine(1 == 2);

string myValue = "a";
Console.WriteLine(myValue == "a");
```

٣. في قائمة **File** حدد Visual Studio Code **Save**

يجب حفظ ملف Program.cs قبل بناء التعليمات البرمجية أو تشغيلها.

٤. في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

سيتم فتح لوحة Terminal يجب أن يظهر موجه الأوامر مسار موقع مجلد TestProject

٥. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ملاحظة

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. مثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن ترى الناتج التالي.

```
True
False
False
True
```

تحسين التحقق من مساواة النصوص string باستخدام الأساليب المساعدة المضمنة built-in helper methods في string

قد تتفاجأ بأن السطر `Console.WriteLine("a" == "A");` يخرج false عند مقارنة النصوص، الحالة مهمة.

أيضاً، ضع في اعتبارك هذا السطر من التعليمة البرمجية:

```
Console.WriteLine("a" == "a ");
```

قمت بإضافة مسافة في نهاية الجملة، سيؤدي هذا التعبير أيضاً إلى إخراج false

في بعض الحالات، قد يكون وجود مسافة قبل النص أو بعده مقبولاً تماماً، ومع ذلك، إذا كنت بحاجة إلى قبول تطابق غير دقيق، يمكنك أولاً تدليك البيانات "Massaging" تدليك البيانات يعني أنك تقوم ببعض التنظيف _الزيادة التي لا تريدها_ قبل إجراء مقارنة للمساواة.

على سبيل المثال، ضع في اعتبارك الحالة عند جمع إدخال المستخدم داخل حلقة، بعد إدخال كل قيمة، يمكنك تزويد المستخدم بمطالبة لتحديد ما إذا كان يريد المتابعة، مثل Do you want to continue (Y/N)? إذا أراد

المستخدم المتابعة، فمن المحتمل أن يدخل إما y, Y ستحتاج إلى تعليمات برمجية توضح أن كلا القيمتين متساوية، على الرغم من أن y لا تعادل Y

قبل التحقق من قيمتي النص للمساواة، خاصة عندما يتم إدخال قيمة واحدة أو كلتيهما من قبل مستخدم، يجب عليك:

- التأكد من أن كلا الجملتين جمعيهما أحرف كبيرة أو جمعيهما أحرف صغيرة باستخدام الأسلوب المساعد `ToUpper()`, `ToLower()` على أي قيمة نصية.
- قم بإزالة أي مسافات فارغة بادئة أو لاحقة باستخدام الأسلوب المساعد Trim() على أي قيمة نصية.

يمكنك تحسين التحقق من المساواة السابقة عن طريق ربط هاتين الطريقتين المساعدتين على كلتا القيمتين، كما هو موضح في قائمة التعليمات البرمجية التالية:

١. استبدل التعليمات البرمجية في محرر Visual Studio Code بالتعليمات التالية:

```
string value1 = " a";
string value2 = "A ";
Console.WriteLine(value1.Trim().ToLower() ==
value2.Trim().ToLower());
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

٣. لاحظ أنه عند تشغيل التعليمات البرمجية هذه المرة، فإنه يقوم بإخراج True

استخدام عامل تشغيل عدم المساواة inequality operator

١. استخدم عامل تعليق السطر // لتعليق تنفيذ كافة التعليمات البرمجية من الخطوة السابقة.

٢. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
Console.WriteLine("a" != "a");
Console.WriteLine("a" != "A");
Console.WriteLine(1 != 2);

string myValue = "a";
Console.WriteLine(myValue != "a");
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

يجب أن ترى الناتج التالي.

```
False
True
True
False
```

كما تتوقع، فإن النتيجة عند استخدام عامل عدم المساواة هي عكس ما رأيت عند استخدام عامل المساواة، وهذا يعني أن التعليمات البرمجية، ستتفرع بطريقة معاكسة أيضاً، والتي يمكن أن تكون بالضبط ما تريده.

تقييم المقارنات Evaluating comparisons

عند العمل مع أنواع البيانات الرقمية، قد تحتاج إلى تحديد ما إذا كانت القيمة أكبر أو أصغر من قيمة أخرى، استخدم عوامل التشغيل التالية لتنفيذ هذه الأنواع من المقارنات:

- أكبر من Greater than `>`
- أصغر من Less than `<`
- أكبر من أو يساوي Greater than or equal to `>=`

- أصغر من أو يساوي <= Less than or equal to

وبطبيعة الحال ستعمل عوامل التشغيل **!=** , **==** التي استخدمتها لمقارنة قيم النصوص أعلاه، أيضاً عند مقارنة أنواع البيانات الرقمية.

استخدام عوامل المقارنة Comparison operators

١. استخدم عامل تشغيل تعليق السطر // للتعليق على كافة التعليمات البرمجية من المهمة السابقة.

٢. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
Console.WriteLine(1 > 2);
Console.WriteLine(1 < 2);
Console.WriteLine(1 >= 1);
Console.WriteLine(1 <= 1);
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
False
True
True
True
```

الأساليب التي ترجع قيمة شرطية Methods that return a Boolean value

ترجع بعض الأساليب قيمة منطقية true أو false في التمرين التالي، ستستخدم أسلوباً مضمناً من الفئة String لتحديد ما إذا كانت جملة نصية تحتوي على كلمة أو عبارة معينة، مهمة لتطبيقك أم لا.

ملاحظة:

تحتوي بعض أنواع البيانات على أساليب تؤدي مهام أداة مساعدة مفيدة يحتوي نوع البيانات String على العديد من هذه، ترجع العديد منها قيمة شرطية بما في ذلك Contains(), StartsWith(), and EndsWith() يمكنك معرفة المزيد عنها في وحدة: معالجة البيانات الأبجدية الرقمية باستخدام أساليب فئة string في C#

استخدام أسلوب يرجع قيمة منطقية method that returns a Boolean

١. استخدم عامل تعليق السطر // لتعليق تنفيذ كافة التعليمات البرمجية من الخطوة السابقة.

٢. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio.

```
string exam = "The quick brown fox jumps over the lazy dog.";
Console.WriteLine(exam.Contains("fox"));
Console.WriteLine(exam.Contains("cow"));
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

يجب أن ترى الناتج التالي.

```
True
False
```

ما هو النفي المنطقي؟ logical negation

يشير مصطلح النفي المنطقي logical negation إلى عامل تشغيل ! الرفض الأحادي، يطلق بعض الأشخاص على عامل التشغيل هذا اسم "not operator" ليس عامل تشغيل. عند وضع عامل التشغيل ! قبل تعبير شرطي أو أي تعليمة برمجية يتم تقييمها إلى true , false يجبر التعليمات البرمجية على عكس تقييمها للمعامل. عند تطبيق النفي المنطقي، ينتج التقييم true إذا تم تقييم المعامل إلى false وينتج التقييم false إذا تم تقييم المعامل إلى true

فيما يلي مثال قد يساعدك على رؤية الاتصال بين هذه الأفكار، ينتج عن سطري التعليمات البرمجية التاليين نفس النتيجة، السطر الثاني هو أكثر إحكامًا.

```
// Console.WriteLine(exam.Contains("fox") == false);
// Console.WriteLine(!exam.Contains("fox"));
```

استخدام عامل تشغيل النفي المنطقي Logical Negation operator

١. استخدم عامل تعليق السطر // لتعليق تنفيذ كافة التعليمات البرمجية من الخطوة السابقة.

٢. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
string exam = "The quick brown fox jumps over the lazy dog.";
Console.WriteLine(!exam.Contains("fox"));
Console.WriteLine(!exam.Contains("cow"));
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

يجب أن ترى الناتج التالي.

```
False
True
```

عامل عدم المساواة inequality operator **مقابل النفي المنطقي** logical negation

يتضمن عامل عدم المساواة **!=** حرف ! ولكن لا ينبغي الخلط بينه وبين الرفض المنطقي، يعود عامل عدم المساواة true إذا لم تكن معاملاته متساوية، ويرجع false إذا كانت المعاملات متساوية، بالنسبة إلى معاملات الأنواع المضمنة، ينتج التعبير x != y نفس نتيجة التعبير !(x == y) مثال على الرفض المنطقي

يوضح نموذج التعليمات البرمجية التالي استخدام عامل التشغيل !=

```
int a = 7;
int b = 6;
Console.WriteLine(a != b); // output: True
string s1 = "Hello";
string s2 = "Hello";
Console.WriteLine(s1 != s2); // output: False
```

خلاصة

إليك المغزى الحقيقي الذي تعلمته عن تقييم التعبيرات الشرطية حتى الآن:

- هناك العديد من أنواع التعبيرات المختلفة التي يتم تقييمها إما true أو false
- تقييم المساواة باستخدام العامل ==
- يتطلب تقييم مساواة النصوص مراعاة إمكانية أن تكون للكلمات حالة مختلفة، ومسافات بادئة، أو لاحقة، اعتمادًا على الموقف الخاص بك، استخدم الأساليب المساعدة ToLower() or ToUpper() والأسلوب المساعد Trim() لتحسين احتمالية تساوي نصين.
- تقييم عدم المساواة باستخدام العامل =!
- قم بتقييم العوامل أكبر من وأقل من وما شابهها باستخدام عوامل المقارنة مثل <و >و =<و=>
- إذا أرجع الأسلوب شرطاً، يمكن استخدامه كتعبير شرطي Boolean expression
- استخدم عامل النفي المنطقي logical negation ! لتقييم نظير تعبير معين.

اختبر معلوماتك

١- أي مما يلي هو قائمة عوامل المقارنة الصالحة؟

- ==, ~=, >, <, >=, <=
- ==, <>, >, <, >=, <=
- ==, !=, >, <, >=, <=

٢- أي من أسطر التعليمات البرمجية التالية يستخدم النفي المنطقي؟

- `Console.WriteLine(myValue!= true);`
- `Console.WriteLine(!myValue);`
- `Console.WriteLine(myValue == false);`

٣- يحتوي التطبيق على متغيري نصي باسم myValue1 و myValue2 يدخل المستخدم القيمتين التاليتين لهذه المتغيرات "Y" و "y" أي من التعبيرات التالية ترجع false

- (myValue1!= myValue2)
- (myValue1.Trim().ToLower() != myValue2.Trim().ToLower())
- (myValue1.Trim().ToLower() == myValue2.Trim().ToLower())

راجع إجابتك

١ ==, !=, >, <, >=, and <=

صحيح جميعها عوامل المقارنة الصالحة.

٢

```
Console.WriteLine(!myValue);
```

صحيح إضافة عامل التشغيل ! قبل تعبير شرطي هو الطريقة الصحيحة لتنفيذ النفي أو الرفض المنطقي.

٣

```
(myValue1.Trim().ToLower() != myValue2.Trim().ToLower())
```

صحيح يرجع هذا التعبير false لأن القيمتين متساويتان ويتم استخدام عامل تشغيل عدم المساواة.

٣ تنفيذ عامل التشغيل الشرطي the conditional operator

لنفترض أنك بحاجة إلى تحديد ما إذا كانت عملية شراء العميل مؤهلة للحصول على خصم، تشير تفاصيل العرض الترويجي إلى أنه عندما تكون قيمة الشراء أكبر من ١٠٠٠ دولار، تكون عملية الشراء مؤهلة للحصول على خصم بقيمة ١٠٠ دولار، وإذا كان مبلغ الشراء ١٠٠٠ دولار أو أقل، فإن الشراء مؤهل للحصول على خصم ٥٠ دولاراً.

بينما يمكنك بالتأكيد استخدام بنية التفريع if ... elseif ... else للتعبير عن قاعدة العمل هذه، قد يكون استخدام The conditional operator عامل التشغيل الشرطي لتقييم الأهلية للخصم الترويجي خياراً أفضل. يستخدم العامل الشرطي تنسيقًا مضغوطًا يحفظ بضعة أسطر من التعليمات البرمجية، وربما يجعل الهدف من التعليمة البرمجية أكثر وضوحًا.

ما هو العامل الشرطي؟ The conditional operator

يقوم عامل التشغيل الشرطي `?:` بتقييم تعبير منطقي، وإرجاع إحدى نتيجتين استناداً إلى ما إذا كان التعبير المنطقي يقيم إلى صواب أو خطأ، يشار إلى عامل التشغيل الشرطي `?:` عادة باسم عامل التشغيل الشرطي الثلاثي the ternary conditional operator

هنا هو النموذج الأساسي:

```
<evaluate this condition> ? <if condition is true, return
this value> : <if condition is false, return this value>
```

من اليسار إلى اليمين:

`?` تقيم هذا الشرط (إذا كان صحيحاً نفذ هذه التعليمة فقط) `:` (إذا كان خطأ نفذ هذه التعليمة فقط)

خذ دقيقة للنظر في كيفية تطبيق عامل التشغيل الشرطي `?:` على الخصم الترويجي، هدفك هو عرض رسالة للعميل تعرض نسبة الخصم الخاصة به.

يعتمد مبلغ الخصم الخاص بهم على ما إذا كانوا قد أنفقوا أكثر من ١٠٠٠ دولار على الشراء.

إضافة التعليمات البرمجية التي تستخدم عامل تشغيل شرطي :?

١. اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
int saleAmount = 1001;
int discount = saleAmount > 1000 ? 100 : 50;
Console.WriteLine($"Discount: {discount}");
```

٢. في قائمة **File** حددVisual Studio Code **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

تلميح: ربما تعمل على أسم مجلد وأسم مشروع مختلفين، تأكد من استخدام الاسم والمسار الصحيحين.

يجب أن تكون قائمة Terminal التي تعرض موجه الأوامر مفتوحة، وأن يعرض موجه الأوامر مسار موقع مجلد TestProject

٤. في موجه الأوامرTerminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

عند تشغيل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
Discount: 100
```

عامل التشغيل الشرطي المضمن conditional operator inline

يمكنك ضغط هذه التعليمات البرمجية أكثر عن طريق إزالة المتغير `discount` المؤقت.

١. تحديث تعليماتك البرمجية في محرر Visual Studio كما يلي:

```
int saleAmount = 1001;
// int discount = saleAmount > 1000 ? 100 : 50;

Console.WriteLine($"Discount: {(saleAmount >
1000 ? 100 : 50)}");
```

٢. في قائمة **File** Visual Studio Code حدد **Save**

٣. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **run** dotnet ثم اضغط على Enter

٤. لاحظ أن الإخراج هو نفسه.

خذ دقيقة لفحص العبارة المحدثة `Console.WriteLine()`

من الضروري وضع عبارة عامل التشغيل الشرطي بالكامل بين أقواس. الأقواس تضمن أن وقت التشغيل يفهم هدفك، وهو عرض النتيجة الشرطية بدلا من نتيجة تقييم الشرط `(saleAmount > 1000)`

ملاحظة

على الرغم من أن هذا المثال محددًا ومضغوطًا ويظهر ما هو ممكن، إلا أنه أكثر صعوبة في القراءة. ليس من الجيد دائمًا دمج أسطر التعليمات البرمجية، خاصةً عندما يؤثر ذلك سلبًا على إمكانية قراءة التعليمات البرمجية بشكل عام. غالبًا ما يكون هذا بمثابة حكم شخصي.

خلاصة

يجب أن تتذكر الحقائق التالية حول عامل التشغيل الشرطي:

- يمكنك استخدام عامل التشغيل الشرطي لتقليل حجم تعليماتك البرمجية، ولكن يجب عليك التأكد من أن التعليمات البرمجية الناتجة يمكن قراءتها بسهولة.
- يمكنك استخدام عامل التشغيل الشرطي عندما تحتاج إلى إرجاع قيمة تستند إلى شرط ثنائي، سترجع التعليمات البرمجية الخيار الأول عند تقييم الشرط إلى صحيح، وسيعيد الخيار الثاني عند تقييم الشرط إلى خطأ.

اختبر معلوماتك

أي من سطور التعليمة البرمجية التالية يعد استخدامًا صالحًا للعامل الشرطي؟

- `int bonus = amount >= 100? 10: 5;`
- `int bonus = amount >= 100: 10? 5;`
- `int bonus = amount >= 100? 10, 5;`

```
int bonus = amount >= 100? 10: 5;
```

صحيح يعرض سطر التعليمات البرمجية هذا بناء الجملة الصحيح لعامل التشغيل الشرطي :?

٤ تمرين - إكمال نشاط التحدي باستخدام عوامل التشغيل الشرطية

ستعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

تحدي عامل التشغيل الشرطي

في هذا التحدي، ستقوم بتنفيذ عامل تشغيل شرطي لمحاكاة قلب العملة المعدنية، سيعرض منطق القرار الناتج إما وجه أو ظهر heads or tails في بعض البلدان تسمي "ملك وكتابة"

تحدي التعليمات البرمجية: كتابة التعليمات البرمجية لعرض نتيجة قلب العملة

فيما يلي متطلبات التحدي الخاصة بك:

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في **القائمة ملف**، حدد **فتح مجلد**.
- استخدم مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه.
- في قائمة استكشاف حدد **Program.cs**

٢. استخدم الفئة Random() لإنشاء قيمة.

ضع في اعتبارك نطاق الأرقام المطلوبة.

٣. استناداً إلى القيمة التي تم إنشاؤها، استخدم عامل التشغيل الشرطي لعرض إما heads or tails

يجب أن يكون هناك احتمال بنسبة ٥٠٪ أن تكون النتيجة إما وجه أو ظهر.

٤. يجب أن تكون التعليمات البرمجية سهلة القراءة، ولكن مع أقل عدد ممكن من الأسطر.

يجب أن تكون قادراً على تحقيق النتيجة المطلوبة في ثلاثة أسطر من التعليمات البرمجية.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٥ مراجعة الحل لنشاط تحدي عامل التشغيل الشرطي

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الوحدة السابقة.

```
Random coin = new Random();
int flip = coin.Next(0, 2);
Console.WriteLine((flip == 0) ? "heads" : "tails");
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن. يمكنك إزالة المتغير المؤقت flip عن طريق استدعاء Next() داخل التعبير المنطقي كما يلي:

```
Random coin = new Random();
Console.WriteLine((coin.Next(0, 2) == 0) ?
"heads" : "tails");
```

ومع ذلك، هذه التعليمة البرمجية مكثفة، ربما من الصعب فهم ما تفعله التعليمة.

إذا نجحت، فتهانينا! تابع التحدي الثاني في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٦ تمرين - إكمال نشاط التحدي باستخدام التعبيرات المنطقية

تحدي منطق القرار

في هذا التحدي، ستقوم بتنفيذ منطق القرار، تحدد قواعد العمل صلاحية الوصول الذي سيتم منحه للمستخدمين، استناداً إلى أذوناتهم المعتمدة على أدوارهم، ومستوى تصنيفهم المهني. تعرض فروع التعليمات البرمجية رسالة مختلفة للمستخدم استناداً إلى أذوناته ومستواه.

تهيئة قيم الأذونات والمستوى

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في **القائمة ملف** حدد File Open Folder **فتح مجلد**
- استخدم قائمة أو مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه.
- في قائمة EXPLORER حدد **Program.cs**
- في قائمة حدد **Selection** أختار تحديد الكل **Select All** ثم اضغط على مفتاح Delete لحذف التعليمات البرمجية الموجودة.

٢. اكتب التعليمات البرمجية التالية في محرر Visual Studio Code

```
string permission = "Admin|Manager";
int level = 55;
```

٣. راجع أسطر التعليمات البرمجية.

يستخدم تطبيقك مزيجاً من الأذونات والمستوى permission, level لتطبيق أو تقييم قواعد العمل، في سيناريو هذا التحدي، تحديد باقي شروط قواعد العمل في الخطوة التالية، يجب أن يستخدم الحل المكتمل permission, level

تلميح

لاختبار جميع المجموعات بشكل كافٍ للحصول على الإذن والمستوى الموضحين في قواعد العمل `permission, level` تحتاج إلى تعيين قيم إضافية لهذه المتغيرات، وتشغيل التطبيق عدة مرات

تنفيذ قواعد العمل

تحتاج إلى استخدام الأسلوب المساعد `Contains()` لتحديد ما إذا كانت القيمة المخصصة لمتغير الأذونات permission تحتوي على إحدى قيم الأذونات المحددة بواسطة "قواعد العمل" على سبيل المثال، التعبير Permission.Contains("Admin") يرجع true عند استخدام قيم البيانات الأولية المحددة في التعليمات.

فيما يلي قواعد العمل التي يجب أن يلبيها الحل الخاص بك:

١. إذا كان المستخدم Admin **مسؤولاً** بمستوى أعلى من ٥٥ فقم بإظهار الرسالة:

Welcome, Super Admin user

٢. إذا كان المستخدم Admin **مسؤولاً** بمستوى ٣٠ أو أعلى فقم بإظهار الرسالة:

Welcome, Admin user

٣. إذا كان المستخدم Manager **مديرًا** بمستوى ٢٠ أو أعلى، فقم بإظهار الرسالة:

Contact an Admin for access

٤. إذا كان المستخدم Manager **مديرًا** بمستوى أقل من ٢٠ فقم بإظهار الرسالة:

You do not have sufficient privileges

٥. إذا لم يكن المستخدم **مسؤولاً أو مديرًا**، فقم بإظهار الرسالة:

You do not have sufficient privileges…!

١. قم بتحديث التعليمات البرمجية Program.cs لاستيعاب كل من قواعد العمل.

٢. تحقق من التعليمات.

اختبار الحل باستخدام قيم البيانات الأولية المقترحة

١. إنشاء التعليمات البرمجية وتشغيلها.

٢. تقييم الإخراج.

عند تشغيل التعليمات البرمجية الخاصة بك، بما في ذلك بيانات التكوين الأولية، يجب أن تشاهد الإخراج التالي:

```
Welcome, Admin user
```

اختبار قواعد العمل الأخرى

١. تحديث القيم المعينة إلى `permission, level`

٢. حفظ التعليمات البرمجية وتشغيلها.

٣. قم بتقييم الإخراج للتحقق من استيفاء قواعد العمل الأخرى.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٧ مراجعة الحل لنشاط تحدي التعبيرات المنطقية

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
string permission = "Admin|Manager";
    int level = 10;

if (permission.Contains("Admin") && level > 55)
{
 Console.WriteLine("Welcome, Super Admin user.");
 }

else if (permission.Contains("Admin") && level >=
30)
{
 Console.WriteLine("Welcome, Admin user.");
 }

else if (permission.Contains("Manager") && level
>= 20)
{

        Console.WriteLine("Contact an Admin for
access.");

}
else if (permission.Contains("Manager") && level <
20)
{

        Console.WriteLine("You do not have
sufficient privileges");

}
else
{
    Console.WriteLine("You do not have sufficient
privileges...!");
}
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، لأنه في بعض الحالات من الممكن استخدام تعبيرات شرطية مختلفة للحصول على نفس النتيجة.

```
Welcome, Admin user
```

إذا نجحت، فتهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة

٨ اختبر معلوماتك

١- أي سطر من التعليمة البرمجية يستخدم النفي أو الرفض المنطقي؟

- `Console.WriteLine(!value);`
- `Console.WriteLine(value != false);`
- `Console.WriteLine(a != b);`

٢- أي من سطور التعليمة البرمجية التالية يعد استخدامًا صالحًا للعامل الشرطي؟

- `int value = amount >= 10? 10: 20;`
- `int value = amount >= 10: 10? 20;`
- `int value = amount >= 10? 10| 20;`

راجع إجابتك

١

```
Console.WriteLine(!value);
```

صحيح إضافة عامل التشغيل ! "not-operator" قبل تعبير شرطي، يفرض على التعليمات البرمجية عكس تقييمها للمعامل. عند تطبيق النفي أو الرفض المنطقي، ينتج التقييم true إذا تم تقييم معامل خطأ، false إذا تم تقييم معامل إلى صحيح

٢

```
int value = amount >= 10? 10: 20;
```

صحيح بناء جملة عامل تشغيل شرطي هو **"? (value 1) : (value 2)"**

٩ الملخص

كان هدفك هو تطوير بيانات القرار، لمختلف المواقف باستخدام التعبيرات المنطقية وعوامل تشغيل C#

باستخدام عوامل التشغيل والتقنيات المختلفة، كتبت التعليمات البرمجية التي قيمت المساواة، وقارنت القيم لمعرفة ما إذا كان أحدهما أكبر أو أقل من (أو يساوي) الآخر. قارنت الجمل، وقمت بتقليمها لإنشاء مقارنة أكثر دقة، تقضي على مشكلات مثل الإطار، والمسافات التي قد تتداخل مع النتيجة التي كنت تأمل فيها.

تعلمت كيفية استخدام النفي أو الرفض المنطقي لعكس قيمة للمقارنة، واستخدمت نتيجة الأساليب في التعبيرات المنطقية، وأخيراً، استخدمت عامل التشغيل الشرطي لتقييم الحالة أو شرط بإيجاز وإرجاع نتيجة.

بدون هذه المجموعة الكاملة من العوامل والتقنيات، ستكون محدودًا في أنواع بيانات القرار التي يمكنك إنشاؤها، يتطلب كل منطق الأعمال بعض التعبيرات المنطقية، أضفت أداة قيمة إلى صندوق أدوات البرمجة لديك.

الوحدة الثانية

التحكم في نطاق المتغير والمنطق باستخدام كتل التعليمات البرمجية في C#

استخدام كتل التعليمات البرمجية بدقة أكبر، ومزيد من الثقة، وفهم طريقة تأثيرها على رؤيتها، وإمكانية الوصول إلى البنيات ذات المستوى الأعلى والأدنى في التعليمات البرمجية.

الأهداف التعليمية

بعد إكمال هذه الوحدة، ستتمكن مما يلي:

- فهم تأثير الإعلان عن المتغيرات وتهيئتها داخل وخارج كتل التعليمات البرمجية.
- إزالة الأقواس المتعرجة {} من عبارات if لتحسين سهولة القراءة، عندما يكون هناك سطر واحد فقط، من التعليمات، داخل الكتلة البرمجية.
- وصف الغرض، والتسلسل الهرمي لنطاق مساحات الأسماء، والفئات والأساليب.

محتويات الوحدة:

١- مقدمة

٢- كتل التعليمات البرمجية ونطاق المتغير

٣- إزالة الأقواس المتعرجة من عبارات if

٤- تمرين - إكمال نشاط تحدي باستخدام نطاق المتغير

٥- مراجعة الحل لنشاط تحدي نطاق المتغير

٦- اختبر معلوماتك

٧- الملخص

١ المقدمة

تستخدم عبارات التحديد والتكرار Selection and iteration statements كتل التعليمات البرمجية لتجميع أسطر التعليمات التي يجب تنفيذها، أو تخطيها، أو تكرارها، ولكن هذا ليس الغرض الوحيد من كتل التعليمات البرمجية، أيضاً تستخدم كتل التعليمات البرمجية للتحكم في إمكانية الوصول، والصلاحيات المتغيرة أو تقييدها.

يشير النطاق المتغير "Variable "scope إلى جزء التطبيق الذي يمكن الوصول إلى المتغير داخله، يعد فهم كيفية تأثير كتلة التعليمات البرمجية على نطاق المتغير جزءاً مهماً من برمجة الكمبيوتر.

لنفترض أنك تعمل على تطبيق كبير، يستخدم عبارات التكرار والتحديد المتداخلة لمعالجة بيانات المصفوفة، يستخدم تطبيقك متغيرات للمساعدة في إنجاز المهام الشائعة عبر التطبيق، تخدم بعض المتغيرات نفس الغرض في أجزاء مختلفة من التطبيق، قمت ببعض المحاولات لإعادة استخدام أسماء المتغيرات، مع نمو تطبيقك، تبدأ في رؤية نتائج غير متوقعة للعمليات الحسابية، والأخطاء التي تُبلغ عن متغير غير مهيأ أو غير موجود، أنت بحاجة إلى تحسين النهج الذي تستخدمه للإعلان عن المتغيرات، والوصول إليها، وتحتاج إلى تحسين فهمك لنطاق المتغير.

في هذه الوحدة، ستقوم بتعريف المتغيرات للاستخدام داخل حدود كتل التعليمات البرمجية وخارجها، وستزيل أقواس كتل التعليمات البرمجية في حالات معينة، لجعلها أكثر سهولة في القراءة، ستتعلم كيف تؤثر كتل التعليمات البرمجية على إمكانية الوصول إلى المتغيرات، ورؤيتها.

في نهاية هذه الوحدة، ستتمكن من استخدام الكتل البرمجية بمزيد من الثقة، وفهم كيفية تأثيرها على رؤية التعليمات البرمجية، وإمكانية الوصول إليها.

٢ كتل التعليمات البرمجية ونطاق المتغير Code blocks and variable scope

تعد كتلة التعليمات البرمجية code block عبارة واحدة أو أكثر تحدد مسار التنفيذ، تؤثر العبارات الموجودة خارج الكتلة البرمجية على موعد تنفيذ هذه الكتلة، وعدد المرات التي يتم تنفيذها وقت التشغيل، عادة ما يتم تعريف حدود كتلة التعليمات البرمجية بواسطة أقواس متعرجة {}

بالإضافة إلى تأثيرها على مسار التنفيذ، يمكن أن تؤثر كتل التعليمات البرمجية أيضاً على نطاق المتغيرات، ستساعدك نماذج التعليمات التي تفحصها أثناء هذا التمرين على فهم العلاقة بين كتل التعليمات البرمجية ونطاق المتغير.

تأثير كتل التعليمات البرمجية في نطاق تعريف المتغير

يشير نطاق المتغير Variable scope إلى إمكانية رؤية المتغير للتعليمات البرمجية الأخرى داخل التطبيق، يمكن الوصول إلى متغير محلي النطاق فقط داخل الكتلة التي تم تعريفه داخلها، وإذا حاولت الوصول إلى المتغير خارج كتلة التعليمات البرمجية، فستحصل على خطأ في برنامج التحويل البرمجي.

يستكشف بقية هذا الدرس العلاقة بين كتل التعليمات البرمجية ونطاق المتغير.

إعداد بيئة الترميز الخاصة بك

تتضمن هذه الوحدة أنشطة عملية ترشدك خلال إنشاء التعليمات البرمجية التوضيحية وتشغيلها، نشجعك على إكمال هذه الأنشطة باستخدام Visual Studio Code كبيئة تطوير. سيساعدك استخدام Visual Studio Code على أن تصبح أكثر راحة في كتابة التعليمات، وتشغيلها في بيئة تطوير يستخدمها المحترفون في جميع أنحاء العالم.

سوف تستخدم مشروع وحدة تحكم console لإنشاء نماذج التعليمات البرمجية، وبنائها، وتشغيلها أثناء هذا الدرس.

إنشاء متغير داخل كتلة التعليمات البرمجية

ستبدأ بالنظر إلى الحالة عند تهيئة متغير داخل كتلة التعليمات البرمجية.

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في المحرر.

يجب أن يكون Program.cs فارغ، إذا لم يكن كذلك، فحدد جميع أسطر التعليمات البرمجية واحذفها.

٢. اكتب التعليمات البرمجية التالية في محرر Visual Studio Code

```
bool flag = true;
if (flag)
{
    int value = 10;
    Console.WriteLine($"Inside the code block:
{value}");
}
```

٣. في قائمة ملف **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل بناء التعليمات البرمجية أو تشغيلها.

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

تلميح: ربما تعمل على أسم مجلد وأسم مشروع مختلفين، تأكد من استخدام الاسم والمسار الصحيحين.

يجب أن تكون قائمة Terminal التي تعرض موجه الأوامر مفتوحة، وأن يعرض موجه الأوامر مسار موقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ملاحظة

إذا رأيت رسالة تقول "تعذر العثور على المشروع لتشغيله" تأكد من أن موجه الأوامر Terminal يعرض مسار موقع المجلد المتوقع. على سبيل المثال: C:\Users\someuser\Desktop\csharpprojects\TestProject

عند تشغيل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
Inside the code block: 10
```

هذا هو الإخراج المتوقع، ولكن ماذا لو كنت تريد الوصول إلى المتغير value خارج كتلة التعليمات البرمجية للعبارة if

محاولة الوصول إلى متغير خارج كتلة التعليمات البرمجية التي تم الإعلان عنه داخلها.

١. في محرر التعليمات البرمجية Visual Studio قم بإنشاء سطر فارغ، أسفل الكتلة البرمجية للعبارة if

٢. في السطر الفارغ الذي أنشأته، أضف التعليمات البرمجية التالية:

```
Console.WriteLine($"Outside the code block:
{value}");
```

٣. تأكد أن التعليمات البرمجية المحدثة تبدو كما يلي:

```
bool flag = true;
if (flag)
{
    int value = 10;
    Console.WriteLine($"Inside the code block:
{value}");
}
Console.WriteLine($"Outside the code block:
{value}");
```

٤. احفظ ملف التعليمات البرمجية، ثم شغلها.

٥. لاحظ أنه عند محاولة تشغيل التطبيق، تحصل على خطأ في التحويل البرمجي:

```
Program.cs(7,46): error CS0103: The name
'value' does not exist in the current context
```

يخبرك `Program.cs(7,46)` جزء الرسالة أن الخطأ مقترن بالسطر 7 في ملف Program.cs الحرف 47

يتم إنشاء هذا الخطأ لأن المتغير الذي تم الإعلان عنه داخل كتلة التعليمات البرمجية يمكن الوصول إليه فقط (يمكن رؤيته) داخل هذه الكتلة. نظراً لأنه لا يمكن الوصول إلى متغير خارج الكتلة التي تم الإعلان عنه فيها، لا يمكن الوصول إلى value من السطر 7 من التعليمات البرمجية.

يشار إلى المتغير الذي تم الإعلان عنه داخل كتلة التعليمات البرمجية كمتغير داخلي أو محلي local variable قد ترى مصطلح المتغير المحلي المستخدم عند مراجعة المقالات التي تناقش نطاق المتغير.

نقل إعلان المتغير فوق كتلة التعليمات البرمجية

للوصول إلى متغير داخل كتلة التعليمات البرمجية وخارجها، ستحتاج إلى تعريف المتغير قبل (أعلى) الكتلة، بحيث يمكن للتعليمات البرمجية خارج الكتلة رؤية المتغير، والوصول إليه.

١. تحديث التعليمات البرمجية في المحرر التعليمات البرمجية كما يلي:

```
bool flag = true;
int value;

if (flag)
{
    Console.WriteLine($"Inside the code block:
{value}");
}
```

```
value = 10;
Console.WriteLine($"Outside the code block:
{value}");
```

٢. خذ دقيقة لمراجعة التحديثات.

٣. لاحظ الإعلان عن value (**ولكن لم تتم تهيئته**) خارج كتلة التعليمات البرمجية if

٤. استخدم Visual Studio Code لحفظ التحديثات، ثم قم بتشغيل التعليمات.

٥. لاحظ أنك لا تزال تحصل على خطأ في التحويل البرمجي.

هذه المرة، عند محاولة تشغيل التطبيق، تحصل على خطأ التحويل البرمجي التالي:

```
Program.cs(6,49): error CS0165: Use of unassigned local
variable 'value'
```

نظرا لأن value لم تتم تهيئته عند الإعلان عنه، لا يمكن الوصول إليه داخل كتلة التعليمات البرمجية.

يرتبط الخطأ بالسطر 6 داخل كتلة التعليمات البرمجية لأن المتغير value غير مهيأ (**لم يتم تعيين قيمة له**) إذا كان سطر التعليمات value = 10; موجوداً فوق كتلة عبارة if فسيتم التشغيل بشكل صحيح.

سيؤدي التأكد من تهيئة المتغيرات قبل محاولة الوصول إليها إلى معالجة هذه المشكلة.

تهيئة متغير كجزء من إعلان المتغير

لتهيئة value كجزء من إعلان المتغير، قم بتعديل تعليماتك البرمجية كما يلي:

```
bool flag = true;
int value = 0;

if (flag)
{
    Console.WriteLine($"Inside the code block:
{value}");
}

value = 10;
Console.WriteLine($"Outside the code block:
{value}");
```

تعالج هذه التعليمة البرمجية خطأ التحويل البرمجي "المتغير غير المهيأ أو المعين" عن طريق تهيئة value كجزء من إعلان المتغير.

١. استخدم Visual Studio Code لحفظ التعليمات البرمجية وتشغيلها.

٢. لاحظ الآن، عند تشغيل التطبيق، سترى الإخراج التالي:

```
Inside the code block: 0
Outside the code block: 10
```

فحص تفسير المترجم "المحول البرمجي" لتعليماتك البرمجية

لتجنب أخطاء وقت التشغيل، يحلل المترجم أو المحول البرمجي C# التعليمات البرمجية عند كتابتها في محرر Visual Studio Code وأثناء عملية الإنشاء، ومع ذلك، قد لا يفسر المحول البرمجي التعليمات البرمجية دائماً، بنفس الطريقة التي تقوم بها.

خذ بعين الاعتبار نموذجي التعليمات البرمجية التاليين اللذين يبدو أنهما يخدمان نفس الغرض:

```
// Code sample 1
bool flag = true;
int value;

if (flag)
{
    value = 10;
    Console.WriteLine($"Inside the code block:
{value}");
}

Console.WriteLine($"Outside the code block:
{value}");


// Code sample 2
int value;

if (true)
{
    value = 10;
    Console.WriteLine($"Inside the code block:
{value}");
}

Console.WriteLine($"Outside the code block:
{value}");
```

قد تشعر أن هذين النموذجين يجب أن ينتجا دائما نفس النتيجة، ولكن المحول البرمجي C# يفسر هذين النموذجين من التعليمات البرمجية بشكل مختلف.

بالنسبة لعينة التعليمات البرمجية الأولى يفسر المحول البرمجي flag كمتغير منطقي يمكن تعيين قيمة له إما صواب أو خطأ true or false

يستنتج المحول البرمجي المسار الأول أن قيمة flag احتمالًا false فلن تتم تهيئة المتغير عند تنفيذ عبارة Console.WriteLine() الثانية بشكل أساسي، فيعطي خطأ وقت التنفيذ.

يستنتج المحول البرمجي المسار الثاني أن قيمة flag احتمالًا true فإنه ينشئ رسالة خطأ أثناء عملية البناء، بالإضافة إلى ذلك، يحذرك محرر التعليمات البرمجية في Visual Studio Code من هذه المشكلة عن طريق عرض خط أحمر متعرج أسفل أسم المتغير.

بالنسبة لنموذج التعليمات البرمجية الثاني، يستنتج المحول أن محتويات كتلة تعليمات برمجية if سيتم تنفيذها true (صحيح دائمًا) لا ينشئ المحول خطأ في البنية، لأنه يفسر هذا النموذج من التعليمات ليكون له مسار تنفيذ واحد كما يلي:

```
int value;
value = 10;
Console.WriteLine($"Inside the code block: {value}");
Console.WriteLine($"Outside the code block:
{value}");
```

خلاصة

فيما يلي بعض الأمور المهمة التي يجب تذكرها حول كتل التعليمات البرمجية:

١. عند الإعلان عن متغير داخل الكتلة البرمجية، تصبح رؤيته داخل هذه الكتلة فقط، ولا يمكن الوصول إلى هذا المتغير خارجها.

٢. للتأكد أن المتغير مرئي داخل الكتلة البرمجية وخارجها، يجب تعريف المتغير في الخارج، وقبل الكتلة البرمجية.

٣. تأكد من تهيئة المتغيرات قبل أن تحاول استدعائها أو الوصول إليها، لجميع مسارات تنفيذ التعليمات البرمجية المحتملة.

اختبر معلوماتك

يكتب مطور بعض التعليمات البرمجية التي تتضمن عبارة if يقوم بتهيئة متغير أول integer أعلى الكتلة البرمجية وخارجها، يعين له قيمة ٥

داخل الكتلة البرمجية، يقوم بتهيئة متغير integer ثاني يعين له قيمة 6 يتم تقييم التعبير المنطقي للكتلة إلى true إذا كان المتغير الأول له قيمة أكبر من 0 يقوم بتعيين مجموع قيمتي المتغيرين، إلى المتغير الأول.

بعد الكتلة البرمجية، خارجها، يكتب التعليمات البرمجية لعرض قيمة المتغير الأول، ما هي النتيجة عند تنفيذ عبارة التعليمات المستخدمة لعرض قيمة المتغير الأول؟

- لن يتم إنشاء أي خطأ، وسيتم عرض قيمة المتغير الأول، القيمة المعروضة هي مجموع المتغيرين الأول والثاني.
- لن يتم إنشاء أي خطأ، وسيتم عرض قيمة المتغير الأول، القيمة المعروضة هي القيمة التي تمت تهيئتها أعلى الكتلة البرمجية.
- يتم إنشاء خطأ لأن المتغير الأول ليس في النطاق بعد الكتلة البرمجية.

الإجابة الصحيحة

لن يتم إنشاء أي خطأ، وسيتم عرض قيمة المتغير الأول، القيمة المعروضة هي مجموع المتغيرين الأول والثاني.

صحيح نظراً لتهيئة المتغير الأول فوق كتلة العبارة if فإنه لا يزال في النطاق بعد الكتلة، أيضاً، لأن كلا المتغيرين في النطاق، وتهيئتهم مع قيم، يتم تنفيذ إضافة القيم بشكل صحيح داخل الكتلة البرمجية، وأخيراً، على الرغم من أن المتغير الثاني غير موجود خارج الكتلة، احتفظ المتغير الأول بالتغييرات التي حدثت على قيمته داخل الكتلة.

٣ إزالة كتل التعليمات البرمجية من عبارات if

يحب مطوري البرامج ذلك، كتابة تعليمات برمجية توفر ضغطات المفاتيح، والمساحة المرئية، دون إخلال بإمكانية القراءة، طبق عبارة **الأقل هو الأكثر**، خلال عملية التطوير، لجعل تعليماتك البرمجية أكثر قابلية للقراءة والفهم.

إذا كانت كتلة التعليمات البرمجية تحتاج سطر واحد فقط من التعليمات، فأنت لا تحتاج إلى تحديد الكتلة باستخدام أقواس متعرجة.

على الرغم من الناحية الفنية لا تحتاج حتى إلى فصل التعليمات البرمجية إلى أسطر متعددة، فإن الجمع بين عبارات على سطر واحد يمكن أن يجعل تعليماتك البرمجية صعبة القراءة.

إزالة الأقواس المتعرجة هو تغيير نمطي، لا ينبغي أن يؤثر على وظيفة التعليمات البرمجية، ومع ذلك، يجب عليك اتخاذ خطوات للتأكد من أن تغييراتك لا تؤثر سلباً على مدى سهولة قراءة التعليمات، يمكنك تقييم تأثير إزالة الأقواس المتعرجة والمساحة البيضاء، ثم العودة إلى التعليمات البرمجية الأصلية، إذا وجدت أن التغييرات جعلت التعليمات البرمجية صعبة القراءة.

إنشاء مثال على عبارة if التي تستخدم كتلة التعليمات البرمجية

١. اكتب التعليمات البرمجية التالية في محرر Visual Studio Code

```
bool flag = true;
if (flag)
{
    Console.WriteLine(flag);
}
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

أدخل dotnet run في موجه الأوامر Terminal لتشغيل التعليمات البرمجية.

٣. تحقق من رؤية الإخراج التالي:

```
True
```

تمثل هذه التعليمات نقطة بداية جيدة، ولكن لديك كتلة تعليمات برمجية تتضمن سطراً واحداً من التعليمات، في هذه الحالة، هل تعريف كتلة التعليمات البرمجية ضروري؟

إزالة الأقواس المتعرجة

تنفذ كتلة التعليمات البرمجية في الأعلى، عندما تكون flag قيمتها true لأن هذه الكتلة تحتوي على سطر تعليمة برمجية واحد، فلديك خيار إزالة الأقواس المتعرجة.

١. تحديث التعليمات البرمجية في محرر Visual Studio كما يلي:

```
bool flag = true;
if (flag)
   Console.WriteLine(flag);
```

لا تؤدي إزالة الأقواس المتعرجة إلى تغيير حقيقة أن `Console.WriteLine(flag);` هي الكتلة البرمجية للعبارة if

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

٣. لاحظ أن الإخراج هو نفسه كما كان من قبل.

```
True
```

وفرت سطران من التعليمات البرمجية، والأهم من ذلك، لا تزال التعليمات البرمجية تحت العبارة if تقرأ بسهولة.

فحص قابلية قراءة نموذج من سطر واحد لعبارات if

في هذه الخطوة، ستنظر في حالة قد تتأثر فيها سهولة قراءة التعليمات البرمجية بالسلب.

نظراً لأن كل من العبارة if واستدعاء الأسلوب Console.WriteLine() قصيران، فقد تميل إلى دمجهما في سطر واحد، يسمح لك بناء جملة if في C# بهذه الطريقة لدمج العبارات.

١. عدل التعليمات البرمجية في المحرر كما يلي:

```
bool flag = true;
if (flag) Console.WriteLine(flag);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

٣. لاحظ أن الإخراج لا يزال كما هو.

٤. خذ دقيقة للنظر في إمكانية قراءة التعليمات البرمجية.

تخيل نموذجي التعليمات السابقين المتداخلين، داخل قسم أكبر من التعليمات البرمجية، قد يؤدي الجمع بين العبارات (كما فعلت) إلى صعوبة القراءة.

عند تنفيذ عبارة if تضم كتلة تعليمات من سطر واحد، توصي Microsoft بمراعاة هذه المبادئ:

- لا تستخدم أبدا نموذج سطر واحد، مثل:

```
if (flag) Console.WriteLine(flag);
```

- يفضل استخدام الأقواس دائماً، وضروري إذا كانت أي كتلة من عبارة if/else if/.../else المركبة تستخدم أقواساً، أو إذا كان نص العبارة الواحدة يمتد عبر أسطر متعددة.
- لا يجوز حذف الأقواس الا إذا تم وضع نص أي كتلة مرتبطة بعبارة if/else if/.../else على سطر واحد.

٥. لفحص تأثير سهولة قراءة عبارات if-elseif-else قم بتعديل التعليمات البرمجية كما يلي:

```
string name = "ahmed";
if (name == "bob") Console.WriteLine("Found
Bob");
else if (name == "ahmed")
Console.WriteLine("Found ahmed");
else Console.WriteLine("Found Ali");
```

٦. احفظ ملف التعليمات البرمجية، ثم شغلها.

عند تشغيل التعليمات البرمجية، يجب أن تحصل على الإخراج التالي:

```
Found ahmed
```

تم تشغيل التعليمات البرمجية، ولكن هذه الأسطر من التعليمات كثيفة ويصعب قراءتها، تحتاج إلى إعادة تهيئة التعليمات البرمجية لإضافة فاصل أسطر بعد العبارات if, else if, and else

٧. قارن التعليمات البرمجية التي قمت بتشغيلها للتو، مع التعليمات البرمجية التالية:

```
string name = "ahmed";

if (name == "bob")
    Console.WriteLine("Found Bob");
else if (name == "ahmed")
    Console.WriteLine("Found ahmed");
else
    Console.WriteLine("Found Ali");
```

لاحظ مدى سهولة قراءة نموذج التعليمات البرمجية الثاني.

خلاصة

فيما يلي بعض الأشياء المهمة التي يجب تذكرها حول الكتل البرمجية لعبارات if وسهولة القراءة:

- إذا كان لديك سطراً واحداً فقط من التعليمات، ضمن كتل برمجية لإحدى العبارات if-elseif-else يمكنك إزالة الأقواس المتعرجة لهذه الكتلة البرمجية والمسافة البيضاء.
- توصي Microsoft باستخدام الأقواس المتعرجة بشكل متناسق لجميع كتل التعليمات البرمجية لعبارات if-elseif-else (إما تكتبها أو تزيلها دائماً)
- إزال الأقواس المتعرجة من الكتلة البرمجية فقط لجعل التعليمات البرمجية أكثر قابلية للقراءة، من المقبول دائماً استخدام أقواس متعرجة.
- لا تدمج التعليمات على سطر واحد، الا إذا رأيت أن ذلك يجعلها سهلة القراءة. ترى Microsoft أن التعليمات البرمجية تكون أكثر قابلية للقراءة عند كتابة كل عبارة على السطر الخاص بها.

٤ تمرين - إكمال نشاط تحدي باستخدام نطاق متغير

تحدي نطاق المتغير

في هذا التحدي، ستستخدم ما تعلمته حول كتل التعليمات البرمجية، ونطاق المتغير، لإصلاح نموذج التعليمات المكتوبة بشكل سيئ، هناك العديد من التحسينات التي يمكنك إجراؤها. حظ سعيد!

تحدي التعليمات البرمجية: تصحيح أخطاء تعليمات برمجية

١. اكتب التعليمات البرمجية التالية في المحرر

```
int[] numbers = { 4, 8, 15, 16, 23, 42 };

foreach (int number in numbers)
{
    int total;

    total += number;

    if (number == 42)
    {
       bool found = true;

    }

}

if (found)
{
    Console.WriteLine("Set contains 42");

}

Console.WriteLine($"Total: {total}");
```

٢. راجع الإخراج المطلوب.

عند الانتهاء من عمليات التصحيح المطلوبة، يجب أن ينتج التطبيق الإخراج التالي:

```
Set contains 42
Total: 108
```

٣. أكمل تعديل التعليمات البرمجية المطلوبة، بحيث تنتج العبارات Console.WriteLine() الإخراج المطلوب.

قد تحتاج إلى تعديل نطاق المتغير.

٤. تحسين سهولة قراءة التعليمات.

اعتماداً على مقدار المسافة البيضاء التي تقوم بإضافتها، وبعض العوامل الأخرى، يجب أن يكون لديك حوالي ١٧ سطراً.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٥ مراجعة الحل لنشاط تحدي نطاق المتغير

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الوحدة السابقة:

```
int[] numbers = { 4, 8, 15, 16, 23, 42 };
int total = 0;
bool found = false;

foreach (int number in numbers)
{
    total += number;
    if (number == 42)
        found = true;
}

if (found)
    Console.WriteLine("Set contains 42");

Console.WriteLine($"Total: {total}");
```

يعد هذا الرمز مجرد حل واحد ممكن، لأنك أختصر الأسطر في مناطق مختلفة، ونسقت التعليمات البرمجية تنسيقًا مختلفًا.

تضمنت أكبر التغييرات على أخطاء التعليمات البرمجية:

- نقل إعلان المتغيرين `total`, `found` خارج عبارة foreach
- تهيئة كل من المتغيرين `total`, `found` بقيم افتراضية معقولة.
- إزالة الأقواس المتعرجة من كتل التعليمات البرمجية من عبارات if

بغض النظر عن كيفية تنسيق التعليمات، يجب أن تشاهد الإخراج التالي:

```
Set contains 42
Total: 108
```

٦ اختبر معلوماتك

١- أي من العبارات التالية صحيحة حول إظهار أو إزالة الأقواس المتعرجة لكتل التعليمات البرمجية لعبارة if؟

- لا يمكن إزالة الأقواس المتعرجة من كتلة عبارات if else, else
- إذا تمت إزالة الأقواس المتعرجة من كتل if-elseif-else يجب أيضاً إزالة المسافة البيضاء.
- اختر دائما نمطاً يحسن من سهولة القراءة.

٢- يقوم مطور بكتابة بعض التعليمات، التي تضم كتلة عبارة if وتهيئة متغير integer عدد صحيح أول، أعلى (خارج) الكتلة البرمجية، قيمته **1** يتم تقييم التعبير المنطقي لعبارة if إلى true إذا كان المتغير الأول قيمة أكبر من 0 وتهيئة متغير integer عدد صحيح ثانٍ قيمته 8 في السطر الأول داخل الكتلة if في السطر الثاني داخل الكتلة، تعيين مجموع القيمتين إلى المتغير الأول. بعد الكتلة if تعرض تعليمة قيمة المتغير الصحيح الأول. ما هي النتيجة عند تنفيذ الكود؟

- لا يتم إنشاء أي خطأ ويتم عرض قيمة المتغير الأول ٩
- لم يتم إنشاء أي خطأ ويتم عرض قيمة المتغير الأول ١
- يتم إنشاء خطأ لأن المتغير الأول ليس في النطاق بعد كتلة التعليمات البرمجية

٣- يكتب مطور بعض التعليمات البرمجية التي تضم كتلة برمجية لعبارة if وتهيئة متغير integer عدد صحيح أول أعلى (خارج) الكتلة، قيمته 5 تم تقييم التعبير المنطقي لكتلة عبارة if إلى true إذا كان المتغير الأول قيمته أكبر من 0 وتهيئة متغير integer عدد صحيح ثانٍ قيمته 6 في السطر الأول داخل كتلة عبارة if بعد الكتلة تم استخدام السطر الأول لإضافة قيمة المتغير الثاني إلى قيمة المتغير الأول، والسطر الثاني لعرض قيمة المتغير الأول. ما هي النتيجة عند تشغيل التعليمات البرمجية؟

- لا ينشئ أي خطأ ويتم عرض قيمة المتغير الأول ١١
- يتم إنشاء خطأ بواسطة سطر التعليمات البرمجية المستخدم لعرض قيمة المتغير الأول
- يتم إنشاء خطأ بناء بواسطة سطر التعليمات البرمجية المستخدم لجمع المتغيرين

راجع إجابتك

١ اختر دائما نمطاً يحسن قابلية القراءة.

صحيح يجب أن تكون سهولة قراءة التعليمات البرمجية دائماً أحد الاعتبارات عند تحديد ما إذا كنت تريد إزالة الأقواس المتعرجة من كتل التعليمات البرمجية المقترنة بعبارة if

٢ لا يتم إنشاء أي خطأ ويتم عرض قيمة المتغير الأول ٩

صحيح نظراً لتهيئة المتغير الأول فوق عبارة if يزال في النطاق بعد كتلة التعليمات البرمجية، أيضاً، لأن كلا المتغيرين في النطاق وتهيئتهما مع قيم داخل الكتلة، تنفذ إضافة القيم بشكل صحيح، وأخيراً، على الرغم من أن المتغير الثاني غير موجود خارج الكتلة البرمجية، لكن احتفظ المتغير الأول بالتغييرات على قيمته التي حدثت داخل الكتلة البرمجية.

٣ يتم إنشاء خطأ بناء بواسطة سطر التعليمات البرمجية المستخدم لجمع المتغيرين

صحيح يتم إنشاء خطأ بناء في التعليمات البرمجية التي تجمع المتغيرين، الخطأ بسبب المتغير الثاني غير ممكن الوصول إليه في السياق الحالي. المتغير الثاني ليس في النطاق خارج الكتلة البرمجية.

٧ الملخص

كان هدفك هو فهم كيفية تأثير التعليمات البرمجية على الوصول إلى المتغيرات، والإجراءات المطلوبة لضمان تحديد نطاق المتغيرات وتهيئتها، والوصول إليها بشكل مناسب عند الحاجة داخل التطبيق.

لقد استكشفت تأثير الإعلان عن المتغيرات داخل حدود كتلة التعليمات البرمجية وخارجها، فحصت أيضاً تأثير تهيئة المتغير على إمكانية الوصول، بالإضافة إلى كيفية تأثير إزالة الأقواس المتعرجة لكتل التعليمات البرمجية غير الضرورية، على سهولة قراءة التعليمات البرمجية.

تؤدي التعليمات البرمجية التي تستخدم متغيرات محددة النطاق أو التي تمت تهيئتها بشكل غير صحيح إلى حدوث أخطاء في وقت التشغيل، تضمن قدرتك على تحديد نطاق المتغيرات، وتهيئتها بشكل صحيح، تجربة استخدام أفضل مع تطبيقاتك.

الوحدة الثالثة

تقسيم تدفق التعليمات البرمجية باستخدام بنية switch-case في C#

تعرّف على كيفية إضافة منطق التفريع، الذي يطابق متغيرًا واحدًا، أو تعبيرًا واحدًا مقابل العديد من القيم المحتملة.

الأهداف التعليمية

خلال هذه الوحدة، سوف تتمكن مما يلي:

- استخدم البنية switch-case لمطابقة متغير أو تعبير مقابل العديد من النتائج المحتملة.
- تحويل التعليمات البرمجية التي تستخدم عبارات if-elseif-else إلى عبارات switch-case

محتويات الوحدة:

١- مقدمة

٢- تنفيذ عبارة switch-case

٣- تمرين - إكمال نشاط تحدي باستخدام عبارات switch-case

٤- مراجعة الحل لنشاط تحدي عبارة switch-case

٥- اختبر معلوماتك

٦- الملخص

١ المقدمة

تشبه لغة البرمجة C# أي لغة بشرية مكتوبة أو منطوقة، كل منهم يدعم طرقاً مختلفة للتعبير عن نفس الفكرة، في اللغات المنطوقة، تكون بعض الكلمات والعبارات وصفية، أو دقيقة، أو مختصرة أكثر من غيرها، في لغة البرمجة C# هناك أكثر من طريقة واحدة لإنشاء منطق تفريع، على سبيل المثال، التحديدات التي تستخدم عبارات if وتحديدات تستخدم عبارات switch اعتماداً على سياق التطبيق، قد يكون أحد أنواع عبارة التحديد أكثر تعبيراً وإيجازاً من الآخر.

لنفترض العمل على التطبيقات التي تستخدم عبارات التحديدات على نطاق واسع، في بعض الحالات تستخدم العبارات `if-elseif-else` لإنتاج تعليمات برمجية موجزة ومعبرة، يسهل قراءتها وصيانتها، في حالات أخرى، تنتج العبارات `if-elseif-else` النتيجة المطلوبة، ولكن يصعب قراءتها وصيانتها، وتم تكليفك بمراجعة التعليمات البرمجية، وتحديد متى يكون مناسباً استخدام عبارة switch بدلاً من عبارة if

في هذه الوحدة، سوف تتحقق من استخدام عبارة switch لتنفيذ منطق التفريع كبديل لعبارة if ستعمل أيضاً على تحويل بنية `if-elseif-else` إلى بنية switch-case خلال هذه العملية، سوف تتعلم، التعرف على فوائد اختيار نوع واحد من بيان التحديد على الآخر.

في نهاية هذه الوحدة، ستكون قادرًا على تنفيذ عبارات `switch` في تطبيقك، وتحديد متى تستخدم عبارة `switch` عبر بنية if-elseif-else وتحويل عبارات if-elseif-else إلى عبارات switch

٢ تنفيذ عبارة تبديل a **switch** statement

العبارة switch هي عبارة تحديد selection statement توفر بديلاً لتفرع بناء if-elseif-else توفر عبارة switch مزايا أكثر من بناء -if-elseif else عند تقييم قيمة واحدة مقابل قائمة من القيم المعروفة، والمتماثلة.

فكر في السيناريو التالي:

- أنت تعمل على تطبيق يتعلق بتغذية الطعام، قسم من التعليمات البرمجية يتعامل مع الفواكه.
- تضم التعليمات البرمجية متغيرا يسمى fruit يستخدم للاحتفاظ باسم أنواع مختلفة من الفاكهة.
- لديك قائمة ب ٢٠ نوع فاكهة يركز عليها تطبيقك.
- تريد تفريع تعليمات برمجية استناداً إلى القيمة المعينة إلى fruit

في هذا السيناريو، يمكنك استخدام عبارة switch لإنشاء فرع منفصل لكل نوع من أنواع الفاكهة.

كيف تعمل عبارة التبديل؟

تختار العبارة switch قسماً أو تفرع واحداً من التعليمات البرمجية لتنفيذه، من أقسام أو تفرعات switch المحتملة، يتم اختيار قسم switch المحدد بناءً على تطابق نمط مع تعبير مطابقة العبارة.

ضع في اعتبارك عينة التعليمات البرمجية التالية التي تعرض البنية الأساسية للبيانات switch

```
var fruit = "apple";


switch (fruit)
{
    case "apple":
        Console.WriteLine($"App will display
information for apple.");
        break;

    case "banana":
        Console.WriteLine($"App will display
information for banana.");
        break;

    case "cherry":
        Console.WriteLine($"App will display
information for cherry.");
        break;
}
```

تعبير المطابقة match expression يمكن الإشارة إليه أيضًا باسم تعبير التبديل switch expression هو القيمة التي تتبع الكلمة الأساسية ل switch الكلمة الأساسية في هذه الحالة هي الفاكهة fruit يتم تعريف كل قسم switch بواسطة نمط الحالة case يتم إنشاء أنماط الحالة Case

patterns باستخدام الكلمة الأساسية `case` متبوعة بقيمة، نمط الحالة الأول في هذا المثال هو `case "apple":`

أنماط الحالة Case patterns هي تعبيرات منطقية يتم تقييمها إما إلى `true` or `false` يضم كل مقطع تبديل عدداً صغيراً من أسطر التعليمات التي سيتم تنفيذها، إذا كان نمط الحالة مطابقة لتعبير المطابقة، في هذا المثال، إذا `fruit` تم تعيين قيمة "apple" فسيتم تقييم نمط الحالة الأولى على أنه true وسيتم تنفيذ قسم case هذا.

يجب أن تضم عبارة `Switch` مقطع `case` واحد على الأقل، ولكن عادة ما تحتوي على ثلاثة مقاطع `case` أو أكثر.

أفضل استخدام لـ `Switch` عندما:

- لديك قيمة واحدة (متغير أو تعبير) تريد مطابقتها مع العديد من القيم المحتملة.
- لأي تطابق معين، تحتاج إلى تنفيذ سطرين من التعليمات البرمجية على الأكثر.

ملاحظة

هذا المثال الأول من العبارة switch سهل، وفحصك لبناء الجملة موجزاً. سوف تقوم بفحص ميزات إضافية من العبارة switch عند العمل من خلال بعض السيناريوهات الأكثر تقدماً في الدروس القادمة.

حان الوقت لإعداد بيئة الترميز الخاصة بك، والبدء في تطوير عباراتك الخاصة switch

إنشاء جملة تبديل واختبارها

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في المحرر.

يجب أن يكون Program.cs فارغ. إذا لم يكن كذلك، فحدد جميع أسطر التعليمات البرمجية واحذفها.

٢. اكتب التعليمات البرمجية التالية في محرر Visual Studio Code

```
int employeeLevel = 200;
string employeeName = "Mohamed Abdullah";

string title = "";

switch (employeeLevel)
{
    case 100:
        title = "Junior Associate";
        break;
    case 200:
        title = "Senior Associate";
        break;
    case 300:
        title = "Manager";
        break;
    case 400:
        title = "Senior Manager";
        break;
    default:
        title = "Associate";
        break;
}

Console.WriteLine($"{employeeName}, {title}");
```

٣. في قائمة **File** في Visual Studio Code حدد **Save**

يجب حفظ ملف Program.cs قبل بناء التعليمات البرمجية أو تشغيلها.

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد المشروع TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

سيتم فتح قائمة Terminal يجب أن يظهر موجه الأوامر مسار موقع مجلد TestProject

٥. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ملاحظة

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع، واسم المجلد الخاص بك.

ينبغي أن تشاهد الإخراج التالي:

```
Mohamed Abdullah, Senior Associate
```

٦. خذ دقيقة لمراجعة العبارة switch التي أدخلتها.

٧. لاحظ أن العبارة switch تحدد كتلة تعليمة برمجية واحدة.

تعرف العبارة switch كتلة تعليمة برمجية واحدة، تضم قائمة بحالات أو أقسام case

على يمين الكلمة الأساسية switch يوجد تعبير switch محاط بين قوسين

```
switch (employeeLevel)
```

٨. لاحظ قائمة أقسام case داخل كتلة التعليمات البرمجية

تحتوي كتلة التعليمات البرمجية switch على قائمة بحالات أو أقسام case كل منها يتضمن تسميات تبديل واحدة أو أكثر، بالإضافة إلى ذلك، يضم كل مقطع case قائمة عبارات سيتم تنفيذها، إذا كانت التسمية مساوية لتعبير case المحدد في أعلى عبارة switch

يتم تقييم عبارة switch مقابل تسميات الحالة case من أعلى إلى أسفل حتى يتم العثور على قيمة مساوية لتعبير switch إذا لم تكن أي من التسميات متطابقة، فسيتم تنفيذ تعليمات الحالة default إذا لم يتم أضافة حالة case افتراضية، يتم نقل عنصر التحكم إلى نقطة نهاية عبارة switch يجب أن توفر كل تسمية نوع قيمة يطابق النوع المحدد في تعبير التبديل switch

ملاحظة

يمكن أن تكتب التسمية الاختيارية default في أي موضع ضمن قائمة أقسام case ومع ذلك، يختار معظم المطورين وضعها أخراً، لأنه من المنطقي أو الأكثر (منطقياً) وضع default كخيار نهائي. لكن بغض النظر عن مكان default سوف يتم تقييمها أخيراً.

في مثالنا:

- تعبير switch هو (employeeLevel)
- يحتوي كل مقطع switch على تسمية واحدة (case or default)
- يتم تعريف قسم switch المطابق بواسطة case: 200 بما أن employeeLevel = 200

٩. لاحظ أن كل مقطع switch مفصول عن المقطع التالي.

يسمح بتنفيذ حالة case واحدة فقط، هذا يعني أن عند تنفيذ case غير مسموح بالانتقال إلى مقطع case التالي، هنا يأتي دور الكلمة الأساسية break; هي واحدة من عدة طرق لإنهاء مقطع case قبل أن تصل إلى القسم التالي. **إذا نسيت الكلمة الأساسية break (أو اختيارياً الكلمة الأساسية return) فسينشئ المحول البرمجي خطأ.**

تغيير قيمة متغير employeeLevel لمعرفة كيف تقوم عبارة switch بتقييمها

لممارسة الحالة الافتراضية، دعنا نغير مستوى الموظف عن طريق تعديل تعيين القيمة.

١. تعديل القيمة المعينة إلى employeeLevel عدل تعليماتك البرمجية كما يلي:

```
int employeeLevel = 201;
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

أدخل dotnet run في موجه الأوامر Terminal لتشغيل التعليمات.

٣. لاحظ أن الإخراج قد تغير.

الآن، عند تشغيل التعليمة البرمجية، ينبغي أن ترى عنوان المستخدم الأكثر عمومية.

```
Mohamed Abdullah, Associate
```

نظرا لأن employeeLevel لا يتطابق مع أي حالة case تفعل الحالة الافتراضية default

تعديل مقطع switch لتضمين حالات case متعددة

لنفترض أن شركتنا قررت منح موظفي المستوى 100 لقب "مساعد أول" نفس لقب موظفي المستوى 200 بصفتك المطور، تقرر تنفيذ ذلك عن طريق إزالة مقطع switch الأول الذي ينتمي إلى الحالة أو التسمية case 100: وبدلا من ذلك تسمح لكل من التسميات case 200:, case 100: بتنفيذ نفس مقطع switch

١. لتعديل القيمة المعينة إلى employeeLevel قم بتعديل التعليمات البرمجية كما يلي:

```
int employeeLevel = 100;
```

٢. لتعيين تسميات case متعددة إلى قسم switch الأول، قم بتعديل التعليمات البرمجية كما يلي:

```
case 100:
case 200:
    title = "Senior Associate";
    break;
```

عند الانتهاء من إجراء التغييرات، يجب أن تتطابق التعديلات مع التعليمة البرمجية التالية:

```
int employeeLevel = 100;
string employeeName = "Mohamed Abdullah";

string title = "";

switch (employeeLevel)
{
    case 100:
    case 200:
        title = "Senior Associate";
        break;
    case 300:
        title = "Manager";
        break;
    case 400:
        title = "Senior Manager";
        break;
```

```
    default:
        title = "Associate";
        break;
}

Console.WriteLine($"{employeeName}, {title}");
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

أدخل dotnet run في موجه الأوامر Terminal لتشغيل التعليمات.

ينبغي أن تشاهد الإخراج التالي:

```
Mohamed Abdullah, Senior Associate
```

الخلاصة

إليك النقاط السريعة الرئيسية التي تعلمتها عن عبارة Switch

- استخدم العبارة switch عندما يكون لديك قيمة واحدة مع العديد من التطابقات المحتملة، كل تطابق يتطلب فرعاً في منطق التعليمات البرمجية.
- يمكن مطابقة قسم switch واحد يحتوي على منطق التعليمات البرمجية باستخدام حالة أو تسمية واحدة أو أكثر بواسطة الكلمة الأساسية case
- استخدم الكلمة الأساسية الاختيارية default لإنشاء تسمية وقسم switch سيتم استخدامه عند عدم تطابق أي تسميات case الأخرى

أختبر معلوماتك

١- يكتب المطور التعليمات البرمجية لتنفيذ عبارة switch-case ما هو الغرض من الكلمة الأساسية break

- تخبر الكلمة الأساسية break وقت التشغيل بمتابعة تقييم الحالات الأخرى في switch
- تخبر الكلمة الأساسية break وقت التشغيل بالتوقف عن تقييم انماط case وتمنع تنفيذ الحالات الأخرى في switch
- تخبر الكلمة الأساسية break وقت التشغيل للخروج من التطبيق

٢- عبارة switch-case تقيم متغيراً مقابل العديد من القيم المطابقة المحتملة، وهي تتضمن الكلمة الأساسية default كجزء من بنيتها، ما الغرض من الكلمة الأساسية default

- توفر الكلمة الأساسية default القيمة الافتراضية للمتغير إذا لم تتم تهيئة المتغير
- تعمل الكلمة الأساسية default كقيمة مطابقة عندما لا تكون أي من القيم المتوفرة في حالات case متطابقة
- توفر الكلمة الأساسية default كتلة التعليمات البرمجية للإجراءات الافتراضية التي يتم تنفيذها دائما بغض النظر عن قيمة حالة المطابقة

٣- أي من العبارات التالية حول عبارة switch-case صحيحة؟

- يمكن أن تحتوي عبارة switch على تسميات حالات case متعددة
- يجب أن تتضمن عبارة switch قسم switch افتراضيًا
- النقطتان في نهاية تسمية الحالة اختيارية

راجع إجابتك

١ تخبر الكلمة الأساسية break وقت التشغيل بالتوقف عن تقييم انماط case وتمنع تنفيذ الحالات الأخرى في switch

صحيح تعيد الكلمة الأساسية break توجيه تنفيذ التعليمات البرمجية وتمنع قسم case من المتابعة إلى القسم التالي

٢ تعمل الكلمة الأساسية default كقيمة مطابقة عندما لا تكون أي من القيم المتوفرة في حالات case متطابقة

صحيح إذا لم تكن هناك تسميات قيمة مطابقة case يتم استخدام التسمية الاختيارية default كقيمة مطابقة

٣ يمكن أن تحتوي عبارة switch على تسميات حالات case متعددة

صحيح

٣ تمرين - إكمال نشاط تحدي باستخدام عبارات switch statements

تحدي التحويل إلى عبارات switch

في هذا التحدي، ستقوم بإعادة كتابة عبارات if-elseif-else بتحويلها إلى عبارات switch يجب أن يساعدك هذا التحدي في رؤية نقاط القوة/نقاط الضعف في عبارة switch عند مقارنتها ببنية if-elseif-else حظ سعيد.

تحدي التعليمات البرمجية: إعادة كتابة if-elseif-else باستخدام عبارة switch

ستبدأ بالتعليمات البرمجية التي تستخدم عبارات if-elseif-else لتقييم مكونات SKU وحدة حفظ المخزون للمنتج. يتم تنسيق SKU (وحدة حفظ المخزون) باستخدام ثلاث قيم مشفرة:

```
<product #>-<2-letter color code>-<size code>
```

على سبيل المثال، قيمة SKU 01-MN-L تتطابق مع (قميص رياضي) - (كستنائي) - (كبير) (maroon)-(large) -(sweat shirt) ويظهر الإخراج وصفًا مثل "المنتج: قميص رياضي كستنائي كبير" Product: Large Maroon Sweat shirt"

يمثل التحدي الذي تواجهه في تحويل تعليمات العبارة if إلى عبارة switch تحقق نفس النتيجة، مثل التعليمات البرمجية الأولية.

١. تأكد من وجود ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

- في **القائمة File** حدد Open Folder

- استخدم مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه.
- في قائمة استكشاف EXPLORER حدد **Program.cs**
- أحذف كل التعليمات البرمجية الموجودة

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
// SKU = Stock Keeping Unit.
// SKU value format: <product #>-<2-letter
color code>-<size code>
string sku = "01-MN-L";

string[] product = sku.Split('-');

string type = "";
string color = "";
string size = "";

if (product[0] == "01")
{
    type = "Sweat shirt";
} else if (product[0] == "02")
{
    type = "T-Shirt";
} else if (product[0] == "03")
{
    type = "Sweat pants";
}
else
{
    type = "Other";
}

if (product[1] == "BL")
{
    color = "Black";
} else if (product[1] == "MN")
```

```
{
    color = "Maroon";
} else
{
    color = "White";
}

if (product[2] == "S")
{
    size = "Small";
} else if (product[2] == "M")
{
    size = "Medium";
} else if (product[2] == "L")
{
    size = "Large";
} else
{
    size = "One Size Fits All";
}

Console.WriteLine($"Product: {size} {color}
{type}");
```

٣. تعديل التعليمات البرمجية لاستخدام عبارة switch بدلاً من عبارات if-elseif-else

بصرف النظر عن كيفية القيام بذلك، يجب أن تنتج تعليمتك البرمجية الإخراج التالي:

```
Product: Large Maroon Sweat shirt
```

سواء واجهتك مشكلة، وتحتاج إلى إلقاء نظرة خاطفة على الحل، أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٤ مراجعة الحل لنشاط تحدي عبارة switch-case

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
// SKU = Stock Keeping Unit
string sku = "01-MN-L";

string[] product = sku.Split('-');

string type = "";
string color = "";
string size = "";

switch (product[0])
{
    case "01":
        type = "Sweat shirt";
        break;
    case "02":
        type = "T-Shirt";
        break;
    case "03":
        type = "Sweat pants";
        break;
    default:
        type = "Other";
        break;
}

switch (product[1])
{
    case "BL":
        color = "Black";
        break;
    case "MN":
        color = "Maroon";
```

```
            break;
        default:
            color = "White";
            break;
}

switch (product[2])
{
        case "S":
            size = "Small";
            break;
        case "M":
            size = "Medium";
            break;
        case "L":
            size = "Large";
            break;
        default:
            size = "One Size Fits All";
            break;
}

Console.WriteLine($"Product: {size} {color}
{type}");
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، بصرف النظر عن طريقة حلك، يجب أن يبقى الإخراج كما هو:

```
Product: Large Maroon Sweat shirt
```

طالما أن الإخراج هو نفسه، واستخدمت عبارة switch-case بدلاً من عبارة if-elseif-else فقد أكملت التحدي بنجاح.

إذا نجحت، فتهانينا! تابع لاختبار المعلومات في الدرس التالي.

٥ اختبر معلوماتك

١- أي من العبارات التالية حول عبارة switch-case صحيحة؟

- يمكن أن تحتوي عبارة switch على تسميات حالات case متعددة
- يجب أن تتضمن عبارة switch قسم switch افتراضيًا
- النقطتان في نهاية تسمية الحالة اختيارية

٢ - يكتب المطور التعليمات البرمجية لتنفيذ عبارة switch-case ما هو الغرض من الكلمة الأساسية break

- تخبر الكلمة الأساسية break وقت التشغيل بمتابعة تقييم الحالات الأخرى في switch
- تخبر الكلمة الأساسية break وقت التشغيل بالتوقف عن تقييم انماط case وتمنع تنفيذ الحالات الأخرى في switch
- تخبر الكلمة الأساسية break وقت التشغيل للخروج من التطبيق

راجع إجابتك

١ يمكن أن تحتوي عبارة switch على تسميات حالات case متعددة

صحيح يمكن أن يحتوي مقطع switch واحد على تسميات حالات case متعددة

٢ تخبر الكلمة الأساسية break وقت التشغيل بالتوقف عن تقييم انماط case وتمنع تنفيذ الحالات الأخرى في switch

صحيح تعيد الكلمة الأساسية break توجيه تنفيذ التعليمات البرمجية، وتمنع قسم case من المتابعة إلى الحالة التالية

٦ الملخص

كان هدفك هو إضافة منطق تفريع يطابق متغيراً أو تعبيراً واحداً، مقابل العديد من القيم المحتملة.

باستخدام عبارة switch-case قمت بمطابقة المستوى الرقمي للموظف مع عنوانه، لقد استخدمت switch-case كبديل للبنية if-elseif-else للتعبير بشكل أكثر إيجازاً عن هدفك، تحويل وحدة حفظ المخزون (SKU) إلى وصف طويل النموذج.

يستخدم مطوري C# عبارات if والتحديد switch جنباً إلى جنب مع عوامل التشغيل الشرطية، لتفريع التعليمات البرمجية الخاصة بهم، تمكنك التجربة التي اكتسبتها باستخدام عبارات التحديد، من اختيار أفضل أداة C# للتعبير عن هدفك في تعليماتك البرمجية.

الوحدة الرابعة

التكرار خلال كتلة التعليمة البرمجية باستخدام العبارة باللغة C#

استخدم العبارة التكرارية for iteration لإجراء تكرار لعدد محدد مسبقًا من المرات، والتحكم في عملية التكرار.

الأهداف التعليمية

بعد إكمال هذه الوحدة، ستتمكن مما يلي:

- استخدام عبارة for للتكرار عبر كتلة من التعليمات البرمجية
- تعديل كيفية تنفيذ وقت التشغيل NET Runtime لمنطق التكرار، وتغيير قيمة التكرار، وشرط التكرار، والنمط أو الأسلوب

محتويات الوحدة:

١- مقدمة

٢- إنشاء وتكوين حلقات التكرار for

٣- إكمال نشاط تحدي باستخدام عبارات for and if

٤- مراجعة الحل لنشاط تحدي العبارات for and if

٥- اختبر معلوماتك

٦- الملخص

١ المقدمة

هناك عدة طرق لإضافة منطق التكرار الحلقي looping logic في تطبيقك، واعتماداً على السياق، يوفر كل منها مجموعة محددة من الميزات التي لها إيجابيات وسلبيات.

لنفترض أنك على وشك البدء في العمل على تطبيق يعالج بيانات نصية ورقمية، باستخدام مصفوفة أحادية، ومتعددة الأبعاد، بعد مراجعة أولية، تدرك أن العبارات foreach لا تدعم منطق التكرار الحلقي الذي سيكون مطلوباً في العديد من الحالات، ستحتاج إلى نهج آخر للتكرار من خلال المصفوفات متعددة الأبعاد، والحالات التي لا توفر فيها حلقات foreach مستوى تحكم مناسب في التكرار المطلوب، تحتاج إلى اكتساب خبرة في استخدام عبارات for إذا كنت ستنجح في هذا المشروع.

في هذه الوحدة، ستبدأ بكتابة عبارات for تتكرر عدداً محدداً من المرات. بعد تنفيذ العبارة الأساسية for ستتعلم كيفية تنفيذ عبارات for التي تتكرر للخلف من خلال مصفوفة، أو تخطي عناصر المصفوفة أثناء التكرار، أو معالجة العناصر المحددة فقط من المصفوفة، عن طريق تغيير قيمة التكرار، وشرط التكرار، والنمط أو أسلوب التكرار.

في نهاية هذه الوحدة، ستتمكن من استخدام عبارات for لتنفيذ منطق التكرار الحلقي عندما لا تدعم عبارات foreach هدفك.

٢ إنشاء وتكوين حلقات التكرار for iteration loops

ظاهريًا، عبارة for هي عبارة تكرار أخرى، تسمح لك بالتكرار عبر كتلة التعليمات البرمجية، وبالتالي تغيير تدفق تنفيذ تعليماتك، ومع ذلك، بمجرد أن ندرس كيفية عمل كل منها، يمكننا تحديد الفروق الدقيقة في كل عبارة تكرار، بشكل أفضل ومتى تستخدمها.

ما هي العبارة for؟

تتكرر العبارة for من خلال كتلة التعليمات البرمجية لعدد معين من المرات، يجعل هذا المستوى من التحكم العبارة for فريدة من نوعها بين عبارات التكرار الأخرى تتكرر العبارة foreach من خلال كتلة من التعليمات البرمجية مرة واحدة، لكل عنصر في تسلسل بيانات مثل مصفوفة أو مجموعة، تتكرر العبارة while _سوف ندرسها لاحقاً_ من خلال كتلة من التعليمات البرمجية حتى يتم استيفاء شرط.

علاوة على ذلك، تمنحك العبارة for المزيد من التحكم في عملية التكرار، عن طريق الكشف عن شروط التكرار.

في هذا التمرين، ستستخدم العبارة for وتعلم كيفية التحكم في الشرط المسبق للتكرار، وحالة الإكمال، ونمط التكرار الخاص، والمزيد، ستتعرف أيضاً على حالات استخدام for الشائعة.

حسنا، الآن دعونا نعد بيئة الترميز الخاصة بنا، ونبدأ فحصنا لعينات التعليمات البرمجية التي تنفذ عبارة for

إعداد بيئة الترميز

تتضمن هذه الوحدة أنشطة عملية ترشدك خلال عملية إنشاء التعليمات البرمجية التوضيحية وتشغيلها، نشجعك على إكمال هذه الأنشطة باستخدام Visual Studio Code سيساعدك استخدام Visual Studio Code لهذه الأنشطة على أن تصبح أكثر راحة في كتابة التعليمات البرمجية، وتشغيلها في بيئة مطور يستخدمها المحترفون في جميع أنحاء العالم.

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة **File** Visual Studio Code حدد **Open Folder**

٣. في **مربع الحوار** **Open Folder** dialog انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام موقعه كبديل لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار** **Open Folder** dialog حدد **Select Folder**

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** Visual Studio Code حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

داخل موجه الأوامر Terminal لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد اكتب **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط علىEnter

في قائمة EXPLORER قم **بتوسيع المجلد** **CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.csوملف مشروع C# يسمى TestProject.csproj

٦. في قائمة EXPLORER لعرض ملف التعليمات البرمجية في المحرر، حدد **Program.cs**

٧. حذف أسطر التعليمات البرمجية الموجودة.

ستستخدم مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

٨. أغلق قائمة Terminal

كتابة عبارة for الأساسية for statement

١. اكتب التعليمات البرمجية التالية في محرر Visual Studio

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

تقدم هذه التعليمة البرمجية عبارة for بسيطة تتكرر عبر كتلة التعليمات البرمجية الخاصة بها 10 مرات، وتطبع القيمة الحالية ل **i**

٢. في قائمة **File** حدد **Save**

يجب حفظ ملف Program.cs قبل build بناء التعليمات أو تشغيلها.

٣. في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

سيتم فتح قائمة Terminal يجب أن يتضمن موجه الأوامر موقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع، على سبيل المثال: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

يجب أن ترى الإخراج التالي:

```
0
1
2
3
4
5
6
7
8
9
```

٥. خذ دقيقة لتحديد الأجزاء الستة من عبارة for

تتضمن عبارة for الأجزاء الستة التالية:

- الكلمة الأساسية for
- مجموعة من الأقواس التي تحدد شروط التكرار for تحتوي الأقواس على ثلاثة أجزاء مميزة، مفصولة عن بعضها، بعامل تشغيل الفاصلة المنقوطة ;

١. الجزء الأول يعمل على تحديد وتهيئة متغير رقمي للعداد، في هذا المثال `int i = 0;` يشار إلى هذا القسم باسم المُهيئ the initializer

٢. الجزء الثاني يحدد شرط الإكمال، في هذه المثال `i < 10;` معنى آخر، سيستمر وقت التشغيل Runtime في تكرار التعليمات البرمجية داخل الكتلة البرمجية أسفل العبارة for بينما **i** أقل من 10 عندما يصبح **i** مساويًا لـ 10 يتوقف وقت التشغيل عن تنفيذ كتلة التعليمات البرمجية الخاصة بالعبارة for يشار إلى هذا القسم باسم الشرط the condition

٣. الجزء الثالث يحدد الإجراء الذي يجب اتخاذه بعد كل تكرار، في هذا المثال، بعد كل تكرار سيؤدي **i++** إلى زيادة قيمة i بمقدار 1 يشار إلى هذا القسم باسم المكرر the iterator

- وأخيرًا، كتلة التعليمة البرمجية تحتوي على التعليمات التي سيتم تنفيذها في كل تكرار، تتم الإشارة إلى قيمة **i** داخل كتلة التعليمات البرمجية، يشار إلى هذا القسم باسم النص الأساسي the body

نظرا لقواعد تسمية المتغيرات، قد تتساءل عما إذا كان **i** اسماً صالحاً للمتغير، الذي يحتويه التكرار الحالي، في هذه الحالة **i** يعتبر في المجمل صالحاً، الخيارات الشائعة الأخرى هي counter , x يتم استخدام الاسم **j** أيضاً في هذه الحالات، عندما يكون لديك عبارة for خارجية تستخدم **i** وتحتاج إلى إنشاء متغير آخر في تكرار لعبارة for داخلية

جميع الأقسام الثلاثة (initializer, condition, and iterator) التهيئة، والحالة، والمكرر، اختيارية، ومع ذلك، في الممارسة العملية، عادةً ما يتم استخدام الأقسام الثلاثة جميعها

تغيير شروط التكرار

وكما ذكرنا في البداية، فإن العبارة for لها صفتان فريدتان بين عبارات التكرار الأخرى:

- يجب استخدام العبارة for عندما تعرف مسبقًا، عدد المرات التي تحتاج إلى تكرار مجموعة من التعليمات البرمجية.
- تسمح لك العبارة for بالتحكم في الطريقة التي تتم بها معالجة كل تكرار.

ماذا لو كنا بحاجة إلى التكرار من خلال كتلة التعليمة البرمجية، ولكننا نريد العدّ تنازليًا بدلاً من العد تصاعديًا؟

١. استخدم المحرر، لتعديل تعليماتك البرمجية كما يلي:

```
for (int i = 10; i >= 0; i--)
{
    Console.WriteLine(i);
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية المعدلة.

من خلال تغيير الأجزاء الثلاثة من العبارة for نقوم بتغيير سلوكها.

- نشرع في تهيئة متغير التكرار **i** ليصبح 10

- نغير شرط الإكمال للخروج من العبارة for عندما يكون i أقل من 0
- نقوم بتغيير نمط التكرار لإنقاص 1 من i كل مرة يحدث فيها التكرار.

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

أدخل dotnet run موجه الأوامر Terminal لتشغيل التعليمات البرمجية.

٤. لاحظ أن الإخراج قد تغير.

عند التشغيل، سترى المخرجات التالية:

```
10
9
8
7
6
5
4
3
2
1
0
```

تجربة نمط المكرر

ماذا لو أردنا تخطي قيم معينة في متغير العداد؟

١. عدل التعليمات البرمجية كما يلي:

```
for (int i = 0; i < 10; i += 3)
{
    Console.WriteLine(i);
}
```

١. خذ دقيقة لمراجعة التعليمات البرمجية المعدلة.

بدلا من زيادة قيمة متغير المكرر أو إنقاصها مرة واحدة، نستخدم `i += 3` لتخطي قيمتين دفعة واحدة، بعد كل تكرار.

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

٣. لاحظ كيف تغير الإخراج.

عند التشغيل، سترى المخرجات التالية:

```
0
3
6
9
```

من المسلم به أنك لن تفعل هذا النوع من الأشياء في كثير من الأحيان، ولكن نأمل أن تتمكن من تقدير أن لديك مستوى دقيقاً من التحكم في التكرارات، إذا كنت بحاجة إليها في وقت ما.

استخدام الكلمة الأساسية break للخروج من عبارة التكرار

ماذا لو كنا بحاجة إلى الخروج من عبارة التكرار مبكرًا، بناءً على بعض الشروط؟ يمكننا استخدام الكلمة الأساسية break

١. استخدم المحرر لتعديل التعليمات البرمجية كما يلي:

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
    if (i == 7) break;
}
```

٢. خذ دقيقة لمراجعة استخدام الكلمة الأساسية `break` في التعليمات المعدلة.

رأينا الكلمة الأساسية break في الوحدة "تدفق التعليمات البرمجية باستخدام عبارة switch-case" كما اتضح هنا، يمكننا استخدام الكلمة الأساسية break للخروج من عبارات التكرار أيضاً.

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

٤. لاحظ كيف تغير الإخراج.

عند التشغيل، سترى المخرجات التالية.

```
0
1
2
3
4
5
6
7
```

التكرار الحلقي عبر كل عنصر من عناصر المصفوفة

الاستخدام الشائع للعبارة for هو التكرار من خلال صفيف من العناصر، خاصة إذا كنت بحاجة إلى بعض التحكم، في الطريقة التي يحدث بها التكرار، بينما foreach يكرر من خلال كل عنصر من عناصر مصفوفة، يمكن تعديل العبارة for لتوفير المزيد من التخصيص.

١. استخدم محرر Visual Studio Code لتعديل التعليمات البرمجية الخاصة بك كما يلي:

```
string[] names = { "Ahmed", "Ali", "Eman",
"Mohamed" };
for (int i = names.Length - 1; i >= 0; i--)
{
    Console.WriteLine(names[i]);
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية المعدلة.

أولا، لاحظ أننا قمنا بإنشاء مثيل مصفوفة string يسمى names يحتوي أربعة أسماء.

بعد ذلك، لاحظ أننا نستخدم الخاصية Array.Length للحصول على عدد العناصر في المصفوفة، واستخدمنا هذه القيمة لتهيئة متغير `int i = names.Length - 1` نطرح 1 من القيمة لأن رقم الفهرس لعناصر المصفوفة يستند إلى الصفر، أرقام الفهرس للعناصر الأربعة هي 0-3

وأخيراً، لاحظ أننا اخترنا التكرار من خلال المصفوفة للخلف - شيء لا يمكننا القيام به مع العبارة foreach نستخدم قيمة متغير التكرار داخل كتلة التعليمات البرمجية لتحديد رقم فهرس عناصر المصفوفة (names[i])

٣. احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيلها.

٤. لاحظ أن عناصر المصفوفة مدرجة بترتيب عكسي، كما ننوه.

عند تشغيل التعليمة البرمجية، سترى المخرجات التالية:

```
Mohamed
Eman
Ali
Ahmed
```

من الممكن التكرار إلى الأمام، خلال عناصر المصفوفة، عن طريق إنشاء العبارة for كما يلي:

```
for (int i = 0; i < names.Length; i++)
```

فحص قيود عبارة foreach

ماذا لو كنت تريد تحديث قيمة في المصفوفة أثناء تكرار foreach ؟

١. استخدم محرر Visual Studio Code لتعديل التعليمات البرمجية الخاصة بك كما يلي:

```
string[] names = { "Ahmed", "Ali", "Eman",
"Mohamed" };
foreach (var name in names)
{
    // لا يمكن التعديل // Can't do this:
    if (name == "Ali") name = "Sammy";
}
```

٢. احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيلها.

٣. لاحظ رسالة الخطأ التي يتم عرضها.

إذا حاولت تحويل هذه التعليمة، برمجيًا وتشغيلها، فستلاحظ وجود خطأ

```
Cannot assign to name because it is a 'foreach
iteration variable'
```

بمعنى آخر، لا يمكنك إعادة تعيين قيمة name لأنها جزء من التنفيذ الداخلي للتكرار foreach

التغلب على قيود عبارة foreach باستخدام عبارة for

لنحاول استخدام عبارة for لتعديل محتويات مصفوفة داخل كتلة التكرار.

١. استخدم محرر Visual Studio Code لتعديل التعليمات البرمجية الخاصة بك كما يلي:

```
string[] names = { "Ahmed", "Ali", "Eman",
"Mohamed" };
for (int i = 0; i < names.Length; i++)
    if (names[i] == "Ali") names[i] = "Sammy";

foreach (var name in names)
Console.WriteLine(name);
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية الجديدة.

لاحظ أننا أزلنا الأقواس المتعرجة من الكتل البرمجية، التي تحتوي على سطر واحد، تستخدم هذه المراجعة نفس التقنية التي تحدثنا عنها في الوحدة "التحكم في نطاق المتغير والمنطق باستخدام كتل التعليمات البرمجية في C#" يجد العديد من المطورين صعوبة في قراءة هذا النمط، بينما يفضل آخرون هذا النمط المختصر، لأنه يساعدهم على الكتابة بإيجاز وتعبير أكثر. إذا وجدت صعوبة في قراءة هذه التعليمات البرمجية، أو إذا كنت لا تفضل هذا النمط، فتأكد من أنه يمكن دائماً استخدام الأقواس المتعرجة في كتل تعليماتك البرمجية. إذا أردت، فقم بتحديث التعليمات البرمجية في المحرر، كالتالية:

```
string[] names = { "Ahmed", "Ali", "Eman",
"Mohamed" };

for (int i = 0; i < names.Length; i++)
{
    if (names[i] == "Ali")
    {
        names[i] = "Sammy";
    }
}
```

```
foreach (var name in names)
{
    Console.WriteLine(name);
}
```

احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيلها.

لاحظ أن التعليمات البرمجية تعمل دون خطأ، وتنشئ الإخراج المطلوب.

عند تشغيل التعليمات البرمجية، سترى المخرجات التالية.

```
Ahmed
Sammy
Eman
Mohamed
```

بما أن المصفوفة ليست جزءًا من تنفيذ عبارة التكرار مباشرةً، يمكنك تغيير القيم داخل المصفوفة.

خلاصة

فيما يلي بعض الدروس المستفادة من هذ الدرس:

- تسمح لك عبارة التكرار for بالتكرار من خلال كتلة من التعليمات البرمجية لعدد معين من المرات.
- تسمح لك عبارة التكرار for بالتحكم في كل جانب من جوانب ميكانيكا التكرار، عن طريق تغيير الشروط الثلاثة داخل الأقواس: المهيئ والشرط، والمكرر.
- من الشائع استخدام العبارة for عندما تحتاج إلى التحكم في كيفية تكرار كل عنصر في مصفوفة.
- إذا كانت كتلة تعليماتك البرمجية تحتوي على سطر واحد من التعليمات، يمكنك إزالة الأقواس المتعرجة، والمسافة البيضاء إذا كنت ترغب في ذلك.

اختبر معلوماتك

١- أي من جمل for التالية صحيحة؟

- `for (int x = 20: x < 80: x++)`
- `for (int j = 0; j < 80; j + 1)`
- `for (int counter = 20; counter < 80; counter++)`

٢- أي من العبارات التالية يجب استخدامها للخروج من حلقة for قبل اكتمال التكرار؟

- `exit;`
- `break;`
- `return;`

راجع إجابتك

```
for (int counter = 20; counter < 80; counter ++)
```

صحيح

٢

```
break;
```

صحيح يمكن استخدام الكلمة الأساسية break لإيقاف حلقة for عند استيفاء الشرط المطلوب

٣ إكمال نشاط تحدي باستخدام عبارات for and if statements

تحدي FizzBuzz

يعد FizzBuzz تحديًا شائعًا للتعليمات، وسؤالًا لمقابلات العمل، إنه يمارس فهمك للعبارة for والعبارة if وعامل التشغيل باقي القسمة % وأمر المنطق الأساسي الخاص بك.

تحدي التعليمات البرمجية ـ تنفيذ قواعد تحدي FizzBuzz

فيما يلي قواعد FizzBuzz التي تحتاج إلى تنفيذها في مشروع التعليمات البرمجية:

- قيم الإخراج من 1 إلى 100 رقم واحد لكل سطر، طبع من داخل كتلة التعليمات لعبارة التكرار.
- عندما تكون القيمة الحالية قابلة للقسمة على 3 اطبع المصطلح Fizz الموجود بجانب الرقم.
- عندما تكون القيمة الحالية قابلة للقسمة على 5 اطبع المصطلح Buzz الموجود بجانب الرقم.
- عندما تكون القيمة الحالية قابلة للقسمة على **كل من** 5،3 اطبع المصطلح FizzBuzz بجانب الرقم.

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات إعداد ملف Program.cs في المحرر، التي درستها سابقاً.

٢. اكتب التعليمات البرمجية التي تنفذ كل قاعدة.

هام

ستحتاج إلى فهم كيفية استخدام عامل التشغيل % باقي القسمة، لتحديد ما إذا كان الرقم قابلاً للقسمة على رقم آخر بالتساوي، لقد تناولنا هذا في الوحدة "تنفيذ العمليات الأساسية على الأرقام بلغة C#"

٣. بعد التنفيذ وتشغيل تطبيقك، تحقق أن الإخراج يفي بالمتطلبات.

بغض النظر عن كيفية تفرعك مع التكرار، والعبارات الشرطية، يجب أن تنتج التعليمات البرمجية الإخراج التالي:

```
1
2
3 - Fizz
4
5 - Buzz
6 - Fizz
7
8
9 - Fizz
10 - Buzz
11
12 - Fizz
13
14
15 - FizzBuzz
16
17
18 - Fizz
19
20 - Buzz
21 - Fizz
22
.
.
.
```

ملاحظة

نعرض القيم ال 22 الأولى فقط، ولكن يجب أن يستمر الإخراج إلى 100 الرقم 15 كما تري قابل للقسمة على كل من 3 و5 لذلك نطبع FizzBuzz بجوار هذا الرقم.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل، أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٤ مراجعة الحل لنشاط تحدي العبارة for and if

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
for (int i = 1; i < 101; i++)
{
    if ((i % 3 == 0) && (i % 5 == 0))
        Console.WriteLine($"{i} - FizzBuzz");
    else if (i % 3 == 0)
        Console.WriteLine($"{i} - Fizz");
    else if (i % 5 == 0)
        Console.WriteLine($"{i} - Buzz");
    else
        Console.WriteLine($"{i}");
}
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن.

العبارة for مهمة لأنها تسمح لك بالتكرار من خلال كتلة التعليمات البرمجية ١٠٠ مرة.

تسمح if-elseif-else لك بالتحقق من المقسومين على ٣ و٥

يسمح لك % عامل التشغيل mod بتحديد ما إذا كان ٣ أو ٥ يقسم إلى رقم آخر دون باقي.

ويضمن عامل التشغيل && إمكانية تقسيم رقم على كلا من ٣ و٥ للشرط FizzBuzz

يجب أن تنتج عن التعليمة البرمجية المخرجات التالية من ١ إلى ١٠٠

```
1
2
3 - Fizz
4
5 - Buzz
6 - Fizz
```

```
7
8
9 - Fizz
10 - Buzz
11
12 - Fizz
13
14
15 - FizzBuzz
16
17
18 - Fizz
19
20 - Buzz
21 - Fizz
22
23
24 - Fizz
25 - Buzz
26
27 - Fizz
28
29
30 - FizzBuzz
31
32
33 - Fizz
34
35 - Buzz
36 - Fizz
37
38
39 - Fizz
40 - Buzz
41
42 - Fizz
43
44
45 - FizzBuzz
46
47
48 - Fizz
49
50 - Buzz
51 - Fizz
52
53
54 - Fizz
55 - Buzz
56
57 - Fizz
58
59
60 - FizzBuzz
61
62
```

```
63 - Fizz
64
65 - Buzz
66 - Fizz
67
68
69 - Fizz
70 - Buzz
71
72 - Fizz
73
74
75 - FizzBuzz
76
77
78 - Fizz
79
80 - Buzz
81 - Fizz
82
83
84 - Fizz
85 - Buzz
86
87 - Fizz
88
89
90 - FizzBuzz
91
92
93 - Fizz
94
95 - Buzz
96 - Fizz
97
98
99 - Fizz
100 - Buzz
```

حل آخر:

```
for (int i = 0; i <= 100; i++ )
    {
      if ( (i % 3 == 0) )
      Console.WriteLine($"{i} Fizz");
      else if ( (i % 5 == 0) )
      Console.WriteLine($"{i} Buzz");
      else if ( (i % 3 == 0) && (i % 5 == 0) )
      Console.WriteLine($"{i} FizzBuzz");
      else Console.WriteLine(i);
```

```
}
```

إذا نجحت، فتهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة، ستعتمد جميع الأفكار الجديدة التي نناقشها في وحدات أخرى على فهمك، للأفكار التي تم تقديمها في هذه الوحدة.

٥ اختبر معلوماتك

١- أي من جمل for التالية صحيحة؟

- `for (int x = 20: x < 80: x++)`
- `for (int j = 0; j < 80; j + 1)`
- `for (int counter = 20; counter < 80; counter++)`

٢- أي من العبارات التالية يجب استخدامها للخروج من حلقة for قبل اكتمال التكرار؟

- `exit;`
- `break;`
- `return;`

راجع إجابتك

```
for (int counter = 20; counter < 80; counter ++)
```

صحيح

٢

```
break;
```

صحيح يمكن استخدام الكلمة الأساسية break لإيقاف حلقة for عند استيفاء الشرط المطلوب

٦ الملخص

كان هدفك هو الحصول على تحكم إضافي في عملية التكرار، عن طريق تنفيذ العبارة for في التعليمات البرمجية.

يوفر تعديل المهيئ initializer والشرط condition والمكرر iterator لعبارة for تحكم دقيق في منطق التكرار.

باستخدام العبارة for قمت بتكرار تعليمات الكتلة البرمجية عدة مرات، واستخدمت خاصية Length للمصفوفة للتكرار عبر الكتلة البرمجية مرة واحدة لكل عنصر في هذه المصفوفة، وعدلت المكرر وشرط الخروج من الحلقة، ونمط التكرار، استخدمت الكلمة الأساسية break; للخروج من كتلة الحلقة عند الحاجة.

بدون عنصر تحكم التكرار الذي توفره العبارة for لن تتمكن من إكمال مهام ترميز معينة، مثل التكرار الحلقي عبر البيانات في مصفوفة متعدد الأبعاد.

الوحدة الخامسة

إضافة منطق تكرار إلى التعليمات البرمجية باستخدام عبارات do-while and while

استخدم عبارات `do-while and while statements` للتكرار طالما تم تقييم التعبير المنطقي على أنه صحيح.

الأهداف التعليمية

بعد إكمال هذه الوحدة، ستتمكن مما يلي:

- كتابة التعليمات التي تستخدم عبارة do-while للتكرار من خلال الكتلة البرمجية
- كتابة التعليمات التي تستخدم عبارة while للتكرار من خلال الكتلة البرمجية
- استخدم عبارة continue للانتقال مباشرة إلى التقييم المنطقي

محتويات الوحدة:

١- مقدمة

٢- إنشاء عبارات حلقات التكرار do and while

٣- إكمال نشاط التحدي باستخدام عبارات do and while

٤- مراجعة حل نشاط تحدي do and while

٥- إكمال نشاط التحدي للتمييز بين عبارات do and while

٦- مراجعة نشاط التحدي do ضد while

٧- اختبر معلوماتك

٨- الملخص

١ المقدمة

كما قلنا عدة مرات في الوحدات السابقة التي تتميز بعبارات التكرار، والقرار، هناك العديد من التقنيات التي يمكنك استخدامها لتحقيق نتائج مماثلة، تمامًا مثل اللغات المكتوبة والمنطوقة، يمكنك في لغات البرمجة التعبير عن نفس الفكرة بطرق مختلفة، رغم ذلك، يكون لكل تعبير اختلاف دقيق في المعنى.

عبارة while , do-while تسمح لنا بالتحكم في تدفق تنفيذ التعليمات، بواسطة تنفيذ حلقات من خلال الكتلة البرمجية حتى يتم استيفاء الشرط، عند العمل مع عبارة foreach نحن نكرر مرة واحدة لكل عنصر في التسلسل، مثل مصفوفة، تسمح عبارة for بتكرار عدد محدد مسبقًا من المرات، والتحكم في عملية التكرار.

عبارات while and do-while تسمح بتكرار البيانات من خلال كتلة التعليمات، بهدف أن يؤثر المنطق الموجود داخل كتلة التعليمات، عند إيقاف التكرار.

لنفترض أنك تريد قبول ومعالجة إدخال المستخدم، تريد الاستمرار في قبول ومعالجة الإدخال حتى يضغط المستخدم على المفتاح q للإشارة إلى إنهاء، يمكنك استخدام عبارات while , do-while للحفاظ على التكرار من خلال منطق، لقبول إدخال المستخدم ومعالجته، حتى يصبح المستخدم جاهزًا للتوقف.

في هذه الوحدة، ستستخدم عبارة `do-while` وعبارة while للتكرار خلال الكتلة البرمجية، ستفهم متى تختار واحده على الأخرى، ستستخدم عبارة continue لتخطي معالجة بقية التعليمات البرمجية داخل الكتلة، والانتقال مباشرة إلى التقييم المنطقي لعبارة while

في نهاية هذه الوحدة، سوف تكون قادرًا على استخدام عبارة do-while , while بثقة لإضافة منطق حلقات إلى تطبيقك.

٢ إنشاء عبارات حلقات التكرار do and while

ظاهريًا عبارة `while` , `do-while` هي عبارة تكرار أخرى، تسمح لك بالتكرار خلال كتلة برمجية، ومن ثم تغيير تدفق تنفيذ تعليماتك، ومع ذلك، بمجرد أن ندرس كيفية عمل كل منهما، يمكننا تحديد الفروق الدقيقة في كل عبارة تكرار، ومتى نستخدمها.

ما هي عبارة التحقق بعد التكرار؟ do-while statement

تقوم عبارة do-while بتنفيذ عبارة أو كتلة من العبارات بينما يتم تقييم تعبير منطقي محدد إلى صحيح `true` نظرًا لأنه يتم تقييم هذا التعبير بعد كل تنفيذ للحلقة، يتم تنفيذ حلقة do-while مرة واحدة أو أكثر.

```
do
{
    // This code executes at least one time
} while (true);
```

يبدأ تدفق التنفيذ داخل القوس المتعرجة، تنفذ التعليمات مرة واحدة على الأقل، ثم يتم تقييم التعبير المنطقي، جوار الكلمة الأساسية `while` إذا كان التعبير المنطقي صحيحًا `true` فسيتم تنفيذ كتلة التعليمات البرمجية مرة أخرى.

من خلال الترميز الثابت للتعبير المنطقي إلى صحيح true أنشأنا حلقة لا نهائية - حلقة لن تنتهي أبدًا، على الأقل، ليس كما هو مكتوب حاليًا، نحتاج إلى طريقة للخروج من الحلقة، داخل الكتلة البرمجية، سنناقش معايير الخروج الخاصة do-while بعد قليل.

حسنا، الآن دعونا نعد بيئة الترميز الخاصة بنا، ونبدأ فحصنا لعينات التعليمات البرمجية التي تنفذ عبارة do-while

كتابة عبارة do-while تتوقف عند إنشاء رقم عشوائي معين

دعونا نكتب التعليمات البرمجية التي تستمر في دوران إنشاء أرقام عشوائية بين ١ و١٠ حتى ننشئ عدد ٧ قد يستغرق الأمر تكراراً واحداً فقط، للحصول على ٧ أو قد يستغرق عشرات التكرارات.

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في المحرر.

ملاحظة

يجب أن يكون Program.cs فارغ. إذا لم يكن كذلك، فحدد كافة أسطر التعليمات البرمجية واحذفها

٢. اكتب التعليمات التالية في محرر التعليمات البرمجيةVisual Studio

```
Random random = new Random();
int current = 0;

do
{
    current = random.Next(1, 11);
    Console.WriteLine(current);
} while (current != 7);
```

٣. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ملاحظة

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع المجلد المتوقع، على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

٤. راجع الإخراج الخاص بك.

نظرا لأن الأرقام التي تم إنشاؤها عشوائية، ستكون نتائجك مختلفة، رغم ذلك، ستكون القيمة ٧ هي آخر رقم ستتم طباعته، حيث سيتم تقييم التعبير

المنطقي إلى false وقت إنشاء ٧ وسيخرج تدفق التنفيذ من كتلة التعليمات البرمجية.

```
2
5
8
2
7
```

٥. خذ دقيقة لمراجعة تعليماتك البرمجية.

الدرس الرئيسي لهذه المهمة الأولى، أن الكتلة البرمجية الخاصة بحلقة do-while سيتم تنفيذها مرة واحدة على الأقل، أو يمكن أن تتكرر عدداً كبيراً من المرات، ومن غير المرجح معرفة عدد التكرارات التي ستحدث، مسبقاً.

من المهم أيضًا ملاحظة أن التعليمات داخل الكتلة البرمجية هي المؤثرة على استمرار التكرار أو توقفه، لذلك التعليمات البرمجية التي تؤثر على معايير الخروج هي سبب أساسي لتحديد do-while or while بدلاً من عبارات التكرار الأخرى.

بينما يعتمد كل من foreach وfor على عوامل خارجية، على كتلة التعليمات البرمجية، لتحديد عدد تكرارات الكتلة.

كتابة عبارة while تتكرر فقط عندما يكون الرقم العشوائي أكبر من قيمة معينة

الآن دعونا نلقي نظرة على العبارة while

١. استخدم محرر Visual Studio Code لتعديل تعليماتك البرمجية كما يلي:

```
Random random = new Random();
int current = random.Next(1, 11);

/*
do
{
```

```
    current = random.Next(1, 11);
    Console.WriteLine(current);
} while (current != 7);
*/

while (current >= 3)
{
    Console.WriteLine(current);
    current = random.Next(1, 11);
}
Console.WriteLine($"Last number: {current}");
```

ملاحظة

في هذه الحالة، توضع الكلمة الأساسية while والتعبير المنطقي قبل الكتلة البرمجية، هذا يغير معنى التعليمات البرمجية ويعمل كـ "بوابة" للسماح فقط بمرور التنفيذ إذا تم تقييم التعبير المنطقي إلى صحيح true

٢. احفظ ملف التعليمات البرمجية.

أدخل dotnet run من موجه الأوامر Terminal لتشغيل التعليمات البرمجية.

٣. راجع قيم الإخراج المدرجة.

نظرا لأن الأرقام عشوائية، سوف تنتج التعليمات البرمجية تسلسلاً مختلفاً، لكل تنفيذ.

```
9
7
5
Last number: 1
```

٤. خذ دقيقة لمراجعة تعليماتك البرمجية.

في الأسطر الأولى استخدمنا random لتهيئة المتغير المسمى current سطر التعليمات البرمجية التالي "النشط" عبارة while

تكرار العبارة while متوقف على التعبير المنطقي (current >= 3) لا تُعرف القيمة التي سيتم تعيينها إلى current ولكن هناك احتمالات تؤثر على حلقة while

- إذا تمت تهيئة current إلى قيمة أكبر من أو تساوي 3 فسيرجع التعبير المنطقي true وسيمر تدفق التنفيذ داخل الكتلة البرمجية، أول شيء نقوم به هو كتابة قيمة current إلى وحدة التحكم، بعد ذلك، لا نزال داخل الكتلة، نقوم بتحديث قيمة current بقيمة عشوائية جديدة، عند هذه النقطة، يعود عنصر التحكم إلى أعلى العبارة while حيث يتم تقييم التعبير المنطقي، تستمر هذه العملية حتى يرجع التعبير المنطقي false ويتوقف تدفق تنفيذ كتلة التعليمات البرمجية.
- إذا تمت تهيئة current (في أعلى تعليماتك البرمجية) إلى قيمة أقل من 3 فسيرجع التعبير المنطقي false ولن يتم تنفيذ كتلة التعليمات البرمجية.

يكتب سطر التعليمات البرمجية النهائي قيمة current إلى وحدة التحكم، تعمل هذه التعليمة سواء تم تنفيذ الكتلة البرمجية للتكرار أم لا، وهي فرصتنا لكتابة قيمة current النهائية إلى وحدة التحكم.

استخدم عبارة continue للانتقال مباشرة إلى التعبير المنطقي

في حالات معينة، نريد وقف ما تبقى من التعليمات البرمجية داخل الكتلة البرمجية، والمتابعة إلى التكرار التالي، يمكننا أن نفعل ذلك باستخدام عبارة continue

١. استخدم محرر Visual Studio Code لتعديل التعليمات البرمجية كما يلي

```
Random random = new Random();
int current = random.Next(1, 11);
```

```
do
{
    current = random.Next(1, 11);

    if (current >= 8) continue;

    Console.WriteLine(current);
} while (current != 7);

/*
while (current >= 3)
{
    Console.WriteLine(current);
    current = random.Next(1, 11);
}
Console.WriteLine($"Last number: {current}");
*/
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية.

لاحظ أننا عدنا إلى `do-while` يضمن تكرار `do-while` تنفيذ التعليمات مرة واحدة على الأقل.

أول شيء نقوم به داخل الكتلة البرمجية هو تعيين قيمة عشوائية جديدة إلى current بعد ذلك، نتحقق لمعرفة ما إذا current أكبر من أو يساوي 8 إذا أرجع هذا التعبير true فإن الكلمة الرئيسية continue ستنقل عنصر التحكم إلى نهاية الكتلة البرمجية while وسيتم تقييم (current != 7) لذلك سوف تستمر الحلقة في التكرار طالماً أن قيمة current لا تساوي 7

مفتاح هذه الخطوة من التمرين هو سطر التعليمات البرمجية الذي يحتوي على الكلمة الرئيسية continue

```
if (current >= 8) continue;
```

نظرا لأن التعليمات البرمجية التي تكتب قيمة current إلى وحدة التحكم موجودة بعد `if (current >= 8) continue;` فإن التعليمات البرمجية تضمن أن قيمة current أكبر من أو تساوي 8 لن تتم كتابتها أبدا إلى نافذة الإخراج.

دعونا نجرب ذلك

٣. احفظ ملف التعليمات البرمجية.

أدخل dotnet run في موجه الأوامر Terminal لتشغيل التعليمات البرمجية.

٤. راجع قيم الإخراج المدرجة.

```
5
1
6
7
```

من المحتمل أن ترى نتائج مختلفة عما هو معروض هنا، رغم ذلك، لن تشاهد أي قيمة 8 أو أكبر في نافذة الإخراج، قبل أن ينتهي تنفيذ التعليمات البرمجية بالقيمة 7

٥. ضع في اعتبارك الفرق بين عبارة continue وعبارة break

كما رأيت في هذه الخطوة الأخيرة، تنقل العبارة continue التنفيذ إلى نهاية التكرار الحالي، يختلف هذا السلوك عن السلوك الذي رأيناه مع العبارة break تقوم العبارة break بإنهاء التكرار أو switch وتنقل عنصر التحكم إلى العبارة التي تلي العبارة التي قامت بإنهائها.

إذا لم يكن هناك عبارة بعد العبارة التي تم إنهاؤها، فسيتم الانتقال إلى نهاية الملف أو الأسلوب.

خلاصة

هناك بعض الأفكار الهامة التي يجب أن تأخذها عن هذا الدرس:

- تتكرر العبارة do-while من خلال كتلة تعليمات برمجية مرة واحدة على الأقل، وقد تستمر في التكرار استناداً إلى تعبير منطقي، يعتمد تقييم التعبير المنطقي عادةً على قيمة ما تم إنشاؤها أو استردادها داخل كتلة التعليمات البرمجية.
- تقوم العبارة while بتقييم تعبير منطقي أولاً، وتستمر في التكرار من خلال الكتلة البرمجية طالما أن التعبير المنطقي يقيم إلى true
- الكلمة الأساسية continue للانتقال مباشرة إلى التعبير المنطقي.

اختبر معلوماتك

١- أي مما يلي يصف بشكل صحيح سلوك عبارة **do-while or while** التكرارية؟

- تقوم العبارة do-while بتنفيذ كتلة التعليمات صفر أو عدة مرات
- تقوم العبارة while بتنفيذ كتلة التعليمات مرة واحدة على الأقل
- تقوم العبارة do-while بتنفيذ كتلة التعليمات مرة واحدة على الأقل

٢- يحتاج المطور إلى حفظ مدخلات المستخدم داخل حلقة، يدخل المستخدم تركيبة لوحة المفاتيح ctrl + Esc للخروج من التكرار عند الانتهاء من إدخال المعلومات، ما هي أفضل عبارة تكرار لهذا الغرض؟

- while
- for
- do-while

راجع إجابتك

١ تقوم العبارة do-while بتنفيذ كتلة التعليمات مرة واحدة على الأقل

صحيح تتكرر do-while من خلال الكتلة البرمجية مرة واحدة على الأقل، يخرج تنفيذ التعليمات من حلقة التكرار بمجرد تقييم while التعبير في نهاية الكتلة البرمجية إلى true

٢ do-while

صحيح do-while تسمح للمطور بالتحقق من كل إدخال من قبل المستخدم حتى يدخل تركيبة لوحة المفاتيح الخاصة للخروج

٣ إكمال نشاط التحدي باستخدام عبارات do and while

ستعزز تحديات التعليمات البرمجية ما تعلمته، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

تحدي لعبة قتال شخصيات

في بعض ألعاب لعب الأدوار، تقاتل شخصية اللاعب شخصيات وهمية، عادة ما تكون وحوشًا أو "الأشرار" في بعض الأحيان، تتكون المعركة من random value يولد قيمة عشوائية باستخدام النرد، ويتم طرح هذه القيمة من مستوي صحة الخصم، بمجرد أن تصل صحة أي من الشخصيين إلى الصفر، يخسران اللعبة.

في هذا التحدي، سوف نلخص هذا التفاعل إلى جوهره، يبدأ البطل والوحش بنفس مستوي الصحة، خلال دور البطل، سوف تنشأ قيمة عشوائية، يتم طرحها من صحة الوحش، إذا كانت صحة الوحش أكبر من الصفر، فسوف يأخذ دوره ويهاجم البطل، طالما أن كلاً من البطل والوحش يتمتعان بصحة أكبر من الصفر، تُستأنف المعركة.

تحدي التعليمات البرمجية ـ كتابة التعليمات البرمجية لتنفيذ قواعد اللعبة

فيما يلي قواعد جولة المعركة التي سوف تكتب تعليماتها:

- يجب استخدام العبارة do-while أو while كحلقة لعبة خارجية
- سيبدأ البطل والوحش ب ١٠ نقاط صحة
- ستكون قيمة جميع الهجمات بين ١ و ١٠
- سيهاجم البطل أولًا
- اطبع كمية الصحة التي فقدها الوحش، والصحة المتبقية لديه
- إذا كانت صحة الوحش أكبر من ٠ يمكنه مهاجمة البطل
- اطبع كمية الصحة التي فقدها البطل، والصحة المتبقية لديه

- استمر في تسلسل الهجوم هذا حتى تصبح صحة الوحش أو صحة البطل صفرًا أو أقل
- اطبع الفائز

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

إذا لزم الأمر، افتح Visual Studio Code ثم أكمل الخطوات التالية لإعداد ملف Program.cs في المحرر:

* في القائمة ملف، حدد فتح مجلد.

* استخدم مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه.

* في قائمة استكشاف، حدد Program.cs

* أحذف أسطر التعليمات البرمجية الموجودة.

٢. اكتب التعليمة البرمجية للعبة التي تنفذ كل قاعدة.

٣. قم بتشغيل التطبيق، وتحقق من الإخراج الخاص بك يفي بالمتطلبات.

بغض النظر عن كيفية القيام بذلك، يجب أن تنتج تعليماتك البرمجية إخراجًا مماثلاً:

```
Monster was damaged and lost 1 health and now
has 9 health.
Hero was damaged and lost 1 health and now has
9 health.
Monster was damaged and lost 7 health and now
has 2 health.
Hero was damaged and lost 6 health and now has
3 health.
Monster was damaged and lost 9 health and now
has -7 health.
Hero wins!
```

من الواضح، بسبب الطبيعة العشوائية للتعليمة البرمجية، ستكون النتيجة مختلفة في كل مرة، لذلك ستكون نتائجك مختلفة بالتأكيد عن الإخراج المعروض أعلاه، رغم ذلك، يمكنك استخدام هذا كمثال على الإخراج الذي يجب أن تخرجه تعليمتك البرمجية.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر في عرض حل لهذا التحدي.

٤ مراجعة حل نشاط تحدي do and while

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
int hero = 10;
int monster = 10;

Random dice = new Random();

do
{
    int roll = dice.Next(1, 11);
    monster -= roll;
    Console.WriteLine($"Monster was damaged and
lost {roll} health and now has {monster}
health.");

    if (monster <= 0) continue;

    roll = dice.Next(1, 11);
    hero -= roll;
    Console.WriteLine($"Hero was damaged and lost
{roll} health and now has {hero} health.");

} while (hero > 0 && monster > 0);

Console.WriteLine(hero > monster ? "Hero wins!" :
"Monster wins!");
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، لأن هناك العديد من الطرق المختلفة لتنفيذ منطق هجوم اللعبة.

بغض النظر، يجب أن يكون الإخراج مشابهًا لمثال الإخراج التالي:

```
Monster was damaged and lost 1 health and now
has 9 health.
Hero was damaged and lost 2 health and now has
8 health.
Monster was damaged and lost 1 health and now
has 8 health.
Hero was damaged and lost 4 health and now has
4 health.
Monster was damaged and lost 7 health and now
has 1 health.
Hero was damaged and lost 6 health and now has
-2 health.
Monster wins!
```

إذا نجحت، فتهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة، ستعتمد جميع الأفكار الجديدة التي نناقشها في الوحدات الأخرى على فهمك للأفكار التي تم تقديمها في هذه الوحدة.

٥ إكمال نشاط التحدي للتمييز بين عبارات التكرار do and while وعبارات التكرار

فحص الفرق بين عبارات التكرار الأخرى والعبارة do and while

كما رأيت، تدعم لغة C# أربعة أنواع من عبارات التكرار for, foreach, do-while, while

تصف وثائق Microsoft المرجعية للغة هذه العبارات على النحو التالي:

- العبارة for تنفذ نصها بينما يتم تقييم التعبير المنطقي المحدد (الشرط) إلى صحيح.
- العبارة foreach تعد عناصر مجموعة ما، وتنفيذ نصها لكل عنصر من عناصر المجموعة.
- العبارة do-while تنفذ نصها بشكل مشروط، مرة واحدة أو أكثر.
- العبارة while تنفذ نصها بشكل مشروط، صفراً أو عدة مرات.

يبدو أن تكرارات for وforeach تختلف بشكل واضح عن بعضها البعض، وأيضاً عن تكرارات do-while وwhile ومع ذلك، يبدو أن تعريفات عبارات do-while وwhile متشابهة تمامًا، معرفة متى تختار بين do-while and while يبدو اعتباطيًا، أي ليس له قاعدة واضحة، ويمكن أن يكون مربكًا بعض الشيء، قد تساعد بعض مشاريع التحدي في توضيح الاختلافات.

في هذا التحدي، سيتم تقديم شروط لثلاثة مشاريع برمجية منفصلة، سيطلب منك كل مشروع تنفيذ كتلة برمجية للتكرار، باستخدام عبارة do-while أو while ستحتاج إلى تقييم الشروط المحددة من أجل الاختيار بين do-while أو while

يمكنك التبديل بعد البدء، إذا لم ينجح اختيارك الأول كما كنت تأمل.

ملاحظة

يمكن استخدام شروط مشروع الترميز الخاص بك لمساعدتك في الاختيار بين عبارات do-while و while قد يساعدك أحيانًا ما تعرفه، أو لا تعرفه، عن التعبير المنطقي الذي سيتم تقييمه في الاختيار بين عبارات do-while و while في تمرين التحدي هذا. تضم شروط المشروع المعلومات التي سيتم استخدامها لإنشاء التعبير المنطقي

إدارة مدخلات المستخدم خلال هذا التحدي

عند استخدام عبارة Console.ReadLine() للحصول على إدخال المستخدم، من الشائع استخدام nullable type string نوع تقبل القيم الخالية (سلسلة محددة؟ designated string) لمتغير الإدخال، ثم تقييم القيمة التي أدخلها المستخدم.

يستخدم نموذج التعليمات البرمجية التالي nullable type string نوع تقبل القيم الخالية لالتقاط إدخال المستخدم، يستمر التكرار عندما تكون القيمة التي يوفرها المستخدم فارغة.

```
string? readResult;
Console.WriteLine("Enter a string:");
do
{
    readResult = Console.ReadLine();
} while (readResult == null);
```

يمكن استخدام التعبير المنطقي الذي تم تقييمه بواسطة العبارة while لضمان تلبية إدخال المستخدم لمتطلبات محددة، على سبيل المثال، إذا طلبت من المستخدم إدخال نص مكون من ثلاثة أحرف على الأقل، يمكن استخدام التعليمات البرمجية التالية:

```
string? readResult;
bool validEntry = false;
Console.WriteLine("Enter a string containing at
least three characters:");
do
{
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        if (readResult.Length >= 3)
        {
            validEntry = true;
        }
        else
        {
            Console.WriteLine("Your input is
invalid, please try again.");
        }
    }
} while (validEntry == false);
```

إذا كنت تريد استخدام Console.ReadLine() لإدخال القيم الرقمية، فستحتاج إلى تحويل قيمة string إلى نوع رقمي.

يمكن استخدام الأسلوب int.TryParse() لتحويل قيمة نصية إلى عدد صحيح، يستخدم الأسلوب معلمتين، سلسلة string سيتم تقييمها واسم متغير عدد صحيح int سيتم تعيين قيمة له، يقوم الأسلوب بإرجاع قيمة منطقية.

يوضح نموذج التعليمات البرمجية التالي _لا ينتج إخراج_ باستخدام الأسلوب int.TryParse()

```
// capture user input in a string variable
named readResult
```

```
int numericValue = 0;
bool validNumber = false;

validNumber = int.TryParse(readResult, out
numericValue);
```

إذا كانت قيمة السلسلة المعينة لـ readResult تمثل عددًا صحيحًا صالحًا، فسيتم تعيين القيمة إلى متغير العدد الصحيح المسمى numericValue وسيتم تعيين true إلى المتغير المنطقي المسمى validNumber

إذا كانت القيمة المعينة لـ readResult لا تمثل عددًا صحيحًا صالحًا، فسيتم تعيين قيمة false إلى validNumber على سبيل المثال، إذا كانت readResult تساوي "٧" فسيتم تعيين القيمة ٧ إلى numericValue

المشروع البرمجي الأول - كتابة التعليمات التي تتحقق من إدخال الأعداد الصحيحة

فيما يلي الشروط التي يجب أن ينفذها المشروع الأول:

- يجب أن يتضمن حلك إما تكراراً `while` أو `do-while`

قبل كتلة التكرار:

- يجب أن يستخدم الحل أسلوب `Console.WriteLine()` لمطالبة المستخدم بقيمة عدد صحيح بين ٥ و١٠

داخل كتلة التكرار:

- يجب أن يستخدم الحل أسلوب `Console.ReadLine()` للحصول على مدخلات المستخدم.
- يجب أن يضمن حلك أن الإدخال عدد صحيح.
- إذا لم تكن قيمة العدد الصحيح بين ٥ و١٠ يجب أن تستخدم التعليمات أسلوب `Console.WriteLine()` لمطالبة المستخدم بقيمة عدد صحيح بين ٥ و١٠
- يجب أن يضمن الحل أن تكون قيمة العدد الصحيح بين ٥ و١٠ قبل إنهاء التكرار.

أسفل (بعد) كتلة التكرار:

- يجب أن يستخدم الحل أسلوب `Console.WriteLine()` لإعلام المستخدم بأنه تم قبول قيمة الإدخال الخاصة به.

١. تأكد من أن لديك ملف Program.cs فارغ، مفتوح في Visual Studio Code

٢. اكتب التعليمات البرمجية التي تنفذ كل شرط للمشروع.

٣. شغل تطبيقك، وتحقق من أن التعليمات تتحقق من صحة إدخال المستخدم استنادا إلى المتطلبات المحددة.

على سبيل المثال، عند تشغيل التطبيق، يجب أن يرفض قيم الإدخال مثل "اثنان" و"٢" ولكن يجب أن يقبل قيمة إدخال "٧"

يجب أن يبدو إخراج وحدة التحكم مشابها للمثال التالي:

```
Enter an integer value between 5 and 10
two
Sorry, you entered an invalid number, please
try again
2
You entered 2. Please enter a number between 5
and 10.
7
Your input value (7) has been accepted.
```

المشروع البرمجي الثاني - كتابة التعليمات التي تتحقق من صحة المدخلات النصية

فيما يلي الشروط التي يجب أن ينفذها المشروع الثاني:

- يجب أن يتضمن حلك إما تكراراً while أو do-while

قبل كتلة التكرار:

- يجب أن يستخدم الحل أسلوب Console.WriteLine() لمطالبة المستخدم بأحد أسماء الوظائف الثلاثة: Administrator, Manager, or User

داخل كتلة التكرار:

- يجب أن يستخدم الحل أسلوب Console.ReadLine() للحصول على مدخلات المستخدم.
- يجب أن يضمن الحل أن القيمة التي تم إدخالها تطابق أحد خيارات الوظائف الثلاثة.
- يجب أن يستخدم الحل الأسلوب Trim() لقيمة الإدخال لتجاهل أحرف المسافة البادئة واللاحقة.
- يجب أن يستخدم الحل الأسلوب ToLower() لقيمة الإدخال لتجاهل حالة الأحرف.
- إذا لم تكن القيمة المدخلة مطابقة لأحد الخيارات الثلاثة، يجب أن تستخدم التعليمات أسلوب Console.WriteLine() لمطالبة المستخدم بإدخال صالح.

أدناه (بعد) كتلة التعليمات البرمجية للتكرار:

- يجب أن يستخدم الحل أسلوب Console.WriteLine() لإعلام المستخدم بأنه تم قبول قيمة الإدخال الخاصة به.

١. اكتب التعليمات البرمجية التي تنفذ كل شروط المشروع.

٢. شغل التطبيق وتتحقق من صحة إدخال المستخدم استناداً إلى المتطلبات المحددة.

على سبيل المثال، عند تشغيل التطبيق، يجب أن يرفض قيمة إدخال مثل Admin ولكن يجب أن يقبل قيمة إدخال administrator

يجب أن يبدو إخراج وحدة التحكم لهذا المثال مشابها لما يلي:

```
Enter your role name (Administrator, Manager,
or User)
Admin
The role name that you entered, "Admin" is not
valid. Enter your role name (Administrator,
Manager, or User)
   Administrator
Your input value (Administrator) has been
accepted
```

المشروع البرمجي الثالث ـ كتابة التعليمات التي تعالج محتويات مصفوفة نصية

فيما يلي الشروط التي يجب أن ينفذها المشروع الثالث:

يجب أن يستخدم حلك مصفوفة نصية التالية، لتمثيل الإدخال إلى منطق تعليماتك

```
string[] myStrings = new string[2] { "I like pizza. I like roast chicken. I like salad", "I like all three of the menu choices" };
```

- يجب أن يعلن حلك عن متغير عدد صحيح يسمى periodLocation يمكن استخدامه للاحتفاظ بموقع دورة الحرف داخل النص.
- يجب أن يتضمن الحل تكراراً خارجياً for or foreach يمكن استخدامه لمعالجة كل عنصر في المصفوفة، يجب تسمية المتغير النصي myString الذي ستعالجه داخل حلقات التكرار.

في الحلقة الخارجية، يجب أن يستخدم الحل أسلوب IndexOf() لفئة String للحصول على موقع النقطة الأولى في متغير myString يجب أن يكون استدعاء الأسلوب مشابهًا لـ myString.IndexOf(".") إذا لم يكن هناك نقطة في النص، فسيتم إرجاع قيمة ـ١

- يجب أن يتضمن الحل حلقة while or do-while داخلية تستخدم لمعالجة المتغير myString

في الحلقة الداخلية، يجب أن يقوم الحل باستخراج وعرض (الكتابة إلى وحدة التحكم) كل جملة مضمنة في كل نص من النصوص التي تتم معالجتها.

- في الحلقة الداخلية، يجب ألا يعرض الحل حرف النقطة.

- في الحلقة الداخلية، يجب أن يستخدم الحل الأساليب Remove(), Substring(), and TrimStart() لمعالجة معلومات النصوص.

١. اكتب التعليمات البرمجية التي تنفذ كل شروط المشروع.

٢. شغل التطبيق وتتحقق من صحة إدخال المستخدم استناداً إلى المتطلبات المحددة.

إذا كان منطق التعليمات البرمجية يعمل بشكل صحيح، يجب أن يبدو الإخراج مشابها للآتي:

```
I like pizza
I like roast chicken
I like salad
I like all three of the menu choices
```

سواء واجهتك مشكلة وتحتاج إلى نظرة خاطفة على الحل أو انتهيت بنجاح، تابع لعرض حل مشاريع هذا التحدي.

٦ مراجعة نشاط التحدي do ضد while

يجب أن تستخدم الأمثلة التالية do لأنك تعرف احتياجك إلى تنفيذ الكتلة البرمجية مرة واحدة على الأقل، يمكنك أيضاً استخدام while لتحقيق نفس النتيجة، يشعر بعض المطورين أن منطق while يجعل التعليمات البرمجية أكثر قابلية للقراءة، إذا كان هذا هو الحال بالنسبة لك، يمكنك اختيار تنفيذ while في هذه الحالة، يجب أن تدرك أن معظم المحولات البرمجية للتعليمات تعمل على تحسين تعليماتك البرمجية، عن طريق تحويل عبارات التكرار إلى do-while

تعليمات المشروع الأول

التعليمات البرمجية التالية هي أحد الحلول الممكنة لتحدي المشروع الأول من الدرس السابق.

تستخدم التعليمات البرمجية عبارة do لوجوب تنفيذ الكتلة البرمجية مرة واحدة على الأقل، يمكنك أيضاً استخدام while لتحقيق نفس النتيجة، قد يشعر بعض المطورين أن منطق while يجعل التعليمات البرمجية أكثر قابلية للقراءة، إذا كان هذا هو الحال بالنسبة لك، يمكنك اختيار تنفيذ عبارة while هنا.

```
string? readResult;
string valueEntered = "";
int numValue = 0;
bool validNumber = false;

Console.WriteLine("Enter an integer value between 5 and 10");

do
{
    readResult = Console.ReadLine();
    if (readResult != null)
    {
```

```
        valueEntered = readResult;
    }

    validNumber   =   int.TryParse(valueEntered,
out numValue);

    if (validNumber == true)
    {
        if (numValue <= 5 || numValue >= 10)
        {
            validNumber = false;
            Console.WriteLine($"You     entered
{numValue}. Please enter a number between 5 and
10.");
        }
    }
    else
    {
        Console.WriteLine("Sorry,  you  entered
an invalid number, please try again");
    }
} while (validNumber == false);

Console.WriteLine($"Your      input      value
({numValue}) has been accepted.");

readResult = Console.ReadLine();
```

تعليمات المشروع الثاني

التعليمات البرمجية التالية هي أحد الحلول الممكنة لتحدي المشروع الثاني من الدرس السابق.

```
string? readResult;
string roleName = "";
bool validEntry = false;

do
{
    Console.WriteLine("Enter your role name
(Administrator, Manager, or User)");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        roleName = readResult.Trim();
    }

    if (roleName.ToLower() == "administrator"
|| roleName.ToLower() == "manager" ||
roleName.ToLower() == "user")
    {
        validEntry = true;
    }
    else
    {
        Console.Write($"The role name that you
entered, \"{roleName}\" is not valid. ");
    }

} while (validEntry == false);

Console.WriteLine($"Your input value
({roleName}) has been accepted.");
readResult = Console.ReadLine();
```

تعليمات المشروع الثالث

التعليمات البرمجية التالية هي أحد الحلول الممكنة لتحدي المشروع الثالث من الدرس السابق.

تستخدم التعليمات عبارة for للحلقة الخارجية لأنه لا يمكنك تعديل القيمة المخصصة لـ متغير التكرار foreach iteration variable يمكنك حل هذه المشكلة بالإعلان عن متغير نصي إضافي، داخل حلقة foreach ولكنك ستضيف تعقيدًا غير مرغوب إلى منطق التعليمات، بمعنى، سيؤدي استخدام عبارة التكرار foreach (string myString in myStrings) ثم محاولة معالجة متغير myString ستؤدي إلى حدوث خطأ في الترجمة.

تستخدم التعليمات البرمجية عبارة while للحلقة الداخلية، اعتماداً على قيمة سلسلة البيانات، قد لا يتم تنفيذ الكتلة البرمجية (عندما لا يحتوي النص على نقطة) يجب عدم استخدام عبارة do في الحالات التي قد لا تحتاج فيها كتلة التكرار إلى التنفيذ.

```
string[] myStrings = new string[2] { "I like
pizza. I like roast chicken. I like salad", "I
like all three of the menu choices" };
int stringsCount = myStrings.Length;

string myString = "";
int periodLocation = 0;

for (int i = 0; i < stringsCount; i++)
{
    myString = myStrings[i];
    periodLocation = myString.IndexOf(".");

    string mySentence;
```

```
    // extract sentences from each string and
display them one at a time
    while (periodLocation != -1)
    {

        // first sentence is the string value
to the left of the period location
        mySentence =
myString.Remove(periodLocation);

        // the remainder of myString is the
string value to the right of the location
        myString =
myString.Substring(periodLocation + 1);

        // remove any leading white-space from
myString
        myString = myString.TrimStart();

        // update the comma location and
increment the counter
        periodLocation = myString.IndexOf(".");

        Console.WriteLine(mySentence);
    }

    mySentence = myString.Trim();
    Console.WriteLine(mySentence);
}
```

٧ اختبر معلوماتك

١- يحتاج مطور إلى التكرار من خلال مصفوفة، عدد العناصر في المصفوفة غير معروف، تقوم التعليمات البرمجية داخل كتلة التكرار بفحص كل عنصر بالتسلسل، لتحديد عنصر يطابق المعايير المحددة، رقم الفهرس لعنصر المصفوفة المحدد غير مهم، تتم الإشارة إلى العنصر المحدد عدة مرات داخل كتلة، ما نوع عبارة التكرار الأنسب لهذا السيناريو؟

- foreach
- for
- while

٢- يحتاج مطور إلى التقاط قائمة بالعناصر من المستخدم، سيدخل المستخدم تركيبة لوحة المفاتيح ctrl + Esc للخروج، ما هي أفضل عبارة تكرار لهذا الغرض؟

- foreach
- do-while
- while

راجع إجابتك

١ Foreach

صحيح من الممكن استخدام أي من أنواع التكرار في الاختيارات، للشروط الموضحة، ولكن foreach الأنسب لهذا السيناريو من السهل تنفيذ foreach عندما يكون عدد عناصر المصفوفة غير معروف، تعد foreach اختياراً جيداً عندما لا يكون رقم الفهرس لعنصر المصفوفة المحدد مهماً، قد تعمل foreach بشكل أسرع عند استخدام العنصر المحدد عدة مرات داخل كتلة التعليمات البرمجية.

٢ do-while

صحيح ستسمح العبارة do-while بالتحقق من كل إدخال من قبل المستخدم حتى يدخل تركيبة لوحة المفاتيح الخاصة للخروج

٨ الملخص

كان هدفك هو استخدام عبارات while and do-while لتنفيذ التكرارات، تعتبر عبارات while and do-while فريدة من نوعها، لأن تعليمات كتلتها البرمجية هي من يحدد إذا كان يجب أن يستمر تدفق التنفيذ أم يتوقف.

باستخدام العبارة do-while قمت بتنفيذ كتلة من التعليمات البرمجية مرة واحدة، قبل تقييم تعبير منطقي، وربما إنهاء التكرار، باستخدام العبارة while أجريت تقييم التعبير المنطقي على الفور، وتابعت تقييمه للخروج من التكرار، لقد استخدمت العبارة continue داخل كتلة التعليمات البرمجية للانتقال مباشرة إلى التعبير المنطقي.

لقد طورت تطبيقاً عملياً يستخدم عبارات do-while and continue لمحاكاة معركة في لعبة قتال، تتضمن سيناريوهات العالم الحقيقي التي تتضمن عبارات do-while and while العمل مع تدفقات البيانات من الملفات، أو من الإنترنت، أو أي سيناريو آخر حيث ستستمر في إجراء التكرار حتى يتم استيفاء الشرط.

بدون عبارات do-while وأثناء التكرار، سيكون من الصعب كتابة تعليمات التكرار، الذي يحدد شرط الخروج داخل كتلة التكرار، والمحافظة عليه.

الوحدة السادسة

مشروع موجه ـ تطوير هياكل التفريع وحلقات التكرار المشروطة

اكتساب خبرة في تطوير تطبيق وحدة تحكم، ينفذ عبارات التحديد والتكرار لتحقيق مواصفات التطبيق.

الأهداف التعليمية

في هذه الوحدة، ستتدرب على كيفية القيام بما يلي:

- استخدم Visual Studio Code لتطوير تطبيق وحدة تحكم C# يستخدم مجموعة من عبارات التحديد والتكرار، لتنفيذ مهام سير العمل المنطقية، وفقاً لبيانات التطبيق المتوفرة وتفاعلات المستخدم.
- تقييم الشروط الأساسية، واتخذ قراراً مستنيراً عند الاختيار بين عبارات `if-else`و`switch` وبين عبارات `foreach` و`for` و`while` و `do`
- متغيرات النطاق على مستوى مناسب داخل التطبيق.

محتويات الوحدة:

١- مقدمة

٢- الإعداد

٣- تمرين - إنشاء نموذج للبيانات وحلقات تحديد القائمة

٤- تمرين - كتابة التعليمات لعرض جميع بيانات مصفوفة ourAnimals

٥- تمرين - إنشاء واختبار حلقة لإدخال بيانات الحيوانات الجديدة

٦- تمرين - كتابة التعليمات لقراءة وحفظ البيانات الجديدة لمصفوفة ourAnimals

٧- اختبر معلوماتك

٨- الملخص

١ المقدمة

من الشائع أن يبدأ المطورون مشروعاً عن طريق تطوير الميزات، التي تستورد بيانات التطبيق أو تنشئها، بمجرد أن يتمكن تطبيقك من الوصول إلى البيانات التي يعتمد عليها، يمكنك البدء في تطوير الميزات التي تعالج البيانات، وتنشئ التقارير.

لنفترض أنك مطور يحب دعم مجتمعك، بدأت أنت وبعض أصدقائك عملاً يساعد على العثور على منازل جديدة للقطط والكلاب الضالة، بدأ عملك صغيراً، مع بعض الحيوانات الضالة فقط، ولكنه بدأ ينمو، تريد إنشاء تطبيق من شأنه أن يساعدك على مطابقة الحيوانات في دار الرعاية الخاصة بك، مع الأشخاص الذين يبحثون عن حيوان أليف، وجدت أنه من المهم أن يكون لديك وصف مفصل للحيوانات، لمشاركتها مع المتبنيين، بالإضافة إلى ذلك، فإن القدرة على وصف شخصية الكلب أو القط يجعلها أكثر جاذبية للمالكين المحتملين، قررت إنشاء تطبيق يساعدك على إدارة معلومات حول الكلاب والقطط التي تهتم بها.

ترشدك هذه الوحدة خلال عملية تطوير الميزات التي تركز على البيانات لتطبيق Contoso Pets ستستخدم عبارات التحديد والتكرار لإنشاء بيانات نموذجية، وإدراج الحيوانات في الرعاية، وإضافة حيوانات جديدة، في جميع أنحاء التطبيق، ستستخدم المتغيرات، والتعبيرات، للتحكم في تنفيذ فروع التعليمات البرمجية، تتضمن أيضاً تحديد نطاق المتغيرات بشكل مناسب.

ميزات التطبيق الذي تطوره:

- إضافة نموذج بيانات معروفة مسبقاً إلى مصفوفة الحيوانات الأليفة.
- تكرار كتلة التعليمات البرمجية خيارات القائمة، وتحديد المستخدم menu options and user selection لإنشاء الحلقة الخارجية للتطبيق.
- تنفيذ فروع التعليمات البرمجية المطابقة لاختيارات قائمة المستخدم.
- عرض جميع المعلومات الموجودة في مصفوفة المستخدم، لتخزين بيانات الحيوانات الأليفة استناداً إلى تحديد قائمة المستخدم based on user's menu selection

- تكرار كتلة التعليمات البرمجية "إضافة معلومات حيوانية جديدة" التي تمكن المستخدم من إضافة حيوان جديد أو أكثر، إلى مجموعة الحيوانات الأليفة (استنادًا إلى تحديد قائمة المستخدم)

في نهاية هذه الوحدة، ستكون قادراً على تطوير التعليمات البرمجية التي تجمع بين عبارات التحديد، والتكرار، لتحقيق أهداف تصميم تطبيقك.

٢ الإعداد

في هذا المشروع الإرشادي، ستستخدم Visual Studio Code لتطوير الإصدار الأولي من تطبيق C# سيستخدم تطبيقك التعبيرات المنطقية، وعبارات التحديد، وعبارات التكرار، لتنفيذ ميزات مواصفات التصميم، أثناء تطوير التطبيق، ستحتاج إلى تحديد نطاق المتغيرات على المستوى المناسب.

نظرة عامة على المشروع

أنت تعمل على تطبيق Contoso Pets وهو تطبيق يساعد على إيواء الحيوانات الأليفة في منازل جديدة، مواصفات التطبيق هي:

- إنشاء تطبيق وحدة تحكم C#
- تخزين بيانات التطبيق في مصفوفة نصية، متعدد الأبعاد، تسمى ourAnimals
- تضم مصفوفة ourAnimals خصائص الحيوانات الأليفة التالية لكل حيوان:
  - رقم معرف الحيوانات الأليفة.
  - أنواع الحيوانات الأليفة (القط أو الكلب).
  - عمر الحيوانات الأليفة (بالسنوات).
  - وصف للحالة/الخصائص البدنية للحيوان الأليف.
  - وصف لشخصية الحيوان الأليف.
  - لقب الحيوان الأليف.
- تنفيذ عينة من مجموعة البيانات تمثل الكلاب والقطط الموجودة حاليًا في رعايتك.
- عرض خيارات القائمة للوصول إلى الميزات الرئيسية للتطبيق.
- تمكنك الميزات الرئيسية من المهام التالية:

- سرد معلومات جميع الحيوانات الأليفة في مصفوفة ourAnimals
- إضافة حيوانات جديدة إلى مصفوفة ourAnimals تنطبق الشروط التالية:
  - يجب إدخال أنواع الحيوانات الأليفة (الكلب أو القط) عند إضافة حيوان جديد إلى مصفوفة ourAnimals
  - يجب إنشاء معرف حيوان أليف برمجياً، عند إضافة حيوان جديد إلى مصفوفة ourAnimals
  - قد تكون بعض الخصائص الجسدية للحيوان غير معروفة حتى فحص الطبيب البيطري، على سبيل المثال: العمر والسلالة وحالة التعقيم.
  - قد يكون لقب الحيوان وشخصيته غير معروفين عند وصول الحيوان أول مرة.
- تأكد من اكتمال أعمار الحيوانات، والأوصاف البدنية، ربما يكون هذا مطلوباً بعد فحص الطبيب البيطري.
- تأكد من اكتمال الأسماء المستعارة للحيوانات، ووصف الشخصية (يمكن أن يحدث هذا الإجراء بعد أن يعرف الفريق حيوانا أليفاً).
- تحرير عمر الحيوان (إذا كان تاريخ ميلاد الحيوان معروفاً، أو ولد في دار الرعاية).
- تحرير وصف شخصية الحيوان (قد يتصرف حيوان بشكل مختلف بعد قضاء المزيد من الوقت في رعايتنا).
- عرض جميع القطط التي تفي بالمظاهر البدنية المحددة للمستخدم.
- عرض جميع الكلاب التي تفي بالمظاهر البدنية المحددة للمستخدم.

تم بالفعل إكمال نسخة الأولية من التطبيق، يتضمن مشروع Starter لوحدة المشروع الإرشادية هذه ملف Program.cs الذي يوفر ميزات التعليمات البرمجية التالية

- تعلن التعليمات البرمجية عن المتغيرات المستخدمة لجمع بيانات الحيوانات، وتحديدات عناصر القائمة ومعالجتها.
- تعلن التعليمات البرمجية عن مصفوفة ourAnimals
- تستخدم التعليمات البرمجية حلقة حول عبارة if-elseif-else لملء مصفوفة ourAnimals مع نموذج مجموعة بيانات.
- تعرض التعليمات البرمجية خيارات القائمة الرئيسية التالية لتحديد المستخدم:

١. سرد جميع معلومات الحيوانات الموجودة حالياً.

٢. تعيين قيم لحقول مصفوفة ourAnimals

٣. تأكد من اكتمال أعمار الحيوانيات، والأوصاف البدنية.

٤. تأكد من اكتمال الأسماء المستعارة للحيوانات، ووصف الشخصية.

٥. تحرير عمر الحيوان.

٦. تحرير وصف شخصية الحيوان.

٧. عرض جميع القطط ذات الخصائص المحددة.

٨. عرض جميع الكلاب ذات الخصائص المحددة.

أختار عنصر القائمة أو اكتب "إنهاء" للخروج من البرنامج.

- تقرأ التعليمات اختيار المستخدم عنصر القائمة، وتعرض رسالة تؤكد اختياره.

هدفك هو تطوير الميزات التي تنفذ خياري القائمة الأولين، لتحقيق هذا الهدف، ستكمل المهام التالية:

١. قم بتعديل التعليمات البرمجية المستخدمة، لإنشاء نموذج البيانات للتطبيق.

٢. إنشاء حلقة تكرار حول القائمة الرئيسية، وإنشاء عبارة تحديد تنشئ فرع التعليمات البرمجية لكل خيار قائمة.

٣. اكتب التعليمات البرمجية لعرض جميع بيانات مصفوفة ourAnimals الخيار الأول في القائمة.

٤. إنشاء حلقة لإدخال البيانات الجديدة لمصفوفة ourAnimals الخيار الثاني في القائمة جزء أول.

٥. اكتب التعليمات البرمجية لقراءة وحفظ البيانات الجديدة لمصفوفة ourAnimals الخيار الثاني في القائمة جزء ثاني.

ستختبر تطبيقك في كل مرحلة من مراحل عملية التطوير.

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي.

١. لتنزيل ملف مضغوط يحتوي على رمز مشروع Starter حدد الارتباط التالي: Lab Files

٢. فك ضغط ملفات التنزيل.

ضع في اعتبارك استخدام الكمبيوتر كبيئة تطوير حتى تتمكن من الوصول إلى التعليمات بعد إكمال هذه الوحدة، إذا كنت لا تستخدم الكمبيوتر كبيئة تطوير فلا بأس.

- على جهازك المحلي، انتقل إلى مجلد التنزيلات downloads folder
- انقر بزر الماوس الأيمن فوق Guided-project-branching-looping-CSharp-main.zip ثم حدد استخراج الكل Extract all
- حدد إظهار الملفات المستخرجة عند اكتمالها Show extracted files when complete ثم حدد استخراج Extract
- دون موقع المجلد المستخرج.

٣. انسخ مجلد GuidedProject المستخرج إلى سطح مكتب Windows

ملاحظة

إذا كان هناك مجلد باسم GuidedProject موجود بالفعل، يمكنك تحديد استبدال الملفات في الوجهة لإكمال عملية النسخ.

٤. افتح مجلد GuidedProject الجديد في Visual Studio Code

- افتح Visual Studio Code
- في القائمة **File** حدد **Open Folder**
- انتقل إلى سطح المكتب، وحدد موقع المجلد GuidedProject
- حدد **GuidedProject** ثم حدد **Select Folder**

يجب أن تظهر قائمة عرض EXPLORER مجلد GuidedProject ومجلدين فرعيين باسم Final وStarter

أنت الآن جاهز لبدء تمارين المشروع الإرشادي. حظ سعيد

٣ تمرين ـ إنشاء نموذج للبيانات وحلقات تحديد القائمة

في هذا التمرين، يمكنك مراجعة التعليمات البرمجية في مشروع Starter وتنفيذ بعض مهام تنظيف التعليمات البرمجية، ثم البدء في إضافة ميزات إلى تطبيقك الخاص.

المهام التي تكملها أثناء هذا التمرين هي:

١. **مراجعة التعليمات البرمجية:** راجع محتويات ملف Program.cs

٢. **عينة البيانات:** تحويل بنية `if-elseif-else` إلى بنية حالة التبديل `switch-case` التي تحسن قابلية القراءة.

٣. **حلقة القائمة:** إحاطة القائمة الرئيسية واختيار عنصر القائمة في حلقة تتكرر حتى يدخل المستخدم "exit" للخروج

٤. **تحديدات القائمة:** اكتب التعليمات البرمجية لعبارة `switch-case` التي تنشئ فروع تعليمات برمجية منفصلة، لكل خيار قائمة.

٥. **فروع التعليمات البرمجية:** اكتب عنصراً نائباً داخل فروع عناصر القائمة، التي توفر ملاحظات المستخدم حتى يتم تطوير ميزات التطبيق.

٦. **اختبار التحقق:** قم بإجراء اختبارات التحقق للتعليمات البرمجية التي تطورها في هذا التمرين.

مراجعة محتويات ملف Program.cs

في هذه المهمة، يمكنك إكمال معاينة التعليمة البرمجية لمشروع Starter يحتوي ملف Program.cs على إصدار أولي من التطبيق الذي تقوم بتحديثه أثناء هذه الوحدة، تنشئ التعليمات البرمجية الموجودة بيانات نموذجية للتطبيق، وتعرض مجموعة من خيارات القائمة، تمثل خيارات القائمة الميزات الرئيسية للتطبيق.

١. تأكد من **فتح مجلد GuidedProject** في Visual Studio Code

يتضمن الدرس السابق قسم الإعداد، الذي يصف كيفية فتح مشروع Starter إذا لزم الأمر، فارجع واتبع إرشادات الإعداد.

٢. في **قائمة استكشاف EXPLORER** قم **بتوسيع مجلد Starter** ثم حدد **Program.cs**

عند تحديد ملف Program.cs يتم فتح محتويات الملف في محرر Visual Studio Code

إذا لم تكن قائمة EXPLORER مفتوحة، فحدد EXPLORER من شريط نشاط Visual Studio Code يوجد الزر EXPLORER في أعلى شريط النشاط.

٣. خذ بضع دقائق لمراجعة التعليمات البرمجية في ملف Program.cs

٤. لاحظ أن الجزء العلوي من التعليمات البرمجية يبدأ ببعض إعلانات المتغيرات.

```
// the ourAnimals array will store the
following:
string animalSpecies = "";
string animalID = "";
string animalAge = "";
string animalPhysicalDescription = "";
string animalPersonalityDescription = "";
string animalNickname = "";

// variables that support data entry
int maxPets = 8;
string? readResult;
string menuSelection = "";

// array used to store runtime data, there is
no persisted data
string[,] ourAnimals = new string[maxPets, 6];
```

في الجزء العلوي من الملف، ترى سطر تعليق، وقائمة من المتغيرات، هذه المتغيرات animalSpecies, animalNickname للاحتفاظ بقيم خصائص الحيوانات الأليفة.

لاحقاً في التعليمات البرمجية تقوم بتعيين القيم المميزة إلى مصفوفة نصية، متعدد الأبعاد تسمى ourAnimals تتم تهيئة كل من هذه المتغيرات لتحتوي على سلسلة فارغة "" تم الإعلان عن المصفوفة ourAnimals أسفل التعليمات.

المجموعة التالية من المتغيرات هي مزيج من المتغيرات string والمتغيرات int التي ستستخدمها لإنشاء بيانات نموذجية، وقراءة إدخال المستخدم، ووضع معايير الخروج لحلقة البرنامج الرئيسية، ربما لاحظت سطر التعليمات البرمجية `string? readResult;` عند استخدامه في تعريف متغير مثل هذا، يعرف الحرف **?** نوع متغير يقبل القيم الخالية، عند قراءة المستخدم أدخل القيم باستخدام الأسلوب Console.ReadLine() **من الأفضل استخدام نوع يقبل القيم الخالية**.

المتغير النهائي هو مصفوفة نصية ثنائية الأبعاد، تسمى ourAnimals نظراً لأنك تقوم بإنشاء مثيل للمصفوفة دون تهيئة أي قيم، يمكنك استخدام عامل التشغيل new (يتم استخدام عامل التشغيل new لإنشاء مثيل جديد من نوع) يتم تعريف عدد الصفوف بواسطة maxPets الذي تمت تهيئته إلى ثمانية، عدد الخصائص التي تقوم بتخزينها ستة، وهي المتغيرات النصية string التي قمت بفحصها أعلاه.

٥. قم بالتمرير لأسفل لفحص حلقة for التي تحتوي على عبارة تحديد if-elseif-else

ملحوظة

إذا رأيت عبارة التبديل switch بدلاً من if-elseif-else داخل كتلة التعليمات البرمجية الخاصة بحلقة for فتأكد من أنك تقوم بمراجعة ملف Program.cs في المجلد Starter وليس ملف Program.cs في المجلد النهائي، لا تريد إجراء تغييرات على التعليمات البرمجية الموجودة في المجلد النهائي، قد تحتاج إلى استخدام المجلد النهائي كمرجع لاحقًا في هذه الوحدة.

٦. لاحظ أن الحلقة for تستخدم المتغير maxPets لإنشاء حد أعلى لعدد التكرارات.

٧. لاحظ أن الإنشاء if-elseif-else يفرع التعليمات البرمجية بشكل انتقائي بناء على خصائص الحيوانات الأليفة.

يتم استخدام البنية if-elseif-else لتعريف قيم مختلفة للتكرارات الأربعة الأولى من الحلقة for بعد التكرار الرابع، يتم تعيين متغير نصي فارغ أو صفري لكافة الخصائص.

يتم تعيين قيم متغيرات خصائص الحيوان إلى مصفوفة ourAnimals في أسفل الحلقة for

٨. قم بالتمرير إلى أسفل ملف التعليمات البرمجية، ثم افحص التعليمات البرمجية المستخدمة لعرض خيارات القائمة، وقراءة تحديد المستخدم.

٩. تعيين القيمة المرجعة بواسطة الأسلوب Console.ReadLine() إلى المتغير readResult الذي يقبل القيم الخالية.

استخدام متغير نصي يقبل القيم الخالية، هو أفضل ممارسة لالتقاط الإدخال من أسلوب ReadLine() بمجرد التحقق من أن قيمة الإدخال ليست خالية، يمكنك تعيين القيمة إلى متغير قياسي يسمى menuSelection تمكنك هذه التقنية من تقييم قيمة تحديد القائمة، دون القلق بشأن القيم الفارغة، ستنشئ العديد من الطرق التي تقبل النصوص كمعلمة إدخال خطأ، إذا تم تمرير قيمة فارغة، إذا لم تتبع نمط الإدخال هذا، فمن المحتمل أن يقوم المحول البرمجي للتعليمات البرمجية بإنشاء تحذير عند إنشاء مشروعك.

تعكس الأسطر النهائية لملف Program.cs تحديد خيار القائمة ثم توقف التنفيذ مؤقتاً حتى يتم الضغط على المفتاح Enter

تحويل عبارة if إلى عبارة switch

في هذه المهمة، يمكنك تحويل البنية الموجودة if-elseif-else إلى switch-case تعمل العبارة switch على تحسين سهولة قراءة التعليمات البرمجية.

١. قم بالتمرير لأعلى إلى بداية الحلقة for المستخدمة لإنشاء نموذج البيانات.

٢. لاحظ أن البنية if-elseif-else تبدأ بالبيان if التالي وكتلة التعليمات البرمجية:

```
if (i == 0)
{
    animalSpecies = "dog";
    animalID = "d1";
    animalAge = "2";
    animalPhysicalDescription = "medium sized
cream colored female golden retriever weighing
about 65 pounds. housebroken.";
    animalPersonalityDescription = "loves to
have her belly rubbed and likes to chase her
tail. gives lots of kisses.";
    animalNickname = "lola";
}
```

٣. استبدل عبارة if الأولية وكتلة التعليمات البرمجية بالتعليمات التالية:

```
switch (i)
{
    case 0:
        animalSpecies = "dog";
        animalID = "d1";
        animalAge = "2";
        animalPhysicalDescription = "medium
sized cream colored female golden retriever
weighing about 65 pounds. housebroken.";
        animalPersonalityDescription = "loves
to have her belly rubbed and likes to chase her
tail. gives lots of kisses.";
        animalNickname = "lola";
        break;
```

تقوم تعليمة الحالة `case 0:` بتنفيذ نفس التحديد `if (i == 0)` الذي تحل محله، ستقوم بإجراء عمليات الاستبدال المقابلة لإكمال التحويل من العبارة `if-elseif-else` إلى العبارة `switch-case`

٤. لاحظ الخط الأحمر المتعرج الذي يظهر الآن أسفل `;` في نهاية عبارة `break`

يستخدم Visual Studio Code خطًا أحمر متعرجًا لمساعدتك في اكتشاف المشكلات في تعليماتك البرمجية، في هذه الحالة، هناك بعض القضايا: أولاً، لم تقم بإغلاق مقطع التعليمات البرمجية الخاص بعبارة `switch` وأيضًا، لديك `else if` بدون `if` وهو أمر غير مسموح به، يمكنك إصلاح كل من هذه المشكلات أثناء إكمال التحويل من `switch` إلى `if`

٥. استبدل العبارة `else if (i == 1)` وكتلة التعليمات البرمجية بالكود التالي:

```
case 1:
    animalSpecies = "dog";
    animalID = "d2";
    animalAge = "9";
    animalPhysicalDescription = "large reddish-
brown male golden retriever weighing about 85
pounds. housebroken.";
    animalPersonalityDescription = "loves to
have his ears rubbed when he greets you at the
door, or at any time! loves to lean-in and give
doggy hugs.";
    animalNickname = "loki";
    break;
```

٦. استبدل عبارة `else if (i == 2)` وكتلة التعليمات البرمجية بالكود التالي:

```
case 2:
    animalSpecies = "cat";
    animalID = "c3";
    animalAge = "1";
```

```
    animalPhysicalDescription = "small white
female weighing about 8 pounds. litter box
trained.";
    animalPersonalityDescription = "friendly";
    animalNickname = "Puss";
    break;
```

٧. استبدل عبارة else if (i == 3) وكتلة التعليمات البرمجية بالكود التالي:

```
case 3:
    animalSpecies = "cat";
    animalID = "c4";
    animalAge = "?";
    animalPhysicalDescription = "";
    animalPersonalityDescription = "";
    animalNickname = "";
    break;
```

٨. استبدل عبارة else وكتلة التعليمات البرمجية بالكود التالي:

```
default:
    animalSpecies = "";
    animalID = "";
    animalAge = "";
    animalPhysicalDescription = "";
    animalPersonalityDescription = "";
    animalNickname = "";
    break;

}
```

٩. لاحظ الآن استبدال عبارة if-elseif-else بالكامل بعبارة switch-case واختفاء الخط الأحمر المتعرج.

إذا بقيت أجزاء من عبارة if-elseif-else أو إذا كانت عبارة switch-case غير مكتملة، فتحقق لمعرفة ما إذا كنت قد فاتك أي خطوة.

١٠. يجب أن يكون للبنية المكتملة switch-case مشابهة للتعليمات البرمجية التالية:

ملاحظة

تمت إزالة المتغيرات المستخدمة لتعيين قيم خصائص الحيوانات الأليفة من نموذج التعليمات البرمجية أدناه، لتقليل عدد أسطر التعليمات البرمجية. يجب أن تتضمن التعليمات البرمجية تعيينات متغيرات string لكل نمط حالة، بما في ذلك الحالة الافتراضية.

```
switch (i)
{
    case 0:
        // variable assignments were removed
for this view of the structure
        break;

    case 1:
        // variable assignments were removed
for this view of the structure
        break;

    case 2:
        // variable assignments were removed
for this view of the structure
        break;

    case 3:
        // variable assignments were removed
for this view of the structure
```

```
            break;

        default:
            // variable assignments were removed
for this view of the structure
            break;
}
```

تذكر عند تنفيذ قسم switch section غير مسموح له بالانتقال إلى قسم التبديل التالي، يتم استخدام عبارة break; من البداية إلى النهاية، لضمان عدم حدوث أخطاء.

١١. من قائمة ملف File أختر حفظ Save

١٢. في قائمة استكشاف EXPLORER انقر بزر الماوس الأيمن فوق Starter ثم حدد Open in Integrated Terminal

يجب فتح لوحة TERMINAL أسفل منطقة محرر التعليمات البرمجية.

هناك عدة طرق لفتح موجة الأوامر integrated terminal على سبيل المثال، يوفر شريط النشاط العلوي الوصول إلى TERMINAL من كل **من القائمة View** وقائمة **Terminal** يمكنك أيضاً معرفة اختصارات لوحة المفاتيح التي تفتح TERMINAL كل الطرق مقبولة.

١٣. لاحظ أن TERMINAL يتضمن موجه سطر أوامر، يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop\GuidedProject\Starter>
```

يمكنك استخدام TERMINAL لتشغيل أوامر واجهة سطر الأوامر(CLI) مثل `dotnet run` and `dotnet build` سيقوم الأمر `dotnet build` بتجميع التعليمات البرمجية، وعرض رسائل الخطأ والتحذير، المتعلقة ببناء جملة تعليماتك البرمجية.

هام: تأكد من أن موجه أوامر terminal command prompt مفتوح في مجلد مشروع Starter حيث توجد ملفات Starter.csproj وملفات

Program.cs عند تشغيل الأوامر في موجة الأوامر، ستقوم بتنفيذ إجراءات باستخدام موقع المجلد، إذا قمت بتشغيل dotnet run or dotnet build من موقع مجلد لا يحتوي على ملفاتك، فسينشئ الأمر رسائل خطأ.

١٤. في موجه الأوامر TERMINAL لإنشاء التعليمات البرمجية للمشروع، أدخل الأمر التالي `dotnet build`

بعد عدة ثوان، يجب أن تشاهد رسالة تخبرك بنجاح البناء، وأن لديك 0 تحذير Warning(s) و0 خطأ Error(s)

```
Determining projects to restore...
All projects are up-to-date for restore.
Starter                                                    ->
C:\Users\someuser\Desktop\GuidedProject\Starter\bi
n\Debug\net6.0\Starter.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)
```

١٥. إذا رأيت رسائل خطأ أو تحذير، فستحتاج إلى إصلاحها قبل المتابعة.

تدل رسائل الخطأ والتحذير على سطر التعليمات البرمجية الذي يمكن العثور على المشكلة فيه، الرسالة التالية هي مثال لرسالة خطأ Build FAILED

```
C:\Users\someuser\Desktop\GuidedProject\Starter\Pr
ogram.cs(53,18): error CS1002: ; expected
```

```
[C:\Users\someuser\Desktop\GuidedProject\Starter\S
tarter.csproj]
```

تخبرك هذه الرسالة بنوع الخطأ الذي تم اكتشافه، ومكان العثور عليه، في هذه الحالة، تخبرك الرسالة بأن الملف Program.cs يحتوي على خطأ `error CS1002: ; expected. The ; expected` يقترح أنك نسيت تضمين `;` في نهاية عبارة `Program.cs(53,18)` يخبرك جزء الرسالة أن الخطأ موجود في سطر التعليمات البرمجية 53 في موضع 18 حرفا من اليسار.

يمنع خطأ في بناء الجملة مثل هذا نجاح البنية (فشل الإنشاء) توفر بعض رسائل الإنشاء "تحذير" بدلاً من "خطأ" ما يعني أن هناك شيئاً يجب الاهتمام به، ولكن يمكنك محاولة تشغيل البرنامج على أي حال (نجح الإنشاء).

بمجرد إصلاح المشكلات، وحفظ التحديثات، يمكنك تشغيل dotnet build الأمر مرة أخرى، تابع حتى يكون لديك 0 تحذير Warning(s) و0 خطأ Error(s)

ملاحظة

إذا واجهت صعوبة في حل مشكلة ما بنفسك، يمكنك فحص التعليمات البرمجية Program.cs في المجلد النهائي المضمن كجزء من التنزيل الذي أكملته أثناء الإعداد، تمثل التعليمات البرمجية Program.cs في المجلد النهائي ختام جميع التدريبات في هذه الوحدة، لذلك ستتضمن التعليمات البرمجية التي لم تقم بإنشائها بعد، قد تبدو مختلفة إلى حد كبير عن التعليمات البرمجية Program.cs التي قمت بتطويرها في هذه المرحلة في المشروع الإرشادي، ومع ذلك، يمكنك محاولة فحص التعليمات البرمجية Program.cs في Final لمساعدتك في عزل مشكلة في التعليمات البرمجية وإصلاحها، تجنب استخدام التعليمات البرمجية في المجلد النهائي كدليل، تذكر أن المطورين يتعلمون من أخطائهم، وأن كل مطور يقضي بعض الوقت في العثور على الأخطاء وإصلاحها.

إنشاء حلقة قائمة البرنامج program menu loop

في هذه المهمة، يمكنك إنشاء حلقة عمل تحيط بخيارات القائمة، والتعليمات البرمجية التي تقرأ إدخالات المستخدم، تضمن هذه الحلقة تحديث القائمة الرئيسية للمستخدم في كل مرة يقوم فيها باختيار القائمة، تتكرر هذه الحلقة حتى يختار المستخدم الخروج من البرنامج

في هذه المهمة، يمكنك إنشاء حلقة do تحيط بخيارات القائمة، والتعليمات البرمجية التي تحفظ إدخال المستخدم، تضمن حلقة التكرار هذه تحديث القائمة

الرئيسية للمستخدم في كل مرة يقوم فيها بتحديد قائمة، يتكرر هذا التكرار حتى يختار المستخدم إنهاء البرنامج.

١. في المحرر، أعلى التعليمات البرمجية المستخدمة لعرض خيارات القائمة، حدد موقع التعليق التالي:

```
// display the top-level menu options
```

٢. أنشئ سطر فارغ أعلى التعليق، ثم أدخل التعليمات التالية:

```
do
{
```

من خلال إحاطة خيارات قائمة البرنامج داخل حلقة do تأكد من أن المستخدم يرى خيارات القائمة مرة واحدة على الأقل.

٣. أذهب إلى أسفل ملف التعليمات البرمجية.

٤. أنشئ سطر فارغ أسفل التعليمات التي توقف تنفيذ التعليمات البرمجية مؤقتا، ثم قم بتحديث التعليمات البرمجية كما يلي:

```
// pause code execution
readResult = Console.ReadLine();

} while ();
```

لاحظ أنك أغلقت كتلة التعليمات البرمجية للحلقة do وبدأت في إنشاء عبارة while

٥. لتحديد التعبير المنطقي الذي يتم تقييمه بواسطة العبارة while قم بتحديث العبارة while كما يلي:

```
while (menuSelection != "exit");
```

يضمن هذا التعبير تكرار عبارة do حتى يختار المستخدم "إنهاء" التطبيق. سيعرض كل تكرار القائمة، ويقرأ تحديد قائمة المستخدم.

٦. لاحظ أن المسافة البادئة لسطر التعليمات البرمجية ليست ما تتوقع رؤيته داخل كتلة do

٧. لجعل Visual Studio Code يصلح المسافة البادئة لخطوط التعليمات البرمجية، انقر بزر الماوس الأيمن داخل محرر التعليمات البرمجية، ثم حدد **تنسيق المستند Format Document**

بالإضافة إلى تنظيم أسطر التعليمات البرمجية بشكل صحيح، سيقوم الأمر **تنسيق المستند Format Document** بتطبيق قواعد تنسيق التعليمات الأخرى، ومع ذلك، فإنه لا يصلح أخطاء بناء الجملة في تعليماتك البرمجية، ويمكن أن يكون له نتائج غير متوقعة، إذا كانت التعليمات تضم كتل برمجية غير مكتملة، أو مشوهة، يعد الأمر تنسيق المستند مفيداً طالماً كنت حذراً.

اختصارات لوحة المفاتيح لاستدعاء هذا الأمر هي:

- On Windows Shift + Alt + F
- On Mac Shift + Option + F
- On Linux Ctrl + Shift + I

يمكن أيضاً العثور على هذا الأمر باستخدام لوحة الأوامر Command Palette للعثور على الأمر تنسيق المستند من لوحة الأوامر:

- في القائمة عرض **View** حدد لوحة الأوامر Command Palette ثم أدخل Format D تنسيق D في المطالبة.
- ستظهر قائمة الأوامر التي تمت تصفيتها تنسيق المستند Format Document كأمر قابل للتحديد

٨. في قائمة **File** حدد حفظ **Save**

٩. افتح موجة الأوامر المتكامل Integrated Terminal panel

يمكنك فتح Integrated Terminal panel عن طريق النقر بزر الماوس الأيمن فوق المجلد Starter في EXPLORER ثم تحديد Open in Integrated Terminal

١٠. في موجه الأوامر Terminal أدخل dotnet build الأمر لإنشاء التعليمات البرمجية المحدثة.

يجب أن تشاهد رسالة تبلغ عن 0 تحذير و0 خطأ

١١. قم بإصلاح أي أخطاء أو تحذيرات إنشاء تم الإبلاغ عنها قبل المتابعة.

١٢. في موجه الأوامر Terminal أدخل الأمر dotnet run لتشغيل التعليمات البرمجية المحدثة.

سيبدأ تشغيل التطبيق.

١٣. تحقق من عرض خيارات القائمة الرئيسية في لوحة Terminal كما هو متوقع.

يجب أن تشاهد ما يلي معروضا في لوحة TERMINAL

```
Welcome to the Contoso PetFriends app. Your
main menu options are:
1. List all of our current pet information
2. Add a new animal friend to the ourAnimals
array
3. Ensure animal ages and physical descriptions
are complete
4. Ensure animal nicknames and personality
descriptions are complete
5. Edit an animal's age
6. Edit an animal's personality description
7. Display all cats with a specified
characteristic
8. Display all dogs with a specified
characteristic

Enter your selection number (or type Exit to
exit the program)
```

١٤. في موجه أوامر Terminal command لتحديد خيار القائمة الأول أدخل 1

إذا تم إيقاف تطبيقك مؤقتًا مع وجود المؤشر على يمين الرقم "1" فاضغط على Enter لإدخال القيمة، تحتاج إلى الضغط على مفتاح Enter بعد كتابة الحرف 1

١٥. تحقق من أن التعليمات البرمجية تعكس تحديد القائمة الذي أدخلته.

يجب أن تشاهد الإخراج التالي معروضاً في Terminal

```
You selected menu option 1.
Press the Enter key to continue
```

إذا لم تشاهد هذه الرسالة، فتأكد من حفظ تعديلات التعليمات البرمجية قبل تشغيل الأمر dotnet build إذا لزم الأمر، احفظ التعليمات المعدلة ثم حاول تشغيل التطبيق مرة أخرى.

١٦. في موجه الأوامر Terminal اضغط على المفتاح Enter للمتابعة.

يؤدي الضغط على Enter إلى تمكين التعليمات البرمجية من المتابعة بعد الأسلوب Console.ReadLine() الموجود قبل تقييم التعبير while في نهاية التطبيق.

١٧. للتحقق من أن التعليمات تستمر في قبول تحديدات قوائم إضافية، حاول إدخال واحد أو أكثر من أرقام عناصر القائمة الأخرى.

إذا توقفت التعليمات البرمجية عن العمل بعد إدخال "1" فتأكد من حفظ التعديلات، إذا لزم الأمر، احفظ التعليمات المعدلة ثم حاول تشغيل التطبيق مرة أخرى.

قد تلاحظ أنه يمكنك إدخال أي نص أو قيمة رقمية، أو حتى مجموعة من الأحرف والأرقام، ستعالج هذه المشكلة عند تطوير التعليمات البرمجية التي تنظم إدخال المستخدم.

١٨. في موجه أوامر Terminal للتحقق من تقييم معايير الخروج للحلقة بشكل صحيح، أدخل إما **Exit** أو **exit**

عند إدخال إما "Exit" أو"exit" يجب تقييم التعبير while إلى false وإنهاء تنفيذ التعليمات البرمجية.

هام

إذا لم يتوقف تنفيذ التعليمات كما هو متوقع عند إدخال إنهاء، اضغط على **Ctrl-C** في الوحدة الطرفية المتكاملة the Integrated Terminal لفرض إيقاف تنفيذ التعليمات البرمجية.

١٩. أغلق قائمة Terminal

كتابة عبارة switch لتحديدات القائمة

في هذه المهمة، تقوم بكتابة التعليمات البرمجية لعبارة التبديل switch الذي يفرع تنفيذ التعليمات بناءً على القيمة المعينة لـ MenuSelection يمكنك إنشاء تسمية تبديل switch تتوافق مع رقم كل عنصر في القائمة. يضمن إنشاء تسميات تبديل switch منفصلة إدارة كل عنصر قائمة بشكل منفصل. تقوم عبارة التبديل بعمل جيد في تنفيذ المنطق المقصود لتطبيقك.

١. في محرر التعليمات البرمجية Visual Studio حدد موقع سطر التعليمات بعد الأسلوب Console.WriteLine() الذي يتم استخدامه لتكرار تحديد القائمة.

```
Console.WriteLine($"You selected menu option
{menuSelection}.");
```

٢. لبدء تطوير العبارة switch قم بتحديث التعليمات البرمجية، على النحو التالي:

```
//Console.WriteLine($"You selected menu option
{menuSelection}.");
//Console.WriteLine("Press the Enter key to
continue");

// pause code execution
```

```
//readResult = Console.ReadLine();

switch(menuSelection)
{

}
```

٣. لإضافة حالة أو نمط case الأول إلى العبارة switch قم بتعديل التعليمات كما يلي:

```
switch(menuSelection)
{
    case "1":
        break;

}
```

ربما لاحظت أن حالة case المستخدمة هنا تبدو مختلفة قليلاً، عن نمط الحالة المستخدمة عند إنشاء نموذج البيانات في وقت سابق داخل التطبيق (case 1: versus case "1":) التفسير بسيط، لقد كنت تتحقق سابقًا من قيم عددية، والآن تقوم بالتحقق من قيم نصية.

٤. لإنشاء فروع تحديد في بنية switch-case لكل رقم من أرقام تحديد القائمة المتبقية، قم بإضافة خيارات case إضافية كما يلي:

```
switch(menuSelection)
{
    case "1":
        break;

    case "2":
        break;

    case "3":
        break;
```

```
    case "4":
        break;

    case "5":
        break;

    case "6":
        break;

    case "7":
        break;

    case "8":
        break;

}
```

٥. لاحظ أنه يمكنك إضافة الحالة الاختيارية default إلى نهاية قائمة التحديد الخاصة بك كما يلي:

```
switch(menuSelection)
{
    // case selections 1-7 have been removed to
simplify this sample code

    case "8":
        break;

    default:
        break;
}
```

تمت إزالة تحديدات الحالة من ١ إلى ٧ لتبسيط نموذج التعليمات البرمجية هذا

٦. في قائم File حدد Save

٧. افتح لوحة الوحدة الطرفية المتكاملة Integrated Terminal panel ثم قم بتشغيل الأمر dotnet build لإنشاء البرنامج.

يجب أن تشاهد الرسالة التي تبلغ عن ٠ تحذير (تحذيرات) و٠ خطأ (أخطاء) 0 Warning(s) and 0 Error(s)

٨. قم بإصلاح أي أخطاء أو تحذيرات إنشاء تراها تم الإبلاغ عنها قبل المتابعة.

٩. أغلق لوحة Terminal

كتابة تعليمات العنصر النائب لكل حالة من عبارة التبديل switch

في هذه المهمة، يمكنك تحديث مسارات تنفيذ التعليمات البرمجية التي تم إنشاؤها بواسطة العبارة switch عند الانتهاء، سيوفر كل قسم switch ملاحظات للمستخدم، تقر باختيار القائمة الخاصة به، يعد تقديم الملاحظات التي تقر بإدخال المستخدم أمراً مهمًا، نظرًا لأنك لن تتمكن من إكمال التعليمات لجميع خيارات القائمة أثناء هذه الدرس، تتيح الملاحظات للمستخدم معرفة أن التعليمات تعرفت على إدخاله، حتى إذا كان التطبيق غير قادر على معالجة الطلب.

١. في محرر Visual Studio Code حدد موقع سطر التعليمات الذي يحتوي على تحديد `case "1":`

```
switch (menuSelection)
{
    case "1":
        break;
```

٢. لتوفير رسالة ملاحظات للمستخدم بين سطري التعليمات البرمجية `case "1": and break;` قم بتعديل التعليمات كما يلي:

```
switch (menuSelection)
{
    case "1":
        // List all of our current pet
information
```

```
        Console.WriteLine("this app feature is
coming soon - please check back to see
progress.");
        Console.WriteLine("Press the Enter key
to continue.");
        readResult = Console.ReadLine();
        break;
```

٣. أضف نفس رسالة الملاحظات للتحديد `case "2":` كما يلي:

```
switch (menuSelection)
{
    case "1":
        // List all of our current pet
information
        Console.WriteLine("this app feature is
coming soon - please check back to see
progress.");
        Console.WriteLine("Press the Enter key
to continue.");
        readResult = Console.ReadLine();
        break;

    case "2":
        // List all of our current pet
information
        Console.WriteLine("this app feature is
coming soon - please check back to see
progress.");
        Console.WriteLine("Press the Enter key
to continue.");
        readResult = Console.ReadLine();
        break;
```

٤. قم بتحديث تعليق الخط للتحديد case "2": ليعكس خيار القائمة الثانية كما يلي:

```
case "2":
    // Add a new animal friend to the
ourAnimals array
    Console.WriteLine("this app feature is
coming soon - please check back to see
progress.");
    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
    break;
```

٥. بالنسبة لتحديد من case "3": إلى case "8": قم بتحديث التعليمات بتعليق سطر يعكس نص عنصر القائمة المرتبط.

على سبيل المثال، أضف // Ensure animal ages and physical descriptions are complete تحت case "3":

يمكنك نسخ نص عنصر القائمة من أسطر التعليمات التي تعرض خيارات القائمة.

ملاحظة: يمكنك استخدام الأمر "تنسيق المستند" لتنظيف التنسيق، قد يؤدي ذلك إلى تسريع عملك، ومنحك المزيد من الوقت للتركيز على التعليمات التي تكتبها، بدلاً من الاهتمام بكيفية ظهورها.

٦. بالنسبة للتحديدين case "3": and case "4": أضف رسالة الملاحظات التالية وعبارة ReadLine() أسفل تعليق السطر:

```
Console.WriteLine("Challenge Project - please
check back soon to see progress.");
Console.WriteLine("Press the Enter key to
continue.");
readResult = Console.ReadLine();
```

٧. بالنسبة للتحديدات case المتبقية قم بتحديث التعليمات برسالة الملاحظات التالية و ReadLine()

```
Console.WriteLine("UNDER CONSTRUCTION - please
check back next month to see progress.");
Console.WriteLine("Press the Enter key to
continue.");
readResult = Console.ReadLine();
```

ملاحظة

إذا أضفت الحالة الاختيارية default إلى قائمة التحديد، فلا تقم بإضافة رسالة الملاحظات إلى هذه الحالة.

٨. في قائمة **File** حدد **Save**

تذكر أن Visual Studio Code سينسق التعليمات، إذا نقرت بزر الماوس الأيمن داخل المحرر ثم حدد تنسيق المستند Format Document من قائمة السياق.

٩. افتح المحطة الطرفية المتكاملة the Integrated Terminal ثم قم بتشغيل الأمر `dotnet build` لإنشاء البرنامج.

١٠. قم بإصلاح أي أخطاء أو تحذيرات إنشاء تراها تم الإبلاغ عنها قبل المتابعة.

تحقق من عملك

في هذه المهمة، يمكنك تشغيل التطبيق، والتحقق من أن العبارة switch تفريع التعليمات البرمجية كما هو مطلوب، يمكنك أيضاً التحقق من عرض رسائل الملاحظات المتوقعة لكل تحديد قائمة، سيتم أيضاً إعادة اختبار معايير الخروج لحلقة القائمة الرئيسية.

١. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

٢. في موجه الأوامر Terminal، أدخل **dotnet run**

٣. في موجه الأوامر Terminal، لتحديد خيار القائمة الأولى، أدخل **1**

٤. تحقق من عرض الرسالة المتوقعة "coming soon"

٥. كرر الخطوة السابقة، وأدخل كل رقم من أرقام خيارات القائمة الأخرى للتحقق من عرض الرسالة المتوقعة في كل حالة.

٦. في موجه الأوامر Terminal، أدخل إما **Exit** أو **exit** للتحقق من أن معايير الخروج لا تزال قيد التقييم بشكل صحيح.

هام

إذا لم يتوقف تنفيذ التعليمات كما هو متوقع، يمكنك الضغط على **Ctrl-C** في الوحدة الطرفية المتكاملة the Integrated Terminal لفرض التنفيذ للإيقاف.

٧. أغلق قائمة Terminal

٤ تمرين ـ كتابة التعليمات لعرض جميع بيانات مصفوفة ourAnimals

في هذا التمرين، ستقوم بكتابة عبارة التكرار المتداخلة، وكتلة تعليمات التحديد المستخدمة لعرض معلومات مصفوفة OurAnimals

المهام التفصيلية التي تكملها خلال هذا التمرين هي:

١. **الحلقة الخارجية:** أنشئ الحلقة الخارجية التي تتكرر عبر الحيوانات الموجودة في مصفوفة ourAnimals array

٢. **التحقق من البيانات:** كتابة التعليمات التي تتحقق من بيانات الحيوانات الأليفة الموجودة، وتعرض معرف الحيوانات إذا كانت بيانات الحيوان موجودة.

٣. **الحلقة الداخلية:** إنشاء حلقة داخلية تعرض جميع خصائص الحيوانات الأليفة مع البيانات المعينة.

٤. **اختبار التحقق:** قم بإجراء اختبارات التحقق للتعليمات البرمجية التي تطورها في هذا التمرين.

هام: يجب إكمال التمرين السابق، في هذه الوحدة، قبل بدء هذا التمرين.

إنشاء حلقة تكرار من خلال مصفوفة ourAnimals

في هذه المهمة، يمكنك إنشاء حلقة for الخارجية التي يتم استخدامها للتكرار عبر الحيوانات الموجودة في مصفوفة ourAnimals يمكنك فحص العلاقة بين أبعاد المصفوفة ومعلمات حلقة for عليك أيضًا مراعاة الاختلافات بين استخدام عبارات for وعبارات foreach عند العمل مع مصفوفات متعددة الأبعاد.

١. تأكد من أن Visual Studio Code مفتوح، وأن ملف Program.cs الخاص بك مرئي في المحرر.

٢. داخل مقطع التعليمات البرمجية لعبارة switch لتحديد (menuSelection) حدد موقع سطور التعليمات التالية:

```
case "1":
    // List all of our current pet information
    Console.WriteLine("this app feature is
coming soon - please check back to see
progress.");
    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
    break;
```

٣. احذف العبارة Console.WriteLine() المستخدمة لعرض الرسالة "coming soon" واترك سطر فارغ أسفل سطر تعليق التعليمات `// List all of our current pet information`

لن تكون هناك حاجة إلى رسالة الملاحظات، لأنك ستعرض محتويات مصفوفة ourAnimals أثناء تطوير التعليمات المطابقة لتحديدات القوائم الأخرى، سيتم تحديث رسائل الملاحظات هذه أيضاً.

اترك الرسالة التي تقول `Press the Enter key to continue`

٤. بدءا من السطر الفارغ التي قمت بإنشائه، ابدأ عبارة for كما يلي:

```
for ()
{
}
```

٥. خذ دقيقة للنظر فيما تحتاج إلى تحقيقه مع عبارة for

تذكر أن عبارة for تحتوي على ثلاثة أجزاء تتحكم في كيفية تنفيذ التكرارات: مُهيئ for؛ الشرط، والمكرر the for initializer; the for condition, the for iterator تعتمد القيم المخصصة للمُهيئ والشرط والمكرر على ما تحتاج إلى تحقيقه باستخدام عبارة for

في هذه الحالة، يتم استخدام for للتكرار من خلال مصفوفة ourAnimals

أنت تعلم أن المصفوفات مفهرسة بصفر، مما يعني مثلاً: أن المصفوفة التي تحتوي على n من العناصر تتم فهرستها من 0 إلى n - 1 لمطابقة أبعاد مصفوفة في هذه الحالة، تريد أن تبدأ الحلقة for من 0 بزيادة بمقدار 1 وتنتهي عند maxPets - 1

تم تعريف مصفوفة ourAnimals على النحو التالي `string[,]` `ourAnimals = new string[maxPets, 6];` أنت تعلم أن القيمة المخصصة لـ maxPets هي 8 في هذا الإعلان، يحدد maxPets عدد العناصر في البعد الأول للمصفوفة، وليس رقم الفهرس الصفري الذي تستخدمه للإشارة إلى العناصر في المصفوفة، ولذلك، على الرغم من أن maxPets = 8 فإن أرقام فهرس المصفوفة تتراوح من 0 إلى 8

٦. لتحديد قيمة التحكم في تكرار for قم بتعديل التعليمات كما يلي:

```
for (int i = 0; i < maxPets; i++)
{

}
```

كما ترون، ضبط مُهيئ التهيئة على int i = 0; يتماشى مع فهرس المصفوفة الصفري، وبالمثل، تعيين الشرط على i < maxPets; لمحاذاة البعد الأول للمصفوفة، أخيرًا، سيؤدي تعيين المكرر `i++` iterator إلى زيادة قيمة التحكم في الحلقة بمقدار 1 لكل تكرار.

٧. خذ دقيقة من وقتك للتفكير في الاختيار بين عبارة for و foreach عند التكرار عبر مصفوفة ourAnimals

الهدف هو تكرار كل حيوان في مصفوفة ourAnimals واحدًا تلو الآخر، فلماذا لا تستخدم حلقة foreach بعد كل شيء، أنت تعلم أن عبارة foreach مصممة للحالات التي تريد فيها التكرار خلال كل عنصر في صفيف من العناصر.

السبب وراء عدم استخدام حلقة foreach في هذه الحالة هو أن مصفوفة ourAnimals عبارة عن مصفوفة متعددة الأبعاد، نظرًا لأنها عبارة عن

مصفوفة نصية متعددة الأبعاد، فإن كل عنصر موجود فيها عبارة عن عنصر منفصل من نوع string إذا استخدمت حلقة foreach للتكرار عبر ourAnimals فستتعرف حلقة foreach على كل سلسلة كعنصر منفصل في قائمة مكونة من ٤٨ عنصر سلسلة (٨ × ٦ = ٤٨) لن تقوم عبارة foreach بمعالجة بعدي المصفوفة بشكل منفصل، بمعنى آخر، لن تتعرف حلقة foreach على ٨ صفوف من عناصر السلسلة، حيث يحتوي كل صف على عمود مكون من ٦ عناصر، نظرًا لأنك تريد العمل مع حيوان واحد في كل مرة، ومعالجة جميع خصائص الحيوانات الستة أثناء تكرار واحد، فإن عبارة foreach ليست الخيار الصحيح.

ومع ذلك، إذا كانت مصفوفة ourAnimals عبارة عن مصفوفة متعرجة (jagged) تم تكوينها كمصفوفة من النوع النصي، فيمكنك استخدام عبارة foreach في هذه الحالة، يمكنك إنشاء foreach للحلقة الخارجية وforeach الثانية للحلقة الداخلية.

تتكرر **الحلقة الخارجية** من خلال عناصر string array الموجودة في المصفوفة المتعرجة، وهي عبارة عن الصفوف **rows** في المصفوفة ثنائية الأبعاد، تتكرر **الحلقة الداخلية** من خلال عناصر "string" الموجودة في مصفوفة string arrays وهي عبارة عن الأعمدة **columns** في المصفوفة ثنائية الأبعاد.

يوضح نموذج التعليمات البرمجية التالي نهج المصفوفة المتعرجة.

```
string[][] jaggedArray = new string[][]
{
    new string[] { "one1", "two1", "three1",
"four1", "five1", "six1" },
    new string[] { "one2", "two2", "three2",
"four2", "five2", "six2" },
    new string[] { "one3", "two3", "three3",
"four3", "five3", "six3" },
    new string[] { "one4", "two4", "three4",
"four4", "five4", "six4" },
```

```
    new string[] { "one5", "two5", "three5",
"four5", "five5", "six5" },
    new string[] { "one6", "two6", "three6",
"four6", "five6", "six6" },
    new string[] { "one7", "two7", "three7",
"four7", "five7", "six7" },
    new string[] { "one8", "two8", "three8",
"four8", "five8", "six8" }
};

foreach (string[] array in jaggedArray)
{
    foreach (string value in array)
    {
        Console.WriteLine(value);
    }
    Console.WriteLine();
}
```

بالنسبة لتطبيق Contoso Pets الأسهل على الراجح استخدام مصفوفة نصية string متعدد الأبعاد، والحلقات المتداخلة for بدلاً من مصفوفة متعرجة، والحلقات المتداخلة foreach الآن بعد أن رأيت كيف يعمل كل خيار، يمكنك تحديد اختيارك الخاص في مشاريع الترميز المستقبلية.

٨. في قائمة File حدد Save

٩. افتح لوحة المحطة الطرفية المتكاملة، وأدخل أمر إنشاء البرنامج.

١٠. يجب إصلاح أي أخطاء في الإنشاء أو تحذيرات تراها قبل المتابعة.

تذكر أن رسالة الخطأ والتحذير الخاصة بالبناء تخبرك عن المشكلة، وأين يمكنك العثور عليها، عند حل المشكلات، من الأفضل البدء بالمشكلات من أعلى التعليمات البرمجية إلى أسفل.

١١. أغلق Terminal

التحقق من وجود بيانات الحيوانات الأليفة الموجودة وعرض النتيجة

في هذه المهمة، يمكنك استخدام عبارة if للعثور على كل حيوان أليف في ourAnimals الذي تم حفظ بيانات خصائص الحيوانات الأليفة داخلها، عند العثور على حيوان مع بيانات معينة، يمكنك عرض petID عندما لا يتم تعيين أي بيانات، لا يتم عرض أي شيء، يمكنك تشغيل التعليمات للتحقق من أن عبارات for and if تعمل بشكل صحيح.

١. إنشاء سطر فارغ داخل الكتلة البرمجية لعبارة for كما يلي:

```
for (int i = 0; i < maxPets; i++)
{

}
```

٢. لإنشاء عبارة if التي تتحقق من بيانات معرف ID data الحيوان، قم بتحديث التعليمات كما يلي:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
    }
}
```

٣. خذ دقيقة للنظر فيما تقوم به العبارة if بتقييمه ولماذا

أولا، ضع في اعتبارك الجانب الأيسر من التعبير ourAnimals[i, 0] لاحظ أن متغير التحكم في التكرار i يستخدم لتحديد الحيوان الذي يتم فحصه، كما قد تتذكر فأن الرقم 0 في [i, 0] يتوافق مع الخاصية petID نظراً لأن البعد الأول من المصفوفة يتوافق مع عدد الحيوان، فإن هذا الجانب من التعبير يضمن أن التعليمات البرمجية تتحقق من القيمة المعينة petID لكل حيوان في المصفوفة.

ثانيا، ضع في اعتبارك اختيار عامل المقارنة، لاحظ أنه يتم استخدام not-equal operator عامل التشغيل =! يتم تقييم التعبير true كلما كانت القيمة

المعينة إلى petID `ourAnimals[i, 0]` لا تساوي القيمة المدرجة على الجانب الأيسر من المعادلة.

ثالثا، ضع في اعتبارك القيمة الموجودة على الجانب الأيمن من المعادلة، يتم استخدام قيمة سلسلة ثابتة من `"ID #: "` هذه هي القيمة الافتراضية المعينة عند petID إنشاء نموذج البيانات، عند تعيين الخصائص إلى حيوان petID يتم تحديث القيمة، ولن تكون مساوية للقيمة الافتراضية.

يخبرك هذا أنه سيتم تنفيذ كتلة التعليمات لعبارة if عندما يكون للحيوان الحالي خصائص محددة.

ملاحظة

هذا مثال جيد على متى يجب استخدام العامل `!=` لا تهتم بالقيمة التي تم تعيينها إلى `petID` طالما أنها ليست القيمة الافتراضية

لإنشاء أسلوب Console.WriteLine() يعرض petID داخل كتلة التعليمات لعبارة if قم بتحديث التعليمات كما يلي:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
    }
}
```

احظ أنه يمكن استخدام عنصر مصفوفة كوسيطة عند استدعاء الأسلوب WriteLine()

٥. في قائمة **File** حدد **Save**

٦. افتح the Integrated Terminal وأدخل أمر بناء إنشاء برنامجك.

٧. يجب إصلاح أي أخطاء في الإنشاء أو تحذيرات تراها قبل المتابعة.

إذا كانت تعليماتك البرمجية تنشئ خطأ وقت التشغيل، فقم بإصلاح الأخطاء وحفظ التحديثات وإعادة تشغيل التطبيق.

٨. في موجه الأوامر Terminal للتحقق من أن منطق التعليمات البرمجية الجديد يعمل كما هو متوقع، أدخل **1**

يجب أن تشاهد قيم `petID` التالية معروضة:

```
ID #: d1
ID #: d2
ID #: c3
ID #: c4
Press the Enter key to continue
```

تتوافق هذه المعرفات مع الحيوانات الأليفة التي قامت بتعيين البيانات.

إذا كانت تعليماتك البرمجية تعرض إخراجاً مختلفاً، عند تحديد خيار القائمة 1 فراجع التعليمات البرمجية وتعديلاتها، تذكر أنك تحتاج إلى حفظ ملف Program.cs بعد إجراء التحديثات.

١٠. قم بإنهاء التطبيق، ثم أغلق Terminal

عرض جميع خصائص الحيوانات الأليفة مع البيانات المعينة

في هذه المهمة، يمكنك إنشاء تكرار `for` داخل كتلة `if` للبيانات المستخدمة لعرض جميع خصائص الحيوان الأليف الحالي.

١. في المحرر حدد موقع أسطر التعليمات التالية في ملف Program.cs

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
    }
}
```

٢. لإنشاء تكرار for الذي سيتكرر من خلال خصائص كل حيوان أليف، قم بتعديل التعليمات على النحو التالي:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
        for (int j = 0; j < 6; j++)
        {
        }
    }
}
```

لاحظ أن لديك الآن حلقة for ثانية متداخلة داخل كتلة الحلقة الأولى for كما تعلمون، التكرار الخارجي يتكرر عبر الحيوانات في مصفوفة ourAnimals الهدف هو تكرار الحلقة الداخلية من خلال خصائص كل حيوان، وبما أن البيانات الحيوانية مخزنة في مصفوفة متعدد الأبعاد، سيكون من السهل الوصول إلى الخصائص الحيوانية.

٣. خذ دقيقة لمراجعة العبارة for التي أدخلتها.

لاحظ أن متغير التحكم في التكرار الحلقي يسمى j عند تداخل حلقات for يكون أحد النهج التقليدية هو استخدام i في الحلقة الخارجية و j في الحلقة الداخلية.

اتباع اصطلاحات مثل هذه، يسهل على الآخرين قراءة تعليماتك البرمجية.

نظرا لأنه يتم تخزين ست خصائص لكل حيوان، المُهيئ for initializer هو int j = 0; والشرط for condition هو j < 6; هذا المزيج من المُهيئ والشرط يطابق نطاق فهرس المصفوفة الذي تحتاجه 0 - 5

٤. لعرض كل خاصية لحيوان أليف على سطر منفصل، قم بتعديل التعليمات البرمجية كما يلي:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine(ourAnimals[i, 0]);
        for (int j = 0; j < 6; j++)
        {
            Console.WriteLine(ourAnimals[i,
j]);
        }
    }
}
```

٥. خذ دقيقة للنظر في البنية المتداخلة التي قمت بإنشائها، والإخراج المعروض الذي ستنتجه تعليماتك البرمجية.

لاحظ أن القيمة المكتوبة إلى وحدة التحكم `ourAnimals[i, j]` تستخدم متغيرات التحكم، داخل حلقات التكرار، مع كل من حلقات `for` الخارجية والداخلية.

تعرف كل عنصر من العناصر التالية:

- يتوافق البعد الأول من مصفوفة ourAnimals مع الحيوانات المختلفة.
- يتوافق البعد الثاني من مصفوفة ourAnimals مع خصائص كل حيوان.
- تمنع العبارة if تشغيل الحلقة الداخلية عندما لا تكون هناك بيانات حيوانات معينة إلى الحيوان الحالي.
- تكمل الحلقة الداخلية جميع تكراراتها لكل تكرار للحلقة الخارجية.

لذلك، أنت تعرف أن خصائص كل حيوان سيتم عرضها على النحو المنشود.

٦. لاستبدال رسالة `petID` بأسلوب `WriteLine()` فارغ قم بتعديل تعليماتك البرمجية كما يلي:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine();
        for (int j = 0; j < 6; j++)
        {
            Console.WriteLine(ourAnimals[i,
j]);
        }
    }
}
```

يسهل هذا التحديث النهائي رؤية الفصل بين الحيوانات، عند عرض الإخراج إلى وحدة التحكم.

٧. في قائمة **File** حدد **Save**

٨. افتح Terminal وأكتب أمر إنشاء البرنامج.

٩. يجب إصلاح أي أخطاء في الإنشاء أو تحذيرات تراها قبل المتابعة.

تحقق من عملك

في هذه المهمة، يمكنك تشغيل التطبيق من الوحدة الطرفية المتكاملة، والتحقق من أن تركيبتك المتداخلة من العبارات `for` and `if` تنتج النتيجة المتوقعة.

١. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة Integrated Terminal panel

٢. في موجه الأوامر Terminal أدخل **dotnet run**

٣. في موجه الأوامر Terminal أدخل **1**

٤. تحقق من عرض بيانات الحيوانات الأليفة الأربعة التي تم تعيين بيانات لها.

```
ID #: d1
Species: dog
```

```
Age: 2
Nickname: lola
Physical description: medium sized cream colored
female golden retriever weighing about 65 pounds.
housebroken.
Personality: loves to have her belly rubbed and
likes to chase her tail. gives lots of kisses.

ID #: d2
Species: dog
Age: 9
Nickname: loki
Physical description: large reddish-brown male
golden retriever weighing about 85 pounds.
housebroken.
Personality: loves to have his ears rubbed when he
greets you at the door, or at any time! loves to
lean-in and give doggy hugs.

ID #: c3
Species: cat
Age: 1
Nickname: Puss
Physical description: small white female weighing
about 8 pounds. litter box trained.
Personality: friendly

ID #: c4
Species: cat
Age: ?
Nickname:
Physical description:
Personality:
Press the Enter key to continue.
```

ملاحظة

إذا لم تشاهد النتائج المتوقعة معروضة، فتأكد من حفظ ملف Program.cs المحدث. إذا كنت لا ترى النتائج المتوقعة ولم تتمكن من تحديد المشكلة، فيمكنك فحص تعليمات Program.cs في المجلد النهائي، تم تضمين المجلد النهائي كجزء من التنزيل الذي أكملته أثناء الإعداد، نوصي بقضاء بعض الوقت في محاولة تحديد وإصلاح مشكلة بناء الجملة، والمنطق في التعليمات البرمجية، قبل التحقق من ملف Program.cs في المجلد النهائي.

٥ تمرين - إنشاء واختبار حلقة لإدخال بيانات الحيوانات الجديدة

في هذا التمرين، يمكنك تطوير التعليمات التي تتحكم في إدخال البيانات الجديدة لمصفوفة ourAnimals وحساب القيم الأولية لمتغيرات التحكم في حلقات التكرار، وإنشاء الحلقة التي تجمع البيانات الحيوانات المحددة للمستخدم، المهام التفصيلية التي تكملها أثناء هذا التمرين هي:

١. **حساب عدد الحيوانات الأليفة:** Calculate petCount كتابة التعليمات التي تحسب عدد الحيوانات الأليفة في مصفوفة ourAnimals التي خصصت البيانات

٢. **الرسائل الشرطية:** Conditional messages اكتب التعليمات لعرض إخراج الرسالة عندما يكون petCount أقل من maxPets

٣. **الحلقة الخارجية:** Outer loop إنشاء عبارة تكرار سيتم استخدامها لإدخال بيانات مصفوفة حيواناتنا الجديدة ourAnimals

٤. **معايير الإنهاء:** Exit criteria اكتب التعليمات التي تقيم شرط الخروج من حلقة "إدخال البيانات الجديدة لمصفوفة OurAnimals"

٥. **اختبار التحقق:** قم بإجراء اختبارات التحقق للتعليمات البرمجية التي تطورها في هذا التمرين

هام

يجب إكمال التمارين السابقة في هذه الوحدة، قبل بدء هذا التمرين

حساب عدد الحيوانات الأليفة في مصفوفة ourAnimals

في هذه المهمة، يمكنك إنشاء معايير الخروج لحلقة إدخال البيانات، وإنشاء حلقة for يمكن استخدامها لحساب عدد الحيوانات الأليفة في ourAnimals التي تم تعيين البيانات إليها.

١. تأكد من أن ملف Program.cs مفتوح في محرر Visual Studio Code

٢. حدد موقع عبارة `switch(menuSelection)` ثم ابحث عن سطر التعليمات `case "2":`

٣. حدد موقع العبارة `Console.WriteLine()` التي تعرض الرسالة `"coming soon"` ثم استبدلها بسطر فارغ.

٤. في السطر الفارغ الذي قمت بإنشائه، لتعريف متغيرات `petCount`, `anotherPet` أدخل التعليمة البرمجية التالية:

```
string anotherPet = "y";
int petCount = 0;
```

يتحكم هذان المتغيران في تكرار `while` المستخدم لإدخال بيانات الحيوانات الجديدة. يمكنك تهيئة كلا المتغيرين كجزء من الإعلان.

- تتم تهيئة `anotherPet` بقيمة y قبل بداية الحلقة `while` سوف يتلقى قيمة من المستخدم، إما y أو n داخل الحلقة `while`
- يمثل `petCount` عدد الحيوانات ذات الخصائص الحيوانية المخصصة، سيتم تعيين قيمة محسوبة له خارج حلقة `while` وستتم زيادتها بمقدار 1 داخل حلقة `while` في كل مرة تتم إضافة حيوان جديد إلى مصفوفة `ourAnimals`

هام

يجب أن يكون نطاق متغيراتك دائمًا ضيقًا قدر الإمكان، في تطبيق Contoso Pets يمكنك تحديد نطاق petCount على مستوى التطبيق بدلاً من تحديد نطاق `case "2":` ادخل الكتلة البرمجية، سيمكّنك النطاق الأكبر من الوصول إلى petCount من أي مكان في التطبيق، إذا تم تحديد نطاق petCount على مستوى التطبيق، فيمكنك تعيين قيمة له عند إنشاء نموذج البيانات، وإدارة قيمتها برمجيًا خلال باقي التطبيق، على سبيل المثال، عند العثور على منزل لحيوان، وإزالة الحيوان من مصفوفة ourAnimals يمكنك تقليل petCount عدد الحيوانات الأليفة بمقدار 1 السؤال هو: على أي مستوى يجب عليك تحديد نطاق المتغير، عندما لا تكون متأكدًا، مما إذا

كان سيتم استخدامه في أجزاء أخرى من تطبيقك؟ في هذه الحالة، من المغري تحديد نطاق petCount على مستوى التطبيق على الرغم من أنك لا تستخدمه في أي مكان آخر، تحديد نطاق petCount على مستوى التطبيق يضمن توفره إذا قررت استخدامه، ربما يمكنك تحديد نطاق المتغيرات الأخرى على مستوى التطبيق أيضًا، بهذه الطريقة، تكون متغيراتك دائمًا في النطاق، ويمكن الوصول إليها، فلماذا لا يتم تحديد نطاق المتغيرات على مستوى التطبيق؟ عندما تعتقد أنه يمكن استخدامها لاحقًا في التطبيق.

يمكن أن يؤدي تحديد نطاق المتغيرات على مستوى أعلى من اللازم إلى حدوث مشكلات، مثل **زيادة متطلبات الموارد لتطبيقك، وربما يعرض تطبيقك لمخاطر أمنية غير ضرورية**، مع نمو تطبيقاتك بشكل أكبر وأكثر تعقيدًا، فإنها تتطلب المزيد من الموارد، تقوم الهواتف وأجهزة الكمبيوتر بتخصيص الذاكرة لهذه الموارد عندما تكون في نطاقها، عندما تعمل تطبيقاتك في الواقع الفعلي real-world يصبح الوصول إليها أكثر سهولة، فغالبًا يمكن الوصول إلى التطبيقات من السحابة أو من التطبيقات الأخرى، ومما يزيد من تفاقم هذه المشكلات، غالبًا ما يتم ترك التطبيقات قيد التشغيل عندما لا يتم استخدامها، **من المهم إبقاء متطلبات موارد التطبيق تحت السيطرة، وإبقاء البصمة الأمنية صغيرة قدر الإمكان**.

على الرغم من أن أنظمة التشغيل الحالية تقوم بعمل رائع في إدارة الموارد وتأمين التطبيقات، إلا أنه لا يزال من أفضل الممارسات الحفاظ على نطاق متغيراتك في المستوى الذي تكون في حاجة ضرورية إليها، في تطبيق Contoso Pets إذا قررت استخدام petCount على نطاق أوسع داخل التطبيق، فيمكنك تعديل التعليمات لنطاق petCount على مستوى أعلى، تذكر أن تبقي متغيراتك في نطاق أضيق قدر الإمكان، وقم بزيادة نطاقها فقط عندما يصبح ذلك ضروريًا.

٥. في سطر التعليمات أسفل إعلان المتغيرات، لإنشاء حلقة تتكرر من خلال الحيوانات في مصفوفة ourAnimals أدخل التعليمات التالية:

```
for (int i = 0; i < maxPets; i++)
{
}
```

يجب أن تبدو هذه التعليمة مألوفة، ستستخدم حلقة for في كل مرة تقوم فيها بالتكرار عبر مصفوفة ourAnimals

٦. داخل الكتلة البرمجية for للتحقق مما إذا تم تعيين بيانات خصائص الحيوانات إلى حيوان، أدخل التعليمات التالية:

```
if (ourAnimals[i, 0] != "ID #: ")
{
}
```

مرة أخرى، يجب أن تبدو هذه التعليمة مألوفة، ستستخدم العبارة if في كل مرة تتحقق فيها مما إذا تم تعيين خصائص الحيوانات الأليفة أم لا.

٧. داخل الكتلة البرمجية if قمت بإنشائها، لزيادة petCount بمقدار 1 أدخل التعليمات التالية:

```
petCount += 1;
```

٨. خذ دقيقة لفحص حلقة for المكتملة.

يجب أن تبدو الحلقة for المكتملة مثل التعليمات التالية:

```
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        petCount += 1;
    }
}
```

سيتم تكرار هذه التعليمات من خلال مصفوفة ourAnimals للتحقق من البيانات المخصصة، عندما يجد حيوانًا تم تعيين بيانات له، فإنه يزيد قيمة petCounter

٩. في قائمة **File** حدد **Save**

١٠ افتح المحطة الطرفية المتكاملة Integrated Terminal panel وأدخل أمر إنشاء البرنامج.

لفتح الوحدة الطرفية المتكاملة من قائمة EXPLORER انقر بزر الماوس الأيمن فوق **Starter** ثم حدد **Open in Integrated Terminal** يمكنك أيضا استخدام **القائمة View** أو **Terminal** menu

لإنشاء البرنامج أدخل build dotnet في موجه الأوامر Terminal

ملاحظة

في الوقت الحالي، يمكنك تجاهل رسالة التحذير حول تعيين anotherPet لم يتم استخدامها حاليا، ستضيف التعليمات التي تستخدم anotherPet لاحقا في هذا التمرين.

إذا كان لديك أي أخطاء في البنية، فتذكر أن خطأ الإنشاء ورسائل التحذير تخبرك ما هي المشكلة، ومكان العثور عليها، إذا قمت بتعديل التعليمات البرمجية، فتذكر حفظ التغييرات قبل إعادة البناء.

عرض إخراج الرسالة عندما يكون petCount أقل من maxPets

في هذه المهمة، يمكنك التحقق لمعرفة ما إذا كان petCount أقل من maxPets وإذا كان كذلك، فإنك تعرض رسالة للمستخدم.

في السطر الفارغ أسفل حلقة for التي قمت بإنشائها، لمعرفة ما إذا كان petCount أقل من maxPets أدخل التعليمات التالية:

```
if (petCount < maxPets)
{

}
```

داخل كتلة العبارة if لعرض رسالة للمستخدم، أدخل التعليمات التالية:

```
Console.WriteLine($"We currently have
{petCount} pets that need homes. We can manage
{(maxPets - petCount)} more.");
```

مستخدمو التطبيق على وشك إدخال خصائص الحيوانات الأليفة، توفر هذه الرسالة سياقاً مهماً.

٣. خذ دقيقة لمراجعة `case "2":` فرع التعليمات لعبارة `switch`

عند هذه النقطة، يجب أن تبدو `case "2":` مثل التالي:

```
case "2":
    // Add a new animal friend to the
ourAnimals array
    string anotherPet = "y";
    int petCount = 0;
    for (int i = 0; i < maxPets; i++)
    {
        if (ourAnimals[i, 0] != "ID #: ")
        {
                petCount += 1;
        }

    }

    if (petCount < maxPets)
    {
        Console.WriteLine($"We currently have
{petCount} pets that need homes. We can manage
{(maxPets - petCount)} more.");
    }

    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
    break;
```

٤. في قائمة **File** حدد **Save**

٥. افتح لوحة المحطة الطرفية المتكاملة وأدخل أمر إنشاء البرنامج.

٦. قم بإصلاح أي أخطاء

مرة أخرى، يمكنك تجاهل رسالة التحذير حول تعيين anotherPet في المهمة التالية، ستبدأ في إنشاء الحلقة while المستخدمة لإدخال البيانات لحيوان أليف واحد أو أكثر، سيستخدم anotherPet التعبير الذي تقوم بإنشائه للحلقة while وستزول رسالة التحذير هذه.

رسائل التحذير هي أشياء يجب أن تقلق بشأنها، ولكنها لن تمنعك من تشغيل برنامجك.

٧. في موجه الأوامر Terminal أدخل الأمر dotnet run لتشغيل البرنامج.

طالما أن التعليمات البرمجية لا تنشئ خطأ في وقت التشغيل، يجب الآن عرض القائمة الرئيسية للتطبيق في لوحة Terminal

٨. في موجه الأوامر Terminal أدخل **2**

تتوافق هذه القيمة مع فرع التعليمات `case "2":`

٩. تحقق من عرض الرسالة التالية.

```
We currently have 4 pets that need homes. We
can manage 4 more.
Press the Enter key to continue.
```

إذا لم تشاهد الرسالة المتوقعة معروضة، فراجع التعليمات البرمجية لتحديد المشكلة وإصلاحها، احفظ التغييرات، وشغله مرة أخرى، تأكد من عمل التعليمات المتوقع قبل المتابعة.

١٠. في موجه الأوامر Terminal اضغط على مفتاح الإدخال Enter لمتابعة تشغيل التطبيق.

١١. قم بإنهاء التطبيق، ثم أغلق Terminal

إنشاء عبارة تكرار سيتم استخدامها لإدخال البيانات الجديدة لمصفوفة ourAnimals

في هذه المهمة، يمكنك إنشاء while تستمر في التكرار طالما anotherPet يساوي y وطالما petCount أقل من maxPets

١. في المحرر، قم بإنشاء سطر فارغ أسفل كتلة التعليمات البرمجية للعبارة

```
if (petCount < maxPets)
```

٢. لبدء عملية إنشاء التكرار الجديد while أدخل التعليمات التالية.

```
while (anotherPet == "y" && petCount < maxPets)
{

}
```

٣. في قائمة **File** حدد **Save**

٤. افتح المحطة الطرفية المتكاملة وأدخل أمر إنشاء البرنامج.

٥. لاحظ أنك لم تعد تتلقى رسالة التحذير حول anotherPet لعدم استخدامها.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة

تحقق من حالة الخروج لحلقة الحيوانات الأليفة الجديدة

في هذه المهمة، يمكنك تحديث الكتلة البرمجية `while (anotherPet == "y" && petCount < maxPets)` تزيد التعليمات الجديد petCount ثم يتحقق مما إذا كان petCount أقل من maxPets إذا كان petCount أقل من maxPets فإنك تسأل المستخدم عما إذا كان يريد إدخال معلومات لحيوان أليف آخر، وتأكد من أن الاستجابة هي y أو n بعد الكتلة البرمجية `while (anotherPet == "y" && petCount < maxPets)` يمكنك التحقق من قيمة petCount إذا كان petCount يساوي maxPets تبلغ المستخدم بأنه لا يمكن إضافة المزيد من الحيوانات الأليفة.

ملاحظة

يتم تطوير التعليمات البرمجية المستخدمة لإدخال بيانات الحيوانات في التمرين التالي، في الوقت الحالي، يتم زيادة petCount كما لو تم إدخال البيانات وحفظها في ourAnimals يمكنك هذا من إنهاء تطوير منطق التعليمات المقترنة بالحلقة while

١. إنشاء سطر فارغ داخل كتلة الحلقة `while (anotherPet == "y" && petCount < maxPets)` التي قمت بإنشائها في المهمة السابقة.

٢. لزيادة petCount أدخل التعليمات التالية:

```
// increment petCount (the array is zero-based, so we increment the counter after adding to the array)
petCount = petCount + 1;
```

٣. للتحقق مما إذا كان petCount أقل من maxPets أدخل التعليمات التالية:

```
// check maxPet limit
if (petCount < maxPets)
{
}
```

٤. داخل كتلة العبارة if التي قمت بإنشائها، لسؤال المستخدم عما إذا كان يريد إضافة حيوان أليف آخر، أدخل التعليمات التالية:

```
// another pet?
Console.WriteLine("Do you want to enter info for another pet (y/n)");
```

٥. أسفل WriteLine() التي قمت بكتابتها، لقراءة رد المستخدم، والتأكد من قيام المستخدم بإدخال "y" أو "n" أدخل التعليمات التالية:

```
do
{
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        anotherPet = readResult.ToLower();
    }

} while (anotherPet != "y" && anotherPet != 
"n");
```

٦. حدد موقع عبارة break التي تفصل case "2"; عنcase "3"; في عبارة switch

٧. لاحظ عبارة Console.WriteLine() وعبارة Console.ReadLine() في نهاية تعليمة case "2";

تعرض هذه التعليمة البرمجية رسالة للمستخدم ثم توقف التطبيق مؤقتاً.

٨. لإحاطة عبارات Console.WriteLine() و Console.ReadLine() داخل عبارة if قم بتحديث التعليمات كما يلي:

```
if (petCount >= maxPets)
{
    Console.WriteLine("Press the Enter key to 
continue.");
    readResult = Console.ReadLine();
}

break;

case "3":
```

تتم زيادة قيمة petCount داخل الحلقة while إذا كان petCount يساوي maxPets فلا يمكن إضافة المزيد من الحيوانات، إلى مصفوفة ourAnimals يجب عليك إخبار المستخدم عند حدوث ذلك.

٩. لإعلام المستخدم بأن Contoso Pets قد وصلت إلى سعتها، قم بتحديث التعليمات على النحو التالي:

```
    if (petCount >= maxPets)
    {
        Console.WriteLine("We have reached our
limit on the number of pets that we can
manage.");
        Console.WriteLine("Press the Enter key
to continue.");
        readResult = Console.ReadLine();
    }

    break;

case "3":
```

١٠. خذ دقيقة لمراجعة التعليمات في تكرار while ورسالة المستخدم التي قمت بإنشائها.

يجب أن تبدو عبارة `while (anotherPet == "y" && petCount < maxPets)` والتعليمة التي تعرض رسالة المستخدم مثل التالي:

```
while (anotherPet == "y" && petCount < maxPets)
{
    // increment petCount (the array is zero-
based, so we increment the counter after adding
to the array)
    petCount = petCount + 1;

    // check maxPet limit
```

```
    if (petCount < maxPets)
    {
        // another pet?
        Console.WriteLine("Do you want to enter
info for another pet (y/n)");
        do
        {
            readResult = Console.ReadLine();
            if (readResult != null)
            {
                anotherPet =
readResult.ToLower();
            }

        } while (anotherPet != "y" &&
anotherPet != "n");
    }
}

if (petCount >= maxPets)
{
    Console.WriteLine("We have reached our
limit on the number of pets that we can
manage.");
    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
}
```

١١. في قائمة **File** حدد **Save**

١٢. افتح لوحة المحطة الطرفية المتكاملة، وأدخل أمر إنشاء البرنامج.

١٣. إصلاح أي أخطاء أو تحذيرات إنشاء تم الإبلاغ عنها قبل المتابعة.

تحقق من عملك

في هذه المهمة، يمكنك تشغيل تطبيقنا، والتحقق من أن منطق التكرار والتفريع الذي قمت بإنشائه يعمل كما هو متوقع.

١. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة

٢. في موجه الأوامر Terminal أدخل **dotnet run**

٣. في موجه الأوامر Terminal أدخل **2**

٤. تحقق من رؤية الرسائل التالية:

```
We currently have 4 pets that need homes. We
can manage 4 more.
Do you want to enter info for another pet (y/n)
```

٥. في موجه الأوامر Terminal أدخل **n**

٦. تحقق أن التعليمات تخرج من الحلقة، لإدخال حيوانات جديدة عند إدخال "n"

إذا كان منطق التعليمات البرمجية يعمل كما هو متوقع، يجب أن ترى القائمة الرئيسية معروضة في المحطة الطرفية the Terminal

إذا لم تخرج التعليمات من التكرار عند المتوقع، فاضغط **على Ctrl + C** في المحطة الطرفية the Terminal لفرض تنفيذ الإيقاف، تحتاج إلى التنقل عبر التعليمات يدوياً، وتتبع قيم متغيرات معايير الخروج، قم بتعديل التعليمات إذا لزم الأمر، للتأكد من إنهاء تكرار while عندما يدخل المستخدم "n" احفظ التغييرات، ثم أعد إنشاء البرنامج، ثم قم بتشغيل اختبار التحقق للوصول مرة أخرى في هذه المرحلة.

٧. في موجه الأوامر Terminal أدخل **2**

مرة أخرى، سترى الرسائل التالية معروضة:

```
We currently have 4 pets that need homes. We
can manage 4 more.
Do you want to enter info for another pet (y/n)
```

٨. في موجه الأوامر Terminal أدخل **y**

٩. خذ دقيقة للنظر في كيفية استخدام petCount في التعليمات البرمجية.

تحتاج إلى فهم منطق التعليمات البرمجية قبل أن تتمكن من التحقق من صحتها.

في هذه الحالة، يعتمد منطق التعليمات البرمجية على العلاقة بين petCount and maxPets أنت تعرف بتعيين قيمة maxPets إلى 8 ولكن ماذا عن petCount تساعد العناصر التالية في تقييم المنطق الذي قمت بتنفيذه.

- تعلم petCount قيمته 4 عند إدخال التكرار الأول while loop
- تعلم بزيادة petCount في كل مرة تتكرر فيها while loop
- تعلم أن القيمة المخصصة لـ petCount والطريقة التي يتم بها زيادة petCount تؤثر على كيفية تخزين البيانات في ourAnimals

تشرح العناصر التالية العلاقة بين petCount والبيانات المخزنة في ourAnimals

١. يضيف التطبيق أربع حيوانات إلى مصفوفة ourAnimals عندما ينشئ نموذج البيانات.

٢. يقوم التطبيق بتخزين البيانات الجديدة إلى ourAnimals عندما تكون قيم petCount هي 4 هذا ليس خطأ، التعليمات منطقية، عندما تتذكر أن عناصر المصفوفة لا تعتمد على الصفر، على سبيل المثال، تحتوي ourAnimals[0,0] على معرف pet ID للحيوان 1 وتحتوي ourAnimals[3,0] على معرف pet ID للحيوان 4 لذلك عندما يكون PetCount 4 تقوم بتخزين البيانات للحيوانات الخامسة.

٣. سيقوم التطبيق بتخزين بيانات الحيوانات إلى المصفوفة، قبل زيادة petCount

٤. يزيد التطبيق من PetCount قبل أن يطالب المستخدم بإضافة حيوان أليف آخر.

٥. عندما يعرض التطبيق **Do you want to enter info for another pet (y/n)** لأول مرة، يتم تعيين petCount بالفعل على 5

- إذا دخل المستخدم Y أولا، **Do you want to enter info for another pet (y/n)** فأنت تعلم ذلك:

١. سوف تكرر

```
while (anotherPet == "y" && petCount < maxPets)
```

أنت تعرف أن الحلقة سوف تتكرر بسبب

```
anotherPet == "y" and petCount < maxPets
```

٢. سيتم زيادة القيمة المعينة لـ petCount (عندما تكرر الحلقة while)

٣. ستكون القيمة المعينة لـ petCount هي 6 (بعد أن يدخل المستخدم Y في المرة الأولى).

ضع هذا التحليل لمنطق التعليمات البرمجية في الاعتبار أثناء متابعة اختبار التطبيق.

١٠. لاحظ تم تحديث لوحة Terminal بالرسالة `another pet?` حيوان أليف آخر؟ ولكن التعليمات البرمجية لا تعرض تحديث petCount

يجب أن تعرض لوحة Terminal الآن الإخراج التالي:

```
We currently have 4 pets that need homes. We
can manage 4 more.
Do you want to enter info for another pet (y/n)
y
Do you want to enter info for another pet (y/n)
```

١١. في موجه الأوامر Terminal أدخل **y**

عند إدخال y مرة ثانية، يتم زيادة petCount إلى 7 لذلك لا يزال petCount أقل من maxPets

١٢. في موجه الأوامر Terminal أدخل **y**

عند إدخال y مرة ثالثة، يتم زيادة petCount إلى 8 لذا petCount الآن يساوي maxPets

١٣. تحقق من أن التعليمات البرمجية تخرج من التكرار while عند إدخال **y** في المرة الثالثة.

يجب أن تعرض لوحة Terminal الآن الإخراج التالي:

```
We currently have 4 pets that need homes. We
can manage 4 more.
Do you want to enter info for another pet (y/n)
y
Do you want to enter info for another pet (y/n)
y
Do you want to enter info for another pet (y/n)
y
We have reached our limit on the number of pets
that we can manage.
Press the Enter key to continue.
```

إذا لم تخرج التعليمات من الحلقة كما هو مطلوب، أفحص التعليمات البرمجية يدويًا، وتتبع قيم متغيرات معايير الخروج، قم بتعديل التعليمات للتأكد من خروجك من الحلقة عندما تصل `petCount` إلى قيمة تساوي `maxPets` استمر في الرد على "y" حتى تعلم أن `petCount` يساوي `maxPets` والتي لها قيمة افتراضية 8

١٤. في موجه الأوامر Terminal اضغط على مفتاح الإدخال Enter لشغيل التطبيق.

١٥. قم بإنهاء التطبيق، ثم أغلق Terminal

٦ تمرين ـ كتابة التعليمات لقراءة وحفظ البيانات الجديدة لمصفوفة ourAnimals

في هذا التمرين، يمكنك تطوير حلقات التحقق من صحة إدخال البيانات لكل خاصية حيوان أليف، ثم تخزين البيانات الجديدة في ourAnimals المهام التفصيلية التي تكملها أثناء هذا التمرين هي:

١. **قراءة الأنواع والتحقق من صحتها:** Read and validate species بناء حلقة، وبنية التعليمات الداخلية المستخدمة للدخول، والتحقق من صحة أنواع الحيوانات الأليفة.

٢. **إنشاء معرف حيوان أليف:** Construct pet ID كتابة التعليمات التي تستخدم petCount واسم الأنواع لإنشاء قيمة petID

٣. **قراءة العمر والتحقق من صحته:** Read and validate age إنشاء حلقة، وبنية التعليمات الداخلية المستخدمة لإدخال عمر الحيوان الأليف، والتحقق من صحته.

٤. **قراءة الوصف البدني والتحقق من صحته:** Read and validate physical description إنشاء حلقة، وبنية التعليمات الداخلية المستخدمة لإدخال وصف فعلي للحيوان الأليف.

٥. **قراءة وصف الشخصية والتحقق من صحته:** Read and validate personality description إنشاء حلقة، وبنية التعليمات الداخلية المستخدمة لإدخال وصف لشخصية الحيوان الأليف.

٦. **قراءة الكنية والتحقق من صحتها:** Read and validate nickname إنشاء حلقة، وبنية التعليمات الداخلية المستخدمة لإدخال لقب للحيوان الأليف.

٧. **اختبار التحقق:** Verification test قم بإجراء اختبارات التحقق للتعليمات التي تطورها في هذا التمرين

هام

يجب إكمال التمرين السابق في هذه الوحدة، قبل بدء هذا التمرين

بناء حلقة لقراءة والتحقق من صحة أنواع الحيوانات الأليفة pet species

في هذه المهمة، يمكنك إنشاء حلقة تكرار do تتكرر حتى يُدخل المستخدم اسمًا صالحًا للأنواع، إما كلبًا أو قطة، يمكنك إعادة استخدام متغير readResult الذي يقبل قيم خالية، لالتقاط إدخال Console.ReadLine() يمكنك أيضًا إعادة استخدام المتغير animalSpecies الذي استخدمته عند إنشاء نموذج البيانات، يمكنك إضافة متغير منطقي جديد باسم validEntry إلى تطبيقك، لاستخدام validEntry في تعبير يتم تقييمه كمعيار خروج لحلقة do

١. تأكد من أن Visual Studio Code مفتوح وأن ملف Program.cs مرئي في المحرر.

٢. حدد موقع العبارة

```
while (anotherPet == "y" && petCount < maxPets)
```

ثم أضف سطر فارغ في الكتلة البرمجية العلوية.

٣. في السطر الفارغ التي قمت بإنشائها، لإعلان validEntry بقيمة أولية false أدخل التعليمات البرمجية التالية:

```
bool validEntry = false;
```

٤. في السطر تحت إعلان validEntry لإنشاء do لإدخال بيانات الأنواع، أدخل التعليمات البرمجية التالية:

```
// get species (cat or dog) - string
animalSpecies is a required field
do
{
} while (validEntry == false);
```

٥. داخل كتلة عبارة do لإنشاء مطالبة عرض وقراءة إدخال المستخدم، أدخل التعليمات البرمجية التالية:

```
Console.WriteLine("\n\rEnter 'dog' or 'cat' to
begin a new entry");
```

```
readResult = Console.ReadLine();
```

٦. للتأكد أن قيمة readResult ليست خالية، قبل تعيين قيمته إلى animalSpecies أدخل التعليمات البرمجية التالية:

```
if (readResult != null)
{
    animalSpecies = readResult.ToLower();

}
```

٧. في السطر أسفل تعيين القيمة animalSpecies للتأكد من أن animalSpecies يحتوي على اسم نوع صالح، أدخل التعليمات البرمجية التالية:

```
if (animalSpecies != "dog" && animalSpecies !=
"cat")
{
    validEntry = false;
}
else
{
    validEntry = true;
}
```

٨. قارن حلقة إدخال بيانات اسم الأنواع المكتملة، بالتعليمات البرمجية التالية:

```
// get species (cat or dog) - string
animalSpecies is a required field
do
{
    Console.WriteLine("\n\rEnter 'dog' or 'cat'
to begin a new entry");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalSpecies = readResult.ToLower();
        if (animalSpecies != "dog" &&
animalSpecies != "cat")
```

```
        {
            //Console.WriteLine($"You entered:
{animalSpecies}.");
            validEntry = false;
        }
        else
        {
            validEntry = true;
        }
    }
} while (validEntry == false);
```

٩. في قائمة **File** حدد **Save**

١٠. افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

بناء قيمة معرف الحيوان animal ID value

في هذه المهمة، يمكنك استخدام المتغيرين petCount , animalSpecies لإنشاء القيمة التي تم تعينها إلى animalID

١. أضف سطر فارغ أسفل كتلة لحلقة إدخال بيانات اسم الأنواع.

٢. لإنشاء القيمة وتعيينها animalID أدخل التعليمات البرمجية التالية:

```
// build the animal the ID number - for example
C1, C2, D3 (for Cat 1, Cat 2, Dog 3)
animalID = animalSpecies.Substring(0, 1) +
(petCount + 1).ToString();
```

٣. في قائمة **File** حدد **Save**

إنشاء حلقة لقراءة عمر الحيوان الأليف pet's age والتحقق من صحته

في هذه المهمة، يمكنك إنشاء حلقة do تتكرر حتى يُدخل المستخدم إما ? أو عددًا صحيحًا يمثل عمر الحيوان الأليف بالسنوات، يمكنك إعادة استخدام readResult الذي يقبل قيم خالية، لالتقاط إدخال Console.ReadLine() يمكنك أيضًا إعادة استخدام متغير animalAge الذي استخدمته عند إنشاء نموذج البيانات، لاختبار ما إذا كان animalAge يمثل عددًا صحيحًا صالحًا أم لا، يمكنك استخدام القيمة المنطقية validEntry تقوم بتعريف متغير عدد صحيح جديد يسمى petAge لتخزين القيمة الرقمية، مرة أخرى، يتم استخدام validEntry في التعبير الذي يتم تقييمه كمعيار خروج لحلقة do الخاصة بنا.

١. أضف سطر تعليمة فارغ أسفل السطر المستخدم لتعيين قيمة إلى المتغير animalID

٢. لإنشاء تكرار do لإدخال بيانات العمر، أدخل التعليمات التالية:

```
// get the pet's age. can be ? at initial
entry.
do
{
} while (validEntry == false);
```

٣. داخل كتلة العبارة do للإعلان عن متغير عدد صحيح يسمى petAge أدخل التعليمات التالية:

```
int petAge;
```

٤. في السطر أسفل إعلان petAge لعرض رسالة، وقراءة إدخال المستخدم، أدخل التعليمات التالية:

```
Console.WriteLine("Enter the pet's age or enter
? if unknown");
readResult = Console.ReadLine();
```

٥. للتأكد أن قيمة readResult ليست خالية، قبل تعيين قيمته إلى animalAge أدخل التعليمات التالية:

```
if (readResult != null)
{
    animalAge = readResult;

}
```

٦. في السطر أسفل تعيين قيمة animalAge للتحقق مما إذا كان المستخدم قد أدخل ? قبل اختبار عدد صالح، أدخل التعليمات التالية:

```
if (animalAge != "?")
{
    validEntry = int.TryParse(animalAge, out
petAge);
}
else
{
    validEntry = true;
}
```

٧. قارن حلقة إدخال بيانات العمر المكتملة بالتعليمات التالية:

```
// get the pet's age. can be ? at initial
entry.
do
{
    int petAge;
    Console.WriteLine("Enter the pet's age or
enter ? if unknown");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalAge = readResult;
        if (animalAge != "?")
        {
```

```
            validEntry =
int.TryParse(animalAge, out petAge);
        }
        else
        {
            validEntry = true;
        }
    }
} while (validEntry == false);
```

٨. في قائمة **File** حدد **Save**

٩. فتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

بناء حلقة لقراءة والتحقق من صحة الوصف الجسدي للحيوان الأليف pet's physical description

في هذه المهمة، يمكنك إنشاء تكرار do يتكرر حتى يدخل المستخدم قيمة نصية تمثل وصفاً جسدياً للحيوان، يمكنك إعادة استخدام readResult لالتقاط الإدخال Console.ReadLine() يمكنك أيضاً إعادة استخدام متغير animalPhysicalDescription الذي استخدمته عند إنشاء نموذج البيانات، تستخدم القيمة المعينة إليه animalPhysicalDescription في التعبير الذي يتم تقييمه كمعيار خروج للحلقة الخاصة بنا do

١. أضف سطر فارغ أسفل كتلة حلقة إدخال بيانات العمر.

٢. لإنشاء تكرار do لإدخال بيانات الوصف الفعلي، أدخل التعليمات التالية:

```
// get a description of the pet's physical
appearance/condition -
animalPhysicalDescription can be blank.
```

```
do
{
} while (animalPhysicalDescription == "");
```

٣. داخل كتلة عبارة do لإنشاء رسالة عرض وقراءة إدخال المستخدم، أدخل التعليمات التالية:

```
Console.WriteLine("Enter a physical description
of the pet (size, color, gender, weight,
housebroken)");
readResult = Console.ReadLine();
```

٤. للتأكد أن قيمة readResult ليست خالية قبل تعيين قيمته إلى animalPhysicalDescription أدخل التعليمات التالية:

```
if (readResult != null)
{
    animalPhysicalDescription =
readResult.ToLower();
}
```

٥. لتعيين قيمة "tbd" إلى animalPhysicalDescription عندما تكون القيمة المدخلة هي "" أدخل التعليمات التالية:

```
if (animalPhysicalDescription == "")
{
    animalPhysicalDescription = "tbd";
}
```

٦. قارن حلقة إدخال بيانات الوصف الجسدي المكتملة بالتعليمات التالية:

```
// get a description of the pet's physical
appearance/condition -
animalPhysicalDescription can be blank.
do
```

```
{
    Console.WriteLine("Enter a physical
description of the pet (size, color, gender,
weight, housebroken)");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalPhysicalDescription =
readResult.ToLower();
        if (animalPhysicalDescription == "")
        {
            animalPhysicalDescription = "tbd";
        }
    }
} while (animalPhysicalDescription == "");
```

٧. في قائمة **File** حدد **Save**

٨. فتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

إنشاء حلقة لقراءة وصف شخصية الحيوان الأليف والتحقق من صحته pet's personality description

في هذه المهمة، يمكنك إنشاء do تتكرر حتى يدخل المستخدم قيمة نصية تمثل وصفاً لشخصية الحيوان، يمكنك إعادة استخدام readResult القابلة للإلغاء، لالتقاط الإدخال Console.ReadLine() يمكنك أيضاً إعادة استخدام متغير animalPersonalityDescription الذي استخدمته عند إنشاء نموذج البيانات، تستخدم قيمته المعينة animalPersonalityDescription في التعبير الذي يتم تقييمه كمعيار خروج للحلقة الخاصة بنا do

١. أضف سطرا فارغا أسفل كتلة حلقة إدخال بيانات الوصف الجسدي.

٢. لإنشاء تكرار do لإدخال بيانات وصف الشخصية، أدخل التعليمات التالية:

```
// get a description of the pet's personality - animalPersonalityDescription can be blank.
do
{
} while (animalPersonalityDescription == "");
```

٣. داخل كتلة العبارة do لإنشاء رسالة عرض، وقراءة إدخال المستخدم، أدخل التعليمات التالية:

```
Console.WriteLine("Enter a description of the pet's personality (likes or dislikes, tricks, energy level)");
readResult = Console.ReadLine();
```

٤. للتأكد من قيمة readResult ليست خالية قبل تعيين قيمته إلى animalPersonalityDescription أدخل التعليمات التالية:

```
if (readResult != null)
{
```

```
    animalPersonalityDescription =
readResult.ToLower();

}
```

٥. لتعيين قيمة "tbd" إلى animalPersonalityDescription عندما تكون القيمة المدخلة هي "" أدخل التعليمات التالية:

```
if (animalPersonalityDescription == "")
{
    animalPersonalityDescription = "tbd";
}
```

٦. قارن حلقة إدخال بيانات وصف الشخصية المكتملة بالتعليمات التالية:

```
// get a description of the pet's personality -
animalPersonalityDescription can be blank.
do
{
    Console.WriteLine("Enter a description of
the pet's personality (likes or dislikes,
tricks, energy level)");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalPersonalityDescription =
readResult.ToLower();
        if (animalPersonalityDescription == "")
        {
            animalPersonalityDescription =
"tbd";
        }
    }
} while (animalPersonalityDescription == "");
```

٧. في قائمة **File** حدد **Save**

٨. فتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

إنشاء حلقة لقراءة الاسم "لقب" للحيوان الأليف والتحقق من صحته pet's nickname

في هذه المهمة، يمكنك إنشاء حلقة do تتكرر حتى يُدخل المستخدم قيمة نصية تمثل لقبًا للحيوان، يمكنك إعادة استخدام readResult لالتقاط إدخال Console.ReadLine() يمكنك استخدام متغير animalNickname الذي استخدمته عند إنشاء نموذج البيانات، يمكنك استخدام القيمة المعينة لـ animalNickname في التعبير الذي تم تقييمه كمعيار خروج لحلقة do الخاصة بنا.

١. أضف سطرًا فارغًا أسفل كتلة حلقة إدخال بيانات وصف الشخصية.

٢. لإنشاء حلقة do لإدخال بيانات الاسم، أدخل التعليمات التالية:

```
// get the pet's nickname. animalNickname can
be blank.
do
{
} while (animalNickname == "");
```

٣. داخل كتلة العبارة do لإنشاء رسالة عرض وقراءة إدخال المستخدم، أدخل التعليمات التالية:

```
Console.WriteLine("Enter a nickname for the
pet");
readResult = Console.ReadLine();
```

٤. للتأكد من قيمة readResult ليست خالية قبل تعيين قيمته إلى animalNickname أدخل التعليمات التالية:

```
if (readResult != null)
{
    animalNickname = readResult.ToLower();

}
```

٥. لتعيين قيمة "tbd" إلى animalNickname عندما تكون القيمة المدخلة هي "" أدخل التعليمات التالية:

```
if (animalNickname == "")
{
    animalNickname = "tbd";
}
```

٦. قارن حلقة إدخال بيانات الكنية المكتملة بالتعليمات التالية:

```
// get the pet's nickname. animalNickname can be
blank.
do
{
    Console.WriteLine("Enter a nickname for the
pet");
    readResult = Console.ReadLine();
    if (readResult != null)
    {
        animalNickname = readResult.ToLower();
        if (animalNickname == "")
        {
            animalNickname = "tbd";
        }
    }
} while (animalNickname == "");
```

٧. في قائمة **File** حدد **Save**

٨. فتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

حفظ معلومات الحيوانات الأليفة الجديدة

في هذه المهمة، يمكنك حفظ القيم التي تم إدخالها لخصائص الحيوانات الأليفة إلى مصفوفة ourAnimals

١. أضف سطرًا فارغًا أسفل كتلة حلقة إدخال بيانات الكنية.

٢. لتخزين قيم البيانات المحددة من قبل المستخدم، أدخل التعليمات التالية:

```
// store the pet information in the ourAnimals
array (zero based)
ourAnimals[petCount, 0] = "ID #: " + animalID;
ourAnimals[petCount, 1] = "Species: " +
animalSpecies;
ourAnimals[petCount, 2] = "Age: " + animalAge;
ourAnimals[petCount, 3] = "Nickname: " +
animalNickname;
ourAnimals[petCount, 4] = "Physical
description: " + animalPhysicalDescription;
ourAnimals[petCount, 5] = "Personality: " +
animalPersonalityDescription;
```

٣. في قائمة **File** حدد **Save**

٤. فتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code وأدخل أمر إنشاء البرنامج.

إذا تم الإبلاغ عن أي أخطاء أو تحذيرات في الإصدار، فصحح المشكلات قبل المتابعة.

تحقق من عملك

في هذه المهمة، يمكنك تشغيل من الوحدة الطرفية المتكاملة، والتحقق من أن إدخال بيانات الحيوانات الأليفة يعمل بشكل صحيح.

١. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

٢. في موجه الأوامر Terminal أدخل **dotnet run**

٣. في موجه الأوامر Terminal أدخل **2**

٤. تحقق من عرض لوحة Terminal الإخراج التالي:

```
We currently have 4 pets that need homes. We can manage 4 more.

Enter 'dog' or 'cat' to begin a new entry
```

٥. أدخل القيم التالية في الأوامر الطرفية the Terminal command وتحقق من عرض كل المطالبات التالية:

- في مطالبة إدخال جديدة `Enter 'dog' or 'cat' to begin a new entry` أدخل كلبًا **dog**
- في المطالبة `Enter the pet's age or enter ? if unknown` أدخل **؟**
- في المطالبة `Enter a physical description of the pet (size, color, gender, weight, housebroken)` اضغط على مفتاح Enter
- في المطالبة `Enter a description of the pet's personality (likes or dislikes, tricks, energy level)` اضغط على مفتاح Enter

- في المطالبة `Enter a nickname for the pet` اضغط على مفتاح Enter

يجب تحديث لوحة Terminal على النحو التالي:

```
Enter 'dog' or 'cat' to begin a new entry
dog
Enter the pet's age or enter ? if unknown
?
Enter a physical description of the pet (size,
color, gender, weight, housebroken)

Enter a description of the pet's personality
(likes or dislikes, tricks, energy level)

Enter a nickname for the pet

Do you want to enter info for another pet (y/n)
```

٦. في موجه الأوامر Terminal أدخل **n**

٧. تحقق من تحديث لوحة Terminal لإظهار خيارات القائمة الرئيسية.

٨. في موجه الأوامر Terminal أدخل **1**

٩. تحقق من تحديث لوحة Terminal لإظهار الإخراج التالي:

```
ID #: d1
Species: dog
Age: 2
Nickname: lola
Physical description: medium sized cream colored
female golden retriever weighing about 65 pounds.
housebroken.
Personality: loves to have her belly rubbed and likes
to chase her tail. gives lots of kisses.

ID #: d2
Species: dog
```

```
Age: 9
Nickname: loki
Physical description: large reddish-brown male golden
retriever weighing about 85 pounds. housebroken.
Personality: loves to have his ears rubbed when he
greets you at the door, or at any time! loves to
lean-in and give doggy hugs.

ID #: c3
Species: cat
Age: 1
Nickname: Puss
Physical description: small white female weighing
about 8 pounds. litter box trained.
Personality: friendly

ID #: c4
Species: cat
Age: ?
Nickname:
Physical description:
Personality:

ID #: d5
Species: dog
Age: ?
Nickname: tbd
Physical description: tbd
Personality: tbd
Press the Enter key to continue
```

إذا لم يتم عرض معلومات الحيوانات الأليفة المضافة حديثا، فتأكد من صحة التعليمات البرمجية لحفظ البيانات في مصفوفة ourAnimals وتأكد من تضمين سطر التعليمات البرمجية لإنشاء petID

١٠. تحقق من إمكانية إنشاء أوصاف حيوانية إضافية للكلاب والقطط، وحفظ خصائص الحيوان في مصفوفة ourAnimals

١١. قم بإنهاء التطبيق، ثم أغلق Terminal

تهانينا على إكمال هذا المشروع الإرشادي! لقد أنشأت تطبيقاً يجمع بين عبارات التحديد والتكرار، لتحقيق أهداف تصميم التطبيق، يتضمن تطبيقك أكثر من ٣٠٠ سطر وينفذ المهام التي قد تجدها في تطبيق احترافي.

يمثل إكمال هذا المشروع إنجازاً هاماً، استمر في التقدم!

٧ اختبر معلوماتك

١- متى يكون من المناسب استخدام عبارة switch-case بدلاً من عبارة if-elseif-else

- تكون العبارة switch-case مناسبة عندما تكون هناك حاجة إلى ٢- ٣ من عبارات else if
- العبارة switch-case مناسبة عندما يكون عدد أقسام case صغيراً
- العبارة switch-case مناسبة عندما تكون عبارة التحديد داخل حلقة التكرار

٢- لماذا يجب على المطور اختيار عبارة for بدلاً من عبارة foreach عند معالجة محتويات مصفوفة متعدد الأبعاد؟

- تقوم العبارات for بعمل أفضل لفحص كل عنصر من المصفوفة بشكل منفصل
- تقوم العبارات foreach بعمل أفضل لفحص كل عنصر من المصفوفة بشكل منفصل
- تمكن عبارات for المطور من التعامل مع أبعاد المصفوفة بشكل منفصل

٣- لماذا من المهم تحديد نطاق متغير عند أدنى مستوى ضروري له؟

- الإعلان عن متغير خارج كتلة برمجية يضمن إمكانية الوصول إليه داخل الكتلة البرمجية قبل تعيين قيمة له
- يضمن الحفاظ على موارد التطبيق وتكون البصمة الأمنية صغيرة
- يضمن إمكانية الوصول إلى متغير داخل فروع التعليمات البرمجية ذات المستوى الأدنى للتطبيق

راجع إجابتك

١ تكون العبارة switch-case مناسبة عندما تكون هناك حاجة إلى ٢-٣ من عبارات else if

صحيح، عندما تكون هناك حاجة إلى ٢-٣ من عبارات else if يمكن أن تصبح التعليمات صعبة القراءة، ويفضل استخدام العبارة switch-case

٢ تمكن عبارات for المطور من التعامل مع أبعاد المصفوفة بشكل منفصل

صحيح، عندما تعالج التعليمات البرمجية محتويات مصفوفة متعدد الأبعاد، غالبا ما يريد المطور التكرار من خلال أبعاد المصفوفة بشكل منفصل، توفر العبارة for دعماً أفضل لمعالجة أبعاد المصفوفة بشكل منفصل

٣ يضمن الحفاظ على موارد التطبيق وتكون البصمة الأمنية صغيرة

صحيح، يتيح الاحتفاظ بالمتغيرات محددة النطاق عند أدنى مستوى ضروري إدارة موارد أفضل ويساعد على تقليل ملف تعريف الهجوم للتطبيق.

٨ الملخص

كان هدفك هو إنشاء تطبيق ينفذ مجموعة من عبارات التحديد والتكرار لتحقيق أهداف تصميم تطبيقك.

من خلال إنشاء مجموعات متداخلة من عبارات التكرار والاختيار، قمت بإنشاء واجهة، تمكن المستخدم من عرض بيانات الحيوانات الأليفة من مصفوفة ourAnimals كما قمت بتطوير الميزات لإدخال خصائص الحيوانات الأليفة الجديدة، والتحقق من صحتها، وإضافتها إلى مصفوفة ourAnimals يمكن لمستخدم التطبيق الاستمرار في إجراء تحديدات القائمة حتى يختار الخروج من التطبيق.

تمكنك القدرة على تنفيذ مجموعة متنوعة من عبارات التكرار والاختيار من تحسين تعليماتك البرمجية.

الوحدة السابعة

مشروع التحدي - تطوير هياكل التفريع وحلقات التكرار

أظهر قدرتك على تطوير تطبيق وحدة التحكم الذي ينفذ بيانات التحديد والتكرار لتحقيق مواصفات التطبيق.

الأهداف التعليمية:

في هذه الوحدة، ستثبت قدرتك على:

- استخدم Visual Studio Code لتطوير تطبيق وحدة التحكم C# الذي يستخدم مجموعة من بيانات التحديد والتكرار، لتنفيذ مهام سير العمل المنطقية وفقًا لبيانات التطبيق المتوفرة وتفاعلات المستخدم.
- تقييم الشروط الأساسية، واتخاذ قرارًا مستنيرًا عند الاختيار بين عبارات if-elseif-else وswitch وبين عبارات foreach وfor وwhile وdo
- تحديد نطاق المتغيرات على المستوى المناسب داخل التطبيق.

محتويات الوحدة:

١- مقدمة

٢- الإعداد

٣- تمرين - تأكد أن petAge and petPhysicalDescription تحتوي على معلومات صالحة

٤- تمرين - تأكد من اكتمال أسماء ألقاب الحيوانات الأليفة، ووصف الشخصية

٥- اختبر معلوماتك

٦- الملخص

١ المقدمة

غالباً ما تستخدم التطبيقات مجموعة من عبارات التحديد والتكرار selection and iteration statements لإنشاء مسارات تنفيذ التعليمات البرمجية، بالإضافة إلى ذلك، تؤثر مدخلات المستخدم، والحسابات على التدفق من خلال تطبيق، قد يكون إنشاء واجهة مستخدم تنفذ مواصفات تصميم أمراً صعباً.

لنفترض أنك مطور يعمل على تطبيق Contoso Pets وهو تطبيق يستخدم للعثور على منازل للحيوانات الضالة، وقد اكتملت بالفعل بعض أعمال التطوير، على سبيل المثال، تم تطوير القائمة الرئيسية للتطبيق، والتعليمات المستخدمة لتخزين معلومات الحيوانات الجديدة، ومع ذلك، لا تتوفر بعض المعلومات، عند إدخال حيوان في نظامك، تحتاج إلى تطوير الميزات التي تضمن وجود مجموعة بيانات كاملة لكل حيوان في رعايتك.

في هذه الوحدة، ستقوم بتطوير الميزات التالية لتطبيق Contoso Pets

- ميزة تضمن اكتمال الأعمار الحيوانية، والأوصاف الجسدية.
- ميزة تضمن اكتمال أوصاف كنية الحيوان، وشخصيته.

في نهاية هذه الوحدة، سيضمن تطبيق Contoso Pets اكتمال كل عنصر في مصفوفة ourAnimals

ملاحظة

هذه وحدة مشروع تحدي حيث ستكمل مشروعاً، تهدف هذه الوحدة إلى أن تكون اختباراً لمهاراتك؛ هناك القليل من التوجيهات، ولا توجد تعليمات خطوة بخطوة.

٢ الإعداد

في مشروع التحدي هذا، سوف تستخدم Visual Studio Code لتطوير أجزاء من تطبيق وحدة تحكم C# سوف تستخدم التعبيرات المنطقية، وعبارات التحديد، وعبارات التكرار، لتنفيذ ميزات مواصفات التصميم، أثناء تطوير التطبيق، ستحتاج إلى تحديد نطاق المتغيرات على المستوى المناسب.

مواصفات المشروع

يتضمن مشروع التعليمات ل Starter لهذه الوحدة ملف Program.cs مع ميزات التعليمات البرمجية التالية:

- تعلن التعليمات عن المتغيرات المستخدمة لجمع بيانات الحيوانات الأليفة وتحديدات عناصر القائمة ومعالجتها.
- تعلن التعليمة البرمجية عن مصفوفة `ourAnimals` الذي تتضمن المعلومات التالية لكل حيوان في المصفوفة:
  - رقم معرف الحيوانات الأليفة.
  - أنواع الحيوانات الأليفة (القط أو الكلب).
  - عمر الحيوانات الأليفة (سنوات).
  - وصف للمظهر الجسدي للحيوان الأليف.
  - وصف لشخصية الحيوان الأليف.
  - لقب الحيوان الأليف
- تستخدم التعليمات البرمجية حلقة عبارة `switch-case` لتعبئة عناصر مصفوفة `ourAnimals`
- تتضمن التعليمات البرمجية حلقة حول قائمة رئيسية تنتهي عندما يدخل المستخدم "exit" تتضمن القائمة الرئيسية ما يلي:

١. سرد جميع معلومات الحيوانات الأليفة الحالية لدينا.

٢. تعيين قيم لحقول مصفوفة `ourAnimals` الخاصة بنا.

٣. تأكد من اكتمال الأعمار الحيوانية، والأوصاف الجسدية.

٤. تأكد من اكتمال الأسماء المستعارة الحيوانية، ووصف الشخصية.

٥. تحرير عمر الحيوان.

٦. تحرير وصف شخصية الحيوان.

٧. عرض جميع القطط بخاصية محددة.

٨. عرض جميع الكلاب بخاصية محددة.

أدخل تحديد عنصر القائمة أو اكتب "إنهاء" للخروج من البرنامج

- تقرأ التعليمات تحديد عنصر القائمة الخاص بالمستخدم، وتستخدم عبارة `switch` لتفرع التعليمات البرمجية لكل رقم عنصر قائمة.
- تتضمن التعليمات تنفيذ خيارات القائمة 1 و2
- تعرض التعليمات رسالة "قيد الإنشاء" لخيارات القائمة 3-8

هدفك في هذا التحدي هو إنشاء ميزات التطبيق المتوافقة مع خيارات القائمة 3 و4

ملاحظة

يجب إضافة الحيوانات الجديدة إلى مصفوفة ourAnimals عند وصولها، ومع ذلك، قد يكون عمر الحيوان، وبعض الخصائص الجسدية لحيوان أليف غير معروف حتى بعد فحص الطبيب البيطري، بالإضافة إلى ذلك، قد يكون لقب الحيوان وشخصيته غير معروفين عند وصول حيوان أليف لأول مرة. ستضمن الميزات الجديدة التي تقوم بتطويرها وجود مجموعة بيانات كاملة لكل حيوان في مصفوفة ourAnimals

لضمان اكتمال أعمار الحيوانات والأوصاف الجسدية، يجب أن تكون التعليمات البرمجية:

- قم بتعيين قيمة رقمية صالحة إلى petAge لأي حيوان تم تعيين بيانات له في مصفوفة ourAnimals ولكن لم يتم تعيين عمر له.
- قم بتعيين نص صالح إلى petPhysicalDescription لأي حيوان تم تعيين بيانات له في مصفوفة ourAnimals ولكن لم يتم تعيين وصف فعلي له.
- تحقق من أن الأوصاف البدنية لها قيمة معينة، لا يمكن أن تحتوي القيم المعينة على صفر من الأحرف، أي متطلبات أخرى متروكة لك.

لضمان اكتمال أوصاف لقب والشخصية الحيوانية، يجب أن تكون تعليماتك البرمجية:

- قم بتعيين نص صالح إلى petNickname لأي حيوان تم تعيين بيانات له في مصفوفة ourAnimals ولكن لم يتم تعيين لقب أو اسم مستعار له.
- قم بتعيين نص صالح إلى petPersonalityDescription لأي حيوان تم تعيين بيانات له في مصفوفة ourAnimals ولكن لم يتم تعيين وصف شخصية له.
- تحقق من أن الأسماء المستعارة ووصف الشخصية لها قيمة معينة، يجب أن تحتوي القيم المعينة على حرف واحد على الأقل، أي متطلبات أخرى متروكة لك

الإعداد

استخدم الخطوات التالية للتحضير لتمارين مشروع التحدي:

١. لتنزيل ملف مضغوط يحتوي على مشروع Starter حدد الارتباط التالي Lab Files

٢. فك ضغط ملفات التنزيل.

دون موقع المجلد المستخرج.

٣. انسخ مجلد ChallengeProject **المستخرج** إلى مجلد سطح مكتب Windows

ملاحظة

إذا كان هناك مجلد يسمى ChallengeProject موجود بالفعل، يمكنك تحديد استبدال الملفات في الوجهة لإكمال عملية النسخ

٤. افتح مجلد ChallengeProject **الجديد**

- في **القائمة File** حدد **Open Folder**
- انتقل إلى مجلد Windows Desktop وحدد موقع مجلد "ChallengeProject"
- حدد **ChallengeProject** ثم حدد **Select Folder**

يجب أن تظهر **طريقة عرض Visual Studio Code EXPLORER** **مجلد ChallengeProject** ومجلدين فرعيين باسم **Final** و Starter

أنت الآن جاهز لبدء تمارين مشروع التحدي. حظ سعيد

٣ تمرين ـ تأكد أن petAge and petPhysicalDescription تحتوي على معلومات صالحة

يتم استخدام تطبيق Contoso Pets للمساعدة في العثور على منازل جديدة للحيوانات الأليفة، هدفك في هذا التحدي هو تطوير ميزات التطبيق المستخدمة، لتضمن أن لديك مجموعة بيانات مكتملة لكل حيوان، في مصفوفة ourAnimals

المواصفات Specification

في هذا التمرين، تحتاج إلى تطوير ميزة تضمن اكتمال أعمار الحيوان والأوصاف الجسدية.

يجب أن تكون هذه الميزة:

- يتم تمكينه داخل فرع التطبيق المناسب (يجب عدم استبدال التعليمات الموجودة في فرع التعليمات لخيار القائمة 2)
- قم بتخطي أي حيوان في مصفوفة ourAnimals عندما يتم تعيين قيمة معرف الحيوان الأليف pet ID على القيمة الافتراضية.
- اعرض قيمة معرف الحيوان الأليف pet ID واطلب من المستخدم قيمة بيانات محدثة إذا كانت بيانات مصفوفة ourAnimals مفقودة أو غير كاملة
- تأكد من تعيين قيمة رقمية صالحة لـ animalAge لجميع الحيوانات الموجودة في مصفوفة ourAnimals التي قامت بتعيين البيانات.
- تأكد من تعيين نص صالحة إلى animalPhysicalDescription لجميع الحيوانات في مصفوفة ourAnimals التي قامت بتعيين البيانات.
- **فرض قواعد التحقق التالية لـ** animalAge
    - يجب أن يكون ممكناً تحويل القيمة التي تم إدخالها إلى نوع البيانات الرقمية.
- **فرض قواعد التحقق التالية ل** animalPhysicalDescription

- لا يمكن أن تكون القيم خالية.
- لا يمكن أن تحتوي القيم على أحرف صفرية.
- أي قيود أخرى متروكة للمطور.

- قم بإعلام مستخدم التطبيق عند استيفاء جميع متطلبات البيانات، مع إيقاف التطبيق مؤقتاً، لضمان إمكانية رؤية الرسالة والاستجابة لها.

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله

ملاحظة

يمكنك إنهاء اختبار التحقق قبل إكمال جميع الخطوات، إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في لوحة Terminal اضغط على Ctrl-C بعد الخروج من التطبيق قيد التشغيل، أكمل عمليات التحرير التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، وحفظ تحديثاتك في ملف Program.cs ثم إعادة إنشاء التعليمات البرمجية وتشغيلها

٢. في موجه الأوامر Terminal أدخل **3**

٣. تحقق من تحديث لوحة Terminal برسالة مشابهة للرسالة التالية:

```
Enter an age for ID #: c4
```

٤. في موجه الأوامر Terminal أدخل **"one"**

٥. تحقق أن تعليماتك البرمجية تكرر المطالبة بطلب قيمة لعمر الحيوان الأليف.

يجب تحديث لوحة Terminal لإظهار المطالبة المتكررة، يجب أن يكون العرض مشابها للآتي:

```
Enter an age for ID #: c4
one
Enter an age for ID #: c4
```

٦. في موجه الأوامر Terminal أدخل **1**

٧. تحقق من أن تعليماتك **تقبل 1** كإدخال رقمي صالح، وأن اللوحة الطرفية تعرض رسالة مشابهة للرسالة التالية:

```
Enter a physical description for ID #: c4
(size, color, breed, gender, weight,
housebroken)
```

٨. في موجه الأوامر Terminal اضغط على المفتاح Enter دون كتابة أي حرف

٩. تحقق من أن تعليماتك تكرر المطالبة، التي تطلب قيمة للوصف البدني للحيوان الأليف.

يجب تحديث لوحة Terminal لإظهار المطالبة المتكررة. يجب أن يكون العرض مشابها للآتي:

```
Enter a physical description for ID #: c4
(size, color, gender, weight, housebroken)

Enter a physical description for ID #: c4
(size, color, gender, weight, housebroken)
```

١٠. في موجه الأوامر Terminal أدخل **small white Siamese cat weighing about 8 pounds. litter box trained.**

١١. تحقق من أن تعليماتك البرمجية **تقبل** **small white Siamese cat weighing about 8 pounds. litter box trained.** كإدخال صالح وأن لوحة Terminal تعرض رسالة مشابهة للرسالة التالية:

```
Age and physical description fields are
complete for all of our friends.
Press the Enter key to continue
```

١٢. إذا حددت المزيد من القيود للإدخالات الصالحة، فقم بتشغيل حالات الاختبار المناسبة للتحقق من عملك

ملاحظة

إذا كانت التعليمات البرمجية تفي بالمتطلبات، يجب أن تكون قادراً على إكمال كل خطوة بالترتيب، ورؤية النتائج المتوقعة في اجتياز اختبار واحد، إذا قمت بإضافة قيود إضافية، فقد تحتاج إلى الخروج من التطبيق ثم تشغيل اجتياز اختبار منفصل لإكمال التحقق الخاص بك

بمجرد التحقق من صحة نتائج هذا التمرين، انتقل إلى التمرين التالي في هذا التحدي.

٤ تمرين ـ تأكد من اكتمال أسماء ألقاب الحيوانات الأليفة ووصف الشخصية

يتم استخدام تطبيق Contoso Pets للمساعدة في العثور على منازل جديدة للحيوانات الأليفة، هدفك في هذا التحدي هو تطوير ميزات التطبيق المستخدمة لضمان أن لدينا مجموعة بيانات مكتملة لكل حيوان في مصفوفة ourAnimals

المواصفات

تحتاج إلى تطوير ميزة تضمن اكتمال الألقاب للحيوانات ووصف الشخصية.

يجب أن تكون هذه الميزة:

- تمكين داخل فرع التطبيق المناسب (يجب عدم الكتابة فوق التعليمات البرمجية في فرع التعليمات البرمجية لخيار القائمة 2)
- تخطي أي حيوان في مصفوفة ourAnimals عند تعيين قيمة معرف الحيوانات الأليفة إلى القيمة الافتراضية.
- عرض قيمة معرف الحيوانات الأليفة ومطالبة المستخدم بقيمة بيانات محدثة إذا كانت بيانات مصفوفة ourAnimals مفقودة أو غير مكتملة.
- تأكد من تعيين نص صالح إلى animalNickname لجميع الحيوانات في مصفوفة ourAnimals الذي قام بتعيين البيانات.
- تأكد من تعيين نص صالح إلى animalPersonalityDescription جميع الحيوانات في مصفوفة ourAnimals الذي قام بتعيين البيانات.
- **فرض قواعد التحقق من الصحة التالية** ل petNickname و petPersonalityDescription
    - لا يمكن أن تكون القيم خالية.
    - لا يمكن أن تحتوي القيم على صفر من الأحرف.

- أي قيود أخرى متروكة للمطور.

- قم بإعلام مستخدم التطبيق عند استيفاء جميع متطلبات البيانات، مع إيقاف التطبيق مؤقتاً لضمان إمكانية رؤية الرسالة، والاستجابة لها.

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله.

٢. في موجه الأوامر Terminal، أدخل 4

٣. تحقق من تحديث لوحة Terminal برسالة مشابهة للرسالة التالية:

```
Enter a nickname for ID #: c4
```

٤. في موجه الأوامر Terminal اضغط على المفتاح Enter دون كتابة أي أحرف

٥. تحقق من أن التعليمات البرمجية تكرر المطالبة بطلب قيمة للقب للحيوان الأليف.

يجب تحديث لوحة Terminal لعرض شيء مشابه للآتي:

```
Enter a nickname for ID #: c4

Enter a nickname for ID #: c4
```

٦. في موجه الأوامرTerminal أدخل **snowflake**

٧. تحقق من أن التعليمات البرمجية **تقبل** **snowflake** كإدخال صالح وأن لوحة Terminal تعرض رسالة مشابهة للرسالة التالية:

```
Enter a personality description for ID #: c4
(likes or dislikes, tricks, energy level)
```

٨. في موجه الأوامر Terminal اضغط على المفتاح Enter دون كتابة أي أحرف

٩. تحقق من أن التعليمات البرمجية تكرر المطالبة التي تطلب قيمة لوصف شخصية الحيوان الأليف.

يجب تحديث لوحة Terminal لعرض شيء مشابه للآتي:

```
Enter a personality description for ID #: c4
(likes or dislikes, tricks, energy level)

Enter a personality description for ID #: c4
(likes or dislikes, tricks, energy level)
```

١٠. في موجه الأوامر Terminal أدخل **loves to curl up in a warm spot**

١١. تحقق من أن التعليمات **تقبل** **loves to curl up in a warm spot** كإدخال صالح وأن لوحة Terminal تعرض رسالة مشابهة للرسالة التالية:

```
Nickname and personality description fields are
complete for all of our friends.
Press the Enter key to continue
```

١٢. إذا حددت المزيد من القيود للإدخالات الصالحة، فقم بتشغيل حالات الاختبار المناسبة للتحقق من عملك.

تهانينا إذا نجحت في هذا التحدي

٥ اختبر معلوماتك

١ يقوم مطور بكتابة تطبيق في Visual Studio Code يحفظ التغييرات في ملف Program.cs ثم يقوم بتشغيل أمر dotnet build في Terminal ما هي المعلومات التي يمكنهم العثور عليها في رسائل الخطأ والتحذير التي تم الإبلاغ عنها؟

- اقتراح واحد أو أكثر حول كيفية إصلاح أي مشكلات في بناء الجملة تم اكتشافها
- رقم السطر ومعلومات حول سبب الخطأ
- ارتباطات إلى وثائق التعليمات

٢ يحتاج المطور إلى إنشاء عبارة تكرار، في أي حالة تكون عبارة while خيارًا أفضل من عبارة do؟

- عندما تكون قيم التعبير التي تم تقييمها بواسطة عبارة التكرار غير معروفة قبل كتلة التعليمات البرمجية للتكرار
- عندما تكون قيم التعبير التي تم تقييمها بواسطة عبارة التكرار معروفة قبل كتلة التكرار
- عندما يقوم المطور بإنشاء حلقات أو while متداخلة do

راجع إجابتك

١ رقم السطر ومعلومات حول سبب الخطأ

صحيح تضم رسائل الخطأ والتحذير وصفًا للمشكلة ورقم السطر الذي حدثت فيه

٢ عندما تكون قيم التعبير التي تم تقييمها بواسطة عبارة التكرار معروفة قبل كتلة التكرار

صحيح. عندما تكون التعليمات قادرة على تقييم التعبير قبل الحلقة، فإن عبارة while تسمح للتعليمات بتنفيذ تكرارات صفرية

٦ الملخص

كان هدفك هو إظهار القدرة على تطوير ميزات تطبيق، يحقق مواصفات التصميم، كنت بحاجة إلى اختيار وتنفيذ أنواع عبارة التكرار والتحديد المناسبة، لتلبية المتطلبات المذكورة.

من خلال إنشاء مجموعات متداخلة من عبارات التكرار والاختيار، قمت بإنشاء واجهة مستخدم، تمكن مستخدم التطبيق من إدخال بيانات الحيوانات الأليفة الصالحة، التي سدت الثغرات في مصفوفة ourAnimals

توضح قدرتك على تنفيذ ميزات تطبيق Contoso Pets استناداً إلى مواصفات التصميم فهمك لعبارات التكرار والاختيار.

الفصل الرابع

العمل مع البيانات المتغيرة في تطبيقات وحدة تحكم باستخدام C#

التعمق في البيانات والأنواع، وتعلم كيفية معالجة البيانات النصية والرقمية، وتنفيذ العمليات على المصفوفات، خلال مسار التعلم هذا، سوف تتمكن مما يلي:

- اختيار نوع البيانات الصحيح للمدخلات التي تعمل معها
- إرسال البيانات وتحويلها من نوع إلى آخر
- تعديل البيانات النصية، مع تنسيق عرضها، أو تغييرها لمحتوى الجمل النصية.
- معالجة المصفوفات، وإضافة البيانات وإزالتها وفرزها

المتطلبات الأساسية

- خبرة بمفاهيم C# بما في ذلك الكشف عن المتغيرات، وتهيئتها، واستخدامها
- خبرة مع عبارة if-elseif-else
- خبرة مع المصفوفات وعبارة foreach
- تطوير التعليمات البرمجية C# وبناها، وتشغيلها، باستخدام Visual Studio Code

المحتويات

الوحدة الأولي:

اختيار نوع البيانات المناسب لتعليماتك البرمجية

الوحدة الثانية:

تحويل أنواع البيانات باستخدام تقنيات الإرسال والتحويل

الوحدة الثالثة:

تنفيذ العمليات على المصفوفات باستخدام الأساليب المساعدة " مُدمجة"

الوحدة الرابعة:

تنسيق البيانات الأبجدية الرقمية للعرض

الوحدة الخامسة:

تعديل محتوى الجمل النصية باستخدام الأساليب المساعدة "مُدمجة" لبيانات string

الوحدة السادسة:

مشروع إرشادي - العمل مع البيانات المتغيرة

الوحدة السابعة:

مشروع التحدي - العمل مع البيانات المتغيرة

الوحدة الأولى

اختيار نوع البيانات المناسب لتعليماتك البرمجية

تعرّف على الفرق بين العديد من أنواع البيانات، وكيفية عملها، وما تفعله، وكيفية تفضيل نوع على آخر.

الأهداف التعليمية:

- تعرف على الاختلافات الأساسية بين أنواع القيم value types وأنواع المراجع reference types
- وصف خصائص العديد من أنواع البيانات الرقمية الجديدة، بما في ذلك الأنواع المتكاملة الجديدة new integral types وأنواع الفاصلة العائمة floating-point types
- كتابة التعليمات البرمجية التي تقوم بإرجاع الحد الأقصى والحد الأدنى للقيم، التي يمكن لأنواع البيانات الرقمية تخزينها.
- استخدم الكلمة الأساسية new لإنشاء مثيلات جديدة من نوع مرجع reference type
- تحديد نوع البيانات المناسب، الذي يجب عليك اختياره لتطبيق معين.

محتويات الوحدة:

١ المقدمة

تعتمد لغة البرمجة C# على أنواع البيانات بشكل كبير، تقوم أنواع البيانات بتقييد أنواع القيم، التي يمكن تخزينها في متغير معين، والتي قد تكون مفيدة عند محاولة إنشاء تعليمات برمجية خالية من الأخطاء، بصفتك مطوراً، يمكنك تنفيذ العمليات بثقة على المتغيرات، لأنك تعرف مسبقاً أنه يخزن القيم المناسبة فقط.

لنفترض أن مهمتك هي بناء تطبيق جديد، يجب عليه استرداد ومعالجة وتخزين أنواع مختلفة من البيانات المتعددة، بما في ذلك القيم الرقمية الفردية، وتسلسلات من قيم رقمية ونصية، اختيار أنواع البيانات الصحيحة أمر بالغ الأهمية، لنجاح جهود تطوير البرامج، ولكن ما هي خياراتك، وما هي المعايير التي يجب أن تستخدمها، عند مواجهة العديد من أنواع البيانات التي تبدو متشابهة؟

في هذه الوحدة، ستتعرف على كيفية تخزين التطبيق للبيانات ومعالجتها، تعلم أن هناك نوعين من أنواع البيانات، التي تتوافق مع الطريقتين لمعالجة البيانات، يمكنك كتابة التعليمات البرمجية التي تحدد القيم القصوى والدنيا التي يمكن تخزينها، في نوع بيانات رقمي معين، وتتعلم المعايير التي يجب استخدامها عند الاختيار بين عدة أنواع بيانات رقمية لتطبيقك.

في نهاية هذه الوحدة، ستكون واثقاً عند العمل مع أنواع بيانات مختلفة في C# وستتمكن من اختيار نوع البيانات المناسب الخاص لتطبيقك.

٢ اكتشاف أنواع القيم value types وأنواع المراجع reference types

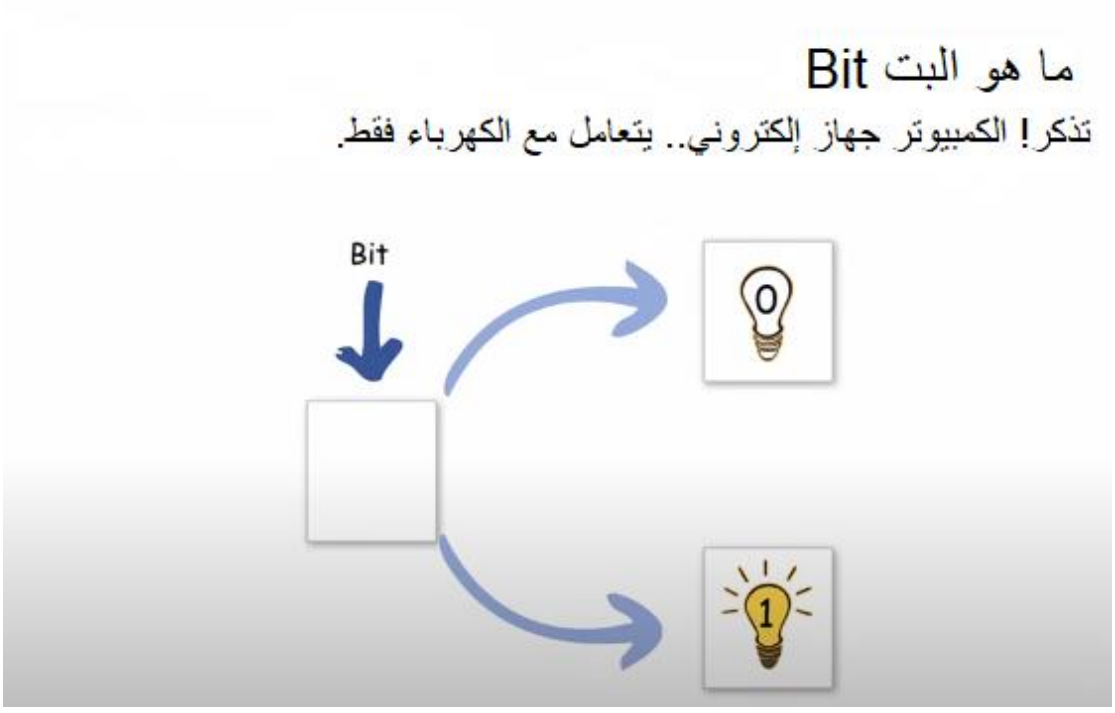
مع توفر العديد من أنواع البيانات في C# يعني اختيار النوع المناسب لاستخدامه، أنت بحاجة إلى فهم متى يمكنك اختيار نوع بيانات على نوع آخر.

قبل مناقشة سبب اختيار نوع بيانات على آخر، تحتاج إلى فهم المزيد حول أنواع البيانات، تحتاج أيضاً إلى معرفة كيفية عمل أنواع البيانات، والبيانات المخزنة في C# وNET.

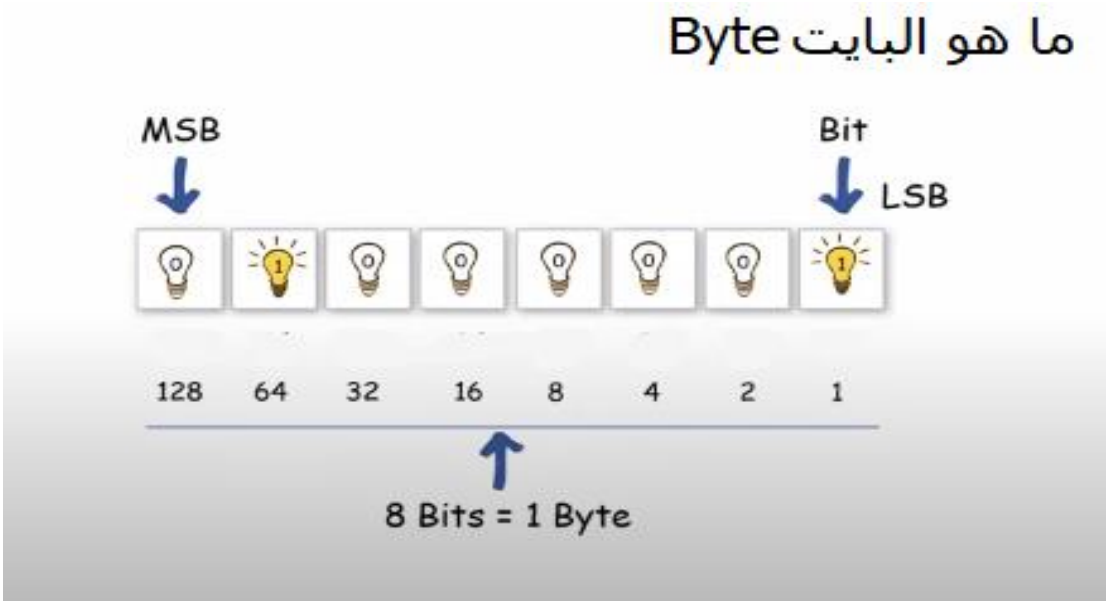
ما المقصود بالبيانات؟

تعتمد الإجابة على سؤال "ما هي البيانات" على من تسأل وفي أي سياق تسأله.

في تطوير البرمجيات، البيانات هي في الأساس قيمة يتم تخزينها في ذاكرة الكمبيوتر كسلسلة من البتات bits البت **bit** هو مفتاح ثنائي بسيط يتم تمثيله بالرقم 0 أو 1 أو بالأحرى "إيقاف" و"تشغيل"



لا يبدو البت الواحد مفيدًا، ولكن عند دمج 8 بتات معًا بالترتيب، فإنها تشكل بايت **byte** عند استخدام البيانات في بايت byte يأخذ كل بت Bit معنى في الترتيب.



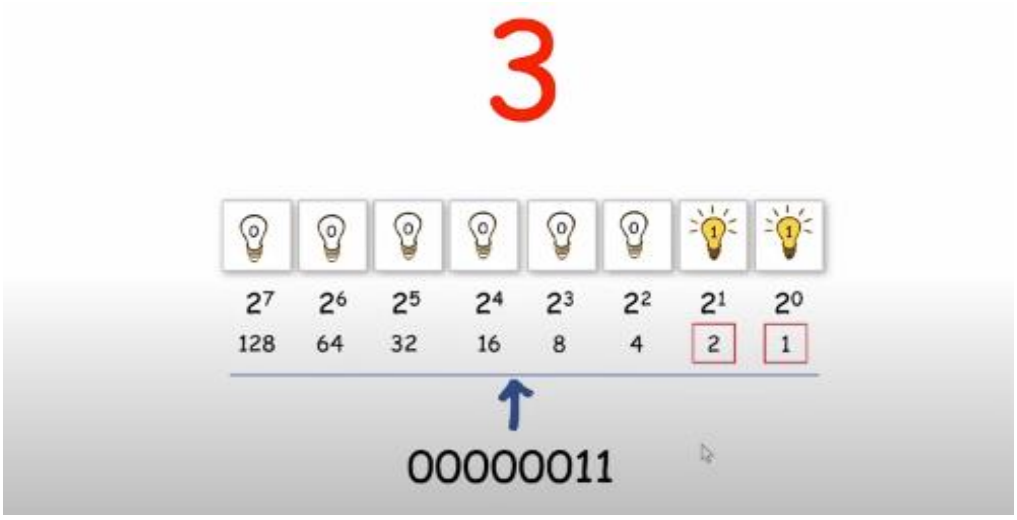
في الواقع، يمكنك تمثيل 256 تركيبة مختلفة بـ 8 بتات فقط، إذا كنت تستخدم نظام رقمي ثنائي (base-2)

على سبيل المثال، في نظام الأرقام الثنائية، يمكنك تمثيل الرقم 195 على أنه `11000011` يساعدك الجدول التالي على تصور كيفية عمل ذلك، يحتوي الصف الأول على ثمانية أعمدة تتوافق مع موضع بالبايت byte يمثل كل موضع قيمة رقمية مختلفة، يمكن للصف الثاني تخزين قيمة البت bit الفردية 0 or 1

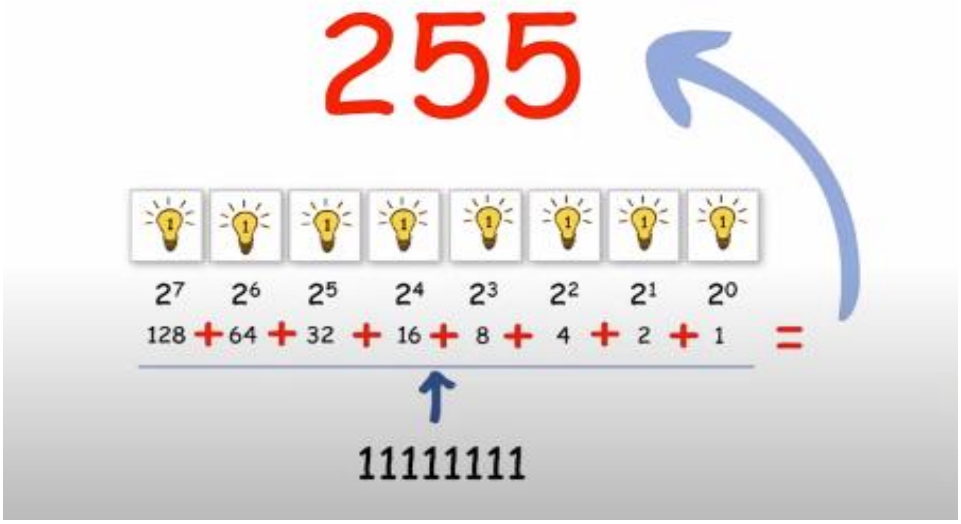
128	64	32	16	8	4	2	1
1	1	0	0	0	0	1	1

إذا قمت بإضافة الرقم من كل عمود في الصف الأول الذي يتوافق مع الرقم 1 في الصف الثاني، فستحصل على الرقم العشري المكافئ لتمثيل النظام الرقمي الثنائي. في هذه الحالة، سيكون `128 + 64 + 2 + 1 = 195`

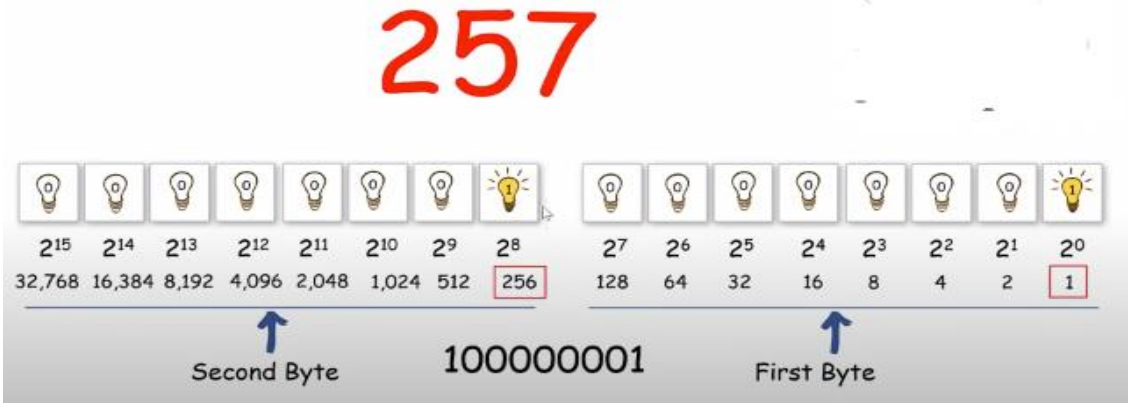
مثال آخر، يمثل الرقم 3 في النظام الثنائي، موضح في الصورة التالية:



لتمثيل الرقم 255



للعمل مع قيم أكبر بعد 255 يخزن الكمبيوتر المزيد من وحدات البايت (عادة 32-bit or 64-bit) إذا كنت تعمل مع ملايين الأرقام الكبيرة في إعداد علمي، فقد تحتاج إلى التفكير بعناية أكبر في أنواع البيانات التي تستخدمها، قد تتطلب التعليمات البرمجية المزيد من الذاكرة لتشغيلها.



ماذا عن البيانات النصية؟ textual data

إذا كان الكمبيوتر يفهم `0s` and `1s` فكيف يسمح لك بالعمل مع النص؟ باستخدام نظام مثل ASCII (رمز قياسي أمريكي لتبادل المعلومات) يمكنك استخدام بايت واحد single byte لتمثيل الأحرف الكبيرة والصغيرة، والأرقام، وعلامة التبويب، ومساحة للخلف backspace، والخط الجديد، والعديد من الرموز الرياضية.

على سبيل المثال، إذا كنت تريد تخزين حرف a صغير كقيمة في تطبيقي، فسيفهم الكمبيوتر الشكل الثنائي لتلك القيمة فقط، لفهم أفضل لكيفية معالجة الكمبيوتر لحرف a صغير، أحتاج إلى تحديد موقع جدول ASCII الذي يوفر قيم ASCII ومعادلتها العشرية، يمكنك البحث عن مصطلحات ASCII البحث العشري، لتحديد موقع مورد مثل هذا على الإنترنت.

Letter	Binary	Decimal	Letter	Binary	Decimal
A	01000001	65	a	01100001	97
B	01000010	66	b	01100010	98
C	01000011	67	c	01100011	99
D	01000100	68	d	01100100	100
E	01000101	69	e	01100101	101
F	01000110	70	f	01100110	102
G	01000111	71	g	01100111	103
H	01001000	72	h	01101000	104
I	01001001	73	i	01101001	105
J	01001010	74	j	01101010	106
K	01001011	75	k	01101011	107
L	01001100	76	l	01101100	108
M	01001101	77	m	01101101	109
N	01001110	78	n	01101110	110
O	01001111	79	o	01101111	111
P	01010000	80	p	01110000	112
Q	01010001	81	q	01110001	113
R	01010010	82	r	01110010	114
S	01010011	83	s	01110011	115
T	01010100	84	t	01110100	116
U	01010101	85	u	01110101	117
V	01010110	86	v	01110110	118
W	01010111	87	w	01110111	119
X	01011000	88	x	01111000	120
Y	01011001	89	y	01111001	121
Z	01011010	90	z	01111010	122

في هذه الحالة، يكون الحرف a الصغير مكافئاً للقيمة 97 العشرية، ثم يمكنك استخدام نفس النظام الرقمي الثنائي في الاتجاه المعاكس، للعثور على كيفية تخزين حرف a ASCII بواسطة الكمبيوتر.

بما أن `64 + 32 + 1 = 97` فإن رمز ASCII الثنائي ذو `8-bit` لـ a هو `01100001`

من المحتمل أنك لن تحتاج أبدًا إلى إجراء هذه الأنواع من التحويلات بنفسك، ولكن فهم منظور الكمبيوتر للبيانات هو مفهوم أساسي، خاصة أثناء التفكير في أنواع البيانات.

ما هو نوع البيانات؟ data type

نوع البيانات هو طريقة تحديد لغة البرمجة مقدار الذاكرة، التي يجب حفظها لقيمة، هناك العديد من أنواع البيانات في لغة C# لاستخدامها، للعديد من التطبيقات، وأحجام البيانات المختلفة.

بالنسبة لمعظم التطبيقات التي تبنيها في حياتك المهنية، سوف تستقر على مجموعة فرعية صغيرة، من جميع أنواع البيانات المتاحة، ومع ذلك، فإنه ما زال من الضروري معرفة الأنواع الأخرى الموجودة، ولماذا.

أنواع القيمة مقابل المراجع Value vs. reference types

تركز هذه الوحدة على نوعين من الأنواع في C# أنواع المراجع، وأنواع القيم.

تقوم متغيرات أنواع المراجع `reference types` بتخزين مراجع إلى بياناتها (الكائنات objects) أي أنها تشير إلى قيم البيانات المخزنة في مكان آخر، وبالمقارنة، تحتوي متغيرات أنواع القيم `data values` على بياناتها مباشرة، بينما تتعلم المزيد عن C# تظهر تفاصيل جديدة تتعلق بالفرق الأساسي بين أنواع القيم، والمراجع.

أنواع القيم البسيطة Simple value types

أنواع القيم البسيطة هي مجموعة من الأنواع المعرّفة مسبقًا التي توفرها C# ككلمات أساسية keywords هذه الكلمات الأساسية هي أسماء مستعارة (كنية) للأنواع المعرفة مسبقاً، المحددة في مكتبة فئات .NET. على سبيل المثال، الكلمة الأساسية **int** هي اسم مستعار لنوع قيمة معرف في مكتبة فئات .NET. ك `System.Int32`

تتضمن أنواع القيم البسيطة العديد من أنواع البيانات، التي ربما استخدمتها بالفعل مثل `bool` and `char` هناك أيضاً العديد من أنواع القيم المتكاملة integral والفاصلة العائمة floating-point لتمثيل مجموعة واسعة من الأرقام الكاملة، والكسرية.

خلاصة

- **يتم تخزين القيم على أنها بتات bits** التي هي مفاتيح تشغيل / إيقاف بسيطة، يسمح لك الجمع بين ما يكفي من هذه المفاتيح بتخزين أي قيمة محتملة.
- **هناك فئتان أساسيتان لأنواع البيانات:** أنواع القيمة value types والمراجع reference types الفرق هو في كيفية، ومكان تخزين القيم، بواسطة الكمبيوتر أثناء تطبيق البرنامج.
- تستخدم أنواع القيم البسيطة Simple value اسماً مستعاراً للكلمة الأساسية keyword لتمثيل الأسماء الرسمية للأنواع formal names of types في مكتبة .NET.

اختبر معلوماتك

ما هو الخيار الأفضل الذي يمثل كيفية تمثيل bits وإيقاف تشغيلها؟

- يتم تمثيل وحدات البت باستخدام الحرف "x" أو"o"
- يتم تمثيل وحدات البت باستخدام رمز "bit+" أو "bit-"
- وحدات البت ثنائية ويتم تمثيلها باستخدام رقم "1" أو "0"

اختبر إجابتك

وحدات البت ثنائية ويتم تمثيلها باستخدام رقم "1" أو "0"

صحيح يتم تمثيل وحدات **Bits** باستخدام الرقم "1" أو "0"

٣ تمرين - اكتشاف الأنواع المتكاملة integral types

في هذا التمرين، يمكنك العمل مع أنواع متكاملة، النوع المتكامل **integral type** هو نوع قيمة بسيط، يمثل أرقاماً كاملة بدون كسر، مثل -1، 0، 1، 2، 3 الأكثر شيوعاً في هذه الفئة هو نوع البيانات **int**

هناك فئتان فرعيتان من الأنواع المتكاملة: **أنواع متكاملة موقعة signed** وغير **موقعة unsigned**

يستخدم النوع الموقع **signed** وحدات bytes الخاصة به لتمثيل عدد متساو من الأرقام الموجبة والسالبة، يمنحك التمرين التالي التعرض للأنواع المتكاملة الموقعة في#C

إعداد بيئة الترميز

١. فتح Visual Studio Code.

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة **File** Visual Studio Code حدد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** Visual Studio Code حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد، للمجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. في موجه الأوامرTerminal لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

يستخدم أمر .NET CLI. هذا قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد. ينشئ الأمر مجلدات CsharpProjects, TestProject نيابة عنك، ويستخدم TestProject كاسم للملف csproj.

٧. في لوحة EXPLORER قم **بتوسيع المجلد CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في لوحة EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**

٩. حذف أسطر التعليمات البرمجية الموجودة.

أنت تستخدم مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق لوحة Terminal

استخدام الخاصيتين MinValue and MaxValue لكل نوع متكامل موقّع signed integral type

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في لوحة المحرر.

يجب أن يكون Program.cs فارغاً، إذا لم يكن كذلك، فحدد جميع أسطر التعليمات البرمجية واحذفها.

٢. لمشاهدة نطاقات القيمة (الأقصى والأدنى) لأنواع البيانات المختلفة، اكتب التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
Console.WriteLine("Signed integral types:");

Console.WriteLine($"sbyte  : {sbyte.MinValue} to {sbyte.MaxValue}");
Console.WriteLine($"short  : {short.MinValue} to {short.MaxValue}");
Console.WriteLine($"int    : {int.MinValue} to {int.MaxValue}");
Console.WriteLine($"long   : {long.MinValue} to {long.MaxValue}");
```

١. في قائمة **File** حددVisual Studio Code **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٢. في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٣. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

ينبغي أن تشاهد الإخراج التالي:

```
Signed integral types:
sbyte  : -128 to 127
short  : -32768 to 32767
int    : -2147483648 to 2147483647
long   : -9223372036854775808 to
9223372036854775807
```

بالنسبة لمعظم التطبيقات غير العلمية، من المحتمل أن تحتاج فقط إلى العمل مع int في معظم الوقت، لن تحتاج إلى أكثر من ٢,١٤ مليار عدد صحيح، في نطاق يتراوح من موجب إلى سالب.

الأنواع المتكاملة غير الموقعة Unsigned integral types

يستخدم النوع غير الموقع، وحدات bytes الخاصة به لتمثيل الأرقام الموجبة فقط، يقدم باقي التمرين الأنواع المتكاملة غير الموقعة في C#

استخدام الخاصيتين MinValue and MaxValue لكل نوع متكامل غير موقع

١. أسفل مقطع التعليمات البرمجية السابقة، أضف التعليمات البرمجية التالية:

```
Console.WriteLine("");
Console.WriteLine("Unsigned integral types:");

Console.WriteLine($"byte   : {byte.MinValue} to
{byte.MaxValue}");
Console.WriteLine($"ushort : {ushort.MinValue}
to {ushort.MaxValue}");
Console.WriteLine($"uint   : {uint.MinValue} to
{uint.MaxValue}");
Console.WriteLine($"ulong  : {ulong.MinValue}
to {ulong.MaxValue}");
```

٢. احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيل التعليمات البرمجية.

ينبغي أن تشاهد الإخراج التالي:

```Output
Signed integral types:
sbyte  : -128 to 127
short  : -32768 to 32767
int    : -2147483648 to 2147483647
long   : -9223372036854775808 to
9223372036854775807

Unsigned integral types:
byte   : 0 to 255
ushort : 0 to 65535
uint   : 0 to 4294967295
ulong  : 0 to 18446744073709551615
```

بينما يمكن استخدام نوع بيانات معين للعديد من الحالات، نظراً لأن نوع البيانات byte يمكن أن يمثل قيمة من 0 إلى 255 فمن الواضح أن الغرض من ذلك هو الاحتفاظ بقيمة تمثل byte من البيانات. البيانات المخزنة في الملفات أو البيانات المنقولة عبر الإنترنت غالبًا ما تكون في شكل ثنائي، عند العمل مع البيانات من هذه المصادر الخارجية، تحتاج إلى تلقي البيانات كمصفوفة من وحدات bytes ثم تحويلها إلى سلاسل strings تتطلب العديد من الأساليب في مكتبة فئات .NET. التي تتعامل مع بيانات الترميز وفك التشفير معالجة مصفوفات البايت byte arrays

خلاصة

- النوع المتكامل integral type هو نوع بيانات قيمة بسيط يمكن أن يحتوي على أرقام كاملة.
- هناك أنواع بيانات رقمية موقعة وغير موقعة. تستخدم الأنواع Signed integral المتكاملة الموقعة 1 bit لتخزين ما إذا كانت القيمة موجبة أو سالبة.
- يمكنك استخدام خصائص البيانات الرقمية MaxValue and MinValue لتقييم ما إذا كان يمكن احتواء رقم في نوع بيانات معين.

اختبر معلوماتك

ما نوع البيانات الذي يحتوي على نطاق من الأرقام الكاملة بين 0 و**65535** في التعليمات البرمجية C#

- Sbyte
- Ushort
- byte

اختبر إجابتك

Ushort

صحيح، يمثل ushort القيم من 0 إلى **65535**

٤ تمرين ـ اكتشاف أنواع الفاصلة العائمة floating-point types

في هذا التمرين، يمكنك العمل مع أنواع بيانات الفاصلة العائمة **floating-point** للتعرف على الاختلافات الدقيقة بين كل نوع من أنواع البيانات.

الفاصلة العائمة هي نوع قيمة بسيط يمثل الأرقام على يمين المكان العشري، بعكس الأعداد المتكاملة، هناك اعتبارات أخرى تتجاوز القيم القصوى والدنيا، التي يمكنك تخزينها في نوع الفاصلة العائمة المعطى.

تقييم أنواع الفاصلة العائمة floating-point

أولا، يجب مراعاة دقة أو ضبط الأرقام the digits of precision التي يسمح بها كل نوع، الدقة هي عدد أماكن القيمة المخزنة بعد الفاصلة العشرية.

ثانياً، يجب مراعاة الطريقة التي يتم بها تخزين القيم، والتأثير على دقة القيمة، على سبيل المثال، يتم تخزين القيم float and double داخلياً بتنسيق ثنائي (base 2) بينما decimal يتم تخزينها بتنسيق عشري (base10) لماذا هذا أمر مهم؟

يمكن أن يؤدي إجراء الرياضيات على قيم الفاصلة العائمة الثنائية، إلى نتائج قد تفاجئك إذا كنت معتادًا على الرياضيات العشرية (base10) في كثير من الأحيان، رياضيات الفاصلة العائمة الثنائية هو تقريب للقيمة الحقيقية، لذلك، float and double مفيدة لأنه يمكن تخزين أعداد كبيرة، باستخدام بصمة ذاكرة صغيرة، ومع ذلك، يجب استخدام float and double فقط عندما يكون التقريب مفيداً. على سبيل المثال، تكون عدة آلاف عند حساب مبعثر كرة الثلج في لعبة فيديو قريبة بما فيه الكفاية.

عندما تحتاج إلى إجابة أكثر دقة، يجب عليك استخدام decimal كل قيمة من النوع decimal لها بصمة ذاكرة كبيرة نسبياً، ولكن تنفيذ العمليات الرياضية يمنحك نتيجة أكثر دقة. لذلك، يجب عليك استخدام decimal عند

العمل مع البيانات المالية، أو أي سيناريو تحتاج فيه إلى نتيجة دقيقة من عملية حسابية.

استخدام الخاصيتين MinValue and MaxValue لكل نوع عائم موقع signed float type

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق على جميع التعليمات البرمجية من التدريبات السابقة.

٢. لمشاهدة نطاقات القيم، لأنواع البيانات المختلفة، قم بتحديث التعليمات البرمجية في محرر التعليمات البرمجية Visual Studio كما يلي:

```
Console.WriteLine("");
Console.WriteLine("Floating point types:");
Console.WriteLine($"float  : {float.MinValue}
to {float.MaxValue} (with ~6-9 digits of
precision)");
Console.WriteLine($"double : {double.MinValue}
to {double.MaxValue} (with ~15-17 digits of
precision)");
Console.WriteLine($"decimal: {decimal.MinValue}
to {decimal.MaxValue} (with 28-29 digits of
precision)");
```

٣. في قائمة **File** Visual Studio Code حدد **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٤. في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال: `C:\Users\someuser\Desktop\csharpprojects\TestProject>`

ينبغي أن تشاهد الإخراج التالي:

```
Floating point types:
float  : -3.402823E+38 to 3.402823E+38 (with
~6-9 digits of precision)
double : -1.79769313486232E+308 to
1.79769313486232E+308 (with ~15-17 digits of
precision)
decimal: -79228162514264337593543950335 to
79228162514264337593543950335 (with 28-29
digits of precision)
```

كما ترى، `float` and `double` واستخدم رمزاً مختلفاً عن الذي يستخدمه `decimal` لتمثيل أكبر وأصغر قيمه الممكنة. ولكن ما معنى الرموز؟

فك رموز قيم الفاصلة العائمة الكبيرة large floating-point

لأن أنواع الفاصلة العائمة، يمكنها الاحتفاظ بأعداد كبيرة بدقة، يمكن تمثيل قيمها باستخدام رمز "E notation" وهو شكل من أشكال التدوين العلمي، الذي يعني "ضرب 10 مرفوعة إلى قوة" _ضرب اس 10_ سوف تكون القيمة مثل 5E+2 القيمة 500 لأنها تعادل $5 * 10^2$, or 5×10^2

خلاصة

- الفاصلة العائمة floating-point هو نوع بيانات بسيط القيمة، يمكنه الاحتفاظ بأرقام كسرية.
- يتطلب اختيار نوع الفاصلة العائمة المناسب لتطبيقك، فهم أعمق، أن تفكر في أكثر من مجرد القيم القصوى والدنيا التي يمكنه الاحتفاظ بها، يجب أيضًا مراعاة عدد القيم التي يمكن الاحتفاظ بها بعد الفاصلة العشرية، وكيفية تخزين الأرقام، وكيف يؤثر تخزينها الداخلي على نتائج العمليات الحسابية.
- يمكن أحياناً تمثيل قيم النقطة العائمة باستخدام الرمز E عندما تكون الأرقام كبيرة جداً.
- هناك اختلاف أساسي في كيفية تعامل المحول البرمجي ووقت التشغيل مع decimal بدلاً من float or double خاصة عند تحديد مقدار الدقة الضرورية من العمليات الرياضية.

اختبر معلوماتك

ما هو أفضل نوع بيانات يمكن استخدامه لمعالجة الودائع المصرفية؟

- Decimal
- Float
- Double

اختبر إجابتك

Decimal

Decimal بدقة تصل إلى ٢٨-٢٩ رقمًا، يتمتع بالدقة المطلوبة للعديد من التطبيقات المالية

٥ تمرين - اكتشاف أنواع المراجع reference types

تتضمن الأنواع المرجعية مصفوفات _قوائم مرتبة_ arrays وفئات classes وسلاسل strings يتم التعامل مع أنواع المراجع reference types بشكل مختلف عن أنواع القيم value types فيما يتعلق بالطريقة التي يتم بها تخزين القيم عند تنفيذ التطبيق.

في هذا التمرين، ستتعرف على اختلاف أنواع المراجع عن أنواع القيم، وكيفية استخدام عامل التشغيل new لإقران متغير بقيمة، في ذاكرة الكمبيوتر.

كيف تختلف الأنواع المرجعية عن أنواع القيم

يقوم متغير نوع القيمة value type variable بتخزين قيمه مباشرة في منطقة تخزين، تسمى المكدس `The stack` التكدس: هو الذاكرة المخصصة للتعليمات البرمجية التي تعمل حاليًا على وحدة المعالجة المركزية (تعرف أيضًا بإطار التكدس أو إطار التنشيط) عند انتهاء إطار التكدس من التنفيذ، تتم إزالة القيم الموجودة في التكدس.

يقوم متغير نوع المرجع reference type variable بتخزين قيمه في منطقة ذاكرة منفصلة، تسمى كومة الذاكرة المؤقتة The heap كومة الذاكرة المؤقتة: هي منطقة ذاكرة مشتركة عبر العديد من التطبيقات التي تعمل على نظام تشغيل في نفس الوقت، يتصل .NET Runtime. مع نظام التشغيل لتحديد ما هي عناوين الذاكرة المتوفرة، ويطلب عنواناً يمكنه تخزين القيمة فيه، يخزن .NET Runtime. القيمة، ثم يرجع عنوان الذاكرة إلى المتغير. عندما تستخدم تعليماتك البرمجية للمتغير، يبحث .NET Runtime. بسلاسة عن العنوان المخزن في المتغير، ويسترد القيمة المخزنة هناك.

ستكتب بعد ذلك بعض التعليمات البرمجية التي توضح هذه الأفكار، بشكل أكثر وضوحاً.

تعريف متغير نوع المرجع reference type variable

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق على جميع التعليمات البرمجية من التدريبات السابقة.

٢. تحديث التعليمات البرمجية الخاصة بك في المحرر، كما يلي:

```
int[] data;
```

تحدد التعليمات البرمجية السابقة متغيراً، يمكنه الاحتفاظ بقيمة مصفوفة من النوع int

عند هذه النقطة data هو مجرد متغير يمكن أن يحتوي على مرجع، أو بدلاً من ذلك، عنوان ذاكرة لقيمة في كومة الذاكرة المؤقتة the heap لأنه لا يشير إلى عنوان ذاكرة، فإنه يُسمى مرجعًا فارغًا.

٣. إنشاء مثيل مصفوفة int باستخدام الكلمة الأساسية new

قم بتحديث التعليمات البرمجية في المحرر لإنشاء وتعيين مثيل جديد من المصفوفة int باستخدام التعليمات البرمجية التالية:

```
int[] data;
data = new int[3];
```

تُعلم الكلمة الأساسية new وقت التشغيل .NET Runtime بإنشاء مثيل مصفوفة int ثم التنسيق مع نظام التشغيل لتخزين حجم المصفوفة، لثلاث قيم int في الذاكرة. يمتثل وقت تشغيل .NET. ويعيد عنوان الذاكرة الخاص بالمصفوفة الجديدة int أخيرًا، يتم تخزين عنوان الذاكرة في variable data القيمة الافتراضية لعناصر المصفوفة int هي 0 لأن هذه هي القيمة الافتراضية لـ int

٤. تعديل مثال التعليمات البرمجية لتنفيذ كلتا العمليتين في سطر واحد من التعليمات البرمجية.

عادة ما يتم اختصار سطري التعليمات البرمجية في الخطوة السابقة إلى سطر واحد، للإعلان عن المتغير وإنشاء مثيل جديد للمصفوفة int تعديل التعليمات البرمجية من الخطوة ٣ إلى ما يلي:

```
int[] data = new int[3];
```

على الرغم من عدم وجود نتائج لملاحظتها، نأمل أن يضيف هذا التمرين توضيحاً لكيفية ارتباط بناء جملة C# بخطوات عملية للعمل مع الأنواع المرجعية reference types

ما المختلف في نوع بيانات سلسلة string data type

نوع البيانات string هو أيضاً نوع مرجع. قد تتساءل عن سبب عدم استخدام عامل تشغيل new عند الإعلان عن سلسلة. هذا للراحة البحتة التي يوفرها مصممو string نظراً لاستخدام نوع البيانات بشكل متكرر، يمكنك استخدام هذا التنسيق:

```
string shortenedString = "Hello World!";
Console.WriteLine(shortenedString);
```

ومع ذلك، في الخلفية، يتم إنشاء مثيل جديد من System.String وتهيئته إلى "Hello World!"

خلاصة

- يمكن أن تحتوي أنواع القيم value types على قيم أصغر، ويتم تخزينها في التكدس the stack يمكن أن تحتوي أنواع المراجع reference types على قيم كبيرة، ويتم إنشاء مثيل جديد لنوع مرجع باستخدام عامل التشغيل new تحتوي متغيرات نوع المرجع على مراجع (عنوان الذاكرة) للقيمة الفعلية المخزنة في كومة الذاكرة المؤقتة the heap
- تضم أنواع المراجع مصفوفات arrays وسلاسل strings وفئات classes

راجع معلوماتك

في C# عند استخدام الكلمة الأساسية new لإنشاء مثيل لفئة class أين يتم تخصيص الذاكرة؟

- Stack
- Heap
- Variable

راجع إجابتك

Heap

صحيح يتم تخصيص مثيلات الفئات في الكومة the heap عند إنشائها باستخدام الكلمة الأساسية new

٦ اختيار نوع البيانات المناسب

لقد تم تعريفك على الفرق بين أنواع القيم value types وأنواع المرجع reference types بالإضافة إلى الأنواع المتكاملة integral والفاصلة العائمة floating point

افترض أن وظيفتك هي إنشاء تطبيق جديد، يقوم باسترداد أنواع مختلفة من البيانات، ومعالجتها وتخزينها، ما هي أنواع البيانات التي تستخدمها؟

في بعض الحالات، هو خيار سهل، على سبيل المثال، عندما تحتاج إلى العمل مع النص، فإنك تستخدم نوع البيانات string افتراضياً، ما لم تكن بحاجة إلى تنفيذ قدر كبير من التسلسل.

ولكن ماذا عن العمل مع البيانات الرقمية؟ هناك ١١ خيارًا مختلفًا، كيف تختار نوع البيانات المناسب؟

اختر نوع البيانات المناسب

مع العديد من أنواع البيانات للاختيار من بينها، ما هي المعايير التي يجب استخدامها، لاختيار نوع البيانات المناسب لحالة معينة؟

عند تقييم الخيارات الخاصة بك، يجب عليك أن تزن عدة اعتبارات هامة، عادة ما لا توجد إجابة واحدة صحيحة، ولكن بعض الإجابات أكثر صحة من غيرها.

اختر نوع البيانات الذي يلبي متطلبات نطاق حدود القيمة لتطبيقك

يمكن أن يساعد اختيارك لنوع البيانات، في تعيين حدود لحجم البيانات التي قد تخزنها في هذا المتغير المحدد، على سبيل المثال، إذا كنت تعرف أن متغيراً معيناً يجب أن يخزن رقما يتراوح بين 1 and 10,000 فقط وإلا فإنه خارج حدود ما يمكن توقعه، فمن المحتمل أن تتجنب byte and sbyte ل وبما أن نطاقاتها منخفضة جداً.

علاوة على ذلك، من المحتمل ألا تحتاج إلى أنواع `int, long, uint, ulong` لأنه يمكنهم تخزين بيانات أكثر مما هو ضروري. وبالمثل، من المحتمل أن تتخطى أنواع `float, double, decimal` إذا لم تكن بحاجة إلى قيم كسرية.

يمكنك تضييقها إلى `short and ushort` والتي قد يكون كلاهما قابلاً للتطبيق، إذا كنت واثقاً من أن القيمة السالبة لن تحتاجها في تطبيقك، فقد تختار `ushort` (عدد صحيح موجب غير موقع، من 0 to 65,535) الآن، أي قيمة تم تعيينها لمتغير من نوع `ushort` خارج حدود 0 to 65,535 ستطرح استثناء، وبالتالي تساعدك بحرفية على فرض درجة من التحقق من السلامة في تطبيقك.

ابدأ باختيار نوع البيانات المناسب لاحتواء البيانات (وليس لتحسين الأداء)

قد تميل إلى اختيار نوع البيانات الذي يستخدم أقل وحدات bits لتخزين البيانات، مع التفكير في أنه يحسن أداء التطبيق، ومع ذلك، فإن بعض أفضل النصائح المتعلقة بأداء التطبيق (أي مدى سرعة تشغيل التطبيق) هو عدم "التحسين المبكر، قبل الأوان" يجب مقاومة إغراء تخمين أجزاء من تعليماتك البرمجية، بما في ذلك تحديد أنواع البيانات التي قد تؤثر على أداء تطبيقك.

يفترض العديد أنه بسبب قيام نوع بيانات ما، بتخزين معلومات أقل، فلا بد أنه يستخدم معالج كمبيوتر وذاكرة أقل، من نوع البيانات الذي يخزن معلومات أكثر، وبدلاً من ذلك، يجب عليك توفيق الملاءمة المناسبة لبياناتك، ثم يمكنك لاحقًا قياس أداء التطبيق، تجريبيًا باستخدام برامج خاصة، توفر رؤى واقعية لأجزاء التطبيق التي تؤثر سلبًا على الأداء.

اختيار أنواع البيانات بناءً على أنواع بيانات الإدخال والإخراج لوظائف المكتبة المستخدمة

افترض أنك تريد العمل مع فترة من السنوات بين تاريخين، نظراً لأن التطبيق هو تطبيق عمل، فقد تحدد أنك تحتاج فقط إلى نطاق من حوالي ١٩٦٠ إلى ٢٢٠٠ قد تفكر في محاولة العمل مع byte لأنه يمكن أن يمثل أرقاماً بين 0 - 255

ومع ذلك، عندما تنظر إلى الأساليب المضمنة في فئتي System.TimeSpan and System.DateTime ستدرك أنها تقبل في الغالب قيمًا من النوع int and double إذا اخترت sbyte فسوف تتنقل باستمرار ذهابًا وإيابًا بين sbyte وdouble أو int. في هذه الحالة، قد يكون من المنطقي أكثر اختيار int إذا كنت لا تحتاج إلى دقة دون الثانية، double إذا كنت بحاجة إلى دقة دون الثانية.

اختيار أنواع البيانات استناداً إلى التأثير على الأنظمة الأخرى

في بعض الأحيان، يجب عليك مراعاة كيفية استهلاك المعلومات، بواسطة تطبيقات أخرى، أو أنظمة أخرى مثل قاعدة بيانات، على سبيل المثال، يختلف نظام الكتابة في SQL Serverعن نظام من نوع C# وكنتيجة لذلك، يجب أن يحدث بعض التخطيط بين الاثنين، قبل أن تتمكن من حفظ البيانات في قاعدة البيانات تلك.

إذا كان الغرض من التطبيق الخاص بك هو الواجهة مع قاعدة بيانات، فمن المحتمل أن تحتاج إلى النظر في كيفية تخزين البيانات، ومقدار البيانات المخزنة، قد يؤثر اختيار نوع بيانات أكبر على مقدار (وتكلفة) التخزين الفعلي، المطلوب لتخزين جميع البيانات التي سينشئها تطبيقك.

عندما تكون في حيرة، التزم بالأساسيات

على الرغم من أنك نظرت إلى العديد من الاعتبارات، عند البدء، فمن أجل التبسيط، يجب أن تفضل مجموعة فرعية من أنواع البيانات الأساسية، بما في ذلك:

- int لمعظم الأرقام الصحيحة
- decimal للأرقام التي تمثل المال
- bool للقيم الحقيقية أو الخاطئة
- string للقيمة الأبجدية الرقمية

اختيار أنواع معقدة، متخصصة للحالات الخاصة

لا تقم بإعادة اختراع أنواع البيانات، إذا كان نوع بيانات واحد أو أكثر موجوداً بالفعل لغرض معين، تحدد الأمثلة التالية أين يمكن أن تكون أنواع بيانات .NET معينة مفيدة:

- byte العمل مع البيانات المشفرة التي تأتي من أنظمة كمبيوتر أخرى، أو باستخدام مجموعات أحرف مختلفة.
- double العمل مع أغراض هندسية أو علمية، يستخدم double بشكل متكرر عند إنشاء ألعاب تتضمن الحركة.
- System.DateTime لقيمة تاريخ ووقت محددة.
- System.TimeSpan لمدة سنوات / أشهر / أيام / ساعات / دقائق / ثوان / مللي ثانية.

خلاصة

هناك اعتبارات عند اختيار أنواع البيانات للتعليمات البرمجية، وغالباً ما يكون أكثر من خيار واحد، فكر في اختياراتك، وما لم يكن لديك سبب وجيه، فحاول الالتزام بالأنواع الأساسية مثل `int`, `decimal`, `string`, and `bool`

٧ اختبر معلوماتك

١- يجب أن يخزن جزء من التعليمات البرمجية القيم الرقمية الكاملة بين القيم السالبة والإيجابية 1,000,000 ما هو أفضل نوع بيانات يمكن استخدامه؟

- float
- double
- int

٢- تحتاج التعليمات البرمجية للعبة، إلى تخزين الكثير من القيم الكسرية التي تمثل مواضع `x, y, z` في مساحة كبيرة ثلاثية الأبعاد، من المحتمل أن تحتاج إلى إجراء حسابات الفيزياء للمسارات وما إلى ذلك، الدقة المطلقة غير مطلوبة، ولكن من المهم أن يعمل البرنامج بأكبر قدر ممكن من الكفاءة، ما نوع البيانات الأفضل؟

- float
- decimal
- int

٣- أي من العبارات التالية صحيحة؟

- يتم تخزين أنواع المراجع في التكدس.
- يمكن لأنواع القيم تخزين الأرقام فقط.
- استخدم عامل التشغيل new لإنشاء مثيل جديد من نوع مرجع، وإرجاع عنوانه إلى المتغير.

راجع إجابتك

١ int

صحيح على الرغم من أن float, double قد يعملان، إلا أن دقتهما الإضافية بعد العلامة العشرية لا تتناسب مع المتطلبات الدقيقة لهذا السيناريو

٢ float

صحيح على عكس int يمكن float تخزين أرقام كسرية، وعلى عكس decimal لا تطلب float ذاكرة إضافية لتخزين دقة إضافية غير مطلوبة في هذا السيناريو

٣ استخدم عامل التشغيل new لإنشاء مثيل جديد من نوع مرجع، وإرجاع عنوانه إلى المتغير

صحيح استخدم عامل التشغيل new لإنشاء مثيل جديد من نوع مرجع وإرجاع عنوانه إلى المتغير

٨ الملخص

اختيار نوع البيانات المناسب لتطبيقك، مهارة برمجة حيوية. كان هدفك هو فهم الاختلافات بين كل نوع بيانات، لاتخاذ قرار مستنير بشأن أنواع البيانات في تعليماتك البرمجية.

لقد بدأت في إنشاء نموذج عقلي لكيفية تخزين البيانات، أثناء تنفيذ التطبيق. لديك خبرة في العمل مع أنواع بيانات متكاملة new integral والفاصلة العائمة floating point لديك معايير يمكنك استخدامها عند تحديد أنواع البيانات، التي سيتم استخدامها في التعليمات البرمجية.

وأخيراً، تفهم سبب الحاجة إلى استخدام الكلمة الأساسية new عند إنشاء مثيلات لأنواع المراجع.

الوحدة الثانية

تحويل أنواع البيانات باستخدام تقنيات الإرسال والتحويل casting and conversion

تحكم في البيانات الموجودة في تطبيقاتك، وتعرف متى يجب تطبيق الأسلوب الصحيح لتغيير أنواع البيانات حسب الحاجة.

الأهداف التعليمية

- استخدام عامل التشغيل casting لتحويل قيمة إلى نوع بيانات مختلف.
- استخدام أساليب التحويل conversion methods لتحويل قيمة إلى نوع بيانات مختلف.
- الحماية من فقدان البيانات عند إجراء عملية إرسال cast أو تحويل conversion
- استخدم أسلوب TryParse() لتحويل سلسلة نصية بأمان إلى نوع بيانات رقمي.

محتويات الوحدة:

١ المقدمة

لنفترض أنك مطور برامج في فريق يعمل على نموذج الاستيعاب الطبي، أنت مكلف بتسليم ميزات التطبيق لجمع البيانات التي أدخلها فني طبي قبل أن يرى الطبيب المريض، يمكن للفني استخدام التطبيق لتسجيل التاريخ والوقت وعمر المريض والطول والوزن والذبذبات وضغط الدم، كما يوفر التطبيق حقولاً نصية للحصول على معلومات أخرى، مثل سبب الزيارة والوصفات الطبية الحالية وعناصر أخرى، يمكنك العمل مع العديد من البيانات الموجودة في مزيج من أنواع البيانات، بالنسبة إلى النموذج الأولي، ستقوم بإنشاء تطبيق وحدة تحكم وجمع جميع الإدخالات ك strings

نظرا لأن الإدخال هو من البداية كسلسلة، تحتاج أحياناً إلى تغيير القيم من نوع بيانات إلى آخر في التعليمات البرمجية، مثال بسيط: أي عملية رياضية تريد تنفيذها مع بيانات السلسلة، ربما تحتاج أولاً إلى تغيير القيمة إلى نوع بيانات رقمية، مثل int ومن ثم يمكنك معالجة العملية، بدلاً من ذلك، قد تحتاج إلى تنسيق وإخراج قيمة رقمية، لتقرير ملخص باستخدام دمج النصوص أو السلسلة string interpolation

يمكنك استخدام تقنيات مختلفة لتغيير نوع البيانات عند الضرورة، ستتعلم متى تستخدم تقنية على أخرى، ومتى قد تخاطر تقنية معينة بفقدان البيانات.

في نهاية هذه الوحدة، يمكنك التحكم في البيانات في تطبيقاتك، مع معرفة متى يتم استخدام التقنية المناسبة، لتغيير أنواع البيانات حسب الحاجة.

٢ تمرين ـ استكشاف ارسال casting نوع البيانات وتحويلها conversion

هناك تقنيات متعددة لإجراء تحويل نوع البيانات، تعتمد التقنية التي تختارها على إجابتك على سؤالين مهمين:

- هل من الممكن، اعتماداً على القيمة، أن تؤدي محاولة تغيير نوع بيانات القيمة إلى طرح استثناء في وقت التشغيل؟
- هل من الممكن، اعتماداً على القيمة، أن تؤدي محاولة تغيير نوع بيانات القيمة إلى فقدان المعلومات؟

في هذا التمرين، يمكنك العمل في طريقك من خلال هذه الأسئلة، والآثار المترتبة على إجاباتها، والتقنية المناسبة التي يجب عليك استخدامها عندما تحتاج إلى تغيير نوع البيانات.

إعداد بيئة الترميز

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** لـ Visual Studio Code حدد فتح مجلد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** في Visual Studio Code حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب في موجه الأوامر Terminal: **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

يستخدم أمر .NET CLI. هذا قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد. ينشئ الأمر نيابة عنك مجلدات CsharpProjects, TestProject ويستخدم TestProject كاسم للملف csproj. أو كامتداد له.

٧. في قائمة استكشاف EXPLORER قم **بتوسيع المجلد CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**.

٩. حذف أسطر التعليمات البرمجية الموجودة.

يمكنك استخدام مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق Terminal

السؤال: هل من الممكن أن تؤدي محاولة تغيير نوع بيانات القيمة إلى حدوث استثناء في وقت التشغيل؟

يحاول المحول البرمجي C# استيعاب التعليمات البرمجية الخاصة بك، ولكنه لا يقوم بتجميع العمليات التي قد تؤدي إلى استثناء، عندما تفهم الشاغل الأساسي للمحول البرمجي C# يكون فهم سبب عمله بطريقة معينة أسهل.

كتابة التعليمات البرمجية التي تحاول إضافة int and a string وحفظ النتيجة في int

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في لوحة المحرر.

يجب أن يكون Program.cs فارغ، إذا لم يكن كذلك، فحدد كافة أسطر التعليمات البرمجية واحذفها.

٢. اكتب التعليمات البرمجية التالية في محرر:

```
int first = 2;
string second = "4";
int result = first + second;
Console.WriteLine(result);
```

هنا، تحاول إضافة القيم 4 - 2 القيمة 4 هي من النوع string هل سينجح هذا؟

٣. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٥. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
C:\Users\someuser\Desktop\csharpprojects\TestProje
ct\Program.cs(3,14): error CS0029: Cannot
implicitly convert type 'string' to 'int'
```

٦. خذ دقيقة للنظر في سبب عدم تمكن المحول البرمجي من تشغيل نموذج التعليمات البرمجية الأول.

يخبرك الجزء المهم من رسالة الخطأ **`(3,14): error CS0029: Cannot implicitly convert type 'string' to 'int'`** أن المشكلة في استخدام نوع البيانات string

ولكن لماذا لا يمكن للمحول البرمجي C# التعامل مع الخطأ؟ بعد كل شيء، يمكنك القيام بالعكس لربط رقم إلى string وحفظه في متغير string variable هنا، يمكنك تغيير نوع بيانات المتغير resultint من int إلى string

٧. تحديث تعليماتك البرمجية، في محرر Visual Studio كما يلي:

```
int first = 2;
string second = "4";
string result = first + second;
Console.WriteLine(result);
```

٨. احفظ ملف التعليمات البرمجية، ثم شغلها.

يجب عليك مراقبة الإخراج التالي:

```
24
```

الإخراج غير صحيح رياضياً، ولكنه يكتمل عن طريق دمج القيم كحرفين "2" , "4"

٩. افحص، مرة أخرى، مثال التعليمات البرمجية الأول حيث المتغير result من النوع int التعليمات مع رسالة الخطأ.

```
int first = 2;
string second = "4";
int result = first + second;
Console.WriteLine(result);
```

لماذا لا يمكن للمحول البرمجي C# معرفة أنك تريد التعامل مع المتغير second الذي يحتوي على 4 كرقم، وليس string

تقوم المحولات البرمجية بإجراء تحويلات آمنة

يرى برنامج التحويل البرمجي The compiler C# مشكلة محتملة في عملية الإنشاء، المتغيّر second من النوع string لذا يمكن تعيين له قيمة مختلفة مثل "hello" إذا حاول المحول البرمجي C# تحويل "hello" إلى رقم، فقد يؤدي إلى حدوث استثناء في وقت التشغيل، لتجنب هذه الإمكانية، لا يقوم المحول البرمجي C# ضمنياً بإجراء التحويل من string إلى int نيابة عنك.

من منظور المحول البرمجي C# ستكون العملية أكثر أمانًا لتحويل int إلى string وتنفيذ تسلسل نصي بدلاً من ذلك.

إذا كنت تنوي إجراء إضافة باستخدام سلسلة، فإن المحول البرمجي C# يطلب منك التحكم بشكل أكثر وضوحًا في عملية تحويل البيانات، وبعبارة أخرى، فإنه يجبرك على المشاركة بشكل أكبر، لتتمكن من وضع الاحتياطات اللازمة، للتعامل مع إمكانية أن يؤدي التحويل إلى حدوث استثناء.

إذا كنت بحاجة إلى تغيير قيمة من نوع البيانات الأصلي إلى نوع بيانات جديد، ويمكن أن ينتج عن التغيير استثناء في وقت التشغيل، يجب إجراء **تحويل بيانات data conversion**

لإجراء تحويل البيانات، يمكنك استخدام إحدى التقنيات المتعددة:

- استخدام أسلوب مساعد على نوع البيانات.
- استخدام أسلوب مساعد على المتغير.
- استخدام أساليب الفئة Convert

يمكنك إلقاء نظرة على بعض الأمثلة على هذه التقنيات لتحويل البيانات لاحقاً، في هذه الوحدة.

السؤال: هل من الممكن أن تؤدي محاولة تغيير نوع بيانات القيمة إلى فقدان المعلومات؟

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
int myInt = 3;
Console.WriteLine($"int: {myInt}");

decimal myDecimal = myInt;
Console.WriteLine($"decimal: {myDecimal}");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد المخرج التالي:

```
int: 3
decimal: 3
```

مفتاح هذا المثال هو هذا السطر من التعليمات البرمجية:

```
decimal myDecimal = myInt;
```

حيث إن أي قيمة int يمكن بسهولة احتواؤها داخل decimal ويعمل المحول البرمجي على إجراء التحويل.

يعني مصطلح تحويل توسيعي widening conversion أنك تحاول تحويل قيمة من نوع بيانات قد يحتوي على معلومات أقل، إلى نوع بيانات يمكن أن يحتوي على المزيد من المعلومات، في هذه الحالة، القيمة المخزنة في متغير من النوع int الذي تم تحويله إلى متغير من النوع decimal لا تفقد المعلومات.

عندما تعرف أنك تقوم بإجراء تحويل موسع widening conversion يمكنك الاعتماد على التحويل الضمني implicit conversion يعالج المحول البرمجي التحويلات الضمنية.

تنفيذ تحويل إرسال Perform a cast

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
decimal myDecimal = 3.14m;
Console.WriteLine($"decimal: {myDecimal}");

int myInt = (int)myDecimal;
Console.WriteLine($"int: {myInt}");
```

لإجراء تحويل، يمكنك استخدام عامل تشغيل التحويل () لإحاطة نوع البيانات، ثم وضعه بجوار المتغير الذي تريد تحويلهن مثال: (int)myDecimal تقوم بإجراء تحويل صريح explicit conversion إلى نوع بيانات التحويل المحدد (int)

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
decimal: 3.14
int: 3
```

مفتاح هذا المثال هو هذا السطر من التعليمات البرمجية:

```
int myInt = (int)myDecimal;
```

يحتوي المتغير myDecimal على قيمة كسور بعد الفاصلة العشرية، بإضافة تعليمات التحويل the casting instruction أو الإرسال إلى (int) فأنت تخبر المحول البرمجي C# بأنك تفهم أنه من الممكن أن تفقد هذه الدقة، وفي هذه الحالة، لا بأس، أنت تخبر المحول البرمجي أنك تقوم بإجراء تحويل مقصود، وهو تحويل صريح explicit conversion

تحديد ما إذا كان التحويل "تحويلاً موسعاً" أو "تحويلاً ضيقاً"

يعني مصطلح تضييق التحويل narrowing conversion أنك تحاول تحويل قيمة من نوع بيانات يمكنه الاحتفاظ بمزيد من المعلومات، إلى نوع بيانات يمكن أن يحتوي على معلومات أقل، في هذه الحالة، قد تفقد معلومات مثل الدقة (الكسور، أي عدد القيم بعد الفاصلة العشرية) مثال على ذلك: تحويل القيمة المخزنة في متغير من النوع decimal إلى متغير من النوع int إذا قمت بطباعة القيمتين، فقد تلاحظ فقدان المعلومات.

عندما تعرف أنك تقوم بإجراء تحويل تضييقي، تحتاج إلى إجراء a cast تحويل إرسال، Casting هو تعليمات إلى المحول البرمجي C# تعلم فيها أن الدقة قد تفقد، لكنك على استعداد لقبولها.

إذا لم تكن متأكداً من فقدان البيانات في التحويل، فاكتب التعليمات البرمجية لإجراء تحويل بطريقتين مختلفتين، ومراقبة التغييرات، يجري المطورون اختبارات صغيرة بشكل متكرر، لفهم السلوكيات بشكل أفضل، كما هو موضح في العينة التالية:

```
decimal myDecimal = 1.23456789m;
float myFloat = (float)myDecimal;

Console.WriteLine($"Decimal: {myDecimal}");
Console.WriteLine($"Float  : {myFloat}");
```

احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
Decimal: 1.23456789
Float:   1.234568
```

يمكنك ملاحظة من الإخراج أن that casting وهو تحويل بإرسال decimal إلى float يعد تحويلاً ضيقاً لأنك تفقد الدقة.

إجراء تحويلات البيانات

في وقت سابق، ذكر أن تغيير القيمة من نوع بيانات إلى آخر قد يتسبب في استثناء وقت التشغيل، ويجب عليك إجراء تحويل البيانات، بالنسبة إلى تحويلات البيانات، هناك ثلاث تقنيات يمكنك استخدامها:

- استخدام أسلوب مساعد helper method على المتغير variable
- استخدام أسلوب مساعد helper method على نوع البيانات data type
- استخدام أساليب الفئة Convert

استخدم ToString() لتحويل رقم إلى string

كل متغير من نوع البيانات يحتوي على وظيفة أسلوب ToString() يعتمد ما يفعله أسلوب ToString() على كيفية تنفيذه على نوع معين، ومع ذلك، بشكل مبدئي، هو ينفذ تحويل توسيع widening conversion

على الرغم من أن هذا ليس ضرورياً تماماً (نظراً لأنه يمكنك الاعتماد على التحويل الضمني implicit conversion في معظم الحالات) يمكن التواصل مع المطورين الآخرين، بأنك تفهم ما تفعله، وأنه مقصود.

فيما يلي مثال سريع على استخدام الأسلوب ToString() لتحويل القيم int إلى قيم string بشكل صريح.

احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
int first = 5;
int second = 7;
string message = first.ToString() +
second.ToString();
Console.WriteLine(message);
```

احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن يعرض الإخراج تسلسلاً للقيمتين:

```
57
```

تحويل string إلى int باستخدام الأسلوب المساعد Parse()

تحتوي معظم أنواع البيانات الرقمية على الأسلوب Parse() والذي يحول string إلى نوع بيانات محدد، في هذه الحالة، يمكنك استخدام الأسلوب Parse() لتحويل two strings إلى قيم int ثم إضافتهما معاً.

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
string first = "5";
string second = "7";
int sum = int.Parse(first) + int.Parse(second);
Console.WriteLine(sum);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن يعرض الإخراج مجموع القيمتين:

```
12
```

خذ دقيقة لمحاولة اكتشاف المشكلة المحتملة مع مثال التعليمات البرمجية السابق؟ ماذا لو تم تعيين أي من المتغيرات first or second إلى قيم لا يمكن تحويلها إلى int؟ يحدث استثناء وقت التشغيل، يتوقع منك المحول البرمجي C# ووقت التشغيل التخطيط مسبقاً، ومنع التحويلات الخاطئة "غير القانونية" يمكنك التخفيف من استثناء وقت التشغيل بعدة خطوات.

أسهل طريقة للتخفيف من هذه المواقف، هي باستخدام الأسلوب TryParse() وهو إصدار أفضل من الأسلوب Parse()

تحويل string إلى int باستخدام فئة التحويل Convert class

تحتوي الفئة Convert على العديد من الأساليب المساعدة، لتحويل قيمة من نوع إلى آخر، في مثال التعليمات البرمجية التالي، يمكنك تحويل عدة نصوص couple of strings إلى قيم من النوع int

ملاحظة

تم تصميم نماذج التعليمات البرمجية في هذا التمرين حسب إعدادات الثقافة في الولايات المتحدة، وتستخدم نقطة (.) كفاصل عشري. قد يؤدي إنشاء التعليمات البرمجية وتشغيلها باستخدام إعداد ثقافة يستخدم فواصل عشرية مختلفة (مثل الفاصلة ,) إلى نتائج أو أخطاء غير متوقعة. لإصلاح هذه المشكلة، استبدل فواصل الفاصلة العشرية في نماذج التعليمات البرمجية بالفاصل العشري المحلي (مثل ,)

بدلا من ذلك، لتشغيل برنامج باستخدام إعداد الثقافة en-US أضف التعليمات البرمجية التالية إلى أعلى البرنامج الخاص بك:

```
using System.Globalization;
```

ثم بعد إضافة العبارة السابقة، أضف التعليمة التالية عند الاستخدام:

```
CultureInfo.CurrentCulture = new CultureInfo("en-US");
```

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
string value1 = "5";
string value2 = "7";
int    result    =    Convert.ToInt32(value1)    *
Convert.ToInt32(value2);
Console.WriteLine(result);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
35
```

ملاحظة:

لماذا يُسمى الأسلوب ToInt32() لماذا لماذا لا يكون ToInt() لأن System.Int32 هو اسم نوع البيانات الأساسي في مكتبة فئات .NET Class Library التي تقوم لغة البرمجة C# بتعيينها للكلمة الأساسية int نظرًا لأن فئة التحويل Convert هي أيضًا جزء من مكتبة فئات .NET. يتم استدعاؤها باسمها الكامل، وليس باسم C# الخاص بها. من خلال تحديد أنواع البيانات كجزء من مكتبة فئات .NET. يمكن للعديد من لغات .NET. مثل Visual Basic, F#, IronPython وغيرها مشاركة نفس أنواع البيانات، ونفس الفئات في مكتبة فئات .NET.

يحتوي الأسلوب ToInt32() على 19 إصداراً محملاً بشكل زائد، مما يسمح له بقبول كل نوع بيانات تقريباً.

لقد استخدمت الأسلوب Convert.ToInt32() مع سلسلة string هنا، ولكن من المحتمل أن تستخدم الأسلوب TryParse() عندما يكون ذلك ممكناً.

لذلك، متى يجب عليك استخدام الفئة Convert؟ الفئة Convert هي الأفضل لتحويل الأرقام الكسرية إلى أرقام صحيحة (int) لأنها تقرب إلى أعلى، بالطريقة التي تتوقعها.

قارن بين casting ما يسمي إرسال أو صب، و converting ما يسمي تحويل. لتحويل decimal إلى int

يوضح المثال التالي ما يحدث عند محاولة تحويل decimal إلى int (تحويل تضييقي) مقابل استخدام الأسلوب Convert.ToInt32() لتحويل decimal ذاته إلى int

```
int value = (int)1.5m; // casting truncates
Console.WriteLine(value);

int value2 = Convert.ToInt32(1.5m); //
converting rounds up
Console.WriteLine(value2);
```

احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
1
2
```

تحويل الاقتطاعات Casting truncates وتحويل الجولات converting rounds

عندما تقوم بتحويل إرسال casting مثلاً `int value = (int)1.5m;` يتم اقتطاع القيمة العائمة (الكسور العشرية) بحيث تكون النتيجة 1 مما يعني تجاهل القيمة بعد الفاصلة العشرية تماماً، يمكنك تغيير القيمة العائمة إلى 1.999m وستكون نتيجة التحويل هي نفسها.

عندما تقوم بالتحويل باستخدام Convert.ToInt32() يتم تقريب القيمة العائمة للأعلى، لأقرب رقم صحيح، إلى 2 لكن إذا قمت بتغيير القيمة العائمة إلى 1.499m فسيتم تقريبها للأقل، إلى 1

خلاصة

لقد قمت بتغطية العديد من المفاهيم المهمة لتحويل البيانات data conversion وتحويل الإرسال casting

- منع حدوث خطأ في وقت التشغيل أثناء إجراء تحويل البيانات data conversion
- قم بإجراء تحويل صريح explicit cast لإخبار المحول البرمجي بأنك تفهم خطر فقدان البيانات
- الاعتماد على المحول البرمجي لتنفيذ تحويل ضمني implicit cast عند إجراء تحويل توسيع expanding conversion
- استخدم عامل تشغيل التحويل () ونوع البيانات لتنفيذ التحويل (على سبيل المثال `(int)myDecimal`)
- استخدم الفئة Convert عندما تريد إجراء تحويل تضييقي narrowing conversion ولكنك تريد إجراء التقريب، وليس اقتطاع المعلومات.

راجع معلوماتك

١- ما هي أفضل تقنية لتحويل نوع decimal إلى نوع int في C#؟

- التحويل المرسل Cast
- تضييق Narrowing
- التحويل الضمني Implicit conversion

٢ أي من التحويل التالي يقرب القيمة (عكس الاقتطاع)؟

- `int cost = (int)3.75m;`
- `int cost = Convert.ToInt32(3.75m);`
- `uint cost = (uint)3.75m;`

راجع إجابتك

١ التحويل **cast**

صحيح تحويل decimal إلى int هو تحويل تضييق لذلك cast هو أفضل إجابة

٢

```
int cost = Convert.ToInt32(3.75m);
```

صحيح Convert.ToInt32() تقريب القيم بدقة كسرية

٣ تمرين ـ فحص أسلوب TryParse()

عند العمل مع البيانات، في بعض الأحيان، تحتاج إلى تحويل بيانات السلسلة إلى نوع بيانات رقمية. كما تعلمت في الدروس السابقة، نظراً لأن نوع بيانات السلسلة يمكن أن يحتوي على قيمة غير رقمية، فمن المحتمل أن يؤدي إجراء تحويل من string إلى نوع بيانات رقمي إلى حدوث خطأ في وقت التشغيل.

على سبيل المثال، التعليمات البرمجية التالية:

```
string name = "Bob";
Console.WriteLine(int.Parse(name));
```

تسبب الاستثناء التالي:

```
System.FormatException: 'Input string was not
in a correct format.'
```

لتجنب استثناء تنسيق، استخدم الأسلوب TryParse() على نوع البيانات المستهدف.

استخدام TryParse()

الأسلوب TryParse() يقوم عدة أشياء في نفس الوقت:

- يحاول تحليل سلسلة string في نوع البيانات الرقمية numeric data type المعطاة.
- إذا نجح، فإنه يخزن القيمة المحولة في معلمة out parameter الموضحة في القسم التالي.
- يقوم بإرجاع قيمة bool للإشارة إلى ما إذا كان الإجراء قد نجح أو فشل.

يمكنك استخدام القيمة المرجعة المنطقية bool لاتخاذ إجراء بشأن القيمة (مثل إجراء بعض العمليات الحسابية) أو عرض رسالة إذا لم تنجح عملية التحليل.

ملاحظة

في هذا التمرين، ستستخدم نوع البيانات int ولكن يتوفر أسلوب TryParse() مماثل، على جميع أنواع البيانات الرقمية الأخرى.

معلمات خارجية Out parameters

يمكن أن ترجع الأساليب قيمة أو ترجع "فارغة" - ما يعني أنها لا ترجع أي قيمة - كما يمكن للأساليب أن ترجع القيم خلال المعلمات out والتي يتم تحديدها تمامًا مثل أي معلمات إدخال أخرى، ولكنها تتضمن الكلمة الأساسية out

تجربة الأسلوب **TryParse() a string into an int**

١. احذف عامل تشغيل // تعليق السطر أو استخدمه للتعليق التعليمات من خطوة التمرين السابقة، وإضافة التعليمات التالية:

```
string value = "102";
int result = 0;
if (int.TryParse(value, out result))
{
   Console.WriteLine($"Measurement: {result}");
}
else
{
   Console.WriteLine("Unable to report the
measurement.");
}
```

٢. افحص هذا السطر من التعليمات البرمجية:

```
if (int.TryParse(value, out result))
```

عند استدعاء أسلوب مع معلمة out parameter يجب استخدام الكلمة الأساسية out قبل المتغير الذي يحتوي على القيمة، يتم تعيين المعلمة out إلى المتغير result في التعليمات int.TryParse(value, out result) يمكنك بعد ذلك استخدام القيمة التي تحتوي عليها المعلمة out في بقية التعليمات البرمجية باستخدام المتغير result

يُرجع الأسلوب int.TryParse() القيمة true إذا نجح في تحويل قيمة متغير value من string إلى قيمة int وإلا فإنها ترجع false لذا، قم بإحاطة البيان في عبارة if ثم قم بتنفيذ منطق القرار، وفقًا لذلك.

يتم تخزين القيمة المحولة في المتغير الرقمي result int variable يتم الإعلان عن المتغير الرقمي result int variable وتهيئته، قبل هذا السطر من التعليمات البرمجية، لذلك يجب أن يسمح الوصول إليه داخل كتل التعليمات البرمجية، التي تنتمي إلى عبارات if - else statements وكذلك خارجها.

ترشد الكلمة الأساسية out المترجم compiler إلى أن أسلوب TryParse() لن يُرجع قيمة بالطريقة التقليدية فقط (كقيمة إرجاع) ولكنه سيقوم أيضًا بتوصيل مخرجات، من خلال هذه المعلمة ثنائية الاتجاه two-way parameter

عند تشغيل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
Measurement: 102
```

استخدم تحليل int لاحقاً في التعليمات البرمجية

١. لإثبات أن المتغير result الذي تم الإعلان عنه سابقاً، يتم ملؤه بواسطة المعلمة out ويمكن استخدامه أيضاً لاحقاً في تعليماتك البرمجية، قم بتحديث التعليمات البرمجية في محرر Visual Studio كما يلي:

```
string value = "102";
int result = 0;
if (int.TryParse(value, out result))
{
   Console.WriteLine($"Measurement: {result}");
}
else
{
   Console.WriteLine("Unable to report the
measurement.");
}
Console.WriteLine($"Measurement (w/ offset):
{50 + result}");
```

٢. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
Measurement: 102
Measurement (w/ offset): 152
```

٥. افحص السطر الأخير من التعليمات البرمجية في النموذج السابق Console.WriteLine($"Measurement (w/ offset): {50 + result}"); نظراً لأن المتغير result معرف خارج عبارة if يمكن الوصول إليه لاحقاً في التعليمات البرمجية.

تعديل متغير السلسلة string variable إلى قيمة لا يمكن تحليلها can't be parsed

وأخيراً، انظر إلى السيناريو الآخر – حيث يتم إعطاء أسلوب TryParse() قيمة سيئة عن قصد، لا يمكن تحويلها إلى int

١. عدل السطر الأول من التعليمات البرمجية، وإعادة تهيئة المتغير value إلى قيمة مختلفة:

```
string value = "bad";
```

٢. أيضاً، عدل السطر الأخير من التعليمات البرمجية للتأكد من أن النتيجة أكبر من 0 قبل عرض الرسالة الثانية:

```
if (result > 0)
   Console.WriteLine($"Measurement (w/ offset):
{50 + result}");
```

٣. يجب أن يتطابق مثال التعليمات البرمجية بالكامل المعدل، مع التعليمات البرمجية التالية:

```
string value = "bad";
int result = 0;
if (int.TryParse(value, out result))
{
   Console.WriteLine($"Measurement: {result}");
}
else
{
```

```
   Console.WriteLine("Unable to report the
measurement.");
}

if (result > 0)
   Console.WriteLine($"Measurement (w/ offset):
{50 + result}");
```

٤. احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيل التعليمات البرمجية. يجب أن تحصل على النتيجة التالية:

```
Unable to report the measurement.
```

٥. افحص آخر سطرين من التعليمات البرمجية المضافة في العينة السابقة

```
if (result > 0)
   Console.WriteLine($"Measurement (w/ offset):
{50 + result}");
```

نظرا لأن المتغير result تم تعريفه خارج العبارة if يمكن الوصول إليه لاحقاً في التعليمات البرمجية خارج الكتلة البرمجية، لذلك يمكن التحقق من result قيمته أكبر من الصفر قبل السماح بكتابة result + كإخراج بديل، يؤدي التحقق من result قيمته أكبر من الصفر إلى تجنب طباعة القيمة البديلة. Unable to report the measurement. بعد الرسالة.

خلاصة

الأسلوب TryParse() هو أداة قيمة، إليك بعض الأفكار السريعة التي يجب تذكرها:

- استخدم الأسلوب TryParse() عند تحويل string إلى نوع بيانات رقمية.
- يعمل الأسلوب TryParse() على إرجاع true إذا كان التحويل ناجحًا، ويرجع false إذا لم يكن التحويل ناجحًا.
- توفر المعلمات Out وسيلة ثانوية لأسلوب يرجع قيمة، في هذه الحالة، ترجع المعلمة out parameter القيمة المحولة the converted value
- استخدم الكلمة الأساسية the keyword out عند تمرير وسيطة argument إلى أسلوب قام بتعريف معلمة out

راجع معلوماتك

١- ما هي التقنية التي يجب استخدامها لتغيير قيمة متغير myInput وهي قيمة string `"2.71828"` إلى `decimal` ووضعها داخل متغير **`myInputDecimal`**

```
decimal myInputDecimal = (decimal)(myInput);
decimal myInputDecimal = myInput + 0;
decimal.TryParse(myInput, out myInputDecimal)
```

٢- ما هو أفضل وصف لنوع الإرجاع `decimal.TryParse()`

- `decimal`
- `bool`
- `out`

راجع إجابتك

١

```
decimal.TryParse(myInput, out myInputDecimal)
```

صحيح استخدام (TryParse (or Parse() هو أسلوب صالح

٢

`bool`

صحيح يقوم الأسلوب TryParse() بإرجاع bool

٤ تمرين - إكمال تحدي لدمج قيم مصفوفة string كسلاسل وكأعداد صحيحة

تعزز تحديات التعليمات البرمجية ما تعلمته وتساعدك على اكتساب بعض الثقة قبل المتابعة.

يتميز هذا الدرس بتحديين للتعليمات البرمجية، يفرض عليك هذا التحدي الأول، تقسيم البيانات حسب نوعها، ودمج البيانات أو إضافتها concatenate or add the data وفقًا لذلك.

ملاحظة

تم تصميم نماذج التعليمات البرمجية في هذا التمرين حسب إعدادات الثقافة في الولايات المتحدة، وتستخدم نقطة (.) كفاصل عشري. قد يؤدي إنشاء التعليمات البرمجية وتشغيلها باستخدام إعداد ثقافة يستخدم فواصل عشرية مختلفة (مثل الفاصلة ,) إلى نتائج أو أخطاء غير متوقعة. لإصلاح هذه المشكلة، استبدل فواصل الفاصلة العشرية في نماذج التعليمات البرمجية بالفاصل العشري المحلي (مثل ,)

بدلا من ذلك، لتشغيل برنامج باستخدام إعداد الثقافة en-US أضف التعليمات البرمجية التالية إلى أعلى البرنامج الخاص بك:

```
using System.Globalization;
```

ثم بعد إضافة العبارة السابقة، أضف التعليمة التالية عند الاستخدام:

```
CultureInfo.CurrentCulture = new
CultureInfo("en-US");
```

١. حدد كافة أسطر التعليمات البرمجية واحذفها في محرر Visual Studio وأحذفها اختيارياً، استخدم عامل تشغيل تعليق السطر // للتعليق على جميع التعليمات البرمجية من الخطوة السابقة.

٢. لإنشاء مثيل مصفوفة سلسلة string array أدخل التعليمة البرمجية الأولية التالية:

```
string[] values = { "12.3", "45", "ABC", "11",
"DEF" };
```

٣. إنشاء بنية تكرار حلقي يمكن استخدامها للتكرار من خلال كل قيمة سلسلة في مصفوفة values

٤. أكمل التعليمات البرمجية المطلوبة، وضعها داخل الكتلة البرمجية لبنية التكرار الحلقي للمصفوفة، من الضروري تنفيذ قواعد العمل التالية في منطق التعليمات البرمجية:

- **القاعدة ١:** إذا كانت القيمة أبجدية، فادمجها لتشكيل رسالة.
- **القاعدة ٢:** إذا كانت القيمة رقمية، أضفها إلى الإجمالي.
- **القاعدة ٣:** يجب أن تتطابق النتيجة مع الإخراج التالي:

```
Message: ABCDEF
Total: 68.3
```

٥. بمجرد اكتمال تعليماتك البرمجية، في قائمة **File** حدد **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٦. في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٧. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

ينبغي أن تشاهد الإخراج التالي:

```
Message: ABCDEF
    Total: 68.3
```

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٥ مراجعة حل تحدي لدمج قيم مصفوفة string كسلاسل وكأعداد صحيحة

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
string[] values = { "12.3", "45", "ABC", "11",
"DEF" };

decimal total = 0m;
string message = "";

foreach (var value in values)
{
    decimal number; // stores the TryParse
"out" value
    if (decimal.TryParse(value, out number))
    {
        total += number;
    } else
    {
        message += value;
    }
}

Console.WriteLine($"Message: {message}");
Console.WriteLine($"Total: {total}");
```

هذه التعليمة البرمجية هي مجرد "حل واحد ممكن" لأن هناك عدة طرق على الأرجح لحل هذا التحدي، يعتمد الحل المقترح على TryParse() ولكن من الممكن أن يكون لديك نهج آخر، يعمل بشكل جيد على قدم المساواة، فقط تأكد من تطابق نتيجتك مع الإخراج المطلوب.

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٦ تمرين - أكمل تحديًا لإخراج العمليات الحسابية math operations كأنواع أرقام محددة specific number types

إليك فرصة ثانية لاستخدام ما تعلمته حول الإرسال والتحويل casting and conversion لحل تحدي الترميز.

يساعدك التحدي التالي على فهم الآثار المترتبة على تحويل القيم، مع الأخذ في الاعتبار تأثير تضييق narrowing التحويلات واتساعها widening

١. احذف كافة التعليمات البرمجية من التمرين السابق، أو علق تنفيذها.

٢. أدخل التعليمة البرمجية الأولية التالية:

```
int value1 = 12;
decimal value2 = 6.2m;
float value3 = 4.3f;

// Your code here to set result1
// Hint: You need to round the result to
nearest integer (don't just truncate)
Console.WriteLine($"Divide value1 by value2,
display the result as an int: {result1}");

// Your code here to set result2
Console.WriteLine($"Divide value2 by value3,
display the result as a decimal: {result2}");

// Your code here to set result3
Console.WriteLine($"Divide value3 by value1,
display the result as a float: {result3}");
```

٣. استبدل تعليقات التعليمات الأولية Starter بالتعليمات الخاص بك لحل التحدي:

- حل result1: قسّمة value1 على value2 وعرض النتيجة كقيمة int

- حل result2: قسمة value2 على value3 وعرض النتيجة كقيمة decimal
- حل result3: قسّمة value3 على value1 وعرض النتيجة كقيمة float

حل التحدي بحيث يشبه الإخراج كالتالي:

```
Divide value1 by value2, display the result as an int: 2

Divide value2 by value3, display the result as a decimal: 1.4418604651162790697674418605

Divide value3 by value1, display the result as a float: 0.35833335
```

٤. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٥. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٦. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
Divide value1 by value2, display the result as
an int: 2

Divide value2 by value3, display the result as
a decimal: 1.441860465116279069767441860 5

Divide value3 by value1, display the result as
a float: 0.35833335
```

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٧ مراجعة حل للعمليات الحسابية الناتجة لأنواع أرقام محددة كتحدي

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
int value1 = 12;
decimal value2 = 6.2m;
float value3 = 4.3f;

// The Convert class is best for converting the
fractional decimal numbers into whole integer
numbers
// Convert.ToInt32() rounds up the way you
would expect.
int result1 = Convert.ToInt32(value1 / value2);
Console.WriteLine($"Divide value1 by value2,
display the result as an int: {result1}");

decimal result2 = value2 / (decimal)value3;
Console.WriteLine($"Divide value2 by value3,
display the result as a decimal: {result2}");

float result3 = value3 / value1;
Console.WriteLine($"Divide value3 by value1,
display the result as a float: {result3}");
```

هذه التعليمات هي "أحد الحلول الممكنة" لأنه من المحتمل أن يكون هناك عدة طرق لحل هذا التحدي. يستخدم الحل الكثير من عمليات التحويل المرسل casting وأسلوب التحويل (and a call to convert) ومع ذلك، فمن الممكن أن يعمل نهج آخر بشكل جيد بنفس القدر. فقط تأكد من أن النتيجة تطابق الإخراج المطلوب.

إذا نجحت، فتهانينا تابع لاختبار المعلومات في الدرس التالي.

٨ اختبر معلوماتك

١- ما هي التقنية التي يجب تجنبها عند تغيير string قيمته `4.123456789` إلى decimal؟

- `Convert.ToDecimal()`
- `decimal.TryParse()`
- `(decimal)`

٢- ما وصف نوع الإجراء الذي يتم تنفيذه عند تغيير float إلى int؟

- تحويل تضييق A narrowing conversion
- تحويل توسيع A widening conversion
- تحويل غير قانوني An illegal conversion

راجع إجابتك

١ `(decimal)`

صحيح لا يمكن تحويل string إلى decimal

٢ تحويل تضييق A narrowing conversion

صحيح يعد تغيير float إلى int تحويل تضييق، لأن float يمكن تخزين بيانات أكثر دقة مقارنة ب int

٩ الملخص

كان هدفك هو استخدام العديد من التقنيات المختلفة لتغيير نوع البيانات لقيمة معينة.

لقد استخدمت implicit conversion التحويل الضمني، والاعتماد على المحول البرمجي C# لإجراء تحويل توسيع widening conversions وعندما يكون المحول البرمجي غير قادر على إجراء تحويل ضمني، استخدمت تحويل صريح explicit conversions لقد استخدمت الأسلوب ToString() لتحويل نوع بيانات رقمية بشكل صريح إلى string

عندما تحتاج إلى إجراء تحويلات تضييق narrowing conversions استخدمت العديد من التقنيات المختلفة، استخدمت عامل تشغيل التحويل () عندما يكون إجراء التحويل امن، وكنت على استعداد لقبول اقتطاع القيم بعد الفاصلة العشرية. واستخدمت الأسلوب Convert() عندما تريد إجراء تحويل، واستخدام قواعد التقريب الشائعة، عند إجراء تحويل تضييقي.

وأخيرا، استخدمت الأساليب TryParse() عندما قد يؤدي التحويل من نوع بيانات string إلى نوع بيانات رقمي إلى خطأ في تحويل نوع البيانات.

بدون هذه الثروة من الخيارات، سيكون من الصعب العمل في لغة برمجة مكتوبة، لحسن الحظ، يمكن تسخير هذا النظام الذي تم تنفيذه جيدًا من الأنواع، والتحويل conversion والتحويل المرسل casting لبناء تطبيقات خالية من الأخطاء.

المصادر:

- التحويل والنوع دليل برمجة C#
- الأنواع المضمنة - مرجع C#
- القيم الافتراضية للأنواع - مرجع C#

الوحدة الثالثة

تنفيذ العمليات على المصفوفات باستخدام الأساليب المساعدة "مُدمجة"

استخدم الأساليب المساعدة القوية لمعالجة محتوى المصفوفات arrays والتحكم فيها.

الأهداف التعليمية

- فرز وترتيب وقلب عناصر المصفوفة.
- إلغاء تحديد عناصر المصفوفة وتغيير حجمها.
- تقسيم سلسلة string إلى مصفوفة من السلاسل أو الأحرف (chars)
- ضم عناصر المصفوفة في السلسلة string

محتويات الوحدة:

١ مقدمة

لنفترض أنك مطور برامج في فريق تم تعيينه للعمل مع شركة لوجستية. لديهم العديد من الاحتياجات لجمع وتنظيم بيانات أعمالهم، تختلف المشاريع من تعقب المخزون، والإبلاغ عنه، مما يتيح تمرير بيانات محددة من وإلى شركاء الأعمال، وتحديد الطلبات الاحتيالية المحتملة، كل مشروع مختلف، ولكن كل شيء يتطلب معالجة البيانات، في هذه المشاريع، ستقوم التطبيقات بتنفيذ عمليات على البيانات، مثل إضافة الإخراج، وحذفه، وفرزه، ودمجه، وحسابه، والتحقق من صحته، وتنسيقه.

في هذه الوحدة، ستستخدم مصفوفات "قوائم مرتبة" C# arrays للسماح لك بتخزين تسلسلات القيم، في بنية بيانات واحدة، بمجرد أن تكون لديك بيانات في مصفوفة ما، يمكنك معالجة ترتيب المصفوفة ومحتوياتها، علاوة على ذلك، ستتمكن من تنفيذ عمليات سلسلة قوية باستخدام الأساليب المساعدة للمصفوفة.

باستخدام العديد من الأساليب المساعدة، ستقوم بفرز البيانات، أو عكس ترتيب البيانات، ستقوم بمسح العناصر الموجودة في المصفوفة، وتغيير حجم المصفوفة لإضافة عناصر جديدة، ستقوم بتحويل سلسلة string إلى مصفوفة بتقسيمها إلى سلاسل أصغر، في كل مرة تصادف فيها حرفًا محدد، مثل الفاصلة، ستقوم أيضًا بتقسيم سلسلة إلى مجموعة من الأحرف. وأخيرًا، سوف تربط كافة عناصر المصفوفة بسلسلة واحدة "جملة نصية"

٢ تمرين – اكتشاف Sort() and Reverse()

تحتوي الفئة `Array` على أساليب، تستخدم لمعالجة محتوى المصفوفة، وحجمها، وترتيبها، في هذا التمرين، ستكتب التعليمات البرمجية التي تنفذ عمليات مختلفة على مصفوفة من معرفات المنصة. قد تكون التعليمات البرمجية الخاصة بك بداية تطبيق لتتبع وتحسين استخدام المنصات للشركة.

إعداد بيئة الترميز

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** حدد فتح مجلد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة Visual Studio Code **Terminal** حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب في موجه Terminal: الأوامر **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

يستخدم أمر .NET CLI. هذا قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد. ينشئ الأمر نيابة عنك مجلدات CsharpProjects, TestProject ويستخدم TestProject كاسم للملف .csproj أو كامتداد له.

٧. في قائمة استكشاف EXPLORER قم **بتوسيع المجلد CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**.

٩. حذف أسطر التعليمات البرمجية الموجودة.

يمكنك استخدام مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق Terminal

إنشاء مصفوفة من المنصات، ثم فرزها

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في لوحة المحرر.

يجب أن يكون Program.cs فارغ، إذا لم يكن كذلك، فحدد كافة أسطر التعليمات البرمجية واحذفها.

٢. اكتب التعليمات البرمجية التالية في محرر Visual Studio Code

```
string[] pallets = { "B14", "A11", "B12", "A13" };

Console.WriteLine("Sorted...");
Array.Sort(pallets);
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٣. خذ دقيقة لمراجعة السطر من التعليمات البرمجية `Array.Sort(pallets);` التي أضفتها.

هنا تستخدم أسلوب `Sort()` الفئة `Array` لفرز العناصر في المصفوفة أبجدياً رقمياً.

٤. في قائمة ملف **File** Visual Studio Code حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٥. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٦. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
Sorted...
-- A11
-- A13
-- B12
-- B14
```

عكس ترتيب المنصات

لعكس ترتيب المنصات باستخدام الأسلوب `Array.Reverse()` قم بتحديث تعليماتك البرمجية كما يلي:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};

Console.WriteLine("Sorted...");
Array.Sort(pallets);
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}

Console.WriteLine("");
Console.WriteLine("Reversed...");
Array.Reverse(pallets);
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٢. ركز على سطر التعليمات البرمجية `Array.Reverse(pallets);` من التعليمات السابقة التي أضفتها.

أنت تستخدم أسلوب `Reverse()` الفئة `Array` لعكس ترتيب العناصر.

٣. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
Sorted...
-- A11
-- A13
-- B12
-- B14

Reversed...
-- B14
-- B12
-- A13
-- A11
```

خلاصة

فيما يلي بعض الأفكار المهمة التي قمت بتغطيتها في هذا الدرس:

- تتمتع فئة المصفوفة Array class بأساليب methods يمكنها معالجة حجم ومحتويات المصفوفة.
- استخدم الأسلوب ()Sort لمعالجة الترتيب، استنادًا إلى نوع بيانات المصفوفة المعطى.
- استخدم الأسلوب ()Reverse لعكس ترتيب العناصر في المصفوفة.

اختبر معلوماتك

ما هو أفضل وصف للتعليمات البرمجية `Array.Sort(pallets);` حيث `pallets` هي `string array`

- Sort يمثل عملية تحويل
- Sort هو أسلوب مصفوفة
- Sort يستخدم لفرز مقدار المصفوفة من الدقة

راجع إجابتك

Sort هو أسلوب مصفوفة

صحيح Sort هو أسلوب مصفوفة

٣ تمرين – استكشاف Clear() and Resize()

بينما تستمر في إنشاء متتبع منصة لشركة الخدمات اللوجستية، افترض أنك تحتاج أيضاً إلى تتبع المنصات الجديدة، وإزالة المنصات القديمة من التتبع، كيف يمكنك إنجاز إنشاء وظيفة التعقب لإضافة المنصات وإزالتها؟

استخدام أساليب المصفوفة array methods لمسح مصفوفة وتغيير حجمها

يسمح لك الأسلوب `Array.Clear()` بإزالة محتويات عناصر معينة في المصفوفة، واستبداله بالقيمة الافتراضية للمصفوفة، على سبيل المثال، في مصفوفة string يتم استبدال قيمة العنصر التي تم مسحها ب null عند مسح عنصر مصفوفة int يتم الاستبدال ب 0 صفر.

يضيف الأسلوب Array.Resize() العناصر أو يزيلها من المصفوفة.

١. تحديث التعليمات البرمجية في محرر Visual Studio كما يلي:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};
Console.WriteLine("");

Array.Clear(pallets, 0, 2);
Console.WriteLine($"Clearing 2 ... count:
{pallets.Length}");
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٢. خذ دقيقة للتركيز على سطر التعليمات البرمجية `Array.Clear(pallets, 0, 2);`

عند تشغيل التعليمات البرمجية، سترى أنه تم مسح القيم المخزنة في العنصرين الأولين من المصفوفة، في الخاصية Length والعبارة foreach لا تزال العناصر موجودة، ولكنها الآن فارغة.

```
Clearing 2 ... count: 4
--
--
-- B12
-- A13
```

سلسلة فارغة Empty مقابل أخرى خالية null

عند استخدام Array.Clear() فإن العناصر التي تم مسحها، لم تعد تشير إلى سلسلة في الذاكرة، في الواقع، العنصر لا يشير إلى لا شيء، إن الإشارة إلى لا شيء، مفهوم مهم ربما يكون من الصعب فهمه في البداية.

ماذا لو حاولت استرداد قيمة عنصر تأثر بالأسلوب Array.Clear() هل يمكن القيام بذلك؟

الوصول إلى قيمة عنصر مسح cleared element

هناك حاجة إلى نهجين لتحديد قيمة عنصر مسح cleared element لمعرفة كيفية عمل المحول البرمجي C# مع قيمة خالية null value

١. إدراج أسطر تعليمات برمجية جديدة حول سطر التعليمات البرمجية Array.Clear(pallets, 0, 2); كما يلي:

```
Console.WriteLine($"Before: {pallets[0]}");
Array.Clear(pallets, 0, 2);
Console.WriteLine($"After: {pallets[0]}");
```

٢. تحقق من أن تعليماتك البرمجية يجب أن تتطابق مع قائمة التعليمات التالية:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};
Console.WriteLine("");

Console.WriteLine($"Before: {pallets[0]}");
Array.Clear(pallets, 0, 2);
Console.WriteLine($"After: {pallets[0]}");

Console.WriteLine($"Clearing 2 ... count:
{pallets.Length}");
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
Before: B14
After:
Clearing 2 ... count: 4
--
--
-- B12
-- A13
```

إذا كنت تركز على سطر الإخراج `After:` فقد تعتقد أن القيمة المخزنة في `pallets[0]` سلسلة فارغة empty string ومع ذلك، يقوم المحول البرمجي C# ضمنيًا بتحويل القيمة الخالية null value إلى سلسلة فارغة للعرض التقديمي.

استدعاء أسلوب مساعد للسلسلة string helper method على عنصر مسح cleared element

لإثبات أن القيمة المخزنة في `pallets[0]` بعد مسحها فارغة null عليك تعديل مثال التعليمات البرمجية لاستدعاء الأسلوب ToLower() على `pallets[0]` إذا كانت سلسلة، فينبغي أن تعمل بشكل جيد، ولكن إذا كانت خالية، فيجب أن يتسبب ذلك في طرح استثناء.

١. لاستدعاء الأسلوب ToLower() في كل مرة تحاول فيها كتابة `pallets[0]` إلى وحدة التحكم، قم بتحديث تعليماتك البرمجية كما يلي:

```
Console.WriteLine($"Before: {pallets[0].ToLower()}");
Array.Clear(pallets, 0, 2);
Console.WriteLine($"After: {pallets[0].ToLower()}");
```

٢. تأكد من أن التعليمات البرمجية تطابق سرد التعليمات التالية:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};
Console.WriteLine("");

Console.WriteLine($"Before: {pallets[0].ToLower()}");
Array.Clear(pallets, 0, 2);
Console.WriteLine($"After: {pallets[0].ToLower()}");

Console.WriteLine($"Clearing 2 ... count:
{pallets.Length}");
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٣. احفظ ملف التعليمات البرمجية، ثم استخدم Visual Studio Code لتشغيلها. هذه المرة، سترى رسالة خطأ كبيرة، إذا قمت بملاحظة النص، فسترى الرسالة التالية:

```
System.NullReferenceException: Object reference
not set to an instance of an object.
```

تم طرح هذا الاستثناء لأن محاولة استدعاء الأسلوب على محتويات العنصر pallets[0] تحدث قبل أن يكون لدى المحول البرمجي C# فرصة لتحويل ضمنياً قيمة خالية null إلى سلسلة فارغة empty string

ومغزى القصة هو أن Array.Clear() سوف يزيل مرجع عنصر المصفوفة إلى القيمة، إذا كانت موجودة، لإصلاح ذلك، تحقق من القيمة الخالية null قبل محاولة طباعة القيمة.

لتجنب الخطأ، أضف عبارة if قبل الوصول إلى عنصر مصفوفة يحتمل أن يكون فارغاً.

```
if (pallets[0] != null)
 Console.WriteLine($"After: {pallets[0].ToLower()}");
```

تغيير حجم المصفوفة لإضافة المزيد من العناصر

١. أعد صياغة سرد التعليمات البرمجية من الخطوة ١ لإضافة تعليمات لتغيير حجم المصفوفة، عند الاستكمال، يجب أن تتطابق التعليمات مع التالية:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};
Console.WriteLine("");

Array.Clear(pallets, 0, 2);
Console.WriteLine($"Clearing 2 ... count:
{pallets.Length}");
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}

Console.WriteLine("");
Array.Resize(ref pallets, 6);
```

```
Console.WriteLine($"Resizing 6 ... count:
{pallets.Length}");

pallets[4] = "C01";
pallets[5] = "C02";

foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٢. خذ بضع دقائق للتركيز على السطر `Array.Resize(ref pallets, 6);`

هنا، تقوم باستدعاء الأسلوب `Resize()` الذي يمر في المصفوفة `pallets` حسب المرجع، باستخدام الكلمة الأساسية **ref** في بعض الحالات، تتطلب الأساليب منك تمرير الوسيطات arguments حسب القيمة (الافتراضية) أو حسب المرجع (باستخدام الكلمة الأساسية المرجعية)

تتطلب أسباب ضرورة ذلك شرحاً طويلاً ومعقداً، حول كيفية إدارة الكائنات في .NET. وللأسف، يتجاوز ذلك نطاق هذا الدرس، عند الشك، يوصى بالنظر إلى Intellisense أو Microsoft Docs للحصول على أمثلة حول كيفية استدعاء أسلوب معين بشكل صحيح.

في هذه الحالة، تقوم بإعادة تغيير حجم المصفوفة pallets من 4 عناصر إلى 6 تتم إضافة العناصر الجديدة في نهاية العناصر الحالية. سيكون العنصران الجديدان خاليين null حتى تقوم بتعيين قيمة لهما.

٣. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن ترى الإخراج التالي:

```
Clearing 2 ... count: 4
--
--
-- B12
-- A13

Resizing 6 ... count: 6
```

```
--
--
-- B12
-- A13
-- C01
-- C02
```

تغيير حجم المصفوفة لإزالة العناصر remove elements

وعلى العكس من ذلك، يمكنك إزالة عناصر المصفوفة باستخدام Array.Resize()

١. تحديث تعليماتك البرمجية في المحرر، كما يلي:

```
string[] pallets = { "B14", "A11", "B12", "A13"
};
Console.WriteLine("");

Array.Clear(pallets, 0, 2);
Console.WriteLine($"Clearing 2 ... count:
{pallets.Length}");
foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}

Console.WriteLine("");
Array.Resize(ref pallets, 6);
Console.WriteLine($"Resizing 6 ... count:
{pallets.Length}");

pallets[4] = "C01";
pallets[5] = "C02";

foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
```

```
}

Console.WriteLine("");
Array.Resize(ref pallets, 3);
Console.WriteLine($"Resizing 3 ... count:
{pallets.Length}");

foreach (var pallet in pallets)
{
    Console.WriteLine($"-- {pallet}");
}
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن ترى الإخراج التالي:

```
Clearing 2 ... count: 4
--
--
-- B12
-- A13

Resizing 6 ... count: 6
--
--
-- B12
-- A13
-- C01
-- C02

Resizing 3 ... count: 3
--
--
-- B12
```

لاحظ أن استدعاء Array.Resize() لم يؤدي إلى إزالة العنصرين الخاليين الأولين null بل أزال العناصر الثلاثة الأخيرة. وتمت إزالة العناصر الثلاثة الأخيرة على الرغم من أنها تحتوي على قيم سلسلة.

هل يمكنك إزالة العناصر الخالية من المصفوفة؟

إذا لم يزيل الأسلوب `Array.Resize()` العناصر الفارغة empty من مصفوفة، فهل هناك أسلوب مساعد آخر سيقوم بالمهمة تلقائيا؟ لا! أفضل طريقة لإفراغ العناصر من مصفوفة هي حساب عدد العناصر غير الفارغة، عن طريق التكرار من خلال كل عنصر، وزيادة قيمة متغير (العداد) بعد ذلك، قد تقوم بـ «إنشاء مصفوفة ثانِية» وهو حجم متغير العداد. وأخيرًا، يمكنك القيام بـ «التكرار الحلقي» خلال كل عنصر في المصفوفة الأصلية، ثم نسخ القيم غير الخالية إلى المصفوفة الجديد.

خلاصة

فيما يلي بعض الأفكار المهمة التي قمت بتغطيتها في هذا الدرس:

- استخدم الأسلوب Clear() لإفراغ القيم من العناصر في المصفوفة.
- استخدم الأسلوب Resize() لتغيير عدد العناصر في المصفوفة، (إزالة أو إضافة العناصر من نهاية المصفوفة)
- تعتبر عناصر المصفوفة الجديدة، والعناصر التي تم مسحها خالية null بمعنى أنها لا تشير إلى قيمة في الذاكرة.

اختبر معلوماتك

ما أفضل وصف للتعليمات البرمجية `Array.Clear(pallets, 0, 2);` حيث `pallets` عبارة عن **`string array`**

- `Array.Clear(pallets, 0, 2);` يزيل البيانات من عنصرين من عناصر المصفوفة بدءًا من العنصر 0
- `Array.Clear(pallets, 0, 2);` يزيل البيانات من مواقع عناصر المصفوفة `pallets[0]` و `pallets[2]`
- `Array.Clear(pallets, 0, 2);` يزيل عناصر المصفوفة التي تساوي 0 أو 2

راجع إجابتك

`Array.Clear(pallets, 0, 2);` يزيل البيانات من عنصرين من عناصر المصفوفة بدءًا من العنصر 0

صحيح Clear هو أسلوب من المصفوفات التي تزيل (تستبدل ب Null) عناصر محددة في مصفوفة

٤ تمرين – اكتشاف Split() and Join()

أثناء متابعة أعمال التطوير لشركة لوجستية، تبدأ في إنشاء سلسلة من التطبيقات الصغيرة، مجتمعة، تستقبل التطبيقات البيانات من نظام أحد الشركاء، وتعدل البيانات، ثم تمررها إلى نظام داخلي بالتنسيق الذي يتطلبه.

لإجراء تحويل البيانات، تحتاج إلى قبول البيانات الواردة كسلسلة، وتحليلها في عناصر بيانات أصغر، ثم معالجتها لمطابقة التنسيق المختلف المطلوب. كيف يمكنك تحليل بيانات السلسلة في عناصر بيانات أصغر؟

أساليب مصفوفة نوع بيانات String

تتمتع متغيرات النوع string بالعديد من الأساليب المدمجة، التي تقوم بتحويل سلسلة فردية إما إلى مصفوفة من سلاسل أصغر، أو مصفوفة من أحرف فردية.

عند معالجة البيانات من أنظمة الكمبيوتر الأخرى، أحيانًا يتم تنسيقها أو ترميزها بطريقة غير مفيدة للغرض، في هذه الحالات، يمكنك استخدام أساليب مصفوفة من نوع بيانات String لتحليل سلسلة في مصفوفة.

استخدام ToCharArray() لعكس string

١. تحديث التعليمات البرمجية في المحرر كما يلي:

```
string value = "abc123";
char[] valueArray = value.ToCharArray();
```

استخدم الأسلوب ToCharArray() لإنشاء مصفوفة من char يحتوي كل عنصر من عناصر المصفوفة على حرف واحد من السلسلة الأصلية.

عكس، ثم دمج مصفوفة char في سلسلة جديدة new string

بعد ذلك، يمكنك عكس ترتيب الأحرف في المصفوفة، ثم استخدام الأسلوب Write لدمجها مرة أخرى في إخراج واحد.

١. تحديث التعليمات البرمجية في المحرر كما يلي:

```
string value = "abc123";
char[] valueArray = value.ToCharArray();
Array.Reverse(valueArray);
string result = new string(valueArray);
Console.WriteLine(result);
```

ينشئ التعبير `new string(valueArray)` مثيلاً فارغًا جديدًا للفئة System.String (هو نفس نوع بيانات string في C#) يمر في مصفوفة الأحرف char array كمنشئ (كالدالة الإنشائية) constructor

ملاحظة

ما هي الكلمة الرئيسية new؟ كيف ترتبط فئة System.String بنوع بيانات السلسلة في C#؟ ما هو المنشئ constructor؟ جميع هذه الأسئلة الرائعة، للأسف خارج نطاق هذه الوحدة. ننصحك بمواصلة التعلم حول مكتبة .NET Class Library بالإضافة إلى الفئات والكائنات classes and objects في C# لفهم ما يحدث خلف الكواليس بشكل كامل مع هذا التعبير من التعليمات البرمجية، حالياً، استخدم محرك بحث ووثائق Microsoft للعثور على أمثلة يمكنك استخدامها في مثل هذه المواقف، حيث تعلم أنك تريد إجراء تحويل، ولكنك غير متأكد من كيفية القيام بذلك باستخدام C#

٢. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
321cba
```

دمج كافة الأحرف في سلسلة جديدة ذات قيمة مفصولة بفواصل a new comma-separated-value string باستخدام Join()

ربما تحتاج إلى فصل كل عنصر من عناصر مصفوفة الأحرف char array باستخدام فاصلة، كما هو شائع عند العمل مع البيانات التي يتم تمثيلها كنص ASCII للقيام بذلك، ستقوم بتعليق سطر التعليمات البرمجية الذي أضفته في الخطوة ٢ واستخدام أسلوب Join() الفئة String وتمرير الحرف (الفاصلة) الذي تحدد به كل مقطع، والمصفوفة نفسها.

١. تحديث التعليمات البرمجية في المحرر كما يلي:

```
string value = "abc123";
char[] valueArray = value.ToCharArray();
Array.Reverse(valueArray);
// string result = new string(valueArray);
string result = String.Join(",", valueArray);
Console.WriteLine(result);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
3,2,1,c,b,a
```

تقسيم سلسلة جديدة ذات قيمة مفصولة بفواصل the comma-separated-value string في مصفوفة من السلاسل باستخدام Split()

لإكمال التعليمات البرمجية، يمكنك استخدام الأسلوب Split() وهو للمتغيرات من النوع string لإنشاء مصفوفة من السلاسل.

أضف الأسطر التالية من التعليمات البرمجية في أسفل الملف:

```
string[] items = result.Split(',');
foreach (string item in items)
{
    Console.WriteLine(item);
}
```

٢. خذ دقيقة لمراجعة التعليمات البرمجية السابقة.

يتم توفير الفاصلة إلى Split() كمحدد لتقسيم سلسلة طويلة واحدة، إلى سلاسل أصغر، ثم تستخدم التعليمات البرمجية حلقة foreach للتكرار من خلال كل عنصر من عناصر المصفوفة النصية items التي تم إنشاؤها حديثاً.

٣. تحقق من أن التعليمات البرمجية الآن، كما يلي:

```
string value = "abc123";
char[] valueArray = value.ToCharArray();
Array.Reverse(valueArray);
// string result = new string(valueArray);
string result = String.Join(",", valueArray);
Console.WriteLine(result);
```

```
string[] items = result.Split(',');
foreach (string item in items)
{
    Console.WriteLine(item);
}
```

٤. احفظ ملف التعليمات البرمجية، ثم شغلها.

ينبغي أن تشاهد الإخراج التالي:

```
3,2,1,c,b,a
3
2
1
c
b
a
```

تم إنشاء مصفوفة items باستخدام `string[] items = result.Split(',');` واستخدامها في حلقة foreach وعرض الأحرف الفردية من السلسلة الأصلية الموجودة في متغير value

خلاصة

إليك بعض الأشياء التي يجب تذكرها عند العمل مع السلاسل والمصفوفات:

- استخدام أساليب مثل `ToCharArray()` and `Split()` لإنشاء مصفوفة.
- استخدام أساليب مثل Join() أو إنشاء سلسلة جديدة new string تمرر في مصفوفة أحرف `char` لتحويل المصفوفة مرة أخرى إلى سلسلة واحدة.

اختبر معلوماتك

١- `String.Join("-", myArray);` مثال على الدمج، ما هو التفسير الأنسب لهذا المثال؟

إنشاء مصفوفة أحرف من عناصر موجودة في myArray متصلة بـ **,** (فواصل)

إنشاء سلسلة من العناصر الموجودة في myArray المرتبطة بـ - (شَرْطَاتٌ)

إنشاء سلسلة باسم myArray متصلة بـ - (شَرْطَاتٌ)

اختبر إجابتك

إنشاء سلسلة من العناصر الموجودة في myArray المرتبطة بـ - (شَرْطَاتٌ)

صحيح String.Join إنشاء سلاسل من عناصر myArray

٥ تمرين ـ أكمل التحدي لعكس الكلمات في جملة

تعزز تحديات التعليمات البرمجية ما تعلمته، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

ستحتاج عدة مرات إلى دمج العديد من الأفكار، التي تغطيها هذه الوحدة في حل واحد، يجب أن تعمل على تحليل مشكلة أكبر إلى عدة مشاكل أصغر، ثم استخدام الأفكار المختلفة في هذه الوحدة لحل كل مشكلة صغيرة.

كتابة التعليمات البرمجية لعكس كل كلمة في رسالة

١. حدد كافة أسطر التعليمات البرمجية في المحرر واحذفها.

٢. أضف التعليمات البرمجية التالية في المحرر:

```
string pangram = "The quick brown fox jumps
over the lazy dog";
```

٣. اكتب الرمز الضروري لقلب حروف كل كلمة في مكانها، وعرض النتيجة.

بمعنى آخر، لا تقم فقط بعكس كل حرف في المتغير pangram بدلًا من ذلك، ستحتاج إلى عكس الحروف الموجودة في كل كلمة فقط، وطباعة الكلمة المعكوسة في موضعها الأصلي في الرسالة.

يجب أن تنتج التعليمات البرمجية الإخراج التالي:

```
ehT kciuq nworb xof spmuj revo eht yzal god
```

هام

هذا تحدي صعب بشكل خاص. ستحتاج إلى دمج العديد من المفاهيم التي تعلمتها في هذا التمرين، بما في ذلك استخدام Split(), ToCharArray(), Array.Reverse(), String.Join() سوف تحتاج أيضًا إلى إنشاء مصفوفات متعددة، وبيان تكرار واحد على الأقل.

حظ سعيد! ما عليك سوى الاستمرار في تحليل المشكلة، وتقسيمها إلى خطوات صغيرة، ثم حل خطوة تلو خطوة، قبل الانتقال إلى الخطوة التالية.

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٦ مراجعة حل الكلمات العكسية في تحدي الجملة

الحل التالي، هو واحد من العديد من الحلول الممكنة، النهج المتخذ لحل هذا التحدي هو تقسيم الحل إلى أربع خطوات:

١. تقسيم السلسلة النصية pangram على حرف المسافة، لإنشاء مصفوفة السلسلة النصية message

٢. إنشاء مصفوفة جديدة newMessage تقوم بتخزين نسخة معكوسة من سلسلة "word" من المصفوفة message

٣. التكرار الحلقي عبر كل عنصر في المصفوفة message وعكسها، وتخزين هذا العنصر في المصفوفة newMessage

٤. انضم إلى سلاسل "word" من المصفوفة newMessage باستخدام مسافة مرة أخرى، لإنشاء السلسلة الفردية المطلوبة للكتابة إلى وحدة التحكم.

النتيجة النهائية لهذا الحل:

```
string pangram = "The quick brown fox jumps
over the lazy dog";

// Step 1
string[] message = pangram.Split(' ');

//Step 2
string[] newMessage = new
string[message.Length];

// Step 3
for (int i = 0; i < message.Length; i++)
{
    char[] letters = message[i].ToCharArray();
    Array.Reverse(letters);
    newMessage[i] = new string(letters);
}
```

```
//Step 4
string result = String.Join(" ", newMessage);
Console.WriteLine(result);
```

هذه التعليمات هو مجرد حل واحد ممكن، لأنه بإمكانك اتباع مناهج مختلفة، لمختلف الخطوات في هذه العملية، طالما الإخراج الخاص بك يطابق ما يلي، فإنك نجحت.

```
ehT kciuq nworb xof spmuj revo eht yzal god
```

إذا نجحت، فتهانينا!

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ تمرين - إكمال تحدي لتحليل مجموعة من الطلبات وفرز هذه الطلبات لاكتشاف الأخطاء المحتملة

تأتي البيانات بتنسيقات عديدة، في هذا التحدي، يجب تحليل "معرفات الطلب" الفردية، وإخراج "OrderIDs" التي تم فرزها، ووضع علامة عليها على أنها "خطأ" إذا لم تكن بطول أربعة أحرف بالضبط.

١. أضف التعليمات البرمجية التالية للحصول على بيانات التحدي:

```
string orderStream =
"B123,C234,A345,C15,B177,G3003,C235,B179";
```

٢. لاحظ في التعليمات البرمجية السابقة، أن متغير orderStream يحتوي على string سلسلة عدة معرفات طلبات، مفصولة بفواصل.

٣. إضافة تعليمة برمجية أسفل التعليمات السابقة، لتحليل "معرفات الطلبات" من سلسلة الطلبات الواردة وتخزين "معرفات الطلب" في مصفوفة

٤. إضافة تعليمة برمجية لإخراج كل "معرف الطلب" بترتيب مفروز، وعلامة الأوامر التي لا يبلغ طولها أربعة أحرف بالضبط ك "- Error"

٥. حفظ التعليمات البرمجية وتشغيلها.

يجب أن تنتج التعليمات البرمجية الإخراج التالي:

```
A345
B123
B177
B179
C15     - Error
C234
C235
G3003   - Error
```

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٨ مراجعة حل تحليل مجموعة من الطلبات وفرز هذه الطلبات لاكتشاف الأخطاء المحتملة

الحل التالي المقدم هو واحد من العديد من الحلول الممكنة.

```
string orderStream =
"B123,C234,A345,C15,B177,G3003,C235,B179";
string[] items = orderStream.Split(',');
Array.Sort(items);

foreach (var item in items)
{
    if (item.Length == 4)
    {
        Console.WriteLine(item);
    }
    else
    {
        Console.WriteLine(item + "\t- Error");
    }
}
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، طالما أن الإخراج الخاص بك يطابق ما يلي، فإنك نجحت.

```
A345
B123
B177
B179
C15     - Error
C234
C235
G3003   - Error
```

تابع التحقق من المعلومات في الدرس التالي.

إذا واجهتك مشكلة في إكمال هذا التحدي، يجب مراجعة الدروس السابقة.

٩ اختبر معلوماتك

١- ما الأسلوب التي يغير ترتيب العناصر في مصفوفة string؟

- `myArray.Resize()`
- `myArray.Clear()`
- `Array.Sort()`

٢- ما تعريف null؟

- مثل السلسلة الفارغة empty string
- متشابهة حيث القيمة صفر
- القيمة التي تشير إلى أن المتغير يشير إلى لا شيء في الذاكرة

راجع إجابتك

١ `Array.Sort()`

صحيح بوضع Array.Sort() عناصر المصفوفة string بترتيب تصاعدي

٢ القيمة التي تشير إلى أن المتغير يشير إلى لا شيء في الذاكرة

صحيح القيمة الخالية Null ليست هي نفسها السلسلة الفارغة empty string أو القيمة الصفرية

١٠ الملخص

في هذه الوحدة، أكملت التدريبات باستخدام الأساليب المساعدة، لاستخدام المصفوفات بشكل أكثر فعالية من أجل:

- مسح العناصر في مصفوفة، وتعلم يتم تعيين العناصر إلى فارغة، باستخدام الأسلوب Array.Clear()
- تغيير حجم مصفوفة لإضافة العناصر وإزالتها باستخدام الأسلوب Array.Resize()
- تحويل سلسلة إلى مصفوفة باستخدام String.Split() تحديد حرف فاصل سلسلة لإنتاج قيمة في المصفوفة الذي تم إرجاعه.
- دمج كافة عناصر المصفوفة في سلسلة واحدة باستخدام الأسلوب String.Join()

سمحت لك الأساليب المساعدة للمصفوفات، بالعمل بمرونة مع البيانات في التطبيقات، بدون هذه الميزات، سوف تكون المصفوفات أقل فائدة.

الوحدة الرابعة

تنسيق البيانات الأبجدية الرقمية للعرض

استخدم C# لتنسيق السلاسل باستخدام المسافات البيضاء، والأحرف الخاصة special characters والمسافة البادئة، والمحاذاة، إعطاء سياق للبيانات الرقمية مثل العملة، والنسب المئوية، والأرقام ذات الكسور العشرية.

الأهداف التعليمية

- استخدم تسلسلات أحرف الإلغاء أو الهروب character escape لإضافة علامات تبويب tabs وخطوط جديدة، وأحرف Unicode إلى سلاسلنا "الجمل النصية"
- إنشاء القيم الفعلية للسلسلة الحرفية string literals وأحرف الهروب الشائعة، مثل شرطة مائلة عكسية backslash (\) علامات اقتباس مزدوجة double-quotes
- دمج القوالب مع المتغيرات، باستخدام التنسيق المركب composite formatting ودمج النصوص أو استيفاء السلسلة string interpolation
- تضمين محددات التنسيق المختلفة، للنسب المئوية والعملة والأرقام.

محتوات الوحدة:

١ المقدمة

لنفترض أنك تعمل في قسم المبيعات والتسويق، الذي يرسل آلاف الرسائل المخصصة إلى عملاء الشركة الحاليين، الذين هم مستثمرون مؤسسون، مهمة فريقك هي الترويج لمنتجات مالية جديدة للعميل، يدمج كل خطاب ترسله معلومات مخصصة حول العميل، يقارن الحرف بين عائدات قوائم المشاريع الحالية، والعائدات المتوقعة، باستخدام أحدث المنتجات، كيف ستقوم بدمج البيانات، وتنسيقها بشكل صحيح؟

من منظور رفيع المستوى، يهتم مطورو البرامج بما يلي:

- **إدخال البيانات**، بما في ذلك البيانات التي قام مستخدم بكتابتها من لوحة مفاتيح، أو باستخدام الماوس أو الجهاز أو نظام برامج آخر عبر طلب شبكة.
- **معالجة البيانات**، بما في ذلك منطق القرار، ومعالجة البيانات، وإجراء العمليات الحسابية.
- **إخراج البيانات**، بما في ذلك العرض التقديمي لمستخدم عبر رسالة سطر أوامر أو نافذة أو صفحة ويب أو حفظ البيانات المعالجة في ملف وإرسالها إلى خدمة شبكة.

في هذه الوحدة، يمكنك التركيز على إخراج البيانات، يمكنك العمل مع بيانات السلسلة التي تقوم بتنسيقها مع المحاذاة والتباعد، يمكنك تنسيق البيانات للعرض التقديمي كعملة وأرقام أخرى.

في نهاية الوحدة، سوف تعرف أدوات جديدة، يمكنك استخدامها لإخراج البيانات للمستخدمين، التي تنقل معناها وتوضحها.

في هذه الوحدة، ستتعلم كيفية العمل مع البيانات النصية، والبيانات الرقمية لحل مشاكل الأعمال في العالم الحقيقي في C# يمكنك إنشاء نسخة تقليدية من إيصال تم إصداره للمنتجات المباعة، ويمكنك دمج البيانات لإرسال رسالة تسويقية مخصصة للعملاء.

٢ تمرين - التحقيق في أساسيات تنسيق السلسلة string formatting

في هذه الوحدة، ستتعلم أساليب لتنسيق السلاسل للعرض الفعال، خاصة بالنسبة للحالات التي تستخدم متغيرات متعددة.

إعداد بيئة الترميز

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** حدد فتح مجلد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** Visual Studio Code حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب في موجه الأوامر Terminal: **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

يستخدم أمر .NET CLI. هذا قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد. ينشئ الأمر نيابة عنك مجلدات CsharpProjects, TestProject ويستخدم TestProject كاسم للملف csproj. أو كامتداد له.

٧. في قائمة استكشاف EXPLORER قم **بتوسيع المجلد CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**

٩. حذف أسطر التعليمات البرمجية الموجودة.

يمكنك استخدام مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق Terminal

ما هو التنسيق المركب؟ Composite Formatting

يستخدم التنسيق المركب العناصر النائبة المرقمة numbered placeholders داخل سلسلة، في وقت التشغيل، يتم حل كل شيء داخل الأقواس إلى قيمة يتم تمريرها، أيضًا بناءً على موضعها.

يستخدم هذا المثال للتنسيق المركب أسلوب Format() مضمناً في الكلمة الأساسية لنوع البيانات string

تحديث التعليمات البرمجية في المحرر كما يلي:

```
string first = "Hello";
string second = "World";
string result = string.Format("{0} {1}!",
first, second);
Console.WriteLine(result);
```

إذا قمت بتشغيل هذه التعليمات البرمجية، فستلاحظ الإخراج التالي:

```
Hello World!
```

هناك بعض الأشياء الهامة التي يجب ملاحظتها حول هذه التعليمة البرمجية.

- تحتوي أنواع البيانات، والمتغيرات الخاصة بنوع بيانات معين، على "أساليب مساعدة" مضمنة لتسهيل مهام معينة.
- السلسلة الحرفية `"{0} {1}!"` تشكل قالباً، يتم استبدال أجزاء منه في وقت التشغيل.
- يتم استبدال الرمز المميز `{0}` بالوسيطة argument الأولى بعد قالب السلسلة، وبعبارة أخرى، قيمة المتغير `first`
- يتم استبدال الرمز المميز `{1}` بالوسيطة argument الثانية بعد قالب السلسلة، وبعبارة أخرى، قيمة المتغير `second`

ملاحظة

قد تظن أنه من الغريب البدء بالرقم 0 هذا شائع جدًا في تطوير البرمجيات، عندما يكون هناك تسلسل من العناصر التي يمكن تحديدها باستخدام رقم، سيبدأ الترقيم عادةً من 0

١. تحديث التعليمات البرمجية كما يلي:

```
string first = "Hello";
string second = "World";
Console.WriteLine("{1} {0}!", first, second);
Console.WriteLine("{0} {0} {0}!", first,
second);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
World Hello!
Hello Hello Hello!
```

وهناك بعض الملاحظات حول هذه الأمثلة:

- بالنسبة إلى العبارة الأولى Console.WriteLine() لاحظ، يمكن ترتيب الرموز المميزة بأي ترتيب، يحتوي نموذج التعليمات البرمجية على {1} قبل {0}
- بالنسبة إلى العبارة الثانية Console.WriteLine() لاحظ، يمكن إعادة استخدام الرموز المميزة بثلاثة مثيلات من {0} ولم يتم استخدام الوسيطة الثانية للمتغير second ومع ذلك، لا تزال التعليمات البرمجية تعمل دون خطأ.

ما هو استيفاء السلسلة؟ string interpolation

استيفاء السلسلة هو تقنية تبسط التنسيق المركب composite formatting

بدلاً من استخدام رمز مميز مرقم، وتضمين القيمة الحرفية أو اسم المتغير في قائمة الوسائط إلى String.Format() أو Console.WriteLine() يمكنك فقط استخدام اسم المتغير داخل الأقواس المعقوفة.

١. لكي تستوفي سلسلة القيم يجب أن تبدأها بالتوجيه $ الآن، قم بإنشاء نفس الأمثلة من وقت سابق باستخدام استيفاء السلسلة بدلاً من التنسيق المركب. حدث التعليمات البرمجية الخاصة بك كما يلي:

```
string first = "Hello";
string second = "World";
Console.WriteLine($"{first} {second}!");
Console.WriteLine($"{second} {first}!");
Console.WriteLine($"{first} {first} {first}!");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
Hello World!
World Hello!
Hello Hello Hello!
```

ملاحظة

إذا نظرت إلى أمثلة التعليمات البرمجية في الكتب وعلى الإنترنت، من المحتمل أن ترى كلاً من التنسيق المركب واستيفاء السلسلة مستخدم، ولكن بشكل عام يجب عليك اختيار استكمال أو استيفاء السلسلة.

تنسيق العملة Formatting currency

يمكن استخدام التنسيق المركب واستيفاء السلسلة، لتنسيق القيم للعرض في ظل لغة وثقافة معينة، في المثال التالي يتم استخدام محدد تنسيق العملة `:C` لتقديم المتغيرات `price`, `discount` كعملة.

حدث التعليمات البرمجية الخاصة بك كما يلي:

```
decimal price = 123.45m;
int discount = 50;
Console.WriteLine($"Price: {price:C} (Save
{discount:C})");
```

إذا قمت بتنفيذ هذه التعليمة البرمجية على كمبيوتر تم تعيين لغة عرض Windows الخاصة به إلى الإنجليزية (الولايات المتحدة) فستلاحظ الإخراج التالي:

```
Price: $123.45 (Save $50.00)
```

لاحظ كيف أن إضافة `:C` إلى الرموز المميزة داخل الأقواس المعقوفة، تُنسق الرقم كعملة بغض النظر عن استخدام `int` or `decimal`

ملاحظة

ماذا يحدث إذا كان بلدك ولغتك غير معروفين؟ إذا قمت بتشغيل التعليمات البرمجية السابقة في محرر .NET. "الموجود في المتصفح" كما هو الحال في TrydotNet فسترى النتيجة التالية **Price: ¤123.45 (Save ¤50.00)** يتم استخدام الرمز ¤ بدلاً من الرمز الخاص بعملة بلدك، هذا

رمز عام يستخدم للإشارة إلى العملة، بغض النظر عن نوع العملة، ترى هذا الرمز في محرر .NET. لأنه يتجاهل موقعك الحالي.

كيفية تأثير بلد/منطقة المستخدم ولغته على تنسيق السلسلة

ماذا لو قمت بتنفيذ التعليمات البرمجية السابقة على جهاز كمبيوتر في فرنسا؟ تم تعيين لغة عرض Windows الخاصة به على الفرنسية، في هذه الحالة سترى الإخراج التالي:

```
Price: 123,45 € (Save 50,00 €)
```

سبب الإخراج السابق "€" هو أن ميزة تنسيق عملة السلسلة تعتمد على إعداد الكمبيوتر المحلي، للثقافة culture وفي هذا السياق، يشير مصطلح "الثقافة" إلى البلد/المنطقة ولغة المستخدم النهائي، رمز الثقافة عبارة عن سلسلة مكونة من خمسة أحرف تستخدمها أجهزة الكمبيوتر، لتحديد موقع المستخدم النهائي ولغته، يضمن رمز الثقافة تقديم معلومات معينة مثل التواريخ، والعملة، بشكل صحيح.

على سبيل المثال:

- رمز الثقافة للمتحدث باللغة الإنجليزية في الولايات المتحدة الأمريكية هي `en-US`
- رمز الثقافة للمتحدث باللغة الفرنسيّة في فرنسا هو `fr-FR`
- رمز الثقافة للمتحدث باللغة الفرنسيّة في كندا هو `fr-CA`
- رمز الثقافة للمتحدث باللغة العربية في مصر هو `ar-eg`
- رمز الثقافة للمتحدث باللغة العربية في السعودية هو `ar-sa`

تؤثر الثقافة على نظام الكتابة، والتقويم المستخدم، وترتيب فرز السلاسل، وتنسيق التواريخ والأرقام (مثل تنسيق العملة)

لسوء الحظ، فإن التأكد من أن التعليمات البرمجية تعمل بشكل صحيح على جميع أجهزة الكمبيوتر، بغض النظر عن البلد/المنطقة أو لغة المستخدم النهائي أمر صعب، تُعرف هذه العملية باسم التوطين أو العولمة، تعتمد

الترجمة على العديد من العوامل التي لم تتم مناقشتها في هذه الوحدة، ولكن ببساطة، ربما يستخدم بناء جملة تنسيق السلسلة تنسيقاً مختلفاً اعتماداً على ثقافة المستخدم.

تنسيق الأرقام Formatting numbers

عند العمل مع البيانات الرقمية، ربما تحتاج إلى تنسيق الرقم لسهولة القراءة، عن طريق تضمين فواصل، لتحديد الآلاف، والملايين، والمليارات، وما إلى ذلك.

محدد التنسيق الرقمي `N` يجعل الأرقام أكثر قابلية للقراءة. حدث التعليمات البرمجية على النحو التالي:

```
decimal measurement = 123456.78912m;
Console.WriteLine($"Measurement:
{measurement:N} units");
```

إذا كنت تعرض هذا من ثقافة `en-US` فستلاحظ الإخراج التالي:

```
Measurement: 123,456.79 units
```

بشكل افتراضي، يعرض محدد التنسيق الرقمي `N` رقمين فقط بعد الفاصلة العشرية.

إذا كنت تريد عرض مزيد من الدقة، يمكنك إضافة رقم بعد المحدد `N` ستعرض التعليمة البرمجية التالية أربعة أرقام بعد الفاصلة العشرية باستخدام محدد `N4` حدث التعليمات البرمجية كما يلي:

```
decimal measurement = 123456.78912m;
Console.WriteLine($"Measurement:
{measurement:N4} units");
```

إذا كنت تعرض هذا من ثقافة `en-US` فستلاحظ الإخراج التالي:

```
Measurement: 123,456.7891 units
```

تنسيق النسب المئوية Formatting percentages

استخدم محدد التنسيق **P** لتنسيق النسبة المئوية، إضافة رقم بعد ذلك للتحكم في عدد القيم المعروضة بعد الفاصلة العشرية. حدث التعليمات البرمجية كما يلي:

```
decimal tax = .36785m;
Console.WriteLine($"Tax rate: {tax:P2}");
```

إذا كنت تعرض هذا من ثقافة `en-US` فستلاحظ الإخراج التالي:

```
Tax rate: 36.79 %
```

الجمع بين نهج التنسيق

يمكن لمتغيرات السلسلة تخزين السلاسل التي تم إنشاؤها، باستخدام تقنيات التنسيق، في المثال التالي، يتم تنسيق الأرقام العشرية، ونتائج الرياضيات العشرية، وتخزينها في متغير نصي `yourDiscount` باستخدام التنسيق المركب.

حدث التعليمات البرمجية كما يلي:

```
decimal price = 67.55m;
decimal salePrice = 59.99m;

string yourDiscount = String.Format("You saved
{0:C2} off the regular {1:C2} price. ", (price
- salePrice), price);

Console.WriteLine(yourDiscount);
```

إذا كنت تعرض هذا من ثقافة `en-US` فستلاحظ الإخراج التالي:

```
You saved $7.56 off the regular $67.55 price.
```

يمكنك الجمع بين سلاسل منسقة متعددة، قم بالبناء على التعليمات السابقة الذي يربط النسبة المئوية المحسوبة باستخدام استيفاء السلسلة، بدلاً من تنسيق النصوص أو تسلسل (ترتيب-تنسيق) السلسلة string concatenation عن طريق إدراج `yourDiscount += $"A discount of {((price - salePrice)/price:P2}!";` في سطر التعليمات البرمجية قبل `Console.WriteLine()`

ملاحظة

لا تحتاج إلى استخدام String.Format() مع نهج استيفاء السلسلة هذا.

حدث التعليمات البرمجية كما يلي:

```
decimal price = 67.55m;
decimal salePrice = 59.99m;

string yourDiscount = String.Format("You saved
{0:C2} off the regular {1:C2} price. ", (price
- salePrice), price);

yourDiscount += $"A discount of {((price -
salePrice)/price):P2}!"; //inserted
Console.WriteLine(yourDiscount);
```

إذا كنت تعرض هذا من ثقافة `en-US` فستلاحظ الإخراج التالي:

```
You saved $7.56 off the regular $67.55 price. A
discount of 11.19%!
```

الخلاصة

فيما يلي أهم الاستنتاجات من هذا الدرس حول تنسيق السلسلة:

- استخدم التنسيق المركب أو استيفاء السلسلة لتنسيق السلاسل أو النصوص.
- باستخدام **التنسيق المركب**، يمكنك استخدام قالب سلسلة يحتوي على واحد أو أكثر من الرموز المميزة البديلة في النموذج {0} كما أنك توفير قائمة بالوسائط التي تتم مطابقتها مع الرموز المميزة البديلة، بناءً على ترتيبها، يعمل التنسيق المركب عند استخدام `string.Format()` or `Console.WriteLine()`
- باستخدام **استيفاء السلسلة**، يمكنك استخدام قالب سلسلة يحتوي على أسماء المتغيرات التي تريد استبدالها محاطة بأقواس متعرجة، استخدم التوجيه $ قبل قالب السلسلة للإشارة إلى رغبتك في استيفاء أو استكمال السلسلة.
- تنسيق العملة باستخدام المحدد **C:**
- تنسيق الأرقام باستخدام المحدد **N:** التحكم في الدقة (عدد القيم بعد الفاصلة العشرية) باستخدام رقم بعد **N:** مثل `{myNumber:N3}`
- تنسيق النسب المئوية باستخدام محدد التنسيق **P:**
- يعتمد تنسيق العملة والأرقام على ثقافة المستخدم النهائي، وهو رمز مكون من خمسة أحرف يتضمن بلد/منطقة المستخدم ولغته (وفقاً للإعدادات الموجودة على جهاز الكمبيوتر الخاص به)

اختبر معلوماتك

١- ما هو ناتج `Console.WriteLine($"Tax rate: {tax:P1}");` حيث `tax` يتم تعريفه بواسطة `decimal tax =.12051 m;`

- `Tax rate: 12.05%`
- `Tax rate: 12.10%`
- `Tax rate: 12.1%`

راجع إجابتك

`Tax rate: 12.1%`

صحيح tax: P1 تقريب النسبة المئوية إلى منزلة عشرية واحدة

٣ تمرين استكشاف استيفاء السلسلة string interpolation

تحتاج إلى إنشاء تعليمات برمجية لطباعة إيصال للعميل بشراء أسهم منتج استثماري، يتم شراء الأسهم تلقائياً في نهاية العام، بناءً إلى سلسلة من استقطاعات الرواتب، وبالتالي فإن عدد الأسهم المشتراة عادةً ما يحتوي على مبلغ عشري. لطباعة الإيصال، من المحتمل أن تحتاج إلى دمج بيانات من أنواع مختلفة، بما في ذلك القيم الكسرية، والعملة، والنسب المئوية، بطرق دقيقة.

عرض رقم الفاتورة باستخدام استيفاء السلسلة

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
int invoiceNumber = 1201;
decimal productShares = 25.4568m;
decimal subtotal = 2750.00m;
decimal taxPercentage = .15825m;
decimal total = 3185.19m;

Console.WriteLine($"Invoice Number:
{invoiceNumber}");
```

٢. في قائمة ملف **File** Visual Studio Code حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
Invoice Number: 1201
```

ملاحظة

قد ترى عدة تحذيرات مثل `warning CS0219: The variable 'productShares' is assigned but its value is never used` لجميع المتغيرات التي تم تعريفها ولكن لم يتم استخدامها بعد، في التعليمات البرمجية.

عرض أسهم المنتج بدقة جزء من الألف من الحصة (0.001)

نظراً لأنك تقوم بفوترة العملاء باستخدام كسور من الأسهم، على الرغم من أن الدقة هي ١٠ آلاف (**0.0001**) فلن تعرض سوى ثلاثة أرقام بعد الفاصلة العشرية.

١. أضف التعليمات التالية أسفل التعليمات البرمجية التي كتبتها سابقاً:

```
Console.WriteLine($"   Shares: {productShares:N3}
Product");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
Invoice Number: 1201
   Shares: 25.457 Product
```

اعرض الإجمالي الفرعي الذي تفرضه على العميل بتنسيق العملة

١. أضف التعليمات التالية أسفل التعليمات البرمجية التي كتبتها في الخطوتين السابقتين:

```
Console.WriteLine($"     Sub Total:
{subtotal:C}");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
Invoice Number: 1201
Shares: 25.457 Product
    Sub Total: $2,750.00
```

ملاحظة

يعرض النموذج "$" ولكن قد ترى رمز عملة إقليمية مختلفًا.

عرض الضريبة المفروضة على عملية البيع بتنسيق النسبة مئوية

١. أضف التعليمات التالية أسفل التعليمات البرمجية التي كتبتها في الخطوات السابقة:

```
Console.WriteLine($"           Tax:
{taxPercentage:P2}");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
Invoice Number: 1201
Shares: 25.457 Product
    Sub Total: $2,750.00
        Tax: 15.83 %
```

إنهاء الإيصال بإجمالي المبلغ المستحق بتنسيق العملة

١. أضف التعليمات التالية أسفل التعليمات البرمجية التي كتبتها في الخطوات السابقة:

```
Console.WriteLine($"     Total Billed:
{total:C}");
```

٢. يجب أن تتطابق التعليمات البرمجية للتمرين بالكامل، كما يلي:

```
int invoiceNumber = 1201;
decimal productShares = 25.4568m;
decimal subtotal = 2750.00m;
decimal taxPercentage = .15825m;
decimal total = 3185.19m;

Console.WriteLine($"Invoice                Number:
{invoiceNumber}");
Console.WriteLine($"                       Shares:
{productShares:N3} Product");
Console.WriteLine($"                  Sub   Total:
{subtotal:C}");
Console.WriteLine($"                          Tax:
{taxPercentage:P2}");
Console.WriteLine($"               Total   Billed:
{total:C}");
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
Invoice Number: 1201
   Shares: 25.457 Product
     Sub Total: $2,750.00
           Tax: 15.83%
     Total Billed: $3,185.19
```

٤ تمرين ـ اكتشاف ترك المسافة البادئة padding والمحاذاة alignment

يتم استخدام الأسلوب `string.Format()` لتنفيذ تنسيق مركب composite formatting كما في المثال:

```
string first = "Hello";
string second = "World";
string result = string.Format("{0} {1}!",
first, second);
Console.WriteLine(result);
```

ربما يبدو غريبًا بعض الشيء أن الكلمة الأساسية التي تمثل نوع بيانات لديها أساليب، يمكنك استدعاءها بنفس طريقة استدعاء أساليب الفئة Console الحقيقة هي أن هناك العديد من الأساليب المشابهة لنوع البيانات string وأي سلسلة حرفية أو متغير من نوع string

إليك قائمة مختصرة بفئات هذه الأساليب المدمجة حتى تتمكن من الحصول على فكرة عما هو ممكن:

- أساليب تضيف مسافات فارغة، لأغراض التنسيق `PadLeft()`, `PadRight()`
- أساليب تقارن بين سلسلتين two strings أو تسهل المقارنة `Trim()`, `TrimStart()`, `TrimEnd()`, `GetHashcode()` والخاصية `Length`
- أساليب تساعدك على تحديد ما هو داخل `string` أو حتى استرداد جزء منها فقط `Contains()`, `StartsWith()`, `EndsWith()`, `Substring()`
- أساليب تغير محتوى `string` عن طريق استبدال أو إدراج أو إزالة أجزاء `Replace()`, `Insert()`, `Remove()`

- أساليب تقسم `string` إلى مصفوفة من السلاسل أو الأحرف `Split()`, `ToCharArray()`

تنسيق سلاسل Formatting strings عن طريق إضافة مسافة فارغة whitespace قبل أو بعد

يضيف الأسلوب `PadLeft()` مسافات فارغة إلى الجانب الأيسر من السلسلة string بحيث يساوي العدد الإجمالي للأحرف الوسيطة argument التي ترسلها، في هذه الحالة، تريد أن يكون الطول الإجمالي للسلسلة ١٢ حرفاً.

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
string input = "Pad this";
 Console.WriteLine(input.PadLeft(12));
```

٢. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

عند تشغيل التعليمات البرمجية، تلاحظ أربعة أحرف مسبوقة إلى يسار السلسلة، وبذلك يصل الطول إلى ١٢ حرفاً:

```
    Pad this
```

٥. لإضافة مسافة أو أحرف إلى الجانب الأيمن من السلسلة، استخدم الأسلوب `PadRight()` بدلاً من ذلك. حدث التعليمات البرمجية في المحرر كما يلي:

```
Console.WriteLine(input.PadRight(12));
```

٦. احفظ ملف التعليمات البرمجية، ثم شغلها. ربما لا تلاحظ أي أحرف تمت إضافتها إلى نهاية السلسلة، ولكنها موجودة.

```
    Pad this
Pad this
```

ما هو الأسلوب المحمل تحميلاً زائداً؟ overloaded method

في C# الأسلوب الزائد، هو إصدار آخر من أسلوب multiple versions مع وسيطات arguments مختلفة أو إضافية، تعدل وظيفة الأسلوب قليلاً، كما هو الحال مع الإصدار الزائد من الأسلوب `PadLeft()` يمكنك أيضًا استدعاء إصدار تحميل زائد ثانٍ من الأسلوب، وتمرير أي حرف تريد استخدامه بدلاً من مسافة، في هذه الحالة، يمكنك ملء المساحة الإضافية بحرف الشرطة.

١. حدث التعليمات من الخطوة السابقة في المحرر، كما يلي:

```
Console.WriteLine(input.PadLeft(12, '-'));
 Console.WriteLine(input.PadRight(12, '-'));
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها. يجب أن تشاهد أربع شرطات بادئة يسار السلسلة التي يبلغ طولها ١٢ حرفاً.

```
----Pad this
 Pad this----
```

الآن، طبق هذه المعرفة الجديدة على سيناريو آخر في العالم الحقيقي.

العمل مع سلاسل مبطنة padded strings

لنفترض أنك تعمل في شركة معالجة المدفوعات التي لا تزال تدعم أنظمة الكمبيوتر الرئيسي القديمة، وكثيرًا ما تتطلب تلك النظم إدخال البيانات في أعمدة محددة، على سبيل المثال، قم بتخزين معرف الدفع في الأعمدة من 1 إلى 6 واسم المستفيد في الأعمدة من 7 إلى 30 ومبلغ الدفع في الأعمدة من 31 إلى 40 والأهم من ذلك أيضاً، أن قيمة الدفع تتم محاذاتها إلى اليمين.

يطلب منك إنشاء تطبيق يقوم بتحويل البيانات في نظام إدارة قاعدة البيانات العلائقية relational database إلى تنسيق الملف القديم، للتأكد من أن التكامل يعمل بشكل صحيح، فإن الخطوة الأولى هي تأكيد تنسيق الملف، عن طريق إعطاء القائمين على صيانة النظام القديم عينة من الإخراج، لاحقاً، يمكنك البناء على هذا العمل لإرسال مئات أو آلاف المدفوعات لتتم معالجتها عبر ملف نصي ASCII

إضافة معرف الدفع Payment ID إلى الإخراج

للبدء، أطبع معرف الدفع في الأعمدة الستة الأولى، يمكنك اختيار بعض بيانات الدفع العشوائية التي يجب أن تكون كافية للأغراض.

١. أضف التعليمات البرمجية التالية في المحرر:

```
string paymentId = "769C";

 var formattedLine = paymentId.PadRight(6);

 Console.WriteLine(formattedLine);
```

أعد استخدام المتغير `formattedLine` لإنشاء سلسلة الإخراج.

٢. احفظ ملف التعليمات البرمجية، ثم شغلها. ينبغي أن تشاهد الإخراج التالي:

```
769C
```

هناك مسافتين فارغتان إلى اليمين غير مرئيتين، ستؤكد وجودها الخطوة التالية.

إضافة اسم المستفيد إلى الإخراج

بعد ذلك، يمكنك إضافة اسم المستفيد الوهمي، مع إضافة المساحة أو التبطين padding إليه بشكل مناسب.

١. حدث التعليمات كما يلي:

```
string paymentId = "769C";
 string payeeName = "Mr. Stephen Ortega";

 var formattedLine = paymentId.PadRight(6);
 formattedLine += payeeName.PadRight(24);

 Console.WriteLine(formattedLine);
```

يقوم عامل التشغيل operator `+=` بدمج سلسلة، مع أخذ القيمة السابقة للمتغير formattedLine وإضافة القيمة الجديدة إليه، إنه مكافئ مختصر لمثال التعليمات البرمجية التالي:

```
formattedLine = formattedLine + payeeName.PadRight(24);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
769C  Mr. Stephen Ortega
```

مرة أخرى، هناك عدد غير قليل من المساحات الفارغة بعد المستفيد الوهمي Payee's Name أيضاً، هناك مسافات فارغة بعد معرف الدفع Payment ID من الخطوة 1

إضافة مبلغ الدفع إلى الإخراج

بعد ذلك، أضف مبلغ دفع وهمي، وتأكد من استخدام PadLeft() لمحاذاة الإخراج إلى اليمين.

١. حدث التعليمات كما يلي:

```
string paymentId = "769C";
string payeeName = "Mr. Stephen Ortega";
string paymentAmount = "$5,000.00";

 var formattedLine = paymentId.PadRight(6);
 formattedLine += payeeName.PadRight(24);
 formattedLine += paymentAmount.PadLeft(10);

 Console.WriteLine(formattedLine);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
769C   Mr. Stephen Ortega        $5,000.00
```

أضف سطرًا من الأرقام أعلى الإخراج لتأكيد النتيجة بسهولة أكبر

نظرًا لصعوبة حساب الأعمدة الدقيقة، التي يظهر فيها كل عنصر بيانات، يمكنك إضافة سطر أعلى المخرجات مباشرة مما يساعدك على حساب الأعمدة.

```
Console.WriteLine("123456789012345678901234567890");
```

١. حدث التعليمات كما يلي:

```
 string paymentId = "769C";
 string payeeName = "Mr. Stephen Ortega";
 string paymentAmount = "$5,000.00";

 var formattedLine = paymentId.PadRight(6);
 formattedLine += payeeName.PadRight(24);
 formattedLine += paymentAmount.PadLeft(10);

Console.WriteLine("123456789012345678901234567890");
Console.WriteLine(formattedLine);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي، الذي يمكنك إرساله إلى القائمين على صيانة النظام القديم، لتأكيد عمل التكامل الجديد بشكل صحيح.

```
12345678901234567890123456789012345678901234567890
 769C  Mr. Stephen Ortega          $5,000.00
```

عمل ناجح

الخلاصة

هناك بعض الاستنتاجات الهامة من هذا الدرس:

- يقوم كل من نوع بيانات string والسلاسل الحرفية والمتغيرات من نوع string بتنفيذ العديد من الأساليب المساعدة، لتنسيق، وتعديل، وتنفيذ عمليات أخرى، على strings النصوص.
- تضيف الأساليب `PadLeft()`, `PadRight()` مسافة فارغة (أو اختياريًا، حرفًا آخر) إلى الطول الإجمالي لسلسلة "النصوص"
- يُستخدم `PadLeft()` لمحاذاة سلسلة إلى اليمين.
- بعض أساليب التحميل الزائد `overloaded` تعني أن لديهم إصدارات متعددة من الأسلوب، مع وسائط مختلفة تؤثر على وظائفها.
- يقوم عامل التشغيل `+=` بربط سلسلة جديدة على اليمين بالسلسلة الموجودة على اليسار.

اختبر معلوماتك

بالنظر إلى `string myWords = "Learning C#"` ما أفضل وصف إخراج ل `Console.WriteLine(myWords.PadLeft(12));`

- تتم إضافة مسافة واحدة إلى بداية السلسلة.
- تتم إضافة أربع مسافات إلى بداية السلسلة.
- تتم إضافة ١٢ مسافات إلى بداية السلسلة.

راجع إجابتك

تتم إضافة مسافة واحدة إلى بداية السلسلة

صحيح myWords يبلغ طوله ١١ حرفاً، وتكتمل إضافة المساحة إلى ١٢ حرفاً.

٥ تمرين ـ تحدي لتطبيق استيفاء السلسلة على خطاب نموذجي

بالنسبة لأحدث المنتجات الاستثمارية لشركة المبيعات والتسويق، يمكنك إرسال آلاف الرسائل الشخصية إلى عملاء الشركة الحاليين، مهمتك هي كتابة التعليمات البرمجية C# لدمج المعلومات الشخصية عن العميل، تحتوي الرسالة على معلومات مثل قائمة مشاريعهم الحالية، وتقارن عائداتهم الحالية بالعائدات المتوقعة، إذا كانوا سيستثمرون في استخدام المنتجات الجديدة.

قرر الكاتب استخدام المثال التالي للرسالة التسويقية. إليك النتيجة المطلوبة (باستخدام بيانات حساب العميل الوهمية).

```
Dear Ms. Barros,
As a customer of our Magic Yield offering we
are excited to tell you about a new financial
product that would dramatically increase your
return.

Currently, you own 2,975,000.00 shares at a
return of 12.75%.

Our new product, Glorious Future offers a
return of 13.13%.  Given your current volume,
your potential profit would be ¤63,000,000.00.

Here's a quick comparison:

Magic Yield          12.75%   $55,000,000.00
Glorious Future      13.13%   $63,000,000.00
```

استخدم معرفتك الجديدة التي اكتسبتها لتنسيق النصوص، لإنشاء تطبيق يمكنه دمج المحتوى المناسب، وتنسيقه، بالنظر إلى إخراج المثال السابق، إيلاء اهتمام خاص بالمسافة البيضاء، وتأكد من أنك تمثل بدقة هذا التنسيق الدقيق باستخدام C#

١. حدد كافة أسطر التعليمات البرمجية واحذفها في محرر Visual Studio

٢. أضف التعليمات البرمجية التالية للحصول على بيانات التحدي:

```
string customerName = "Ms. Barros";

string currentProduct = "Magic Yield";
int currentShares = 2975000;
decimal currentReturn = 0.1275m;
decimal currentProfit = 55000000.0m;

string newProduct = "Glorious Future";
decimal newReturn = 0.13125m;
decimal newProfit = 63000000.0m;

// Your logic here

Console.WriteLine("Here's a quick
comparison:\n");

string comparisonMessage = "";

// Your logic here

Console.WriteLine(comparisonMessage);
```

٣. استخدم محرر التعليمات البرمجية Visual Studio لإنشاء الرسالة أثناء استخدام المتغيرات والرمز المحددين.

لا يجوز لك حذف أي من التعليمات البرمجية الموجودة باستثناء التعليقات.

٤. تأكد من إخراج التعليمات البرمجية للرسالة كالتالي:

```
Dear Ms. Barros,
As a customer of our Magic Yield offering we
are excited to tell you about a new financial
product that would dramatically increase your
return.

Currently, you own 2,975,000.00 shares at a
return of 12.75%.

Our new product, Glorious Future offers a
return of 13.13%.  Given your current volume,
your potential profit would be ¤63,000,000.00.

Here's a quick comparison:

Magic Yield          12.75%   $55,000,000.00
Glorious Future      13.13%   $63,000,000.00
```

حظ سعيد!

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

٦ مراجعة حل تحدي استيفاء السلسلة

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق.

```
string customerName = "Ms. Barros";

string currentProduct = "Magic Yield";
int currentShares = 2975000;
decimal currentReturn = 0.1275m;
decimal currentProfit = 55000000.0m;

string newProduct = "Glorious Future";
decimal newReturn = 0.13125m;
decimal newProfit = 63000000.0m;

Console.WriteLine($"Dear {customerName},");
Console.WriteLine($"As a customer of our
{currentProduct} offering we are excited to
tell you about a new financial product that
would dramatically increase your return.\n");
Console.WriteLine($"Currently, you own
{currentShares:N} shares at a return of
{currentReturn:P}.\n");
Console.WriteLine($"Our new product,
{newProduct} offers a return of {newReturn:P}.
Given your current volume, your potential
profit would be {newProfit:C}.\n");

Console.WriteLine("Here's a quick
comparison:\n");

string comparisonMessage = "";
```

```
comparisonMessage =
currentProduct.PadRight(20);
comparisonMessage += String.Format("{0:P}",
currentReturn).PadRight(10);
comparisonMessage += String.Format("{0:C}",
currentProfit).PadRight(20);

comparisonMessage += "\n";
comparisonMessage += newProduct.PadRight(20);
comparisonMessage += String.Format("{0:P}",
newReturn).PadRight(10);
comparisonMessage += String.Format("{0:C}",
newProfit).PadRight(20);

Console.WriteLine(comparisonMessage);
```

إن هذه التعليمة البرمجية هي مجرد حل واحد محتمل، لأن الكثير من الحلول تعتمد على الطريقة التي قررت بها تنفيذ المنطق، طالما استخدمت التقنيات التي تغطيها هذه الوحدة لتنسيق السلاسل النصية format strings وتبطين السلاسل pad strings وما إلى ذلك، والنتيجة تطابق مخرجات التحدي، فأنت قمت بعمل رائع!

تفاصيل الحل

يمكنك متابعة هذا القسم للحصول على شرح الحل المعطى لهذا التحدي.

١. خذ دقيقة لمراجعة رمز الحل.

يمكنك البدء في تقسيم الحل، الخطوة الأولى كتابة فقرة الترحيب وعرضها في المحطة الطرفية، التعليمات البرمجية التالية تحل عرض الترحيب Dear Ms. Barros, باستخدام استنتاج السلسلة string interpolation يجب أن

تكون على دراية الآن بالنمط `Console.WriteLine($"Your text {yourVariable}");`

```
string customerName = "Ms. Barros";
Console.WriteLine($"Dear {customerName},");
```

إخراج التعليمات البرمجية هو:

```
Dear Ms. Barros,
```

راجع نموذج الحل الكامل مرة أخرى، يستخدم النصف الأول من الحل استنتاج السلسلة string interpolation لعرض كل جزء من الفقرة الأولى.

ملاحظة: يعد التنسيق المركب حلاً آخر ممكنًا، مثل `Console.WriteLine("Dear {0},", customerName)`

٢. الخطوة الثانية من الحل تعرض جدول المقارنة، عن طريق إنشاء سلسلة طويلة، خطوة بخطوة باستخدام تسلسل السلسلة string concatenation "تنسيق النصوص" و `string.Format()` مع التنسيق المركب composite formatting ومحددات التنسيق (النسبة المئوية والعملة) و `PadRight()`

تنشئ التعليمة البرمجية التالية السطر الأول من الجدول مع إضافات Console.WriteLine() بعد كل خطوة من خطوات إنشاء السلسلة comparisonMessage

```
string currentProduct = "Magic Yield";
int currentShares = 2975000;
decimal currentReturn = 0.1275m;
decimal currentProfit = 55000000.0m;
string comparisonMessage = "";

comparisonMessage =
currentProduct.PadRight(20);
Console.WriteLine(comparisonMessage);
```

```
comparisonMessage += String.Format("{0:P}",
currentReturn).PadRight(10);
Console.WriteLine(comparisonMessage);

comparisonMessage += String.Format("{0:C}",
currentProfit).PadRight(20);
Console.WriteLine(comparisonMessage);
```

يوضح إخراج النموذج التالي كيفية إنشاء السطر الأول من جدول المقارنة في ثلاث خطوات.

```
Magic Yield
Magic Yield          12.75%
Magic Yield          12.75%    $55,000,000.00
```

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ اختبر معلوماتك

١- بالنسبة للتعليمات البرمجية التالية، ما هو الإخراج المتوقع؟

```
Console.WriteLine("C110".PadLeft(6, '0'));
```

- `C11000`
- `C110000000`
- `00C110`

٢- محدد التنسيق الذي يضيف قيمة عشرية بالتنسيق التالي، للجمهور في الولايات المتحدة `12,345.67`

- `0:C`
- `0:H`
- `0:N2`

٣- ما الحرف الذي يجب استخدامه كتوجيه لتنفيذ استيفاء السلسلة؟

- `$`
- `@`
- `%`

ر اجع إجابتك

١

`00C110`

صحيح يلحق `.PadLeft(6, '0')` الأصفار بالجانب الأيسر من السلسلة، حتى تصل السلسلة إلى ستة أحرف

٢

`0:N2`

صحيح N2 هو محدد التنسيق الصحيح

٣

`$`

صحيح يتم استخدام التوجيه $ لتنفيذ استنتاج السلسلة

٨ الخلاصة

تحتاج شركتك إلى إنشاء رسالة تسويق مخصصة للترويج لمنتج مالي جديد لعملائك، باستخدام C# تمكنت من كتابة التعليمات البرمجية التي تجمع بين النصوص، والبيانات الرقمية، حول عملائك الحاليين.

مارست تقنيات مختلفة لدمج قالب نصي مع متغيرات باستخدام التنسيق المركب واستيفاء أو استكمال السلسلة.

تسمح لك المحددات المختلفة بتنسيق الأرقام الكبيرة، والعملة، والنسب المئوية، بشكل صحيح.

سمحت لك الأساليب المضمنة في متغيرات النوع string بإضافة padding مساحة بادئة لمحاذاة البيانات إلى اليسار واليمين.

تخيل مقدار العمل الإضافي المطلوب إذا لم يكن لديك ثروة من أدوات، وتقنيات تنسيق النصوص، والأرقام المتوفرة لك في C# من المحتمل أن تحاول استخدام دمج النصوص في كل مكان، مما قد يجعل من الصعب صيانتها أو تحديثها.

تقنيات وأدوات تنسيق السلاسل والبيانات الرقمية لديها مجموعة واسعة من التطبيقات. على سبيل المثال، يمكنك استخدام هذه التقنيات لتقديم البيانات للعرض على الشاشة وتنسيق البيانات لنقلها بين الأنظمة المختلفة.

المصادر:

- String.Format Method
- Standard Numeric Format Strings

الوحدة الخامسة

تعديل محتوى الجمل النصية باستخدام الأساليب المساعدة "مُدمجة" لبيانات string

استخدم أساليب المساعدة، المُدمجة، لاستخراج البيانات، أو إزالتها أو استبدالها في السلاسل.

الأهداف التعليمية

- تحديد موضع حرف character أو نص string داخل نص string أخر.
- استخراج أجزاء من الجمل النصية strings
- إزالة أجزاء من الجمل النصية strings
- استبدال القيم في الجمل النصية strings بقيم مختلفة

محتويات الوحدة:

١ المقدمة

لنفترض أنك مطور لتطبيق، يسمح لشركة ما بتحديث موقع ويب "صفقات الفرصة الأخيرة" الخاص بها، عن طريق إرسال بريد إلكتروني، يستخدم البريد الإلكتروني الخاص بالتحديث نصًا خاصًا مطلوبًا في نص وعنوان البريد الإلكتروني، لإرشاد عملية التشغيل الآلي إلى كيفية تحديث موقع الويب، يتضمن البريد عنوان الصفقة التالية، والخصم٪ وانتهاء الصلاحية، ومتى يتم نشر العرض مباشرة.

في كثير من الأحيان، تكون بيانات التطبيق التي تحتاج إلى العمل معها من أنظمة برامج أخرى، وتحتوي على بيانات لا تريدها أو تحتاج إليها، وفي بعض الأحيان تكون البيانات بتنسيق غير قابل للاستخدام، وتحتوي على معلومات إضافية تجعل من الصعب استخراج المعلومات المهمة، لضبط البيانات لتطبيقك، تحتاج إلى أدوات وتقنيات لتحليل بيانات السلسلة، وعزل المعلومات التي تحتاجها، وإزالة المعلومات التي لا تحتاج إليها.

في هذه الوحدة، يمكنك استخدام الأساليب المساعدة ل string لتحديد المعلومات الهامة وعزلها، ستتعلم كيفية نسخ جزء أصغر من نص أكبر، ويمكنك استبدال الأحرف أو إزالة الأحرف من نص string

في نهاية هذه الوحدة، يمكنك تعديل محتويات string أو عزل أجزاء معينة لاستخراجها أو استبدالها أو إزالتها.

٢ استخدم أساليب المساعدة IndexOf() و Substring() للسلسلة

في هذا التمرين، يمكنك استخدام الأسلوب IndexOf() لتحديد موقع سلسلة أحرف واحدة أو أكثر داخل سلسلة نصية أكبر، يمكنك استخدام الأسلوب Substring() لإرجاع جزء السلسلة الأكبر، الذي يتبع مواضع الأحرف التي تحددها.

ستستخدم أيضاً إصداراً محملاً بشكل زائد من الأسلوب Substring() لتعيين طول الأحرف، لإرجاعها بعد موضع محدد في سلسلة نصية.

إعداد بيئة الترميز

١. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة ملف **File** حدد فتح مجلد **Open Folder**

٣. في **مربع الحوار فتح مجلد**، انتقل إلى مجلد سطح مكتب Windows

إذا كان لديك موقع مجلد مختلف حيث تحتفظ بمشاريع التعليمات البرمجية، يمكنك استخدام هذا المجلد لهذا التدريب، الشيء المهم هو أن يكون لديك موقع يسهل تحديد موقعه وتذكره.

٤. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

٥. في قائمة **Terminal** Visual Studio Code حدد **New Terminal**

لاحظ أن موجه الأوامر في لوحة Terminal يعرض مسار المجلد الحالي. على سبيل المثال:

```
C:\Users\someuser\Desktop>
```

٦. لإنشاء تطبيق وحدة تحكم جديد في مجلد محدد، اكتب في موجه Terminal: الأوامر **dotnet new console -o ./CsharpProjects/TestProject** ثم اضغط على Enter

يستخدم أمر .NET CLI. هذا قالب برنامج .NET. لإنشاء مشروع تطبيق وحدة تحكم C# جديد في موقع المجلد المحدد. ينشئ الأمر نيابة عنك مجلدات CsharpProjects, TestProject ويستخدم TestProject كاسم للملف csproj. أو كامتداد له.

٧. في قائمة استكشاف EXPLORER قم **بتوسيع المجلد CsharpProjects**

يجب أن تشاهد مجلد TestProject وملفين، ملف برنامج C# المسمى Program.cs وملف مشروع C# يسمى TestProject.csproj

٨. في قائمة استكشاف EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد **Program.cs**

٩. حذف أسطر التعليمات البرمجية الموجودة.

يمكنك استخدام مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها وتشغيلها أثناء هذه الوحدة.

١٠. أغلق Terminal

كتابة التعليمات البرمجية للعثور على الأقواس الموجودة داخل سلسلة نصية

١. تأكد من فتح Visual Studio Code وعرضه Program.cs في لوحة المحرر

٢. اكتب التعليمات البرمجية التالية في المحرر:

```
string message = "Find what is (inside the
parentheses)";
```

```
int openingPosition = message.IndexOf('(');
int closingPosition = message.IndexOf(')');

Console.WriteLine(openingPosition);
Console.WriteLine(closingPosition);
```

٣. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٤. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٥. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
13
36
```

في هذه الحالة، يكون فهرس) الحرف 13 تذكر، تستند هذه القيم إلى الصفر، لذلك ترتيبه الحرف 14 في السلسلة، فهرس (الحرف هو 36

الآن لديك فهرسين يمكنك استخدامهما كحدود لاسترداد القيمة بينهما.

إضافة تعليمة برمجية لاسترداد القيمة بين الأقواس

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "Find what is (inside the
parentheses)";

int openingPosition = message.IndexOf('(');
int closingPosition = message.IndexOf(')');

// Console.WriteLine(openingPosition);
// Console.WriteLine(closingPosition);

int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosi
tion, length));
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
(inside the parentheses
```

يحتاج الأسلوب `Substring()` إلى موضع البداية، وعدد الأحرف أو الطول لاسترداد القيمة، لذلك، يمكنك حساب الطول في متغير مؤقت يسمى length وتمريره بالقيمة openingPosition لاسترداد السلسلة داخل القوس.

النتيجة قريبة، ولكن الناتج يتضمن قوس الفتح، في هذا التمرين، ليس من المرغوب فيه إدراج الأقواس، لإزالة القوس من الإخراج، يجب عليك تحديث التعليمات البرمجية لتخطي فهرس القوس نفسه.

تعديل موضع البداية للسلسلة الفرعية sub string

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "Find what is (inside the
parentheses)";

int openingPosition = message.IndexOf('(');
```

```
int closingPosition = message.IndexOf(')');

openingPosition += 1;

int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosi
tion, length));
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
inside the parentheses
```

٣. خذ لحظة لمراجعة التعليمات البرمجية السابقة والسطر `openingPosition += 1;` بزيادة openingPosition بمقدار 1 فإنك تتخطى حرف قوس الفتح.

السبب في استخدامك للقيمة 1 هو أن هذا هو طول الحرف، إذا حاولت تحديد موقع قيمة، تبدأ بعد سلسلة أطول، على سبيل المثال `<div>` or `---` فيمكنك استخدام طول تلك السلسلة بدلاً من ذلك.

٤. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "What is the value
<span>between the tags</span>?";

int openingPosition = message.IndexOf("<span>");
int closingPosition = message.IndexOf("</span>");

openingPosition += 6;
int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosi
tion, length));
```

٥. خذ لحظة لمراجعة التعليمات البرمجية السابقة والسطر `openingPosition += 6;`

يوضح مقتطف التعليمات البرمجية السابق، كيفية العثور على القيمة، داخل علامة فتح وإغلاق

في هذه الحالة، تقوم بإضافة 6 إلى `openingPosition` كإزاحة لحساب طول السلسلة الفرعية.

تجنب القيم السحرية Avoid magic values

تُعرف السلاسل المشفرة Hardcoded strings مثل "" في قائمة التعليمات البرمجية السابقة باسم "السلاسل السحرية" magic strings والقيم الرقمية المشفرة مثل 6 تُعرف باسم "الأرقام السحرية" magic numbers هذه القيم "السحرية" غير مرغوب فيها لعدة أسباب ويجب عليك محاولة تجنبها إن أمكن.

١. راجع التعليمات البرمجية السابقة للنظر في كيفية كسر التعليمات البرمجية، إذا قمت بترميز السلسلة "" عدة مرات في التعليمات البرمجية الخاصة بك، ولكن أخطأت في كتابة مثيل واحد منها كـ "<sapn>".

لا يلتقط المترجم "<sapn>" في وقت التحويل البرمجي، لأن القيمة موجودة في سلسلة، يؤدي الخطأ الإملائي إلى حدوث مشكلات في وقت التشغيل، واستنادًا إلى مدى تعقيد التعليمات البرمجية الخاصة بك، قد يكون من الصعب تعقبها.

علاوة على ذلك، إذا قمت بتغيير السلسلة "" إلى " " الأقصر، ونسيت تغيير الرقم 6 إلى 5 فإن التعليمات البرمجية الخاصة بك تنتج نتائج غير مرغوب فيها.

٢. حدث التعليمات البرمجية في المحرر، كما يلي:

```
string message = "What is the value
<span>between the tags</span>?";

const string openSpan = "<span>";
const string closeSpan = "</span>";

int openingPosition = message.IndexOf(openSpan);
```

```
int closingPosition = message.IndexOf(closeSpan);

openingPosition += openSpan.Length;
int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosition, length));
```

٣. خذ دقيقة لفحص التعليمات البرمجية المحدثة، واستخدام الكلمة الأساسية const كما هو مستخدم في `const string openSpan = "<span>";`

تستخدم التعليمات ثابتًا مع الكلمة الأساسية const تتيح لك القيمة الثابتة بتعريف متغير وتهيئته، هذا المتغير لا يمكن تغيير قيمته أبدًا، حينئذٍ، يمكنك استخدام هذه القيمة الثابتة في بقية التعليمات البرمجية كلما احتجت إلى تلك القيمة، وهذا يضمن أن يتم تعريف القيمة مرة واحدة فقط، وأن المترجم يلتقط الخطأ الإملائي لمتغير const

تعد قائمة التعليمات البرمجية السابقة طريقة أكثر أمانًا لكتابة نفس التعليمات البرمجية التي قمت بفحصها في القسم السابق. الآن، إذا تغيرت قيمة openSpan إلى فسيظل سطر التعليمات البرمجية الذي يستخدم خاصية الطول صالحًا.

خلاصة

هذا الدرس غطى الكثير من المواد، إليك أهم الأشياء التي يجب تذكرها:

- يمنحك IndexOf() الموضع الأول لحرف أو نص داخل سلسلة نصية أخرى.
- يقوم IndexOf() بإرجاع -1 إذا لم يتمكن من العثور على تطابق.
- ترجع Substring() فقط الجزء المحدد من سلسلة، باستخدام موضع بدء وطول اختياري.

- هناك في كثير من الأحيان أكثر من طريقة واحدة لحل مشكلة، لقد استخدمت تقنيتين منفصلتين، للعثور على جميع مثيلات حرف أو سلسلة معينة.
- تجنب القيم السحرية المشفرة magic values بدلاً من ذلك، حدد متغير const لا يمكن تغيير قيمته الثابتة بعد التهيئة.

اختبر معلوماتك

ما هي القيمة المرجعة ل `myString.IndexOf('C');` عندما تكون `string myString = "C# Time";`

- `0`
- `1`
- `-1`

راجع إجابتك

0

صحيح "C" هو العنصر الأول في myString ويتطابق مع الموضع 0

٣ استخدم أساليب المساعدة LastIndexOf() و IndexOfAny() للسلسلة

في هذا التمرين، يمكنك استخدام الأسلوب IndexOfAny() للعثور على الموقع الأول لأي نص string من مصفوفة محددة، يمكنك أيضاً استخدام LastIndexOf() للعثور على الموقع النهائي لنص داخل نص string آخر.

استرداد آخر تكرار لسلسلة فرعية

يمكنك زيادة تعقيد المتغير message عن طريق إضافة العديد من مجموعات الأقواس، ثم كتابة التعليمات البرمجية لاسترداد المحتوى داخل المجموعة الأخيرة من الأقواس.

١. حدد كافة أسطر التعليمات البرمجية في المحرر واحذفها.

٢. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "(What if) I am (only
interested) in the last (set of parentheses)?";
int openingPosition = message.LastIndexOf('(');

openingPosition += 1;
int closingPosition = message.LastIndexOf(')');
int length = closingPosition - openingPosition;
Console.WriteLine(message.Substring(openingPosi
tion, length));
```

٣. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
set of parentheses
```

استرداد جميع مثيلات instances السلاسل الفرعية داخل الأقواس

هذه المرة، قم بتحديث message ليكون لديك ثلاث مجموعات من الأقواس، واكتب التعليمات البرمجية لاستخراج أي نص داخل الأقواس، يمكنك إعادة استخدام أجزاء من العمل السابق، ولكن تحتاج إلى إضافة عبارة while للتكرار عبر السلسلة، حتى يتم اكتشاف جميع مجموعات الأقواس واستخراجها وعرضها.

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "(What if) there are (more
than) one (set of parentheses)?";
while (true)
{
    int openingPosition = message.IndexOf('(');
    if (openingPosition == -1) break;

    openingPosition += 1;
    int closingPosition = message.IndexOf(')');
    int length = closingPosition -
openingPosition;

Console.WriteLine(message.Substring(openingPosi
tion, length));

    // Note the overload of the Substring to
return only the remaining
    // unprocessed message:
    message = message.Substring(closingPosition
+ 1);
}
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
What if
more than
set of parentheses
```

٣. خذ دقيقة لفحص السطر الأخير من التعليمات البرمجية داخل الحلقة while المستخرج من التعليمات التالية:

```
message = message.Substring(closingPosition + 1);
```

عند استخدام Substring() دون تحديد معلمة إدخال الطول length input parameter فإنها ستعيد كل حرف بعد موضع البداية الذي تحدده، مع السلسلة التي تتم معالجتها `message = "(What if) there are (more than) one (set of parentheses)?"` هناك ميزة لإزالة المجموعة الأولى من الأقواس (What if) من قيمة message تتم معالجة ما تبقى بعد ذلك في التكرار التالي للحلقة while

٤. خذ دقيقة للتفكير فيما يحدث أثناء التكرار النهائي للحلقة while عندما يبقى الحرف النهائي **?** فقط.

عناوين التعليمات البرمجية التالية التعامل مع نهاية السلسلة:

```
int openingPosition = message.IndexOf('(');
if (openingPosition == -1) break;
```

يقوم الأسلوب IndexOf() بإرجاع `-1` إذا لم تتمكن من العثور على معلمة الإدخال في السلسلة، ما عليك سوى التحقق من القيمة `-1` والخروج من الحلقة `break`

العمل مع أنواع مختلفة من مجموعات الرموز symbol sets

هذه المرة، ابحث عن عدة رموز مختلفة، وليس مجموعة من الأقواس.

قم بتحديث message بإضافة أنواع مختلفة من الرموز مثل الأقواس المربعة [] والأقواس المتعرجة {} للبحث عن رموز متعددة في وقت واحد، استخدم IndexOfAny() يمكنك البحث باستخدام IndexOfAny()

لإرجاع فهرس الرمز الأول من المصفوفة openSymbols الموجود في سلسلة message

١. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "Help (find) the {opening
symbols}";
Console.WriteLine($"Searching THIS Message:
{message}");
char[] openSymbols = { '[', '{', '(' };
int startPosition = 5;
int openingPosition =
message.IndexOfAny(openSymbols);
Console.WriteLine($"Found WITHOUT using
startPosition:
{message.Substring(openingPosition)}");

openingPosition = message.IndexOfAny(openSymbols,
startPosition);
Console.WriteLine($"Found       WITH       using
startPosition                    {startPosition}:
{message.Substring(openingPosition)}");
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
Searching THIS message: Help (find) the {opening symbols}
Found WITHOUT using startPosition: (find) the {opening symbols}
Found WITH using startPosition 5:  (find) the {opening symbols}
```

٣. خذ دقيقة لمراجعة التعليمات البرمجية التي تم إدخالها مسبقاً.

لقد استخدمت IndexOfAny() بدون التحميل الزائد لموضع البداية.

الآن بعد أن عثرت على رمز الفتح، تحتاج إلى العثور على رمز الإغلاق المطابق له.

٤. حدث التعليمات البرمجية في المحرر كما يلي:

```
string message = "(What if) I have [different
symbols] but every {open symbol} needs a
[matching closing symbol]?";

// The IndexOfAny() helper method requires a
char array of characters.
// You want to look for:

char[] openSymbols = { '[', '{', '(' };

// You'll use a slightly different technique
for iterating through
// the characters in the string. This time, use
the closing
// position of the previous iteration as the
starting index for the
//next open symbol. So, you need to initialize
the closingPosition
// variable to zero:

int closingPosition = 0;

while (true)
{
    int openingPosition =
message.IndexOfAny(openSymbols,
closingPosition);

    if (openingPosition == -1) break;

    string currentSymbol =
message.Substring(openingPosition, 1);

    // Now  find the matching closing symbol
    char matchingSymbol = ' ';
```

```
    switch (currentSymbol)
    {
        case "[":
            matchingSymbol = ']';
            break;
        case "{":
            matchingSymbol = '}';
            break;
        case "(":
            matchingSymbol = ')';
            break;
    }

    // To find the closingPosition, use an
overload of the IndexOf method to specify
    // that the search for the matchingSymbol
should start at the openingPosition in the
string.

    openingPosition += 1;
    closingPosition =
message.IndexOf(matchingSymbol,
openingPosition);

    // Finally, use the techniques you've
already learned to display the sub-string:

    int length = closingPosition -
openingPosition;

Console.WriteLine(message.Substring(openingPosi
tion, length));
}
```

٥. خذ بضع دقائق لفحص التعليمات البرمجية السابقة وقراءة التعليقات التي تساعد في شرح التعليمات البرمجية.

٦. تابع الفحص، وحدد موقع السطر التالي من التعليمات باستخدام `IndexOf()` لتحديد موقع الإغلاق `closingPosition`

```
closingPosition = message.IndexOf(matchingSymbol,
openingPosition);
```

يتم استخدام المتغير `closingPosition` للعثور على الطول الذي تم تمريره إلى الأسلوب `Substring()` وللبحث عن القيمة التالية `openingPosition`

```
int openingPosition =
message.IndexOfAny(openSymbols, closingPosition);
```

لهذا السبب، يتم تعريف المتغير closingPosition خارج نطاق حلقة التكرار while وتهيئته إلى 0 للتكرار الأول.

٧. احفظ ملف التعليمات البرمجية، ثم شغلها، ينبغي أن تشاهد الإخراج التالي:

```
What if
different symbols
open symbol
matching closing symbol
```

الخلاصة

فيما يلي أمران مهمان يجب تذكرهما:

- يرجع `LastIndexOf()` الموضع الأخير لحرف أو سلسلة داخل سلسلة نصية أخرى.
- يرجع `IndexOfAny()` الموضع الأول لمصفوفة من الأحرف `char` التي تحدث داخل سلسلة string أخرى.

اختبر معلوماتك

ما الأسلوب الذي يجب استخدامه للبحث عن الموضع الأول لمصطلح البحث في سلسلة طويلة؟

- `IndexOfAny()`
- `LastIndexOf()`
- `Substring()`

راجع إجابتك

`IndexOfAny()`

صحيح يقوم IndexOfAny() بإرجاع الموضع الأول لمصفوفة من الأحرف التي تحدث داخل سلسلة أخرى.

٤ استخدم أساليب المساعدة Remove() و Replace()

في هذا التمرين، يمكنك إزالة الأحرف من سلسلة نصية، باستخدام الأسلوب Remove() واستبدال الأحرف باستخدام الأسلوب Replace()

في بعض الأحيان، تحتاج إلى تعديل محتويات سلسلة نصية، أو إزالة الأحرف أو استبدالها، على الرغم من أنه يمكنك استبدال الأحرف بالأدوات التي تعرفها بالفعل، إلا أن ذلك يتطلب القليل من تخزين السلاسل وربطها معًا بشكل مؤقت، لحسن الحظ، يحتوي نوع البيانات string على أساليب مضمنة أخرى `Remove()` and `Replace()` لهذه السيناريوهات الخاصة.

استخدام الأسلوب Remove()

عادة ما تستخدم Remove() عندما يكون هناك موضع قياسي ومتسق للأحرف التي تريد إزالتها من السلسلة.

يحتوي هذا التمرين على بيانات مخزنة في ملفات قديمة، ذات طول ثابت، ومع مواضع أحرف مخصصة لحقول معينة من المعلومات، تمثل الأرقام الخمسة الأولى رقم تعريف العميل، تحتوي الأرقام العشرين التالية على اسم العميل، تمثل المراكز الستة التالية آخر مبلغ فاتورة للعميل، وتمثل المراكز الثلاثة الأخيرة عدد الأصناف المطلوبة في تلك الفاتورة.

في الخطوات التالية، تحتاج إلى إزالة اسم العميل لتنسيق البيانات، بحيث يمكن إرسالها إلى عملية منفصلة، نظراً لأنك تعرف الموضع الدقيق وطول اسم المستخدم، يمكنك إزالته بسهولة باستخدام الأسلوب Remove()

إزالة الأحرف في مواقع محددة من السلسلة

١. حدث التعليمات البرمجية في المحرر، كما يلي:

```
string data = "12345John Smith         5000  3  ";
string updatedData = data.Remove(5, 20);
Console.WriteLine(updatedData);
```

٢. في قائمة ملف Visual Studio Code **File** حدد حفظ **Save**

يجب حفظ ملف Program.cs قبل إنشاء التعليمات البرمجية أو تشغيلها.

٣. في قائمة استكشاف EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

يجب فتح Terminal ويجب أن يتضمن موجه أوامر يظهر أن Terminal مفتوحة لموقع مجلد TestProject

٤. في موجه الأوامر Terminal لتشغيل التعليمات البرمجية، اكتب **dotnet run** ثم اضغط على Enter

إذا رأيت رسالة تقول "تعذر العثور على مشروع لتشغيله" فتأكد من أن موجه الأوامر Terminal يعرض موقع مجلد TestProject المتوقع. على سبيل المثال:

```
C:\Users\someuser\Desktop\csharpprojects\TestProject>
```

يجب أن تشاهد الإخراج التالي:

```
123455000  3
```

يعمل الأسلوب `Remove()` بشكل مشابه للأسلوب Substring() يمكنك توفير موضع البداية والطول، لإزالة هذه الأحرف من السلسلة.

استخدام الأسلوب Replace()

يتم استخدام الأسلوب `Replace()` عندما تحتاج إلى استبدال حرف واحد أو أكثر بحرف مختلف (أو بدون حرف) يختلف الأسلوب `Replace()` عن الأساليب الأخرى المستخدمة حتى الآن، فهو **يستبدل كل مثيل instance للأحرف المحددة، وليس فقط المثيل** الأول أو الأخير.

إزالة الأحرف بغض النظر عن مكان ظهورها في السلسلة

١. حدث التعليمات البرمجية في المحرر، كما يلي:

```
string message = "This--is--ex-amp-le--da-ta";
message = message.Replace("--", " ");
message = message.Replace("-", "");
Console.WriteLine(message);
```

٢. احفظ ملف التعليمات البرمجية، ثم شغلها، يجب أن تشاهد الإخراج التالي:

```
This is example data
```

هنا استخدمت الأسلوب Replace() مرتين، في المرة الأولى استبدلت فيها السلسلة – بمسافة space في المرة الثانية استبدلت السلسلة - بسلسلة فارغة empty string مما يؤدي إلى إزالة الحرف بالكامل من السلسلة.

الخلاصة

فيما يلي أمران مهمان يجب تذكرهما:

- يعمل الأسلوب Remove() مثل الأسلوب Substring() باستثناء أنه يحذف الأحرف المحددة في السلسلة.
- يقوم الأسلوب Replace() بتبديل كافة مثيلات instances سلسلة نصية، بسلسلة جديدة.

راجع معلوماتك

إذا تم تشغيل التعليمات `message = message.Replace("B", "D");` حيث `string message = "Big Dog";` ما هي القيمة الجديدة `message`

- `"Big Dog"`
- `"Big Bog"`
- `"Dig Dog"`

راجع إجابتك

"Dig Dog"

صحيح تم استبدال "B" بـ "D"

٥ تمرين ـ إكمال تحدي استخراج البيانات واستبدالها وإزالتها من سلسلة إدخال

تعزز تحديات التعليمات البرمجية التعلم، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

في هذا التحدي، يمكنك العمل مع سلسلة تحتوي على جزء من HTML يمكنك استخراج البيانات من جزء HTML واستبدال بعض محتواه، وإزالة أجزاء أخرى من محتواه لتحقيق الإخراج المطلوب.

إذا لم تكن معتاداً على تعليمة HTML البرمجية، فراجع قسم "Quick HTML primer" في نهاية هذا الدرس.

استخراج البيانات واستبدالها وإزالتها من سلسلة إدخال input string

١. أضف التعليمات البرمجية التالية كبداية، في المحرر، للحصول على بيانات التحدي:

```
const string input = "<div><h2>Widgets
&trade;</h2><span>5000</span></div>";

string quantity = "";
string output = "";

// Your work here

Console.WriteLine(quantity);
Console.WriteLine(output);
```

إذا قمت بتشغيل التعليمات البرمجية، فسيعرض الإخراج خطوطاً فارغة، فإن قيم البداية ل `quantity` and `output` هي قيم فارغة empty string

٢. خذ دقيقة لمراجعة السطر الأولي من التعليمات البرمجية التي تحتوي على نص HTML

```
const string input = "<div><h2>Widgets &trade;</h2><span>5000</span></div>";
```

لاحظ العلامات tags التالية `<div>`,`<h2>`,`<span>` ورمز symbol code التالي `&trade;` المضمن في المتغير input

٣. فحص الإخراج المطلوب لإخراج البرنامج النهائي:

```
Quantity: 5000
Output: <h2>Widgets &reg;</h2><span>5000</span>
```

٤. ابدأ بإضافة الحل، بداية التعليمات أسفل التعليق `// Your work here`

٥. قم بتهيئة المتغير `quantity` على القيمة التي تم الحصول عليها عن طريق استخراج النص بين علامتي `<span>` and `</span>`

٦. قم بتهيئة المتغير `output` على قيمة `input` ثم قم بإزالة العلامات `<div>` and `</div>`

٧. استبدل حرف HTML (`&trade;`) ™ب (`&reg;`) ® في المتغير `output`

٨. قم بتشغيل تعليماتك، وتحقق من أن وضع الإخراج يطابق الإخراج المتوقع.

```
Quantity: 5000
Output: <h2>Widgets &reg;</h2><span>5000</span>
```

سواء واجهتك مشكلة وتحتاج إلى إلقاء نظرة خاطفة على الحل أو انتهيت بنجاح، استمر لعرض حل لهذا التحدي.

تمهيد HTML سريع

في حالة عدم إلمامك ب HTML فهي لغة ترميز "توصيف النص" markup language تستخدم لإنشاء جميع صفحات الويب.

تخطي هذا القسم إذا كان لديك فهم جيد ل HTML تم تصميم هذه المعلومات لتوفير معلومات كافية لإكمال هذا التحدي، وليس لتكون برنامجًا تعليميًا شاملاً ل HTML

في HTML يمكنك تعريف بنية مستند باستخدام العلامات tags تتكون العلامات tags من الأجزاء التالية:

- قوس زاوية الفتح `<`
- قوس زاوية الإغلاق `>`
- كلمة تصف نوع العلامة، على سبيل المثال `<h2>` ,`<span>` ,`<div>` إلخ.

تحتوي كل علامة tag على علامة إغلاق closing tag مقابلة، تقدم حرف شرطة مائلة للأمام `/` لذلك، إذا رأيت `<div>` فيجب أن تكون هناك علامة `<div/>` مقابلة.

المحتوى بين علامة الفتح والإغلاق opening and closing tag هو محتوى هذه العلامة، قد يضم المحتوى نصاً text وعلامات أخرى.

يمكن تضمين مجموعة من العلامات داخل مجموعة أخرى من العلامات، ما يمنح مستند HTML بنيته الهرمية.

٦ راجع حل استخراج البيانات واستبدالها وإزالتها من سلسلة إدخال

تُعد التعليمات البرمجية التالية أحد الحلول الممكنة للتحدي من الدرس السابق:

```
const string input = "<div><h2>Widgets
&trade;</h2><span>5000</span></div>";

string quantity = "";
string output = "";

// Your work here

// Extract the "quantity"
const string openSpan = "<span>";
const string closeSpan = "</span>";

int quantityStart = input.IndexOf(openSpan) +
openSpan.Length; // + length of <span> so index
at end of <span> tag
int quantityEnd= input.IndexOf(closeSpan);
int quantityLength = quantityEnd - quantityStart;
quantity = input.Substring(quantityStart,
quantityLength);
quantity = $"Quantity: {quantity}";

// Set output to input, replacing the trademark
symbol with the registered trademark symbol
const string tradeSymbol = "&trade;";
const string regSymbol = "&reg;";
output = input.Replace(tradeSymbol, regSymbol);

// Remove the opening <div> tag
```

```
const string openDiv = "<div>";
int divStart = output.IndexOf(openDiv);
output = output.Remove(divStart, openDiv.Length);

// Remove the closing </div> tag and add
"Output:" to the beginning
const string closeDiv = "</div>";
int divCloseStart = output.IndexOf(closeDiv);
output = "Output: " +
output.Remove(divCloseStart, closeDiv.Length);

Console.WriteLine(quantity);
Console.WriteLine(output);
```

هذه التعليمة البرمجية هي مجرد حل واحد ممكن، طالما أن تعليماتك البرمجية تنتج الإخراج التالي، فقد نجحت.

```
Quantity: 5000
Output: <h2>Widgets &reg;</h2><span>5000</span>
```

إذا نجحت، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا كان لديك مشكلة في إكمال هذا التحدي، ربما يجب عليك مراجعة الدروس السابقة قبل المتابعة.

٧ اختبر معلوماتك

١- ما هو الأسلوب الذي يعثر على الفهرس التالي إما للحرف `-` أو الحرف `=` أو الحرف `_`

- `IndexOfAny()`
- `Remove()`
- `IndexOf()`

٢- ما هو المتغير الثابت constant variable

- سلسلة سحرية أو رقم
- متغير لابد يتطابق اسمه مع قيمته
- لا يمكن تغيير القيمة بمجرد تهيئتها

راجع إجابتك

١

IndexOfAny()

صحيح استخدم IndexOfAny() لاسترداد الفهرس التالي، لأي من الأحرف المتعددة في سلسلة

٢ لا يمكن تغيير القيمة بمجرد تهيئتها

صحيح لا يمكن تغيير قيمة متغير ثابت، بمجرد تهيئته

٨ الملخص

كان هدفك هو استخراج القيم وإزالتها واستبدالها في سلاسل نصية، غالباً ما تحتوي البيانات التي تتلقاها على بيانات أو أحرف غريبة، تحتاج إلى تجنبها أو إزالتها قبل أن تتمكن من استخدام البيانات الهدف.

باستخدام الأسلوب IndexOf() يمكنك من تحديد موضع حرف أو كلمة داخل سلسلة نصية أخرى، الموضع الذي تم إرجاعه من الأسلوب IndexOf() هو أول كتلة إنشاء لاستخدام الأسلوب Substring() لاستخراج جزء من سلسلة، بالنظر إلى موضع البداية وعدد الأحرف المراد استخراجها (الطول)، كما أنه مكنك من استخدام الأسلوب Remove() لإزالة الأحرف من سلسلة محددة بموضع البداية والطول.

لقد تعلمت أشكالًا مختلفة مثل الأسلوب LastIndexOf() للعثور على الحرف الأخير لنص داخل سلسلة نصية أخرى، و IndexOfAny() للعثور على موضع أي قيمة لمصفوفة حروف char array

استخدمت عبارة while للتكرار عبر سلسلة أطول للعثور على كافة مثيلات الحرف أو الكلمة، واستخراجها داخل سلسلة المصدر الأكبر larger source string وأخيرًا، استخدمت أسلوب Replace() لتبديل كافة مثيلات الحرف أو الكلمة داخل سلسلة نصية أكبر.

في حين أنه قد يكون من الممكن تنفيذ هذه الأنواع من العمليات باستخدام مصفوفة char والتكرار من خلال كل char للعثور على التطابقات، وتتبع نقاط البداية والنهاية التي تريد تحديد موقعها، وما إلى ذلك. قد يستغرق ذلك العديد من الخطوات لإنجاز ما يمكن للأساليب المساعدة string إنجازه في استدعاء واحد.

الوحدة السادسة

مشروع إرشادي ـ العمل مع البيانات المتغيرة

أظهر قدرتك على تطوير تطبيق وحدة تحكم، ينفذ تنسيق البيانات، واستخدام المصفوفات، لتقديم ميزة بحث.

الأهداف التعليمية

- تطبيق عبارات التكرار لجمع البيانات المُدخلة.
- استخدام معالجة البيانات.
- تنسيق إخراج البيانات.
- اختر أنواع البيانات المناسبة، وتحويل أنواع البيانات بأمان.
- معالجة مصفوفات الأرقام والنصوص والأحرف، بالإضافة إلى إضافة البيانات وإزالتها وفرزها.
- تعديل وإنشاء سلاسل معقدة من مصادر بيانات متعددة، وتنسيق البيانات لعرضها عبر الحقول.

محتويات الوحدة:

١- مقدمة

٢- الاستعداد للمشروع الإرشادي

٣- تمرين - مراجعة تعليمات البداية

٤- تمرين - إضافة بيانات التبرع المقترحة

٥- تمرين - إضافة بحث الكلب

٦- اختبر معلوماتك

٧- الملخص

١ المقدمة

لنفترض أنك مطور يحب دعم المجتمع المحلي. بدأت أنت وبعض أصدقائك مشروعًا تجاريًا يساعد في العثور على منازل جديدة للقطط والكلاب الضالة أو المهجورة، لقد بدأ عملك صغيرًا، مع وجود عدد قليل من الحيوانات فقط، لكنه بدأ في النمو، قام أصدقاؤك بتطوير تطبيق أولي يتيح دخول وعرض الحيوانات المتاحة للتبني، يطلبون منك التحقيق في إضافة ميزات إلى التطبيق يمكن أن تساعد في مطابقة الحيوانات التي تحت رعايتك، مع الأشخاص الذين يبحثون عن حيوان أليف جديد للعائلة.

عند إكمال مشروع الترميز هذا، يمكنك تطبيق معرفتك ومهاراتك في بيانات C#

- اختيار أنواع البيانات المناسبة، وتحويل أنواع البيانات بشكل آمن.
- التعامل مع مصفوفات الأرقام والسلاسل والأحرف، وإضافة البيانات وإزالتها وفرزها.
- تعديل وبناء سلاسل معقدة من مصادر بيانات متعددة، وتنسيق البيانات لعرضها عبر المناطق

لقد وجد فريقك أنه من المهم البحث في بيانات الحيوانات الأليفة، لتحديد الحيوانات ذات التطابقات المحتملة، استناداً إلى الخصائص التي يوفرها المالكون المحتملون، علاوة على ذلك، يريد الفريق إضافة ميزة جمع التبرعات، إضافة بيانات التبرع المقترحة، وتقديم بيانات الحيوانات الأليفة بخطوط إخراج أقل.

تبدأ بتطبيق البداية، الذي يضيف نموذج بيانات محدد مسبقًا إلى مصفوفة الحيوانات الأليفة، يحتوي التطبيق على عنصرين في القائمة. يعرض العنصر الأول جميع بيانات الحيوانات الأليفة، والتعليمات البرمجية مكتملة، والثاني هو عنصر القائمة، "عرض جميع الكلاب ذات الخصائص المحددة"، وهو "العمل قيد التقدم" التي تكملها.

يمكنك أيضًا إجراء تحديثات على التعليمات البرمجية الموجودة لإضافة بيانات التبرع المقترحة suggestedDonation وعرض جميع البيانات بتنسيق مختصر.

في نهاية هذه الوحدة، يمكنك تطوير التعليمات البرمجية التي تجمع بين عبارات التكرار، وإدخال البيانات ومعالجتها، وفهرسة السلسلة string وإخراج البيانات.

٢ الاستعداد للمشروع الإرشادي

استخدم Visual Studio Code لتطوير إصدار من تطبيق وحدة التحكم C# يأتي التطبيق مزودًا بالميزات الأساسية التي تنشئ نموذج بيانات عن الحيوانات الأليفة المتاحة للتبني، ويكون قادرًا على عرض معلومات الحيوان الأليف، الميزة الرئيسية التي يجب إضافتها هي البحث عن الكلاب المتاحة، باستخدام مصطلح بحث واحد، تتضمن المهام الثانوية إضافة وعرض بيانات التبرع المقترحة `suggestedDonation`

مواصفات التصميم

بالنسبة للميزات الجديدة لتطبيق Contoso Pets توفر مواصفات التصميم تفاصيل حول البحث عن الكلاب، وميزات التبرع المقترحة:

- **البحث في سمة الكلب**
  - جمع المدخلات لمصطلح البحث الخاص بالحيوانات الأليفة
  - حلقة عبر مصفوفة الحيوانات وحدد "الكلاب"
  - لكل كلب، اجمع بين أوصاف الشخصية والجسدية للبحث
  - البحث في الوصف المدمج لمطابقة مصطلح الإدخال
  - إخراج الكلاب التي لها مصطلح مطابق
- **بيانات التبرع المقترحة**
  - تعريف متغير السلسلة `suggestedDonation`
  - توسيع مصفوفة ourAnimals لتحتوي على التبرعات المقترحة `suggestedDonation` وتعبئة نموذج البيانات للتبرعات المقترحة `suggestedDonation`
  - تأكد من استخدام جميع حسابات المصفوفة ourAnimals للبيانات التبرع المقترحة المضافة suggestedDonation

- إخراج التبرع المقترح `suggestedDonation` برمز العملة الإقليمية (`$, €, ¥,...`)

نظرة عامة على التعليمات البرمجية الأولية Starter

اكتمل تطوير تعليمات البداية الأولية.

١. يتضمن المشروع الأولي، لوحدة المشروع الإرشادية هذه ملف Program.cs يوفر ميزات التعليمات البرمجية التالية:

- تعلن التعليمات البرمجية عن المتغيرات المستخدمة لجمع ومعالجة بيانات الحيوانات الأليفة واختيارات عناصر القائمة.
- تعلن التعليمات البرمجية عن مصفوفة `ourAnimals`
- تستخدم التعليمات البرمجية لحلقة حول بنية `if-else if-else` لملء مصفوفة ourAnimals بمجموعة بيانات نموذجية.

- تعرض التعليمات البرمجية خيارات القائمة الرئيسية التالية لتحديد المستخدم:

```
1. List all of our current pet information
2. Display all dogs with a specified characteristic

Enter menu item selection or type "Exit" to exit the program
```

- تقرأ التعليمات البرمجية اختيار عنصر القائمة الخاص بالمستخدم، وتعرض رسالة تؤكد اختياره.
- فقط الاختيار `"1. List all of our current pet information"` باستخدام التعليمات البرمجية الأولية.

هدفك هو تحديث التعليمات البرمجية الموجودة، لتطوير ميزات التطبيق الموضحة مسبقًا.

الميزات الرئيسية:

- إضافة بحث عن سمة الكلب
- تضمين بيانات التبرع المقترحة

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادية.

١. قم بتنزيل الملف المضغوط، الذي يحتوي على مجلدات التعليمات البرمجية للمشروع الإرشادي.

- في المستعرض، انتقل إلى Guided-project-Work-with-variable-data-in-CSharp.zip تنزيل الملف المضغوط.

٢. فك ضغط الملفات التي تم تنزيلها

- على جهازك المحلي، انتقل إلى مجلد التنزيلات.
- انقر بزر الماوس الأيمن فوق ملف Guided-project-Develop-conditional-branching-and-looping-structures-in-CSharp.zip ثم حدد استخراج الكل.
- دون موقع المجلد المستخرج.

٣. افتح Visual Studio Code في قائمة File حدد Open Folder

- انتقل إلى المجلد الذي يحتوي على الملفات المستخرجة، وقم بتوسيع بنية المجلد لتحديد موقع المجلد المسمى "GuidedProject"
- حدد Guided-project-Work-with-variable-data-in-CSharp-main ثم حدد Select Folder

يجب أن تظهر قائمة عرض EXPLORER مجلدين فرعيين باسم **Final** and **Starter**

أنت الآن جاهز لبدء تمارين المشروع الإرشادي. حظ سعيد!

٣ مراجعة مراجعة تعليمات البداية

في هذه الخطوة الأولى من عملية التطوير، يمكنك مراجعة التعليمات البرمجية المتوفرة في مجلد مشروع Starter

مراجعة محتويات ملف Program.cs

يحتوي ملف Program.cs على إصدار أولي من التطبيق الذي تعمل عليه، تتضمن التعليمات البرمجية ميزات لإنشاء وعرض نموذج بيانات للتطبيق، وتعرض خيارات القائمة التي تحدد الميزات الرئيسية للتطبيق.

١. تأكد من فتح **مجلد GuidedProject** في Visual Studio Code

يتضمن الدرس الاستعداد للمشروع الإرشادي (الدرس السابق في هذه الوحدة) قسم **إعداد** الذي يصف عملية تنزيل النموذج الأولي Starter للمشروع الإرشادي، إذا لزم الأمر، ارجع واتبع إرشادات الإعداد.

٢. في قائمة EXPLORER قم بتوسيع مجلد Starter ثم حدد Program.cs

عند تحديد ملف Program.cs يتم فتح محتويات الملف في منطقة المحرر الرئيسي على يمين EXPLORER

إذا لم تكن طريقة عرض EXPLORER مفتوحة، يمكنك تحديد/فتح طريقة عرض EXPLORER من شريط النشاط Activity Bar في أقصى الجانب الأيسر من EXPLORER هي الأيقونة العلوية في شريط النشاط.

٣. خذ بضع دقائق لمراجعة إعلانات المتغيرات الأولية الموجودة أعلى ملف Program.cs

```
// #1 the ourAnimals array will store the
following:
string animalSpecies = "";
string animalID = "";
string animalAge = "";
string animalPhysicalDescription = "";
string animalPersonalityDescription = "";
string animalNickName = "";
```

```
// #2 variables that support data entry
int maxPets = 8;
string? readResult;
string menuSelection = "";

// #3 array used to store runtime data, there
is no persisted data
string[,] ourAnimals = new string[maxPets, 6];
```

أولاً، ترى تعليقًا (التعليق رقم ١) متبوعًا بقائمة من المتغيرات، تُستخدم هذه المتغيرات، من أنواع الحيوانات `animalSpecies` إلى أسماء الحيوانات الأليفة `animalNickName` للاحتفاظ بقيم خصائص الحيوانات الأليفة ضمن مصفوفة سلسلة متعددة الأبعاد تسمى `ourAnimals` وتمت تهيئتها لتحتوي على سلسلة ذات طول صفري "" تعريف مصفوفة `ourAnimals` موجود في أسفل الكود قليلًا.

المجموعة التالية من المتغيرات (تحت التعليق رقم ٢) عبارة عن مزيج من متغيرات `string and int` المستخدمة للمساعدة في إنشاء نموذج أو عينة بيانات، وقراءة مدخلات المستخدم، ووضع معايير الخروج لحلقة البرنامج الرئيسية، لاحظ سطر التعليمات `string? readResult;` يمكنك استخدام الحرف **?** عادةً لتحويل نوع متغير non-nullable لا يقبل القيم الخالية (...,int, string, bool) إلى نوع nullable يدعم القيم الخالية.

ملاحظة

عند قراءة القيم التي أدخلها المستخدم باستخدام الأسلوب `Console.ReadLine()` فمن الأفضل تمكين nullable type string نوع يقبل القيم الخالية باستخدام `string?` لتجنب إنشاء المترجم تحذيراً عند بناء المشروع.

المتغير الأخير (تحت التعليق رقم ٣) هو مصفوفة سلسلة ثنائية الأبعاد تسمى `ourAnimals` لقد قمت بتهيئة عدد الصفوف، المحددة بواسطة maxPets إلى 8 وعدد الخصائص التي تقوم بتخزينها في البداية هو ستة، تتطابق

الخصائص الست مع عدد متغيرات السلسلة string التي فحصتها في نموذج التعليمات البرمجية، ولكن يجب توسيع عدد الخصائص لإضافة حقل للتبرع المقترح `suggestedDonation`

٤. مرر لأسفل ملف Program.cs لفحص الحلقة `for` التي تحتوي على عبارة `switch` داخل كتلتها البرمجية.

نموذج التعليمات البرمجية هو إصدار مختصر لتوفير مساحة

```
// #4 create sample data ourAnimals array
entries
for (int i = 0; i < maxPets; i++)
{
    switch (i)
    {
        case 0:
            animalSpecies = "dog";
            animalID = "d1";
            animalAge = "2";
            animalPhysicalDescription = "medium
sized cream colored female golden retriever
weighing about 45 pounds. housebroken.";
            animalPersonalityDescription =
"loves to have her belly rubbed and likes to
chase her tail. gives lots of kisses.";
            animalNickname = "lola";
            break;

        case 1:
            animalSpecies = "dog";
            animalID = "d2";
            animalAge = "9";
```

```
            animalPhysicalDescription = "large
reddish-brown male golden retriever weighing
about 85 pounds. housebroken.";
            animalPersonalityDescription =
"loves to have his ears rubbed when he greets
you at the door, or at any time! loves to lean-
in and give doggy hugs.";
            animalNickName = "gus";
            break;

        // case 2: deleted for brevity
        // case 3: deleted for brevity

        default:
            animalSpecies = "";
            animalID = "";
            animalAge = "";
            animalPhysicalDescription = "";
            animalPersonalityDescription = "";
            animalNickName = "";
            break;
    }
    ourAnimals[i, 0] = "ID #: " + animalID;
    ourAnimals[i, 1] = "Species: " +
animalSpecies;
    ourAnimals[i, 2] = "Age: " + animalAge;
    ourAnimals[i, 3] = "Nickname: " +
animalNickname;
    ourAnimals[i, 4] = "Physical description: "
+ animalPhysicalDescription;
    ourAnimals[i, 5] = "Personality: " +
animalPersonalityDescription;
}
```

٥. لاحظ أن الحلقة for تستخدم المتغير maxPets لإنشاء حد أعلى على عدد التكرارات التالية، تحت التعليق رقم ٤

٦. لاحظ أيضاً بنية switch construct تفرع التعليمات البرمجية بشكل انتقائي، بحيث يمكنك تحديد خصائص الحيوانات الأليفة المختلفة، للحيوانات الأليفة في نموذج أو عينة البيانات.

يمكنك استخدام عبارة switch لتعريف قيم مختلفة، للتكرارات الأربعة الأولى من حلقة التكرار for بعد معالجة بيانات العينة، تكون كافة الخصائص عبارة عن سلسلة فارغة أو ذات طول صفري empty, or zero-length, string

يتم تعيين قيم متغيرات خصائص الحيوان إلى مصفوفة ourAnimals الموجودة أسفل الحلقة for

٧. قم بالتمرير إلى أسفل ملف التعليمات البرمجية في Visual Studio Code افحص التعليمات المستخدمة لعرض خيارات القائمة، والتقاط مدخلات اختيار المستخدم.

يجب عليك مراقبة التعليمات البرمجية التالية:

```
// #5 display the top-level menu options
do
{
    // NOTE: the Console.Clear method is
throwing an exception in debug sessions
    Console.Clear();

    Console.WriteLine("Welcome to the Contoso
PetFriends app. Your main menu options are:");
    Console.WriteLine(" 1. List all of our
current pet information");
    Console.WriteLine(" 2. Display all dogs
with a specified characteristic");
    Console.WriteLine();
```

```
    Console.WriteLine("Enter your selection
number (or type Exit to exit the program)");

    readResult = Console.ReadLine();
    if (readResult != null)
    {
        menuSelection = readResult.ToLower();
    }

    // use switch-case to process the selected
menu option
    switch (menuSelection)
    {
        case "1":
            // list all pet info
            for (int i = 0; i < maxPets; i++)
            {
              if (ourAnimals[i, 0] != "ID #: ")
                {
                    Console.WriteLine();
                    for (int j = 0; j < 6; j++)
                    {

Console.WriteLine(ourAnimals[i, j]);
                    }
                }
            }
            Console.WriteLine("\n\rPress the
Enter key to continue");
            readResult = Console.ReadLine();
            break;
```

```
        case "2":
            // Display all dogs with a
specified characteristic");
            Console.WriteLine("\nUNDER
CONSTRUCTION - please check back next month to
see progress.");
            Console.WriteLine("Press the Enter
key to continue.");
            readResult = Console.ReadLine();
            break;

        default:
            break;
    }

} while (menuSelection != "exit");
```

٨ خذ دقيقة لمراجعة عبارتي two case statements

يوجد خيارين فقط للقائمة يعملان مع إصدار محدود من التطبيق، يوفر التطبيق الأولي Starter فقط الإمكانات اللازمة لتشغيل واختبار ميزات النموذج الأولي.

٩. لاحظ أن سطر التعليمات البرمجية `readResult = Console.ReadLine();` متبوعًا بالتحقق من القيمة الخالية null value

تحتوي التعليمات البرمجية التي تستخدم الأسلوب `Console.ReadLine()` على القيمة المعينة إلى nullable string المسماة `readResult` التي تقبل القيم الخالية، لتجنب إنشاء المحول البرمجي "المترجم" تحذيرًا عند إنشاء المشروع.

تحقق من عملك

١. اختبر التعليمات البرمجية لتطبيق وحدة تحكم الأولي Starter في موجه الأوامر TERMINAL قم بإنشاء وتشغيل التعليمات البرمجية للمشروع، باستخدام أمر واحد عن طريق إدخال `dotnet run`

ملاحظة

يجب فتح موجه TERMINAL في مجلد start ويجب أن يبدو TERMINAL مشابهًا `..\ArrayGuidedProject\starter>`

عند تشغيل التعليمات البرمجية، يتم عرض عنصري قائمة.

- أدخل "1" لاختبار إخراج، سرد جميع معلومات الحيوانات الأليفة الحالية "List all of our current pet information"
- أدخل "2" لاختبار رسالة العنصر النائب "قيد الإنشاء" "under construction"

٢. أدخل 1 ثم أضغط Enter لعرض جميع الحيوانات الأليفة "display all pets"

٣. تحقق من عرض جميع معلومات الحيوانات الأليفة.

أثناء ملاحظة البيانات الموجودة على جميع الحيوانات الأليفة، يجب أن يتطابق آخر حيوان أليف معروض مع الإخراج التالي:

```
ID #: c4
Species: cat
Age: 3
Nickname: Lion
Physical description: Medium sized, long hair,
yellow, female, about 10 pounds. Uses litter
box.
Personality: A people loving cat that likes to
sit on your lap.

Press the Enter key to continue
```

٤. اضغط على المفتاح Enter للمتابعة والعودة إلى القائمة الرئيسية.

٥. في موجه القائمة، أدخل 2 ثم أضغط على Enter

يعد هذا الاختيار عنصرًا نائبًا لوظيفة، عرض جميع الكلاب ذات الخصائص المحددة "Display all dogs with a specified characteristic"

٦. تحقق من عرض رسالة "UNDER CONSTRUCTION" لتحديد عرض جميع الكلاب ذات الخصائص المحددة.

يجب عليك مراقبة الإخراج التالي:

```
UNDER CONSTRUCTION - please check back next
month to see progress.
Press the Enter key to continue.
```

٧. اكتب exit في قائمة التطبيق، لإنهاء البرنامج ثم أغلق لوحة المحطة الطرفية the terminal panel

يجب إنهاء البرنامج.

أنت الآن جاهز لبدء تطوير الميزات الجديدة.

٤ تمرين ـ إضافة بيانات التبرع المقترحة

في هذه الخطوة من عملية التطوير، يمكنك تحديث التعليمات البرمجية المتوفرة في مجلد مشروع البداية، وإضافة ميزات "التبرع المقترح" إلى التطبيق، يجب أن يكون ملف The starter project.cs مفتوحاً في Visual Studio Code ارجع إلى درس الإعداد السابق، إذا كنت بحاجة إلى إرشادات للبدء.

إنشاء ميزات التبرع المقترح

تتطلب إضافة ميزات التبرع إنشاء متغير suggestedDonation وتوسيع المصفوفة ourAnimals لاستيعاب بيانات التبرع الجديدة، تحتاج أيضاً إلى إضافة مبالغ التبرع لكل حيوان، وتنفيذ مبلغ افتراضي عند فقدان معلومات التبرع المقترحة.

إضافة دعم لبيانات التبرع المقترحة

تحتاج إلى إنشاء متغير جديد للاحتفاظ بالتبرعات المقترحة، وتوسيع المصفوفة ourAnimals للاحتفاظ بالبيانات الجديدة.

١. لاحظ التعليمات البرمجية ضمن التعليق رقم ١ التي تعلن عن المتغيرات المستخدمة لملء المصفوفة ourAnimals لكل حيوان.

تحتاج إلى الإعلان عن متغير string آخر لبيانات التبرع المقترحة

```
// #1 the ourAnimals array will store the
following:
string animalSpecies = "";
string animalID = "";
string animalAge = "";
string animalPhysicalDescription = "";
string animalPersonalityDescription = "";
string animalNickName = "";
```

٢. قم بإنشاء المتغير `suggestedDonation` أسفل الإعلان عن `animalNickName`

تتم إضافة الإعلان `suggestedDonation` مع التعليمات البرمجية التالية:

```
string suggestedDonation = "";
```

٣. حدد موقع التعليمات البرمجية لإنشاء المصفوفة `ourAnimals` بعد التعليق رقم 3

ينشئ السطر التالي من التعليمات البرمجية المصفوفة

```
string[,] ourAnimals = new string[maxPets, 6];
```

الأحجام التي تحدد بعدي المصفوفة هي الحد الأقصى لعدد الحيوانات الأليفة `maxPets` والرقم 6 للسلاسل الستة المعرفة في الأصل، ولكن بدون مساحة لبيانات التبرع المقترحة الجديدة `suggestedDonation`

٤. تحديث المصفوفة `ourAnimals` لتحتوي على 7 "أعمدة" من البيانات لكل حيوان بدلاً من 6

يعرض السطر التالي التعليمات البرمجية المحدثة:

```
string[,] ourAnimals = new string[maxPets, 7];
```

لقد قمت بتوسيع مصفوفة ourAnimals لدعم إضافة بيانات التبرع المقترحة.

أضف مبالغ التبرع المقترحة suggestedDonation إلى عينة البيانات

١. خذ دقيقة للمراجعة `case 0:` داخل عبارة `switch` بعد التعليق رقم 4

التعليمة البرمجية التالية التي تحدد عينة البيانات للحيوان الأليف الأول قبل إنشاء بيانات التبرع المقترح `suggestedDonation` سيكون مناسباً أسفل `animalNickName`

```
case 0:
        animalSpecies = "dog";
        animalID = "d1";
        animalAge = "2";
        animalPhysicalDescription = "medium
sized cream colored female golden retriever
weighing about 45 pounds. housebroken.";
        animalPersonalityDescription = "loves
to have her belly rubbed and likes to chase her
tail. gives lots of kisses.";
        animalNickName = "lola";
        break;
```

٢. قم بإدراج قيمة التبرع المقترح `suggestedDonation` أعلى عبارة `break` للحالة `case 0;` من خلال `default:` بالقيم التالية:

- `Case 0: suggestedDonation = "85.00";`
- `Case 1: suggestedDonation = "49.99";`
- `Case 2: suggestedDonation = "40.00";`
- `Case 3: suggestedDonation = "";`
- `default: suggestedDonation = "";`

تعرض التعليمات التالية الحالة `case 0:` مع إضافة "suggestedDonation"

```
case 0:
        animalSpecies = "dog";
        animalID = "d1";
        animalAge = "2";
        animalPhysicalDescription = "medium
sized cream colored female golden retriever
weighing about 45 pounds. housebroken.";
```

```
        animalPersonalityDescription = "loves
to have her belly rubbed and likes to chase her
tail. gives lots of kisses.";
        animalNickName = "lola";
        suggestedDonation = "85.00";
        break;
```

٣. في ملف project.cs حدد موقع المصفوفة التالية المملوءة ببيانات الحيوانات الأليفة من عبارات case (قبل التعليق رقم 5)

```
ourAnimals[i, 0] = "ID #: " + animalID;
    ourAnimals[i, 1] = "Species: " +
animalSpecies;
    ourAnimals[i, 2] = "Age: " + animalAge;
    ourAnimals[i, 3] = "Nickname: " +
animalNickName;
    ourAnimals[i, 4] = "Physical description: "
+ animalPhysicalDescription;
    ourAnimals[i,  5]  =  "Personality:  "  +
animalPersonalityDescription;
```

لاحظ أن بيانات التبرعات المقترحة `suggestedDonation` غير مضافة كجزء من نموذج التعليمات البرمجية ل starter لملء بيانات المصفوفة.

من المنطقي ملء المصفوفة ببيان مثل:

```
ourAnimals[i, 6] = "Suggested Donation: "  +
suggestedDonation;
```

ولكن، لن تضيف هذه التعليمة البرمجية، في القسم التالي، ستستخدم نهجاً آخر.

التحقق من صحة البيانات باستخدام TryParse()

الغرض من المتغير suggestedDonation أن يكون قيمة رقمية، ولكن يتم جمعه وتخزينه ك string في المستقبل قد تحتاج إلى التحقق من أن suggestedDonation يخزن رقم عشري `decimal` ويمكنك تحويله إلى `decimal` بحيث يكون متاحاً للاستخدام لحسابات الفوترة، لتجنب حدوث خطأ أثناء محاولة استخدام إدخال مثلاً عشرين، تحتاج إلى استخدام TryParse التحقق من الصحة

ملاحظة

تم تصميم نماذج التعليمات البرمجية الموجودة في هذا التمرين استنادًا إلى إعدادات ثقافة الولايات المتحدة، واستخدم النقطة (.) كفاصل عشري، قد يؤدي إنشاء التعليمات البرمجية وتشغيلها باستخدام إعداد ثقافة يستخدم فواصل عشرية مختلفة (مثل الفاصلة, comma) إلى نتائج أو أخطاء غير متوقعة، لحل هذه المشكلة، استبدل الفواصل العشرية في نماذج التعليمات البرمجية بالفاصل العشري المحلي (مثل،) وبدلاً من ذلك، لتشغيل برنامج باستخدام إعداد ثقافة en-US قم بإضافة التعليمة البرمجية التالية إلى الجزء العلوي من البرنامج `using System.Globalization;` و بعد أي عبارات `using` الأخرى، أضف `CultureInfo.CurrentCulture = new CultureInfo("en-US");`

١. قبل التعليق رقم 5 داخل نهاية كتلة التعليمات البرمجية، أضف تعليمات للتحقق من إمكانية تحويل suggestedDonation إلى رقم عشري.

يمكنك إضافة ما يلي:

```
if (!decimal.TryParse(suggestedDonation, out
decimalDonation)){
    decimalDonation = 45.00m; // if
suggestedDonation NOT a number, default to
45.00
}
```

إذا لم يكن من الممكن تحويل cast متغير `suggestedDonation` كرقم عشري decimal فستقوم التعليمات بتعيين قيمة افتراضية `decimalDonation = 45.00m;` إذا نجح الإرسال cast فسيقوم TryParse بملء `decimalDonation` وفي كلتا الحالتين، يمثل `decimalDonation` عددًا عشريًا مناسبًا.

٢. تأكد من أن تعليمة التحقق، في المكان الصحيح!

يجب أن يظهر السطران اللذان أضفتهما في التعليمات، أعلى السطرين العلويين من التعليمات التالية:

```
if (!decimal.TryParse(suggestedDonation, out
decimalDonation)){
        decimalDonation = 45.00m; // if
suggestedDonation NOT a number, default to
45.00
    }
}

// #5 display the top-level menu options
```

لاحظ، لا يزال التحقق من الصحة لا يعمل، لا تنسى ضرورة الإعلان عن `decimalDonation` لاستخدامه في التعليمات البرمجية.

٣. بعد التعليق رقم 2 قم بإعلان `decimalDonation` بعد `string menuSelection = "";`

```
decimal decimalDonation = 0.00m;
```

أخيرًا، أنت جاهز لملء بيانات suggestedDonation لكل حيوان أليف.

٤. أعلى التعليق رقم 5 مباشرةً، بعد قوس الإغلاق } ل `TryParse` أضف التعليمات التالية:

```
ourAnimals[i, 6] = $"Suggested Donation:
{decimalDonation:C2}";
```

لقد استخدمت نسخة `decimalDonation` من بيانات التبرع المقترحة، واستخدمت أيضًا استيفاء السلسلة، وتنسيق العملة في الإخراج.

٥. خذ دقيقة لمراجعة كيفية تحويل بيانات التبرع المقترحة أخيرا إلى مصفوفة `ourAnimals`

التعليمة البرمجية التالية تملأ `decimalDonation` في سياق TryParse()

```
if (!decimal.TryParse(suggestedDonation, out
decimalDonation)){
    decimalDonation = 45.00m; // if
suggestedDonation NOT a number, default to
45.00
    }
ourAnimals[i, 6] = $"Suggested Donation:
{decimalDonation:C2}";
}

// #5 display the top-level menu options
```

لاحظ أنه باستخدام التعليمة `{decimalDonation:C2}` سيتم عرض التبرع المقترح من `decimalDonation` مع رمز العملة المحلية، ومنزلتين عشريتين وفقًا لتوجيهات تنسيق العملة `:C2`

مراجعة وتحديث متى ينبغي استخدام المصفوفة ourAnimals

أدت إضافة بيانات التبرع المقترحة `suggestedDonation` إلى الحاجة إلى مزيد من التحديثات، يفتقد خيار القائمة 1 `"1. List all of our current pet information"` إلى البيانات المضافة.

١. لاحظ التعليمات أسفل التعليق رقم 5 لحلقة القائمة داخل `case 1`

تحتاج الحلقة الداخلية `for (int j = 0; j < 6; j++)` التي تطبع سمات الحيوانات الأليفة إلى مراعاة بيانات التبرع المقترحة `suggestedDonation` المضافة.

٢. تحديث شرط إنهاء التعليمات، والخروج من الحلقة الداخلية، زيادة شرط الخروج بمقدار 1 بحيث يصبح `j < 7` تحقق من أن التعليمات البرمجية تطابق ما يلي:

```
case "1":
// list all pet info
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 0] != "ID #: ")
    {
        Console.WriteLine();
        for (int j = 0; j < 7; j++) //
increased exit condition
        {
           Console.WriteLine(ourAnimals[i, j]);
        }
    }
}
```

نظرة عامة على الاختبار

مع وجود العديد من إضافات التعليمات البرمجية، تحتاج إلى التأكد من أن تعليماتك البرمجية تعمل كما هو متوقع، مجالي الاختبار المهمين هما:

١. تتوافق التعليمات البرمجية دون أخطاء.

٢. يؤدي تحديد خيار القائمة ١ إلى عرض جميع معلومات الحيوانات الأليفة:

- يتضمن الإخراج جميع معلومات الحيوانات الأليفة، بما في ذلك: المعرف، والأنواع، والعمر، والكنية، والوصف المادي، ووصف الشخصية.
- لكل حيوان أليف، يتم عرض التبرع المقترح، برمز العملة، وبرقمين عشريين من الدقة.

راجع عملك

قم بالبناء والتشغيل لاختبار التعليمات البرمجية، استخدم هذه الخطوات في كل مرة تحتاج فيها إلى اختبار تعليماتك البرمجية.

١. في قائمة ملف file حدد حفظ save

٢. في قائمة استكشاف EXPLORER انقر بزر الماوس الأيمن فوق **Starter** ثم حدد **Open in Integrated Terminal**

يجب فتح لوحة TERMINAL أسفل منطقة محرر التعليمات البرمجية.

هناك عدة طرق لفتح المحطة الطرفية المتكاملة ل Visual Studio Code على سبيل المثال، توفر القائمة العلوية الوصول إلى لوحة TERMINAL من كل **من القائمة View** وقائمة **Terminal** يمكنك أيضاً معرفة اختصارات لوحة المفاتيح التي تفتح لوحة TERMINAL كل أسلوب مقبول.

٣. لاحظ أن لوحة TERMINAL تتضمن موجه سطر أوامر command line prompt ويعرض مسار المجلد الحالي، على سبيل المثال.

```
C:\Users\someuser\Desktop\GuidedProject\Starter>
```

يمكنك استخدام لوحة TERMINAL لتشغيل أوامر واجهة سطر الأوامر (CLI) مثل إنشاء `dotnet build` and `dotnet run` يقوم أمر dotnet build بتجميع التعليمات البرمجية، وعرض رسائل الخطأ والتحذير، المتعلقة ببناء جملة التعليمات البرمجية code syntax

هام

تحتاج إلى التأكد من أن موجه الأوامر الطرفي terminal command prompt مفتوح لجذر مساحة عمل مشروعك، في هذه الحالة، يكون جذر مساحة عمل المشروع هو المجلد Starter حيث توجد ملفات `Starter.csproj` and `Program.cs` عند تشغيل الأوامر في الوحدة الطرفية، ستحاول الأوامر تنفيذ الإجراءات باستخدام موقع المجلد الحالي، إذا حاولت تشغيل أوامر dotnet build أو dotnet run من موقع مجلد لا يحتوي على الملفات، فستقوم الأوامر بإنشاء رسائل خطأ.

٤. في موجه الأوامر TERMINAL لإنشاء التعليمات البرمجية للمشروع، أدخل الأمر التالي `dotnet build`

بعد ثوان، يجب أن تشاهد رسالة تخبرك بنجاح البناء، ولديك `0 Warning(s) and 0 Error(s)`

```
Determining projects to restore...
All projects are up-to-date for restore.
Starter ->
C:\Users\someuser\Desktop\GuidedProject\Starter
\bin\Debug\net6.0\Starter.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)
```

استخدم خطوات dotnet build and dotnet run المذكورة أعلاه في كل مرة تحتاج فيها إلى اختبار تعليماتك البرمجية، في التمارين التالية في هذه الوحدة.

٥. إذا رأيت رسائل خطأ أو تحذير، فستحتاج إلى إصلاحها قبل المتابعة.

تسرد رسائل الخطأ والتحذير سطر التعليمات البرمجية الذي تم العثور على المشكلة فيه، الرسالة التالية هي مثال لرسالة خطأ Build FAILED

```
C:\Users\someuser\Desktop\GuidedProject\Starter
\Program.cs(53,18): error CS1002: ; expected
[C:\Users\someuser\Desktop\GuidedProject\Starte
r\Starter.csproj]
```

تخبرك هذه الرسالة بنوع الخطأ الذي تم اكتشافه، ومكان العثور عليه، في هذه الحالة، تخبرك الرسالة أن ملف Program.cs يحتوي على خطأ `error CS1002: ; expected`. The ; expected يقترح أنك نسيت تضمين ; في نهاية العبارة، يخبرك جزء `Program.cs(53,18` من الرسالة بموقع الخطأ، على سطر التعليمات البرمجية 53 في موضع يبعد 18 حرفًا من اليسار.

خطأ في بناء الجملة مثل هذا، يمنع نجاح البناء (فشل البناء Build FAILED) توفر بعض رسائل البناء "تحذير Warning" بدلاً من "خطأ Error" مما يعني أن هناك شيئًا يجب الاهتمام به، ولكن يمكنك تجربة تشغيل البرنامج على أي حال (نجح البناء Build succeeded).

بمجرد إصلاح المشكلات وحفظ تحديثاتك، يمكنك تشغيل أمر dotnet build مرة أخرى، استمر حتى تحصل على 0 تحذير و0 خطأ 0 Warning(s) and 0 Error(s)

إذا واجهت صعوبات في حل مشكلة بنفسك، يمكنك فحص التعليمات البرمجية Program.cs في المجلد النهائي Final المضمن كجزء من التنزيل الذي أكملته أثناء الإعداد، تمثل التعليمات البرمجية Program.cs في المجلد النهائي Final ختام جميع التدريبات في هذه الوحدة، لذلك تتضمن التعليمات البرمجية التي لم تقم بإنشائها بعد.

قد تبدو التعليمات البرمجية للحل مختلفة إلى حد كبير عن التعليمات البرمجية Program.cs التي قمت بتطويرها في هذه المرحلة في المشروع الإرشادي، ومع ذلك، يمكنك محاولة فحص التعليمات البرمجية Program.cs في Final لمساعدتك في عزل مشكلة في التعليمات البرمجية، وإصلاحها، إذا كنت بحاجة إلى ذلك.

حاول ألا تتوقف عن محاولة حل مشاكل التعليمات البرمجية، تذكر أنك تتعلم من الأخطاء، وأن كل مطور يقضي وقتًا في البحث عن الأخطاء وإصلاحها.

٦. اختبر تطبيق وحدة التحكم المحدث، في موجه الأوامر TERMINAL، قم بإنشاء وتشغيل التعليمات البرمجية لمشروعك، باستخدام أمر واحد عن طريق إدخال: `dotnet run` عند تشغيل التعليمات، يتم عرض عنصرين في القائمة.

- أدخل "1" لاختبار إخراج عرض جميع الحيوانات الأليفة "display all pets"
- أدخل "2" لاختبار رسالة العنصر النائب قيد الإنشاء "under construction"

يجب أن يتطابق إخراج عنصر القائمة #1 بشكل وثيق مع النموذج التالي:

ID #: d1
Species: dog
Age: 2
Nickname: lola
Physical description: medium sized cream colored female golden retriever weighing about 45 pounds. housebroken.
Personality: loves to have her belly rubbed and likes to chase her tail. gives lots of kisses.
Suggested Donation: $85.00

ID #: d2
Species: dog
Age: 9
Nickname: gus
Physical description: large reddish-brown male golden retriever weighing about 85 pounds. housebroken.
Personality: loves to have his ears rubbed when he greets you at the door, or at any time! loves to lean-in and give doggy hugs.
Suggested Donation: $49.99

ID #: c3
Species: cat
Age: 1
Nickname: snow
Physical description: small white female weighing about 8 pounds. litter box trained.
Personality: friendly
Suggested Donation: $40.00

```
ID #: c4
Species: cat
Age:
Nickname: lion
Physical description:
Personality:
Suggested Donation: $45.00
```

إذا كان كل شيء يعمل كما هو متوقع، تهانينا! وإلا، أبحث عن الخطأ عن طريق التحقق من خطوات تعليمات التعليمات البرمجية السابقة، إذا لزم الأمر، ابدأ من جديد بملف Project.cs جديد، وإذا كانت لا تزال لديك مشكلات، فتحقق من رمز مجلد الحل لهذا التمرين.

٧. اكتب exit في قائمة التطبيق، لإنهاء البرنامج ثم أغلق اللوحة الطرفية terminal panel

٥ تمرين - إضافة بحث الكلب

في هذا التمرين، يمكنك إضافة الميزة عرض جميع الكلاب بخاصية محددة "Display all dogs with a specified characteristic" عنصر القائمة الثاني (menu item #2) إلى التطبيق، يستخدم التمرين مشروع الحل من التمرين السابق الذي أضاف بيانات التبرعات المقترحة `suggestedDonation`

ملاحظة

هذه ميزة "الحد الأدنى من المنتجات القابلة للتطبيق" (MVP) تهدف ميزة MVP إلى أن تكون نموذجًا أوليًا بسيطًا للعمل، لميزة تتيح التسليم السريع والسهل، لا يعد MVP عادةً منتجًا نهائيًا، بل يهدف إلى مساعدتك في العمل على فكرة، واختبارها، وجمع المزيد من المتطلبات.

تطالب ميزة البحث، المستخدم بإدخال مصطلح (أو عبارة) بحث واحد، يصف السمة المرغوبة في حيوان أليف ليتبناه، بعد ذلك، يتم البحث في أوصاف الكلاب القابلة للتبني، عن التطابقات الدقيقة لإدخالات المستخدم، يتم عرض المعلومات حول الكلاب المطابقة إلى وحدة التحكم، إذا لم يتم تحديد أي تطابقات، فسيتم عرض رسالة لا أحد من كلابنا متطابق " None of our dogs are a match" مع مصطلح البحث المستخدم.

المهام التي ستكملها أثناء هذا التمرين هي:

١. جمع إدخال المستخدم لمصطلح البحث، الخاص بالحيوانات الأليفة.

٢. حلقة من خلال مصفوفة الحيوانات وتحديد "الكلاب".

٣. لكل كلب، ابحث في وصف الحيوانات الأليفة، عن مصطلح مطابق.

٤. عرض الكلاب التي تحتوي على تطابق، في المصطلح.

في Visual Studio Code افتح ملف Project.cs المكتمل من التمرين السابق، الذي أضاف معلومات التبرع، للبدء.

جمع إدخال المستخدم للبحث عن خصائص الحيوانات الأليفة

١. راجع بيان تبديل القائمة التالي للتعليق رقم 5 داخل تعليمات Project.cs اكتشف التعليمات البرمجية التي تعرض رسالة تحت الإنشاء " UNDER CONSTRUCTION"

٢. أحذف التعليمات `Console.WriteLine("\nUNDER CONSTRUCTION - please check back next month to see progress.");` بين `case "2":` وعبارة `Console.WriteLine("Press the Enter key to continue.");` بحيث تتطابق التعليمات البرمجية مع النموذج التالي:

```
case "2":
    // Display all dogs with a specified
characteristic
    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
    break;
```

٣. أضف التعليمات البرمجية لجمع مدخلات المستخدم لسلسلة `dogCharacteristic` يتطلب جمع الإدخال حلقة تكرار `while` تستمر في مطالبة المستخدم حتى يرسل إدخالاً، ترشد الحلقة المستخدم إلى إدخال خاصية كلب واحدة مطلوبة للبحث عنها " Enter one desired dog characteristic to search for" يؤدي إدخال سلسلة فارغة empty string إلى تكرار الحلقة، ضع التعليمات التالية بعد `case "2:` قبل `break;` مباشرة، كما هو موضح:

```
case "2":
    // Display all dogs with a specified
characteristic
    string dogCharacteristic = "";

    while (dogCharacteristic == "")
```

```
    {
        // have the user enter physical
characteristics to search for
        Console.WriteLine($"\nEnter one desired
dog characteristics to search for");
        readResult = Console.ReadLine();
        if (readResult != null)
        {
            dogCharacteristic =
readResult.ToLower().Trim();
        }
    }
    Console.WriteLine("Press the Enter key to
continue.");
    readResult = Console.ReadLine();
    break;
```

`case "2":` راجع التعليمات البرمجية المضافة، تضم الميزات الرئيسية تعليمات تشمل:

- تبدأ التعليمات بإعلان `string dogCharacteristic = "";` يتم تحديد نطاقها إلى `case "2":` لن تتمكن من استخدام `dogCharacteristic` في أي مكان خارج تعليمات عبارة case statement
- بعد أن يجمع `Console.ReadLine()` مدخلات المستخدم، كميزة `dogCharacteristic` فإنه يضمن أن القيمة ليست فارغة value isn't null ويضبط السلسلة string على أحرف صغيرة lowercase ويقطع trims المسافات المحيطة.
- إذا كانت قيمة `dogCharacteristic` فارغة، فستكرر حلقة جمع المدخلات.

٤. احفظ التعليمات البرمجية، ثم قم بالتحويل البرمجي باستخدام dotnet build في TERMINAL (يمكنك الرجوع إلى الخطوات في التمرين السابق) وإصلاح أي أخطاء.

٥. قم بتشغيل التعليمات البرمجية باستخدام dotnet run

٦. اختبر وظيفة إدخال مصطلح البحث في التطبيق، عن طريق إدخال 2 لتحديد خيار القائمة "2" للبحث.

٧. اضغط على "Enter" دون إدخال البيانات في الموجه، أدخل أحد خصائص الكلب المطلوبة للبحث عنها " Enter one desired dog characteristic to search for" يجب أن يعود البرنامج إلى المطالبة، أدخل أحد خصائص الكلب المطلوبة للبحث عنها "Enter one desired dog characteristics to search for"

٨. يجب أن يعود البرنامج إلى المطالبة حتى يتم إدخال الأحرف.

٩. في الموجه لاختبار مصطلح البحث، إدخال "golden" يجب أن يعود البرنامج إلى القائمة دون خطأ.

١٠. في القائمة، اكتب Exit للخروج من البرنامج.

تحديد أي الحيوانات هي كلاب

الآن يمكنك إضافة ميزة للبحث باستخدام إدخال المستخدم `dogCharacteristic` ضمن أوصاف الكلب بعد التعليمات البرمجية السابقة ضمن `case "2"` لكن أولاً عليك التعرف على الكلاب.

١. في نهاية التعليمات البرمجية `case "2"` قبل التعليمات `Console.WriteLine("\n\rPress the Enter key to continue");` وتكون قبل `break;` أضف التعليمات التالية:

```
// #6 loop through the ourAnimals array to
search for matching animals
for (int i = 0; i < maxPets; i++)
{
    if (ourAnimals[i, 1].Contains("dog"))
    {
        // #7 Search combined descriptions and
report results
    }
}
```

٢. خذ دقيقة لفحص الحلقة المضافة إلى نهاية التعليمات البرمجية `case "2":`

تقوم التعليمات البرمجية بتصفية "الكلاب" باستخدام `ourAnimals[i,1]` حيث يتم تخزين بيانات أنواع الحيوانات `animalSpecies` إذا كان `animalSpecies` يخزن "dog" ثم تنتقل التعليمات البرمجية إلى أقواس عبارة `if` حيث يمكن إجراء البحث عن الأوصاف المجمعة.

٣. قم بالتجميع أو التحويل البرمجي للتعليمات البرمجية، باستخدام dotnet build وقم بإصلاح أي أخطاء.

البحث المركب في معلومات وصف الكلاب

تضم التعليمات البرمجية السابقة البحث في أوصاف الكلاب فقط، الآن تحتاج إلى البحث في أوصاف الكلب، وإخراج معلومات حول التطابقات.

أثناء التفكير في الأوصاف، تدرك أن هناك وصفين `animalPhysicalDescription and animalPersonality Description;` بعد التشاور، يقرر الفريق أن الوصف المركب مناسب للبحث.

ملاحظة

يشير بعض المطورين إلى إضافة المتطلبات أثناء التطوير باسم "زحف النطاق" على الرغم من أن الجمع بين الأوصاف لا يتطلب الكثير من العمل، إلا أنه يضيف الوقت والتعقيد، لهذا السبب، يجب عليك إعلام الفريق بأن المتطلبات الإضافية من المحتمل أن تؤخر إكمال المشروع.

الجمع بين أوصاف الكلاب لتسهيل عملية البحث

تحتاج إلى الإعلان عن سلسلة `dogDescription` للاحتفاظ بالبيانات المجمعة التي نشأت من `animalPhysicalDescription and animalPersonalityDescription;`

١. قم بتعريف dogDescription مباشرة قبل التعليق رقم 6 باستخدام التعليمات التالية:

```
string dogDescription = "";
```

يمكنك الآن استخدام المتغير dogDescription الذي أعلنته، تحتاج إلى تعبئته بوصفين لكل حيوان.

٢. استخدام المتغير dogDescription بملئه ب animalPhysicalDescription and animalPersonality Description;

بعد التعليق رقم 7 أضف التعليمات التالية:

```
dogDescription = ourAnimals[i, 4] + "\n" + ourAnimals[i, 5];
```

البحث في أوصاف الكلاب المجمعة/المركبة وعرضها

الآن يمكنك إضافة البحث عن dogCharacteristic في البيانات المجمعة لـ dogDescription تحتاج إلى إضافة عبارة if لتحديد ما إذا كان لديك تطابق للبحث لكل كلب.

١. قم بتحديث التعليمات أسفل التعليق رقم 7 بعد التعليمات dogDescription = ourAnimals[i, 4] + "\n" + ourAnimals[i, 5]; بالتعليمات التالية:

```
if (dogDescription.Contains(dogCharacteristic))
{
    Console.WriteLine($"\nOur dog {ourAnimals[i, 3]} is a match!");
    Console.WriteLine(dogDescription);
}
```

افحص مثال التعليمات البرمجية السابق، عندما تجد عبارة if تطابقًا لـ dogCharacteristic في dogDescription يتم إخراج رسالة

حول تطابق الكلب والوصف إلى وحدة التحكم، لا تزال بحاجة إلى إنشاء رسالة لم يتم العثور على تطابقات "no matches found"

٢. أضف التعليمات `noMatchesDog = true`

قبل التعليق رقم 6 أضف التعليمات التالية:

```
bool noMatchesDog = true;
```

الآن، يمكنك تتبع عدم العثور على أي تطابقات باستخدام متغير التتبع هذا، عندما يتم تعيين الإعداد الافتراضي على صحيح true فهذا يعني: صحيح لا توجد كلاب تتطابق مع البحث "it's true that no dogs match for the search."

عندما يتم العثور على كلب، يمكنك "flip" قلب `noMatchesDog` من صحيح إلى خطأ from `true` to `false`

٣. داخل أقواس العبارة `if (dogDescription.Contains(dogCharacteristic))` أضف التعليمات التالية:

```
noMatchesDog = false;
```

٤. تأكد أن تعليماتك البرمجية موجودة بين أقواس `if (dogDescription.Contains(dogCharacteristic))` وتمت كتابتها بشكل صحيح.

يجب أن يكون لديك التعليمات البرمجية التالية:

```
if (dogDescription.Contains(dogCharacteristic))
    {
        Console.WriteLine($"\nOur dog
{ourAnimals[i, 3]} is a match!");
        Console.WriteLine(dogDescription);

        noMatchesDog = false;
    }
```

أخيرًا، تحتاج إلى إنشاء تعليمات تقرر ما إذا كان يجب كتابة رسالة: لم يتم العثور على تطابقات "no matches found" إلى وحدة التحكم.

٥. في نهاية التعليمات `case "2"` قبل التعليمات `Console.WriteLine("\n\rPress the Enter key to continue");` وقبل `break;` أضف التعليمات التالية:

```
if (noMatchesDog)
{
    Console.WriteLine("None of our dogs are a
match found for: " + dogCharacteristic);
}
```

هذه هي التعليمة البرمجية النهائية لهذا التمرين!

٦. احفظ تغييراتك.

راجع عملك

١. في موجه الأوامر TERMINAL command prompt لإنشاء التعليمات البرمجية للمشروع، أدخل الأمر التالي dotnet build

إذا كانت هناك أخطاء، فأنت بحاجة إلى قراءة رسائل الخطأ، أو استكشاف الأخطاء، وإجراء الإصلاحات، أو مراجعة الحل في المجلد النهائي Final folder

٢. قم بتشغيل التعليمات البرمجية لمشروعك في المحطة الطرفية terminal باستخدام الأمر dotnet run

عند تشغيل التعليمات البرمجية يتم عرض عنصرين من القائمة.

٣. في القائمة، أدخل 2 ثم أضغط Enter لاختبار بحث الكلب بشكل متكرر.

```
 - Enter nothing as input to test the null
entry behavior
 - Enter "scuba" as input to test the "match
not found"
 - Enter "golden" to get two matches
```

```
 - Enter "medium" to get one match
```

إذا سار كل شيء كما هو متوقع في كلتا الخطوتين، فتهانينا! بخلاف ذلك، ابحث عن الأخطاء، عن طريق التحقق من خطوات التمرين، إذا لزم الأمر، ابدأ من جديد، أو تحقق من المجلد النهائي Final للحصول على رمز الحل لهذا التمرين.

٤. اكتب exit في قائمة التطبيق، لإنهاء البرنامج، ثم أغلق لوحة المحطة الطرفية terminal panel

٦ اختبر معلوماتك

١- ما هو الخيار الذي يصف الغرض من TryParse()على نحو أفضل؟

- لتقسيم الجمل إلى كلمات مخزنة في مصفوفة.
- لاختبار عملية التحويل cast operation بأمان
- للعثور على سلسلة فرعية في سلسلة أكبر.

٢- بالنسبة للتعليمات `decimal.TryParse(numberString, out myConvert))` عندما يكون نوع `numberString` سلسلة، ما هو أفضل وصف لنوع القيمة لـ `myConvert` عندما ينجح `TryParse()`

- سلسلة تمثل أرقاماً
- قيمة منطقية
- رقم عشري decimal

راجع إجابتك

١ لاختبار عملية التحويل cast operation بأمان

صحيح يقوم TryParse() بإرجاع قيمة منطقية، إذا كان (true or false) للإشارة إلى إمكانية إكمال الإرسال.

٢ رقم عشري decimal

صحيح بالنسبة إلى المتغير `decimal.TryParse()` فإن `myConvert` هو رقم عشري.

٧ الملخص

لقد قمت بتحديث تطبيق تبني الحيوانات الأليفة، في هذا المشروع الإرشادي، لدعم وظيفة البحث بمصطلح واحد، وتحسين إمكانية قراءة المخرجات، وإضافة تبرع مقترح إلى الأوصاف.

في هذه الوحدة، مارست قدرتك على:

- تطبيق عبارات التكرار لجمع البيانات المُدخلة.
- استخدام معالجة البيانات.
- تنسيق إخراج البيانات.
- اختيار أنواع البيانات المناسبة، وتحويل أنواع البيانات بأمان.
- معالجة مصفوفات الأرقام، والنصوص، والأحرف، بالإضافة إلى إضافة البيانات، وإزالتها، وفرزها.
- تعديل وإنشاء سلاسل معقدة، من مصادر بيانات متعددة، وتنسيق البيانات للعرض عبر الحقول.

لقد قمت بتوزيع وظائف على فريق تطبيق Contos pet بما في ذلك:

- جمع إدخال مصطلح البحث الخاص بالحيوانات الأليفة.
- التكرار من خلال مصفوفة الحيوانات وتحديد "الكلاب"
- البحث في الوصف عن مصطلح مطابق، لكل كلب.
- عرض الكلاب التي تحتوي على مصطلح متطابق.
- تقديم بيانات التبرع المقترحة، المنسقة بالعملة المحلية.

الوحدة السابعة

مشروع التحدي - العمل مع البيانات المتغيرة

أظهر قدرتك على تطوير تطبيق وحدة تحكم، ينفذ تنسيق البيانات، واستخدام المصفوفات، لتقديم ميزة بحث متعددة المصطلحات.

الأهداف التعليمية

- تطبيق عبارات التكرار باستخدام البيانات المُدخلة.
- معالجة البيانات.
- تنسيق إخراج البيانات.
- اختيار أنواع البيانات المناسبة، وتحويل أنواع البيانات بأمان.
- إنشاء مصفوفات السلسلة ومعالجتها، وفرز بيانات المصفوفة.
- تعديل وإنشاء سلاسل معقدة من مصادر بيانات متعددة، وتنسيق البيانات للعرض.

محتويات الوحدة: -

١- مقدمة

٢- الاستعداد للتحدي

٣- إضافة ميزة بحث متعدد المصطلحات

٤- تمرين - إضافة رسوم متحركة محسنة للبحث

٥- اختبر معلوماتك

٦- الملخص

١ المقدمة

لنفترض أنك أحد المطورين الذين يعملون على تطبيق Contoso Pets للمساعدة في العثور على منازل جديدة للحيوانات الأليفة، يجمع الإصدار الأولي Starter من التطبيق، مصطلح بحث واحد يُستخدم للبحث في أوصاف الكلاب، يعد البحث عن مصطلح واحد مفيدًا، ولكن التعليقات المقدمة من الفريق تشير إلى أن التطبيق يحتاج إلى المزيد من وظائف البحث، يريد الفريق منك توسيع وظيفة البحث للسماح بالبحث عن مصطلحات متعددة، بالإضافة إلى ذلك، يريد الفريق تجربة تحديث للرسوم المتحركة "حالة البحث search status" التي تعرض تقدم البحث.

في مشروع ترميز التحدي هذا، يتم تطبيق معرفتك ومهاراتك في بيانات C# من أجل:

- إنشاء مصفوفات سلسلة string arrays عن طريق تقسيم السلاسل splitting strings
- معالجة بيانات المصفوفة، والتكرار عبر العناصر الموجودة في المصفوفة، وفرز بيانات المصفوفة.
- تعديل وإنشاء سلاسل strings من مصادر بيانات متعددة

باختصار، سوف تقوم بما يلي:

- تطوير وظيفة البحث متعدد المصطلحات، للبحث عن الكلاب.
- تحديث الرسوم المتحركة لحالة البحث search status animation والمعلومات والعد التنازلي.

في نهاية هذه الوحدة، ستكمل إصدارًا محدثًا من تطبيق Contoso Pets الذي يبحث في جميع الكلاب المتاحة للتبني، باستخدام مصطلحات بحث متعددة، ومحاكاة محسنة لحالة البحث.

٢ الاستعداد للتحدي

استخدم Visual Studio Code لتطوير تحديث تطبيق وحدة تحكم C# للتبني Contoso Pets يقوم التطبيق بالفعل بإنشاء بيانات نموذجية عن الحيوانات الأليفة المتاحة للتبني، وهو قادر على عرض معلومات وصف الحيوان الأليف، يوفر التطبيق البحث بمصطلح واحد ضمن بيانات الوصف لكل كلب، الميزة الرئيسية المضافة هي البحث بمصطلحات متعددة، التحسين الثانوي يتعلق بالرسوم المتحركة لحالة البحث search status animation ويتضمن إضافة عد تنازلي للبحث، في الحركة الرسومية.

نظرة عامة على مواصفات المشروع

تحديث تطبيق Contoso Pets الأولي starter الحالي، ليشمل بحثًا متعدد المصطلحات، وميزات الرسوم المتحركة "حالة البحث" المحسنة.

إضافة بحث متعدد المصطلحات لصفات الكلب

جمع مدخلات المستخدم للبحث عن خصائص الحيوانات الأليفة، بمصطلحات متعددة:

- يحتاج المستخدمون إلى توفير مصطلحات البحث مفصولة بفواصل.
- تخزين مصطلحات البحث في مصفوفة وفرز المصطلحات أبجديًا.

داخل حلقة مصفوفة الحيوانات التي تحدد الكلاب:

- التكرار من خلال مصطلحات البحث، للبحث في وصف كل كلب.
- البحث في الوصف المدمج/المركب combined description لمطابقة مصطلح.
- إخراج وصف كل كلب حيث يوجد مصطلح واحد أو أكثر متطابق.

بعد الخروج من حلقة مصفوفة البحث عن الحيوانات search Animals التي تحدد الكلاب:

إذا لم تكن توجد أي كلاب متطابقة مع أي من مصطلحات البحث المقدمة من المستخدمين، فقم بإخراج رسالة لا توجد كلاب متطابقة

إضافة تحسينات حالة البحث search status

تحديث الرسوم المتحركة animation

- ضبط المصفوفة `searchingIcons` لتشبه قرصًا دوارًا.
- اضبط حلقة الرسوم المتحركة، بحيث تعرض الحركة عدًا تنازليًا رقميًا من اثنين إلى صفر `(2.., 1.., 0..)`

تعليمات البدء Starter

يتضمن مشروع التعليمات البرمجية المبدئية Starter لدرس مشروع التحدي هذا، ملف Program.cs الذي يوفر ميزات التعليمات البرمجية التالية:

- تعلن التعليمات البرمجية عن المتغيرات المستخدمة لجمع بيانات الحيوانات الأليفة، وتحديدات عناصر القائمة، ومعالجتها.
- تعلن التعليمات البرمجية عن مصفوفة ourAnimals array
- تستخدم التعليمات البرمجية حلقة for حول بنية if-elseif-else لملء مصفوفة ourAnimals بمجموعة بيانات كنموذج/عينة.
- تعرض التعليمات البرمجية خيارات القائمة الرئيسية التالية، لاختيار المستخدم:

```
1. List all of our current pet information
2. Display all dogs with a specified characteristic

Enter menu item selection or type "Exit" to exit the program
```

- تقرأ التعليمة البرمجية لقائمة التطبيق، اختيار عنصر القائمة الخاص بالمستخدم، ويعرض رسالة تؤكد اختياره.
- Case 1 الحالة ١: يعرض سرد جميع معلومات الحيوانات الأليفة الحالية "List all of our current pet information" بيانات النموذج/العينة لجميع الحيوانات (كلبين وقطتين)
- Case 2 الحالة ٢: عرض جميع الكلاب ذات الخصائص المحددة .2" Display all dogs with a specified characteristic" هي المنطقة الأساسية التي تتم فيها إضافة تعليمة برمجية لوظيفة جديدة.
  - ضمن الحالة ٢ تحدد التعليمة البرمجية الأولية، الكلاب وتبحث عن مصطلح إدخال مستخدم واحد.
  - قبل البحث عن كل كلب، تعرض وحدة التحكم "رسومًا متحركة" بسيطة تحاكي تقدم حالة البحث.

هدفك هو تحديث التعليمات البرمجية الموجودة، لتطوير الميزات الرئيسية التي يطلبها فريقك:

- إضافة بحث خصائص/صفات متعددة للكلب.
- تحديث حركة البحث.

استخدم Visual Studio Code كبيئة تطوير، واختبر تطبيقك في كل مرحلة من مراحل عملية التطوير.

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي.

١. قم بتنزيل الملف المضغوط، يحتوي على مجلدات التعليمات البرمجية لمشروع التحدي.

- في المستعرض، افتح الارتباط Challenge-Project-variable-data-in-CSharp-main.zip لتنزيل الملف المضغوط.

٢. فك ضغط الملفات التي تم تنزيلها.

٣. افتح مجلد GuidedProject المصدر، في Visual Studio Code

- في Visual Studio Code في القائمة File حدد Open Folder
- انتقل إلى المجلد الذي يحتوي على الملفات المستخرجة، وحدد موقع المجلد المسمى Challenge-Project-variable-data-in-CSharp-main
- حدد Challenge-Project-variable-data-in-CSharp-main ثم حدد Select Folder

يجب أن تظهر طريقة عرض EXPLORER مجلدين فرعيين باسم Final وStarter

مراجعة ملف Starter

١. في Visual Studio Code انتقل إلى مجلد مشروع التحدي **starter**

٢. في شريط القائمة menu Bar حدد `Terminal > New Terminal`

٣. يجب أن تكون نافذة Terminal مفتوحة للمجلد Starter الذي يحتوي على ملف Program.cs قم ببناء الملف، وتشغيله عن بإدخال الأمر `dotnet run` في المحطة الطرفية.

٤. يجب عرض القائمة مع خيارين، أدخل 1 كما هو موضح في مثال التالي:

```
Welcome to the Contoso PetFriends app. Your main menu options are:
  1. List all of our current pet information
  2. Display all dogs with a specified characteristic

Enter your selection number (or type Exit to exit the program)
1
```

يجب عرض البيانات النموذجية بالحيوانات الأليفة الحالية، متبوعة بالرسالة:
اضغط على مفتاح الإدخال للمتابعة `Press the Enter key to continue`

٥. بعد الضغط على Enter يجب أن تظهر القائمة مرة أخرى، اختر الخيار 2 الموضح في المثال التالي:

```
Welcome to the Contoso PetFriends app. Your main menu options are:
 1. List all of our current pet information
 2. Display all dogs with a specified characteristic

Enter your selection number (or type Exit to exit the program)
2

Enter one desired dog characteristic to search for
```

٦. أدخل "large" لمصطلح البحث ثم اضغط على مفتاح Enter

٧. لاحظ رسالة البحث searching كما هو موضح:

```
searching our dog Nickname: gus for large ...
```

٨. لاحظ أن الرسالة يتم تشغيلها قبل كل بحث عن حيوان أليف، وتتغير النقاط `(.  ..  ...)` في النهاية في الرسم المتحرك، قم بتشغيل البحث مرة أخرى إذا فاتك ذلك.

٩. بمجرد انتهاء البحث، اضغط على Enter للعودة إلى القائمة، ثم اكتب "exit" واضغط على "Enter" لإغلاق التطبيق.

١٠. خذ بضع دقائق لتتعرف على تعليمات مشروع Project.cs started

ركز على المجالات التي تتطلب التحديثات والبحث والرسوم المتحركة.

لاحظ أن هناك بعض التعليقات المتبقية في التعليمات البرمجية التي تشير إلى مكان إجراء التحديث.

أنت الآن جاهز لبدء تمارين المشروع الإرشادية. حظ سعيد!

٣ إضافة ميزة بحث متعدد المصطلحات

يساعد تطبيق Contoso Pets في العثور على منازل جديدة للحيوانات الأليفة المهجورة، يريد الفريق تحسين ميزة البحث للسماح للمستخدمين بإدخال مصطلحات متعددة، عند البحث عن الكلاب لتبنيها.

مواصفات البحث

في تمرين التحدي الأول هذا، تحتاج إلى تحديث ميزة البحث عن مصطلح واحد، للسماح للمستخدمين بإدخال مصطلحات بحث متعددة، مفصولة بفواصل.

جمع مصطلحات بحث متعددة من المستخدم

- السماح للمستخدم بإدخال مصطلحات بحث متعددة عند البحث عن الكلاب
- يحتاج المستخدم إلى إرشادات مثل: إدخال مصطلحات البحث مفصولة بفواصل "enter the search terms separated by commas"
- فصل مصطلحات البحث الفردية عن سلسلة إدخال المستخدم، وتخزينها كقيم في مصفوفة.
- فرز المصطلحات في المصفوفة، بترتيب أبجدي رقمي.

تحديد الكلاب ذات الأوصاف المتطابقات لبحث المستخدم بمصطلح واحد أو أكثر

- أثناء تحديد كلب في `animalsArray` ابحث عن تطابقات لكل مصطلح أدخله المستخدم.
- لمطابقة المصطلح، قم بإخراج رسالة باسم الكلاب والمصطلح المطابق.

- مثال: كلبنا جيك يطابق بحثك عن شيبارد! Our dog Jake is a match for your search for sheppard!

- عند اكتمال جميع عمليات البحث عن مصطلح لوصف الكلب الحالي:
  - للحصول على تطابق واحد أو أكثر، قم إخراج الاسم البديل، ووصف الكلب الحالي.
  - بالنسبة للحصول على تطابق واحد أو أكثر، تأكد من عدم وجود تطابق، وعرض رسالة: لم يتم العثور على تطابقات لأي كلاب متوفرة "no matches found for any available dogs" (راجع العنصر التالي)
  - بعد اكتمال جميع عمليات البحث عن الكلاب دون أي تطابقات، اعرض رسالة: لم يتم العثور على أي تطابقات لأي كلاب متوفرة "No matches found for any available dogs"

فرض قواعد التحقق من الصحة التالية

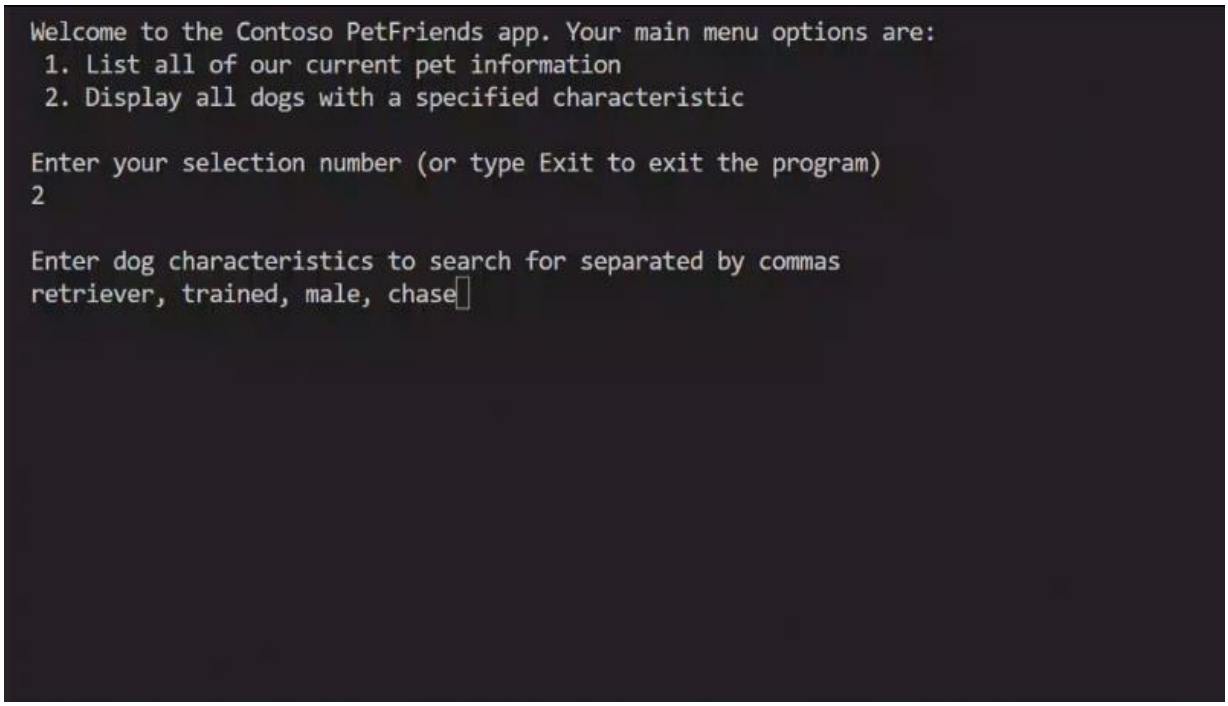
- لا يمكن أن تكون القيم خالية values can't be null
- لا يمكن أن تحتوي القيم على صفر من الأحرف zero characters
- أي قيود أخرى متروكة للمطور

تعليقات التعليمات البرمجية

- تعليقات التعليمات البرمجية المرقمة في تعليمات starter تقدم اقتراحات.
- التعليقات مرتبة تسلسليًا ولا تنطبق على تمرين التحدي هذا إلا التعليقات التي تبدأ برقم 1 ورقم 2 ورقم 3

عينة الإخراج

- مراجعة الصورة المتحركة التالية
  - لاحظ العرض المؤقت ل "`searching...`" متبوعاً بالمصطلح الحالي مثال: `"searching...retriever"`
  - لاحظ أن الترتيب الذي تظهر به المصطلحات في البحث أصبح أبجديًا رقميًا.
  - لاحظ استمرار رسائل التطابقات (على سبيل المثال: `"Our dog lola is a chase match!"`)



راجع نموذج الإخراج في القسم التالي، "كيفية التحقق من عملك" لمزيد من التوضيح لكيفية تنفيذ الميزات.

كيفية التحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. استخدم Visual Studio Code لبناء تطبيقك وتشغيله.

ملاحظة

يمكنك إنهاء اختبار التحقق قبل استكمال كافة خطوات التحقق، إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في لوحة Terminal اضغط على **Ctrl-C** بعد الخروج من التطبيق قيد التشغيل، أكمل عمليات التعديل التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، وحفظ تحديثاتك في ملف Program.cs ثم إعادة بناء التعليمات البرمجية وتشغيلها.

يجب أن يظهر موجه الأوامر الطرفي the starting point كنقطة للمشروع.

٢. في موجه الأوامر، أدخل **القائمة menu 2**

```
Welcome to the Contoso PetFriends app. Your
main menu options are:
 1. List all of our current pet information
 2. Display all dogs with a specified
characteristic

Enter your selection number (or type Exit to
exit the program)
2

Enter dog characteristics to search for
separated by commas
```

٣. في موجه الأوامر، أدخل **كبير، كريمي، ذهبي large, cream, golden** لاختباره عندما يتطابق أكثر من مصطلح بحث واحد مع أوصاف الكلب. تحقق من تحديث لوحة الوحدة الطرفية Terminal برسالة مشابهة لنموذج/لعينة إخراج التعليمات البرمجية:

```
Enter dog characteristics to search for separated by commas
large, cream, golden

Our dog Nickname: lola matches your search for cream
Our dog Nickname: lola matches your search for golden
Nickname: lola (ID #: d1)
Physical description: medium sized cream colored female golden retriever weighing about 45 pounds. housebroken.
Personality: loves to have her belly rubbed and likes to chase her tail. gives lots of kisses.

Our dog Nickname: gus matches your search for golden
Our dog Nickname: gus matches your search for large
Nickname: gus (ID #: d2)
Physical description: large reddish-brown male golden retriever weighing about 85 pounds. housebroken.
Personality: loves to have his ears rubbed when he greets you at the door, or at any time! loves to lean-in and give doggy hugs.

Press the Enter key to continue
```

٤. في موجه الأوامر، اضغط على مفتاح الإدخال enter للمتابعة إلى القائمة الرئيسية.

٥. في موجه الأوامر، أدخل **القائمة menu 2**

```
Welcome to the Contoso PetFriends app. Your
main menu options are:
 1. List all of our current pet information
 2. Display all dogs with a specified
characteristic

Enter your selection number (or type Exit to
exit the program)
2

Enter dog characteristics to search for
separated by commas
```

٦. في موجه الأوامر، أدخل **خطوطًا، كبيرة، ورمادية big, grey, stripes** لاختبارها عندما لا تتطابق أي من مصطلحات البحث مع أوصاف الكلاب، تحقق من تحديث لوحة الوحدة الطرفية Terminal برسالة مشابهة لنماذج إخراج التعليمات البرمجية:

```
Enter dog characteristics to search for
separated by commas
big, grey, stripes

None of our dogs are a match for: big, grey,
stripes


Press the Enter key to continue
```

٧. إذا أضفت المزيد من القيود للإدخالات الصالحة، فقم بتشغيل حالات الاختبار المناسبة للتحقق من عملك.

ملاحظة

إذا كانت التعليمات البرمجية تلبي بالمتطلبات، فيجب أن تكون قادرًا على إكمال كل خطوة بالترتيب، ورؤية النتائج المتوقعة في اجتياز اختبار واحد، إذا أضفت قيودًا إضافية، فقد تحتاج إلى الخروج من التطبيق، ثم تشغيل اجتياز اختبار منفصل لإكمال عملية التحقق.

تهانينا إذا نجحت في تمرين التحدي هذا.

٤ تمرين ـ إضافة رسوم متحركة محسنة للبحث

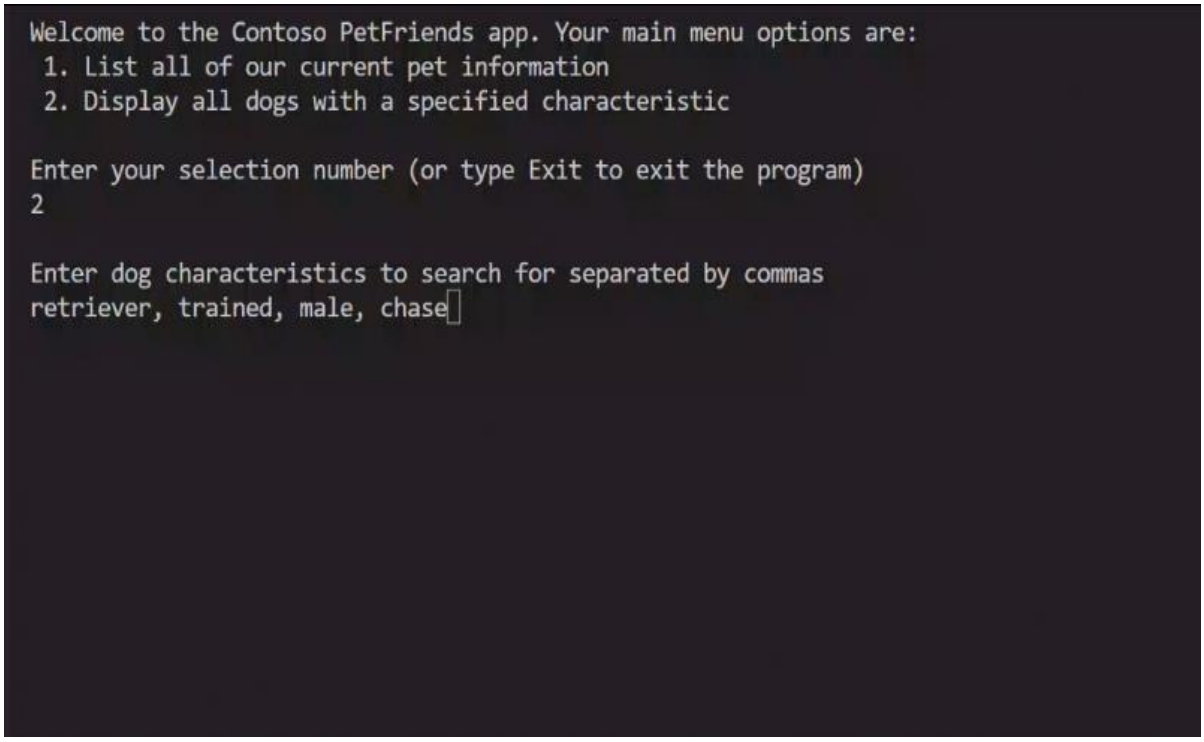
يساعد تطبيق Contoso Pets في العثور على منازل جديدة للحيوانات الأليفة المهجورة، جزء من طلب تحسين ميزة البحث، هو تحديث المحاكاة المتحركة animated simulation التي تشير إلى تقدم البحث.

مواصفات البحث بالرسوم المتحركة

في تمرين التحدي الثاني هذا، تحتاج إلى تحديث الرسوم المتحركة للبحث لتشبه الدوران، وإضافة العد التنازلي (2, 1, 0)

تغيير أيقونات الرسوم المتحركة "البحث" الحالية

- تحديث أيقونات الرسوم المتحركة الحالية `string[] searchingIcons = {". ", ".. ", "..."};`
- استخدام أيقونات جديدة تحاكي الدوران
- راجع صورة gif المتحركة التالية للحصول على مثال
- يمكنك تصميم حركة "البحث" لعرض "الدوران" والعمل بشكل مختلف عن المعروض في الصورة المتحركة.
- يجب الاستمرار في الكتابة فوق الرسوم المتحركة للبحث "`searching...`" بعد اكتمال كل رسم متحرك بحيث يبقى على نفس السطر، ولا يتم عرضه بعد توقف الرسم المتحرك.



إضافة العد التنازلي إلى الرسوم المتحركة "البحث"

- راجع الصورة المتحركة السابقة - لاحظ العد التنازلي في الإخراج
  - `searching...retriever / 2`
  - يتم عرض الرقم السابق "2" على أنه "1" وأخيرًا على أنه "0" مع العد التنازلي.
- تحديث الحلقة التي تحتوي على الرسوم المتحركة "البحث" بحيث يمكن للحلقة عرض العد التنازلي

تعليقات التعليمات البرمجية

- تعليقات التعليمات البرمجية المرقمة في تعليمات starter تقدم اقتراحات.
- التعليقات مرتبة تسلسليًا ولا تنطبق على تمرين التحدي هذا إلا التعليقات التي تبدأ برقم 4 ورقم 5

كيفية التحقق من عملك

للتحقق من أن التعليمات البرمجية تلبي المتطلبات المحددة، أكمل خطوات التحقق:

١. استخدم Visual Studio Code لبناء تطبيقك وتشغيله.

ملاحظة

يمكنك إنهاء اختبار التحقق قبل استكمال كافة خطوات التحقق، إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في لوحة Terminal اضغط على **Ctrl-C** بعد الخروج من التطبيق قيد التشغيل، أكمل عمليات التعديل التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، وحفظ تحديثاتك في ملف Program.cs ثم إعادة بناء التعليمات البرمجية وتشغيلها.

يجب أن يظهر موجه الأوامر الطرفي the starting point كنقطة للمشروع.

٢. في موجه الأوامر، أدخل **القائمة menu 2**

```
Welcome to the Contoso PetFriends app. Your
main menu options are:
 1. List all of our current pet information
 2. Display all dogs with a specified
characteristic

Enter your selection number (or type Exit to
exit the program)
2

Enter dog characteristics to search for
separated by commas
```

٣. في موجه الأوامر، أدخل **الذهبي، والكبير golden, big** واضغط على Enter لاختبار رسائل حالة البحث مع الرسوم المتحركة والعد التنازلي

٤. تحقق من تحديث لوحة التحكم console panel برسائل حالة "البحث searching" مؤقتة مشابهة لنماذج مخرجات التعليمات:

```
Enter dog characteristics to search for separated
by commas
golden, big
searching our dog Nickname: lola for big / 2


searching our dog Nickname: lola for big -- 1


searching our dog Nickname: lola for golden \ 1


searching our dog Nickname: lola for golden * 0
```

٥. يجب أن يتم عرض جميع مصطلحات البحث التي أدخلها المستخدم مع الرسوم المتحركة الدورانية "البحث searching" والعد التنازلي لكل كلب حيث يقوم سطر "البحث عن كلبنا ...searching our dog" في الإخراج بالكتابة فوق السطر السابق "يحل محله" لإنشاء رسم متحرك.

٦. راجع الصورة المتحركة السابقة للبرنامج قيد التشغيل للحصول على مزيد من الأمثلة على حركة الحالة "البحث" مع العد التنازلي.

٧. التحقق من الصحة بعد توقف البحث، توقف عرض الحركة "searching" والعد التنازلي.

ملاحظة

إذا كانت التعليمات البرمجية تلبي بالمتطلبات، فيجب أن تكون قادرًا على إكمال كل خطوة بالترتيب ورؤية النتائج المتوقعة في اجتياز اختبار واحد.

تهانينا إذا نجحت في تمرين التحدي هذا، أنت تستحق مكافئة.

٥ اختبر معلوماتك

١- لفرز مصفوفة string لماذا من المهم استخدام الأسلوب `String.Trim()` على كل عنصر من عناصر المصفوفة قبل الفرز؟

- يمكن فقط فرز مصفوفات string التي تحتوي على عناصر مقتطعة/مقلمة باستخدام `String.Trim()`
- يقوم `String.Trim()` بإزالة المسافة البيضاء البادئة التي قد يتم فرزها قبل الأرقام والحروف.
- يكون فرز المصفوفة أسرع بشكل ملحوظ باستخدام `String.Trim()` على كل عنصر.

٢- تعد كل من foreach و for خيارين جيدين لتكرار خلال المصفوفات الصغيرة، ذات البعد الواحد مثل {"cat", "fox", "dog", "snake", "eagle"} ولكن متى يكون من الأفضل استخدام حلقة for؟

- عندما يكون للعناصر ترتيب فرز أبجدي.
- لإجراء بحث على كل عنصر في المصفوفة.
- لتقييم مجموعة محددة من العناصر

راجع إجابتك

١ يقوم `String.Trim()` بإزالة المسافة البيضاء البادئة التي قد يتم فرزها قبل الأرقام والحروف

صحيح يقوم String.Trim() بإزالة كل من المسافة البيضاء البادئة واللاحقة

٢ لتقييم مجموعة محددة من العناصر

صحيح تكون for أفضل عندما تكون نطاقات الفهرس من الأول إلى الفهرس الأخير ليست مطلوبة.

٦ الملخص

التحدي الذي واجهته هو تحديث تطبيق لدعم البحث متعدد المصطلحات، ولتحسين حركة معلومات حالة البحث

في هذه الوحدة، قمت بتطوير التعليمات البرمجية التي تجمع بين:

- التكرار باستخدام إدخال البيانات.
- معالجة البيانات.
- تنسيق إخراج البيانات.
- اختيار أنواع البيانات المناسبة وتحويل أنواع البيانات بأمان.
- إنشاء مصفوفات السلسلة string ومعالجتها، وفرز بيانات المصفوفة.
- تعديل وإنشاء سلاسل معقدة complex strings من مصادر بيانات متعددة، وتنسيق البيانات للعرض.

لقد قدمت وظائف لفريق تطبيق Contoso pet، بما في ذلك:

- جمع خصائص الحيوانات الأليفة لمدخلات متعددة لمصطلح البحث.
- تخزين مصطلحات إدخال البحث في مصفوفة مرتبة.
- عرض الكلاب التي لها مصطلح مطابق.
- توفير رسوم متحركة محسنة لحالة البحث مع العد التنازلي

الحصول على شهادة مجانية تم التحقق منها

قامت Microsoft بشراكة مع freeCodeCamp.org لتقديم برنامج تدريب وشهادة ل C# من خلال إكمال هذا الفصل، تكون بالفعل على بعد خطوة واحدة من أن يتم اعتمادك، لاستكشاف الشهادة التأسيسية التي تقدمها freeCodeCamp تفضل بزيارة https://aka.ms/csharp-certification

محتويات الكتاب:

إضافة ميزة بحث متعدد المصطلحات

تمرين - إضافة رسوم متحركة محسنة للبحث

اختبر معلوماتك

الملخص

الحصول على شهادة مجانية، معتمدة من مايكروسوفت تم التحقق منها ٨٢٨