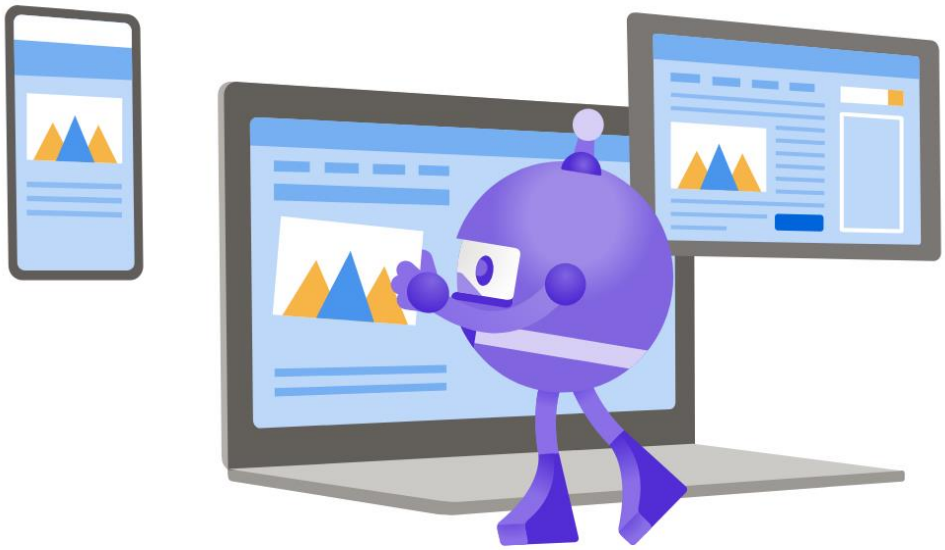
C Sharp بالعربي

إنشاء أساليب في تطبيقات C#

Object-oriented programming

مجتمع المبرمجين العرب

تعرف على كيفية إنشاء أساليب لتطبيقات وحدة تحكم C# استخدم المعلمات parameters والقيم المرجعة return values لمعالجة البيانات عبر التطبيقات، تعلم كيفية استخدام الوسيطات المسماة والاختيارية named and optional arguments لتبسيط التعليمات البرمجية.



المتطلبات الأساسية

- خبرة في مهام الترميز الأساسية مثل إنشاء مثيل للمتغيرات، واستخدام أنواع البيانات المختلفة، وإرسال الإخراج إلى نافذة وحدة التحكم.
- تجربة استخدام Visual Studio Code لإنشاء تطبيقات وحدة تحكم بسيطة وتشغيلها.
- تجربة استخدام أنواع بيانات C# بما في ذلك int, string, double, and arrays
- خبرة في استخدام عبارات switch وعبارات if-else وحلقات for
- تجربة استخدام string.Split

محتويات الكتاب

الوحدة الأولي:
كتابة أسلوب C# الأول

تعلم كيفية إنشاء أساليبك الخاصة، لتنفيذ مهام معينة.

الأهداف التعليمية

- إنشاء أسلوب C# الأول.
- تحديد أجزاء التعليمات البرمجية التي يمكن تقسيمها إلى وحدات.
- استخدام أساليب لتنظيم التعليمات البرمجية في مهام محددة.

محتويات الوحدة: -

١- مقدمة

٢- فهم بناء جملة الأساليب

٣- إنشاء أسلوبك الأول

٤- إنشاء أساليب قابلة لإعادة الاستخدام reusable methods

٥- إنشاء التعليمات البرمجية باستخدام الأساليب

٦- تمرين - تحدي إنشاء أسلوب قابل لإعادة الاستخدام

٧- مراجعة حل إنشاء أسلوب قابل لإعادة الاستخدام

٨- اختبر معلوماتك

٩- الملخص

١ المقدمة

تسمح لك لغة البرمجة C# بإنشاء جميع أنواع التطبيقات القوية، لنفترض أنك تريد كتابة تعليمات برمجية لتنفيذ عمليات على مدخلات مستخدم غير متوقعة، وترغب في إضافة منطق للتعامل مع الإدخال غير الصحيح، وتحويل الإدخال إلى تنسيق قابل للعمل، وتنفيذ العمليات الصحيحة، مع زيادة حجم تطبيقاتك، يمكن أن يصبح تتبع التعليمات البرمجية بسرعة تحدياً.

تعد الأساليب، التي تسمى أيضاً الدوال أو الوظائف، جزءًا أساسياً من الحفاظ على التعليمات البرمجية منظمة وفعالة، وسهلة القراءة، الأسلوب هو وحدة معيارية/نمطية من التعليمات البرمجية، ومفهوم برمجة أساسي، غالباً ما يتم تصميم أسلوب لتنفيذ مهمة معينة، ويحتوي على التعليمات البرمجية لتنفيذ تلك المهمة، يجب أن يعكس اسم الأسلوب مهمته بوضوح، ما يحسن سهولة قراءة التعليمات البرمجية، سيساعدك تعلم كيفية استخدام الأساليب على إنشاء تطبيقات غنية بالميزات بسرعة أكبر.

في هذه الوحدة، ستتعلم إنشاء أساليبك الخاصة لتنفيذ مهام معينة. ستتعلم كيف يمكن للأساليب تبسيط التعليمات البرمجية، وتقصيرها، بالإضافة إلى الحفاظ على تنظيم الأشياء، وسهولة العثور عليها.

٢ فهم بناء جملة الأساليب

أنت على دراية بالفعل ببعض الأساليب، مثل `Console.WriteLine()` or `random.Next()` ربما تقدر كيف تعمل الأساليب على تبسيط المهام، وتسمح لك بإنشاء تعليماتك البرمجية بسهولة أكبر، في هذه الوحدة، ستتعلم كيفية إنشاء أساليبك الخاصة.

كيفية عمل الأساليب

تبدأ عملية تطوير أسلوب بإنشاء توقيع أسلوب method signature يعلن توقيع الأسلوب عن نوع الإرجاع method's return type والاسم name ومعلمات الإدخال input parameters الخاصة بالأسلوب، على سبيل المثال، ضع في اعتبارك توقيع الأسلوب التالي.

```
void SayHello();
```

اسم الأسلوب هو SayHello نوع الإرجاع الخاص به هو `void` مما يعني أن الأسلوب لا يقوم بإرجاع أي بيانات. ومع ذلك، يمكن أن ترجع الأساليب قيمة من أي نوع بيانات، مثل bool, int, double, and arrays يجب إضافة معلمات الأسلوب Method parameters إن وجدت، داخل قوسين () يمكن أن تقبل الأساليب معلمات متعددة من أي نوع بيانات، في هذا المثال، لا يحتوي الأسلوب على معلمات.

قبل أن تتمكن من تشغيل أسلوب، تحتاج إلى إضافة تعريف، يستخدم تعريف الأسلوب أقواساً {} لاحتواء التعليمات البرمجية، التي يتم تنفيذها عند استدعاء الأسلوب، على سبيل المثال:

```
void SayHello()
{
    Console.WriteLine("Hello World!");
}
```

الآن سيقوم الأسلوب بطباعة Hello World! كلما تم استدعاؤه.

استدعاء أسلوب

يتم استدعاء أسلوب باستخدام اسمه، وتضمين أي وسيطات arguments مطلوبة، ضع في الحسبان ما يلي:

```
Console.Write("Input!");
```

الكلمة "Input!" هي الوسيطة المقدمة إلى الأسلوب Write()

يمكن استدعاء أسلوب قبل تعريفه أو بعده، على سبيل المثال، يمكن تعريف الأسلوب SayHello واستدعاؤه باستخدام بناء الجملة التالي:

```
SayHello();

void SayHello()
{
    Console.WriteLine("Hello World!");
}
```

لاحظ أنه ليس من الضروري تعريف الأسلوب قبل استدعائه، تتيح لك هذه المرونة تنظيم التعليمات البرمجية بالشكل الذي تراه مناسبًا، من الشائع تحديد كافة الأساليب في نهاية البرنامج، على سبيل المثال:

```
int[] a = {1,2,3,4,5};

Console.WriteLine("Contents of Array:");

PrintArray();

void PrintArray()
{
    foreach (int x in a)
    {
        Console.Write($"{x} ");
    }
```

```
    Console.WriteLine();
}
```

تنفيذ الأسلوب

عند استدعاء أسلوب، سيتم تنفيذ التعليمات البرمجية في نص الأسلوب "الكتلة البرمجية الخاصة به"، وهذا يعني أن التحكم في التنفيذ يتم تمريره من مستدعي الأسلوب، إلى الأسلوب، يتم إرجاع عنصر التحكم إلى المتصل بعد إكمال الأسلوب من تنفيذها، على سبيل المثال، ضع في اعتبارك التعليمات البرمجية التالية:

```
Console.WriteLine("Before calling a
method");

SayHello();

Console.WriteLine("After calling a method");

void SayHello()
{
    Console.WriteLine("Hello World!");
}
```

تعرض هذه التعليمات البرمجية الإخراج التالي:

```
Before calling a method
Hello World!
After calling a method
```

بمجرد تعريف الأسلوب، يمكن استدعاؤه في أي وقت، عدة مرات، بقدر ما تحتاج إلى استخدامه، يمكنك استخدام أساليب داخل عبارات if-else وعبارات التكرار for وعبارة switch حتى لتهيئة المتغيرات، وأكثر من ذلك بكثير!

أفضل الممارسات

عند اختيار اسم أسلوب method name من المهم اختيار اسم موجزًا، وتوضيح المهمة التي يؤديها الأسلوب، يجب أن تكون أسماء الأساليب مختصرة، ويجب ألا تبدأ عادة بأرقام، وتستخدم نسق الجمل Pascal case هو كتابة العبارات دون مسافات أو علامات ترقيم، بل يُفصل بين الكلمات بحرف كبير وحيد، أشهر الأمثلة على ذلك تتضمن iPhone

يجب أن تصف أسماء المعلمات parameters نوع المعلومات التي تمثلها المعلمة، ضع في اعتبارك تواقيع الأسلوب التالية:

```
void ShowData(string a, int b, int c);
void DisplayDate(string month, int day, int year);
```

يصف الأسلوب الثاني نوع البيانات التي يتم عرضها، ويوفر أسماء وصفية للمعلمات parameters

الآن بعد أن انتهيت من الأساسيات، أنت جاهز للبدء في كتابة أساليبك الخاصة.

٣ إنشاء أسلوبك الأول

عادة ما يتم إنشاء أسلوب لتنفيذ مهمة معينة، في هذا التمرين، ستقوم بإنشاء أسلوب يقوم بإنشاء وعرض خمسة أرقام عشوائية. لنبدأ!

إعداد بيئة الترميز

تتضمن هذه الوحدة أنشطة عملية، ترشدك خلال عملية إنشاء التعليمات البرمجية التوضيحية وتشغيلها، يتم تشجيعك على إكمال هذه الأنشطة باستخدام Visual Studio Code كبيئة تطوير، سيساعدك استخدام Visual Studio Code لهذه 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 مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٢. في قائمة Visual Studio Code File 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 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 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. في موجه الأوامر Terminal 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 لك الأمر مجلدات CsharpProjects وTestProject ويستخدم TestProject كاسم للملف csproj.

٧. في لوحة 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 لوحة EXPLORER لعرض ملف التعليمات البرمجية في لوحة المحرر، حدد Program.cs

٩. احذف أسطر التعليمات البرمجية الموجودة.

ستستخدم مشروع وحدة تحكم C# هذا لإنشاء نماذج التعليمات البرمجية وبنائها، وتشغيلها أثناء هذه الوحدة.

١٠. أغلق لوحة Terminal

إنشاء أسلوب لعرض أرقام عشوائية

لإنشاء أسلوب، قم أولاً بإنشاء توقيع الأسلوب method signature ثم أضف نص الأسلوب method body لإنشاء توقيع الأسلوب، عليك أن تعلن عن نوع الإرجاع واسم الأسلوب، والمعلمات return type, method name, and parameters

إنشاء نص الأسلوب باستخدام الأقواس {} التي تحتوي على التعليمات البرمجية.

١. أدخل التعليمات البرمجية التالية في محرر Visual Studio Code

```
void DisplayRandomNumbers();
```

في هذه الحالة، يحتاج الأسلوب فقط إلى إنشاء المعلومات وعرضها، لذا فإن نوع الإرجاع هو void بمعني لاغي أو فارغ، في الوقت الحالي، لا تحتاج إلى تضمين أي معلمات parameters

٢. لإنشاء نص الأسلوب، قم بإزالة الفاصلة المنقطة semicolon ; وتحديث التعليمات البرمجية إلى ما يلي:

```
void DisplayRandomNumbers()
{
    Random random = new Random();
}
```

هنا، يمكنك إنشاء كائن Random object يُستخدم لإنشاء الأرقام العشوائية.

٣. لعرض خمسة أعداد صحيحة عشوائية، أضف حلقة تكرار for إلى أسلوبك:

```
void DisplayRandomNumbers()
{
    Random random = new Random();

    for (int i = 0; i < 5; i++)
    {
        Console.Write($"{random.Next(1, 100)} ");
    }
}
```

في هذه التعليمات البرمجية، يمكنك إنشاء رقم بين ١ و ٩٩ يمكنك أيضاً إضافة مسافة بعد طباعة الرقم، بعد ذلك، ستعرض سطرًا جديدًا قبل إنهاء الأسلوب.

٤. تحديث الأسلوب الخاص بك بالتعليمات البرمجية التالية:

```
void DisplayRandomNumbers()
{
    Random random = new Random();

    for (int i = 0; i < 5; i++)
    {
        Console.Write($"{random.Next(1, 100)} ");
    }

    Console.WriteLine();
}
```

الآن سيضيف أسلوبك سطرًا جديدًا بعد عرض الأرقام.

استدعاء الأسلوب

١. أدخل سطر فارغ جديد أعلى الأسلوب DisplayRandomNumbers

٢. أدخل التعليمات البرمجية التالية في السطر الفارغة:

```
Console.WriteLine("Generating random numbers:");
DisplayRandomNumbers();
```

٣. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
Console.WriteLine("Generating random numbers:");
DisplayRandomNumbers();

void DisplayRandomNumbers()
{
    Random random = new Random();

    for (int i = 0; i < 5; i++)
```

```
    {
        Console.Write($"{random.Next(1, 100)} ");
    }

    Console.WriteLine();
}
```

لاحظ كيف أن استخدام أسلوب يجعل التعليمات البرمجية سهلة الفهم، بدلاً من قضاء الوقت في محاولة فك الحلقة for من تلقاء نفسها، يمكنك قراءة اسم الأسلوب بسرعة لمعرفة أن هذه التعليمات تعرض أرقاماً عشوائية.

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقك من الوحدة الطرفية المتكاملة the Integrated Terminal والتحقق من أن التعليمات تعمل بشكل صحيح. لنبدأ.

١. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٢. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٣. في موجه الأوامر Terminal أدخل dotnet run

٤. للتحقق من أن التعليمات البرمجية تعمل كما هو متوقع، تحقق من أن إخراج تطبيقك مشابه للإخراج التالي (مع مراعاة أن الأرقام تم إنشاؤها عشوائيًا):

```
17 29 46 36 3
```

إذا كانت التعليمات البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها للعثور على الخطأ وإجراء التعديلات، قم بتشغيل التعليمات مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، تابع تحديث التعليمات البرمجية وتشغيلها حتى تنتج النتائج المتوقعة.

خلاصة

إليك ما تعلمته عن الأساليب حتى الآن:

- إنشاء أسلوب عن طريق الإعلان عن نوع الإرجاع، والاسم، ومعلمات الإدخال، والنص الأساسي للأسلوب.
- يجب أن تعكس أسماء الأساليب بوضوح، المهمة التي يقوم بها الأسلوب.
- استخدم أسلوباً عن طريق استدعاء اسمه، وتضمين الأقواس ()

٤ إنشاء أساليب قابلة لإعادة الاستخدام reusable methods

أثناء تطوير التطبيقات، قد تجد نفسك تكتب التعليمات البرمجية للقيام بنفس الشيء مرارًا وتكرارًا، بدلًا من كتابة التعليمات البرمجية المكررة، يؤدي استخدام أسلوب لتنفيذ نفس المهمة، إلى تقصير التعليمات البرمجية الخاصة بك، ويساعدك على تطوير التطبيقات بشكل أسرع، في هذا التمرين، ستحدد التعليمات البرمجية المتكررة، وتستبدلها بأسلوب قابل لإعادة الاستخدام. لنبدأ!

تحديد التعليمات البرمجية المكررة Identify duplicated code

في هذه المهمة، ستلقي نظرة على تطبيق يتتبع أوقات تناول الدواء عبر مناطق زمنية مختلفة، يدخل المستخدم المنطقة الزمنية الحالية، والمنطقة الزمنية الوجهة الخاصة به، يتم عرض جدول أدويتهم ثم تعديله ليناسب المنطقة الزمنية الجديدة.

١. في محرر Visual Studio Code، احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. انسخ التعليمات البرمجية التالية، وألصقها في محرر Visual Studio Code

```
using System;

int[] times = {800, 1200, 1600, 2000};
int diff = 0;

Console.WriteLine("Enter current GMT");
int currentGMT =
Convert.ToInt32(Console.ReadLine());
```

```
Console.WriteLine("Current Medicine
Schedule:");

/* Format and display medicine times */
foreach (int val in times)
{
    string time = val.ToString();
    int len = time.Length;

    if (len >= 3)
    {
        time = time.Insert(len - 2, ":");
    }
    else if (len == 2)
    {
        time = time.Insert(0, "0:");
    }
    else
    {
        time = time.Insert(0, "0:0");
    }

    Console.Write($"{time} ");
}

Console.WriteLine();

Console.WriteLine("Enter new GMT");
int newGMT =
Convert.ToInt32(Console.ReadLine());
```

```
if (Math.Abs(newGMT) > 12 ||
Math.Abs(currentGMT) > 12)
{
    Console.WriteLine("Invalid GMT");
}
else if (newGMT <= 0 && currentGMT <= 0 ||
newGMT >= 0 && currentGMT >= 0)
{
    diff = 100 * (Math.Abs(newGMT) -
Math.Abs(currentGMT));

    /* Adjust the times by adding the
difference, keeping the value within 24 hours
*/
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
else
{
    diff = 100 * (Math.Abs(newGMT) +
Math.Abs(currentGMT));

    /* Adjust the times by adding the
difference, keeping the value within 24 hours
*/
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
```

```
Console.WriteLine("New Medicine Schedule:");

/* Format and display medicine times */
foreach (int val in times)
{
    string time = val.ToString();
    int len = time.Length;

    if (len >= 3)
    {
        time = time.Insert(len - 2, ":");
    }
    else if (len == 2)
    {
        time = time.Insert(0, "0:");
    }
    else
    {
        time = time.Insert(0, "0:0");
    }

    Console.Write($"{time} ");
}

Console.WriteLine();
```

٣. لاحظ أن هناك العديد من حلقات for التي يتم تكرارها بتعليمات برمجية مماثلة.

توجد حلقتان من حلقات foreach تقومان بتنسيق وعرض أوقات الدواء، هناك حلقتان أخريات لضبط الوقت وفقًا لفارق المنطقة الزمنية.

أثناء كتابة التعليمات البرمجية، قد تجد نفسك تكرر كتل التعليمات البرمجية للقيام بنفس المهمة، هذه فرصة مثالية لدمج تعليماتك البرمجية باستخدام أسلوب، لتنفيذ المهمة بدلاً من ذلك. دعونا نتمرن.

إنشاء أساليب لتنفيذ المهام المتكررة

الآن بعد أن حددت التعليمات البرمجية المتكررة، يمكنك إنشاء أسلوب لاحتواء التعليمات البرمجية، وإزالة التكرارات، استخدام الأساليب له ميزات إضافية، تتمثل في تقصير التعليمات البرمجية، وتحسين قابلية القراءة!

تقوم حلقات foreach بتنسيق وعرض قيم الوقت، لكي تتمكن من إعطاء الطريقة اسمًا يعكس تلك المهمة بوضوح، يمكنك أن تفعل الشيء نفسه مع حلقة for التي تضبط الأوقات. هيا بنا نبدأ!

١. إنشاء سطر فارغ، في نهاية التعليمات البرمجية السابقة.

٢. في السطر الفارغ الجديد، قم بإنشاء توقيع أسلوب method signature عن طريق إدخال التعليمات البرمجية التالية:

```
void DisplayTimes()
{

}
```

٣. لتعريف نص الأسلوب، قم بتحديث الأسلوب DisplayTimes عن طريق نسخ ولصق كتلة foreach كما يلي:

```
void DisplayTimes()
{
    /* Format and display medicine times */
    foreach (int val in times)
    {
        string time = val.ToString();
        int len = time.Length;

        if (len >= 3)
```

```
        {
            time = time.Insert(len - 2, ":");
        }
        else if (len == 2)
        {
            time = time.Insert(0, "0:");
        }
        else
        {
            time = time.Insert(0, "0:0");
        }

        Console.Write($"{time} ");
    }

    Console.WriteLine();
}
```

في هذا الأسلوب، يمكنك تضمين استدعاء Console.WriteLine في النهاية، لإلحاق سطر جديد بعد عرض الأوقات، بعد ذلك، ستقوم بإنشاء أسلوب أخر لضبط الأوقات، وفقًا لفرق المنطقة الزمنية.

٤. أدخل سطر فارغ جديدا في نهاية التعليمات البرمجية السابقة.

٥. في السطر الفارغ، قم بإنشاء توقيع أسلوب عن طريق إدخال التعليمات البرمجية التالية:

```
void AdjustTimes()
{

}
```

٦. قم بتحديث أسلوب AdjustTimes عن طريق نسخ ولصق حلقة for كما يلي:

```
void AdjustTimes()
{
    /* Adjust the times by adding the
difference, keeping the value within 24
hours */
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
```

الخطوة ٣: استدعاء الأساليب

في هذه المهمة، ستقوم بحذف كتل التعليمات البرمجية المتكررة، واستبدالها باستدعاءات للأساليب التي قمت بإنشائها.

١. حدد موقع المثيل الأول لحلقة foreach المتكررة، ضمن التعليق تنسيق وعرض أوقات الدواء "Format and display medicine times"

```
Console.WriteLine("Current Medicine Schedule:");

/* Format and display medicine times */
foreach (int val in times)
{
    ...
}

Console.WriteLine();

Console.WriteLine("Enter new GMT");
```

٢. استبدل التعليمات التي حددتها باستدعاء الأسلوب DisplayTimes يجب أن ينتج عن الاستبدال التعليمات البرمجية التالية:

```
Console.WriteLine("Current Medicine Schedule:");
DisplayTimes();

Console.WriteLine("Enter new GMT");
```

بعد ذلك، ستقوم باستبدال المثيل الثاني من التعليمات البرمجية المتكررة.

٣. حدد موقع المثيل الثاني للحلقة foreach تحت التعليق تنسيق وعرض أوقات الدواء "Format and display medicine times"

```
Console.WriteLine("New Medicine Schedule:");

/* Format and display medicine times */
foreach (int val in times)
{
    ...
}

Console.WriteLine();
```

٤. استبدل التعليمات التي حددتها باستدعاء الأسلوب DisplayTimes يجب أن ينتج عن الاستبدال التعليمات البرمجية التالية:

```
Console.WriteLine("New Medicine Schedule:");
DisplayTimes();
```

لاحظ كيف أن استخدام أسلوب بدلًا من كتلة كبيرة من التعليمات البرمجية، يوفر المزيد من الوضوح، ويجعل التعليمات البرمجية أسهل في الفهم، دعونا نفعل الشيء نفسه مع الأسلوب AdjustTimes الذي قمت بإنشائه.

٥. حدد موقع التعليمات التالية باستخدام حلقات for المكررة:

```
else if (newGMT <= 0 && currentGMT <= 0 ||
newGMT >= 0 && currentGMT >= 0)
{
    diff = 100 * (Math.Abs(newGMT) -
Math.Abs(currentGMT));

    /* Adjust the times by adding the
difference, keeping the value within 24 hours
*/
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
else
{
    diff = 100 * (Math.Abs(newGMT) +
Math.Abs(currentGMT));

    /* Adjust the times by adding the
difference, keeping the value within 24 hours
*/
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
```

٦. استبدل مثيلات التعليمات المتكررة، ضمن التعليق ضبط الأوقات عن طريق إضافة الفرق "Adjust the times by adding the difference" باستدعاءات أسلوب AdjustTimes يجب أن ينتج عن الاستبدال التعليمات البرمجية التالية:

```
else if (newGMT <= 0 && currentGMT <= 0 ||
newGMT >= 0 && currentGMT >= 0)
{
    diff = 100 * (Math.Abs(newGMT) -
Math.Abs(currentGMT));
    AdjustTimes();
}
else
{
    diff = 100 * (Math.Abs(newGMT) +
Math.Abs(currentGMT));
    AdjustTimes();
}
```

الآن تم استبدال كل التعليمات البرمجية المتكررة، بأسلوبك الجديد. لاحظ مدى سهولة قراءة التعليمات البرمجية وإيجازها!

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق الخاص بك، من الوحدة الطرفية المتكاملة والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
int[] times = {800, 1200, 1600, 2000};
int diff = 0;

Console.WriteLine("Enter current GMT");
int currentGMT =
Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Current Medicine
Schedule:");
```

```
DisplayTimes();

Console.WriteLine("Enter new GMT");
int newGMT =
Convert.ToInt32(Console.ReadLine());

if (Math.Abs(newGMT) > 12 ||
Math.Abs(currentGMT) > 12)
{
    Console.WriteLine("Invalid GMT");
}
else if (newGMT <= 0 && currentGMT <= 0 ||
newGMT >= 0 && currentGMT >= 0)
{
    diff = 100 * (Math.Abs(newGMT) -
Math.Abs(currentGMT));
    AdjustTimes();
}
else
{
    diff = 100 * (Math.Abs(newGMT) +
Math.Abs(currentGMT));
    AdjustTimes();
}

Console.WriteLine("New Medicine Schedule:");
DisplayTimes();

void DisplayTimes()
{
    /* Format and display medicine times */
    foreach (int val in times)
    {
        string time = val.ToString();
```

```
        int len = time.Length;

        if (len >= 3)
        {
            time = time.Insert(len - 2, ":");
        }
        else if (len == 2)
        {
            time = time.Insert(0, "0:");
        }
        else
        {
            time = time.Insert(0, "0:0");
        }

        Console.Write($"{time} ");
    }
    Console.WriteLine();
}

void AdjustTimes()
{
    /* Adjust the times by adding the
difference, keeping the value within 24 hours
*/
    for (int i = 0; i < times.Length; i++)
    {
        times[i] = ((times[i] + diff)) % 2400;
    }
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في لوحة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٤. في موجه الأوامر Terminal أدخل dotnet run

أدخل 6+ and 6- لمطالبات GMT

٥. تحقق من أن تعليماتك البرمجية تنتج الإخراج التالي:

```
Enter current GMT
-6
Current Medicine Schedule:
8:00 12:00 16:00 20:00
Enter new GMT
+6
New Medicine Schedule:
20:00 0:00 4:00 8:00
```

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

اختبر معلوماتك

١ لنفترض أن هناك أسلوب يسمى ConvertCurrency أي من سطور التعليمات البرمجية التالية يعد توقيعًا صالحًا للأسلوب؟

- ConvertCurrency();
- ConvertCurrency() { }
- void ConvertCurrency(){};

٢ لنفترض أن هناك أسلوب يسمى Update أي مما يلي يستدعي الأسلوب بشكل صحيح؟

- var update;
- Update();
- void Update();

راجع إجابتك

١ `void ConvertCurrency(){};`

هذا السطر صحيح يتم إنشاء توقيع أسلوب عن طريق الإعلان عن نوع الإرجاع، متبوعًا باسم الأسلوب، والأقواس التي تحتوي على أي معلمات إدخال.

٢ `Update();`

هذه الإجابة صحيحة! يتم استدعاء أسلوب باستخدام اسمه متبوعا بأقواس.

٥ إنشاء التعليمات البرمجية باستخدام الأساليب

تعد الأساليب مفيدة لتنظيم التعليمات البرمجية وإعادة استخدامها، ومعالجة المشكلات بكفاءة، يمكنك التفكير في أسلوب مثل مربع أسود يأخذ الإدخال، وينفذ المهمة المحددة، ويرجع الإخراج، مع هذا الافتراض، يمكنك إنشاء البرامج بسرعة عن طريق تسمية مهامك كأساليب، ثم قم بملء المنطق بعد تحديد جميع المهام الضرورية.

عندما تستخدم لغة بسيطة لوصف خطوات التعليمات البرمجية، دون الالتزام الصارم بقواعد بناء الجملة، فإنك تستخدم التعليمات البرمجية المستعارة/الزائفة "pseudo-code" تسمي أيضاً التعليمات المفترضة، يعد الجمع بين الأساليب والتعليمات المستعارة/الزائفة طريقة رائعة لإنجاز أي مهمة برمجية صعبة بسرعة.

هام

التعليمات الزائفة: هي ببساطة تطبيق لخوارزمية في شكل تعليقات توضيحية "تعليمات برمجية جاهزة يمكن الاستدلال بها" نص إعلامي مكتوب باللغة الإنجليزية البسيطة، ليس لديها بناء جملة مثل أي لغة برمجة، وبالتالي لا يمكن تجميعها أو تفسيرها بواسطة الكمبيوتر.

استخدام أساليب لهيكلة التعليمات البرمجية

لنفترض أنك مرشح في مقابلة برمجة، يريد منك القائم بالمقابلة أن تكتب برنامجًا يتحقق مما إذا كان عنوان IPv4 صالحًا أم غير صالح، يتم إعطاؤك القواعد التالية:

- يتكون عنوان IPv4 صالح من أربعة أرقام مفصولة بنقاط
- يجب ألا يحتوي كل رقم على أصفار بادئة
- يجب أن يتراوح كل رقم من 0 إلى 255

`1.1.1.1` and `255.255.255.255` هي أمثلة على عناوين IP صالحة.

يتم توفير عنوان IPv4 كسلسلة string يمكنك افتراض أنه يتكون فقط من أرقام ونقاط (لا توجد أحرف في السلسلة string المتوفرة)

كيف تتعامل مع هذه المهمة؟

ملاحظة: حتى إذا لم تكن على دراية بعناوين IP فلا تقلق! لا يزال بإمكانك إكمال التعليمات البرمجية في هذا التمرين باتباع الخطوات.

تقسيم المشكلة

في هذه المهمة، ستحدد الخطوات اللازمة لحل المشكلة، إذا تأملت القواعد، فقد تدرك أن الأمر يستغرق ثلاث خطوات فقط، لتحديد ما إذا كان عنوان IPv4 صالحا أم لا.

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية المستعارة/الزائفة التالية في المحرر:

```
/*
if ipAddress consists of 4 numbers
and
if each ipAddress number has no leading
zeroes
and
if each ipAddress number is in range 0 - 255

then ipAddress is valid

else ipAddress is invalid
*/
```

تعد التعليمات البرمجية المستعارة/الزائفة طريقة رائعة لبدء معالجة أي مشكلة، باستخدام كتلة التعليق هذه، يمكنك سد الفجوة بين قواعد المطالبة ورمز البرنامج، وتوضيح المهام الرئيسية التي ستنفذها تعليماتك البرمجية، ليس لزاماً أن تعمل التعليمات البرمجية الزائفة، أو تلتزم بقواعد بناء الجملة، ومع ذلك، يجب أن تكون شرحاً واضحاً لما ستفعله التعليمات البرمجية، الآن دعونا نحول هذا إلى تعليمات برمجية حقيقية!

٣. أدخل سطر فارغ جديد، ثم اكتب التعليمات التالية في المحرر:

```
if (ValidateLength() && ValidateZeroes() &&
ValidateRange())
{
    Console.WriteLine($"ip is a valid IPv4
address");
}
else
{
    Console.WriteLine($"ip is an invalid IPv4
address");
}
```

في هذه الخطوة، يمكنك تحويل العبارات if من التعليمات البرمجية الزائفة إلى أساليب قابلة للاستدعاء وإخراج النتائج، لا تقلق بشأن تحديد الأساليب حتى الآن؛ يمكنك افتراض أن كل أسلوب ينفذ المهمة التي يصفها اسمه، ستقوم بإصلاح أخطاء التحويل البرمجي وإنشاء منطق الأسلوب قريباً، ولكن التركيز على الصورة الكبيرة في الوقت الحالي، عندما تبدأ العمل على برنامج جديد، فإن التركيز على التصميم العام يساعدك على البقاء منظمًا، وتطوير تطبيقك بشكل أسرع.

٤. أدخل سطر جديد فارغ أسفل التعليمات البرمجية الموجودة، ثم اكتب التعليمات التالية في المحرر:

```
void ValidateLength() {}
```

```
void ValidateZeroes() {}
void ValidateRange() {}
```

لاحظ كيف أن استخدام أساليب العناصر النائبة placeholder methods سمح لك بالتعامل مع المشكلة بسرعة، وهيكلة تعليماتك البرمجية لتطوير الحل، الآن بعد أن أصبح لديك خطة منظمة، يمكنك الاستمرار في حل المشكلة عن طريق ملء التعليمات البرمجية قطعة قطعة.

تطوير الحل الخاص بك

بعد أن أصبح لديك جميع أساليب العنصر النائب placeholder methods اللازمة لحل المشكلة، يمكنك البدء في التركيز على تفاصيل الحل. ضع في اعتبارك أن تنسيق الإدخال لعنوان IPv4 سيكون نصي string تتكون من أرقام مفصولة بنقاط، دعونا نري!

١. في بداية البرنامج، قم بإنشاء متغيرات لتخزين حالات الإدخال والتحقق من الصحة:

```
string ipv4Input = "107.31.1.5";
bool validLength = false;
bool validZeroes = false;
bool validRange = false;
```

٢. حدّث التعليمات البرمجية للحل لاستخدام متغيرات التحقق من الصحة كما يلي:

```
ValidateLength();
ValidateZeroes();
ValidateRange();

if (validLength && validZeroes && validRange)
```

```
{
    Console.WriteLine($"ip is a valid IPv4
address");
}
else
{
    Console.WriteLine($"ip is an invalid IPv4
address");
}
```

٣. حدّث الأسلوب ValidateLength على النحو التالي:

```
void ValidateLength()
{
    string[] address = ipv4Input.Split(".");
    validLength = address.Length == 4;
};
```

تنص القاعدة الأولى على أن عنوان IPv4 يجب أن يحتوي على أربعة أرقام، لذلك في هذه التعليمات البرمجية، يمكنك استخدام string.Split لفصل الأرقام، والتحقق من وجود أربعة منها.

٤. حدّث الأسلوب ValidateZeroes على النحو التالي:

```
void ValidateZeroes()
{
    string[] address = ipv4Input.Split(".");

    foreach (string number in address)
    {
        if (number.Length > 1 &&
number.StartsWith("0"))
        {
            validZeroes = false;
        }
    }
```

```
    validZeroes = true;
}
```

خذ لحظة للنظر في كيفية ترجمة القاعدة إلى التعليمات البرمجية.

تنص القاعدة الثانية على أن الأرقام في عنوان IPv4 يجب ألا تبدأ بأصفار، لذلك يحتاج الأسلوب إلى التحقق من الأرقام بحثًا عن الأصفار البادئة، مع قبول 0 كرقم صالح، إذا كانت جميع الأرقام تحتوي على أصفار صالحة، يجب أن يكون validZeroes يساوي true وبخلاف ذلك false لذلك في هذه التعليمة البرمجية، يمكنك التحقق من أن كل رقم يحتوي على أكثر من رقم واحد، ولا يبدأ بصفر.

لاحظ أنه تم تعيين validZeroes على true بعد اكتمال حلقة foreach ومع ذلك، أنت لا تريد تعيين validZeroes على true إذا لم يتم العثور على أصفار بادئة، يمكنك تصحيح هذا الخطأ عن طريق تعيين validZeroes = true قبل تشغيل حلقة foreach ومع ذلك، يمكنك أيضًا تصحيح هذا الخطأ باستخدام عبارة return

٥. حدث التعليمات البرمجية الخاصة بك إلى ما يلي:

```
foreach (string number in address)
{
    if (number.Length > 1 &&
number.StartsWith("0"))
    {
        validZeroes = false;
        return;
    }
}
```

تنهي عبارة الإرجاع `return;` تنفيذ الأسلوب، وتعيد عنصر التحكم إلى مستدعي الأسلوب، تؤدي إضافة عبارة return بعد validZeroes = false إلى إنهاء الأسلوب بعد العثور على أول صفر غير صالح.

إذا لم يتم العثور على صفر غير صالح، فسيتم إنهاء الأسلوب بعد تعيين validZeroes إلى true دعونا ننتقل إلى الأسلوب التالي.

٦. حدّث الأسلوب ValidateRange على النحو التالي:

```
void ValidateRange()
{
    string[] address = ipv4Input.Split(".");

    foreach (string number in address)
    {
        int value = int.Parse(number);
        if (value < 0 || value > 255)
        {
            validRange = false;
            return;
        }
    }
    validRange = true;
}
```

تنص القاعدة الثالثة على أن كل رقم في عنوان IPv4 يجب أن يتراوح من 0 إلى 255 لذلك في هذه التعليمة البرمجية، يمكنك التحقق من أن كل رقم أقل من 255 وإذا لم يكن كذلك، فقم بإنهاء التنفيذ بعد تعيين validRange إلى false نظراً لأن سلسلة الإدخال تحتوي على أرقام ونقاط فقط، فلن تحتاج إلى التحقق من الأرقام السالبة.

ومع ذلك، قد تكون هناك حالة لا توجد فيها أرقام بين النقاط، على سبيل المثال "255...255" في هذه الحالة، سيرجع `string.Split(".")` إدخالات فارغة، مما يؤدي إلى فشل int.Parse يمكنك منع ذلك عن طريق تحديد `StringSplitOptions`

٧. حدث التعليمات البرمجية الخاصة بك كما يلي:

```
string[] address = ipv4Input.Split(".",
StringSplitOptions.RemoveEmptyEntries);
```

يؤدي استخدام `StringSplitOptions.RemoveEmptyEntries` إلى حذف الإدخالات الفارغة من المصفوفة address ومنع محاولات تحليل السلاسل الفارغة.

إكمال الحل الخاص بك

الآن بعد أن اكتملت جميع الأساليب للتحقق من صحة عنوان IP حان الوقت لإعادة النظر في حلك الأولي في هذه المهمة، ستضيف المزيد من قيم الإدخال، وتستعد لاختبار تعليماتك البرمجية.

١. حدد موقع التعليمات البرمجية التالية التي كتبتها سابقا في البرنامج:

```
string ipv4Input = "107.31.1.5";
```

٢. حدث التعليمات البرمجية كما يلي:

```
string[] ipv4Input = {"107.31.1.5", "255.0.0.255",
"555..0.555", "255...255"};
```

عند تطوير حل، من المهم اختبار التعليمات البرمجية مع حالات إدخال مختلفة، في هذه التعليمات البرمجية، يمكنك توفير مجموعة كافية من قيم الاختبار، الآن بعد أن قمت بتحديث إدخال الاختبار الخاص بك، ستحتاج إلى تحديث تعليماتك البرمجية لاستخدام القيم الجديدة، نظراً لأن القيم موجودة في مصفوفة، فستحتاج إلى تحديث التعليمات البرمجية لاختبار كل واحد باستخدام تكرار حلقي.

٣. حدث التعليمات البرمجية الخاصة بك كما يلي:

```
foreach (string ip in ipv4Input)
{
    ValidateLength();
    ValidateZeroes();
    ValidateRange();
```

```
    if (validLength && validZeroes &&
validRange)
    {
        Console.WriteLine($"{ip} is a valid
IPv4 address");
    }
    else
    {
        Console.WriteLine($"{ip} is an invalid
IPv4 address");
    }
}
```

أخيرًا، تحتاج إلى إصلاح بيانات الإدخال التي يستخدمها كل أسلوب، منذ أن قمت بتحديث ipv4Input من سلسلة إلى مصفوفة، نظرًا لأن كل أسلوب يستخدم `string.Split` يمكنك الإعلان عن متغير لتخزين نتيجة `string.Split` واستخدامها في كل أسلوب بدلاً من ذلك.

٤. أضف متغيراً لتخزين عنوان IPv4 الحالي الذي سيشير إليه كل أسلوب:

```
string[]     ipv4Input     =     {"107.31.1.5",
"255.0.0.255", "555..0.555", "255...255"};
string[] address;
bool validLength = false;
bool validZeroes = false;
bool validRange = false;
```

٥. تهيئة `address` باستخدام `string.Split` كما يلي:

```
foreach (string ip in ipv4Input)
{
    address = ip.Split(".",
StringSplitOptions.RemoveEmptyEntries);
```

٦. قم بإزالة المراجع references إلى `string.Split` من كل أساليب التحقق من الصحة، بحيث تستخدم متغير address عموماً بدلاً من ذلك، على سبيل المثال:

```
void ValidateLength()
{
    validLength = address.Length == 4;
};
```

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقك من الوحدة الطرفية المتكاملة، وتتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
string[] ipv4Input = {"107.31.1.5",
"255.0.0.255", "555..0.555", "255...255"};
string[] address;
bool validLength = false;
bool validZeroes = false;
bool validRange = false;

foreach (string ip in ipv4Input)
{
    address = ip.Split(".",
StringSplitOptions.RemoveEmptyEntries);

    ValidateLength();
    ValidateZeroes();
    ValidateRange();

    if (validLength && validZeroes &&
validRange)
    {
```

```
        Console.WriteLine($"{ip} is a valid
IPv4 address");
    }
    else
    {
        Console.WriteLine($"{ip} is an invalid
IPv4 address");
    }
}

void ValidateLength()
{
    validLength = address.Length == 4;
};

void ValidateZeroes()
{
    foreach (string number in address)
    {
        if (number.Length > 1 &&
number.StartsWith("0"))
        {
            validZeroes = false;
            return;
        }
    }

    validZeroes = true;
}

void ValidateRange()
{
    foreach (string number in address)
    {
```

```
            int value = int.Parse(number);
            if (value < 0 || value > 255)
            {
                validRange = false;
                return;
            }
        }
        validRange = true;
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٣. في موجه الأوامر Terminal أدخل dotnet run

٤. تحقق من أن تعليماتك البرمجية تنتج الإخراج التالي:

```
107.31.1.5 is a valid IPv4 address
255.0.0.255 is a valid IPv4 address
555..0.555 is an invalid IPv4 address
255...255 is an invalid IPv4 address
```

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

خلاصة

إليك ما تعلمته عن استخدام الأساليب حتى الآن:

- يمكن استخدام الأساليب لبناء التطبيقات بسرعة.
- يمكن استخدام الكلمة الأساسية return لإنهاء تنفيذ الأسلوب.
- يمكن غالباً ترجمة كل خطوة من خطوات المشكلة إلى أسلوب خاص.
- استخدام الأساليب لحل المشكلات الصغيرة لبناء الحل الخاص بك.

٦ تمرين ـ تحدي إنشاء أسلوب قابل لإعادة الاستخدام

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

ينصب تركيز هذا التحدي على تعديل التعليمات البرمجية، بحيث تكون قابلة لإعادة الاستخدام، ويمكن تنفيذها في أي وقت.

تحدث مع الثروة

أنت تساعد على تطوير لعبة ضخمة متعددة اللاعبين، كل لاعب لديه إحصائيات حظ، تؤثر على احتمالات العثور على كنز نادر، كل يوم، يمكن للاعب التحدث إلى صراف ثروة داخل اللعبة، يكشف ما إذا كان إحصائيات حظه عالية أو منخفضة أو محايدة.

تحتوي اللعبة حاليًا على تعليمات لتوليد ثروة اللاعب، لكنها غير قابلة لإعادة الاستخدام، مهمتك هي إنشاء أسلوب `tellFortune` يمكن استدعاؤها في أي وقت، واستبدال المنطق الموجود باستدعاء أسلوبك.

في هذا التحدي، يتم منحك بعض التعليمات البرمجية للبدء، يجب أن تقرر كيفية إنشاء الأسلوب `tellFortune` واستدعاءه.

تحدي التعليمات البرمجية: إنشاء أسلوب قابل لإعادة الاستخدام

داخل التعليمات البرمجية التي تبدأ بها، هناك مصفوفة نص عامة generic text array تليها مصفوفات نص جيدة وسيئة ومحايدة، اعتمادًا على قيمة luck يتم تحديد أحد المصفوفات وعرضها جنباً إلى جنب مع النص العام.

يتمثل التحدي الذي تواجهه في إنشاء أسلوب قابل لإعادة الاستخدام reusable method يطبع ثروة اللاعب في أي وقت، يجب أن يحتوي الأسلوب على المنطق الموجود بالفعل في التعليمات البرمجية المقدم.

١. تأكد من أن لديك ملف Program.cs فارغا مفتوحا 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 لوحة Visual Studio Code EXPLORER حدد Program.cs
- في قائمة Visual Studio Code Selection حدد Select All ثم اضغط على مفتاح Delete

٢. انسخ التعليمات البرمجية التالية وألصقها في المحرر:

```
Random random = new Random();
int luck = random.Next(100);

string[] text = {"You have much to", "Today is a day to", "Whatever work you do", "This is an ideal time to"};
string[] good = {"look forward to.", "try new things!", "is likely to succeed.", "accomplish your dreams!"};
string[] bad = {"fear.", "avoid major decisions.", "may have unexpected outcomes.", "re-evaluate your life."};
string[] neutral = {"appreciate.", "enjoy time with friends.", "should align with your values.", "get in tune with nature."};

Console.WriteLine("A fortune teller whispers the following words:");
```

```
string[] fortune = (luck > 75 ? good : (luck < 25 ? bad : neutral));
for (int i = 0; i < 4; i++)
{
    Console.Write($"{text[i]} {fortune[i]} ");
}
```

١. عدل التعليمات البرمجية لاستخدام أسلوب لعرض الثروة.

استخدم ما تعلمته حول إنشاء الأساليب واستدعاءها لإكمال التعديل.

٢. اختبر تعليماتك البرمجية عن طريق تغيير قيمة الأسلوب luck واستدعاءه مرة أخرى.

٣. تحقق من أن التعليمات البرمجية تنتج إحدى الرسائل التالية:

```
A fortune teller whispers the following words:
You have much to look forward to. Today is a day to try new things! Whatever work you do is likely to succeed. This is an ideal time to accomplish your dreams!
```

```
A fortune teller whispers the following words:
You have much to appreciate. Today is a day to enjoy time with friends. Whatever work you do should align with your values. This is an ideal time to get in tune with nature.
```

```
A fortune teller whispers the following words:
You have much to fear. Today is a day to avoid major decisions. Whatever work you do may have unexpected outcomes. This is an ideal time to re-evaluate your life.
```

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

٧ مراجعة حل إنشاء أسلوب قابل لإعادة الاستخدام

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

```
Random random = new Random();
int luck = random.Next(100);

string[] text = {"You have much to", "Today is
a day to", "Whatever work you do", "This is an
ideal time to"};
string[] good = {"look forward to.", "try new
things!", "is likely to succeed.", "accomplish
your dreams!"};
string[] bad = {"fear.", "avoid major
decisions.", "may have unexpected outcomes.",
"re-evaluate your life."};
string[] neutral = {"appreciate.", "enjoy time
with friends.", "should align with your
values.", "get in tune with nature."};

TellFortune();
luck = random.Next(100);
TellFortune();

void TellFortune()
{
    Console.WriteLine("A fortune teller
whispers the following words:");
    string[] fortune = (luck > 75 ? good :
(luck < 25 ? bad : neutral));
    for (int i = 0; i < 4; i++)
    {
        Console.Write($"{text[i]} {fortune[i]}
");
    }
}
```

تعد هذه التعليمات البرمجية مجرد حل واحد ممكن، بسبب ربما تكون قد أضفت موجزات الأسطر في أماكن مختلفة أو ربما قمت بتنسيق التعليمات بشكل مختلف.

بغض النظر عن الاختلافات الطفيفة في التعليمات البرمجية، عند تشغيل التعليمات، يجب أن تشاهد اثنتين من رسائل الإخراج التالية:

```Output A
A fortune teller whispers the following words:
You have much to look forward to. Today is a day to try new things! Whatever work you do is likely to succeed. This is an ideal time to accomplish your dreams!
```

```Output B
A fortune teller whispers the following words:
You have much to appreciate. Today is a day to enjoy time with friends. Whatever work you do should align with your values. This is an ideal time to get in tune with nature.
```

```Output C
A fortune teller whispers the following words:
You have much to fear. Today is a day to avoid major decisions. Whatever work you do may have unexpected outcomes. This is an ideal time to re-evaluate your life.
```

إذا أكملت التحدي، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا واجهت مشكلة في إكمال هذا التحدي، راجع الوحدات السابقة قبل المتابعة، ستعتمد جميع الأفكار الجديدة التي سوف نناقشها في وحدات أخرى على فهمك للأفكار التي تم تقديمها في هذه الوحدة.

٨ اختبر معلوماتك

١- أي مما يلي يعلن بشكل صحيح عن أسلوب؟

- `DisplayNumbers();`
- `void DisplayNumbers{};`
- `void DisplayNumbers() { }`

٢- أي مما يلي يستدعي أسلوباً بشكل صحيح؟

- `GenerateID();`
- `void GenerateID() {}`
- `GenerateID;`

٣- أي مما يلي صحيح حول return الكلمة الأساسية؟

- إنها مطابقة للكلمة break الأساسية
- تنهي تنفيذ الأسلوب
- يجب استخدامها دائما مع قيمة أو تعبير

راجع إجابتك

١ void DisplayNumbers() { }

صحيح يتم تعريف أسلوب باستخدام نوع الإرجاع، متبوعاً بأقواس () تحتوي على أي معلمات، وأقواس {} لاحتواء نص الأسلوب.

٢ GenerateID();

صحيح يتم استدعاء الأساليب باستخدام الاسم، متبوعة بأقواس () وفواصل منقوطة.

٣ تنهي تنفيذ الأسلوب

صحيح تنهي عبارة الإرجاع return تنفيذ أسلوبها وتعيد عنصر التحكم إلى المتصل.

٩ الملخص

كان هدفك هو تعلم كيفية إنشاء أساليبك الخاصة، لأداء مهام محددة. لقد تعلمت نقل التعليمات البرمجية القابلة لإعادة الاستخدام إلى أساليب لتقليل التعليمات البرمجية المكررة، تعلمت أيضاً استخدام أساليب التعليمات البرمجية المستعارة/الزائفة والعناصر النائبة لمعالجة مشكلة معقدة، من خلال إنشاء أساليب صغيرة ذات مسؤوليات محددة، تعلمت إنشاء حلول بسرعة للمشاكل باستخدام تعليمات برمجية فعالة وسهلة القراءة.

الوحدة الثانية

إنشاء أساليب C# باستخدام المعلمات parameters

تعرف على كيفية استخدام أنواع مختلفة من معلمات الإدخال input parameters في الأساليب.

الأهداف التعليمية

- تعرف على المزيد حول استخدام المعلمات parameters
- فهم نطاق الأسلوب method scope
- فهم أنواع معلمات التمرير حسب المرجع والتمرير حسب القيمة pass-by-reference and pass-by-value parameter types
- تعرف على كيفية استخدام الوسيطات الاختيارية والمسماة optional and named arguments

محتويات الوحدة: -

١ المقدمة

الأساليب لديها القدرة على تنفيذ عمليات على الإدخال، يتيح لك تمرير المعلمات إلى أساليبك تنفيذ مهمة الأسلوب بقيم إدخال مختلفة، يتيح لك استخدام معلمات الأسلوب method parameters توسيع تعليماتك البرمجية مع الحفاظ على تنظيم البرنامج وسهولة قراءته، إذا كنت تعتبر أن الأسلوب بمثابة صندوق أسود يقبل الإدخال، ويؤدي مهمة واحدة، فيمكنك تقسيم مشكلة كبيرة إلى أجزاء قابلة للتنفيذ بسرعة.

لنفترض أنك بحاجة إلى كتابة تعليمات برمجية تنفذ نفس العملية على مجموعات مختلفة من المدخلات، قد يكون لديك ثلاث مصفوفات مختلفة، وتحتاج إلى عرض محتويات كل منها، يمكنك إنشاء أسلوب `DisplayArray` الذي يقبل مصفوفة واحدة كإدخال، ويعرض محتوياتها، بدلاً من كتابة تعليمات برمجية لعرض كل مصفوفة على حدة، يمكنك استدعاء نفس الأسلوب، وتوفير المصفوفات المختلفة كمدخلات.

يمكن أن تجعل المعلمات Parameters أساليبك أكثر قوة، مع الاستمرار في تنفيذ نفس المهمة العامة. في هذه الوحدة، ستتعلم المزيد حول العمل مع المعلمات وترسيخ فهمك للأساليب.

٢ استخدام المعلمات في الأساليب

عند إنشاء الأساليب، ستحتاج غالبًا إلى توفير بعض المعلومات للأسلوب لاستخدامه، تسمى المعلومات التي يستهلكها أسلوب معلمة parameter يمكنك توفير العديد من المعلمات حسب الحاجة لإنجاز مهمتها، أو لا شيء على الإطلاق.

غالبا ما يتم استخدام مصطلحي معلمة 'parameter' ووسيطة 'argument' بالتبادل، ومع ذلك، تشير **المعلمة parameter** إلى المتغير في توقيع الأسلوب أما **الوسيطة argument** هي القيمة التي يتم تمريرها عند استدعاء الأسلوب.

إضافة معلمات إلى الأساليب parameters to methods

تعمل المعلمات في أسلوب، مشابهة للمتغيرات، يتم تعريف المعلمة عن طريق تحديد نوع البيانات، متبوعًا باسم المعلمة، يتم الإعلان عن المعلمات في توقيع الأسلوب، ويتم توفير قيم المعلمات بواسطة مستدعي الأسلوب بدلاً من تهيئتها داخل الأسلوب نفسه، راجع التعليمة البرمجية التالية:

```
CountTo(5);
```

```
    void CountTo(int max)
    {
        for (int i = 0; i < max; i++)
        {
            Console.Write($"${i}, ");
        }
    }
```

في هذا المثال، يقبل الأسلوب CountTo معلمة عددية integer تسمى max تتم الإشارة إلى المعلمة في حلقة for الخاصة parameter

بالأسلوب، عند استدعاء CountTo يتم توفير العدد الصحيح 5 كوسيطة argument

في هذا التمرين، ستتعلم كيفية إنشاء معلمات لأسلوبك، واستخدامها.

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

إنشاء أسلوب باستخدام المعلمات method with parameters

في هذه المهمة، ستقوم بإنشاء أسلوب يضبط الأوقات المجدولة إلى منطقة زمنية مختلفة بتوقيت جرينتش GMT يجب أن يقبل الأسلوب قائمة بالوقت، والمنطقة الزمنية الحالية، والمنطقة الزمنية الجديدة. لنبدأ!

١. أدخل التعليمات البرمجية التالية في محرر التعليمات البرمجية Visual Studio

```
int[] schedule = {800, 1200, 1600, 2000};
```

٢. لإنشاء أسلوب مع معلمات، أدخل التعليمات البرمجية التالية في سطر فارغ جديد:

```
void DisplayAdjustedTimes(int[] times, int
currentGMT, int newGMT)
{

}
```

لاحظ أن المعلمات parameters معلنة بطريقة مشابهة للتي تعلن بها عن المتغيرات، باستخدام نوع البيانات متبوعًا باسم المتغير، يمكنك استخدام معلمات من أي نوع بيانات، مثل string, bool, int, arrays والمزيد! دائما ما يتم فصل معلمات متعددة في أسلوب بفاصلة , comma

٣. أدخل التعليمات التالية في الأسلوب DisplayAdjustedTimes

```
int diff = 0;
if (Math.Abs(newGMT) > 12 || Math.Abs(currentGMT) > 12)
{
    Console.WriteLine("Invalid GMT");
}
```

لاحظ كيف أنه ليس عليك، الإعلان عن المتغيرين `newGMT` and `currentGMT` لأنه تم الإعلان عنهما بالفعل في توقيع الأسلوب، كما أنك لا تقوم بتهيئة المتغيرات، لأن الأسلوب يفترض أن المستدعي يوفر هذه الوسيطات arguments بقيم مخصصة.

في هذه الخطوة، يمكنك إنشاء `int diff` لتخزين فارق التوقيت ثم التحقق لمعرفة قيم توقيت جرينتش GMT المتوفرة تتراوح بين 12 and -12 ويمنحك استخدام Math.Abs القيمة المطلقة لرقم ما، لذا تكون قيم GMT غير صالحة إذا كانت أكبر من 12

٤. لحساب الفرق الزمني، قم بتعديل الأسلوب `DisplayAdjustedTimes` كما يلي:

```
int diff = 0;
if (Math.Abs(newGMT) > 12 || Math.Abs(currentGMT)
> 12)
{
    Console.WriteLine("Invalid GMT");
}
else if (newGMT <= 0 && currentGMT <= 0 || newGMT
>= 0 && currentGMT >= 0)
{
```

```
    diff = 100 * (Math.Abs(newGMT) -
Math.Abs(currentGMT));
}
else
{
    diff = 100 * (Math.Abs(newGMT) +
Math.Abs(currentGMT));
}
```

في هذه التعليمة البرمجية، يمكنك التحقق لمعرفة ما إذا كنت بحاجة إلى إضافة أو طرح القيم المطلقة للمناطق الزمنية بتوقيت جرينتش GMT للحصول على الفرق بالساعات، إذا كانت قيم GMT تشترك في نفس العلامة (كلاهما موجب أو كلاهما سالب) فإن فرق الساعات يساوي الفرق بين الرقمين، إذا كانت قيم GMT لها علامات عكسية، فإن الفرق يساوي مجموع الرقمين، نظراً لأن الساعات ممثلة بالمئات، فإنك تضرب النتيجة في 100

٥. لعرض النتائج، أدخل التعليمات البرمجية التالية في نهاية الأسلوب `DisplayAdjustedTimes`

```
for (int i = 0; i < times.Length; i++)
{
    int newTime = ((times[i] + diff)) % 2400;
    Console.WriteLine($"{times[i]} ->
{newTime}");
}
```

٦. لاستدعاء الأسلوب الخاص بك، أدخل التعليمات البرمجية التالية بعد إعلان متغير الجدول الزمني `int[] schedule`

```
DisplayAdjustedTimes(schedule, 6, -6);
```

لاحظ أنه يمكن توفير كل من المتغيرات والقيم الحرفية literal arguments كوسيطات arguments للأسلوب، باستخدام معلمات الإدخال input parameters لا يقتصر الأسلوب على استخدام قيم المتغيرات العامة.

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق الخاص بك من الوحدة الطرفية المتكاملة والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٢. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٣. في موجه الأوامر Terminal أدخل **dotnet run**

٤. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
800 -> 2000
1200 -> 0
1600 -> 400
2000 -> 800
```

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

خلاصة

إليك ما تعلمته عن المعلمات parameters حتى الآن:

- يمكن تمرير المعلومات إلى الأساليب في شكل معلمات.
- يتم الإعلان عن المعلمات في توقيع الأسلوب.
- يتم فصل المعلمات المتعددة بفواصل.
- يمكن أن تقبل الأساليب الوسيطات المتغيرة أو الحرفية variable or literal arguments

٣ فهم نطاق الأسلوب method scope

تمثل حلقات for وعبارات if-else والأساليب جميعها أنواعًا مختلفة من كتل التعليمات البرمجية، كل كتلة برمجية لها نطاقها الخاص Scope النطاق هو منطقة البرنامج التي يمكنه الوصول إلى بيانات معينة فيها، لا يمكن الوصول إلى المتغيرات المعلنة داخل إحدى الطرق، أو أي كتلة برمجية، إلا داخل تلك المنطقة، عندما تصبح البرامج أكثر تعقيدًا، يساعد هذا النمط المبرمجين على استخدام المتغيرات ذات الأسماء الواضحة باستمرار، والحفاظ على سهولة قراءة التعليمات البرمجية.

في هذا التمرين، ستتعرف على المزيد حول نطاق الأسلوب من خلال العمل مع أنواع مختلفة من الأساليب والمتغيرات.

اختبار نطاق المتغير

تسمى العبارات المعلنة خارج أي كتلة برمجية عبارات المستوى الأعلى top-level statements تسمى المتغيرات المعلنة في عبارات المستوى الأعلى المتغيرات العامة/الشاملة global variables لا تقتصر المتغيرات العامة على أي نطاق، ويمكن استخدامها في أي مكان في جميع أنحاء البرنامج، يمكن أن تكون المتغيرات العامة مفيدة للأساليب المختلفة التي تحتاج إلى الوصول إلى نفس البيانات، ومع ذلك، من المهم الانتباه إلى أسماء المتغيرات في نطاقات مختلفة.

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
string[] students = {"Jenna", "Ayesha",
"Carlos", "Viktor"};

DisplayStudents(students);
DisplayStudents(new string[]
{"Robert","Vanya"});
```

```
void DisplayStudents(string[] students)
{
    foreach (string student in students)
    {
        Console.Write($"{student}, ");
    }
    Console.WriteLine();
}
```

٣. في هذه التعليمة البرمجية، يمكنك إنشاء مصفوفة عامة `students` وأسلوب `DisplayStudents` يقبل معلمة parameter بنفس الاسم.

٤. احفظ التعليمات البرمجية وشغلها لمراقبة الإخراج التالي:

```
Jenna, Ayesha, Carlos, Viktor,
Robert, Vanya,
```

لاحظ أن معلمة الأسلوب student لها الأسبقية على المصفوفة العامة student من المهم أن تكون متأن بشأن المتغيرات العامة global variables التي تريد أن تستخدمها أساليبك.

٤. حذف التعليمات البرمجية السابقة.

٥. أدخل التعليمات البرمجية التالية في المحرر:

```
PrintCircleArea(12);

void PrintCircleArea(int radius)
{
    double pi = 3.14159;
    double area = pi * (radius * radius);
    Console.WriteLine($"Area = {area}");
}
```

تحسب هذه التعليمة البرمجية منطقة الدائرة وتعرضها.

٦. حاول الإشارة إلى المتغيرات الموجودة داخل الأسلوب `PrintCircleArea` عن طريق تحديث التعليمات البرمجية على النحو التالي:

```
PrintCircleArea(12);
double circumference = 2 * pi * radius;
```

تظهر رسائل خطأ تخبرك بأن الاسمين `pi` and `radius` غير موجودين في النطاق الحالي، هذه المتغيرات موجودة فقط ضمن نطاق أسلوب `PrintCircleArea`

٧. احذف التعليمات غير الصحيحة، وأضف التعليمات التالية:

```
void PrintCircleCircumference(int radius)
{
    double pi = 3.14159;
    double circumference = 2 * pi * radius;
    Console.WriteLine($"Circumference =
{circumference}");
}
```

نظرًا لتعيين المتغير pi على نفس القيمة الثابتة، واستخدامه في كلا الأسلوبين، فإن هذه القيمة تعد مرشحًا جيدًا لمتغير عام. في هذا المثال، `radius` ليس متغيرًا عامًا، لذا يمكنك استدعاء الأساليب بقيم مختلفة ل `radius` دون تحديث متغير في كل مرة.

٨. حدث التعليمات البرمجية إلى ما يلي:

```
double pi = 3.14159;

void PrintCircleArea(int radius)
{
    double area = pi * (radius * radius);
    Console.WriteLine($"Area = {area}");
}
```

```
void PrintCircleCircumference(int radius)
{
    double circumference = 2 * pi * radius;
    Console.WriteLine($"Circumference =
{circumference}");
}
```

الآن يمكن لكلا الأسلوبين الرجوع إلى نفس قيمة pi دون الحاجة إلى تعريفها، ربما تكون قد خمنت بالفعل أن الأساليب يمكن أن تستدعي أساليب أخرى، بشكل عام، طالما تم تعريف أسلوب ضمن نطاق برنامجك، فيمكن استدعاؤه في أي مكان.

٩. أضف أسلوبًا جديدًا إلى تعليماتك البرمجية كما يلي:

```
double pi = 3.14159;
PrintCircleInfo(12);
PrintCircleInfo(24);

void PrintCircleInfo(int radius)
{
    Console.WriteLine($"Circle with radius
{radius}");
    PrintCircleArea(radius);
    PrintCircleCircumference(radius);
}
```

في هذه التعليمات البرمجية، يمكنك إنشاء أسلوب PrintCircleInfo جديد لاستدعاء الأساليب الموجودة، يتم تمرير قيمة radius أيضاً إلى كل أسلوب، يمكن أن يساعد إنشاء أساليب نمطية في الحفاظ على تنظيم التعليمات البرمجية، وسهولة قراءتها.

١٠. تأكد أن تعليماتك تشبه التعليمات التالية:

```
double pi = 3.14159;
PrintCircleInfo(12);
PrintCircleInfo(24);

void PrintCircleArea(int radius)
{
    double area = pi * (radius * radius);
    Console.WriteLine($"Area = {area}");
}

void PrintCircleCircumference(int radius)
{
    double circumference = 2 * pi * radius;
    Console.WriteLine($"Circumference =
{circumference}");
}

void PrintCircleInfo(int radius)
{
    Console.WriteLine($"Circle with radius
{radius}");
    PrintCircleArea(radius);
    PrintCircleCircumference(radius);
}
```

١١. احفظ التعليمات البرمجية وشغلها لمراقبة الإخراج التالي:

```
Circle with radius 12
Area = 452.38896
Circumference = 75.39815999999999
Circle with radius 24
Area = 1809.55584
Circumference = 150.79631999999998
```

خلاصة

إليك ما تعلمته حول نطاق الأسلوب حتى الآن:

- المتغيرات المعلنة داخل أسلوب لا يمكن الوصول إليها إلا من خلال هذا الأسلوب.
- يمكن الوصول إلى المتغيرات المعلنة في عبارات المستوى الأعلى top-level statements في جميع أنحاء البرنامج.
- لا تمتلك الأساليب إمكانية الوصول إلى المتغيرات المحددة ضمن أساليب مختلفة.
- يمكن للأساليب استدعاء أساليب أخرى.

٤ استخدام معلمات القيمة ونوع المرجع value and reference type parameters

في C# يمكن تصنيف المتغيرات إلى نوعين رئيسيين، أنواع القيم value types وأنواع المراجع reference types تصف هذه الأنواع كيفية تخزين المتغيرات لقيمها.

تحتوي أنواع القيم مثل int, bool, float, double, and char على قيم مباشرة، لا تخزن الأنواع المرجعية مثل string, array, and objects (مثل مثيلات Random) قيمها مباشرة، بدلاً من ذلك، تخزن الأنواع المرجعية عنواناً حيث يتم تخزين قيمتها.

المعلمات التي تم تمريرها حسب القيمة وتم تمريرها حسب المرجع

عند تمرير وسيطة argument إلى أسلوب، يتم نسخ قيم متغيرات نوع القيمة إلى الأسلوب، كل متغير له نسخته الخاصة من القيمة، لذلك لا يتم تعديل المتغير الأصلي.

مع أنواع المراجع، يتم تمرير عنوان القيمة إلى الأسلوب، يشير المتغير المعطى للأسلوب إلى القيمة الموجودة في هذا العنوان، لذلك تؤثر العمليات على هذا المتغير، على القيمة المشار إليها من قبل المتغير الآخر.

ملاحظة

من المهم أن نتذكر أن string نوع مرجعي reference type ولكنها غير قابل للتغيير، وهذا يعني أنه بمجرد تعيين قيمة لها، لا يمكن تغييرها، في C# عند استخدام الأساليب وعوامل التشغيل لتعديل سلسلة string تكون النتيجة التي يتم إرجاعها هي في الواقع كائن سلسلة جديد new string object

في هذا التمرين، ستتعلم المزيد حول تمرير وسيطات arguments نوع المرجع والقيمة إلى الأساليب.

اختبار النجاح حسب القيمة

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
int a = 3;
int b = 4;
int c = 0;

Multiply(a, b, c);
Console.WriteLine($"global statement: {a} x {b} = {c}");

void Multiply(int a, int b, int c)
{
    c = a * b;
    Console.WriteLine($"inside Multiply method: {a} x {b} = {c}");
}
```

يتم تمرير المتغيرات `a`, `b`, and `c` إلى الأسلوب `Multiply` تتم طباعة قيم المتغيرات أثناء تنفيذ الأسلوب، وتتم طباعتها مرة أخرى بعد اكتمال الأسلوب.

الأعداد الصحيحة هي أنواع قيم، يتم نسخ قيمها عند تمريرها إلى الأساليب. ما الذي تعتقد أن ناتج `c` سيكون؟

٣. احفظ التعليمات البرمجية وقم بتشغيلها لمراقبة الإخراج التالي:

```
inside Multiply method: 3 x 4 = 12
global statement: 3 x 4 = 0
```

لاحظ أن قيمة c يتم تغييرها فقط في أسلوب `Multiply` خارج الأسلوب يحتفظ c بقيمته الأصلية.

اختبار التمرير حسب المرجع

١. حذف التعليمات البرمجية السابقة من محرر Visual Studio

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
int[] array = {1, 2, 3, 4, 5};

PrintArray(array);
Clear(array);
PrintArray(array);

void PrintArray(int[] array)
{
    foreach (int a in array)
    {
        Console.Write($"{a} ");
    }
    Console.WriteLine();
}

void Clear(int[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        array[i] = 0;
    }
}
```

تبدأ التعليمات البرمجية بتهيئة array لتحتوي على بعض قيم الأعداد الصحيحة، يتم عرض القيم باستخدام الأسلوب PrintArray يتم استدعاء الأسلوب Clear على المصفوفة، ثم تتم طباعة المصفوفة مرة أخرى.

المصفوفات هي أنواع مرجعية reference types تخزن أنواع المراجع عنوان قيمها في الذاكرة، برأيك ماذا سيكون الناتج؟

٣. احفظ التعليمات البرمجية وقم بتشغيلها لمراقبة الإخراج التالي:

```
1 2 3 4 5
0 0 0 0 0
```

لاحظ أن المصفوفة تظل معدلة/متغيرة خارج نطاق أسلوب Clear يحدث هذا لأن الأسلوب Clear قام بتحديث القيم المخزنة في كل عنوان.

اختبار باستخدام السلاسل Test with strings

تعلمت سابقًا أن السلاسل النصية هي نوع غير قابل للتغيير immutable type على الرغم من أن السلسلة هي نوع مرجع، عكس المصفوفة، لا يمكن تغيير قيمتها بمجرد تعيينها، ربما لاحظت ذلك إذا كنت قد استخدمت أساليب مثل ل في هذه المهمة، ستتعلم تصحيح خطأ شائع عند العمل مع السلاسل strings

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
string status = "Healthy";

Console.WriteLine($"Start: {status}");
SetHealth(status, false);
Console.WriteLine($"End: {status}");

void SetHealth(string status, bool isHealthy)
{
    status = (isHealthy ? "Healthy" :
"Unhealthy");
    Console.WriteLine($"Middle: {status}");
}
```

٣. احفظ التعليمات البرمجية وقم بتشغيلها لمراقبة الإخراج التالي:

```
Start: Healthy
Middle: Unhealthy
End: Healthy
```

إذا لم يقم الأسلوب `SetHealth` بإخراج الحالة، فربما افترضت أن الأسلوب لم ينفذ بشكل صحيح، بدلاً من ذلك، تم إنشاء سلسلة جديدة بالقيمة "Unhealthy" ثم فقدانها في نطاق الأسلوب.

لتصحيح هذه المشكلة، يمكنك تغيير `SetHealth` لاستخدام متغير الحالة العامة global status variable بدلاً من ذلك.

٤. حدث تعليماتك كما يلي:

```
string status = "Healthy";

Console.WriteLine($"Start: {status}");
SetHealth(false);
Console.WriteLine($"End: {status}");

void SetHealth(bool isHealthy)
{
    status = (isHealthy ? "Healthy" :
"Unhealthy");
    Console.WriteLine($"Middle: {status}");
}
```

في هذه التعليمة البرمجية، تقوم بالكتابة فوق المتغير العام `status` بقيمة السلسلة الجديدة.

٥. احفظ التعليمات البرمجية وقم بتشغيلها لمراقبة الإخراج التالي:

```
Start: Healthy
Middle: Unhealthy
End: Unhealthy
```

الآن يتم التقاط السلسلة المحدثة وتخزينها بشكل صحيح.

خلاصة

إليك ما تعلمته عن معلمات نوع القيمة ونوع المرجع حتى الآن:

- يمكن تصنيف المتغيرات على أنها أنواع قيم وأنواع مرجعية.
- تحتوي أنواع القيم على القيم مباشرة، وتخزن أنواع المراجع عنوان القيمة.
- الأساليب التي تستخدم وسيطات arguments نوع القيمة تنشئ نسختها الخاصة من القيم.
- تؤثر الأساليب التي تقوم بإجراء تغييرات على معلمة مصفوفة على مصفوفة الإدخال الأصلية.
- السلسلة string هي نوع مرجع غير قابل للتغيير.
- لا تؤثر الأساليب التي تقوم بإجراء تغييرات على معلمة parameter سلسلة على السلسلة الأصلية.

٥ أساليب ذات معلمات اختيارية Methods with optional parameters

تسمح لغة C Sharp باستخدام المعلمات المسماة والاختيارية named and optional parameters تتيح لك هذه الأنواع من المعلمات تحديد الوسيطات/الوسائط التي تريد توفيرها للأسلوب، بحيث لا تكون مقيدًا بالبنية المحددة في توقيع الأسلوب.

تسمح لك الوسيطات المُسماة named arguments بتحديد قيمة معلمة باستخدام اسمها بدلاً من موضعها، تسمح لك المعلمات الاختيارية optional parameters بحذف هذه الوسيطات عند استدعاء الأسلوب.

في هذا التمرين، ستتعلم كيفية استخدام كل من المعلمات المسماة والاختيارية

إنشاء تطبيق الرد على الدعوة RSVP

في هذه المهمة، ستقوم بإنشاء تطبيق مختصر للضيوف للرد على الدعوة RSVP لحضور حدث ما، سيقدم الضيوف حجم حفلتهم، وإذا كان لديهم حساسية، ستضيف أيضًا خيار تقييد الرد على الدعوات، لتقتصر على قائمة المدعوين فقط.

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. اكتب التعليمات البرمجية التالية في المحرر:

```
string[] guestList = {"Rebecca", "Nadia",
"Noor", "Jonte"};
string[] rsvps = new string[10];
int count = 0;

void RSVP(string name, int partySize, string
allergies, bool inviteOnly)
{
    if (inviteOnly)
```

```
    {
        // search guestList before adding rsvp
    }

    rsvps[count] = $"Name: {name}, \tParty
Size: {partySize}, \tAllergies: {allergies}";
    count++;
}

void ShowRSVPs()
{
    Console.WriteLine("\nTotal RSVPs:");
    for (int i = 0; i < count; i++)
    {
        Console.WriteLine(rsvps[i]);
    }
}
```

في هذه التعليمات البرمجية، يمكنك إنشاء متغيرات لتخزين قائمة الضيوف و rsvps يقوم الأسلوب `RSVP` بإلحاق معلومات الضيف بالقائمة، ويعرض الأسلوب `ShowRSVPs` إجمالي الردود على الدعوات RSVPs باستخدام تسلسلات الهروب/الإلغاء، علامة التبويب tab escape لفصل معلومات الضيف.

٣. أدخل التعليمات البرمجية التالية في الأسلوب `RSVP` للبحث في قائمة الضيوف، فوق التعليمات الموجودة:

```
if (inviteOnly)
{
    bool found = false;
    foreach (string guest in guestList)
    {
        if (guest.Equals(name)) {
```

```
                found = true;
                break;
            }
        }
        if (!found)
        {
            Console.WriteLine($"Sorry, {name} is
not on the guest list");
            return;
        }
}
```

في هذه التعليمة البرمجية، يمكنك التحقق لمعرفة ما إذا كان الاسم المحدد يساوي أيًا من الأسماء الموجودة في قائمة الضيوف، إذا تم العثور على تطابق، يمكنك تعيين `found` إلى true والخروج من حلقة foreach إذا كان `found` يساوي false يمكنك عرض رسالة، واستخدام الكلمة الأساسية return لإنهاء الأسلوب.

٤. استدعاء الأسلوب الخاص بك عن طريق إضافة التعليمات البرمجية التالية فوق توقيع الأسلوب `RSVP`

```
RSVP("Rebecca", 1, "none", true);
RSVP("Nadia", 2, "Nuts", true);
RSVP("Linh", 2, "none", false);
RSVP("Tony", 1, "Jackfruit", true);
RSVP("Noor", 4, "none", false);
RSVP("Jonte", 2, "Stone fruit", false);
ShowRSVPs();
```

٥. احفظ التعليمات البرمجية وشغلها لمراقبة الإخراج التالي:

```
Sorry, Tony is not on the guest list
```

```
Total RSVPs:
Name: Rebecca,  Party Size: 1,  Allergies: none
Name: Nadia,    Party Size: 2,  Allergies: Nuts
Name: Linh,     Party Size: 2,  Allergies: none
Name: Noor,     Party Size: 4,  Allergies: none
Name: Jonte,    Party Size: 2,  Allergies: Stone fruit
```

استخدام الوسيطات المسماة named arguments

عند استدعاء أسلوب يقبل العديد من المعلمات many parameters قد يكون من الصعب فهم ما تمثله الوسيطات arguments يمكن أن يؤدي استخدام الوسيطات المسماة named argument إلى تحسين إمكانية قراءة التعليمات البرمجية، استخدم وسيطة مسماة عن طريق تحديد اسم المعلمة متبوعاً بقيمة الوسيطة، في هذه المهمة، ستتدرب على استخدام الوسيطات المسماة.

١. حدد موقع السطر التالي من التعليمات البرمجية:

```
RSVP("Linh", 2, "none", false);
```

٢. حدث استدعاء الأسلوب كما يلي:

```
RSVP(name: "Linh", partySize: 2, allergies: "none", inviteOnly: false);
```

لاحظ أنك توفر اسم المعلمة متبوعاً بنقطتين والقيمة يُعرف بناء الجملة هذا وسيطة مسماة named argument ليس من الضروري تسمية جميع الوسيطات، على سبيل المثال، بناء الجملة syntax التالي صالح أيضاً:

```
RSVP("Linh", 2, allergies: "none", inviteOnly: false);
RSVP("Linh", partySize: 2, "none", false);
```

الوسيطات المُسماة، عند استخدامها مع الوسيطات الموضعية positional arguments تكون صالحة إذا تم استخدامها في الموضع الصحيح. الوسيطات المُسماة صالحة أيضاً طالما لا يتبعها أي وسيطات موضعية، على سبيل المثال، تضمين "Linh" and 2 في النهاية سيكون غير صالح:

```
RSVP(allergies: "none", inviteOnly: false, "Linh", 2);
```

إذا أدخلت هذه التعليمة البرمجية، فستحصل على الخطأ التالي:

```
Named argument 'allergies' is used out-of-position but is followed by an unnamed argument
```

٣. حدد موقع السطر التالي من التعليمات `RSVP("Tony", 1, "Jackfruit", true);`

٤. حدث استدعاء الأسلوب كما يلي:

```
RSVP("Tony", inviteOnly: true, allergies: "Jackfruit", partySize: 1);
```

لاحظ أنه لا يجب أن تظهر الوسيطات المسماة بالترتيب الأصلي، ومع ذلك، فإن الوسيطة غير المسماة Tony هي وسيطة موضعية، ويجب أن تظهر في الموضع المطابق.

٥. احفظ التعليمات البرمجية وشغلها لمراقبة الإخراج التالي:

```
Sorry, Tony is not on the guest list

Total RSVPs:
Name: Rebecca, Party Size: 1, Allergies: none
Name: Nadia, Party Size: 2, Allergies: Nuts
Name: Linh, Party Size: 2, Allergies: none
Name: Noor, Party Size: 4, Allergies: none
Name: Jonte, Party Size: 2, Allergies: Stone fruit
```

لاحظ أن استخدام الوسيطات المسماة لا يغير الإخراج.

الإعلان عن المعلمات الاختيارية Declare optional parameters

تصبح المعلمة اختيارية عندما يتم تعيين قيمة افتراضية لها، إذا تم حذف معلمة اختيارية من الوسائط، فسيتم استخدام القيمة الافتراضية عند تنفيذ الأسلوب، في هذه الخطوة، ستجعل المعلمات `partySize, allergies and inviteOnly` اختيارية.

١. لتعريف المعلمات الاختيارية، حدث توقيع الأسلوب `RSVP` كما يلي:

```
void RSVP(string name, int partySize = 1, string allergies = "none", bool inviteOnly = true)
```

خذ لحظة لمراقبة بناء الجملة syntax لا تزال المعلمات مفصولة بفواصل، ولكن يتم تعيين قيمة لكل من المعلمات `partySize, allergies, and inviteOnly`

بعد ذلك، ستقوم بتحديث الاستدعاءات للرد على الدعوة RSVP لتطبيق المعلمات الاختيارية.

٢. حدث التعليمات البرمجية إلى ما يلي:

```
RSVP("Rebecca");
RSVP("Nadia", 2, "Nuts");
RSVP(name: "Linh", partySize: 2, inviteOnly: false);
RSVP("Tony", allergies: "Jackfruit", inviteOnly: true);
RSVP("Noor", 4, inviteOnly: false);
RSVP("Jonte", 2, "Stone fruit", false);
```

في كل استدعاء أسلوب، لاحظ لا يتم حذف الاسم مطلقًا، عند استدعاء أسلوب ما، يجب تضمين كافة الوسيطات المطلوبة، ومع ذلك، يمكن حذف أية وسيطات اختيارية.

في هذه التعليمة البرمجية، قمت بإزالة الوسيطات `1, "none", true` من Rebecca's rsvp وبما أن هذه الوسيطات تطابق القيمة الافتراضية، فإن نتيجة rsvp الرد على دعوة Rebecca هي نفسها.

لقد أزلت الوسيطة inviteOnly من الرد الخاص بـ Nadia بما أن القيمة الافتراضية inviteOnly هي true فإن نتيجة rsvp الرد على الدعوة الخاصة ب Nadia هي نفسها.

لقد أزلت الوسيطة partySize من الرد الخاص بـ Tony إذا كان لدى Tony دعوة، فسيتم استخدام القيمة الافتراضية لـ partySize في الرد على الدعوة RSVP

لقد قمت بإزالة الوسيطة allergies من الرد الخاص بكل Linh and Noor's rsvps من ل ستعرض rsvps الخاصة بهم القيمة الافتراضية "لا شيء" none لـ "الحساسية" Allergies

٣. احفظ التعليمات البرمجية وشغلها لمراقبة الإخراج التالي:

```
Sorry, Tony is not on the guest list

Total RSVPs:
Name: Rebecca,  Party Size: 1,  Allergies: none
Name: Nadia,    Party Size: 2,  Allergies: Nuts
Name: Linh,     Party Size: 2,  Allergies: none
Name: Noor,     Party Size: 4,  Allergies: none
Name: Jonte,    Party Size: 2,  Allergies: Stone
fruit
```

لاحظ أنه يتم استخدام القيم الافتراضية بدلاً من الوسيطات المحذوفة، مثل partySize and allergies

خلاصة

إليك ما تعلمته عن الوسيطات الاختيارية والمسماة optional and named arguments حتى الآن:

- تصبح المعلمات Parameters اختيارية عن طريق تعيين قيمة افتراضية في توقيع الأسلوب method signature
- يتم تحديد الوسيطات المسماة باسم المعلمة، متبوعة بنقطتين وقيمة الوسيطة.
- عند دمج الوسيطات المسماة والموضعية positional arguments يجب استخدام الترتيب الصحيح للمعلمات

٦ تمرين ـ تحدي عرض عناوين البريد الإلكتروني

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

ينصب تركيز هذا التحدي على إنشاء أسلوب مع المعلمات المناسبة، بما في ذلك معلمة اختيارية optional parameter

عرض عناوين البريد الإلكتروني

يتمثل التحدي الذي تواجهه في إنشاء أسلوب يعرض عنوان البريد الإلكتروني الصحيح، للموظفين الداخليين والخارجيين، يتم إعطاؤك قوائم بأسماء الموظفين الداخليين والخارجيين، يتكون عنوان البريد الإلكتروني للموظف من اسم المستخدم، واسم مجال الشركة.

تنسيق اسم المستخدم هو أول حرفين من الاسم الأول للموظف، متبوعاً باسم العائلة، على سبيل المثال، سيكون لدى موظف يسمى "Robert Bavin" اسم المستخدم "robavin" مجال الموظفين الداخليين هو "contoso.com"

في هذا التحدي، يتم منحك بعض التعليمات البرمجية للبدء، يجب أن تقرر كيفية إنشاء أسلوب، واستدعاءه لعرض عناوين البريد الإلكتروني.

تحدي التعليمات البرمجية: إضافة أسلوب لعرض عناوين البريد الإلكتروني

في التعليمات البرمجية التي تبدأ بها، هناك مصفوفتان للموظفين الداخليين والخارجيين، تذكر أن مجال الموظفين الداخليين هو "contoso.com" واسم المستخدم لجميع الموظفين هو أول حرفين من الاسم الأول، متبوعاً باسم العائلة الكامل.

يتمثل التحدي الذي تواجهه في إنشاء أسلوب يعرض عنوان البريد الإلكتروني، للموظفين الداخليين والخارجيين، يجب أن يتضمن الأسلوب معلمة اختيارية، لاسم مجال الموظفين الخارجيين.

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 ملف، حدد فتح مجلد.
- استخدم مربع الحوار فتح مجلد للانتقال إلى المجلد CsharpProjects ثم فتحه.
- في قائمة Visual Studio Code EXPLORER، حدد Program.cs
- في قائمة حدد Selection حدد Select All ثم اضغط على مفتاح Delete

٢. انسخ التعليمات البرمجية التالية وألصقها في محرر Visual Studio Code

```
string[,] corporate =
{
    {"Robert", "Bavin"}, {"Simon", "Bright"},
    {"Kim", "Sinclair"}, {"Aashrita",
"Kamath"},
    {"Sarah", "Delucchi"}, {"Sinan", "Ali"}
};

string[,] external =
{
    {"Vinnie", "Ashton"}, {"Cody", "Dysart"},
    {"Shay", "Lawrence"}, {"Daren", "Valdes"}
};

string externalDomain = "hayworth.com";

for (int i = 0; i < corporate.GetLength(0);
i++)
{
```

```
    // display internal email addresses
}

for (int i = 0; i < external.GetLength(0); i++)
{
    // display external email addresses
}
```

٣. حدث التعليمات البرمجية، لاستخدام أسلوب لعرض عناوين البريد الإلكتروني، وفقاً لمواصفات التحدي.

استخدم ما تعلمته حول استخدام المعلمات والوسيطات الاختيارية لإكمال التحديث.

٤. تحقق من أن تعليماتك البرمجية تنتج الإخراج التالي:

```
robavin@contoso.com
sibright@contoso.com
kisinclair@contoso.com
aakamath@contoso.com
sadelucchi@contoso.com
siali@contoso.com
viashton@hayworth.com
codysart@hayworth.com
shlawrence@hayworth.com
davaldes@hayworth.com
```

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

٧ مراجعة حل تحدي عرض عناوين البريد الإلكتروني

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

```
string[,] corporate =
    {
        {"Robert", "Bavin"}, {"Simon", "Bright"},
        {"Kim", "Sinclair"}, {"Aashrita",
"Kamath"},
        {"Sarah", "Delucchi"}, {"Sinan", "Ali"}};

    string[,] external =
    {
        {"Vinnie", "Ashton"}, {"Cody", "Dysart"},
        {"Shay", "Lawrence"}, {"Daren", "Valdes"}
    };

    string externalDomain = "hayworth.com";

    for (int i = 0; i < corporate.GetLength(0);
i++)
    {
        DisplayEmail(first: corporate[i,0],
last: corporate[i,1]);
    }

    for (int i = 0; i < external.GetLength(0);
i++)
    {
        DisplayEmail(first: external[i,0],
last: external[i,1], domain: externalDomain);
    }
```

```
    void DisplayEmail(string first, string
last, string domain = "contoso.com")
    {
        string email = first.Substring(0, 2) +
last;
        email = email.ToLower();
        Console.WriteLine($"{email}@{domain}");
    }
```

هذه التعليمة البرمجية ليست سوى "حل واحد ممكن" لأنك ربما أضفت موجزات الأسطر line feeds في نقاط مختلفة، أو استخدمت المتغيرات variables بشكل مختلف، أو قمت بتنسيق التعليمات البرمجية بشكل مختلف.

بغض النظر عن اختلافات التعليمات البرمجية الثانوية، يجب أن يطبق الأسلوب الخاص بك القيمة الافتراضية لمعلمة اختيارية لعرضها `contoso.com`

عند تشغيل التعليمات البرمجية، يجب أن تشاهد الإخراج التالي:

```
    robavin@contoso.com
    sibright@contoso.com
    kisinclair@contoso.com
    aakamath@contoso.com
    sadelucchi@contoso.com
    siali@contoso.com
    viashton@hayworth.com
    codysart@hayworth.com
    shlawrence@hayworth.com
    davaldes@hayworth.com
```

إذا أكملت هذا التحدي، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

٨ اختبر معلوماتك

١- أي من تواقيع الأسلوب التالية يعرف بشكل صحيح ثلاث معلمات عدد صحيح؟

- `void Multiply(int a, b, c)`
- `void Multiply(int a, int b, int c)`
- `void Multiply(int a; int b; int c)`

٢- بالنظر إلى توقيع الأسلوب `void Print(string name, string number = "", bool member = false)` أي من الخيارات التالية يستخدم الوسيطات المسماة والاختيارية بشكل صحيح؟

- `Print("Tony", member: true);`
- `Print(number: "555", member: false)`
- `Print("Tony", false)`

٣- بالنظر إلى توقيع الأسلوب `void SetHealth(string health)` لماذا لا تؤثر العمليات داخل الأسلوب على سلسلة Strings الإدخال الأصلية `health`

- يتم تمرير نوع بيانات السلسلة حسب القيمة
- يجب أن تكون التعليمات البرمجية للأسلوب غير صحيحة
- لا يمكن تغيير السلاسل بمجرد تعيينها، لا يمكن الكتابة فوقها إلا بقيمة جديدة

راجع إجابتك

١ void Multiply(int a, int b, int c)

صحيح يتم تضمين نوع البيانات لكل معلمة ويتم فصل المعلمات بفواصل

٢ Print("Tony", member: true);

صحيح يتم تضمين الوسيطة الاختيارية باستخدام اسم المعلمة متبوعاً ب :

٣ لا يمكن تغيير السلاسل بمجرد تعيينها، لا يمكن الكتابة فوقها إلا بقيمة جديدة

صحيح يتم تمرير السلاسل حسب المرجع، ولكنها غير قابلة للتغيير ولا يمكن تغييرها بمجرد تعيينها

٩ الملخص

كان هدفك هو معرفة المزيد حول استخدام المعلمات في الأساليب، وفهم نطاق الأسلوب، لقد تعرفت على أنواع القيم، والمراجع، وكيفية تأثر البيانات داخل أسلوب.

كما تعلمت كيفية استخدام الوسيطات المسماة، والاختيارية، لتوسيع قدرات الأسلوب الخاص بك.

الوحدة الثالثة

إنشاء أساليب C# ترجع القيم methods that return values

تعلم كيفية إنشاء أساليب ترجع القيم

الأهداف التعليمية

- فهم أنواع الإرجاع return types
- تعلم المزيد حول الكلمة الأساسية return keyword
- تعلم المزيد حول التقاط، قيم إرجاع الأسلوب method return values

محتويات الوحدة: -

١- مقدمة

٢- تمرين - فهم بناء جملة نوع الإرجاع return type syntax

٣- تمرين - إرجاع الأرقام من الأساليب Return numbers from methods

٤- تمرين - إرجاع السلاسل من الأساليب Return strings from methods

٥- تمرين - إرجاع القيم المنطقية من الأساليب Return Booleans from methods

٦- تمرين - إرجاع المصفوفات من الأساليب Return arrays from methods

٧- تمرين أكمل التحدي - لإضافة أساليب لجعل اللعبة صالحة للعب

٨- راجع حل تحدي إضافة أساليب لجعل اللعبة صالحة للعب

٩- اختبر معلوماتك

١٠- الملخص

١ المقدمة

يمكن أن توفر الأساليب قيم الإرجاع بعد تنفيذ مهامها، باستخدام المعلمات وأنواع الإرجاع معاً، يمكنك إنشاء أساليب مبسطة تتلقى المدخلات، وتنفذ مهمة، وتوفر الإخراج، يسمح لك هذا التنسيق، ببناء وظائف في برامجك بكفاءة، مع الحفاظ على تعليمات برمجية نظيفة وسهلة القراءة.

لنفترض أنك بحاجة إلى إنشاء تطبيق يستخدم العديد من الأساليب، لإجراء عمليات حسابية على قيم الإدخال، تحتاج إلى طريقة لاسترداد نتائج العمليات الحسابية، واستخدام هذه النتائج في البرنامج، يمكنك القيام بذلك عن طريق إنشاء أساليب ذات قيم إرجاع.

ضع في اعتبارك لعبة حيث يجب على اللاعب محاربة الأعداء، تحتوي اللعبة على بعض التعليمات البرمجية التي تحدد ما إذا تم ضرب الشخصية عند استدعاء أسلوب `Update()` قد تحتوي التعليمات البرمجية على الأساليب التالية:

```
void Update();

int[] GetEnemyCoordinates(string enemyId);
int[] GetDistanceFromHero(string enemyId);
int[] GetHeroCoordinates();

bool EnemyCanHitHero(string enemyId);
int GetEnemyDamageOutput(string enemyId);
void UpdateHeroHP(int damage);
```

بالنظر إلى تواقيع الأسلوب، يمكنك تخيل كيفية استخدام الإدخال والإخراج، لكل أسلوب عبر البرنامج، تجعل الأساليب التعليمات البرمجية للعبة أكثر قوة، لأن كل واحدة لها قيم إرجاع، يمكن استخدامها لأنواع مختلفة من السيناريوهات.

يعد التقاط القيم المرجعة من الأساليب مفيداً، بشكل لا يصدق لجميع أنواع التطبيقات. في هذه الوحدة، ستتعرف على المزيد حول تنفيذ الأسلوب، والعمل مع أنواع إرجاع الأسلوب.

٢ تمرين - فهم بناء جملة نوع الإرجاع return type syntax

يمكن للأساليب تنفيذ العمليات، ويمكنها أيضاً إرجاع قيمة، ترجع الأساليب قيمة عن طريق تضمين نوع الإرجاع، في توقيع الأسلوب، يمكن أن ترجع الأساليب أي نوع بيانات، أو لا يمكنها إرجاع أي شيء على الإطلاق، يجب دائماً تحديد نوع الإرجاع قبل اسم الأسلوب.

استخدام void كنوع الإرجاع يعني أن الأسلوب ينفذ العمليات فقط، ولا يرجع قيمة. على سبيل المثال:

```
void PrintMessage(string message)
```

عند استخدام نوع بيانات مثل int، string، bool، وما إلى ذلك، يقوم الأسلوب بتنفيذ العمليات، ثم إرجاع النوع المحدد عند الانتهاء. داخل الأسلوب، يتم استخدام الكلمة الأساسية return لإرجاع النتيجة. في أساليب void يمكنك أيضاً استخدام الكلمة الأساسية return لإنهاء الأسلوب.

في هذا التمرين، ستتعرف على المزيد حول استخدام الكلمة الأساسية return

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** 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

استخدام الأساليب لحساب إجمالي سعر الشراء

يحظى مركز التسوق Contoso بتخفيضات رائعة! العديد من العناصر لديها سعر مخفض، يتم إعطاؤك قائمة بأسعار العناصر وقائمة بالخصومات المقابلة، يتم تمثيل الخصومات بنسب مئوية، على سبيل المثال 0.5 = 50% إذا كان العميل ينفق أكثر من 30.00$ دولار، فسيتلقى 5.00$ دولار من إجمالي مشترياته، في هذه المهمة، ستقوم بكتابة التعليمات البرمجية لحساب إجمالي العميل. لنبدأ!

١. أدخل التعليمات البرمجية التالية في المحرر:

```
double total = 0;
double minimumSpend = 30.00;

double[] items = {15.97, 3.50, 12.25, 22.99, 10.98};
double[] discounts = {0.30, 0.00, 0.10, 0.20, 0.50};

Console.WriteLine($"Total: ${total}");

void GetDiscountedPrice(int itemIndex)
{
    // Calculate the discounted price of the
item
}

void TotalMeetsMinimum()
{
    // Check if the total meets the minimum
}

void FormatDecimal(double input)
{
    // Format the double so only 2 decimal
places are displayed
}
```

في هذه الخطوة، يمكنك إعداد المتغيرات التي سيحتاجها البرنامج، وإنشاء أساليب نائبة placeholder methods سيتم استخدامها لتنفيذ المهام للحصول على إجمالي سعر الشراء.

ملاحظة:

تمثل العناصر النائبة نوع الإرجاع أو فئة التصريح أو وسيطة أسلوب يمكن ملء العناصر النائبة تلقائيًا أو يدويًا وفقًا لنوع العنصر النائب، العناصر النائبة للوسيطات، التي تقوم بإكمالها يدويًا عند تحرير القواعد، ويتم تمثيلهم بواسطة فهرس الوسيطة.

٢. قم بتغيير الأسلوب `GetDiscountedPrice` لإرجاع رقم عشري `double` عن طريق تحديث التعليمات البرمجية إلى ما يلي:

```
double GetDiscountedPrice(int itemIndex)
{
    // Calculate the discounted price of the
item
}
```

لاحظ أن هذا الأسلوب ينتج عنه خطأ التحويل البرمجي `not all code paths return a value` يجب أن يرجع أسلوب نوع الإرجاع دائماً قيمة من هذا النوع، دعونا نصلح هذا الخطأ.

٣. حدث الأسلوب `GetDiscountedPrice` بالتعليمات البرمجية التالية:

```
double GetDiscountedPrice(int itemIndex)
{
    double result = items[itemIndex] * (1 -
discounts[itemIndex]);
    return result;
}
```

لإرجاع قيمة من أسلوب، أضف قيمة أو تعبيراً بعد الكلمة الأساسية return يجب أن تتطابق القيمة التي تم إرجاعها، مع نوع البيانات المحدد في توقيع الأسلوب.

في هذه التعليمة البرمجية، يمكنك حساب الخصم للعنصر في itemIndex ثم إرجاع النتيجة، ومع ذلك لا تقتصر الكلمة الأساسية return على إرجاع المتغيرات أو القيم الحرفية، دعونا نتخطى المتغير ونعيد التعبير بدلاً من ذلك.

٤. حدث الأسلوب `GetDiscountedPrice` بالتعليمات البرمجية التالية:

```
double GetDiscountedPrice(int itemIndex)
{
    return items[itemIndex] * (1 -
discounts[itemIndex]);
}
```

لأن التعليمات `items[itemIndex] * (1 - discounts[itemIndex])` تقيم إلى قيمة `double` فإن العبارة return صالحة.

٥. قم بتغيير أسلوب `TotalMeetsMinimum` لإرجاع `bool` عن طريق تحديث التعليمات البرمجية إلى ما يلي:

```
bool TotalMeetsMinimum()
{
    return total >= minimumSpend;
}
```

في هذه التعليمة البرمجية، يمكنك إرجاع نتيجة المقارنة، التي يتم تقييمها إلى bool يعد إرجاع التعبيرات من الأساليب، طريقة رائعة لتنظيم التعليمات البرمجية، وتحسين سهولة القراءة.

٦. قم بتغيير أسلوب `FormatDecimal` لإرجاع `string` عن طريق تحديث التعليمات البرمجية إلى ما يلي:

```
string FormatDecimal(double input)
{
    return input.ToString().Substring(0, 5);
}
```

لاحظ أنه يمكنك استدعاء أساليب أخرى في تعبير بيان الإرجاع `return` عندما يتم استدعاء هذا الأسلوب، يقوم عنصر التحكم في التنفيذ، أولاً بتقييم `input.ToString` ثم تقييم وإرجاع قيمة السلسلة الفرعية، تسمح لك مرونة الكلمة الأساسية return بإنشاء أسطر قصيرة من التعليمات البرمجية مليئة بالوظائف.

التقاط القيم المرجعة

الآن بعد أن أرجعت أساليبك القيم، يمكنك التقاط هذه القيم، واستخدامها في بقية التعليمات البرمجية، يعد استخدام النتائج التي يتم إرجاعها من الأساليب طريقة رائعة للحفاظ على تعليمتك البرمجية منظمة، وسهلة القراءة. دعونا نتدرب!

١. حدد موقع السطر التالي من التعليمات البرمجية:

```
Console.WriteLine($"Total: ${total}");
```

٢. حدث التعليمات البرمجية الخاصة بك إلى ما يلي:

```
for (int i = 0; i < items.Length; i++)
{
    total += GetDiscountedPrice(i);
}

Console.WriteLine($"Total: ${total}");
```

في هذه التعليمة البرمجية، يمكنك استخدام التكرار for للحصول على مجموع جميع أسعار العناصر المخفضة، لأن `GetDiscountedPrice` يرجع قيمة double يمكنك استخدام الأسلوب لتهيئة أو العمل على أي متغيرات أخرى double في هذه الحالة، يمكنك استخدام نتيجة `GetDiscountedPrice` لزيادة إجمالي `total`

٣. أدخل سطر فارغ قبل `Console.WriteLine` ثم أضف التعليمات البرمجية التالية:

```
if (TotalMeetsMinimum())
{
```

```
    total -= 5.00;
}
```

لأن `TotalMeetsMinimum` يعيد قيمة منطقية `bool` يمكنك استدعاء الأسلوب الموجود داخل الشرط `if` يمكنك أيضًا استخدام هذا الأسلوب في تعبير ثلاثي على النحو التالي:

```
total -= TotalMeetsMinimum() ? 5.00 : 0.00;
```

٤. استخدم `FormatDecimal` لتنسيق سعر الشراء المعروض عن طريق تحديث التعليمات البرمجية كما يلي:

```
Console.WriteLine($"Total: ${FormatDecimal(total)}");
```

على غرار التعليمات السابقة، يقوم `FormatDecimal` بإرجاع قيمة السلسلة string بحيث يمكنك استدعاء الأسلوب بدلاً من استخدام متغير سلسلة أو قيمة حرفية، في هذه التعليمات، يتم تمرير التحكم في التنفيذ إلى `FormatDecimal` لتقييم النتيجة قبل تقييم `Console.WriteLine`

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقك من الوحدة الطرفية المتكاملة والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
double total = 0;
double minimumSpend = 30.00;

double[] items = {15.97, 3.50, 12.25, 22.99,
10.98};
double[] discounts = {0.30, 0.00, 0.10, 0.20,
0.50};

for (int i = 0; i < items.Length; i++)
{
    total += GetDiscountedPrice(i);
```

```
}

total -= TotalMeetsMinimum() ? 5.00 : 0.00;

Console.WriteLine($"Total:
${FormatDecimal(total)}");

double GetDiscountedPrice(int itemIndex)
{
    return items[itemIndex] * (1 -
discounts[itemIndex]);
}

bool TotalMeetsMinimum()
{
    return total >= minimumSpend;
}

string FormatDecimal(double input)
{
    return input.ToString().Substring(0, 5);
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٤. في موجه الأوامر Terminal أدخل dotnet run

٥. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
Total: $44.58
```

إذا كانت التعليمات البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها للعثور على الخطأ وإجراء التعديلات، شغل التعليمات البرمجية مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، تابع تعديل التعليمات حتى تنتج النتائج المتوقعة.

خلاصة

إليك ما تعلمته عن الكلمة الأساسية return حتى الآن:

- يمكن أن ترجع الأساليب قيمة عن طريق تحديد نوع بيانات الإرجاع، أو void بدون قيمة إرجاع.
- يمكن استخدام الكلمة الأساسية return مع المتغيرات والقيم الحرفية والتعبيرات.
- يجب أن تتطابق القيمة التي تم إرجاعها من أسلوب مع نوع الإرجاع المحدد.
- يمكن التقاط البيانات التي يتم إرجاعها من الأساليب واستخدامها بواسطة مستدعي الأسلوب.

٣ تمرين - إرجاع الأرقام من الأساليب Return numbers from methods

قد تحتاج غالباً إلى إرجاع أرقام من الأساليب، واستخدام النتائج للمهام الأخرى، في هذا التمرين الموجز، سوف تمارس إرجاع أنواع البيانات int and double والتقاط قيم الإرجاع.

إنشاء أسلوب يرجع عدداً صحيحاً integer

لنفترض أنك تزور فيتنام وتريد إنشاء برنامج موجز يحول العملة، يمكنك افتراض أن سعر الصرف الحالي هو `1 USD = 23500 VND` في هذه المهمة، ستكتب أسلوباً يحول الدولار الأمريكي إلى VND

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
double usd = 23.73;
int vnd = UsdToVnd(usd);

Console.WriteLine($"${usd} USD = ${vnd} VND");

int UsdToVnd(double usd)
{

}
```

في هذه الخطوة، يمكنك تهيئة متغيرين لتخزين قيم USD and VND لاحظ أنه يتم تهيئة `vnd` إلى نتيجة الأسلوب `UsdToVnd` يقوم الأسلوب بإرجاع قيمة عدد صحيح لأن `vnd` يتم تمثيله عادة بأرقام كاملة، لعرض نتائج تحويل العملة يتم استخدام `Console.WriteLine`

٣. بعد ذلك، ستضيف تعليمة برمجية لإجراء التحويل، حدث الأسلوب `UsdToVnd` بالتعليمات البرمجية التالية:

```
int UsdToVnd(double usd)
{
    int rate = 23500;
    return (int) (rate * usd);
}
```

إذا حذفت التحويل من نتيجة الإرجاع، فسترى الخطأ التالي:

```
Cannot implicitly convert type 'double' to 'int'.
```

يحدث هذا لأن المحول البرمجي يحاول "cast" تحويل القيمة التي تم إرجاعها، لمطابقة نوع البيانات المحدد في توقيع الأسلوب، ومع ذلك، التحويل المرسل الضمني implicit casting يتاح عندما لا يحدث فقدان للبيانات نتيجة التحويل، يجب أن تتطابق القيمة المرجعة دائماً مع نوع البيانات المحدد في توقيع الأسلوب، لذلك في هذه الحالة، يجب تحويل النتيجة cast the result

٤. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

٥. في موجه الأوامر Terminal أدخل dotnet run وقارن الإخراج مع ما يلي:

```
$23.73 USD = $557655 VND
```

إذا كانت التعليمات البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها للعثور على الخطأ وإجراء التعديلات، شغل التعليمات البرمجية مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، تابع تعديل التعليمات حتى تنتج النتائج المتوقعة.

إنشاء أسلوب يقوم بإرجاع رقم عشري double

بعد ذلك، ستقوم بإنشاء طريقة لتحويل VND مرة أخرى إلى الدولار الأمريكي.

١. إنشاء سطر فارغ في نهاية الأسلوب `UsdToVnd`

٢. أدخل التعليمات التالية:

```
double VndToUsd(int vnd)
{

}
```

٣. حدث الأسلوب `VndToUsd` بالتعليمات البرمجية التالية:

```
double VndToUsd(int vnd)
{
    double rate = 23500;
    return vnd / rate;
}
```

في هذه الحالة، تحتاج `rate` لأن تكون double وإلا فإن المحول البرمجي يستخدم قسمة عدد صحيح وإرجاع قيمة int مقتطعة، يجب تمثيل الدولار الأمريكي بعدد عشري.

إذا قمت بتعيين `rate` إلى int بدلاً من double فستلاحظ أن المحول البرمجي لا يقدم لك أي أخطاء، يحدث هذا لأن قيمة `vnd / rate` يتم تحويلها ضمنياً إلى double نوع البيانات المحدد في توقيع الأسلوب، عند إنشاء أساليب تُرجع القيم الرقمية، من المهم مراعاة أنواع البيانات في العمليات التي ينفذها أسلوب.

٤. حدد موقع الاستدعاء `Console.WriteLine` وألحق سطر فارغ جديد، ثم أدخل التعليمات التالية لاستدعاء أسلوبنا الجديد وطباعة الإخراج:

```
Console.WriteLine($"${vnd}          VND          =
${VndToUsd(vnd)} USD");
```

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقنا من الوحدة الطرفية المتكاملة، والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
double usd = 23.73;
int vnd = UsdToVnd(usd);

Console.WriteLine($"${usd} USD = ${vnd} VND");
Console.WriteLine($"${vnd} VND =
${VndToUsd(vnd)} USD");

int UsdToVnd(double usd)
{
    int rate = 23500;
    return (int) (rate * usd);
}

double VndToUsd(int vnd)
{
    double rate = 23500;
    return vnd / rate;
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٤. في موجه الأوامر Terminal أدخل dotnet run

٥. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
$23.73 USD = $557655 VND
$557655 VND = $23.73 USD
```

اختبر معلوماتك

١. نوع البيانات الذي يتم إرجاعه من العبارة `return 100 * 0.5;`

- An int type
- A decimal type
- A double type

٢. ما نوع الأسلوب الذي لا يحتاج إلى تضمين عبارة return

- string methods
- void methods
- array methods

راجع إجابتك

١ A double type

صحيح لأن القيمة الحرفية لعدد صحيح integer يتم ضربها بواسطة double فإن هذا السطر من التعليمات البرمجية يرجع قيمة double

٢ void methods

صحيح لا تحتاج أساليب void إلى استخدام عبارة return

٤ تمرين - إرجاع السلاسل من الأساليب Return strings from methods

قد تجد في كثير من الأحيان أنك بحاجة إلى كتابة أسلوب يرجع سلسلة string على سبيل المثال، قد تحتاج إلى استرداد سلسلة من مجموعة من البيانات أو تعديل سلسلة بطريقة ما، في هذا التمرين، ستكتسب بعض الخبرة في العمل مع السلاسل strings في الأساليب أثناء ممارسة سؤال مقابلة شائع.

إنشاء أسلوب يقوم بإرجاع سلسلة returns a string

لنفترض أنك مرشح في مقابلة ترميز، يطلب منك المحاور كتابة أسلوب لعكس سلسلة دون استخدام `string.Reverse` خذ لحظة للتفكير في كيفية إنجاز هذه المهمة.

ربما قررت أنه يمكنك عكس سلسلة string بالتكرار من نهاية السلسلة، يمكنك استخدام سلسلة مؤقتة لتخزين كل حرف من النهاية إلى البداية، نبدأ!

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
string ReverseWord(string word)
{
    string result = "";

    return result;
}
```

٣. يحتاج الأسلوب إلى التكرار من خلال الكلمة المحددة، وتحديث النتيجة، للقيام بذلك، قم بتحديث الأسلوب `ReverseWord` بالتعليمات التالية:

```
string ReverseWord(string word)
{
    string result = "";
    for (int i = word.Length - 1; i >= 0; i--)
    {
```

```
        result += word[i];
    }
    return result;
}
```

في كتلة التعليمات البرمجية هذه، تبدأ من نهاية الكلمة باستخدام `word.Length - 1` تقوم بطرح واحد من الطول، نظرًا لأن فهرس المصفوفة يبدأ من الصفر، وتريد تجنب الوصول إلى عنصر خارج الحدود، ثم تقوم بإضافة الحرف الموجود في الفهرس الحالي إلى السلسلة `result` وتحريك الفهرس إلى الخلف، يمكنك استخدام `i >= 0` حيث يتم تحديث `i` بعد تنفيذ التعليمات البرمجية الموجودة في الحلقة، وتريد التأكد من تضمين الفهرس الصفري.

تحقق من الكود

عند كتابة التعليمات البرمجية، من المهم التحقق بشكل متكرر من عملك، يتيح لك العثور على الأخطاء، وتصحيحها في وقت مبكر من عملية الترميز، قضاء المزيد من الوقت في البناء على التعليمات البرمجية الصحيحة، بدلاً من تصحيح أحد البرامج الكبيرة، يعد التحقق من عملك بشكل متكرر مهارة يقدرها القائمون على مقابلات البرمجة، بشكل كبير أيضاً.

١. أدخل سطر فارغ، ثم قم بإنشاء بعض نص الإدخال input text واستدعاء أسلوبك عن طريق إدخال التعليمات التالية فوق الأسلوب `ReverseWord`

```
string input = "snake";

Console.WriteLine(input);
Console.WriteLine(ReverseWord(input));
```

٢. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

٣. في موجه الأوامر Terminal أدخل dotnet run قارن الإخراج مع ما يلي:

```
snake
ekans
```

إنشاء أسلوب لعكس الكلمات في جملة reverse words in a sentence

لنفترض أن المحاور يسألك سؤال متابعة، يريد منك عكس كل كلمة في جملة معينة، مع الحفاظ على الموضع الأصلي لكل كلمة، يمكنك افتراض أن كل كلمة مفصولة بمسافة، على سبيل المثال، ستصبح جملة نوع إرجاع السلسلة "string return type" هكذا "gnirts nruter epyt" خذ لحظة للتفكير في كيفية تنفيذ هذه المهمة.

إذا كنت تستخدم الأسلوب الذي كتبته في المهمة السابقة، فقد تدرك أنه يمكنك استخدام الأسلوب لعكس كل كلمة في السلسلة بشكل فردي، يمكنك إنشاء جملة جديدة، وإضافة كل كلمة أثناء عكسها، لنبدأ!

١. إنشاء سطر فارغ، في نهاية البرنامج الحالي، ثم أدخل التعليمات التالية لإنشاء أسلوب جديد:

```
string ReverseSentence(string input)
{
    string result = "";

    return result;
}
```

٢. بعد ذلك، يمكنك استخراج الكلمات الفردية من السلسلة باستخدام `string.Split` حدث الأسلوب `ReverseSentence` إلى ما يلي:

```
string ReverseSentence(string input)
{
    string result = "";
    string[] words = input.Split(" ");

    return result;
}
```

الآن بعد أن أصبح لديك حق الوصول إلى كل كلمة فردية في الجملة، يمكنك استخدام الأسلوب `ReverseWord` على كل كلمة وتخزينها في `result`

٣. حدث الأسلوب `ReverseSentence` إلى ما يلي:

```
string ReverseSentence(string input)
{
    string result = "";
    string[] words = input.Split(" ");

    foreach(string word in words)
    {
        result += ReverseWord(word) + " ";
    }

    return result;
}
```

لاحظ كيف يمكنك استدعاء الأسلوب `ReverseWord` داخل عامل التعيين المركب compound assignment operator في هذه التعليمة البرمجية، يتم التقاط القيمة المرجعة من `ReverseWord` وإضافتها إلى `result` يمكن استخدام الأساليب ذات القيم المرجعة return values في أي مكان تحتاجه، طالما أن نوع البيانات يناسب المتطلبات.

في هذه التعليمة البرمجية، يتم إلحاق كل كلمة معكوسة في `result` بمسافة إضافية، ومع ذلك، يترك هذا مسافة إضافية extra space في نهاية الجملة المعكوسة.

٤. يمكنك إزالة المسافة الإضافية في النهاية باستخدام `string.Trim` حدث الأسلوب إلى التعليمات البرمجية التالية:

```
string ReverseSentence(string input)
{
    string result = "";
    string[] words = input.Split(" ");

    foreach(string word in words)
```

```
    {
        result += ReverseWord(word) + " ";
    }

    return result.Trim();
}
```

خذ لحظة للنظر في نتيجة إرجاع هذا الأسلوب، يمكن للأسلوب استخدام أساليب أخرى خلال تنفيذه، وحتى في عبارة الإرجاع، طالما أن أنواع البيانات متطابقة.

الآن أنت مستعد لاستدعاء الأسلوب الخاص بك!

٥. حدث `input` text and the `Console.WriteLine` statement إلى ما يلي:

```
string input = "there are snakes at the zoo";

Console.WriteLine(input);
Console.WriteLine(ReverseSentence(input));
```

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقك من الوحدة الطرفية المتكاملة، والتحقق من أن التعليمات تعمل بشكل صحيح، لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
string input = "there are snakes at the zoo";

Console.WriteLine(input);
Console.WriteLine(ReverseSentence(input));

string ReverseSentence(string input)
{
    string result = "";
```

```
    string[] words = input.Split(" ");
    foreach(string word in words)
    {
        result += ReverseWord(word) + " ";
    }
    return result.Trim();
}

string ReverseWord(string word)
{
    string result = "";
    for (int i = word.Length - 1; i >= 0; i--)
    {
        result += word[i];
    }
    return result;
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٤. في موجه الأوامر Terminal أدخل **dotnet run**

٥. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
there are snakes at the zoo
ereht era sekans ta eht ooz
```

٥ تمرين - إرجاع القيم المنطقية من الأساليب Return Booleans from methods

يمكن أن تكون الأساليب ذات أنواع الإرجاع المنطقية Boolean return types بسيطة ولكنها مفيدة في دمج التعليمات البرمجية، يمكن استدعاء الأساليب التي تُرجع القيم المنطقية `bool` لتقييم إدخال البيانات في أي مكان، وفي عبارات `if` وفي إعلانات المتغيرات variable declarations والمزيد في هذا التمرين، ستكتسب بعض الخبرة في إنشاء واستخدام أساليب نوع إرجاع منطقية Boolean return type methods

إنشاء أسلوب يرجع قيمة منطقية

لنفترض أنك مرشح في مقابلة ترميز، يريد منك المحاور أن تتحقق مما إذا كانت عدة كلمات متناظرة words are a palindrome تكون الكلمة متناظرة تقرأ بنفس الطريقة من أولها مثل أخرها، على سبيل المثال، الكلمة `racecar` هي كلمة متناظرة، لنبدأ!

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
string[] words = {"racecar" ,"talented",
"deified", "tent", "tenet"};

Console.WriteLine("Is it a palindrome?");
foreach (string word in words)
{
    Console.WriteLine($"{word}:
{IsPalindrome(word)}");
}
```

تؤسس هذه التعليمة البرمجية بعض حالات الاختبار، وتشير إلى أسلوب يسمى `IsPalindrome` تتم طباعة كلمات وإخراج الأسلوب `IsPalindrome` في عبارات `Console.WriteLine`

٣. أدخل سطر فارغ، وأنشئ أسلوب منطقي `bool` method بإدخال التعليمات التالية:

```
bool IsPalindrome(string word)
{
    return true;
}
```

٤. ضع في اعتبارك كيف يمكنك التحقق مما إذا كانت الكلمة متناظرة.

إحدى طرق التحقق هي مقارنة الحروف الأولى والأخيرة من الكلمة. إذا كانت متطابقة، فقم بمقارنة الحرف الثاني والحرف قبل الأخير من الكلمة، إذا وصلت إلى منتصف الكلمة، فقد تمت مقارنة جميع الحروف ومطابقتها، إذا لم تتطابق أي أحرف، فإن الكلمة ليست متناظرة.

٥. حدث الأسلوب `IsPalindrome` بالتعليمات البرمجية التالية:

```
bool IsPalindrome(string word)
{
    int start = 0;
    int end = word.Length - 1;

    while (start < end)
    {
        if (word[start] != word[end])
        {
            return false;
        }
        start++;
        end--;
    }

    return true;
}
```

لاحظ أن المتغيرات `start` and `end` للإشارة إلى الأحرف الأولى والأخيرة في السلسلة characters in the string تنقطع حلقة التكرار عند استيفاء منتصف الكلمة؛ عندما يشير `start` and `end` إلى نفس الحرف أو تتقاطع مع بعضها البعض، يتم نقل المؤشرات إلى الداخل في كل مرة يكون هناك تطابق، إذا لم تكن متطابقة، ينتهي الأسلوب ويرجع خطأ false

الآن أسلوبك يتحقق بنجاح ما إذا كانت الكلمة متناظرة palindrome وإرجاع true أو false وفقاً لذلك.

التحقق من عملك

في هذه المهمة، ستقوم بتشغيل تطبيقك من الوحدة الطرفية المتكاملة والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٢. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٣. في موجه الأوامر Terminal أدخل **dotnet run**

٤. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
Is it a palindrome?
racecar: True
talented: False
deified: True
tent: False
tenet: True
```

٦ تمرين - إرجاع المصفوفات من الأساليب Return arrays from methods

عند تطوير التطبيقات، ستحتاج غالبا إلى إنشاء مجموعات من البيانات وتعديلها، تعد الأساليب مفيدة لتنفيذ العمليات على البيانات، وهي أدوات قوية بشكل خاص لبناء مجموعات البيانات نفسها، يساعد تطوير أساليب لإنشاء مصفوفات تمثل مجموعة بياناتك، على الحفاظ على التعليمات البرمجية قابلة لإعادة الاستخدام، وتنظيمها وتبسيطها. في هذا التمرين، ستتدرب على إرجاع المصفوفات من الأساليب.

البحث عن عملات معدنية لإجراء التغيير

لنفترض أن لديك العديد من العملات المعدنية من قيم مختلفة، يتم تكليفك بالعثور على عملتين معدنيتين يساوي مجموعهما القيمة المستهدفة، في هذا التمرين، يتم تمثيل العملات المعدنية المتوفرة في مصفوفة عدد صحيح integer array ستحتاج إلى إرجاع فهارس العملتين في مصفوفة جديدة. لنبدأ!

١. في محرر Visual Studio Code احذف أي تعليمة برمجية موجودة من التدريبات السابقة.

٢. أدخل التعليمات البرمجية التالية في المحرر:

```
int[] TwoCoins(int[] coins, int target)
{
    return  new int[0];
}
```

في حالة عدم العثور على عملتين معدنيتين، يرجع الأسلوب بإرجاع مصفوفة فارغة empty array خذ لحظة للنظر في بناء جملة النتيجة التي تم إرجاعها returned result بينما يمكنك إنشاء متغير لتخزين مصفوفة جديدة `int[]` array وإرجاع المتغير، تسمح لك العبارة return بإنشاء القيم وإرجاعها في وقت واحد.

٣. هناك العديد من الأساليب لحل هذه المشكلة، خذ لحظة للنظر في كيفية البحث عن رقمين في مصفوفة، يكون مجموعه مساوياً لقيمة معينة.

في هذا التمرين، سيتم استخدام النهج التالي:

- اختر رقمًا واحدًا من المصفوفة.
- تحقق من الأرقام الأخرى واحدًا تلو الآخر لمعرفة ما إذا كانت تضيف ما يصل إلى القيمة المستهدفة.
- إرجاع النتيجة بمجرد العثور على تطابق.

٤. للتحقق من كل رقم في المصفوفة، قم بتحديث الأسلوب `TwoCoins` بالتعليمات البرمجية التالية:

```
int[] TwoCoins(int[] coins, int target)
{
    for (int curr = 0; curr < coins.Length;
curr++)
    {
        for (int next = curr + 1; next <
coins.Length; next++)
        {

        }
    }

    return  new int[0];
}
```

هنا يمثل `curr` فهرساً واحدًا index للعملة الثابتة، ويمثل `next` فهارس العملة اللاحقة، ستحاول إضافة كل عملة معدنية `next` مع العملة الثابتة `curr` لمعرفة ما إذا كانت تساوي القيمة المستهدفة.

٥. بعد ذلك، أضف منطقًا للتحقق من مجموع عملتين للقيمة المستهدفة، للقيام بذلك، قم بتحديث حلقات for السابقة بالتعليمات البرمجية التالية:

```
for (int curr = 0; curr < coins.Length; curr++)
{
    for (int next = curr + 1; next < 
coins.Length; next++)
    {
        if (coins[curr] + coins[next] == 
target)
        {
            return new int[]{curr, next};
        }

    }
}
```

في هذه التعليمة البرمجية، تتحقق مما إذا كان مجموع القيم curr and next في المصفوفة مساوياً للقيمة المستهدفة، إذا كان المجموع متساوياً، يمكنك إنشاء مصفوفة لتخزين هذه الفهارس وإعادتها، إذا لم تكن متساوية، يمكنك تجاهلها، ومتابعة التدقيق.

اختبر الحل الخاص بك

في هذه الخطوة، ستختبر التعليمات البرمجية للتأكد من أنها تعمل بشكل صحيح، أولاً ستقوم بتهيئة بعض المتغيرات لتخزين بيانات الإدخال، ثم ستستدعي الأسلوب، وتطبع النتائج.

١. أدخل سطر فارغ، أعلى توقيع الأسلوب TwoCoins ثم أدخل التعليمات التالية:

```
int target = 60;
int[] coins = new int[] {5, 5, 50, 25, 25, 10, 5};
int[] result = TwoCoins(coins, target);
```

تذكر أن الأسلوب `TwoCoins` يرجع مصفوفة فارغة empty array إذا لم يتم العثور على أي تغيير، ستحتاج إلى التحقق من حجم المصفوفة قبل محاولة طباعة مصفوفة `result`

٢. أدخل سطر فارغ، ثم أدخل التعليمات البرمجية التالية:

```
if (result.Length == 0)
{
    Console.WriteLine("No two coins make
change");
}
else
{
    Console.WriteLine($"Change found at
positions {result[0]} and {result[1]}");
}
```

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

٤. في موجه الأوامر Terminal أدخل dotnet run قارن الإخراج الخاص بك مع ما يلي:

```
Change found at positions 2 and 5
```

إذا كانت التعليمات البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها للعثور على الخطأ وإجراء التعديلات، شغل التعليمات البرمجية مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، تابع تعديل التعليمات حتى تنتج النتائج المتوقعة.

العثور على أزواج متعددة من العملات التي تُحدث التغيير

في هذه الخطوة، ستقوم بتوسيع الأسلوب `TwoCoins` للعثور على المزيد من أزواج العملات، التي يساوي مجموعها القيمة المستهدفة، في هذا التمرين، ستجد خمسة أزواج كحد أقصى، وهذا يعني أن نوع الإرجاع سيكون مصفوفة ثنائية الأبعاد 2D array بدلاً من مصفوفة أحادية الأبعاد 1D array وستحتاج إلى تعديل الطريقة التي تعرض بها التعليمات البرمجية النتائج. لنبدأ!

١. قم بتغيير نوع الإرجاع في توقيع الأسلوب من `int[]` to `int[,]` عن طريق تحديث تعليماتك إلى ما يلي:

```
int[,] TwoCoins(int[] coins, int target)
```

بعد ذلك، ستقوم بإنشاء مصفوفة `int[,]` لتخزين نتائجك وإرجاعها، ومتغير عداد counter لتتبع عدد الأزواج المضافة إلى المصفوفة.

٢. حدث الأسلوب `TwoCoins` بالتعليمات البرمجية التالية:

```
int[,] TwoCoins(int[] coins, int target)
{
    int[,] result = {{-1,-1},{-1,-1},{-1,-1},{-
1,-1},{-1,-1}};
    int count = 0;
```

لاحظ أنك قمت بتهيئة عناصر مصفوفة `result` إلى -١ سيساعدك هذا لاحقًا عندما تريد طباعة الأزواج التي تم العثور عليها.

بعد ذلك، ستستخدم مصفوفة `result` لتخزين كل زوج تم العثور عليه بدلاً من إرجاع التطابق الأول.

٣. حدث الأسلوب `TwoCoins` بالتعليمات البرمجية التالية:

```
int[,] TwoCoins(int[] coins, int target)
{
    int[,] result = {{-1,-1},{-1,-1},{-1,-1},{-
1,-1},{-1,-1}};
```

```
    int count = 0;

    for (int curr = 0; curr < coins.Length;
curr++)
    {
        for (int next = curr + 1; next <
coins.Length; next++)
        {
            if (coins[curr] + coins[next] ==
target)
            {
                result[count, 0] = curr;
                result[count, 1] = next;
                count++;
            }

        }
    }
```

لاحظ أن العدد `count` يزداد في كل مرة تتم فيها إضافة زوج إلى المصفوفة، يمكن أن يتسبب هذا في حدوث خطأ في الفهرس، خارج الحدود، إذا تم العثور على أكثر من خمسة أزواج، لمنع هذا الخطأ، يمكنك إضافة تعليمات برمجية للتحقق من قيمة `count` وإرجاع النتيجة التي تم ملء مصفوفة `result` بها.

٤. حدث المنطق "logic" في الأسلوب `TwoCoins` مع التعليمات البرمجية التالية:

```
for (int next = curr + 1; next < coins.Length;
next++)
{
    if (coins[curr] + coins[next] == target)
    {
```

```
            result[count, 0] = curr;
            result[count, 1] = next;
            count++;
        }
        if (count == result.GetLength(0))
        {
            return result;
        }
}
```

وأخيرًا، ستحتاج إلى تحديث بيان الإرجاع النهائي لإرجاع النتيجة الصحيحة إذا لم يتم العثور على أي تطابقات على الإطلاق، أو إذا تم العثور على أقل من خمس تطابقات.

٥. انتقل إلى العبارة return في الأسلوب `TwoCoins` وعدل العبارة return لمطابقة التعليمات البرمجية التالية:

```
if (count == 0)
{
    return new int[0,0];
}
return result;
```

يمكنك أيضًا تقصير رمز الإرجاع هذا، باستخدام عامل التشغيل الثلاثي كما يلي :

```
return (count == 0) ? new int[0,0] : result;
```

هام

تسمح لك مرونة الكلمة الأساسية return بإرجاع قيمة التعبير، طالما أن النتيجة تطابق نوع الإرجاع المحدد للأسلوب.

٦. عند هذه النقطة، يجب أن يتطابق الأسلوب `TwoCoins` مع التعليمات البرمجية التالية:

```
int[,] TwoCoins(int[] coins, int target)
{
    int[,] result = {{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}};
    int count = 0;

    for (int curr = 0; curr < coins.Length; curr++)
    {
        for (int next = curr + 1; next < coins.Length; next++)
        {
            if (coins[curr] + coins[next] == target)
            {
                result[count, 0] = curr;
                result[count, 1] = next;
                count++;
            }
            if (count == result.GetLength(0))
            {
                return result;
            }
        }
    }
    return (count == 0) ? new int[0,0] : result;
}
```

التقاط مصفوفة الإرجاع الجديدة new return array

الآن بعد أن ارجع أسلوبك مصفوفة ثنائية الأبعاد 2D يمكنك تحديث تعليماتك البرمجية لاسترداد النتائج وطباعتها، نظرًا لأنه تمت تهيئة عناصر المصفوفة result إلى -١ يمكنك إضافة علامة اختيار لطباعة جميع الأزواج حتى يتم العثور على القيمة -١

١. انتقل إلى بداية البرنامج حيث يتم تعريف المتغير `target` عدل تعليماتك البرمجية كما يلي:

```
int target = 30;
int[] coins = new int[] {5, 5, 50, 25, 25, 10, 5};
int[,] result = TwoCoins(coins, target);
```

بعد ذلك، ستقوم بتحديث استدعاء `Console.WriteLine` لطباعة قيم `result` بشكل صحيح.

٢. انتقل إلى استدعاء Console.WriteLine وحدث تعليماتك لتطابق ما يلي:

```
if (result.Length == 0)
{
    Console.WriteLine("No two coins make
change");
}
else
{
    Console.WriteLine("Change found at
positions:");
    for (int i = 0; i < result.GetLength(0);
i++)
    {
        if (result[i,0] == -1)
        {
            break;
```

```
        }

Console.WriteLine($"{result[i,0]},{result[i,1]}
");
    }
}
```

هنا، يمكنك الاحتفاظ بالتحقق من وجود مصفوفة فارغة كما هي، وطباعة قيم المصفوفة ثنائية الأبعاد في حلقة for عند العثور على قيمة -١ فإنك تخرج من الحلقة نظرًا لعدم وجود أزواج تالية.

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق الخاص بك من الوحدة الطرفية المتكاملة والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
int target = 30;
int[] coins = new int[] {5, 5, 50, 25, 25, 10, 5};
int[,] result = TwoCoins(coins, target);

if (result.Length == 0)
{
    Console.WriteLine("No two coins make
change");
}
else
{
    Console.WriteLine("Change found at
positions:");
    for (int i = 0; i < result.GetLength(0); i++)
    {
```

```
        if (result[i,0] == -1)
        {
            break;
        }

Console.WriteLine($"{result[i,0]},{result[i,1]}");
    }
}

int[,] TwoCoins(int[] coins, int target)
{
    int[,] result = {{-1,-1},{-1,-1},{-1,-1},{-
1,-1},{-1,-1}};
    int count = 0;

    for (int curr = 0; curr < coins.Length;
curr++)
    {
        for (int next = curr + 1; next <
coins.Length; next++)
        {
            if (coins[curr] + coins[next] ==
target)
            {
                result[count, 0] = curr;
                result[count, 1] = next;
                count++;
            }
            if (count == result.GetLength(0))
            {
                return result;
            }
```

```
        }
    }
    return (count == 0) ? new int[0,0] : result;
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق **TestProject** ثم حدد **Open in Integrated Terminal**

٤. في موجه الأوامر Terminal أدخل **dotnet run**

٥. تحقق من أن التعليمات البرمجية تنتج الإخراج التالي:

```
Change found at positions:
0,3
0,4
1,3
1,4
3,6
```

إذا كانت التعليمات البرمجية تعرض نتائج مختلفة، فستحتاج إلى مراجعتها للعثور على الخطأ وإجراء التعديلات، شغل التعليمات البرمجية مرة أخرى لمعرفة ما إذا كنت قد أصلحت المشكلة، تابع تعديل التعليمات حتى تنتج النتائج المتوقعة.

٦. بعد ذلك، قم بتحديث قيمة `target` إلى قيمة 80

```
int target = 80;
```

٧. احفظ عملك ثم أدخل dotnet run في موجه أوامر Terminal

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

```
No two coins make change
```

٧ تمرين أكمل التحدي - لإضافة أساليب لجعل اللعبة صالحة للعب

تعزز تحديات التعليمات البرمجية في جميع هذه الوحدات ما تعلمته، وتساعدك على اكتساب بعض الثقة قبل المتابعة.

ينصب تركيز هذا التحدي على إنشاء أساليب صحيحة، باستخدام المعلمات parameters وأنواع الإرجاع return types

تحدي لعبة حجر النرد المصغرة

التحدي الذي تواجهه هو تصميم لعبة صغيرة، يجب أن تختار اللعبة رقمًا مستهدفًا وهو رقم عشوائي يتراوح بين واحد وخمسة، يتعين على اللاعب رمي حجر النرد ذو الستة جوانب، للفوز يجب على اللاعب الحصول على رقم أكبر من الرقم المستهدف، في نهاية كل جولة، يجب سؤال اللاعب عما إذا كان يرغب في اللعب مرة أخرى، ويجب أن تستمر اللعبة أو تنتهي وفقًا لذلك.

في هذا التحدي، يتم منحك بعض التعليمات البرمجية للبدء، يجب تحديد الأساليب التي يجب إنشاؤها، والمعلمات الخاصة بها، وأنواع إرجاعها.

تحدي التعليمات البرمجية: إضافة أساليب لجعل اللعبة قابلة للعب

في التعليمات البرمجية الأولية، هناك أسلوبين غير متوفرين، تمت الإشارة إليهما:

`ShouldPlay` يجب أن يسترد هذا الأسلوب إدخال المستخدم ويحدد ما إذا كان المستخدم يريد اللعب مرة أخرى.

`WinOrLose` يجب أن يحدد هذا الأسلوب ما إذا كان اللاعب قد فاز أو خسر.

هناك أيضًا متغيران غير مهيأين:

`target` رقم الهدف العشوائي بين 1 و 5

`roll` نتيجة لفة النرد العشوائية ذات الجوانب الستة

يتمثل التحدي الذي تواجهه في إنشاء الأسلوبين `ShouldPlay` و `WinOrLose` وإنشاء أساليب تقوم بتعيين `roll` و `target` إلى قيم عشوائية في النطاق الصحيح، عند اكتمال جميع الأساليب، يجب تشغيل اللعبة بنجاح.

١. تأكد من أن لديك ملف Program.cs فارغاً، مفتوحاً 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 Visual Studio Code EXPLORER حدد Program.cs
- في قائمة Visual Studio Code Selection حدد Select All ثم اضغط على مفتاح Delete

٢. انسخ التعليمات البرمجية التالية وألصقها في المحرر:

```
Random random = new Random();

Console.WriteLine("Would you like to play?
(Y/N)");
if (ShouldPlay())
{
    PlayGame();
}

void PlayGame()
{
    var play = true;

    while (play)
```

```
    {
        var target;
        var roll;

        Console.WriteLine($"Roll a number
greater than {target} to win!");
        Console.WriteLine($"You rolled a
{roll}");
        Console.WriteLine(WinOrLose());
        Console.WriteLine("\nPlay again?
(Y/N)");

        play = ShouldPlay();
    }
}
```

٣. حدث التعليمات البرمجية لاستخدام أساليب لجعل اللعبة تعمل وفقاً لمواصفات التحدي.

استخدم ما تعلمته حول قيم الإرجاع، والمعلمات، لإكمال التحديث.

٤. تحقق من أن اللعبة تعمل.

٥. يجب أن تنتج تعليماتك البرمجية إخراجاً مشابهاً للآتي:

```
Would you like to play? (Y/N)
Y
Roll a number greater than 1 to win!
You rolled a 2
You win!

Play again? (Y/N)
Y
Roll a number greater than 4 to win!
You rolled a 6
```

```
You win!

Play again? (Y/N)
Y
Roll a number greater than 1 to win!
You rolled a 1
You lose!

Play again? (Y/N)
N
```

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

٨ راجع حل تحدي إضافة أساليب لجعل اللعبة صالحة للعب

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

```
Random random = new Random();

Console.WriteLine("Would you like to play?
(Y/N)");
if (ShouldPlay())
{
    PlayGame();
}

bool ShouldPlay()
{
    string response = Console.ReadLine();
    return response.ToLower().Equals("y");
}

void PlayGame()
{
    var play = true;

    while (play) {
        var target = GetTarget();
        var roll = RollDice();

        Console.WriteLine($"Roll a number
greater than {target} to win!");
        Console.WriteLine($"You rolled a
{roll}");
```

```
        Console.WriteLine(WinOrLose(roll,
target));
        Console.WriteLine("\nPlay again?
(Y/N)");

        play = ShouldPlay();
    }
}

int GetTarget()
{
    return random.Next(1, 6);
}

int RollDice()
{
    return random.Next(1, 7);
}

string WinOrLose(int roll, int target)
{
    if (roll > target)
    {
        return "You win!";
    }
    return "You lose!";
}
```

هذه التعليمة البرمجية هي مجرد "حل واحد ممكن" لأنك ربما أضفت موجزات الأسطر في نقاط مختلفة، أو أرجعت القيم بشكل مختلف، أو قمت بتنسيق التعليمات البرمجية بشكل مختلف.

بغض النظر عن الاختلافات الطفيفة في التعليمات البرمجية، عند تشغيل التعليمات البرمجية، يجب أن تشاهد إخراجاً مشابهاً لما يلي:

```
Would you like to play? (Y/N)
Y
Roll a number greater than 2 to win!
You rolled a 1
You lose!

Play again? (Y/N)
Y
Roll a number greater than 3 to win!
You rolled a 5
You win!

Play again? (Y/N)
Y
Roll a number greater than 2 to win!
You rolled a 3
You win!

Play again? (Y/N)
N
```

إذا أكملت هذا التحدي، تهانينا! تابع لاختبار المعلومات في الدرس التالي.

هام

إذا واجهت صعوبة في إكمال هذا التحدي، ففكر في مراجعة الدروس السابقة قبل المتابعة. ستعتمد جميع الأفكار الجديدة التي نناقشها في الوحدات الأخرى على فهمك للأفكار التي تم تقديمها في هذه الوحدة.

٩ اختبر معلوماتك

١- أي من الخيارات التالية يرجع قيمة string بشكل صحيح؟

- `return "";`
- `return 'a';`
- `return 5;`

٢- أي من العبارات التالية صحيحة حول الكلمة الأساسية return؟

- تبدأ تكرارًا جديدًا للأسلوب.
- تنهي تنفيذ الأسلوب.
- يجب استخدامه دائمًا مع قيمة أو تعبير.

٣- أي من الخيارات التالية عبارة إرجاع صالحة؟

- `return void;`
- `return 5 % 2;`
- `return Console.WriteLine();`

راجع إجابتك

١ `return "";`

صحيح السلسلة الفارغة هي نتيجة سلسلة صالحة

٢ تنهي تنفيذ الأسلوب

صحيح تنهي عبارة الإرجاع return تنفيذ أسلوبها وتعيد عنصر التحكم إلى المستدعي

٣ `return 5 % 2;`

صحيح ترجع هذه العبارة قيمة التعبير 5 % 2

١٠ الملخص

كان هدفك في هذه الوحدة هو فهم كيفية تأثير الكلمة الأساسية return على تنفيذ الأسلوب، والتدرب على استخدام تعبيرات مختلفة في عبارات الإرجاع return statements

تعلمت أيضًا كيفية التقاط واستخدام القيم التي يتم إرجاعها من الأساليب، حتى أنك اكتسبت خبرة في استخدام أنواع الإرجاع أثناء ممارسة بعض الأسئلة الشائعة في مقابلات البرمجة.

الوحدة الرابعة:

مشروع إرشادي - التخطيط لزيارة حديقة الحيوانات الأليفة

أظهر قدرتك على تطوير تطبيق وحدة تحكم، يعتمد على أساليب ذات معلمات مختلفة، وقيم الإرجاع، لتنسيق زيارة لحديقة الحيوانات الأليفة.

الأهداف التعليمية

- تفاصيل تصميم الخطة باستخدام التعليمات المستعارة pseudo-code
- استخدام أساليب لأداء مهام محددة perform specific tasks
- إنشاء أساليب تقبل المعلمات الإجبارية والاختيارية require and optional parameters
- استخدام القيم التي تم إرجاعها من الأساليب values returned from methods

محتويات الوحدة: -

١- مقدمة

٢- التحضير للمشروع الإرشادي

٣- تمرين - تنظيم تعليماتك البرمجية باستخدام الأساليب

٤- تمرين - إنشاء أسلوب لتبديل مصفوفة

٥- تمرين - إنشاء أسلوب باستخدام معلمة اختيارية

٦- تمرين - إنشاء أسلوب لعرض النتائج

٧- اختبر معلوماتك

٨- الملخص

١ المقدمة

الأساليب هي مفهوم برمجة أساسي، وهي توفر إمكانية إعادة استخدام التعليمات البرمجية، وتحسين سهولة القراءة، وتقلل حجم تعليماتك البرمجية، تساعدك الأساليب على تقسيم المشاكل المعقدة إلى مهام نمطية، وتساعد على تنظيم التعليمات البرمجية، الآن بعد أن تعرفت على الأساليب، فأنت مستعد لبدء تطوير المزيد من الوظائف في برامجك!

لنفترض أنك منسق حدث لحديقة الحيوانات الأليفة، يجب عليك تنسيق الزيارات بانتظام من المدارس المختلفة، وتنظيم زيارات الحيوانات لعدة مجموعات من الطلاب، قررت كتابة تطبيق يساعدك على التخطيط للزيارات المدرسية، يقسم التطبيق الطلاب إلى مجموعات، ويعين لهم مجموعة من الحيوانات للزيارة، سيقوم الطلاب بالتناوب بين المجموعات بعد زيارة الحيوانات المخصصة لهم، قررت ترتيب الحيوانات بطريقة عشوائية بحيث تكون زيارة كل مدرسة فريدة من نوعها.

سترشدك هذه الوحدة خلال الخطوات المطلوبة لتطوير تطبيق حديقة الحيوان، ستستخدم في تعليماتك البرمجية أساليب لتخطيط زيارة لثلاث مدارس، وترتيب الحيوانات بعشوائية، وتعيين الطلاب إلى مجموعات، وعرض النتائج.

ستستخدم الأساليب التي تقبل المعلمات، وترجع القيم، وستتضمن بعض المعلمات الاختيارية أيضاً.

٢ التحضير للمشروع الإرشادي

ستستخدم Visual Studio Code لتطوير تطبيق يعتمد على أساليب مختلفة لتنفيذ مهمة، تستخدم بعض الأساليب المعلمات الاختيارية والقيم المرجعة، هنا، ستجد الأهداف الإجمالية للمشروع وكيفية إنشاء تطبيقك، واختباره. ستقوم أيضاً بإعداد بيئة التطوير الخاصة بك باستخدام بعض التعليمات البرمجية ل Starter

نظرة عامة على المشروع

تقوم بتطوير تطبيق لحديقة حيوان Contoso Petting التي تنسق الزيارات المدرسية، حديقة حيوان Contoso Petting هي موطن ل ١٨ نوعًا مختلفًا من الحيوانات، في حديقة الحيوان، يتم تقسيم الطلاب الزائرين لمجموعات، وكل مجموعة تزور مجموعة من الحيوانات المخصصة لها، بعد زيارة مجموعة حيوانات، يقوم الطلاب بتغيير المجموعات حتى يتمكنوا من رؤية جميع الحيوانات في الحديقة.

بشكل افتراضي، يتم تقسيم الطلاب إلى ست مجموعات، ومع ذلك، هناك بعض الفصول التي تحتوي على عدد صغير أو كبير من الطلاب، لذلك يجب تعديل عدد المجموعات وفقاً لذلك، وينبغي أيضاً أن يتم تعيين الحيوانات عشوائياً لكل مجموعة طلاب، وذلك للحفاظ على تجربة فريدة من نوعها.

مواصفات تصميم تطبيق Contoso Petting Zoo كما يلي:

- **توجد حالياً ثلاث مدارس زائرة**
  - لدى المدرسة **أ** ست مجموعات زائرة (الرقم الافتراضي)
  - المدرسة **ب** لديها ثلاث مجموعات زائرة
  - لدى المدرسة **ج** مجموعتان زائرتان
- **لكل مدرسة زائرة، قم بتنفيذ المهام التالية:**
  - عشوائية الحيوانات
  - تعيين الحيوانات إلى العدد الصحيح من المجموعات

- طباعة اسم المدرسة
- طباعة مجموعات الحيوانات

الإعداد

استخدم الخطوات التالية للتحضير لتمارين المشروع الإرشادي.

إعداد بيئة الترميز

٦. فتح Visual Studio Code

يمكنك استخدام القائمة Windows (أو مورد مكافئ لنظام تشغيل آخر) لفتح Visual Studio Code

٧. في قائمة ملف **File** حدد فتح مجلد **Open Folder**

٨. 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.

٩. في **مربع الحوار فتح مجلد**، حدد **تحديد مجلد**.

إذا رأيت مربع حوار أمان يسألك عما إذا كنت تثق بالمؤلفين، فحدد **نعم**.

١٠. 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.

١٠. أغلق Terminal

١١. انسخ التعليمات البرمجية التالية وألصقها في محرر Visual Studio Code تمثل هذه القيم الأنواع المختلفة في حديقة الحيوانات الأليفة.

```
using System;

string[] pettingZoo =
{
    "alpacas", "capybaras", "chickens",
"ducks", "emus", "geese",
    "goats", "iguanas", "kangaroos", "lemurs",
"llamas", "macaws",
    "ostriches", "pigs", "ponies", "rabbits",
"sheep", "tortoises",
};
```

أنت الآن جاهز لبدء تمارين المشروع الإرشادي. حظ سعيد!

٣ تمرين ـ تنظيم تعليماتك البرمجية باستخدام الأساليب

في هذه الخطوة الأولى من عملية التطوير الخاصة بك، ستبدأ في تخطيط تطبيقك من خلال التعليمات البرمجية المستعارة/الزائفة، ستحدد الخطوات والأساليب المطلوبة لإكمال المهمة، تحتاج أيضاً إلى النظر في إدخال وإخراج هذه الأساليب أيضاً، يعد إعداد الأساليب عبر التعليمات البرمجية المستعارة خطوة مفيدة في أي مهمة ترميز، ويمكن أن يساعدك على تطوير تعليمة برمجية خالية من الأخطاء بسرعة.

هام

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

تحويل المهام إلى تعليمات برمجية مستعارة

في هذه المهمة، ستقوم بتحويل الخطوات المحددة في مواصفات التطبيق إلى تعليمات برمجية مستعارة، من خلال إعداد التعليمات المستعارة، يصبح تطوير التطبيق الكامل مهمة أسهل، لنبدأ!

١. خذ لحظة للنظر في مواصفات التصميم:

```
- There will be three visiting schools
    - School A has six visiting groups (the
default number)
    - School B has three visiting groups
    - School C has two visiting groups

- For each visiting school, perform the
following tasks
    - Randomize the animals
    - Assign the animals to the correct number of groups
    - Print the school name
    - Print the animal groups
```

للبدء، ستركز على كتابة التعليمات البرمجية المستعارة لمدرسة واحدة فقط، عند معرفة التعليمات البرمجية لمدرسة واحدة، يمكنك بسهولة تطبيق نفس المنطق على الآخرين.

٢. أدخل سطر فارغ، أسفل تهيئة المصفوفة pettingZoo

٣. المهمة الأولى هي عشوائية الحيوانات. إنشاء أسلوب تعليمات مستعارة، لتقسيم حيوانات حديقة الحيوان بشكل عشوائي، عن طريق إدخال التعليمات التالية في المحرر:

```
// RandomizeAnimals();
```

لن تملأ أي منطق لتقسيم الحيوانات عشوائياً حتى الآن، في الوقت الحالي، ستركز على تخطيط التطبيق ككل، يمكنك افتراض أن الأسلوب `RandomizeAnimals()` موجود، وسيقوم بترتيب مصفوفة pettingZoo عشوائيًا بشكل صحيح.

٤. بعد ذلك، قم بإنشاء أسلوب تعليمات مستعارة، لتعيين مجموعات الحيوانات، أدخل التعليمات التالية في المحرر:

```
// AssignGroup();
```

كما كان الحال من قبل، يمكنك افتراض أن هذا الأسلوب يعيين الحيوانات بشكل صحيح إلى مجموعة فرعية، والمتابعة إلى المهمة التالية.

٥. يمكنك بسهولة طباعة اسم المدرسة باستخدام Console.Write أدخل التعليمات التالية في المحرر:

```
Console.WriteLine("School A");
```

٦. وأخيرا، ستحتاج إلى طباعة مجموعات الحيوانات، أدخل التعليمات التالية في المحرر:

```
// PrintGroup(group);
```

لاحظ أنك أضفت معلمة إلى هذا الأسلوب، أنت تعرف أنك تحتاج إلى طباعة مجموعات مختلفة من الحيوانات عدة مرات، لذلك يمكنك استخدام أسلوب لتنفيذ هذه المهمة مع المجموعة group كمعلمة إدخال، خذ لحظة للنظر في المكان الذي يمكنك فيه تهيئة الوسيطة `group` في التعليمات المستعارة.

٧. قم بتعيين الوسيطة `group` عن طريق تحديث سطر التعليمات `AssignGroup` إلى ما يلي:

```
// var group = AssignGroup();
```

٨. نظرا لأنك أكملت التعليمات البرمجية الزائفة للمهام العامة، يمكنك أن تأخذ لحظة للنظر في نوع البيانات الذي يجب تعيينه للمتغير `group`

تريد المدرسة أ أن يكون لها حيوانات مقسمة إلى ست مجموعات، وأنت تعرف أن هناك ١٨ حيواناً، تعرف أيضاً أن الحيوانات ممثلة باستخدام string حتى تتمكن من استخدام مصفوفة ثنائية الأبعاد، تحتوي على ست مجموعات من ثلاث حيوانات لكل منها.

٩. قم بتعيين نوع بيانات `group` عن طريق تحديث سطر التعليمات `AssignGroup` إلى ما يلي:

```
// string[,] group = AssignGroup();
```

لا تضيف أي وسيطات إلى أسلوب ل فقيمة المجموعة الافتراضية هي ستة.

تلميح:

الهدف الرئيسي التعليمات الزائفة هو: شرح ما يجب أن يفعله كل سطر من البرنامج بالضبط، وبالتالي تسهيل مرحلة بناء التعليمات على المبرمج.

تحقق من عملك

في هذه المهمة، ستتحقق من أن التعليمات البرمجية الزائفة، تنفذ المهام العامة المدرجة لكل مدرسة، ستقوم أيضاً بتشغيل التعليمات البرمجية للتأكد من عدم وجود مشكلات.

١. قارن تعليماتك البرمجية بما ما يلي:

```
using System;

string[] pettingZoo =
{
    "alpacas", "capybaras", "chickens",
"ducks", "emus", "geese",
    "goats", "iguanas", "kangaroos", "lemurs",
"llamas", "macaws",
    "ostriches", "pigs", "ponies", "rabbits",
"sheep", "tortoises",
};

// RandomizeAnimals();
// string[,] group = AssignGroup();
Console.WriteLine("School A");
// PrintGroup(group);
```

٢. في موجه الأوامر Terminal أدخل dotnet run

٣. تحقق من عرض رسالة "School A"

٤ تمرين ـ إنشاء أسلوب لتبديل مصفوفة

في هذه الخطوة، ستقوم بتطوير التعليمات البرمجية لإكمال المهمة الفرعية الأولى، وهي عشوائية حيوانات الحديقة، تذكر أنك أشرت إلى أسلوب `RandomizeAnimals()` في التعليمات البرمجية الزائفة. لنبدأ!

تبديل/خلط مصفوفة

قد يبدو ترتيب عناصر المصفوفة بشكل عشوائي بمثابة مهمة شاقة، قبل أن تبدأ، توقف للحظة للتفكير في كيفية تبديل قيم متغيرين a وb

إذا قمت على الفور بتعيين a = b فسوف تفقد القيمة الأصلية لـ a وينتهي الأمر بمتغيرين يساويان b تحتاج إلى استخدام متغير مؤقت لتخزين قيمة a قبل أن تتمكن من الكتابة فوقها، دعونا نستخدم هذا المفهوم لبدء مهمتك.

١. أدخل سطر فارغ، ثم حدد الأسلوب `RandomizeAnimals` عن طريق إدخال التعليمات التالية:

```
void RandomizeAnimals()
{

}
```

بالنسبة لهذا الأسلوب، لا تحتاج إلى أي معلمات إدخال، لأنك ستستخدم المتغير العام `pettingZoo` الموجود، لا تحتاج أيضاً إلى أي معلمات إدخال لهذه المهمة، لبدء هذا الأسلوب، دعنا نكتب بعض التعليمات لتبديل بعض العناصر في المصفوفة.

٢. أدخل التعليمات التالية في الأسلوب `RandomizeAnimals()`

```
int i = 0;
int r = 1;

string temp = pettingZoo[i];
pettingZoo[i] = pettingZoo[r];
pettingZoo[r] = temp;
```

في هذه التعليمات البرمجية، أنت تستخدم مؤقتًا `r` and `i` لتمثيل بعض الفهارس العشوائية في المصفوفة، ثم تقوم بتبديل القيم بين الفهرسين بمساعدة المتغير `temp` لاحظ كيف يمنعنا استخدام `temp` من فقدان القيمة الأصلية لـ `pettingZoo[i]` بعد الكتابة فوقها، بعد ذلك، لنقم بتعيين r إلى قيمة عشوائية بدلاً من 1

٣. حدث تعليماتك إلى ما يلي:

```
Random random = new Random();

int i = 0;
int r = random.Next(pettingZoo.Length);

string temp = pettingZoo[i];
pettingZoo[i] = pettingZoo[r];
pettingZoo[r] = temp;
```

في هذه الخطوة، قمت بتهيئة كائن random لمساعدتنا في إنشاء رقم عشوائي، وقمت بتهيئة `r` إلى عدد صحيح عشوائي بين 0 وطول فهرس مصفوفة `pettingZoo` سيضمن لك هذا عدم الخروج من حدود المصفوفة، يمكنك تطبيق هذه التعليمة على المصفوفة بأكملها، لإضفاء عشوائية على كافة العناصر.

٤. عدل العناصر باستخدام حلقة for-loop عن طريق تحديث التعليمات إلى ما يلي:

```
Random random = new Random();

for (int i = 0; i < pettingZoo.Length; i++)
{
    int r = random.Next(pettingZoo.Length);

    string temp = pettingZoo[i];
    pettingZoo[i] = pettingZoo[r];
    pettingZoo[r] = temp;
}
```

الآن يمكنك التنقل عبر كل عنصر في المصفوفة، وتحديد فهرس عشوائي، واستبداله بالعنصر الحالي، ومع ذلك، إذا قمت بتشغيل هذه التعليمات البرمجية وراقبت التغييرات على `pettingZoo` فقد تلاحظ وجود مشكلة، لا يتم تبديل بعض العناصر على الإطلاق، ويتم تبديل بعض العناصر عدة مرات.

يمكنك تحسين توزيع العناصر المحددة عشوائياً، عن طريق تحديث النطاق أثناء التنقل عبر المصفوفة.

٥. حدث التعليمات إلى ما يلي:

```
int r = random.Next(i, pettingZoo.Length);
```

الآن، أثناء تكرار حلقة for يمكنك تحديث نطاق الفهرس المحدد عشوائيًا، لاستبعاد القيم الأقل من `i` وذلك لأن الفهارس ذات القيم الأقل من `i` قد تم تبديلها بالفعل في التكرارات السابقة للحلقة.

٦. قم بإلغاء التعليق على استدعاء `RandomizeAnimals()` عن طريق إزالة أحرف البداية `//`

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق من الوحدة الطرفية المتكاملة، والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
using System;

string[] pettingZoo =
{
    "alpacas", "capybaras", "chickens",
"ducks", "emus", "geese",
    "goats", "iguanas", "kangaroos", "lemurs",
"llamas", "macaws",
```

```
    "ostriches", "pigs", "ponies", "rabbits",
"sheep", "tortoises",
};

RandomizeAnimals();
// string[,] group = AssignGroup();
Console.WriteLine("School A");
// PrintGroup(group);

void RandomizeAnimals()
{
    Random random = new Random();

    for (int i = 0; i < pettingZoo.Length; i++)
    {
        int r = random.Next(i,
pettingZoo.Length);

        string temp = pettingZoo[r];
        pettingZoo[r] = pettingZoo[i];
        pettingZoo[i] = temp;
    }
}
```

٢. لاختبار الأسلوب `RandomizeAnimals` انسخ التعليمات التالية، وألصقها في المحرر، نهاية التعليمات البرمجية:

```
foreach(string animal in pettingZoo)
{
    Console.WriteLine(animal);
}
```

٣. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٤. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٥. في موجه الأوامر Terminal أدخل dotnet run

٦. تحقق من أن تعليماتك البرمجية تنتج الإخراج التالي:

٧. تحقق من عرض رسالة "School A" المتوقعة، متبوعة بقائمة عشوائية من الحيوانات، لا يجب أن تظهر الحيوانات بالترتيب الأبجدي، على سبيل المثال:

```
School A
ponies
iguanas
goats
tortoises
ducks
rabbits
lemurs
alpacas
capybaras
emus
sheep
llamas
pigs
macaws
kangaroos
geese
chickens
ostriches
```

٨. قم بإزالة الحلقة foreach التي أضفتها لاختبار الأسلوب الخاص بك.

٥ تمرين ـ إنشاء أسلوب باستخدام معلمة اختيارية

في هذه الخطوة، ستقوم بتطوير التعليمات البرمجية، لإكمال مهمة فرعية أخرى، وهي تقسيم حيوانات الحديقة إلى مجموعات، تذكر أنك أشرت إلى أسلوب `AssignGroup()` في التعليمات البرمجية المستعارة، لنبدأ!

تطوير أسلوبك

في هذه المهمة، ستكمل أسلوباً يقسم حيوانات حديقة الحيوان إلى مجموعات، ستستخدم ما تعلمته حول أنواع إرجاع الأسلوب، والمعلمات، لإنشاء أسلوبك، لنبدأ!

١. أدخل سطر فارغ، ثم حدد الأسلوب `assignGroups` عن طريق إدخال التعليمات التالية:

```
string[,] AssignGroup(int groups)
{

}
```

تذكر أنك قررت أن هذا الأسلوب سيعيد مصفوفة ثلاثية الأبعاد، يمكن أن يتغير عدد المجموعات أيضاً اعتماداً على حجم المدرسة الزائرة، ومع ذلك، هناك حجم مجموعة افتراضي من ٦ لذلك يجب أن تجعل هذه المعلمة اختيارية.

٢. قم بالتغيير `groups` إلى معلمة اختيارية، عن طريق تحديث الأسلوب إلى التعليمات التالية:

```
string[,] AssignGroup(int groups = 6)
{

}
```

تذكر، يتم تعريف معلمة اختيارية optional parameter عن طريق تعيين قيمة افتراضية لها، يجب الإعلان عن المعلمات الإجبارية Required

parameters أولاً، قبل أي معلمات اختيارية، بعد ذلك ستبدأ في إضافة منطق إلى تعليماتك البرمجية.

٣. حدث أسلوبك باستخدام التعليمات التالية:

```
string[,] AssignGroup(int groups = 6)
{
    string[,] result = new string[groups,
pettingZoo.Length/groups];

    return result;
}
```

لأن الأسلوب الخاص بك يحدد قيمة إرجاع، يمكنك بدء تعريف الأسلوب عن طريق إنشاء قيمة لإرجاعها، يسمح لك هذا بمتابعة إضافة منطق دون أخطاء وقت التحويل البرمجي.

`[groups, pettingZoo.Length/groups]` لاحظ أنك استخدمت لتعريف حجم المصفوفة، تمثل `groups` عدد مجموعات الحيوانات التي تريد إنشاؤها، يعكس `pettingZoo.Length/groups` عدد الحيوانات التي المخصصة لكل مجموعة، على سبيل المثال، لأن `pettingZoo` مصفوفة ثابتة من 18 عنصراً، فإن حجم المصفوفة ثنائية الأبعاد للمدرسة أ هو `[6,3]`

بعد ذلك، ستقوم بتعيين الحيوانات لكل مجموعة.

٤. حدث تعليماتك باستخدام حلقة for المتداخلة التالية:

```
string[,] AssignGroup(int groups = 6)
{
    string[,] result = new string[groups,
pettingZoo.Length/groups];

    for (int i = 0; i < groups; i++)
    {
        for (int j = 0; j <
result.GetLength(1); j++)
```

```
        {

        }
    }

    return result;
}
```

في هذه التعليمة البرمجية، تمر حلقة for الخارجية عبر كل مجموعة، دورات حلقة for الداخلية لعدد الحيوانات التي يجب أن تحتويها المجموعة، بعد ذلك، تحتاج إلى تعيين الحيوانات.

٥. حدث التكرار الحلقي for باستخدام التعليمات التالية:

```
int start = 0;

for (int i = 0; i < groups; i++)
{
    for (int j = 0; j < result.GetLength(1); j++)
    {
        result[i,j] = pettingZoo[start++];
    }
}
```

في هذه التعليمة البرمجية، قمت بتعريف `start` لتمثيل موضع البداية ل `pettingZoo` يسمح لك استخدام هذه العلامة بالتنقل عبر المصفوفة `pettingZoo` عنصرًا واحدًا في كل مرة، على الرغم من أن حلقات for لها تكرارات مختلفة.

الآن أسلوبك يعين حيوانات الحديقة إلى عدد معين من المجموعات، يمكنك الآن إلغاء تعليق استدعائك ل `AssignGroup`

٦. حدد موقع استدعاء أسلوب `AssignGroup()` وقم بإزالة أحرف البداية //

تذكر، بما أنك حذفت المعلمة الاختيارية، سيكون المتغير `group` مصفوفة ثنائية الأبعاد من `[6,3]`

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق من الوحدة الطرفية المتكاملة، والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
using System;

string[] pettingZoo =
{
    "alpacas", "capybaras", "chickens",
"ducks", "emus", "geese",
    "goats", "iguanas", "kangaroos", "lemurs",
"llamas", "macaws",
    "ostriches", "pigs", "ponies", "rabbits",
"sheep", "tortoises",
};

RandomizeAnimals();
string[,] group = AssignGroup();
Console.WriteLine("School A");
// PrintGroup(group);

void RandomizeAnimals()
{
    Random random = new Random();

    for (int i = 0; i < pettingZoo.Length; i++)
    {
```

```
        int r = random.Next(i,
pettingZoo.Length);

        string temp = pettingZoo[r];
        pettingZoo[r] = pettingZoo[i];
        pettingZoo[i] = temp;
    }
}

string[,] AssignGroup(int groups = 6)
{
    string[,] result = new string[groups,
pettingZoo.Length/groups];
    int start = 0;

    for (int i = 0; i < groups; i++)
    {
        for (int j = 0; j <
result.GetLength(1); j++)
        {
            result[i,j] = pettingZoo[start++];
        }
    }

    return result;
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject
انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٣. في موجه الأوامر Terminal أدخل dotnet run

٤. تحقق من عرض رسالة "School A" المتوقعة ومن عدم وجود رسائل خطأ

```
School A
```

٦ تمرين ـ إنشاء أسلوب لعرض النتائج

في هذه الخطوة، ستقوم بتعريف الأسلوب لعرض نتائج المجموعات الحيوانية، كما ستكمل مهام مجموعة الحيوانات للمدارس الزائرة الأخرى. لنبدأ!

تطوير أسلوبك

١. أدخل سطر فارغ، ثم حدد الأسلوب `PrintGroup` عن طريق إدخال التعليمات التالية:

```
void PrintGroup(string[,] group)
{

}
```

تذكر أنك قررت أن هذا الأسلوب سيأخذ مصفوفة ثنائية الأبعاد كإدخال، الآن تحتاج فقط إلى الأسلوب الخاص بك لطباعة محتويات المصفوفة ثنائية الأبعاد 2D array

٢. أضف التعليمات البرمجية التالية إلى الأسلوب `PrintGroup`

```
void PrintGroup(string[,] group)
{
    for (int i = 0; i < group.GetLength(0); i++)
    {
        for (int j = 0; j < group.GetLength(1); j++)
        {
            Console.Write($"{group[i,j]}  ");
        }
    }
}
```

هنا، يمكنك استخدام `GetLength(0)` للحصول على عدد الصفوف في المصفوفة، وعدد الأعمدة `GetLength(1)` ثم يمكنك استخدام `Console.Write` لطباعة العنصر مع بعض المسافات الإضافية بين الحيوانات، بعد ذلك، دعنا نضيف بعض النصوص المساعدة، وسطراً جديداً لفصل المجموعات.

٣. حدث الأسلوب الخاص بك مع التعليمات البرمجية التالية:

```
void PrintGroup(string[,] group)
{
    for (int i = 0; i < group.GetLength(0); i++)
    {
        Console.Write($"Group {i + 1}: ");
        for (int j = 0; j < group.GetLength(1); j++)
        {
            Console.Write($"{group[i,j]}  ");
        }
        Console.WriteLine();
    }
}
```

الآن لديك كل التعليمات البرمجية لإكمال مهمة التحضير لزيارة المدرسة.

٤. حدد موقع استدعاء الأسلوب `PrintGroup` وأزل أحرف البداية `//`

بعد ذلك، ستقوم بتشغيل تعليماتك البرمجية، ومراقبة النتائج.

٥. احفظ، ثم شغيل التعليمات البرمجية عن طريق إدخال `dotnet run tt` موجه الأوامر Terminal

٦. للتحقق من أن التعليمات البرمجية تعمل كما هو متوقع، تحقق من أن إخراج التطبيق مشابه للإخراج التالي، مع مراعاة الترتيب العشوائي للحيوانات:

```
School A
Group 1: ducks  emus  rabbits
Group 2: lemurs  chickens  geese
Group 3: iguanas  macaws  alpacas
Group 4: goats  kangaroos  pigs
Group 5: capybaras  llamas  tortoises
Group 6: sheep  ostriches  ponies
```

ضع في اعتبارك أنك بحاجة إلى تنفيذ نفس المهام لكل مدرسة، وستكون الاختلافات هي اسم المدرسة، وعدد المجموعات، هذه فرصة مثالية لاستخدام أسلوب مع تلك المعلمات.

٧. قم بإنشاء سطر فارغ، أعلى استدعاء الأسلوب `RandomizeAnimals()` ثم قم بإنشاء أسلوب جديد، عن طريق إدخال التعليمات التالية:

```
void PlanSchoolVisit(string schoolName, int
groups = 6)
{
    RandomizeAnimals();
    string[,] group = AssignGroup();
    Console.WriteLine("School A");
    PrintGroup(group);
}
```

باستخدام هذه التعليمات البرمجية، يمكنك تنفيذ نفس المهام لكل مدرسة، يمكنك أيضا تعيين `groups` إلى القيمة الافتراضية ل 6 بعد ذلك، ستقوم بتحديث الأسلوب لاستخدام قيم الإدخال.

٨. حدث الأسلوب باستخدام التعليمات البرمجية التالية:

```
void PlanSchoolVisit(string schoolName, int
groups = 6)
{
```

```
    RandomizeAnimals();
    string[,] group = AssignGroup(groups);
    Console.WriteLine(schoolName);
    PrintGroup(group);
}
```

الآن أنت مستعد لاستدعاء هذا الأسلوب لكل مدرسة من المدارس.

٩. إنشاء سطر فارغ أعلى الأسلوب `PlanSchoolVisit` ثم أدخل التعليمات التالية:

```
PlanSchoolVisit("School A");
PlanSchoolVisit("School B", 3);
PlanSchoolVisit("School C", 2);
```

تحقق من عملك

في هذه المهمة، ستقوم بتشغيل التطبيق من الوحدة الطرفية المتكاملة، والتحقق من أن التعليمات البرمجية تعمل بشكل صحيح. لنبدأ.

١. قارن التعليمات البرمجية بما يلي للتأكد من صحتها:

```
using System;

string[] pettingZoo =
{
    "alpacas", "capybaras", "chickens",
"ducks", "emus", "geese",
    "goats", "iguanas", "kangaroos", "lemurs",
"llamas", "macaws",
    "ostriches", "pigs", "ponies", "rabbits",
"sheep", "tortoises",
};

PlanSchoolVisit("School A");
```

```
PlanSchoolVisit("School B", 3);
PlanSchoolVisit("School C", 2);

void PlanSchoolVisit(string schoolName, int
groups = 6)
{
    RandomizeAnimals();
    string[,] group1 = AssignGroup(groups);
    Console.WriteLine(schoolName);
    PrintGroup(group1);
}

void RandomizeAnimals()
{
    Random random = new Random();

    for (int i = 0; i < pettingZoo.Length; i++)
    {
        int r = random.Next(i,
pettingZoo.Length);

        string temp = pettingZoo[r];
        pettingZoo[r] = pettingZoo[i];
        pettingZoo[i] = temp;
    }
}

string[,] AssignGroup(int groups = 6)
{
    string[,] result = new string[groups,
pettingZoo.Length/groups];
    int start = 0;
```

```
    for (int i = 0; i < groups; i++)
    {
        for (int j = 0; j < result.GetLength(1); j++)
        {
            result[i,j] = pettingZoo[start++];
        }
    }

    return result;
}

void PrintGroup(string[,] groups)
{
    for (int i = 0; i < groups.GetLength(0); i++)
    {
        Console.Write($"Group {i + 1}: ");
        for (int j = 0; j < groups.GetLength(1); j++)
        {
            Console.Write($"{groups[i,j]}  ");
        }
        Console.WriteLine();
    }
}
```

٢. احفظ عملك باستخدام Ctrl + S أو باستخدام قائمة Visual Studio Code File

٣. إذا لزم الأمر، افتح لوحة المحطة الطرفية المتكاملة في Visual Studio Code

في قائمة EXPLORER لفتح Terminal في موقع مجلد TestProject
انقر بزر الماوس الأيمن فوق TestProject ثم حدد Open in Integrated Terminal

٤. في موجه الأوامر Terminal أدخل dotnet run

٥. تحقق من أن إخراج تطبيقك يشبه ما يلي، مع مراعاة الترتيب العشوائي للحيوانات:

```
School A
Group 1: kangaroos lemurs pigs
Group 2: alpacas sheep chickens
Group 3: ducks geese capybaras
Group 4: ponies iguanas tortoises
Group 5: ostriches llamas rabbits
Group 6: macaws goats emus
School B
Group 1: llamas ducks ponies geese chickens goats
Group 2: iguanas capybaras macaws kangaroos rabbits sheep
Group 3: lemurs tortoises alpacas pigs emus ostriches
School C
Group 1: sheep ducks pigs macaws kangaroos ostriches rabbits goats lemurs
Group 2: iguanas capybaras chickens emus tortoises geese ponies alpacas llamas
```

يجب عليك التحقق من أن School A تحتوي على ست مجموعات، School B ثلاث مجموعات، School C مجموعتين، يجب أن يكون ترتيب الحيوانات عشوائياً لكل مدرسة.

٧ اختبر معلوماتك

١- ما الغرض من تعريف معلمة اختيارية في أسلوب؟

- للتسبب في تنفيذ أسلوب لمهمة مختلفة.
- لتحميل أسلوب بشكل زائد.
- لتبسيط المعلمات الاجبارية عندما لا تكون المعلمة مهمة للنتيجة.

٢- ما الغرض من التعليمات البرمجية المستعارة؟

- لتوفير قالب تصميم للتعليمات البرمجية المقصودة.
- لتشغيل العمليات التي يتم تنفيذها في الأساليب بأمان.
- لتطوير التعليمات البرمجية الصحيحة من الناحية التركيبية.

راجع إجابتك

١ لتبسيط المعلمات الاجبارية عندما لا تكون المعلمة مهمة للنتيجة.

صحيح تمكنك المعلمات الاختيارية من تمرير وسيطات أقل، وإنجاز مهمة الأسلوب.

٢ لتوفير قالب تصميم للتعليمات البرمجية المقصودة.

صحيح تساعد التعليمات البرمجية المستعارة على سد الفجوة بين الفكرة والتعليمات.

٨ الملخص

كان هدفك هو إنشاء تطبيق يستخدم أساليب لتحقيق مهمة معقدة، لقد قسمت المشكلة إلى أساليب، لإنشاء حل منظم وسهل القراءة.

كما استخدمت المعلمات parameters وأنواع الإرجاع return types لإكمال تعليماتك البرمجية.

الوحدة الخامسة

مشروع التحدي - إنشاء لعبة مصغرة

أظهر قدرتك على إنشاء واستخدام أساليب مختلفة، لتطوير ميزات لعبة مصغرة، لوحدة تحكم.

الأهداف التعليمية

- استخدم Visual Studio Code لتطوير تطبيق وحدة تحكم C# يستخدم أساليب لتنفيذ مهام سير العمل المنطقية.
- فهم التعليمات البرمجية الموجودة وإجراء تغييرات مستنيرة للتصميم.
- إنشاء قيم إرجاع بالإضافة إلى المعلمات الاجبارية والاختيارية في الأساليب.

محتويات الوحدة: -

١- مقدمة

٢- الاستعداد للتحدي

٣- تمرين - إضافة تعليمات برمجية لإنهاء اللعبة

٤- تمرين - جعل اللاعب يستهلك الطعام

٥- تمرين - إضافة تعليمات برمجية لتعديل الحركة

٦- اختبر معلوماتك

٧- الملخص

١ المقدمة

إنشاء لعبتك الخاصة وسيلة مثيرة لممارسة مهاراتك في البرمجة، تعتمد الألعاب بشكل كبير على معالجة مدخلات المستخدم لاتخاذ قرارات ديناميكية/تفاعلية يجب أن تحتوي كل لعبة أيضاً على مجموعة من القواعد المحددة التي تحدد الإجراءات والأحداث في اللعبة.

لنفترض أنك تريد إنشاء لعبتك الخاصة، قد لا تكون مستعدًا لتطوير لعبة كاملة الميزات، لذا تقرر أن تبدأ بأصغر حجم ممكن، تريد تحريك شخصية عبر الشاشة وجعلها تستهلك كائنًا، يمكن أن يؤثر استهلاك الطعام على حالة اللاعب، للاستمرار في اللعبة، أردت إعادة عرض الطعام في موقع جديد بمجرد استهلاكه، قررت أنك ستحتاج إلى استخدام أساليب للحفاظ على تنظيم التعليمات البرمجية للعبتك.

في هذه الوحدة، ستقوم بتطوير الميزات التالية لتطبيق اللعبة المصغرة:

- ميزة لتحديد ما إذا كان اللاعب قد استهلك الطعام player consumed the food
- ميزة تحدث حالة اللاعب player status حسب الطعام المستهلك
- ميزة توقف سرعة الحركة مؤقتًا pauses movement speed اعتمادًا على الطعام المستهلك
- ميزة عرض الطعام في موقع جديد regenerate food in a new location
- خيار لإنهاء اللعبة إذا تم الضغط على مفتاح غير مدعوم unsupported character
- ميزة لإنهاء اللعبة إذا تم تغيير حجم النافذة الطرفية Terminal window was resized

في نهاية هذه الوحدة، ستقوم بإنشاء تطبيق لعبة mini-game قابل للعب!

٢ الاستعداد للتحدي

سوف تستخدم Visual Studio Code لتطوير لعبة صغيرة mini-game يجب أن يحدد تطبيقك أساسيات اللعبة the basics of the game بما في ذلك تحديث حالة اللاعب updating player state ومعالجة حركة اللاعب، واستهلاك وإعادة عرض الطعام، ستقوم بتطوير كل من هذه الميزات، وتشغيل اختبار لعبة مبسط.

مواصفات المشروع

يتضمن مشروع التعليمات البرمجية الأولية Starter لهذه الوحدة ملف Program.cs مع ميزات التعليمات البرمجية التالية:

- تعلن التعليمات عن المتغيرات التالية:

- متغيرات لتحديد حجم نافذة المحطة الطرفية.
- متغيرات لتتبع مواقع اللاعب والطعام.
- مصفوفات `states, foods` لتوفير مظهر اللاعب والطعام المتاح
- متغيرات لتتبع اللاعب الحالي وظهور الطعام

- توفر التعليمات الأساليب التالية:

- أسلوب لتحديد ما إذا كان قد تم تغيير حجم نافذة الوحدة الطرفية.
- أسلوب لعرض مظهر الطعام العشوائي في مكان عشوائي.
- أسلوب تغيير مظهر اللاعب ليتناسب مع الطعام المستهلك.
- أسلوب تجميد حركة اللاعب مؤقتا.
- أسلوب تحرك اللاعب حسب إدخال الاتجاه.
- أسلوب تقوم بإعداد حالة اللعبة الأولية.

- التعليمات لا تستدعي الأساليب بشكل صحيح لجعل اللعبة قابلة للعب.
الميزات التالية مفقودة:

- تعليمات لتحديد ما إذا كان اللاعب قد استهلك الطعام المعروض.
- تعليمات لتحديد ما إذا كان الطعام المستهلك يجب أن يجمد حركة اللاعب.
- تعليمات لتحديد ما إذا كان الطعام المستهلك يجب أن يزيد من حركة اللاعب.
- تعليمات لزيادة سرعة الحركة.
- تعليمات لإعادة عرض الطعام بعد تناوله من قبل اللاعب.
- تعليمات لإنهاء التنفيذ في حالة إدخال مفتاح غير مدعوم.
- تعليمات لإنهاء التنفيذ إذا تم تغيير حجم الجهاز.

هدفك في هذا التحدي هو استخدام الميزات الموجودة، وإنشاء الميزات المفقودة لجعل اللعبة قابلة للعب.

الإعداد

استخدم الخطوات التالية للتحضير لتمارين مشروع التحدي:

١. نزيل الملف المضغوط الذي يحتوي على تعليمات المشروع الأولية Starter حدد الارتباط التالي: Lab Files

٢. فك ضغط ملفات التنزيل.

- على جهازك المحلي، انتقل إلى مجلد التنزيلات
- انقر بزر الماوس الأيمن فوق Challenge-project-Create-methods-in-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 سطح مكتب Windows

٤. افتح مجلد ChallengeProject الجديد في Visual Studio Code

- افتح Visual Studio Code
- في Visual Studio Code 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 وStarter

أنت الآن جاهز لبدء تمارين مشروع التحدي. حظ سعيد!

٣ تمرين ـ إضافة تعليمات برمجية لإنهاء اللعبة

هدفك هو تطوير تطبيق لعبة صغيرة، تحتاج إلى إنهاء اللعبة إذا غير المستخدم حجم نافذة وحدة التحكم، التي تعمل فيها اللعبة، كما تريد إضافة خيار لإنهاء اللعبة، إذا أدخل المستخدم أي حرف غير الاتجاهات.

المواصفات

في تمرين التحدي هذا، تحتاج إلى تحديث التعليمات البرمجية الموجودة لدعم خيار لإنهاء اللعب، إذا تم إدخال حرف غير اتجاهي، تريد أيضاً إنهاء اللعبة إذا تم تغيير حجم نافذة المحطة الطرفية، تحتاج إلى تحديد الأساليب الصحيحة لاستخدام التعليمات البرمجية.

الإنهاء عند تغيير الحجم

يجب أن تكون هذه الميزة:

- تحديد ما إذا كان قد تم تغيير حجم النافذة الطرفية قبل السماح بمواصلة اللعبة
- تنظيف وحدة التحكم وإنهاء اللعبة إذا تم تغيير حجم النافذة
- عرض الرسالة التالية قبل إنهاء البرنامج `Console was resized. Program exiting.`

إضافة إنهاء اختياري

- تعديل الأسلوب الموجود `Move` لدعم معلمة اختيارية optional parameter
- إذا تم تمكينها، يجب أن تكتشف المعلمة الاختيارية إدخال المفتاح غير الاتجاهي

- إذا تم الكشف عن إدخال غير اتجاهي، فاسمح بإنهاء اللعبة

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. تمكين المعلمة الاختيارية.

٢. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله.

ملاحظة

يمكنك إنهاء اختبار التحقق قبل إكمال جميع خطوات التحقق إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في Terminal اضغط على Ctrl-C. بعد الخروج من التطبيق أكمل عمليات التحرير التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، واحفظ تحديثاتك في ملف Program.cs ثم قم بإعادة بناء التعليمات البرمجية وتشغيلها.

٣. في موجه الأوامر Terminal قم بتغيير حجم النافذة.

٤. أدخل مفتاحا اتجاهياً.

٥. تحقق من انتهاء البرنامج بعد عرض الرسالة التالية:

```
Console was resized. Program exiting.
```

٦. شغّل التطبيق مرة أخرى.

٧. في موجه الأوامر Terminal اضغط على مفاتيح الاتجاه لتحريك اللاعب.

٨. اضغط على مفتاح غير اتجاهي.

٩. تحقق من انتهاء البرنامج.

١٠. تعطيل المعلمة الاختيارية، ثم بناء التطبيق وتشغيله.

١١. في موجه الأوامر Terminal اضغط على مفاتيح الاتجاه لتحريك اللاعب.

١٢. اضغط على مفتاح غير اتجاهي.

١٣. تحقق من استمرار البرنامج.

١٤. تغيير حجم نافذة Terminal

١٥. تحقق من انتهاء البرنامج.

بمجرد التحقق من صحة نتائج هذا التمرين، انتقل إلى التمرين التالي في هذا التحدي.

٤ تمرين ـ جعل اللاعب يستهلك الطعام

هدفك هو تطوير تطبيق لعبة صغيرة، تعرض اللعبة المصغرة الطعام الذي يمكن للاعب تناوله، تحتاج إلى اكتشاف ما إذا كان اللاعب قد استهلك الطعام بنجاح، وإذا كان الأمر كذلك، فأعد عرض الطعام، تريد أيضًا تغيير شكل/مظهر اللاعب اعتمادًا على الطعام الذي تم تناوله.

المواصفات

في تمرين التحدي هذا، تحتاج إلى إنشاء أسلوب تحدد ما إذا كان اللاعب قد استهلك الطعام الذي تم عرضه، إذا تم استهلاك الطعام، فأنت تريد تحديث مظهر اللاعب، وإعادة عرض الطعام.

تحقق مما إذا كان اللاعب قد استهلك الطعام

- إنشاء أسلوب يستخدم متغيرات المواقع الحالية للاعب والغذاء
- يجب أن يرجع الأسلوب قيمة
- بعد أن يحرك المستخدم الشخصية/اللاعب قم باستدعاء أسلوبك لتحديد ما يلي:
  - ما إذا كان يجب استخدام الأسلوب الحالي الذي يغير مظهر اللاعب أم لا
  - ما إذا كان يجب استخدام الأسلوب الموجود لإعادة عرض الطعام أم لا

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 خطوات التحقق إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في Terminal اضغط على Ctrl-C. بعد الخروج من التطبيق أكمل عمليات التحرير التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، واحفظ تحديثاتك في ملف Program.cs ثم قم بإعادة بناء التعليمات البرمجية وتشغيلها.

٢. في موجه الأوامر Terminal اضغط على مفاتيح الاتجاه لتحريك اللاعب.

٣. حرك اللاعب عبر سلسلة الطعام المعروضة.

٤. تحقق من عرض سلسلة طعام جديدة.

٥. تحقق من أن مظهر اللاعب يتغير اعتمادًا على سلسلة الطعام التي تم استهلاكها.

بمجرد التحقق من صحة نتائج هذا التمرين، انتقل إلى التمرين التالي في هذا التحدي.

٥ تمرين ـ إضافة تعليمات برمجية لتعديل الحركة

هدفك هو تطوير تطبيق لعبة صغيرة، حالياً، اللعبة المصغرة لديها بعض قدرات اللعب الأساسية! وتنتهي بشكل صحيح، وتكتشف متى يستهلك اللاعب الطعام، وتغير مظهر اللاعب، وتعرض المزيد من الطعام، الآن تريد أن يؤثر الطعام الذي يستهلكه اللاعب، على قدرة اللاعب على الحركة.

المواصفات

في تمرين التحدي هذا، تحتاج إلى إنشاء أسلوب يحدد ما إذا كان اللاعب قد استهلك الطعام الذي يؤثر على حركته، عندما يستهلك اللاعب سلسلة الطعام بقيمة ##### يتم تحديث المظهر إلى (X_X) ستضيف ميزة للكشف عما إذا كان مظهر اللاعب هو (X_X) وإذا كان الأمر كذلك، فأمنع اللاعب مؤقتًا من التحرك.

إضافة أيضاً ميزة اختيارية تكتشف إذا كان مظهر اللاعب هو (^-^) وإذا كان الأمر كذلك، فقم بزيادة أو تقليل سرعات الحركة اليمنى والأيسر بقيمة 3 بينما يكون هذا المظهر نشطاً، عندما تكون حالة اللاعب هي ('-') فأنت تريد أن تعود السرعة إلى طبيعتها، تريد جعل هذه الميزة اختيارية نظراً لأن استهلاك الطعام في هذه الحالة، يتطلب اكتشاف تصادم أكثر مما تريد تطويره في الوقت الحالي.

تحقق مما إذا كان يجب تجميد اللاعب

- إنشاء أسلوب يتحقق من ظهور اللاعب الحالي (X_X)
- يجب أن يرجع الأسلوب قيمة
- قبل السماح للمستخدم بتحريك الشخصية، استدع الأسلوب لتحديد ما يلي:

- ما إذا كان يجب استخدام الأسلوب الموجود الذي يجمد حركة الشخصية أم لا

- تأكد من تجميد الشخصية مؤقتاً فقط ولا يزال بإمكان اللاعب الانتقال بعد ذلك

إضافة خيار لزيادة سرعة اللاعب

- تعديل الأسلوب الموجود `Move` لدعم معلمة اختيارية لسرعة الحركة
- استخدم المعلمة لزيادة أو تقليل سرعة الحركة اليمنى واليسرى بمقدار 3
- إنشاء أسلوب يتحقق من ظهور اللاعب الحالي `(^-^)`
- يجب أن يرجع الأسلوب قيمة
- استدعاء أسلوبك لتحديد ما إذا كان يجب على `Move` استخدام معلمة سرعة الحركة

تحقق من عملك

للتحقق من أن التعليمات البرمجية تفي بالمتطلبات المحددة، أكمل الخطوات التالية:

١. تمكين المعلمات الاختيارية.

٢. استخدم Visual Studio Code لإنشاء تطبيقك وتشغيله.

ملاحظة

يمكنك إنهاء اختبار التحقق قبل إكمال جميع خطوات التحقق إذا رأيت نتيجة لا تفي بمتطلبات المواصفات، لفرض الخروج من البرنامج قيد التشغيل، في Terminal اضغط على Ctrl-C. بعد الخروج من التطبيق أكمل عمليات

التحرير التي تعتقد أنها ستعالج المشكلة التي تعمل عليها، واحفظ تحديثاتك في ملف Program.cs ثم قم بإعادة بناء التعليمات البرمجية وتشغيلها.

٣. في موجه الأوامر Terminal اضغط على مفاتيح الاتجاه لتحريك اللاعب.

٤. حرك اللاعب عبر سلسلة الطعام المعروضة.

٥. تحقق من عرض سلسلة طعام جديدة.

٥. تحقق من أن مظهر اللاعب يتغير اعتمادًا على سلسلة الطعام التي تم استهلاكها.

٧. تحقق من إيقاف الحركة مؤقتاً عندما يكون مظهر اللاعب (X_X)

٨. تحقق من أن الحركة اليميني واليسرى أسرع في التوجيهات الصحيحة عندما يكون مظهر اللاعب (^-^)

٩. اضغط على مفتاح غير اتجاهي لإنهاء البرنامج.

١٠. تعطيل معلمة سرعة الحركة الاختيارية وإعادة تشغيل التطبيق.

١١. تحقق من أن الحركة طبيعية عندما يكون مظهر اللاعب هو (^-^)

تهانينا إذا نجحت في هذا التحدي!

٦ اختبر معلوماتك

١- يريد مطور إنشاء أسلوب يقارن بين قيمتين، أي من الخيارات التالية هو أفضل نوع بيانات لأسلوب، لإرجاعه؟

- A double value
- لا شيء، يجب أن يكون الأسلوب باطلاً void
- A bool value

٢- يريد مطور إنشاء أسلوب يرجع قيمة في مصفوفة، أي من الخيارات التالية سيكون خياراً جيداً لتوقيع أسلوب؟

- string GetValueAtIndex(string[] array, int index)
- bool GetValueAtIndex(string[] array, int index)
- int GetValueAtIndex(int index)

راجع إجابتك

١ A bool value

صحيح يجب أن يرجع تعبير المقارنة true أو false

٢ string GetValueAtIndex(string[] array, int index)

صحيح يقبل هذا الأسلوب مصفوفة سلسلة وقيمة رقمية، ويعيد قيمة سلسلة.

٧ الملخص

كان هدفك هو إظهار القدرة على تطوير ميزات تطبيق يحقق مواصفات التصميم، كنت بحاجة إلى فك تشفير التعليمات البرمجية الموجودة للعبة، لاتخاذ قرارات مستنيرة لإجراء تعديلات، وإضافات على التعليمات البرمجية.

من خلال إنشاء أساليب جديدة، وإضافة معلمات اختيارية، واستخدام التعليمات البرمجية الموجودة، قمت ببناء لعبة مصغرة وظيفية وممتعة!

إن قدرتك على تنفيذ ميزات تطبيق اللعبة هذا استنادًا إلى مواصفات التصميم، توضح قدرتك على إنشاء أساليب واستدعاءها.

الحصول على شهادة مجانية تم التحقق منها

قامت Microsoft بشراكة مع freeCodeCamp.org لتقديم برنامج تدريب وشهادة ل C# من خلال إكمال هذه الوحدة من Microsoft Learn تكون بالفعل على بعد خطوة واحدة من أن يتم اعتمادك. لاستكشاف شهادة التأسيسية التي تقدمها freeCodeCamp تفضل بزيارة https://aka.ms/csharp-certification