تعلـــــــم البايثون (بالعربي)

للمبتدئين

By

Hadeel M.Taher

Learn Python For Beginners

مقدمـــــــــة

بسم الله الرحمن الرحيم

والحمد لله رب العالمين والصلاة والسلام على سيد المرسلين محمد (صلى الله عليه وسلم) اما بعد:

تمت كتابة تعلم البايثون(بالعربي) لمساعدة اي مبتدأ على تعلم برمجة Python بشكل جيد .

حتى وان كنت مبتدئًا بشكل مطلقً في البرمجة ، اتمنى ان تجد ما يساعدك على فهم الامور المعقدة بطريقة سلسة وبسيطة فضلا عن فهم البايثون بطريقة مختلفة .

سيكون في هذا الكتاب قاعدة جيدة من المفاهيم التي يمكنك من خلالها استكشاف البايثون. كما سنقدم الكثير من التجارب والشروحات والاخطاء التي قد تمر بك في بعض الاحيان ... سيتم الاستفادة من تجارب كثيرة وكورسات عالمية تمت ترجمتها للعربية لسهولة فهم اللغة ....

وفي النهاية انا ايضا سأتعلم معك وسأنقل تجربتي الصغيرة اليك مع عالم البايثون وكلما اتطور في هذه اللغة سأزيد من تطورك ايضا بإذن الله...

إن أحسنت فمن الله، وإن أسأت أو أخطأت

فمن نفسي والشيطان

(صدقة جارية)

Hadeel M.Taher

hadeilmt@gmail.com

Python, what Python?

البايثون ؟ وماهي لغة بايثون ؟

البايثون هي لغة برمجية تستخدم على نطاق واسع جدا خلال الفترة الحالية, تم إنشاؤها من قبل (Guido van Rossum) في أواخر الثمانينيات من القرن الماضي اما معنى كلمة بايثون فهي تمثل احدى انواع الافاعي .... تعد لغة البايثون من اللغات القوية التي يمكن قراءة تعليماتها البرمجية بسهولة وبساطة مما يسهل الامر على المبرمجين من تطوير التطبيقات بسرعة . اضافة الى ذلك فان لغة البايثون قابلة للتنفيذ لبعض أشهر أنظمة التشغيل مثل Windows و Mac OS. هذا يسمح لنا بتوزيع برامج Python الخاصة بنا بطريقة مريحة بدون مطالبة المستخدمين بتثبيت مترجم Python.

تعد البايثون لغة مثالية للمبتدئين وذلك لبساطتها حيث انها تتطلب عدد أقل من الأسطر البرمجية لتنفيذ نفس المهمة مقارنةً باللغات البرمجية الاخرى.

اذا اردت التعلم هيا بنا نبدأ..

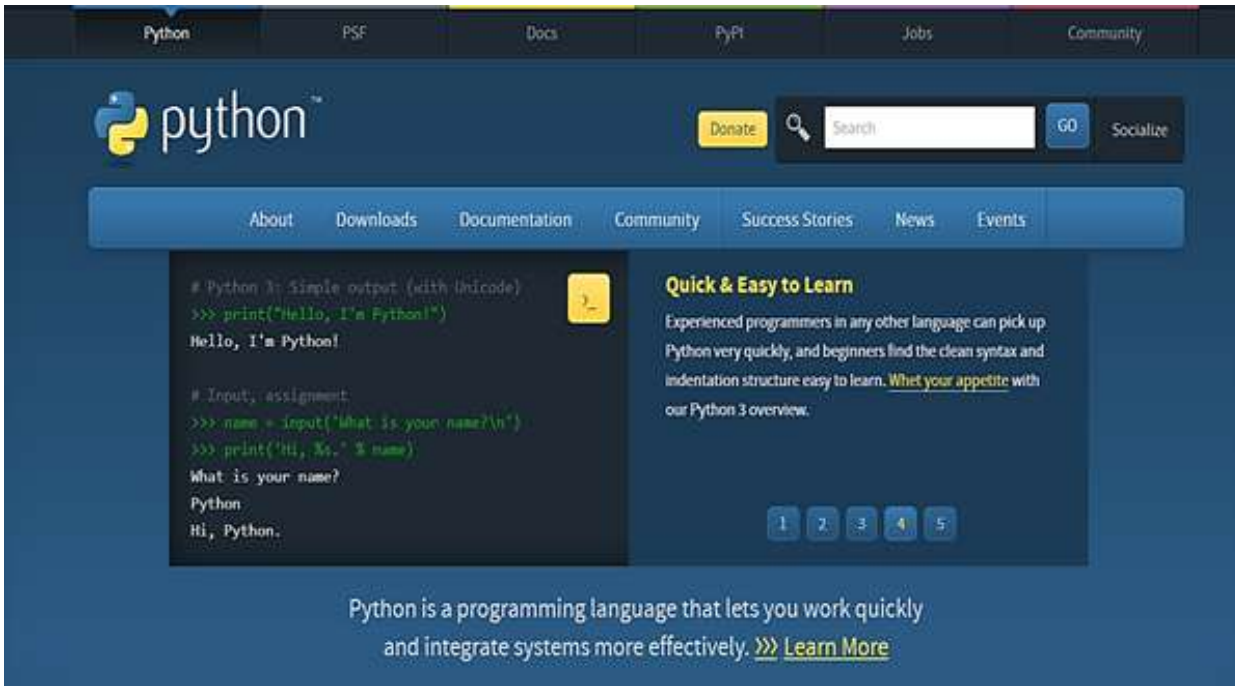
Getting ready for Python: Installing

انت مستعد للبايثون: تثبيت البايثون

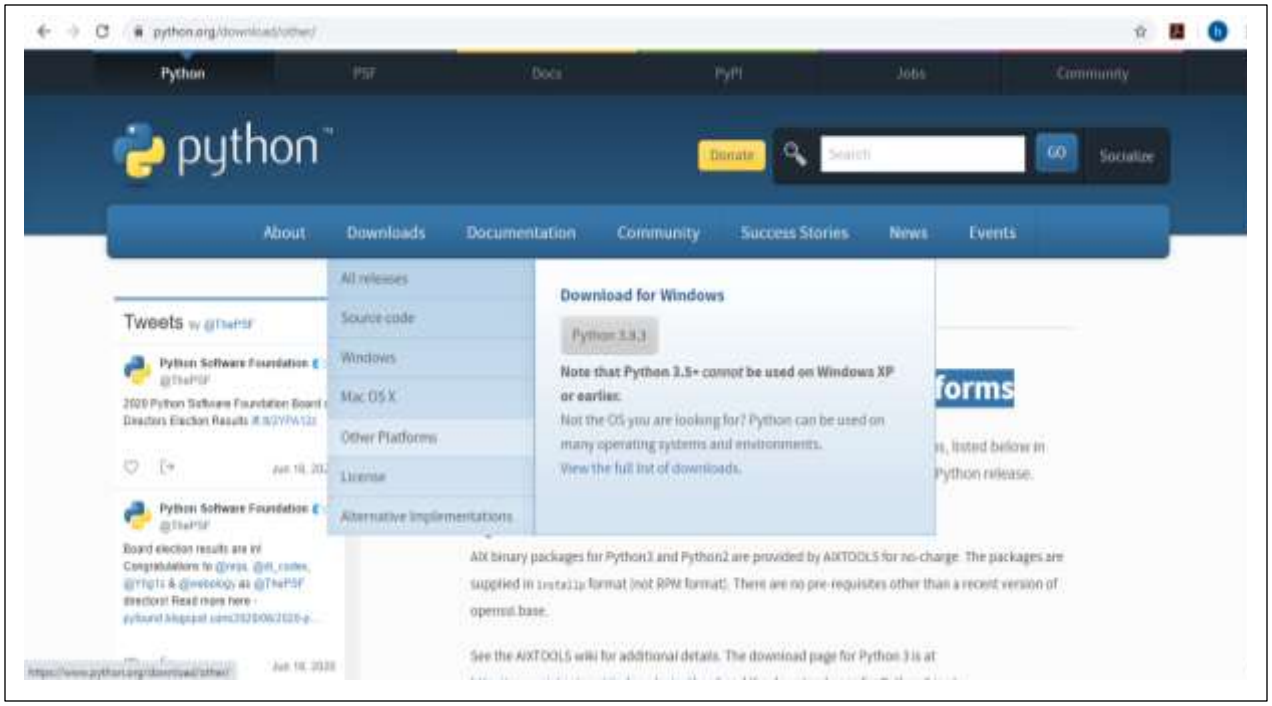
ان الطريق الامثل لتثبيت البايثون على جهازك من خلال الرابط التالي: ⇩

https://www.python.org/

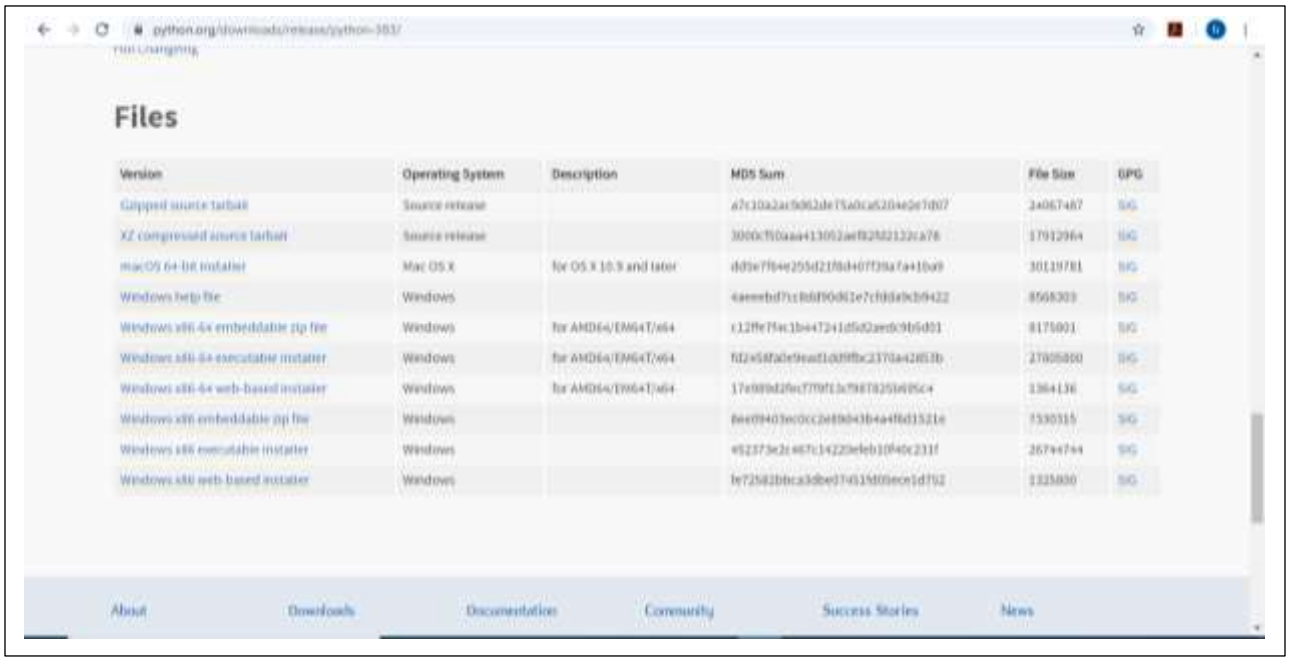
يعد الرابط اعلاه الموقع الرسمي لتنزيل احدث نسخة من البايثون والان Python 3.8.3 هي النسخة الاحدث حاليا.



حيث يمكننا اختيار البايثون المناسب على حسب نظامك التشغيلي وprocessor المستخدم سواء كان 32-bit او 64-bit فضلا عن ذلك يوجد في الموقع الرسمي تثبيت البايثون لأغلب Platforms كما في الصورة ادناه :

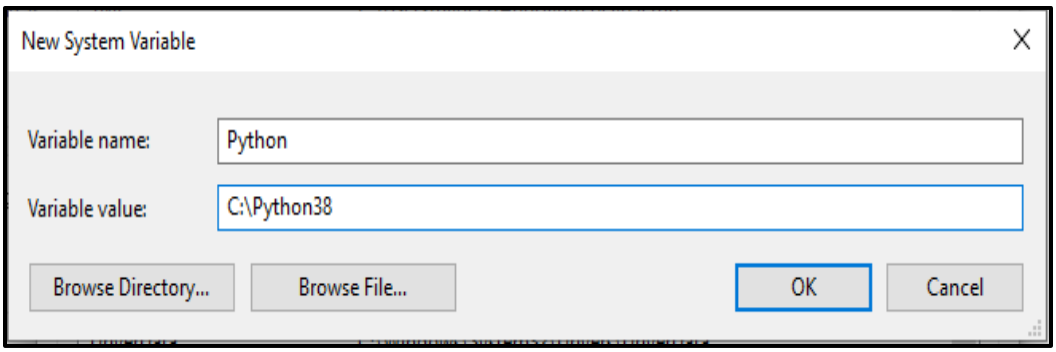


ولتثبيت بايثون لنظام ويندوز ٦٤ فقط نضغط على تنزيل(Download) نختار الwindows 64



لكن قبل البدء بكتابة البرنامج هناك بعض الملاحظات التي جمعتها من هنا وهناك للاستفادة منها :

- بعد تثبيت البايثون على الحاسوب نذهب الى Start ثم نبحث عن IDLE(python3.8) ثم نضغط open والان الواجهة الموجودة هي الواجهة التي ننفذ البرامج من خلالها
- لحفظ اي ملف Python على جهازك يجب ان يكون بصيغة (.py) وهذه الصيغة غير موجودة في نظامك لذلك ستحتاج الى اضافتها بنفسك من خلال بالضغط click يمين على ايقونة جهاز الحاسوب ثم نختار(خصائص) ثم (إعدادات النظام المتقدمة) ثم (متغيرات البيئة) بعدها نذهب الى (متغيرات النظام) وننقر على (جديد) لتظهر الواجهة التي يجب ان نملأها كالاتي:



حيث تم كتابة كلمة Python في اسم المتغير الجديد ومن ثم اضفنا الامتداد حسب نسخة البايثون فاذا كان (Python 3.8) سنكتب الامتداد (C:\Python38) واخيرا نضغط على OK ثم نكون بذلك اتممنا عملية تهيئة الويندوز لاستخدام البايثون.

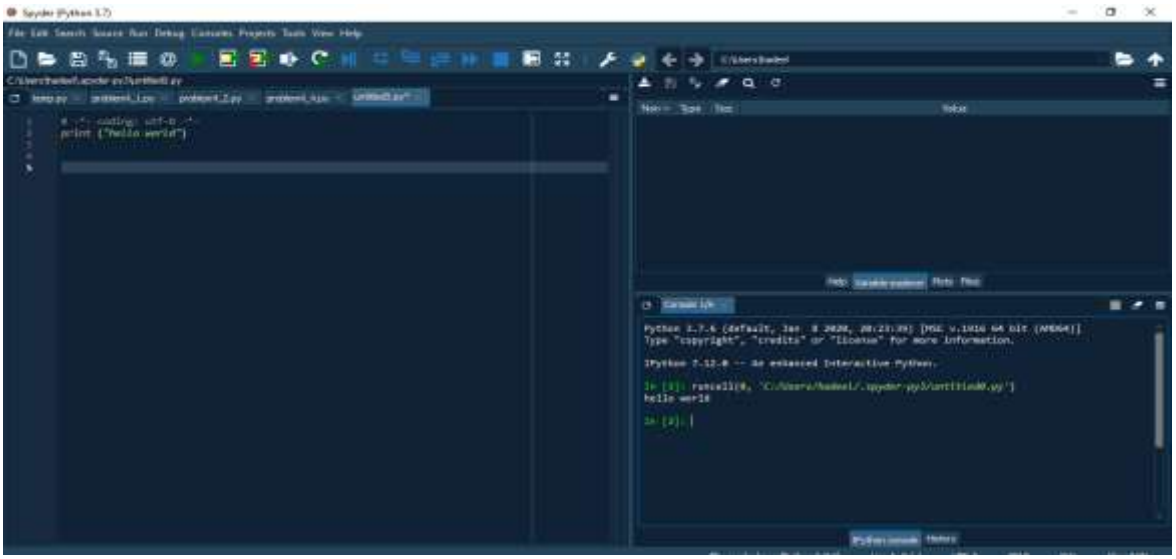
- في حال عدم حفظ الملف بالصيغة (.py) يمكننا حفظ الملف باستخدام الصيغة (.pyw)
- يمكن ايضا تنزيل برنامج pycharm من خلال الرابط التالي:

⇩

https://www.jetbrains.com/pycharm/

حيث يعد pycharm بيئة تطويرية متكاملة تستخدم في لغات برمجة الحاسوب، خاصة البرمجة بلغة بايثون. حيث يتميز بدعم وتطوير المكتبات الاساسية للبايثون مثل مكتبة Django لذلك انصح ان تجرب ال pycharm لأنه ممتع بالعمل .

- Spyder هو برنامج شهير تم اعتماده واستخدامه من قبل الجامعات العالمية حيث ان بعض الكورسات التي اطلقتها الجامعات العريقة تستخدم هذه البيئة المتكاملة مفتوحة المصدر لتعلم البايثون وذلك لأنها تتميز بضمها مجموعة عديدة من المكتبات اهمها NumPyو SciPy و Matplotlib .



يمكننا ملاحظة وجود ثلاث اقسام للشاشة في الصورة اعلاه :

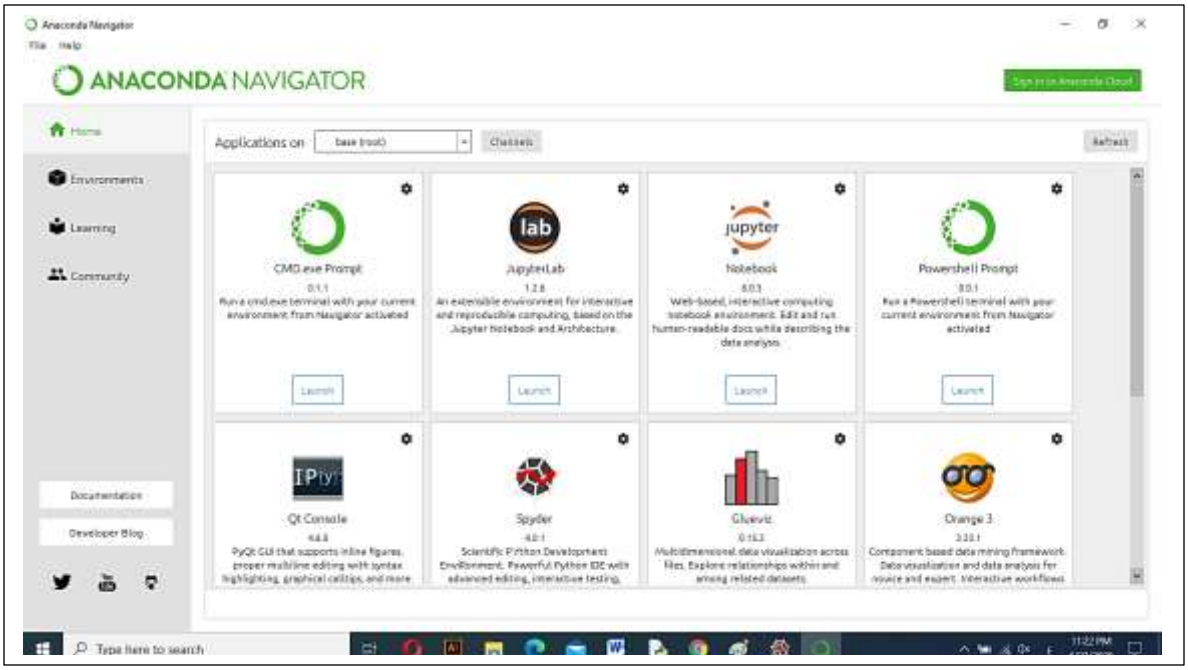
الجزء الاول : في حال كتابة برنامج (hello world) نلاحظ ان برنامج سبايدر قام بإنشاء ملف بصيغة (.py) مباشرة.

الجزء الثاني: يمكنك الحصول على المساعدة وايضا تستطيع التعرف على جميع انواع المتغيرات التي تم كتابتها في البرنامج فضلا عن امكانية استعراض كافة الملفات python الموجودة في جهازك.

الجزء الثالث : هو Console والذي يشابه تماما عمل python shell والصورة اعلاه توضح اليه العمل ببرنامج سبايدر.

- اخيرا وليس اخرا برنامج Anaconda الشهير المكتوب بلغة البايثون يعد الافضل في مجال البيانات Data Science Platform اضافة الى ذلك يعد أسهل الطُرق لتثبيت المكتبات العلمية في البايثون.

ادناه واجهة البرنامج:



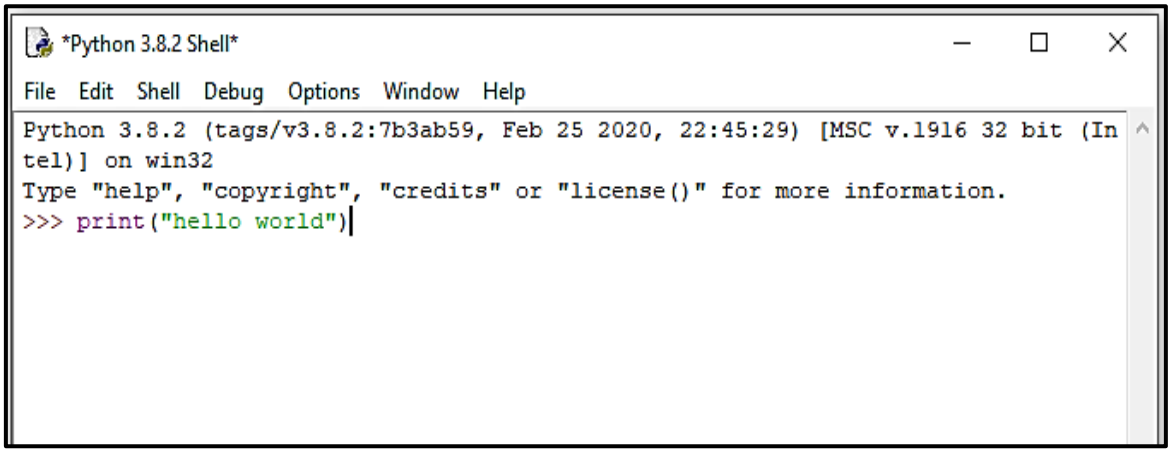
والان يمكننا كتابة اول برنامج في البايثون☺

FIRST program Using the Python Shell, IDLE and Writing our

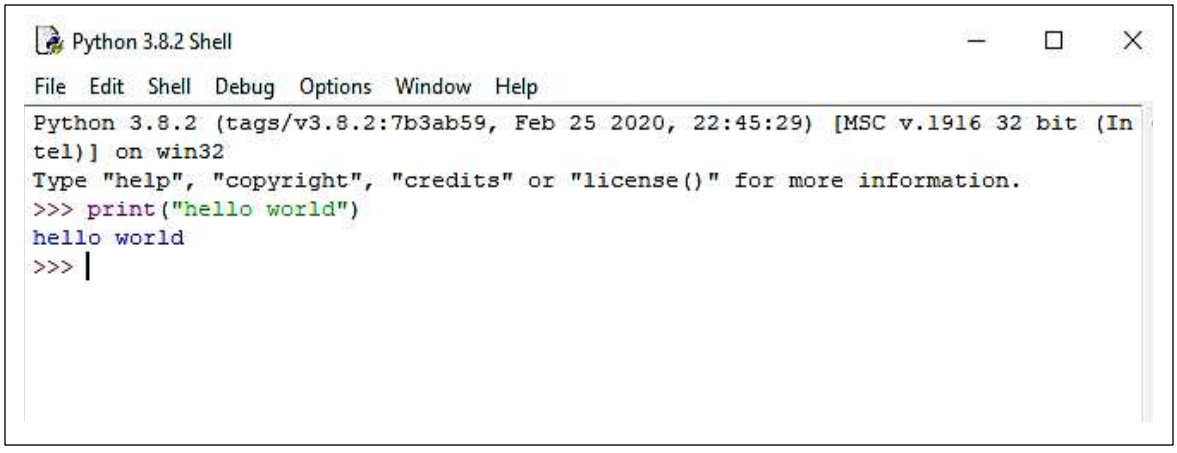
Writing the first program

لنكتب برنامجنا الاول

من قائمة Start نختار Python IDLE ستفتح لنا الواجهة ادناه بعدها تستطيع كتابة برنامج " hello world" الشهير .



وكما نشاهد في الصورة اعلاه لقد قمنا بطباعة عبارة"hello world" من خلال الدالة print بعدها بمجرد الضغط على enter ستكون النتيجة كالتالي :

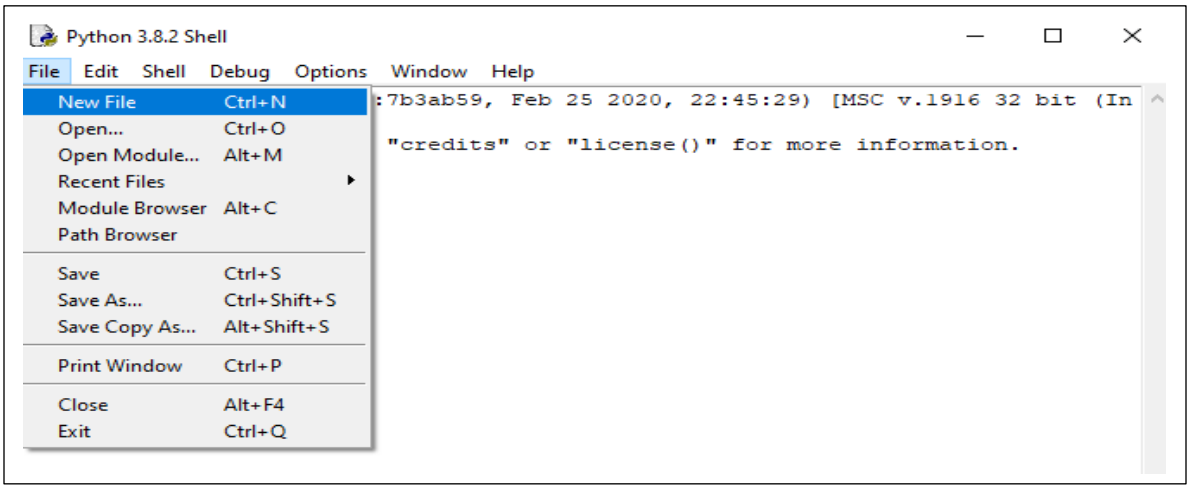


مبارك لك لقد تم تنفيذ اول برنامج بنجاح

ملاحظة عامة عن البايثون

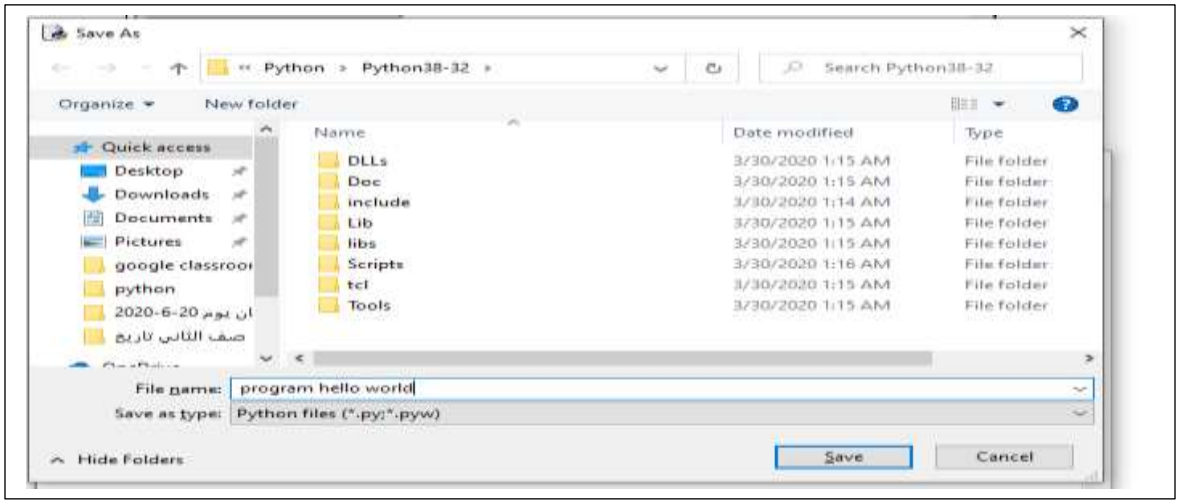
- البايثون لا تحتاج الى فارزة منقوطة في نهاية الجملة
- يجب الكتابة بعد <<< مباشرة
- البايثون يحسب عدد ال space خلال التنفيذ فيجب ان تكون دقيق باستخدامها خصوصا في for و if و statement و function لأنها قد تسبب لك الاخطاء وسيتم توضيحها لاحقا.
- لغة البايثون تعد لغة مرنة جدا

ولكتابة البرنامج وتنفيذه من خلال فتح File كما في الصورة ادناه:



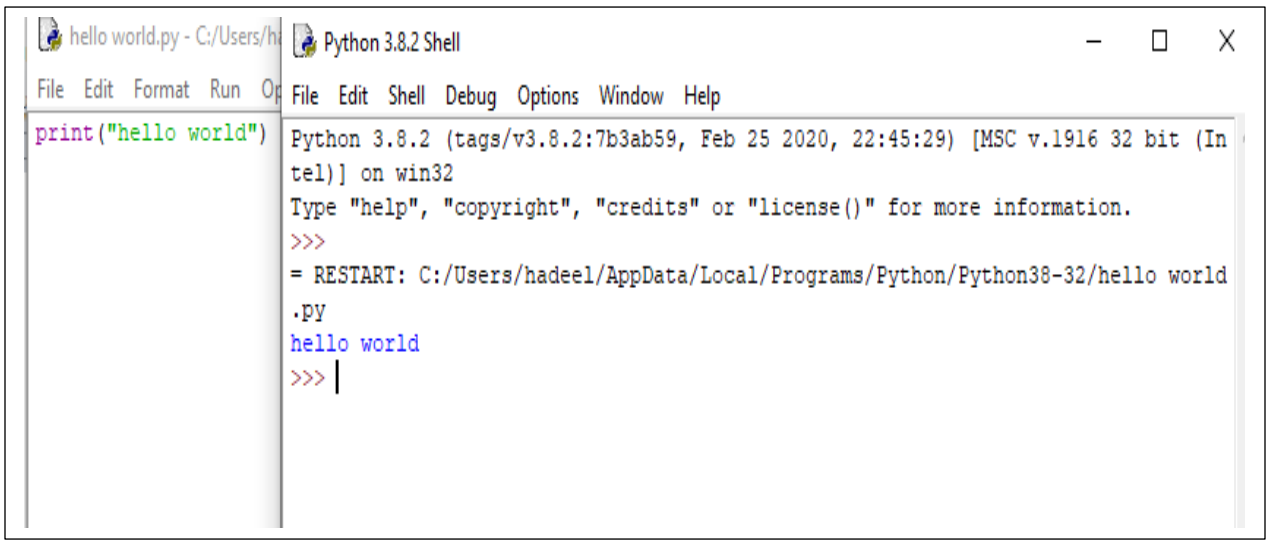
عند فتح الفايل سنكتب بداخله البرنامج بدون اي مقدمات كأقواس او main لان البايثون لا تحتاج لذلك نستطيع الطباعة مباشرة :

اما تنفيذ البرنامج سيكون من خلال كلمة run الموجودة في الاعلى او بالضغط على F5 لكن يجب حفظ الملف قبل ان عمل run له , والصورة ادناه تبين كيفية الحفظ من خلال الضغط على كلمة file ثم save as:



سيتم حفظ البرنامج بالاسم الذي تريده وبصيغة البايثون(.py)(.pyw).

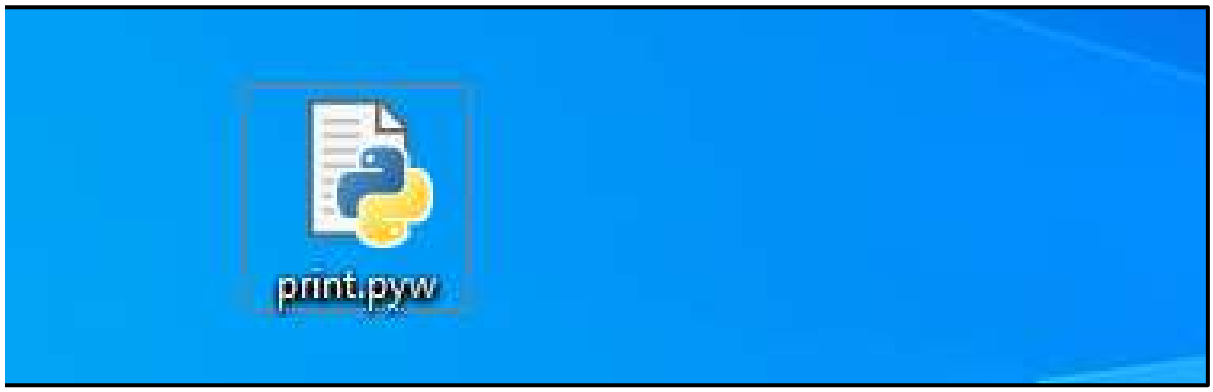
بعد ان تم حفظ البرنامج سيكون من السهولة فتحه وعمل run له والنتيجة ستكون كالتالي:



نلاحظ عند التنفيذ ظهرت python shell وتم طباعة جملة "hello world" ببساطة ....

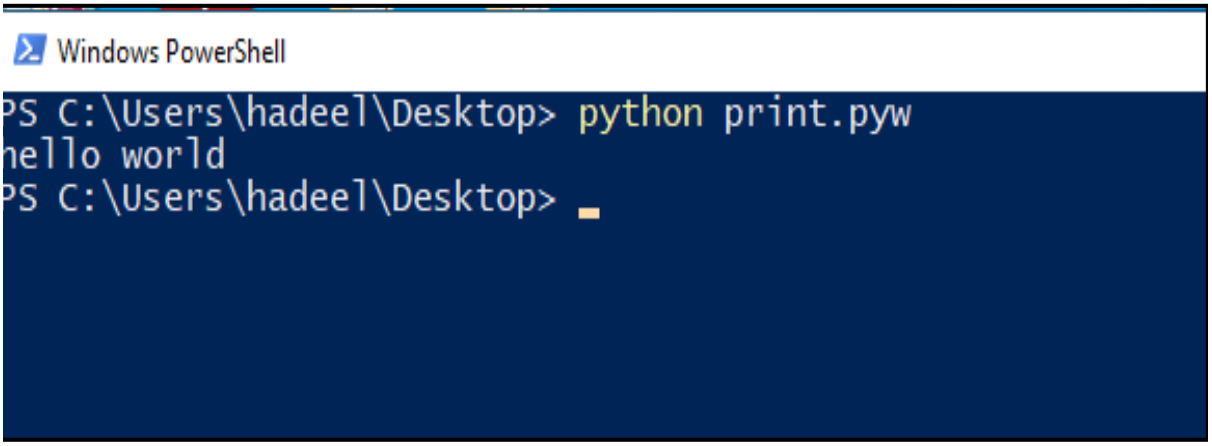
هناك طريقة اخرى لتنفيذ برنامج البايثون من خلال cmd من خلال تتبع خطوات بسيطة :

- نحفظ الملف باسم print مثلا و سيكون الملف بصيغة .py او .pyw على سطح المكتب



- نضغط على زر Shift و زر الفارة الايمن في اي مكان فارغ على سطح المكتب حتى يظهر لنا عبارة Open power shell windows here

- بعد فتح power shell سنكتب الامر التالي: python print .pyw (البايثون بعدها اسم الملف ثم الامتداد)سنلاحظ انه سيتم تنفيذ البرنامج بيسر .



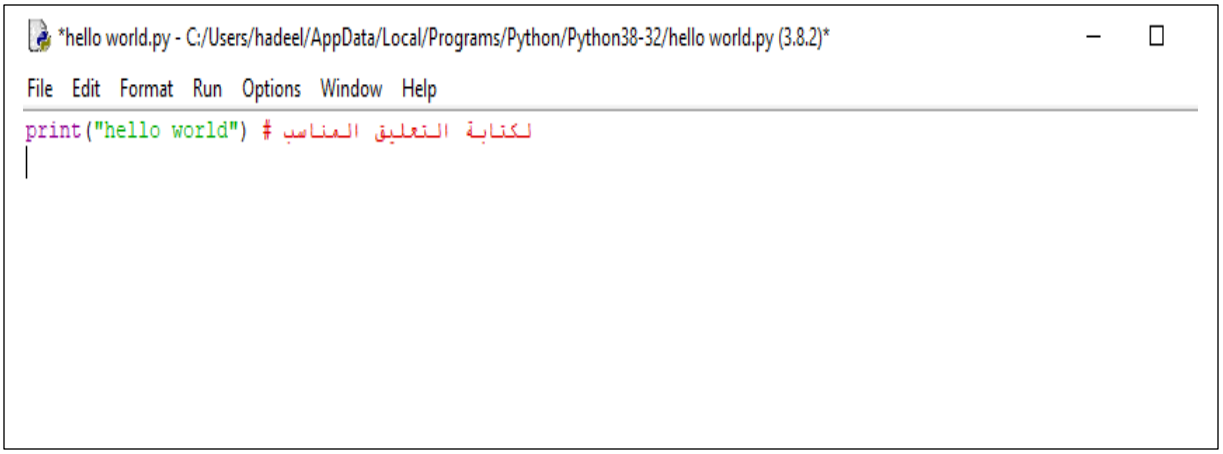
تعد الطريقة اعلاه طريقة فرعية يمكن تجربتها في حال حدوث خلل ما ...

لقد اجتزنا اول برنامج الان لنبدأ بأساسيات اللغة

Comments in Python

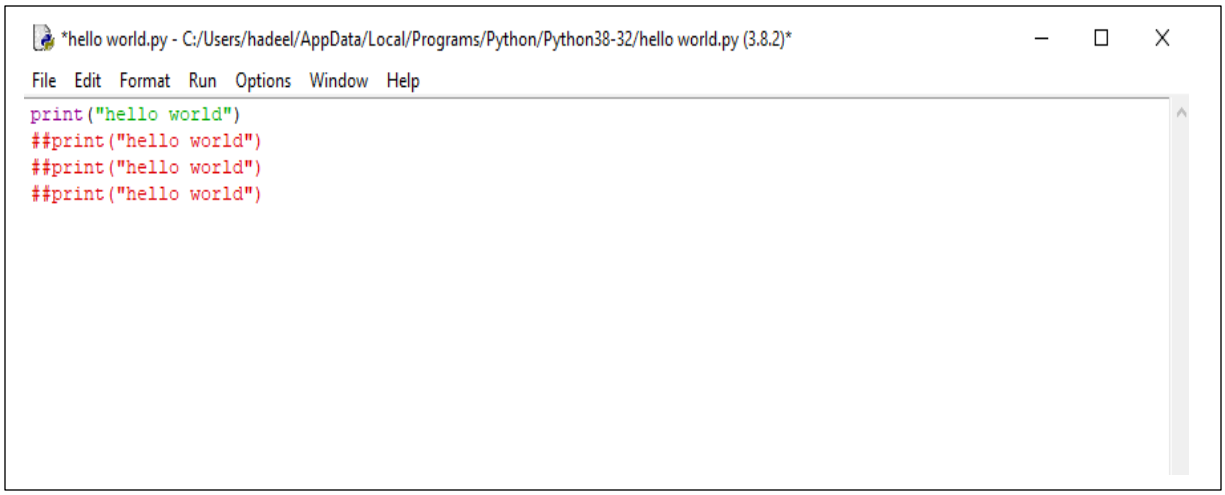
التعليقات في البايثون

تعد التعليقات مهمة جدا في اي لغة برمجية ولإضافة تعليقات إلى برنامج مكتوب بلغة البايثون نكتب علامة # أمام كل سطر من التعليق كالتالي:



نلاحظ ان التعليق اصبح باللون الاحمر فهذا يعني انه سيتم تجاهله من قبل البايثون ولكنه سيوضح الخطوات البرمجية.

اما اذا اردنا اضافة # لإخفاء عدة خطوات برمجية يمكن ذلك من خلال الضغط على Alt+3 او بالذهاب الى formatسنجد عبارة Comment Out Region يتم الضغط عليها وسيظهر لنا #في جميع الخطوات المطلوب اخفائها, كما موضح ادناه:



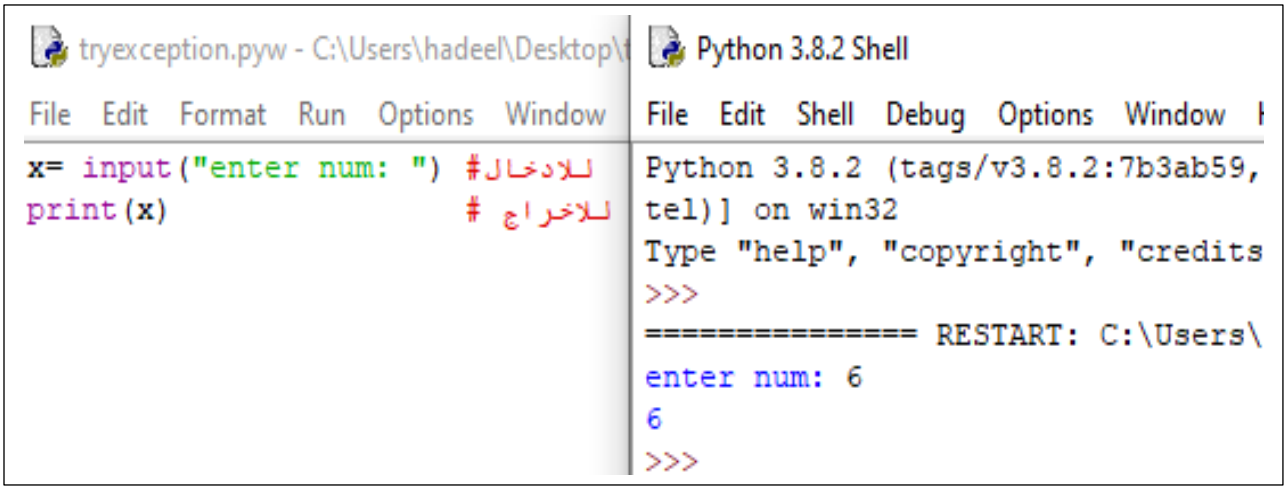
طريقة الاخرى لإضافة تعليقات(هذه الطريقة مفيدة جدا) :

في حال لدينا تعليقات متعددة او شروحات لأكثر من سطر يمكننا ايضا استخدام ثلاث علامات اقتباس حيث نطبع جملة hello world وبين علامات الاقتباس كتابة لن تظهر في التنفيذ ولن توقف البرنامج عن التنفيذ كما ادناه :

Python I/O

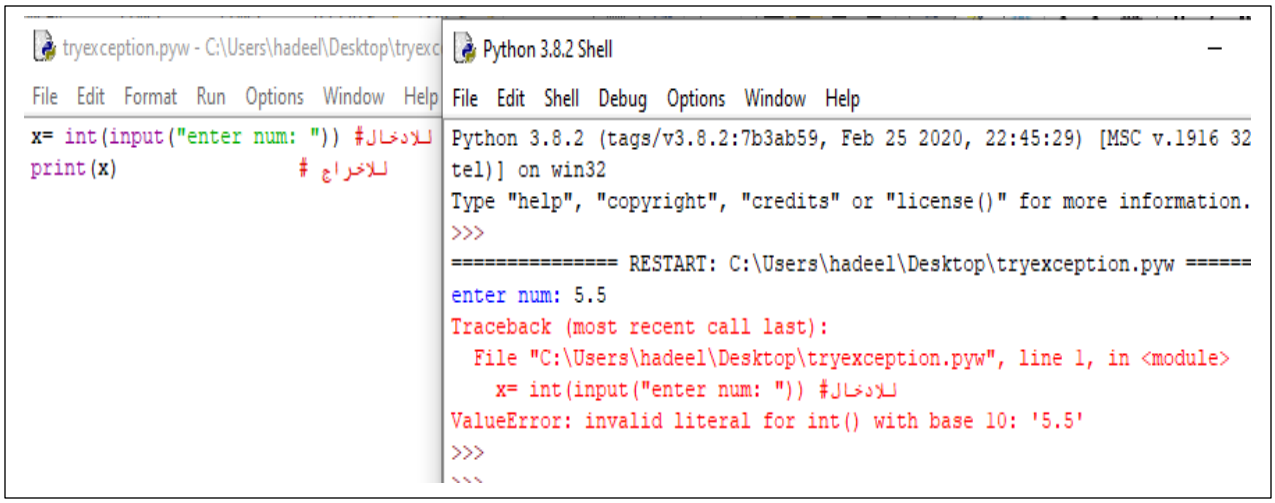
الادخال والاخراج في البايثون

في لغة البايثون الادخال و الاخراج يختلف نوعا ما عن اللغات الاخرى فمثلا في C++ نلاحظ ان الادخال عن طريق (cin>>) والاخراج عن طريق (cout<<) لكن هنا في البايثون الامر يحتاج الى دالتين فقط هما واحدة للإخراج للطباعة تسمى (print) والإدخال عن طريق (input) يعملان على أداء مهمة الإدخال / الإخراج في Python. والدالتين تكتبان بطريقة يسيرة بدون تعقيدات كالاتي:

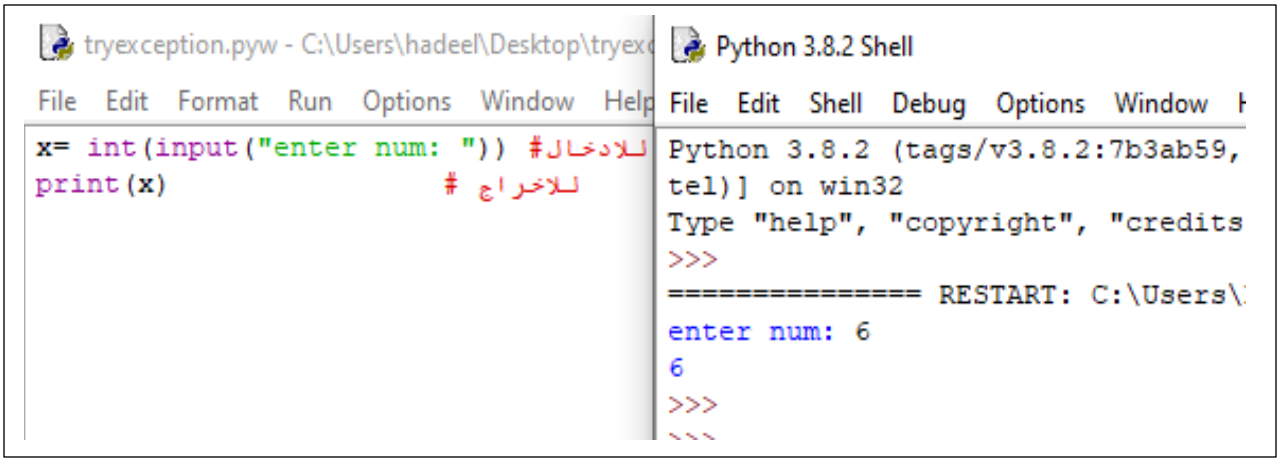


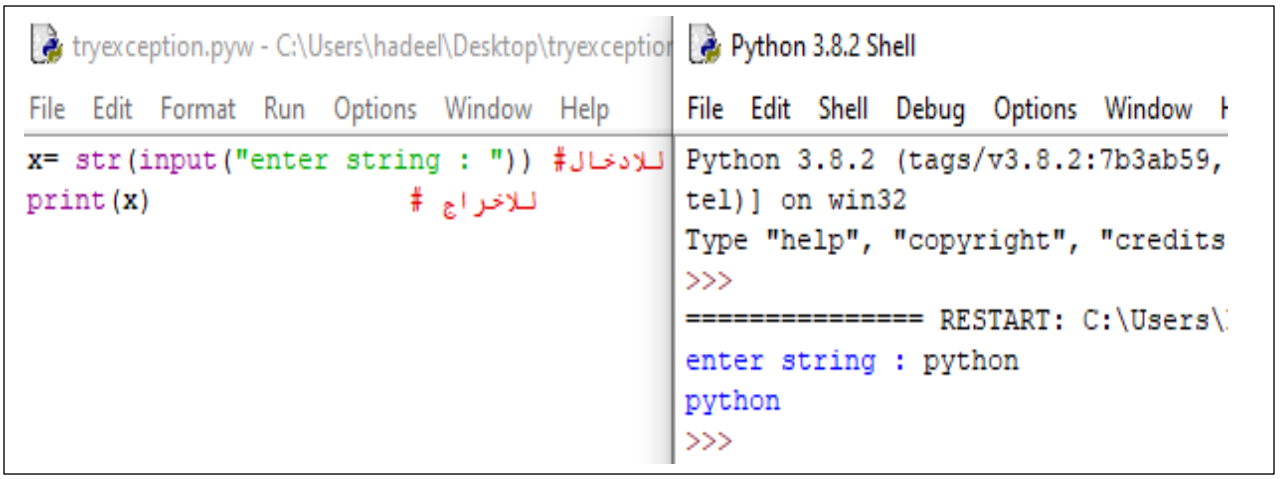
في المثال اعلاه نلاحظ ان (input) المستخدمة للإدخال من قبل المستخدم هي مرنة للغاية حيث يمكن ادخال رقم , حرف , جملة ...الخ. اما الطباعة (print) تعمل لإخراج البيانات إلى الشاشة .

من جهة اخرى يمكننا ان نحدد الادخال برقم صحيح او رقم عشري او جملة بمجرد كتابة intاو str او float قبل عبارة inputولتمثيل ذلك كالتالي:



الخطأ الظاهر في التنفيذ انه تم ادخال float فظهر الخطأ لان البرنامج لا يقبل سوى رقم int (اي تم تحديد القيمة المدخلة برقم صحيح فقط) فالتنفيذ الصحيح كالاتي:

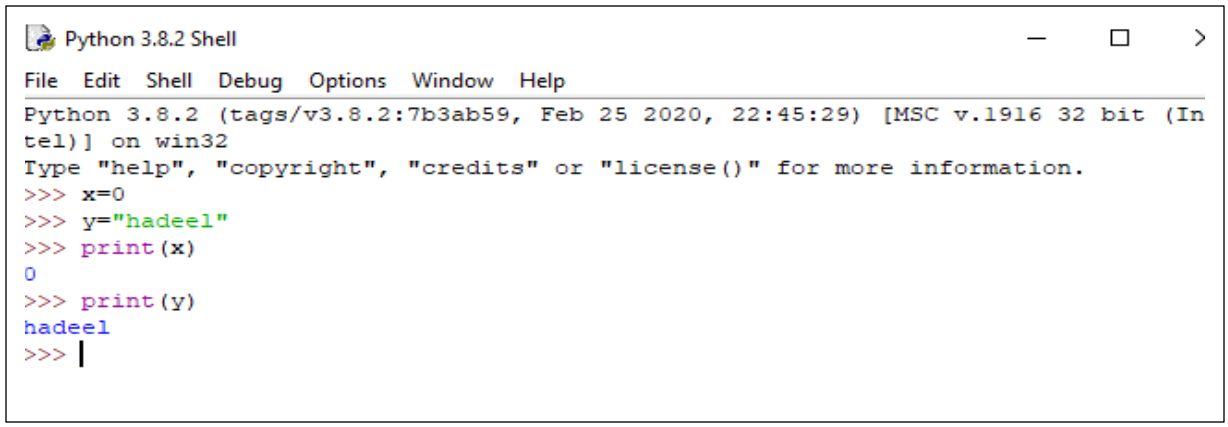




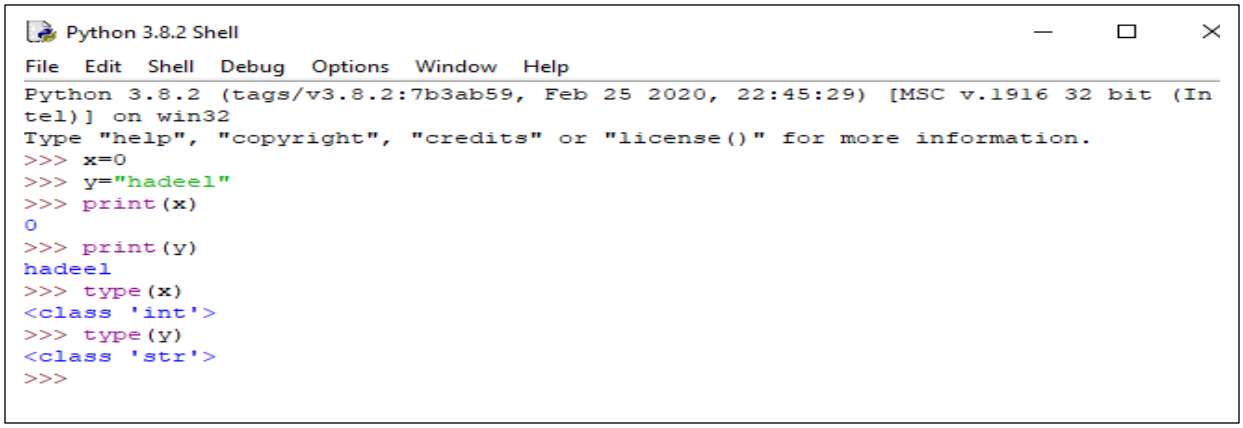
Python Variables

متغيرات البايثون

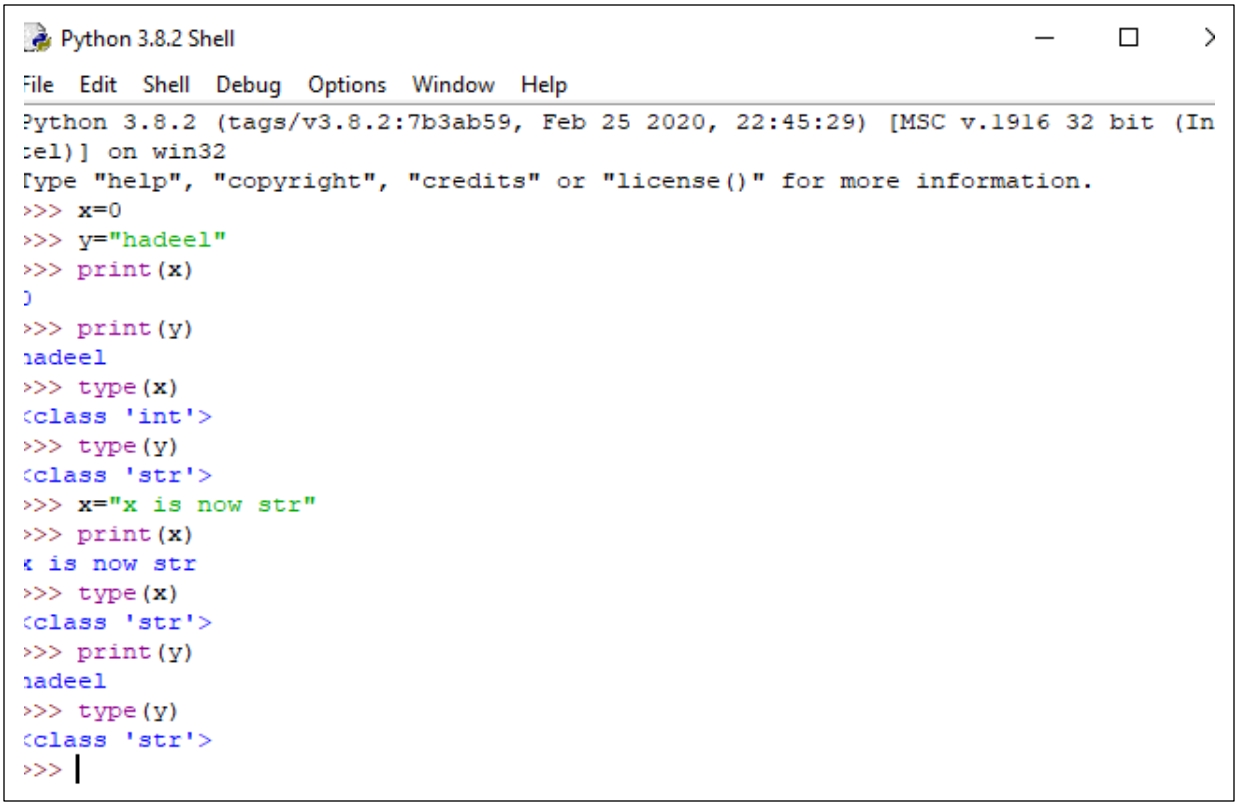
المتغيرات هي أسماء تُعطى للبيانات التي نحتاج إلى تخزينها ومعالجتها في برامجنا. على عكس لغات البرمجة الأخرى ، ليس لدى Python تعريف declaring للمتغير (تم استخدام Python Shell للسهولة والسرعة) كما سنلاحظ في الامثلة ادناه:



نلاحظ في المثال اعلاه انه تم تعريف متغيرين مختلفين لكن البايثون تعامل معهما بكل احترافية. ولنثبت ذلك سنقوم بكتابة الامر type والذي سيبين ان البايثون ميزَ بين int و str كما نلاحظ ادناه:



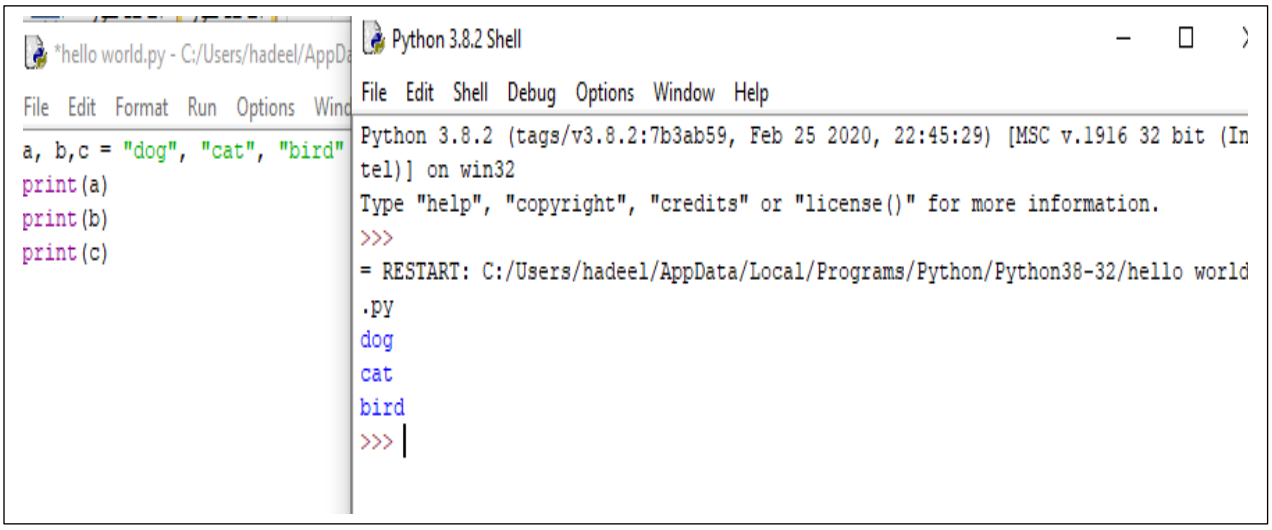
يمكن تغيير نوع المتغير في البايثون حتى بعد تعيينها كالاتي:



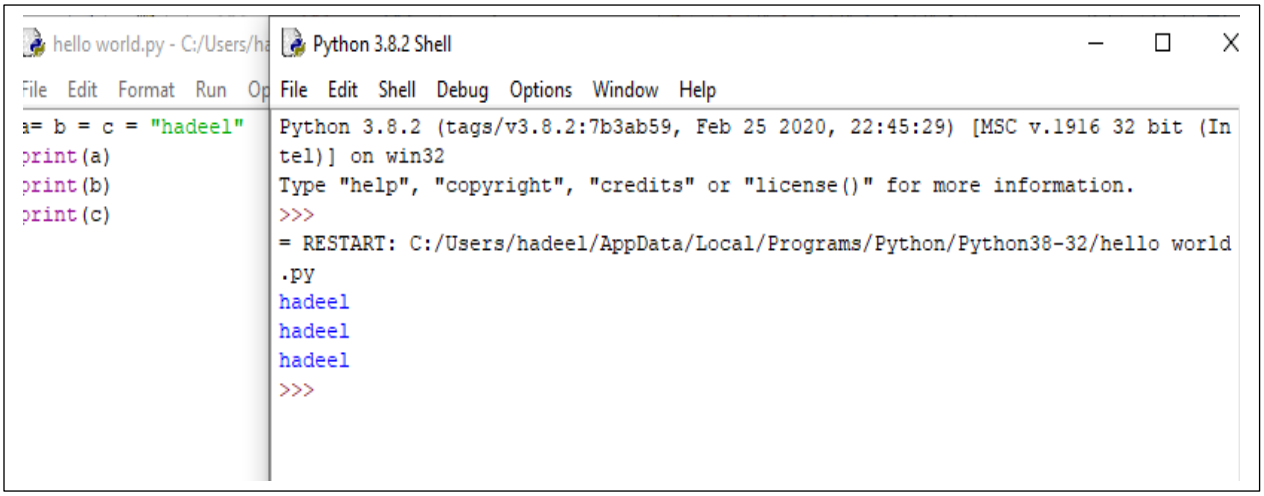
لقد تم تغيير نوع المتغير x من int الى str مع الابقاء على المتغير y كما هو , فضلا عن انه يمكن التعريف عن متغير str إما باستخدام علامات اقتباس مفردة أو مزدوجة بالحالتين ستكون النتيجة صحيحة:

```
>>> x= "hadeel"
>>> print(x)
hadeel
>>> y='hadeel'
>>> print(y)
hadeel
>>>
```

هناك ميزة في البايثون وهي يمكن تعيين قيم لمتغيرات متعددة في سطر واحد:

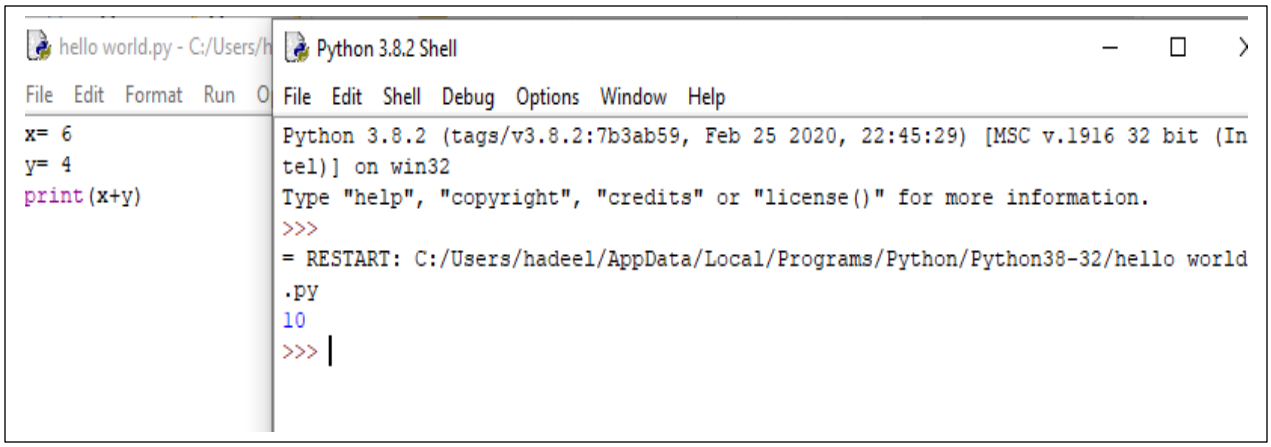


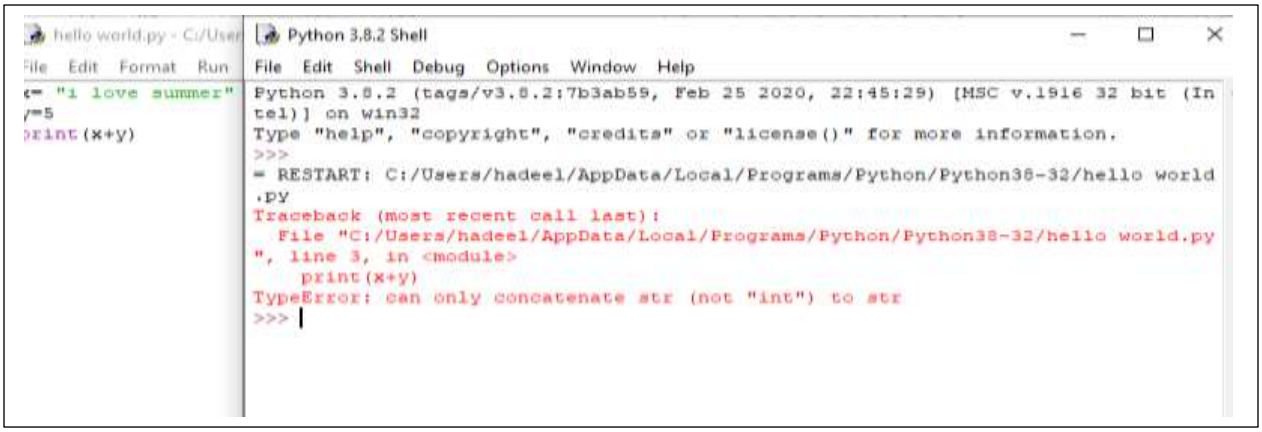
نلاحظ ان كلا من a,b,c اخذت القيم dog ,cat ,bird بالتتابع من جهة اخرى يمكنك تعيين نفس القيمة لمتغيرات متعددة في سطر واحد:



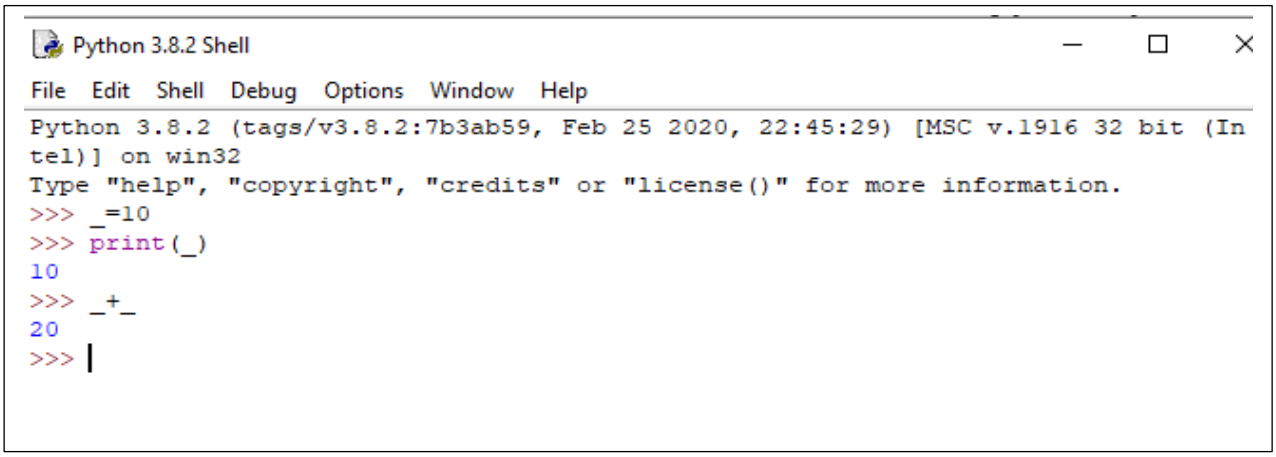
واخيرا يمكن جمع جملتين(str) معا وذلك بإضافة (+) فقط.

إلا أنه يجب الانتباه الى ان علامة (+) تأتي ايضا ك operator عند جمع عددين من نوع intو float اضافة الى ذلك لا يمكن جمع strو int سيظهر لنا error حيث انه من الخطأ الجمع بينهما ب(+) . وكما ادناه مثالين الاول جمعنا عددين int والثاني تم جمع strو int وسنبين ال error:





في البايثون يمكن ان يكون المتغير Underscore(_) ويمكن لهذا المتغير ان يحفظ قيمة وكذلك تستخدم (_ لإرجاع آخر قيمة تم تنفيذها في Python Prompt / Interpreter والمثالين ادناه يوضحان العملية:



```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=4
>>> y=6
>>> x+y
10
>>> _
10
>>> _+10
20
>>> _*4
80
>>> _/2
40.0
>>>
```

والان

بعد تعلمنا بعض الاساسيات للمتغيرات

سنأتي لأنواع المتغيرات

Data Types in Python

أنواع البيانات في بايثون

تعد البيانات وانوعها في البرمجة بشكل عام البايثون بشكل خاص مفهومًا مهمًا جدا . في البايثون تنقسم انواع البيانات الى سبعة انواع اساسية منها ما هو معروف في اللغات البرمجية الاخرى كـ (str, int , float) ومن جهة اخرى هناك ثلاث انوع بيانات اكثر تقدما في البايثون مثل(list, tuple and dictionary) حيث سنقوم بشرح كافة الانواع تباعا. في الجدول ادناه ملخص لأنواع البيانات في البايثون:

ت	المتغير	نوعه
١	نصوص (Strings)	str
٢	أرقام (Numbers)	int, float, complex
٣	منطقية (Booleans)	bool
٤	متسلسلات او مصفوفات (Sequences)	list, tuple
٥	مصفوفات ليس لها حجم ثابت, و لا يمكن حذف قيمها	Sets, frozenset
٦	القاموس Dictionaries	dict
٧	الثنائي Binary	bytes, bytearray

والان لنبدأ بشرحها بتفصيل كل نوع بشكل منفصل :

١. النصوص (Strings)

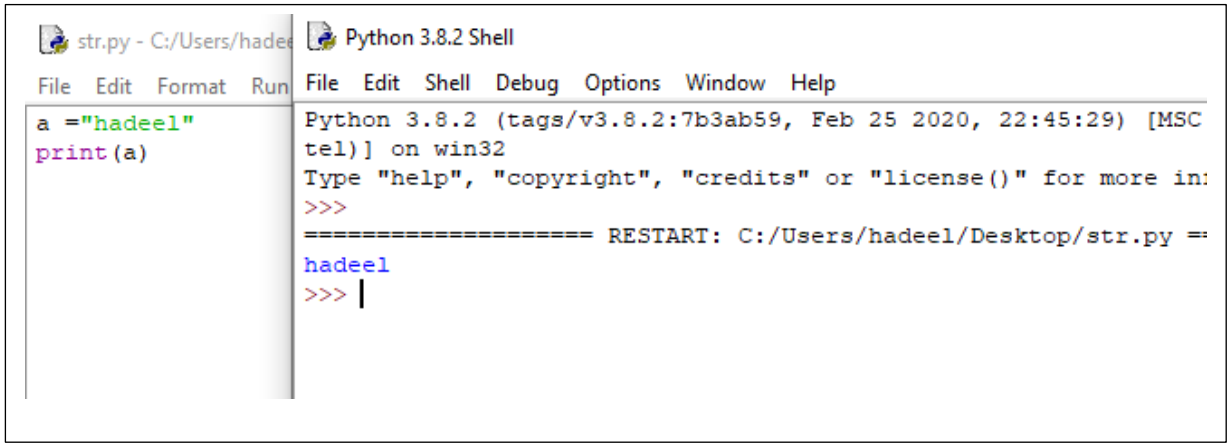
يمكن تعريف (declare)النصوص في البايثون من خلال(اسم متغير= " النص " او اسم المتغير = 'النص')
اي بوضع str ما بين علامة اقتباس واحدة single quotes او علامة اقتباس مزدوجة double quotes او ثلاث علامات اقتباس والتي تدعى ب(multiline string).
كما سنوضح بالأمثلة التالية :

```
str.py - C:/Users/hadeel/Desktop/str.py (3.8.2)
File  Edit  Format  Run  Options  Window  Help
name ='hadeel'
print("Name: ", name)
```

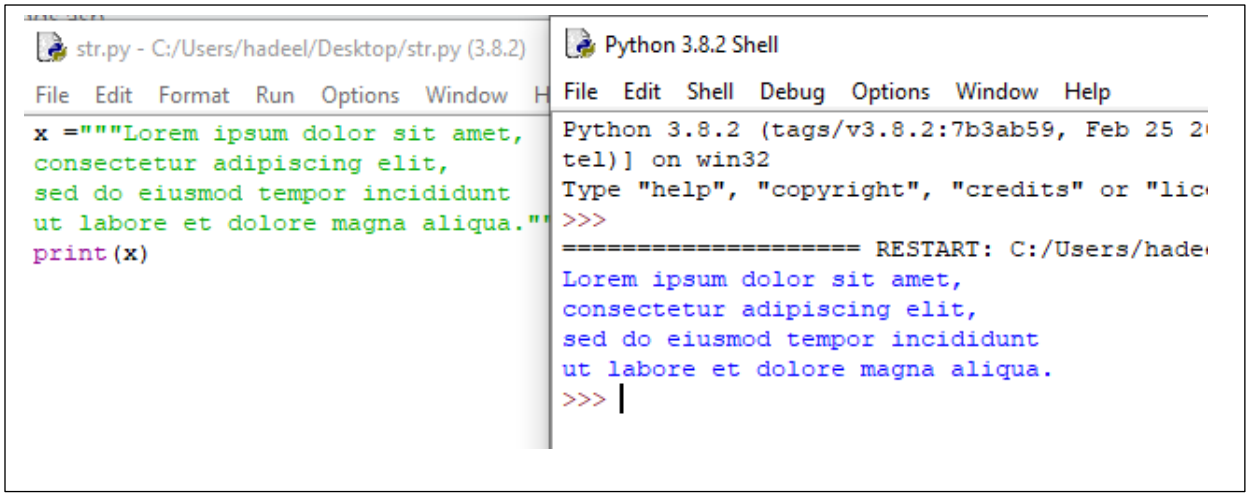
عند التنفيذ سيكون الناتج :

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/hadeel/Desktop/str.py ==========
Name:  hadeel
>>>
```

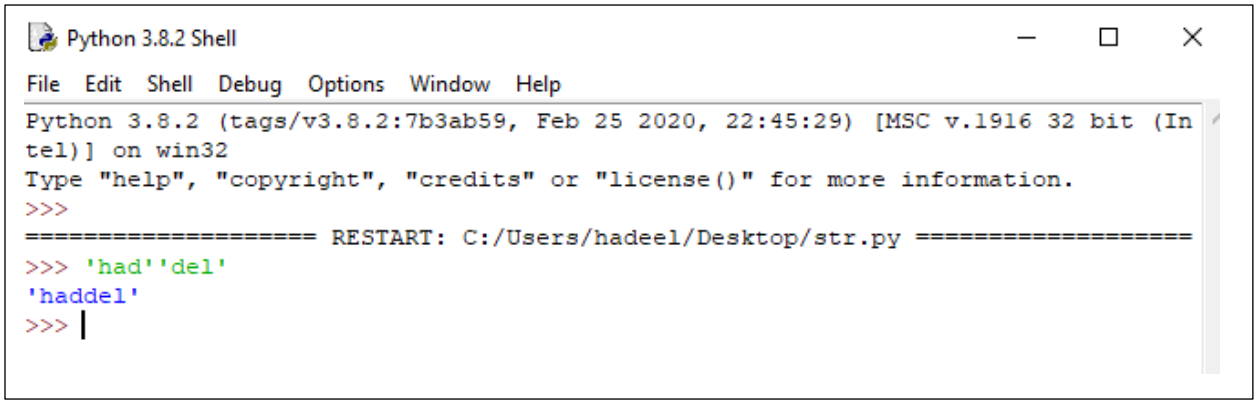
في حال علامتين اقتباس سيكون كالتالي:



واخيرا ان كانت ثلاث علامات اقتباس سيكون البرنامج في البايثون كما ادناه:



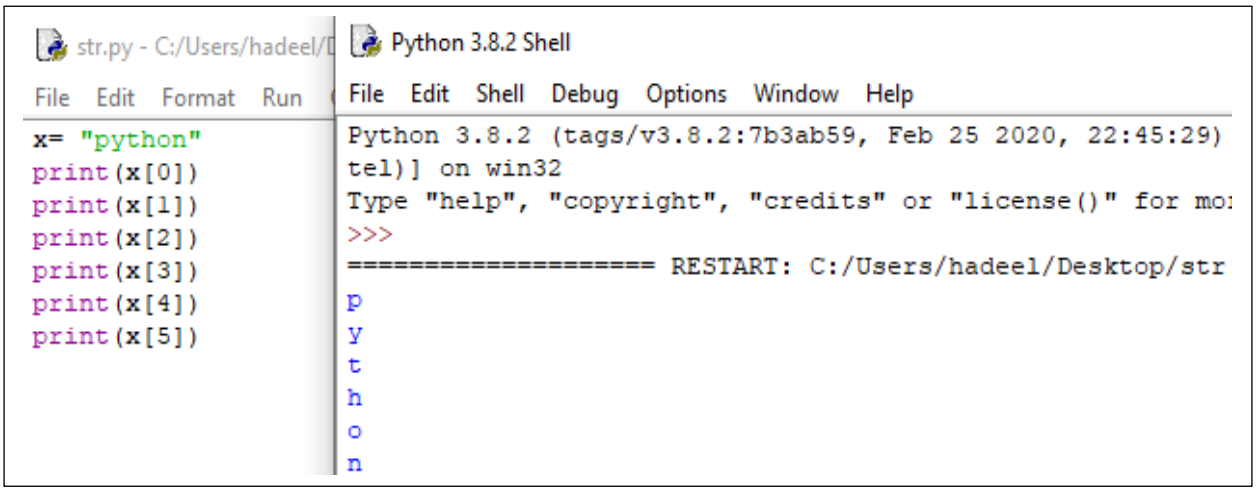
تربط لغة البايثون النصوص المتجاورة مع بعضها البعض تلقائيًا فمثلا لدينا جملتين تمت كتابتهما بالسطر نفسة سيعمل البايثون على دمجهما معا في النتيجة وسيكون شكل الجملة النهائية كالتالي:



فضلا عن فهرسة النصوص من خلال indexing حيث نلاحظ مثلا ان في كلمة python الحرف الاول p يأخذ index[0] والحرف y يأخذ index[1] ....الخ. كما في التطبيق التالي:

String	p	y	t	h	o	n
Index[]	0	1	2	3	4	5

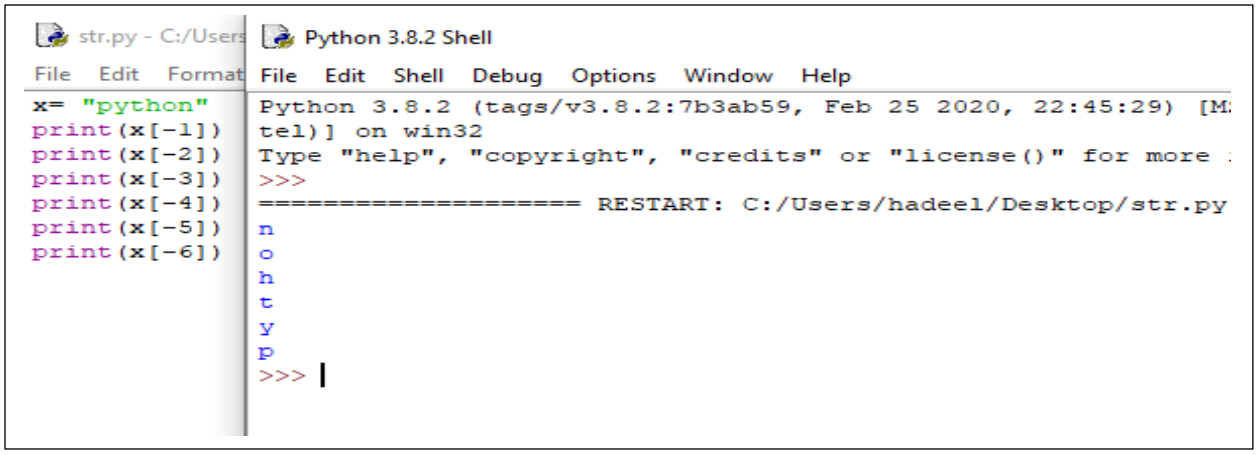
الفهرسة من اليسار الى اليمين



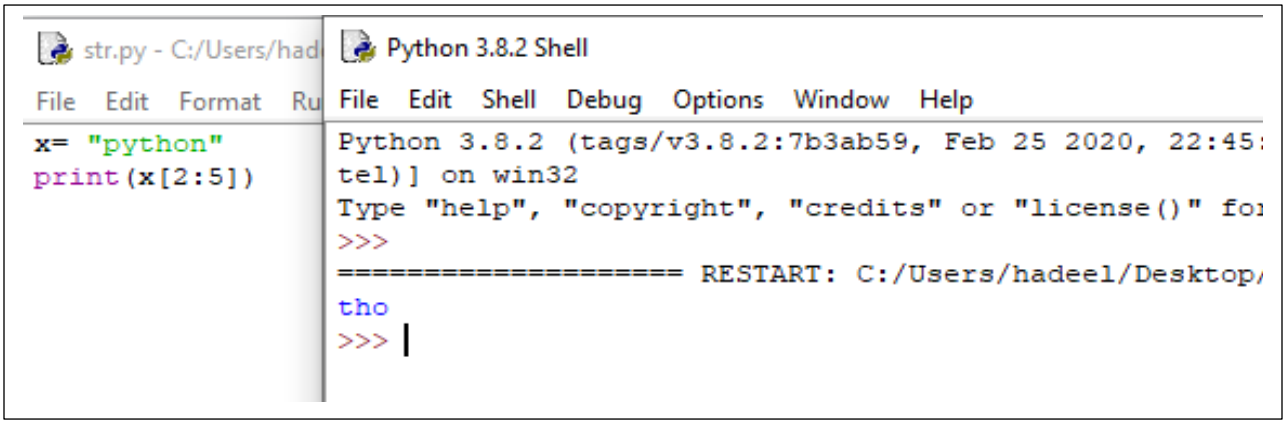
لكن في حال اردنا ان نقوم بالفهرسة العكسية اي ان نبدأ من n باتجاه p ستكون الفهرسة index[-1] لحرف n و index[-2] لحرف o ..... الخ

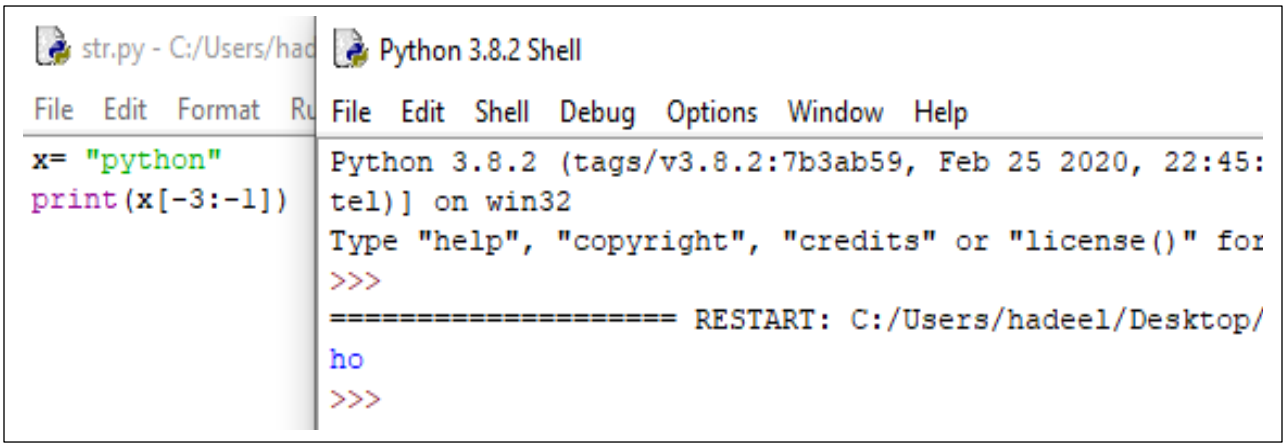
String	p	y	t	h	o	n
Index[]	-6 or 0	-5	-4	-3	-2	-1

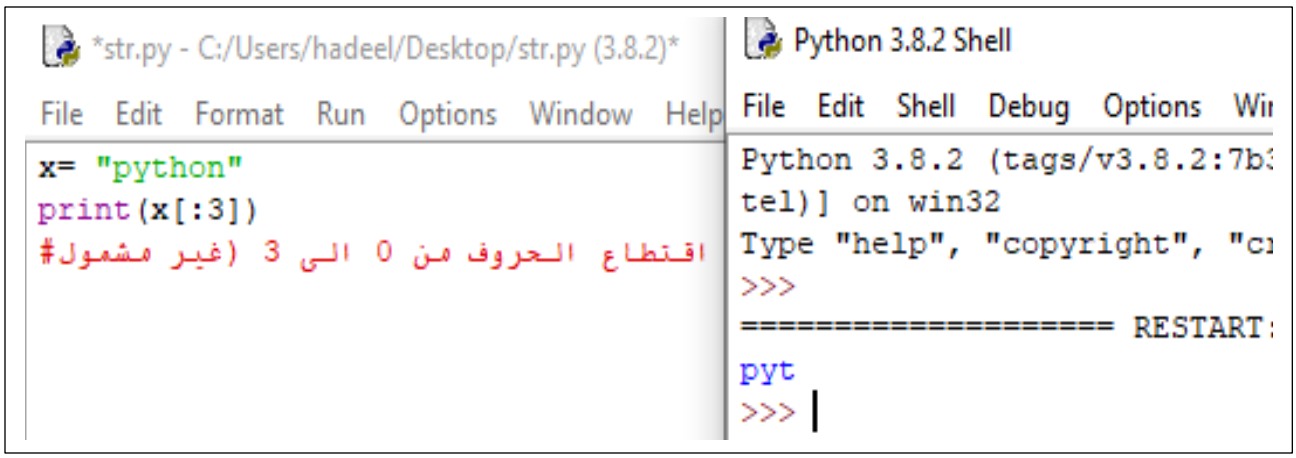
الفهرسة من اليمين الى اليسار

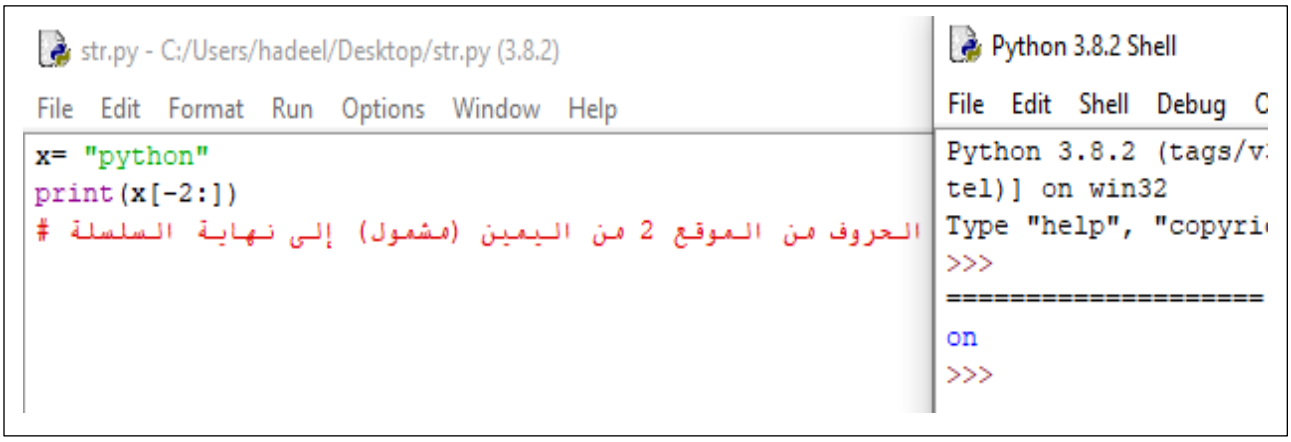


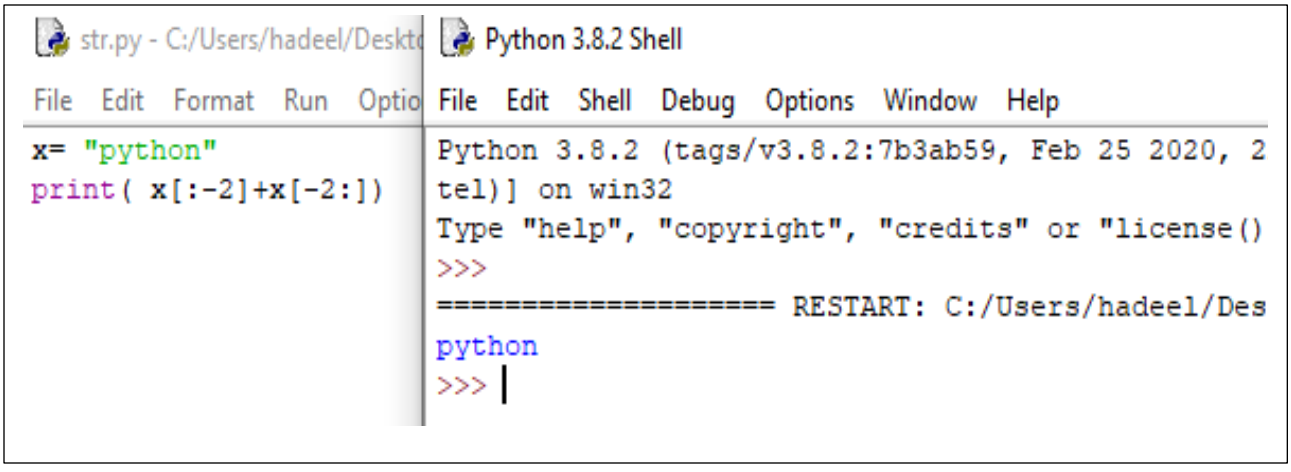
امثلة اخرى على indexing والتي يمكننا من خلالها فهرسة مدى معين من الاحرف حيث تسمى هذه العملية بالاقتطاع (slicing)من خلال وضع (:) النقطتين المتعامدتين فمثلا اقتطاع الحروف من الموقع[2] إلى الموقع [5] سيكون تمثيله [2:5] اي ثلاث حروف وهم (tho) سيكون الى index[4] اي [5] لا يدخل من ضمنها وكما موضح في الامثلة ادناه:





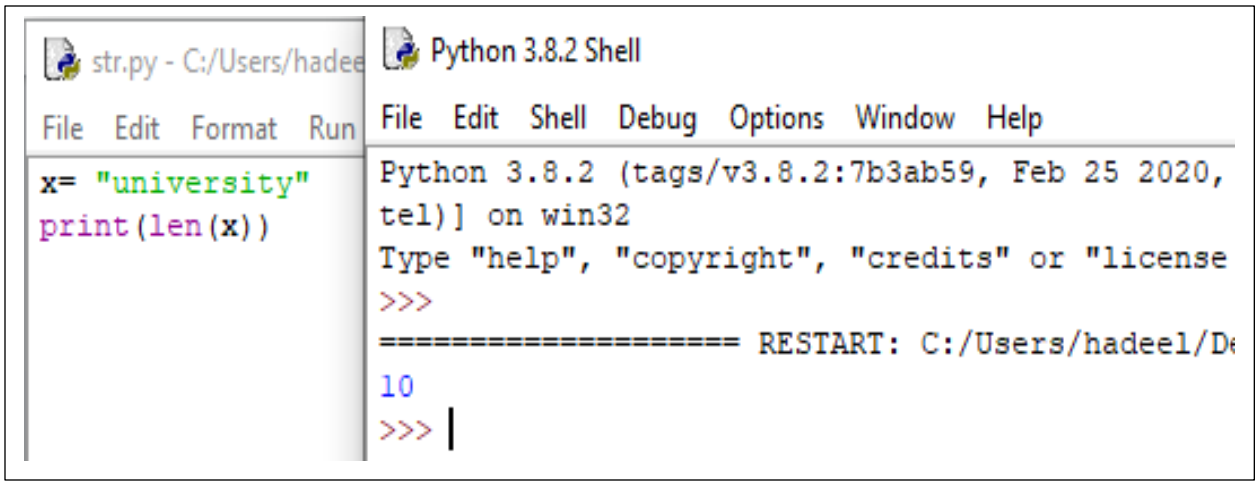






والامثلة كثيرة عن الاقتطاع يمكن ان تبدأ بتجربتك الخاصة مع البايثون وتجربة String slicing

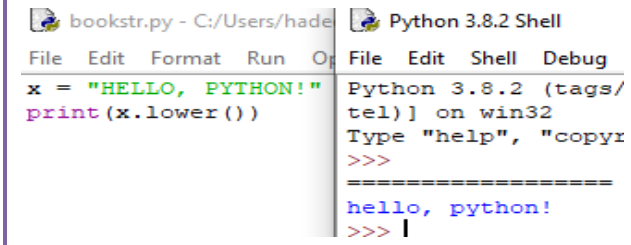
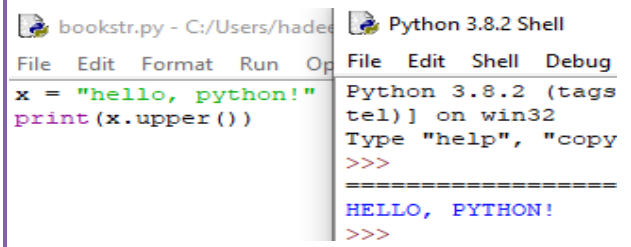
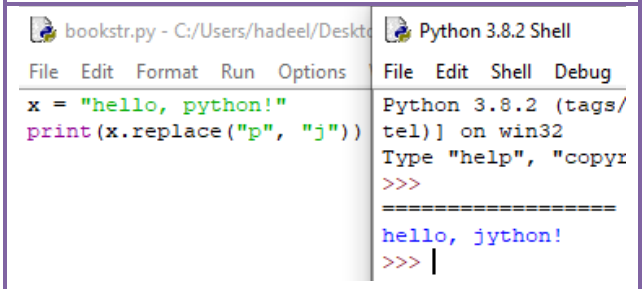
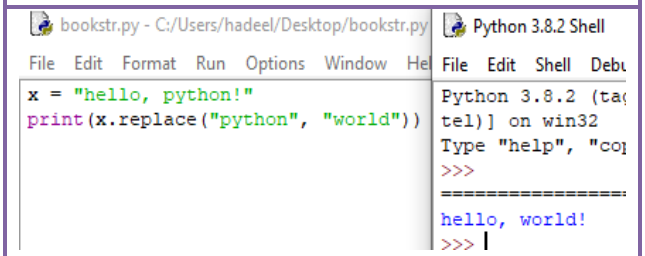
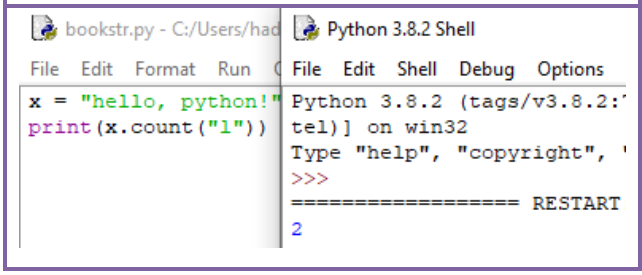
اما الان فلنأتي لشرح طول النص (String Length) حيث يمكن حساب طول النص من خلال الدالة الداخلية للبايثون والتي تدعى len() والتي تعمل على حساب عدد الحروف في النص واظهار النتيجة على شاشة التنفيذ. كما في الامثلة ادناه:

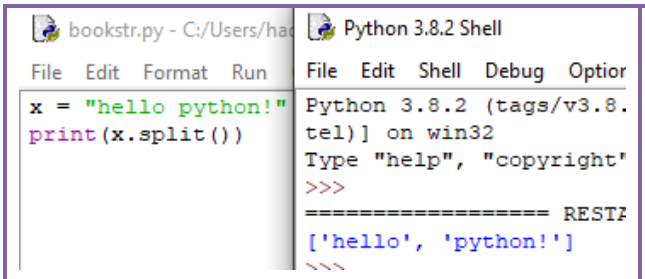
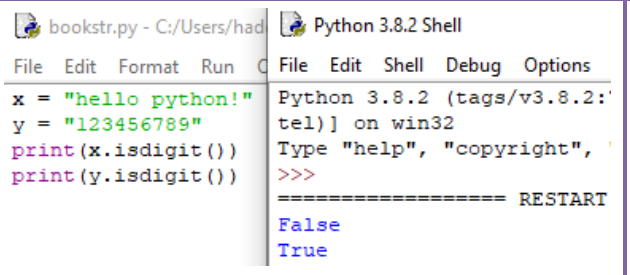
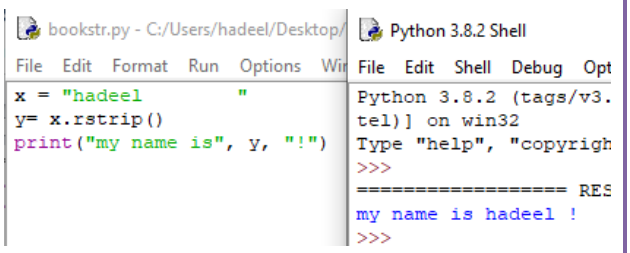
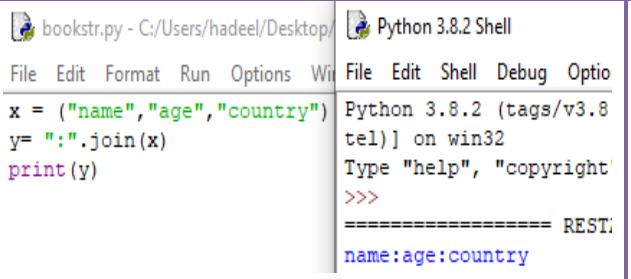
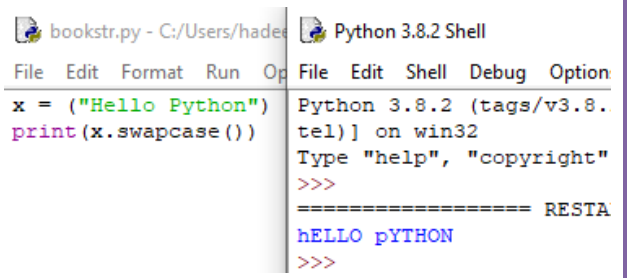


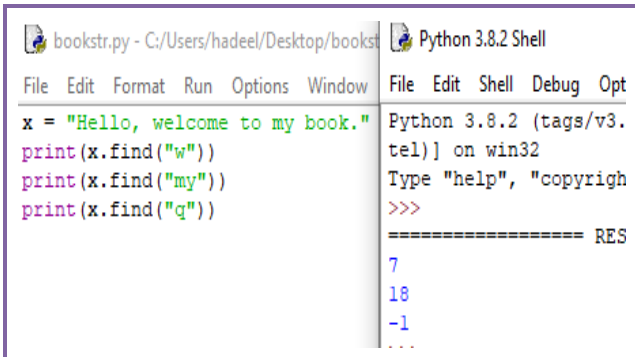
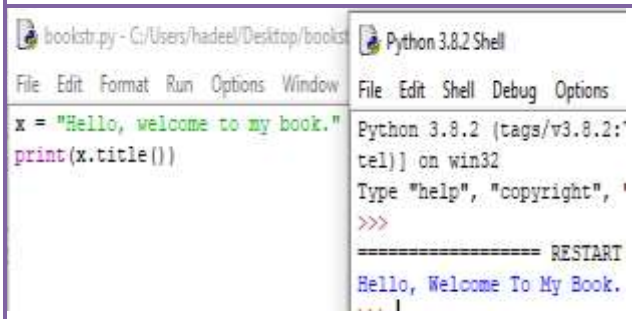
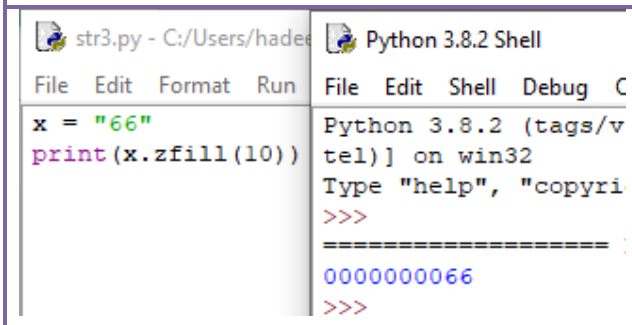
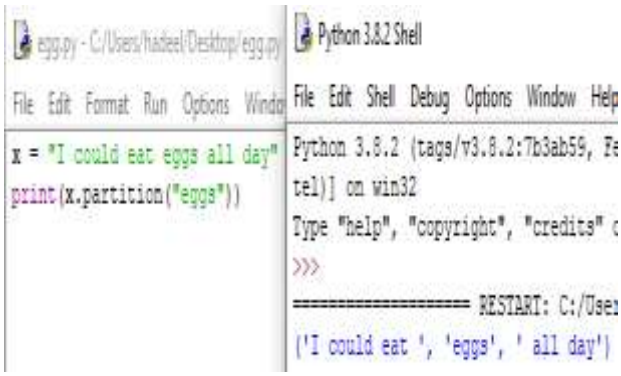
تحتوي البايثون ايضا على مجموعة من الدوال

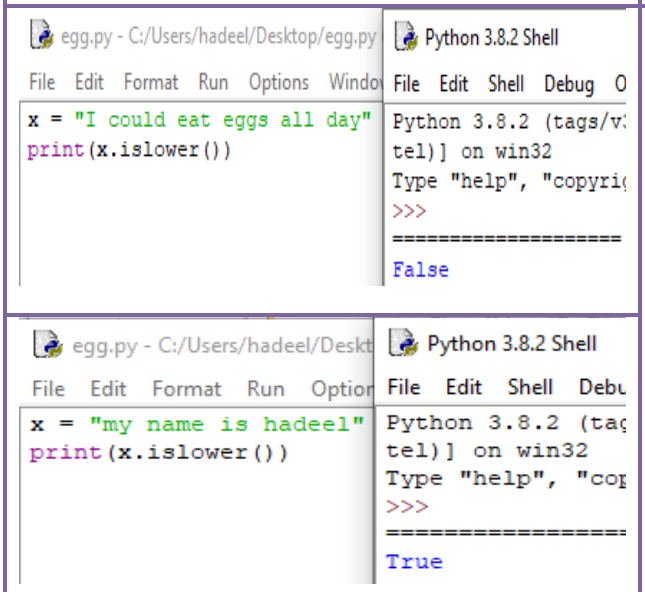
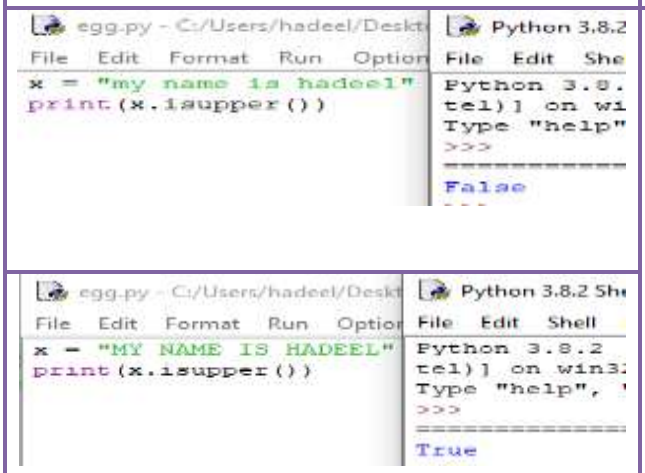
والتي يمكن استخدامها في النصوص بكل سهولة , وهي كثيرة ومتنوعة سنذكر منها اكثرها شيوعا:

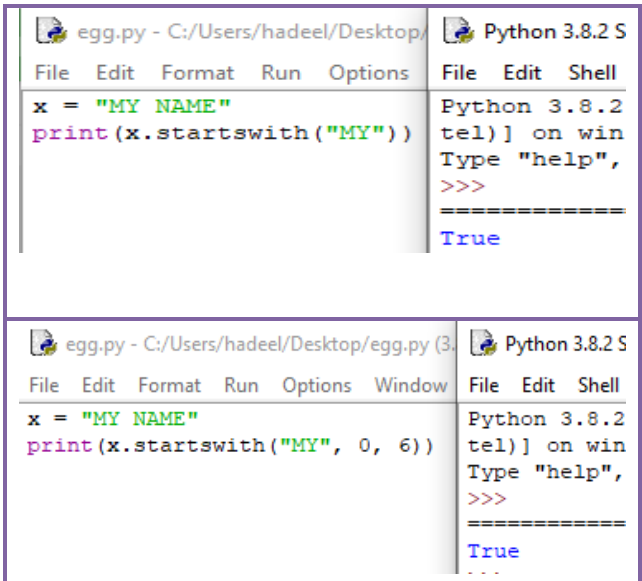
⇩

ت	اسم الدالة	عن الدالة	مثال عملي
١	lower()	تحوّل هذه الدالة الحروف في السلسلة النصية إلى حروف صغيرة.	
٢	upper()	تحوّل هذه الدالة الحروف في السلسلة النصية إلى حروف كبيرة.	
٣	replace()	تبدّل هذه الدالة الجملة او الحروف التي يختارها المستخدم الى جمل وحروف اخرى	 
٤	count()	تعيد هذه الدالة عدد مرات تكرار حرف معين او جملة معين في النصوص	

٥	split()	تعمل هذه الدالة على تقسيم الجملة الى كلمات منفصلة اي الى list	
٦	isdigit()	تتحقق ما إذا كان النص أرقامًا	
٧	rstrip()	تعمل على ازالة اي مسافات زائدة	
٨	Join()	ضم جميع العناصر في مجموعة كنص واحد من خلال استخدام حرف او رمز للتجزئة والذي يعمل كفاصل	
٩	swapcase ()	تجعل هذه الدالة الأحرف الكبيرة صغيرة والكبيرة تحولها الى احرف الصغيرة (اي تبديل حالة الحرف)	

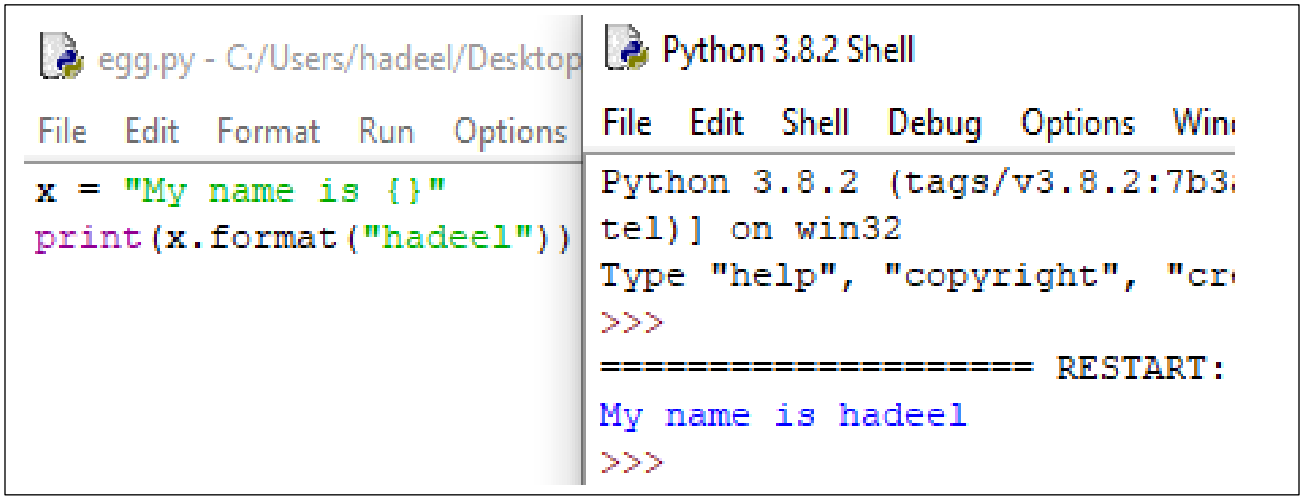
١٠	find()	تعد هذه الدالة مشابهة لعمل index حيث انها تقوم بالبحث عن حرف معين او جملة وترجع قيمة التواجد الاول للحرف المحددة كما في المثال الموضح وفي حال عدم وجود ذلك الحرف ترجع (-1)	
١١	title	تعمل هذه الدالة على تحويل الجملة الى حالة العنوان من خلال تحويل الحرف الأول في كل كلمة الى حرف كبير	
١٢	zfill()	تملء السلسلة بالأصفار حتى يبلغ طولها ١٠ أحرف	
١٣	Partition ()	تعمل هذه الدالة تقسيم الجملة إلى على ثلاثة عناصر: • العنصر الأول يحتوي على الجزء قبل الجملة المحددة. • يحتوي العنصر الثاني على الجملة المحددة. • يحتوي العنصر الثالث على الجزء ما بعد الجملة المحددة.	

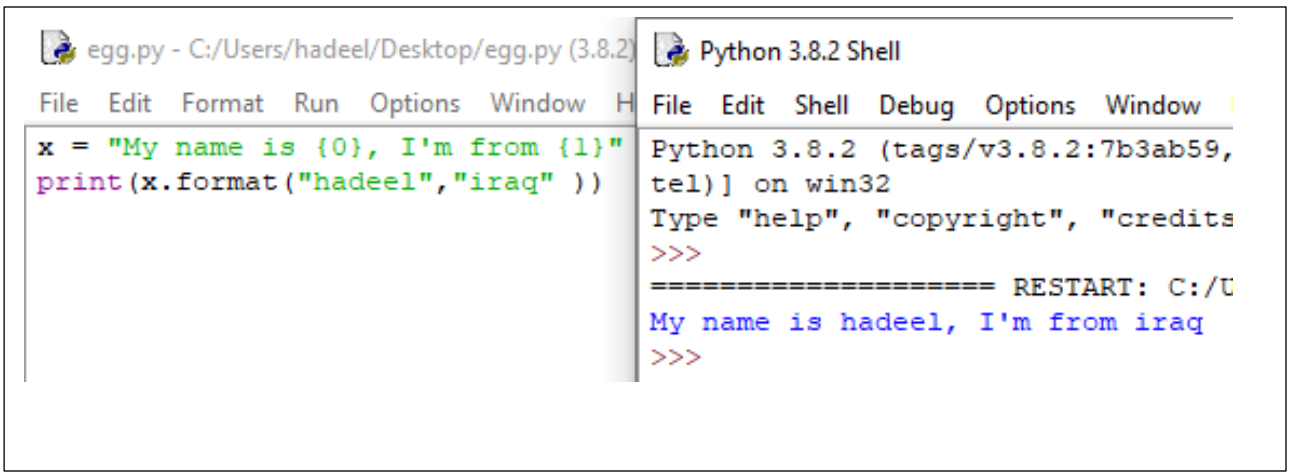
		ملاحظة: تبحث هذه الدالة عن التواجد الأول للجملة المحدد في حال تم تكرارها اكثر من مرة	
١٤	islower()	تتحقق هذه الدالة مما إذا كانت جميع الأحرف في النص حروف صغيرة..... هذه الدالة تعطي اجابة True or False	
١٥	isupper()	تتحقق هذه الدالة مما إذا كانت جميع الأحرف في النص حروف كبيرة..... هذه الدالة تعطي اجابة True or False	

١٦	startswith()	تتحقق هذه الدالة مما إذا كانت الstring تبدأ بكلمة محددة . اضافة الى ذلك في حال بحثنا في string عن كلمة محددة ونحدد القيمة البدائية والنهائية للجملة ستكون النتيجة طبعا اما True او ستكون False	

يحتوي الجدول اعلاه على بعض الدوال المهمة لكن هناك العديد من الدوال الاخرى التي يمكنك ان تبحث عنها ايضا في حال تم الاحتياج لها .

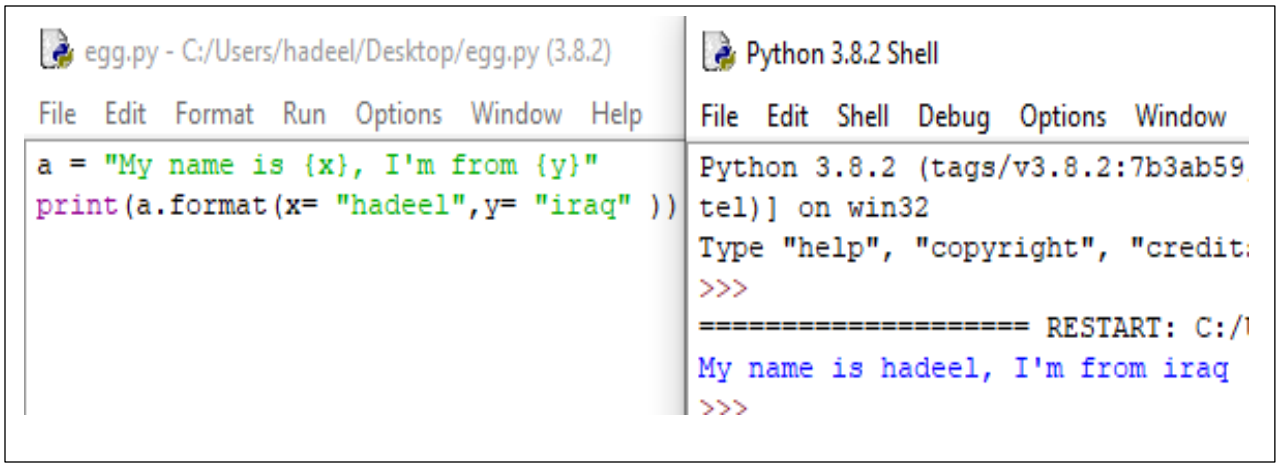
لكن قبل ان ننتهي من string يجب ان نوضح دالة مهمة وهي str.format() حيث تستخدم لإدراج ارقام او كلمات معينة خلال الجملة وذلك باستخدام الاقواس المتعرجة {} والامثلة ادناه ستوضح كيفية التعامل مع هذه الدالة:





نلاحظ في المثال اعلاه ان الكلمة الاولى "Hadeel" اخذت الموقع 0 و كلمة "Iraq" استقرت في موقع 1

ولكن يمكن استخدام طريقة اخرى وهي من خلال المتغيرات:



فضلا عن ذلك هناك العديد من Formatting Types سأذكر خمسة انواع منها بشكل سريع وكما هي

القاعدة تقول اذا اردت المزيد من formatting ابحث بسهولة عنها مع العم Google:

انواع Formatting	عملها	مثال
:<	محاذاة الكلمة إلى اليسار من خلال مسافة محددة	egg.py - C:/Users/hadeel/Desktop/egg.py (3.8.2) File Edit Format Run Options Window Help a = "My name is {:<12} i am from iraq" print(a.format("hadeel")) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:/Users/ My name is hadeel i am from iraq >>>
:>	محاذاة الكلمة الى اليمين من خلال مسافة محددة	egg.py - C:/Users/hadeel/Desktop/egg.py (3.8.2) File Edit Format Run Options Window Help a = "My name is {:>12} i am from iraq" print(a.format("hadeel")) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:/Users/ My name is hadeel i am from iraq >>>
:^	توسيط المسافة للكلمة في المنتصف	egg.py - C:/Users/hadeel/Desktop/egg.py (3.8.2) File Edit Format Run Options Window Help a = "My name is {:^12} i am from iraq" print(a.format("hadeel")) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:/Users/ My name is hadeel i am from iraq >>>
,	لإضافة فاصلة ما بين الارقام	egg.py - C:/Users/hadeel/Desktop/eg File Edit Format Run Options W a = "i have {:,}" print(a.format(3000555666)) Python 3.8.2 Shell File Edit Shell Debug Options Wir Python 3.8.2 (tags/v3.8.2:7b3 tel)] on win32 Type "help", "copyright", "cr >>> ========== RESTART: i have 3,000,555,666 >>>
:b	تعمل على تحويل الرقم الى رقم ثنائي	egg.py - C:/Users/hadeel/Desktop/egg.py (3.8.2) File Edit Format Run Options Window Help x = "The binary version of {0} is {0:b}" print(x.format(2)) print(x.format(5)) Python 3.8.2 Shell File Edit Shell Debug Options Windc Python 3.8.2 (tags/v3.8.2:7b3ak tel)] on win32 Type "help", "copyright", "crec >>> ========== RESTART: (The binary version of 2 is 10 The binary version of 5 is 101

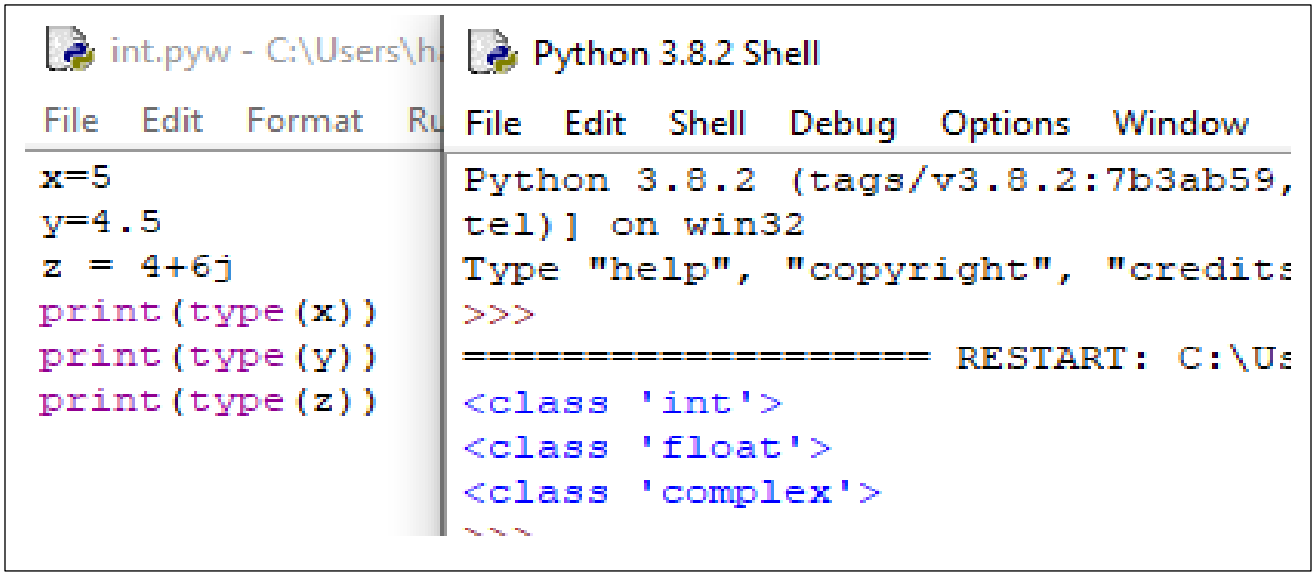
٢. الارقام(Numbers)

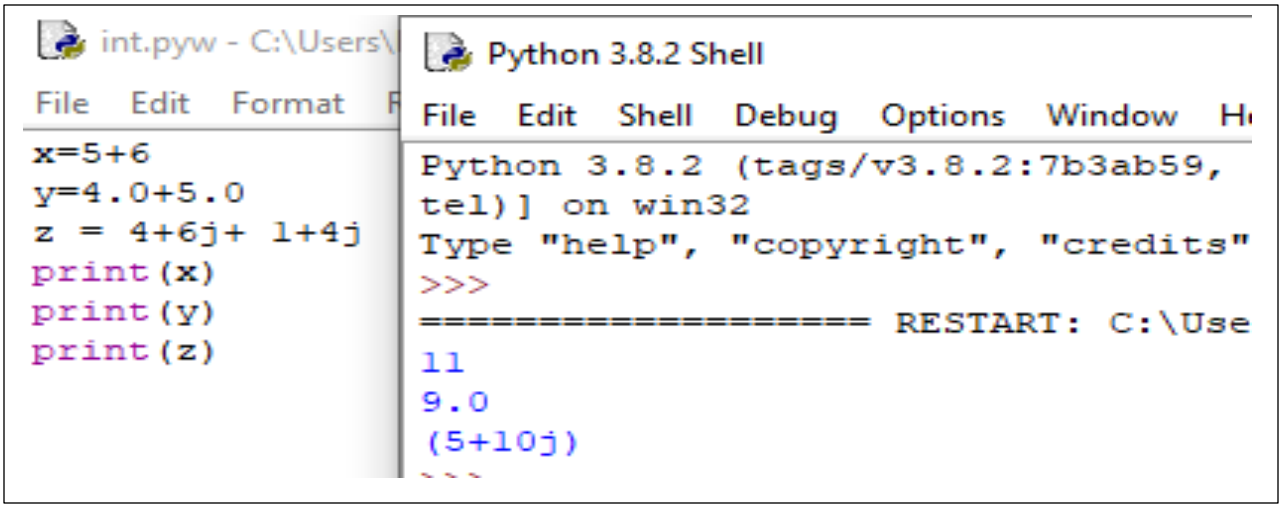
سبق وان تحدثنا عن الارقام int , float في (متغيرات البايثون) وتعلمنا هناك انه نستطيع معرفة نوع المتغير من خلال type() اضافة الى ذلك استطعنا جمع عددين int وايضا عددين float من خلال operator (+)

لكن الان يجب ان نعترف بان هناك في الحقيقة ثلاث انواع من الارقام وليس اثنان وهي كالاتي:

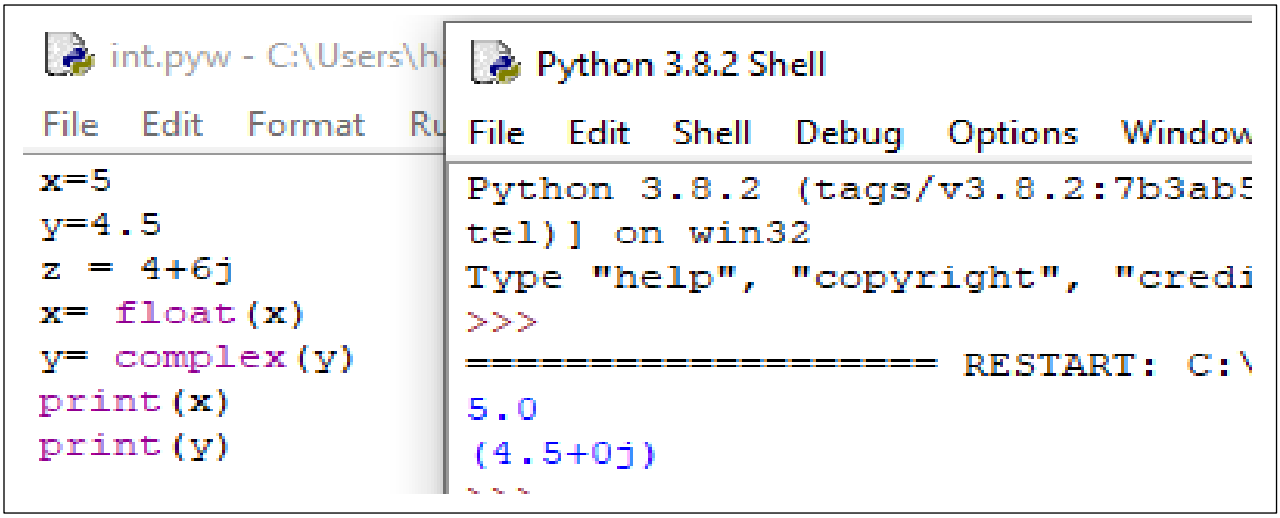
- int رقم صحيح: عدد صحيح (integer)مختصره (int)هو عدد صحيح سواء كان موجب أو سالب ، بدون كسور عشرية ، وبطول غير محدود مثال(x = 12345678910)(x= -156)
- Floatرقم عشري: هو رقم يحتوي على فاصلة .والرقم العشري يمكن ان يكون موجب أو سالب (x= -10.5) (x=1.0)
- Complex رقم المركب : هو عدد يتكون من جزئين جزء حقيقي وجزء خيالي (j) وسيكون شكل الرقم كالاتي (4+6j)

الأمثلة ادناه تمثل الانواع الثلاثة في البايثون :





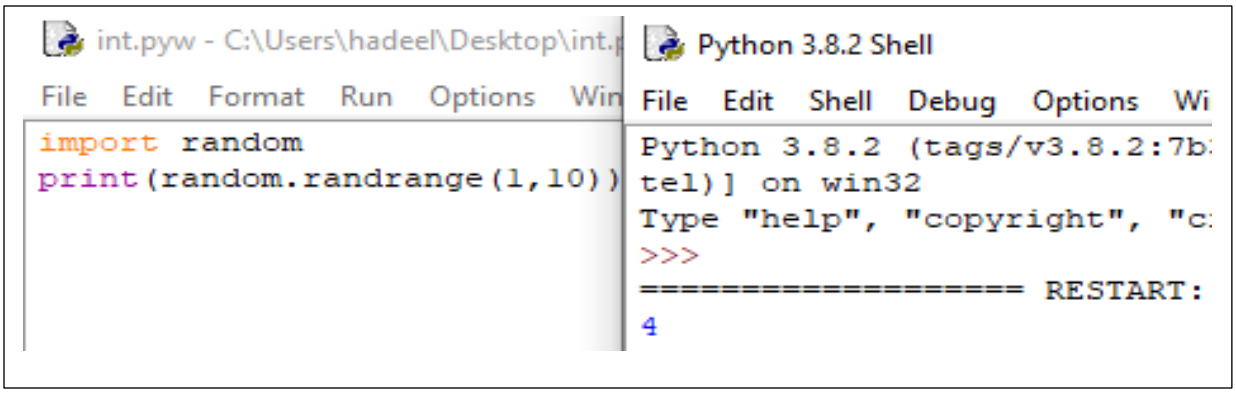
يمكنك التحويل من نوع إلى آخر باستخدام الطرق int() و float() complex():

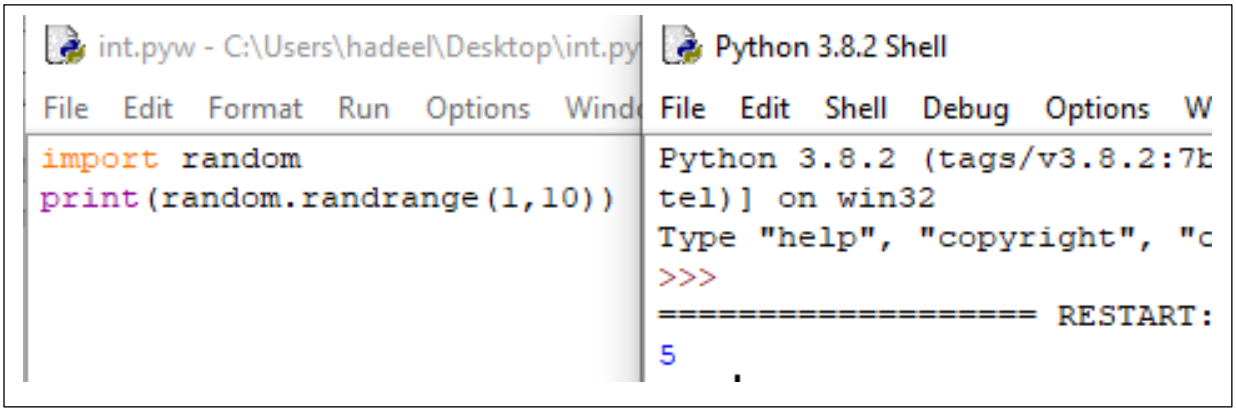


هناك ملاحظة مهمة وهي ان العدد المركب لا يمكن تحويله الى عدد صحيح او عدد عشري كما مبين في المثال اعلاه حيث لم نستطيع ان نحول العدد المركب الى صحيح .

لكن يمكن تحويل العدد الصحيح والعشري الى عدد مركب

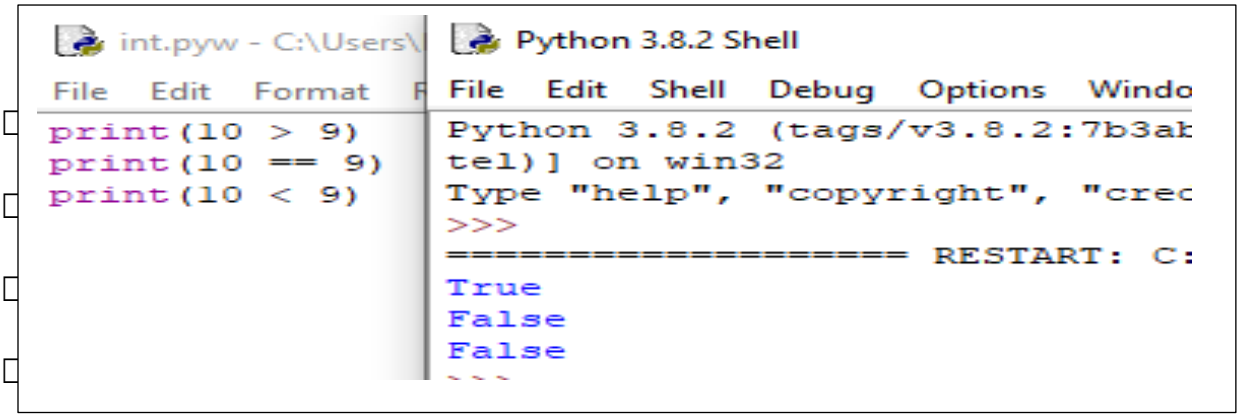
وفي ختام فقرة الارقام يجب ان نتطرق الى الرقم العشوائي(Random Number) حيث تسطيع ان تستدعي مكتبة random ومن خلالها نقوم بتوليد رقم عشوائي و في حال تكرار التنفيذ ستعطي رقم ضمن المدى المحدد كما في المثال ادناه ستكون الارقام عشوائية من ١ الى ١٠ لكن من الضروري البدء باستدعاء المكتبة:

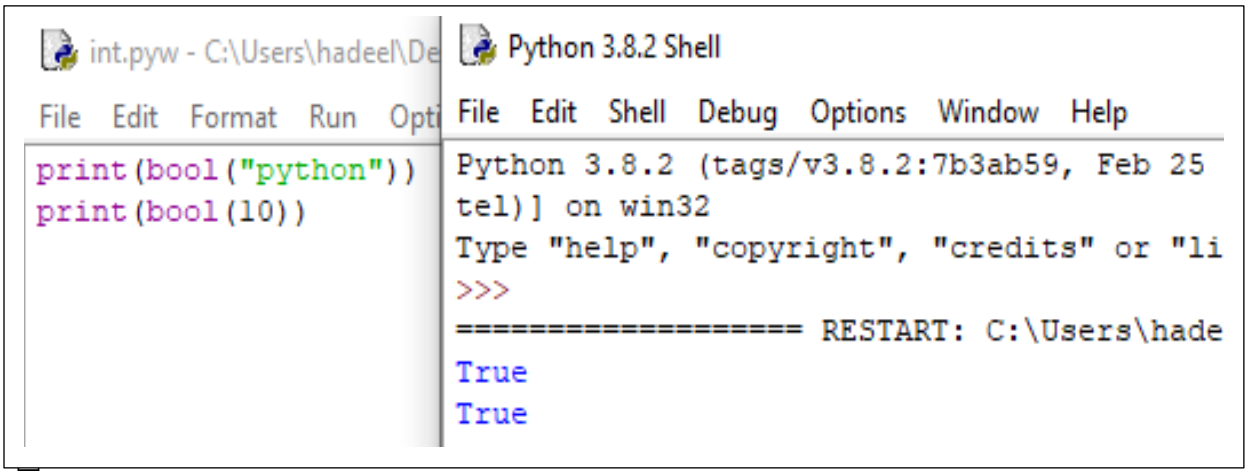




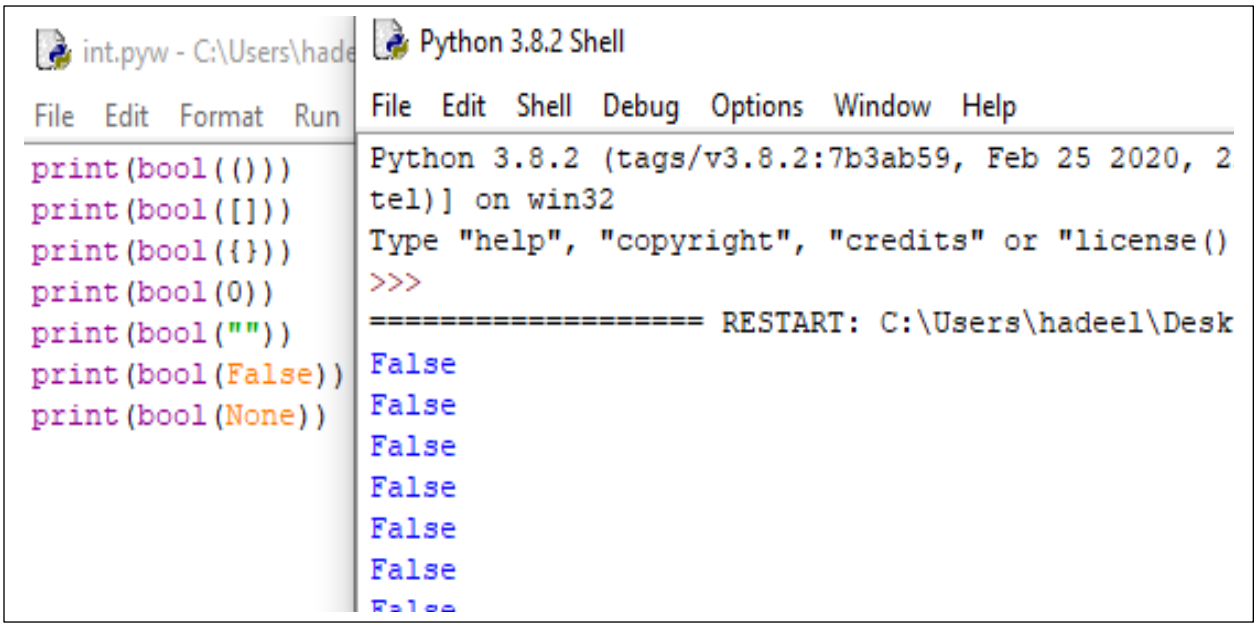
٣. القيم المنطقية (Boolean)

تمثل القيم المنطقية إحدى القيمتين: صواب أو خطأ تستخدم كثيرا مع if ولكن يمكن استخدامها بشكل منفرد كالمقارنة حيث تعطي True or false او من خلال الدالة bool () والتي تسمح بتقييم القيمة التي بداخلها وغالبا ما تكون True .





لكن هناك حالات خاصة عادة ما تكون النتيجة (False) وهي كالتالي:

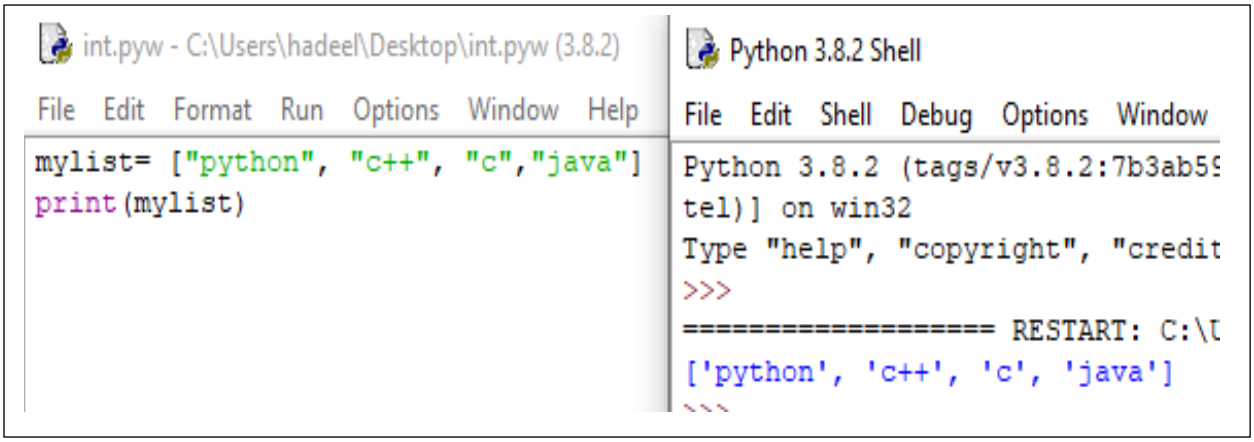


٤.متسلسلات او مصفوفات list & tuple

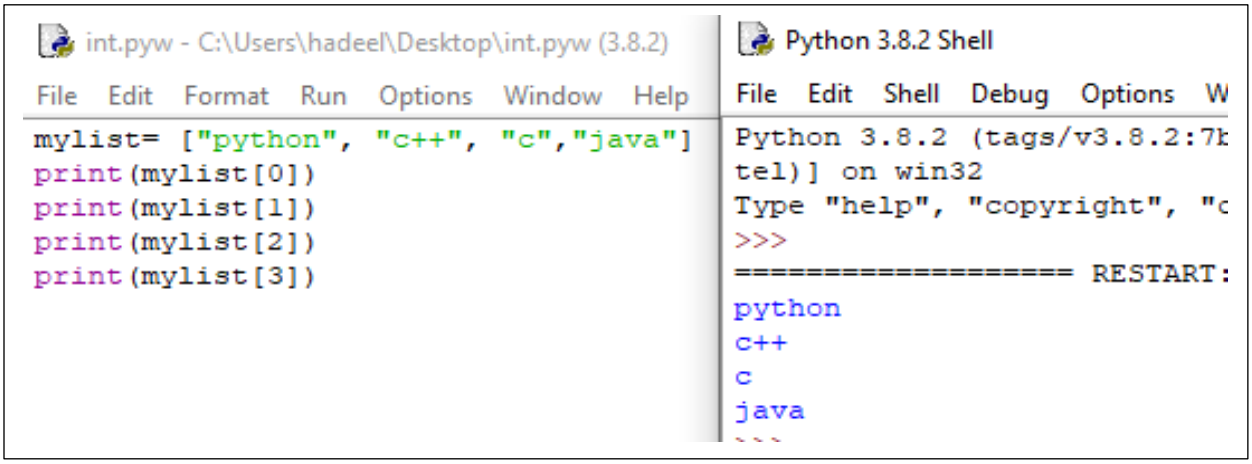
القائمة list هي عبارة عن مجموعة مرتبة ومتغيرة بينما ال tuple هي مجموعة مرتبة وغير قابلة للتغيير.

لذا لنبدأ ب list :

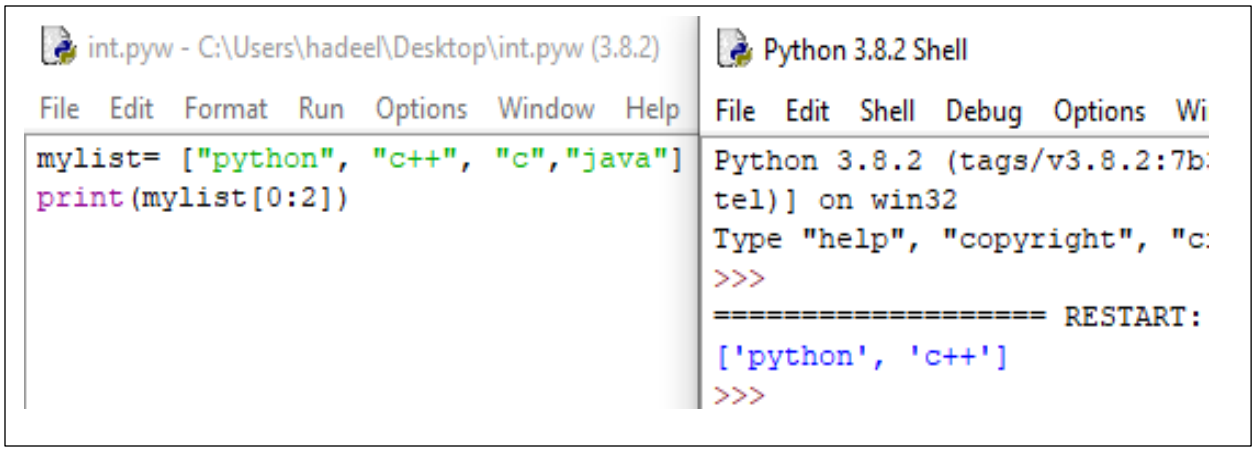
تتميز هذه القائمة ب الاقواس المربعة [] كما نلاحظ في المثال ادناه:

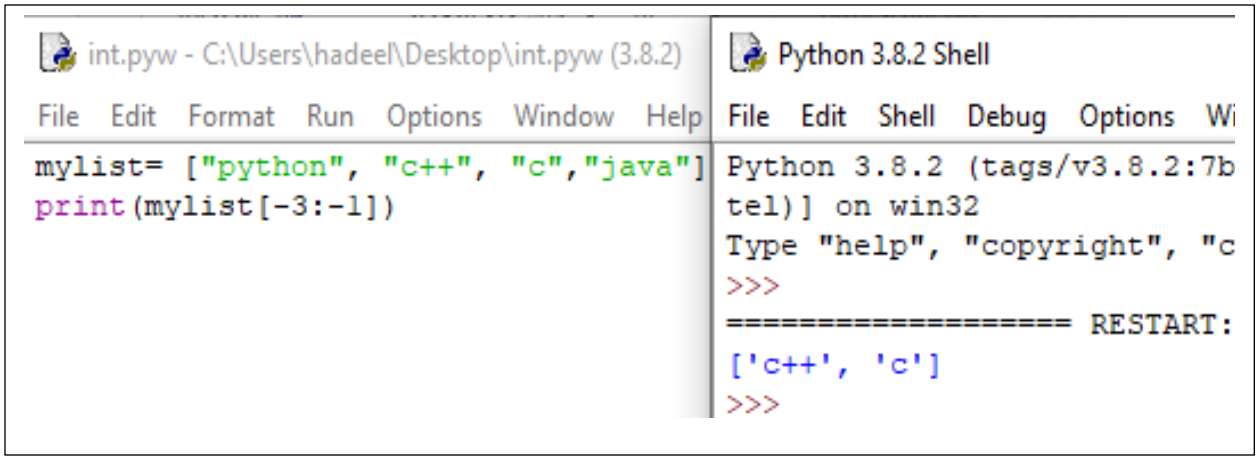


يمكننا الوصول إلى عناصر القائمة list من خلال الرجوع إلى رقم الفهرس (كما هو الحال في string):

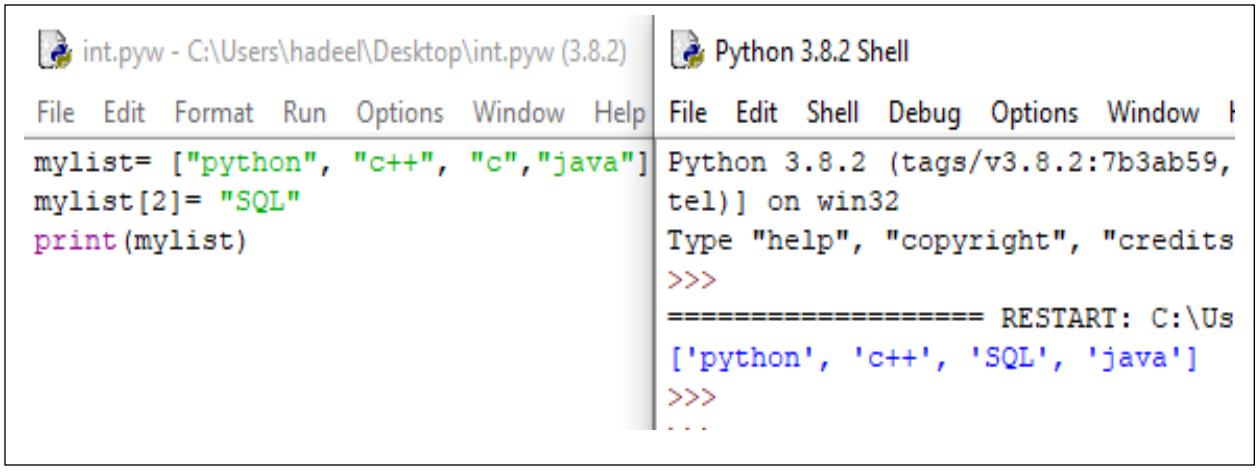


وايضا يمكننا تحديد مدى معين من الفهرس (كما في الفهرسة التي تم شرحها مسبقا).

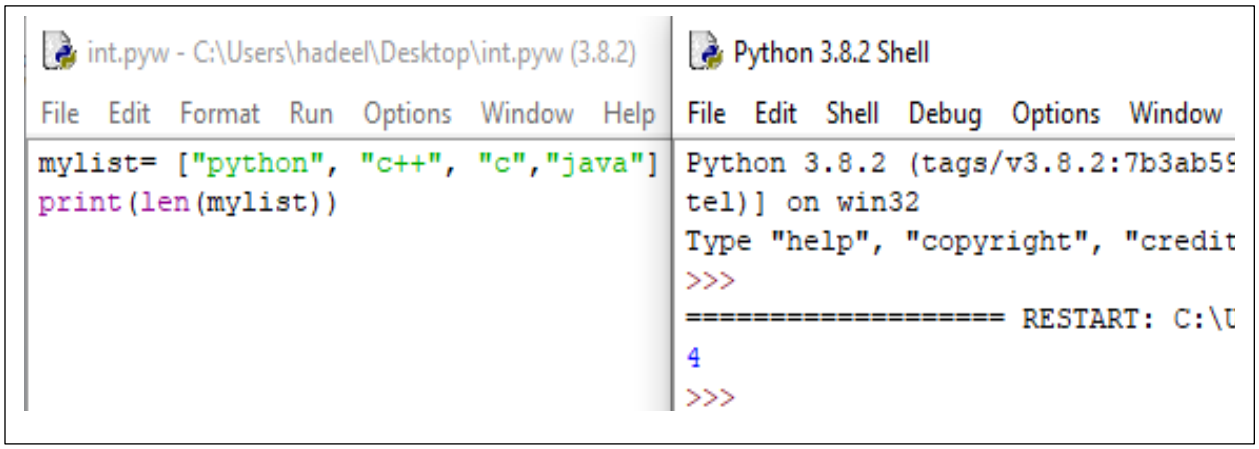




يمكن ايضا تغيير قيمة عنصر معين ، من خلال رقم الفهرس فمثلا يمكننا تغيير لغة C واستبدالها بSQL

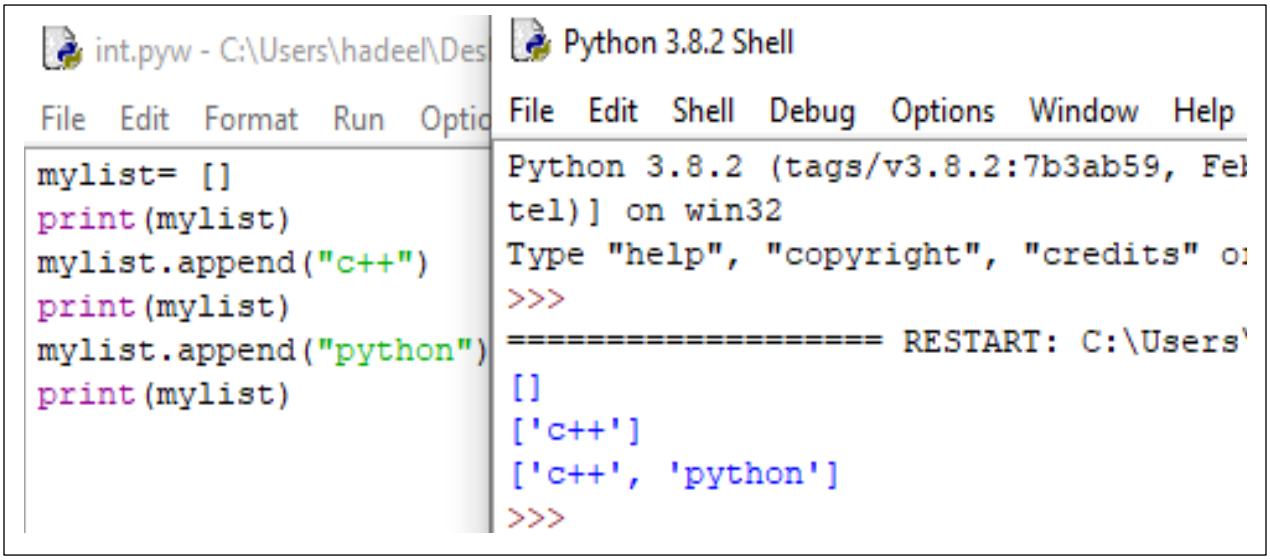


لحساب عدد العناصر الموجودة بالقائمة ، نستخدم الدالة ( )len و التي سيتضح لنا من خلالها ان عدد العناصر الموجودة في القائمة هي اربعة عناصر فقط.

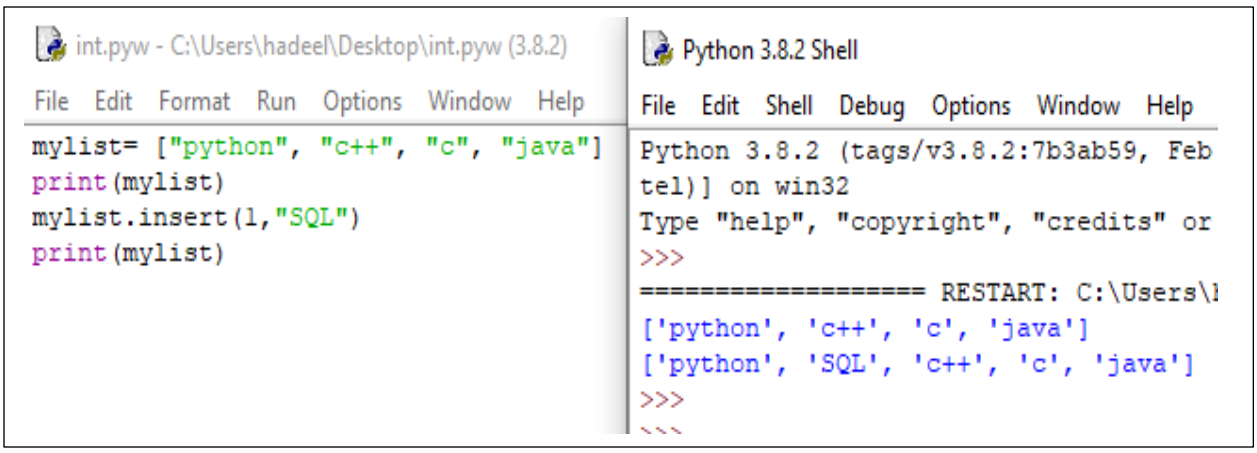


يمكن ملء قائمة فارغة بالعناصر من خلال دالة append() وايضا يمكن اضافة عنصر لقائمة تحتوي على عناصر من خلال نفس الدالة :

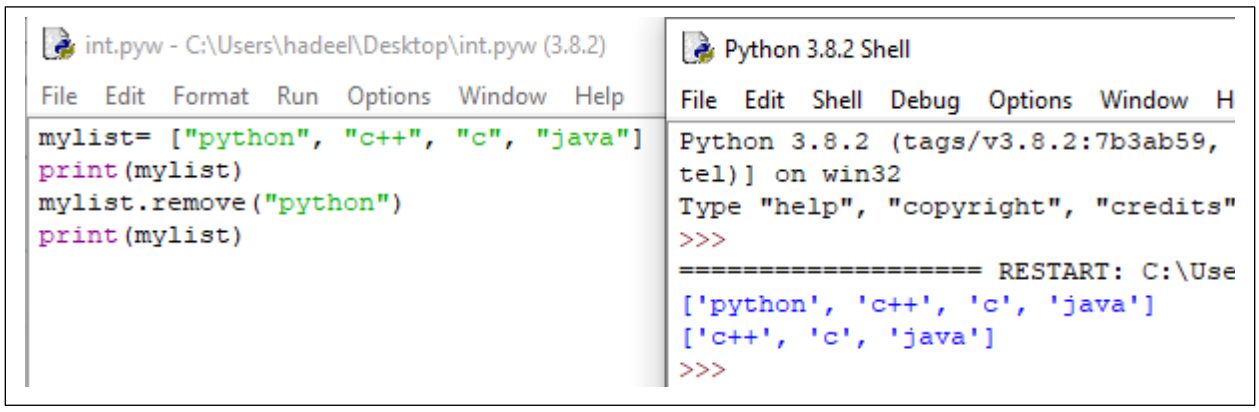
لكي نستعرض دالة append بداية سنضع قائمة فارغة ثم نقوم بملئها بالعناصر :



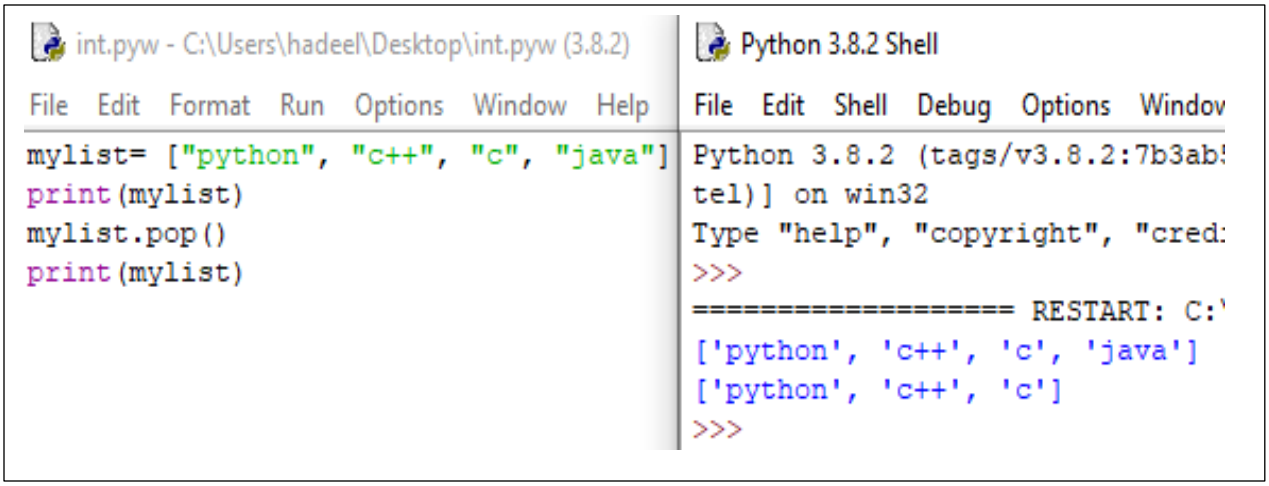
نلاحظ ان دالة append() تعمل على اضافة العناصر بنهاية القائمة فقط لكن يمكننا ايضا اضافة عناصر محددة بموقع محدد بالاعتماد على الفهرسة اي specified index من خلال دالة insert()



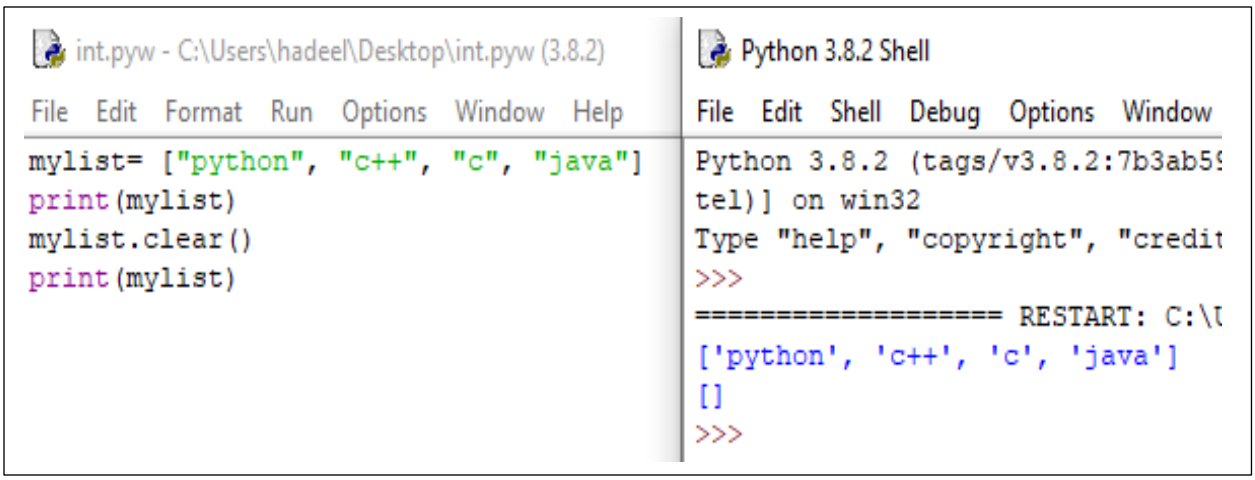
من الجهة الاخرى نستطيع حذف عنصر محدد(specified item) ونركز على كلمة محدد من القائمة باستخدام دالة remove() كالاتي :



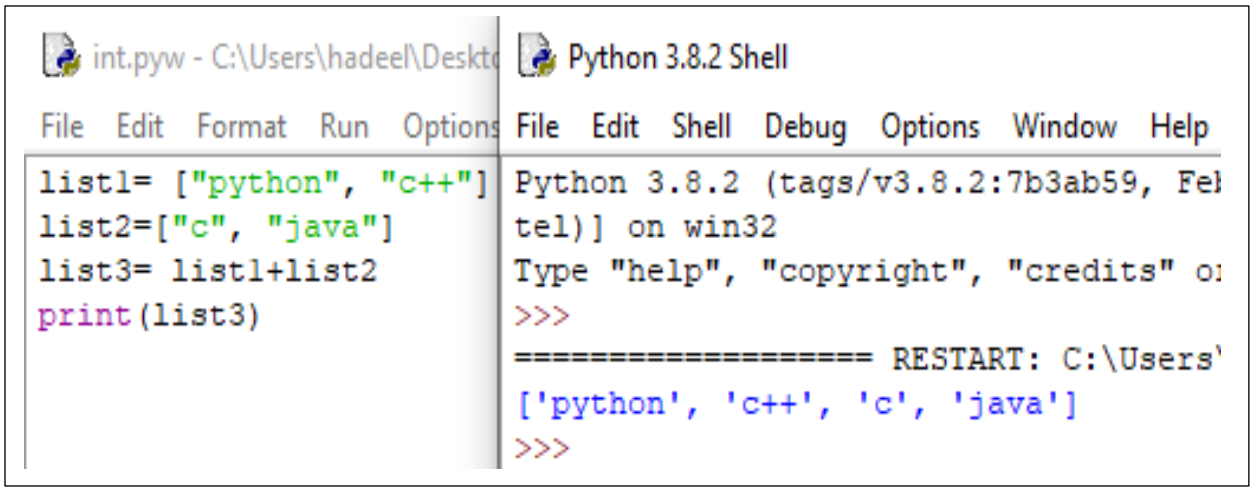
اما اذا اردنا حذف عنصر بدون تحديد (اي سيقوم البايثون بحذف العنصر الاعلى index بالاعتماد على الفهرسة)سيكون هذا من خلال دالة pop():



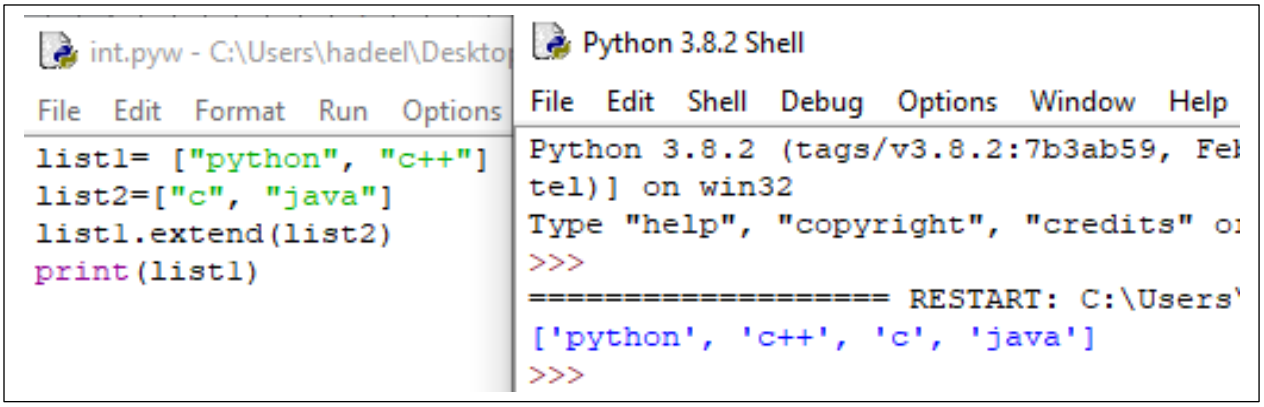
ونستطيع حذف العناصر جميعها من القائمة وجعل القائمة فارغة تماما وبسهولة من خلال دالة عملية جدا وهي clear()



يمكننا الجمع بين قائمتين من خلال operator(+)وبسهولة كما في المثال التالي:



يمكننا ايضا توسيع القائمة بدمج قائمة اخرى معها اي اضافة عناصر من قائمة الى قائمة ثانية من خلال دالة extend()



فضلا عن ذلك يمكن استخدام list لتكون constructor الى list اخرى وسنلاحظ ان القائمة ستكون بأقواس مدورة () وذلك لأنها ستكون constructor الى القائمة الجديدة :

int.pyw - C:\Users\hadeel\Desktop\int.pyw

File Edit Format Run Options Window

```
listl = list(("python", "c++"))
print(listl)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Wind

```
Python 3.8.2 (tags/v3.8.2:7b3a
tel)] on win32
Type "help", "copyright", "cre
>>>
========== RESTART: C
['python', 'c++']
```

ولكن هل يمكننا عكس القائمة list؟ ؟ الجواب نعم لان البايثون سهلت تلك العملية من خلال دالة reverse()

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)

File Edit Format Run Options Window Help

```
mylist= ["python", "c++", "c", "java"]
print(mylist)
mylist.reverse()
print(mylist)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Hel

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, F
tel)] on win32
Type "help", "copyright", "credits"
>>>
========== RESTART: C:\User
['python', 'c++', 'c', 'java']
['java', 'c', 'c++', 'python']
>>>
```

واخيرا يمكننا عمل فرز للقائمة أبجديا من خلال دالة sort():

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)

File Edit Format Run Options Window Help

```
mylist= ["python", "c++", "c", "java"]
mylist.sort()
print(mylist)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

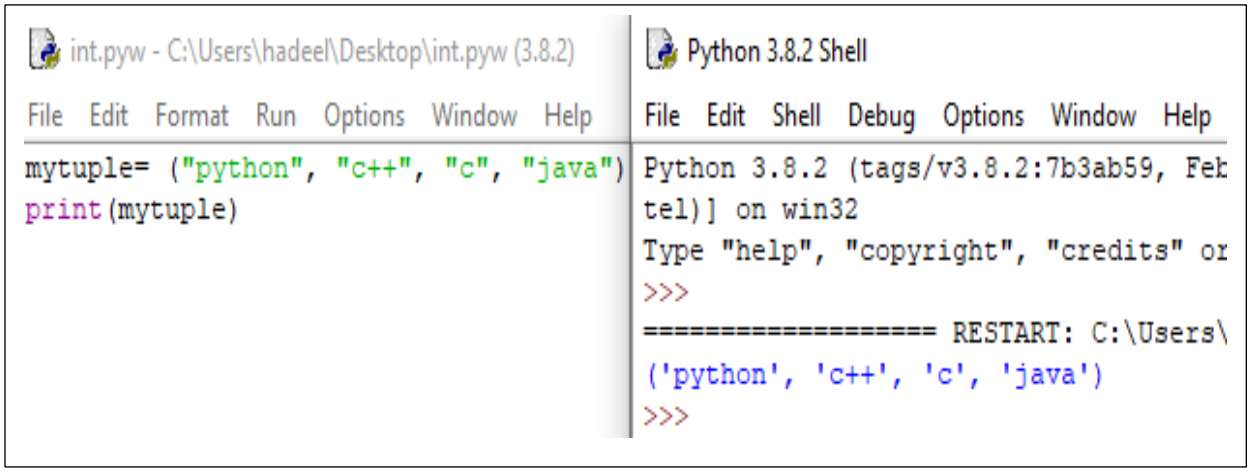
```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
['c', 'c++', 'java', 'python']
>>>
```

لقد انتهينا من التفاصيل المتعلقة بال list

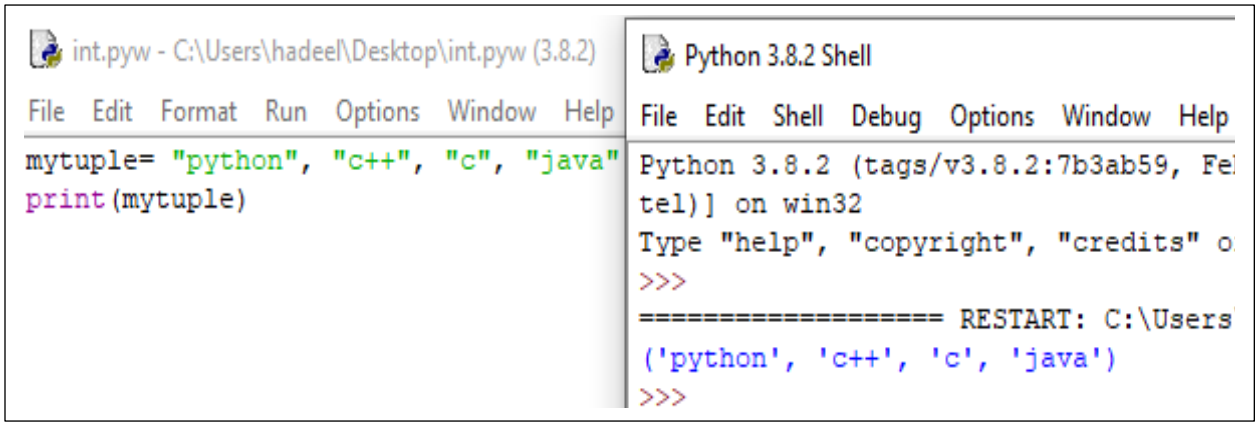
والان حان دور التعرف على Tuple:-

tuple هي مجموعة مرتبة وغير قابلة للتغيير حيث يتم كتابة tuple بأقواس مستديرة () .

ويمكن القول ان طباع tuple تشابه الى حد ما list وسنلاحظ الان اوجه التشابه والاختلاف من خلال صفاتها التالية :

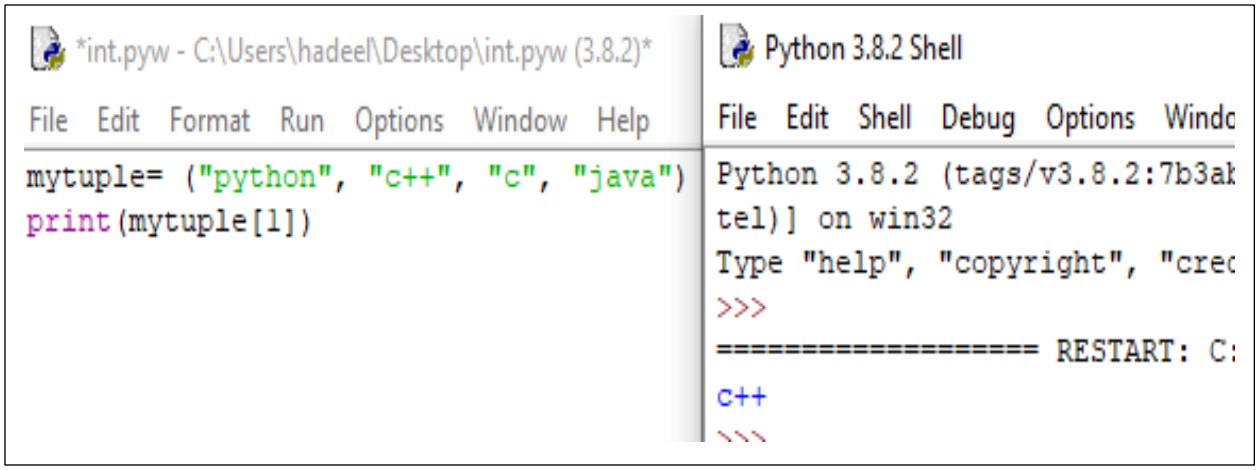


في البايثون يتم التعرف على العناصر التي بدون اقواس واعطاء النتيجة بانها tuple:



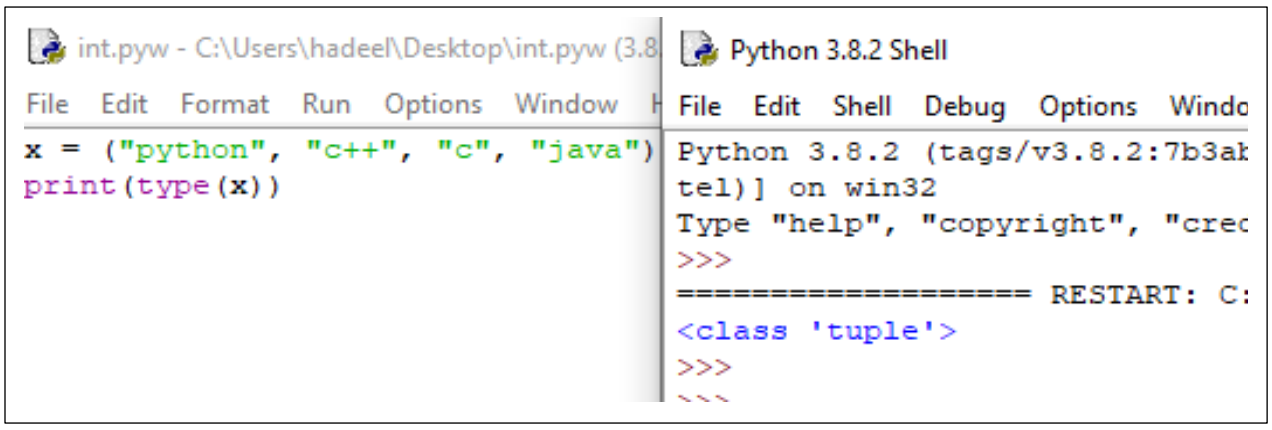
نلاحظ اعلاه ان البايثون اعطى النتيجة ذاتها في الحالتين بأقواس او بدون اقواس وتعرف عليها على انها tuple

يمكننا ايضا الوصول إلى اي عنصر داخل tuple من خلال الفهرسة index number واعتقد ان هذه القاعدة سبق وان تم شرحها في list.

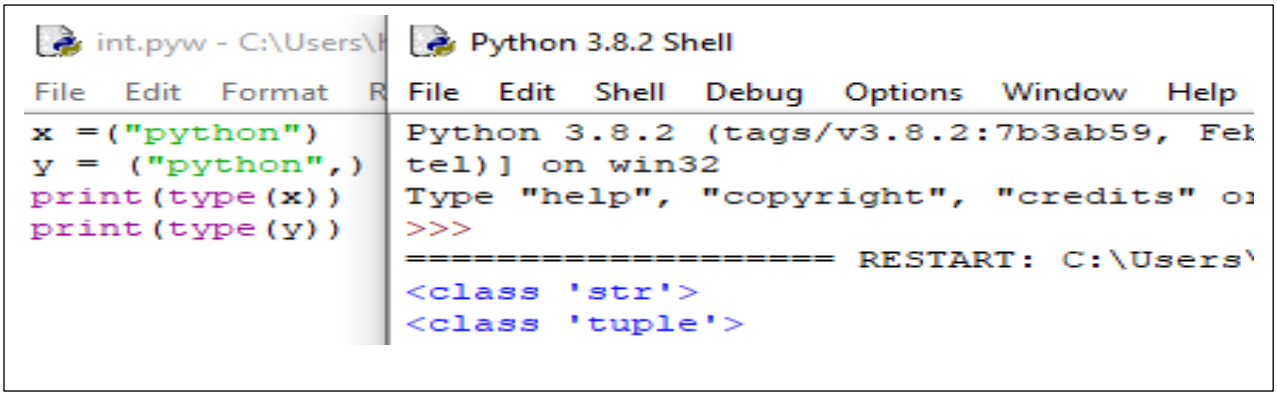


قبل ان نكمل في tuple يمكننا توضيح الفرق الرئيسي بين tuple وlist هو أن list قابلة للتغيير وtuple ليست كذلك اي انها تحتوي على قيمة ثابتة لا يمكن تغييرها في هذه الحالة اذا اردنا تغيير قيمة فيجب انشاء tuple جديدة .

يمكننا معرفة نوع tuple من خلال الدالة type():

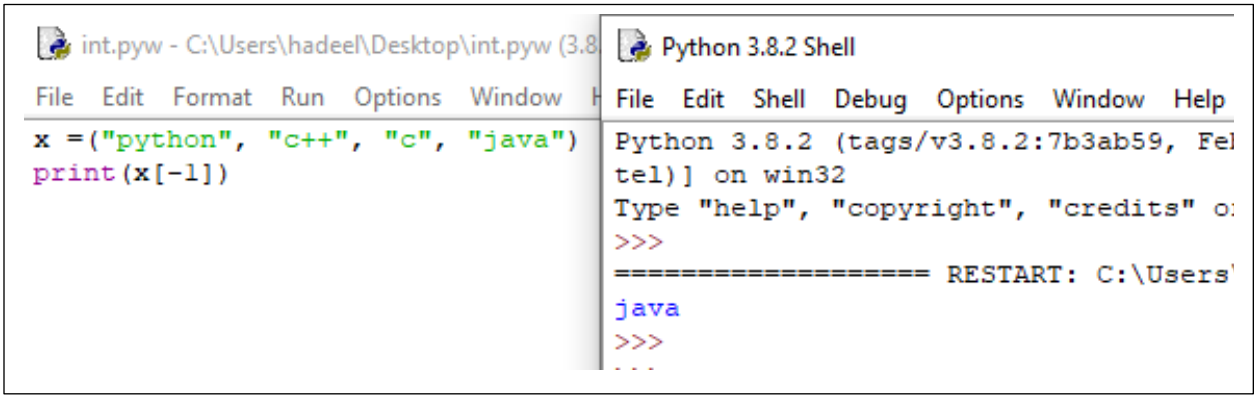


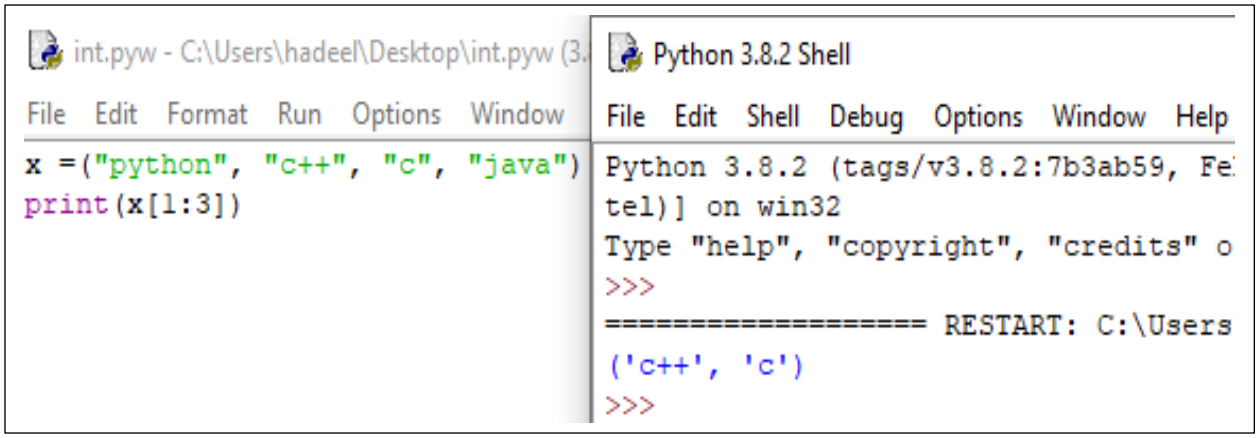
لكن كيف يميز البايثون string من tuple سنرى قوة الملاحظة لديك في المثال التالي:



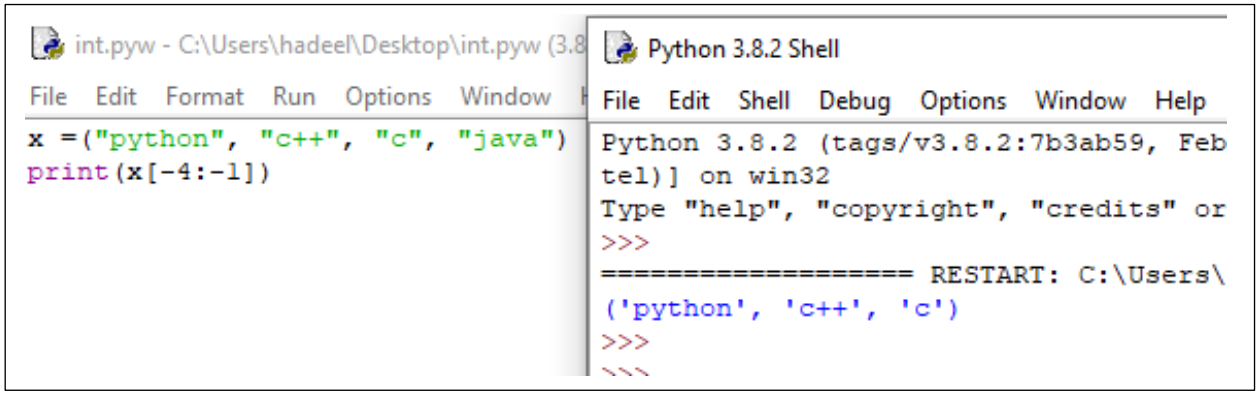
الفرق بينهما واضح جدا وهي الفاصلة حيث نلاحظ ان البايثون ميز بان تكون الجملة بدون فاصلة ولكن بمجرد اضافتها تم تحويل نوعها الى tuple.

والان سنستكمل شرح الفهرسة بالأمثلة التالية :

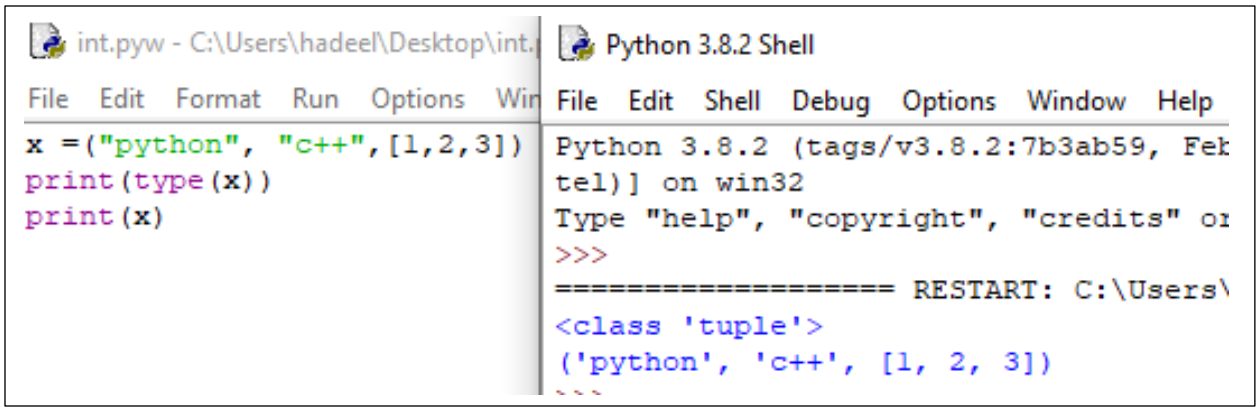


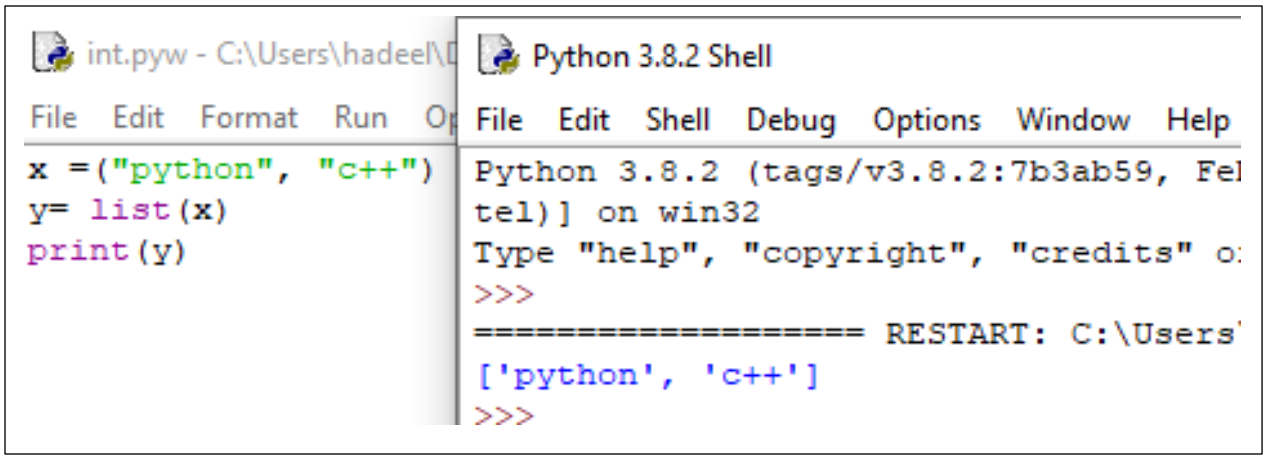


تذكر دائما أن العنصر الأول يحتوي على فهرس 0 وان نهاية tuple تبدأ ب1-

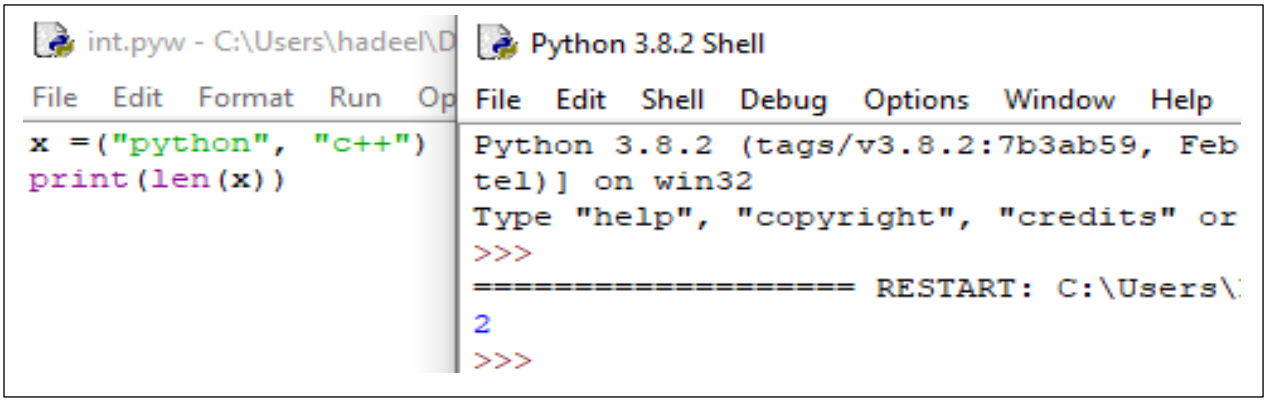


يمكن tuple ان تحتوي بداخلها على list , فضلا عن ذلك يمكن ايضا تحويل tuple الى list كالتالي:

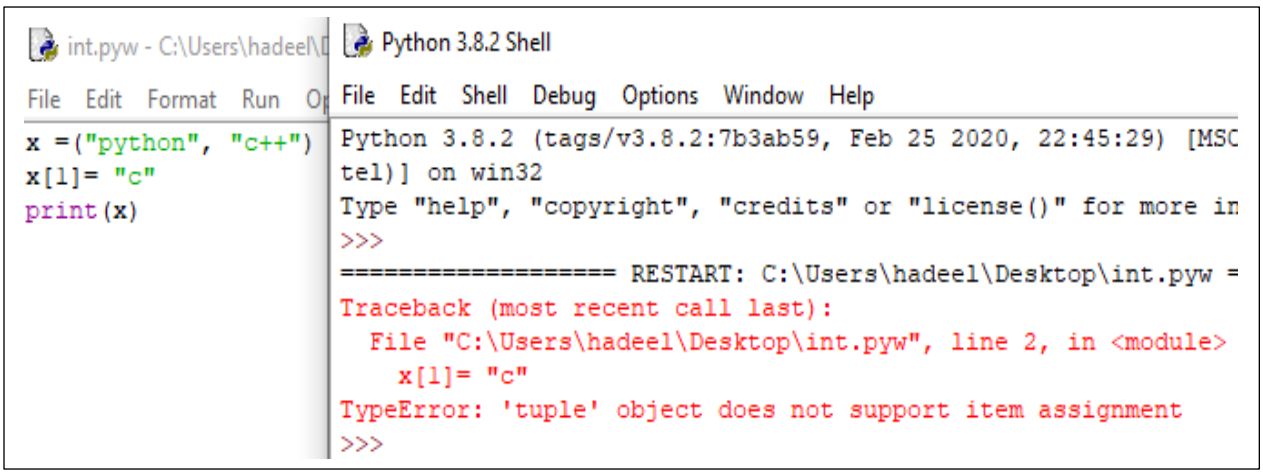




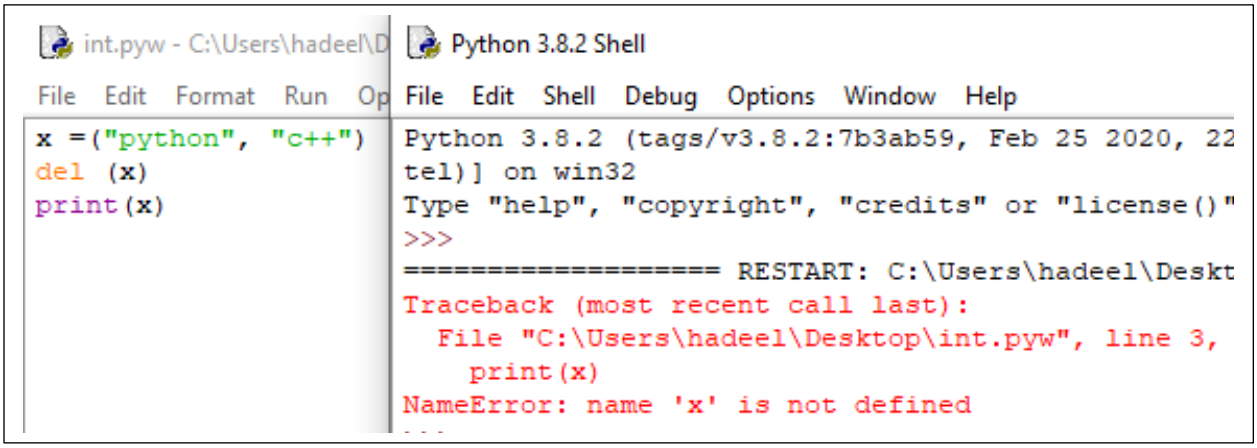
ولمعرفة طول tuple نستخدم الدالة len() كما هو الحال في list:



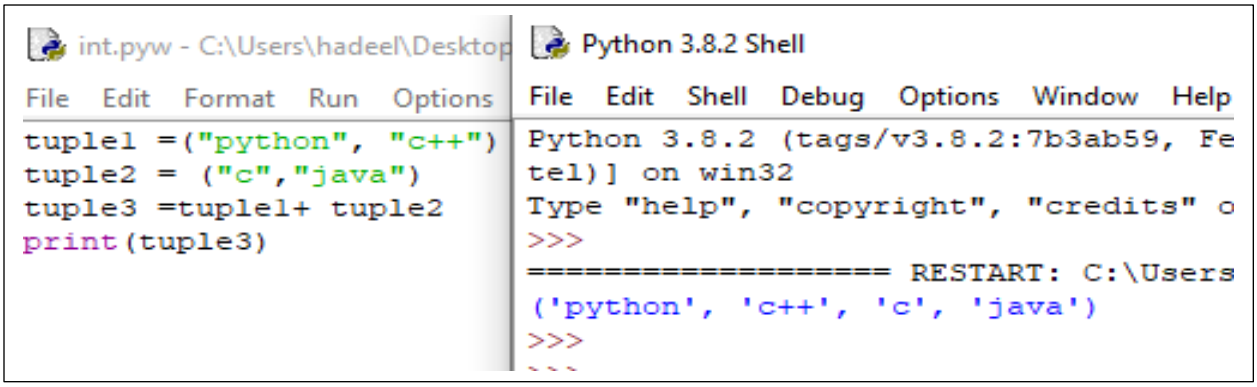
وللتأكد من ان tuple غير قابل للتغيير سنثبت ذلك (حيث سيظهر لنا ال error) :



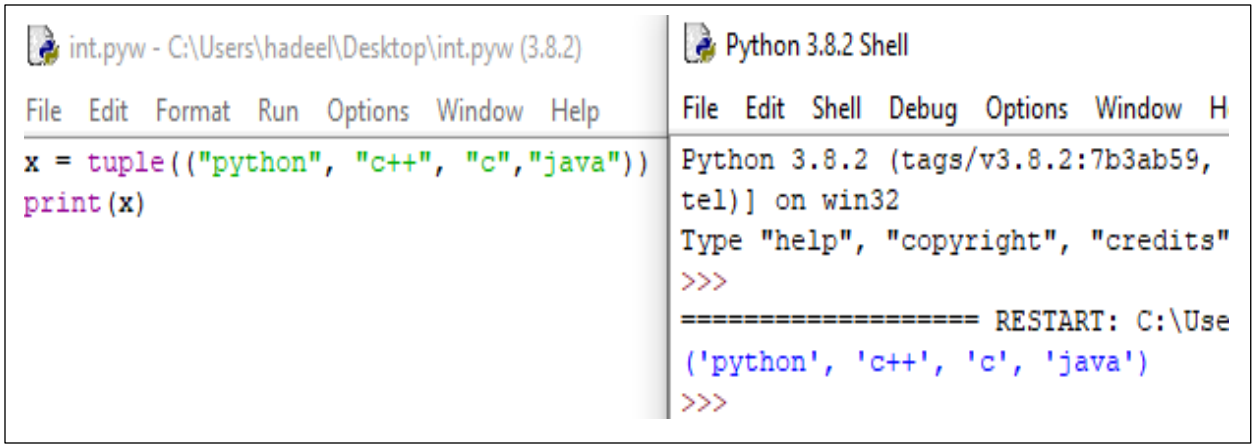
ولحذف عناصر نستخدم دالة ()del وذلك لحذف tuple بأكملها لأننا لا نستطيع حذف عنصر منها لأنها غير قابلة للتغيير



المثال اعلاه لا يوجد فيه خطا والسبب اننا قمنا بحذف tuple ولذلك بالنسبة للبايثون لا يوجد شيء يدعي x لذا ظهر لنا error. يمكن دمج tuple مع بعضهما البعض كالتالي:



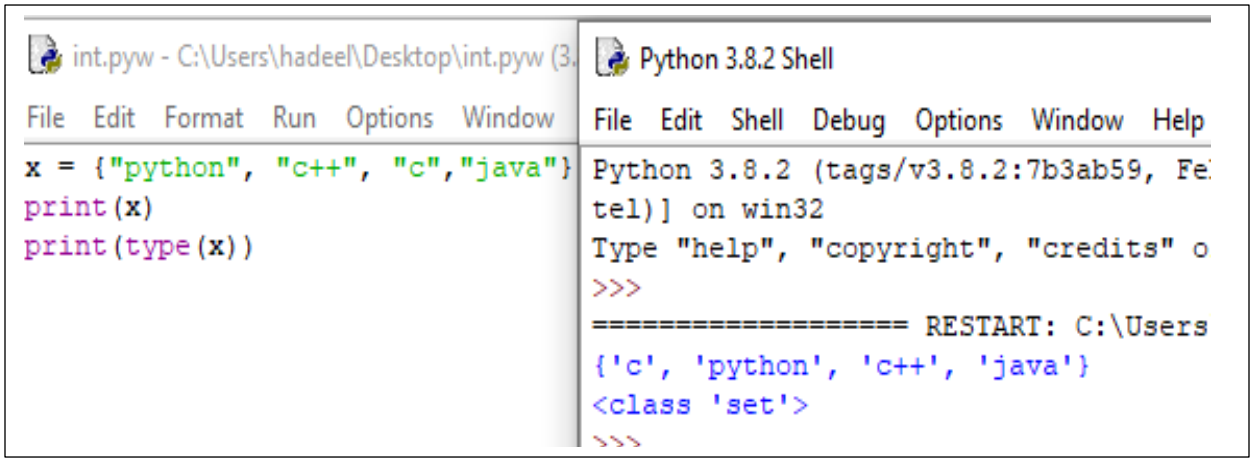
من الممكن أيضًا استخدام constructor في tuple حيث نلاحظ اننا استخدمنا الاقواس مرتين :



٥. Sets, frozenset

وهي مجموعات ليس لها حجم ثابت, و لا يمكن حذف قيمها , في لغة البايثون يتم كتابة المجموعات(Set) بأقواس متعرجة{}

يمكننا انشاء المجموعات كالاتي:

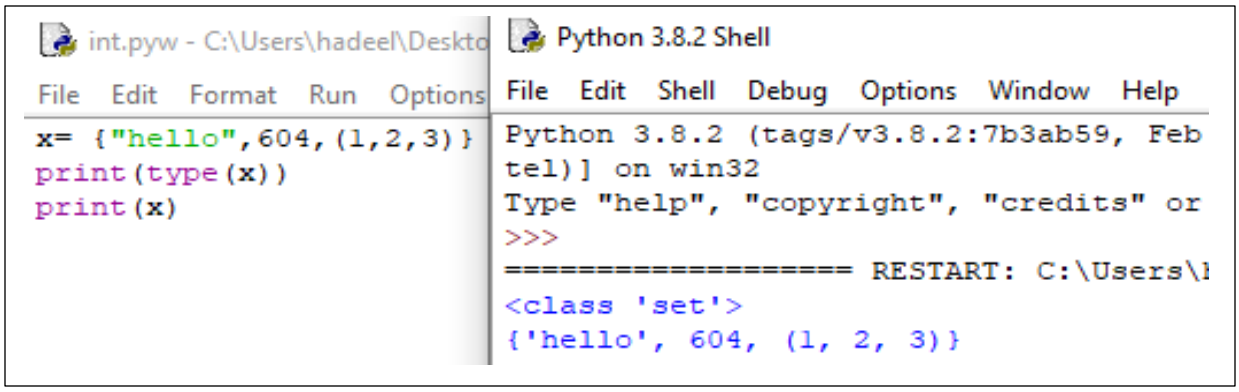


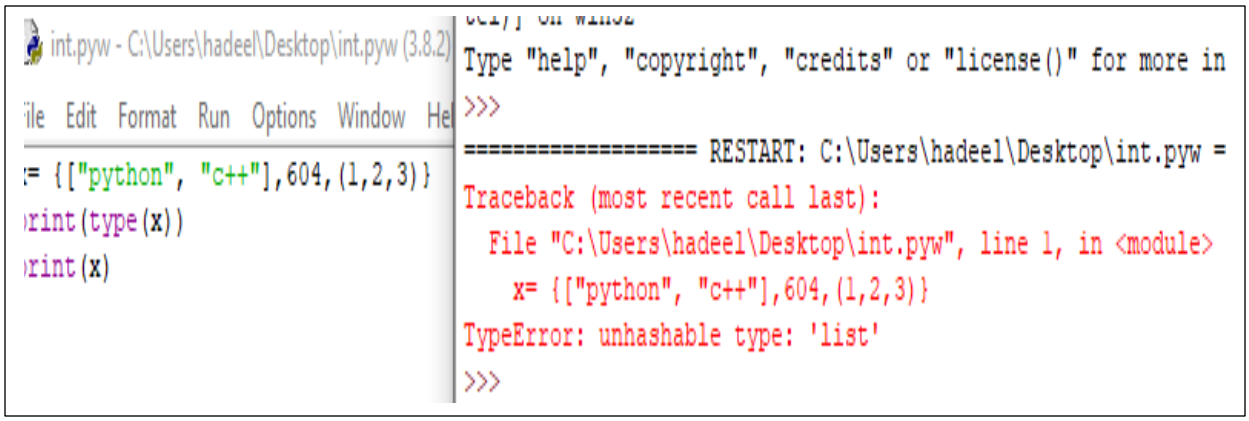
نلاحظ ان البايثون قد تعرف على المجموعة فقط بعد ان قمنا بتغيير الاقواس الى الاقواس المتعرجة .

ملاحظة مهمة جدا يجب الانتباه لها في المجموعات وهي:

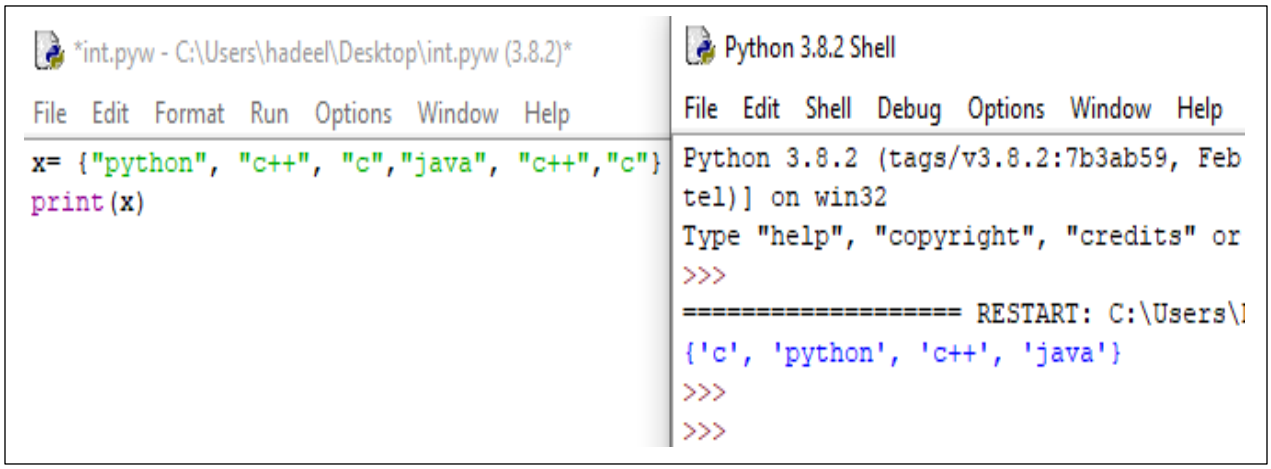
لا يمكنك الوصول إلى العناصر في المجموعة بالرجوع إلى فهرس ، وذلك لأن set غير مرتبة لذا ان العناصر ليس لها فهرس.

نستطيع تضمين: string ,number, tuple داخل المجموعة لكن لا يمكن تضمين list داخلها كالتالي:



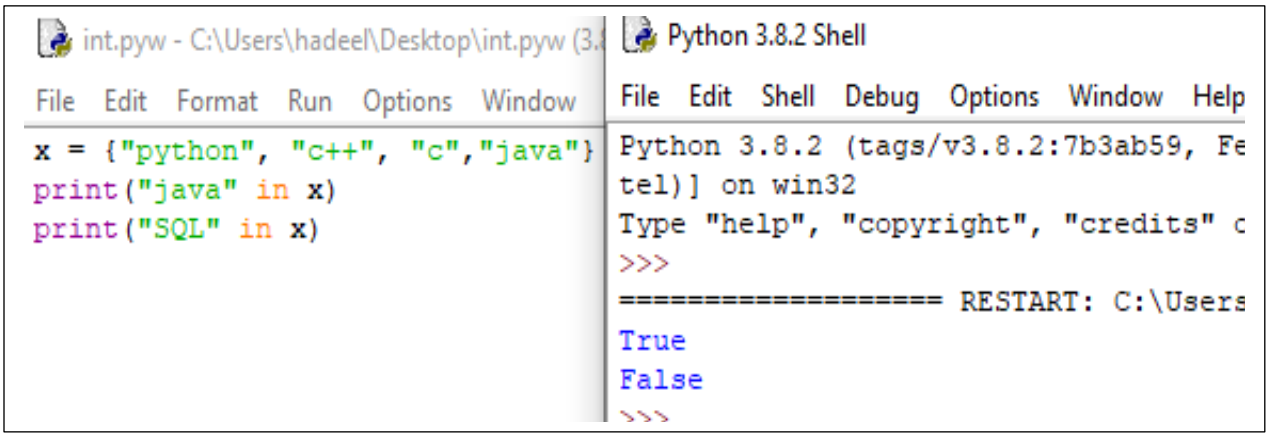


في المجموعة Set لا يمكن أن تحتوي على تكرارات(اي العناصر لا تتكرر داخل المجموعة)

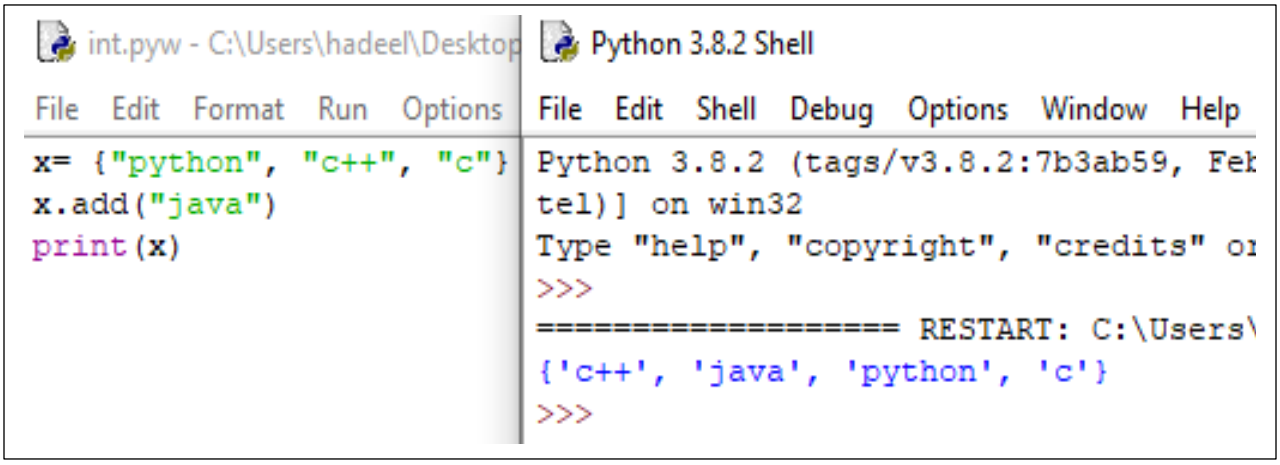


على الرغم من الترتيب العشوائي للعناصر داخل المجموعة الا ان العناصر المتكررة تم حذفها عند طباعة النتيجة .

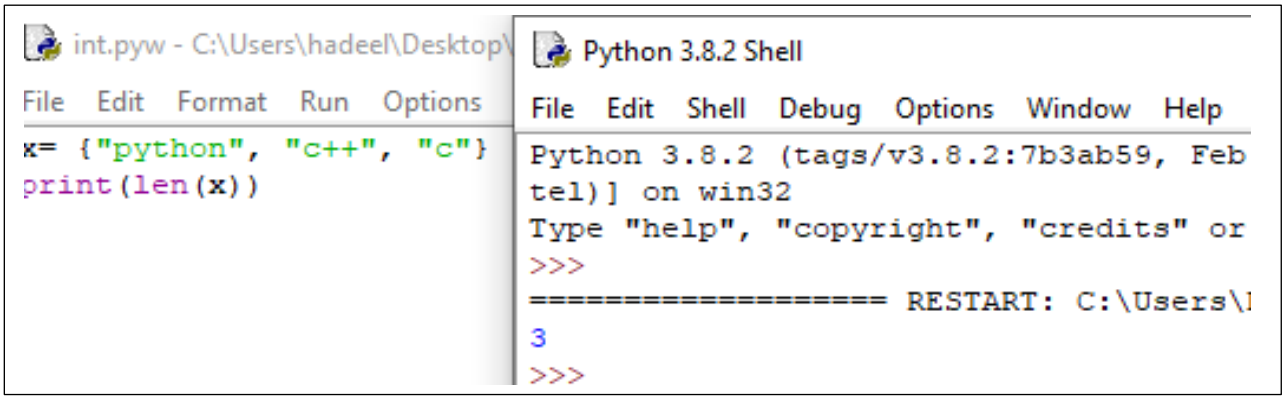
ايضا يمكننا التحقق داخل المجموعة مما إذا كان احد العناصر موجودًا في المجموعة ام لا من خلال (in) والاجابة ستكون فقط بنعم او لا



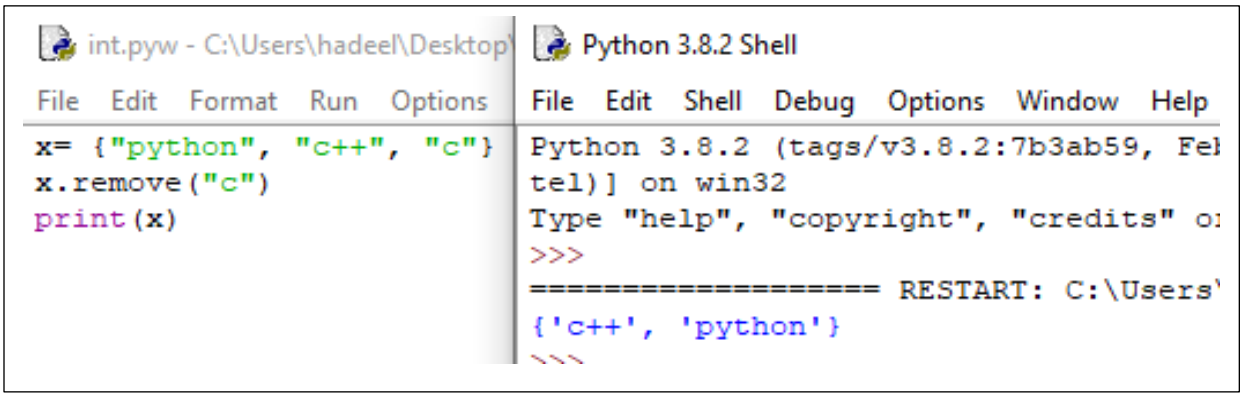
في المثال اعلاه نلاحظ ان الجافا(java) موجودة ضمن المجموعة فكانت النتيجة True اما SQL فبحث عنها داخل المجموعة ولم يجدها فأعطاها النتيجة False. ولإضافة عنصر واحد إلى مجموعة ، استخدم طريقة add().

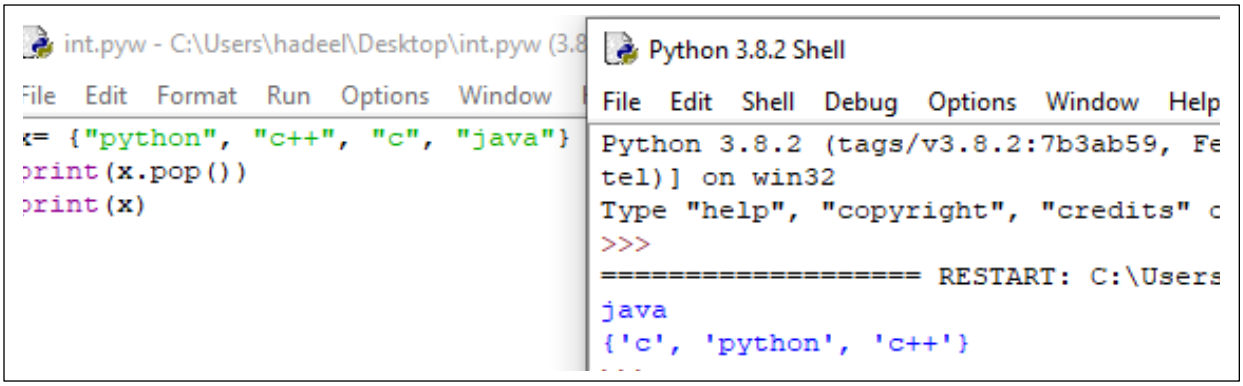


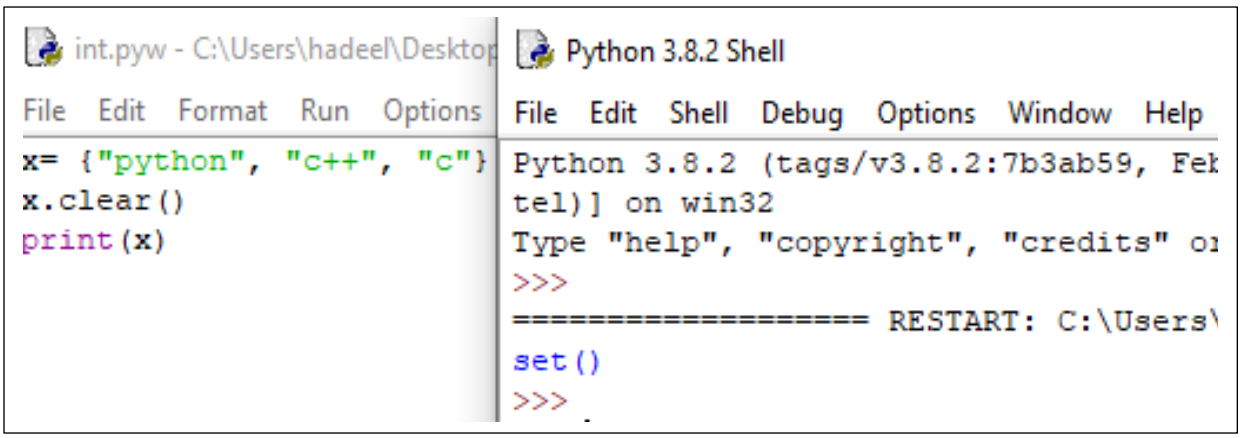
ولتحديد عدد العناصر الموجودة في المجموعة(set)، نستخدم الدالة المتعارف عليها والتي استخدمناها كثيرا جدا وهي ال() len.

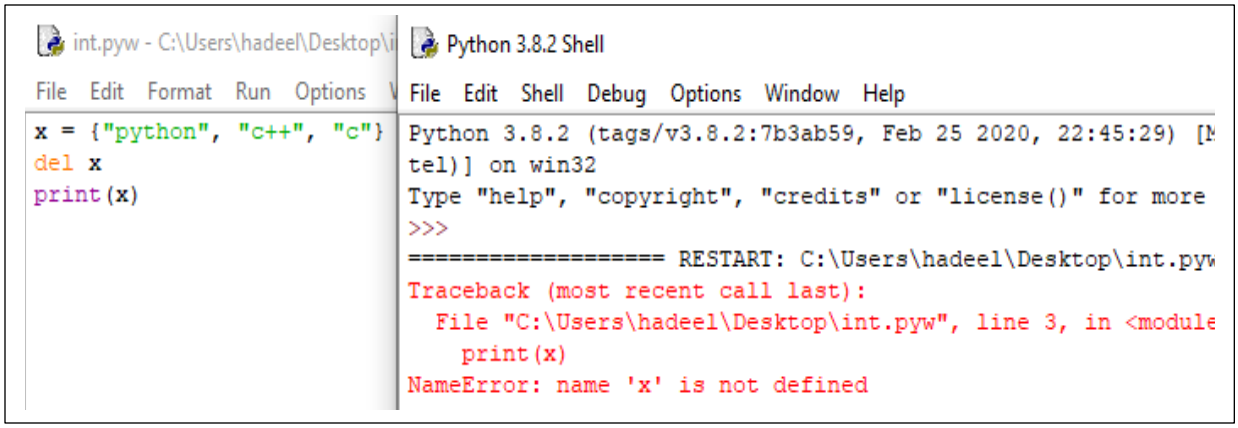


فضلا عن يمكننا استخدام الدوال السابقة وهي remove() في حال ان العنصر المراد حذفه موجود في المجموعة , ودالة pop() لإزالة العنصر الاخير الموجود في المجموعة , ودالة clear() لإفراغ المجموعة بأكملها , ودالة del() لحذف المجموعة وستكون الامثلة التالية للتطبيق العملي لتلك الدوال بالتتالي:

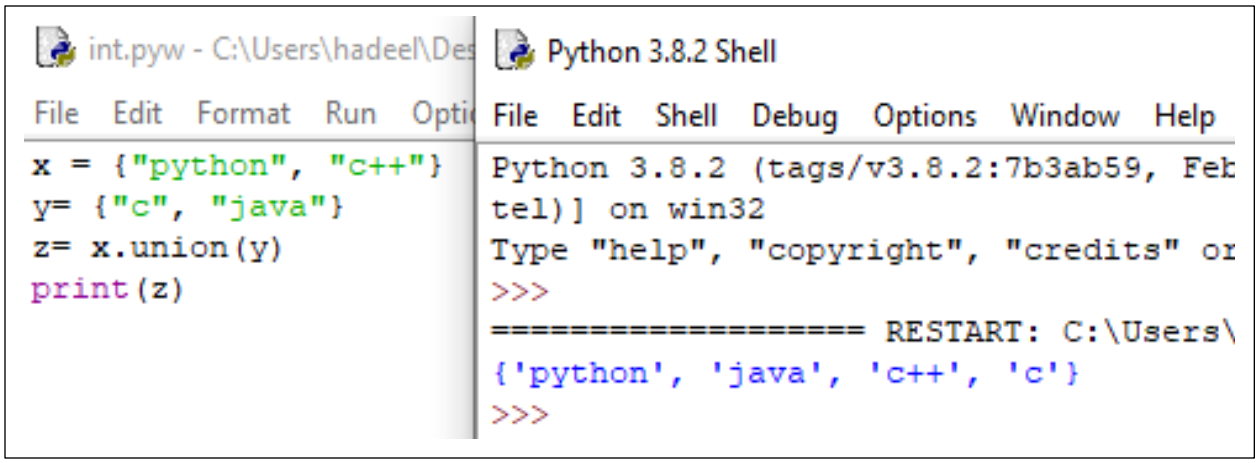




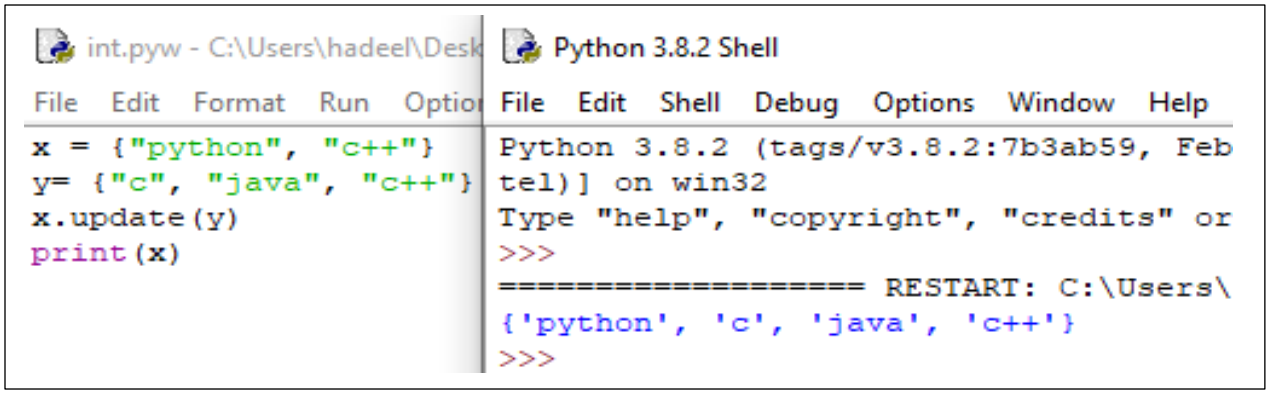




في مثال دالة del لم تتعرف البايثون على x وذلك لان تم حذفها قبل الطباعة لهذا لم تعد x موجودة . وبما اننا الان في المجموعة فمن المؤكد يمكننا عمل اتحاد بين مجموعتين من خلال الدالة union() على ان لا تتكرر العناصر :



ويمكن عمل تحديث لمجموعة مع مجموعة اخرى من خلال update() على ان لا تتكرر العناصر ايضا لان هذه القاعدة اساسية في المجموعات:



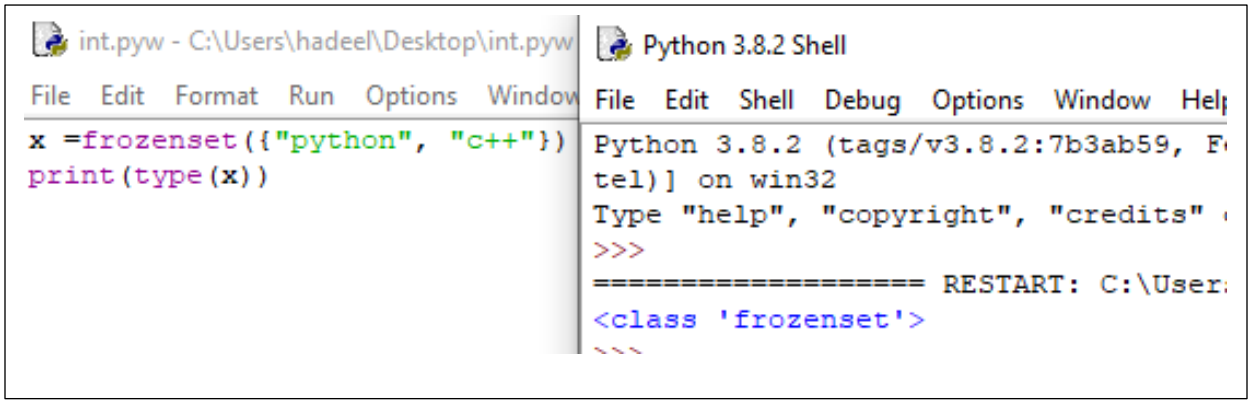
لقد انتهينا من المجموعات لكن يجب ان نوضح فكرة المجموعات المجمدة والتي تسمى ب(frozenset) :

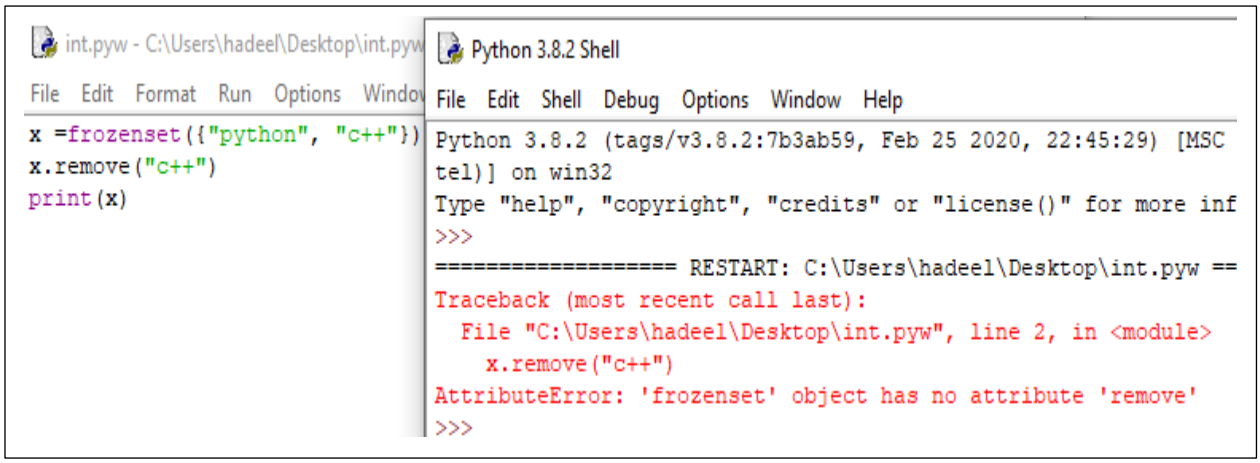
حيث تعد frozenset فئة جديدة لها خصائص مجموعة ، ولكن لا يمكن تغيير عناصرها ابدا بمجرد انشائها .
كما هو الحال في tuples حيث تعد قوائم ثابتة ، لذا فإن frozensets هي مجموعات ثابتة . من ناحية أخرى ،
غالبا ما تستخدم تجميد مجموعات frozensets كمفاتيح للقاموس(والذي سيتم شرحها لاحقا)

يمكن تطبيق الدوال التالية على هذا النوع من المجاميع :

copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union().

لكن لا يمكن تطبيق (add or remove) وذلك لاننا عرفناها سابقا بانها ثابتة (جامدة)





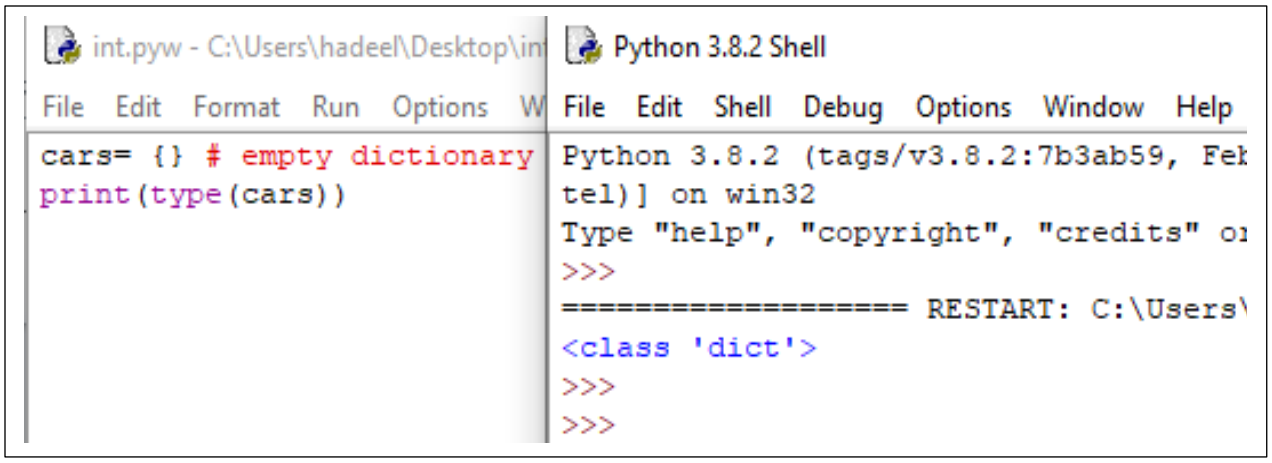
٦. القاموس Dictionaries

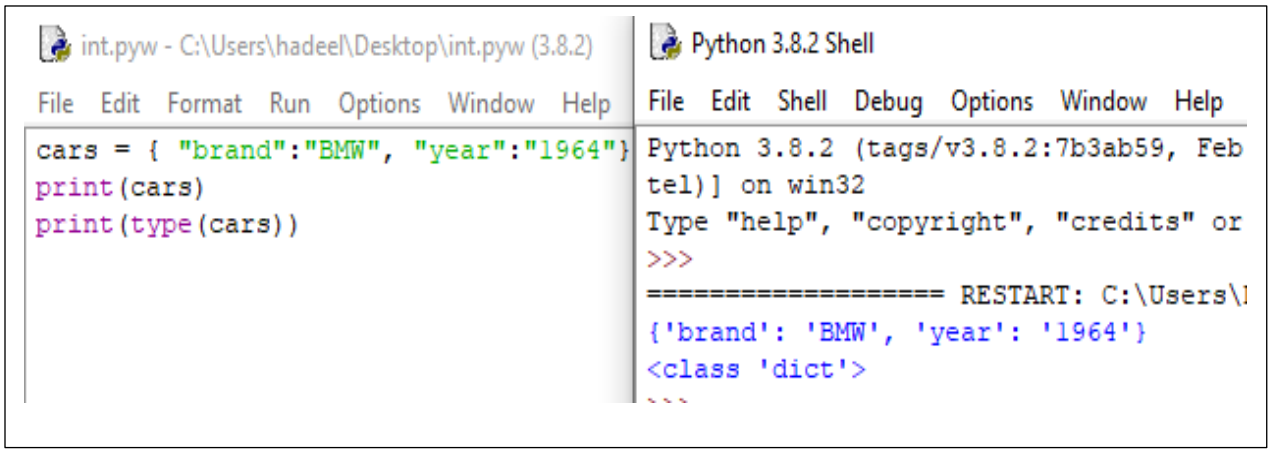
القاموس هو مجموعة غير منظمة وقابلة للتغيير والفهرسة. في البايثون تتم كتابة القواميس بأقواس متعرجة {} ولها ما يعرف ب (keys and values) حيث يفصل ما بين المفتاح والقيمة نقطتين متعامدتين (key: value) . وهذه المفاتيح والقيم مهمة جدا في القاموس .

هناك ملاحظة مهمة او قد تكون قاعدة في القاموس :

أن values يمكن أن تكون من أي نوع بيانات حيث من الممكن أن تتكرر ، على عكس keys فيجب ان تكون من النوع غير القابل للتغيير ويجب أن تكون فريدة.

ولإنشاء قاموس:





ولمعرفة عدد عناصر القاموس نستخدم ☺ دالتنا الشهيرة :

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File  Edit  Format  Run  Options  Window  Help
cars = { "brand":"BMW", "year":"1964"}
print(len(cars))
```

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
2
>>>
```

حيث تعد كل (قيمة ومفتاح) عنصر واحد لذا فان قاموسنا الذي أنشأناه يحتوي على عنصرين فقط .

والان سوف نتعلم كيفية طباعة value القيم لكل key بشكل منفرد كالاتي:

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2
File  Edit  Format  Run  Options  Window  H
nums = {1:"one",2:"two", 3:"three"}
print(nums)
print(nums[1])
print(nums[2])
print(nums[3])
```

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
{1: 'one', 2: 'two', 3: 'three'}
one
two
three
>>>
```

حيث تعد الطريقة اعلاه هي الاسهل لان هناك طريقة اخرى وهي من خلال دالة get() حيث نستطيع الحصول على قيمة المفتاح من خلالها:

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File  Edit  Format  Run  Options  Window  He
nums = {1:"one",2:"two", 3:"three"}
print(nums.get(1))
print(nums.get(2))
print(nums.get(3))
```

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
one
two
three
>>>
```

لطباعة كل values القيم الموجودة في القاموس من خلال دالة values():

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window Hel
nums = {1:"one",2:"two", 3:"three"}
print(nums.values())
```

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 20
tel)] on win32
Type "help", "copyright", "credits" or "lice
>>>
========== RESTART: C:\Users\hadeel
dict_values(['one', 'two', 'three'])
>>>
```

نلاحظ ان النتيجة ذاتها فقد حصلنا على قيم المفاتيح في القاموس بطرق مختلفة.

من جهة اخرى نستطيع طباعة مفاتيح القاموس من خلال keys()

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window He
nums = {1:"one",2:"two", 3:"three"}
print(nums.keys())
```

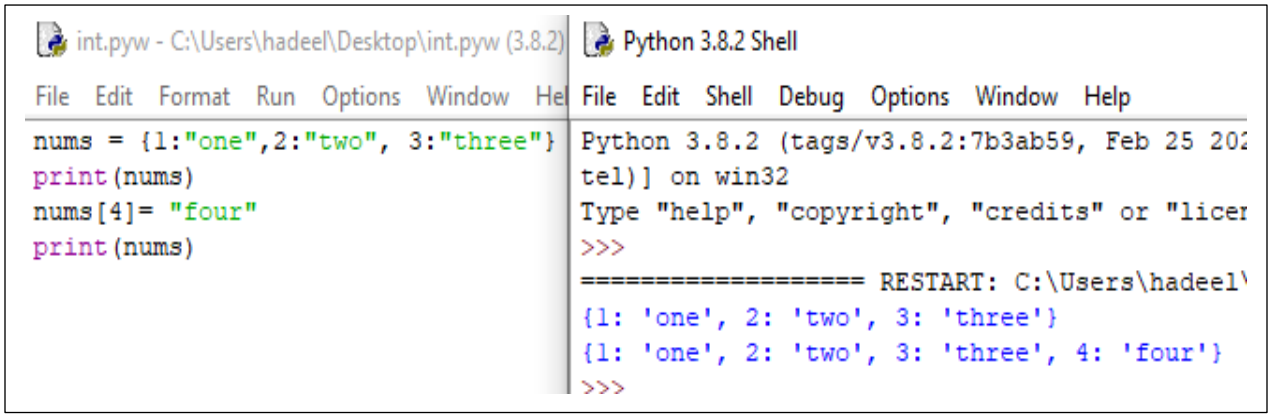
```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\h
dict_keys([1, 2, 3])
>>>
```

سناتي لمفهوم تغيير(تحديث) القيم ليس المفتاح لنركز معا يمكننا تغيير القيم فقط من خلال اضافة قيمة جديدة وهذا ما يسمى بتغيير القيم (Change Values):

```
int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window Help
nums = {1:"one",2:"two", 3:"three"}
nums[1]= "ten"
print(nums)
```

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, F
tel)] on win32
Type "help", "copyright", "credits"
>>>
========== RESTART: C:\Users
{1: 'ten', 2: 'two', 3: 'three'}
>>>
```

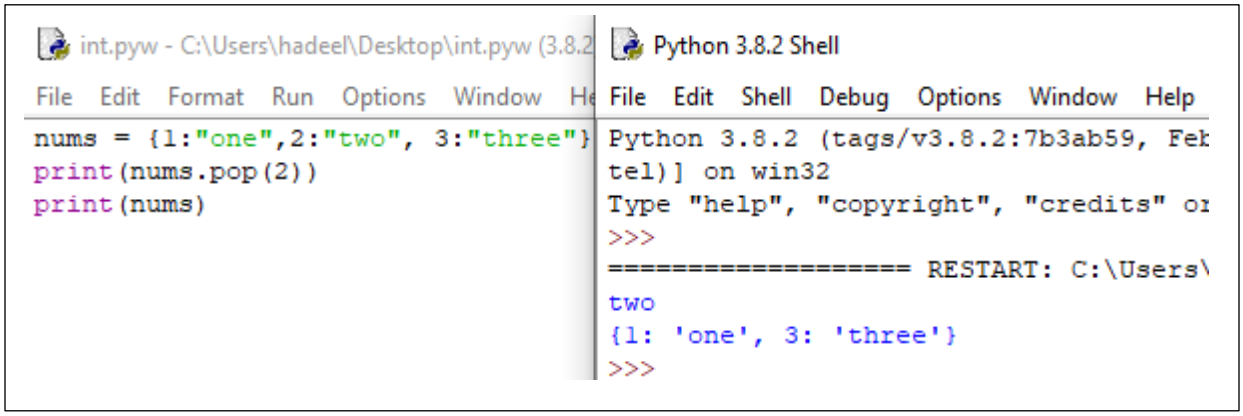
فضلا عن ذلك يمكن اضافة عنصر كامل للقاموس (العنصر يحتوي على مفتاح وقيمة) كالتالي:

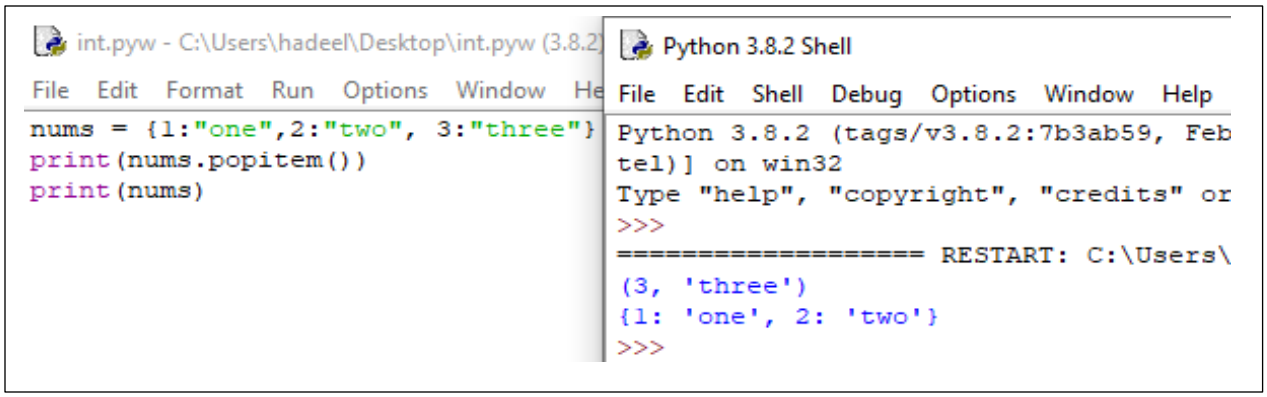


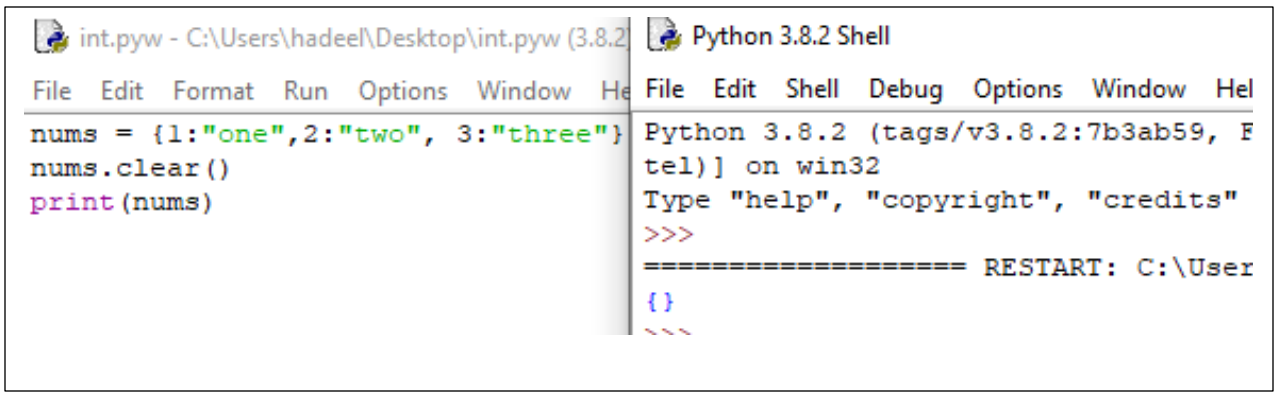
عند طباعة القاموس الاول كان يحتوي على ٣ عناصر لكن بعد الاضافة وطباعة القاموس لاحظنا ان القاموس احتوى على اربعة عناصر كما في المثال الموضح اعلاه.

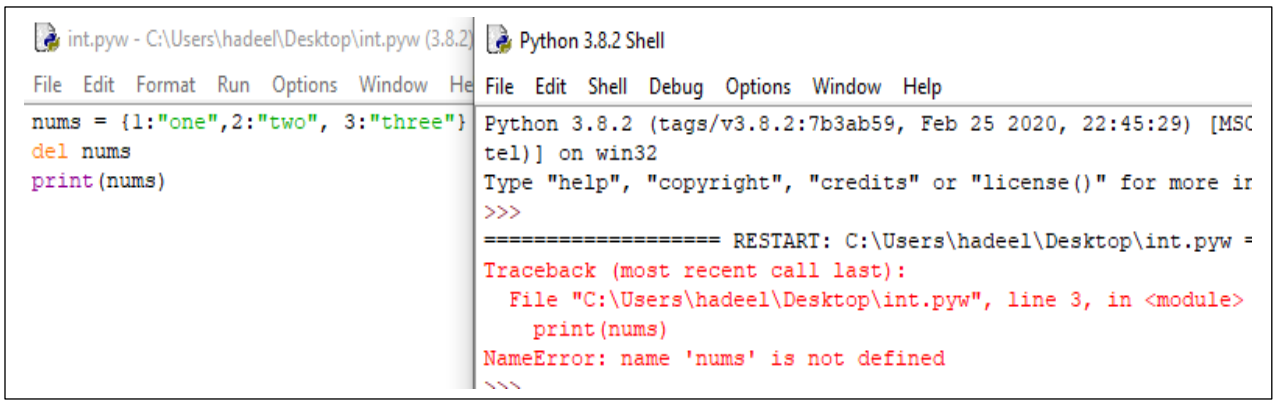
بعد أضافة عناصر هل يمكننا ازالة العناصر ياترى وكيف ؟

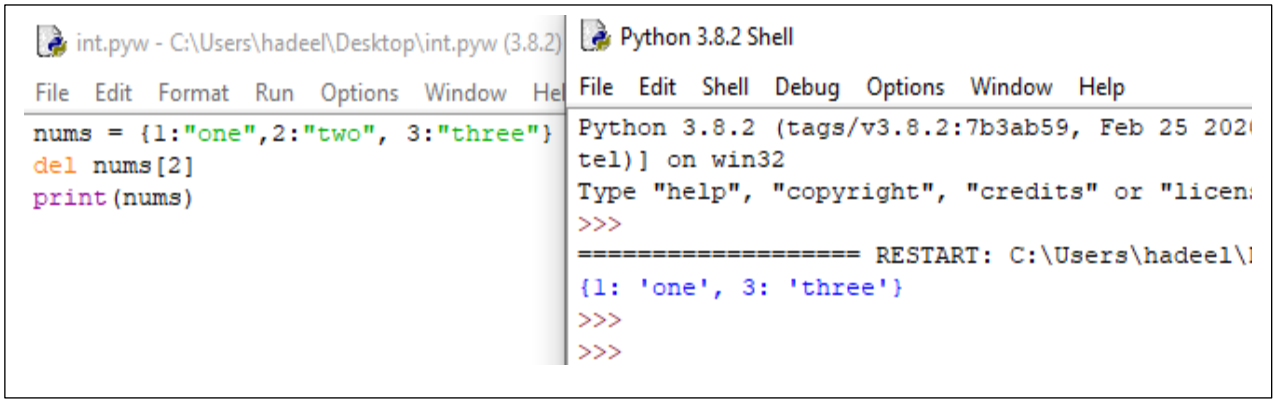
نعم يمكن إزالة العناصر من القاموس باستخدام دالة ال() pop حيث تؤدي هذه الطريقة إلى إزالة عنصر باستخدام المفتاح المقدم وإرجاع القيمة. كذلك يمكن استخدام طريقة () popitem لإزالة عنصر عشوائي (مفتاح: قيمة) من القاموس. اضافة الى ذلك يمكن إزالة جميع العناصر مرة واحدة باستخدام طريقة() clear واخيرا يمكننا أيضًا استخدام del لإزالة العناصر بشكل فردي لعنصر أو القاموس بأكمله.

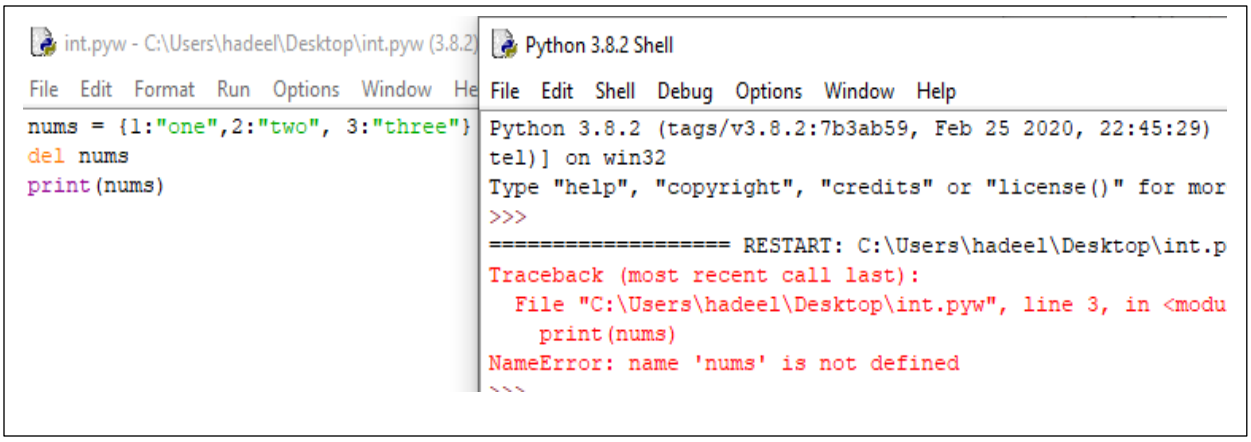



int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window He
nums = {1:"one",2:"two", 3:"three"}
print(nums.popitem())
print(nums)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
(3, 'three')
{1: 'one', 2: 'two'}
>>>

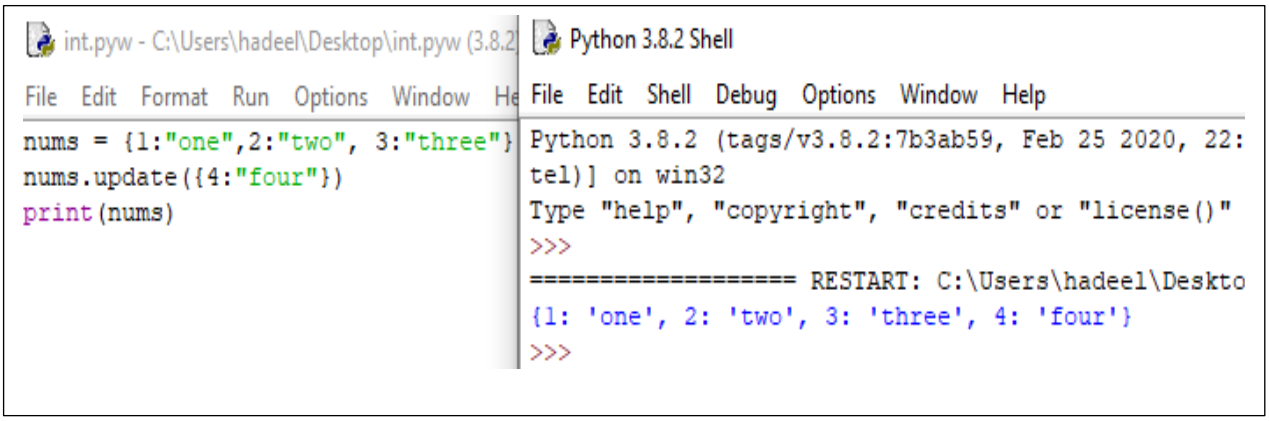

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window He
nums = {1:"one",2:"two", 3:"three"}
nums.clear()
print(nums)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Hel
Python 3.8.2 (tags/v3.8.2:7b3ab59, F
tel)] on win32
Type "help", "copyright", "credits"
>>>
========== RESTART: C:\User
{}
>>>


int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window He
nums = {1:"one",2:"two", 3:"three"}
del nums
print(nums)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more ir
>>>
========== RESTART: C:\Users\hadeel\Desktop\int.pyw =
Traceback (most recent call last):
File "C:\Users\hadeel\Desktop\int.pyw", line 3, in <module>
print(nums)
NameError: name 'nums' is not defined
>>>

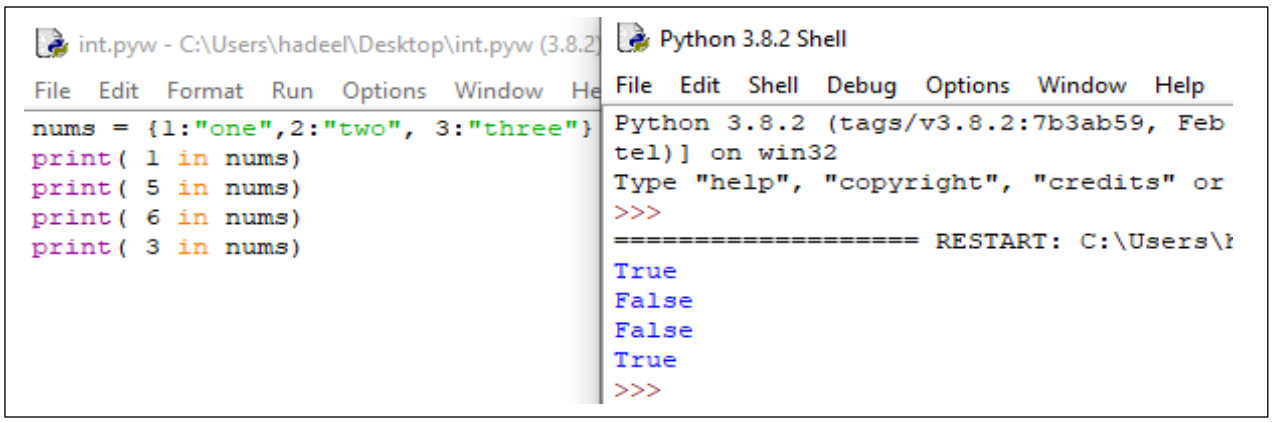

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)
File Edit Format Run Options Window He
nums = {1:"one",2:"two", 3:"three"}
del nums[2]
print(nums)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 202
tel)] on win32
Type "help", "copyright", "credits" or "licen
>>>
========== RESTART: C:\Users\hadeel\
{1: 'one', 3: 'three'}
>>>
>>>



كما تعلمنا ان del قامت بحذف القاموس بكلمه فلم يعد البايثون يستطيع ان يتعرف على nums و من اتجاه اخر نستطيع عمل تحديث للقاموس من خلال دالة update():



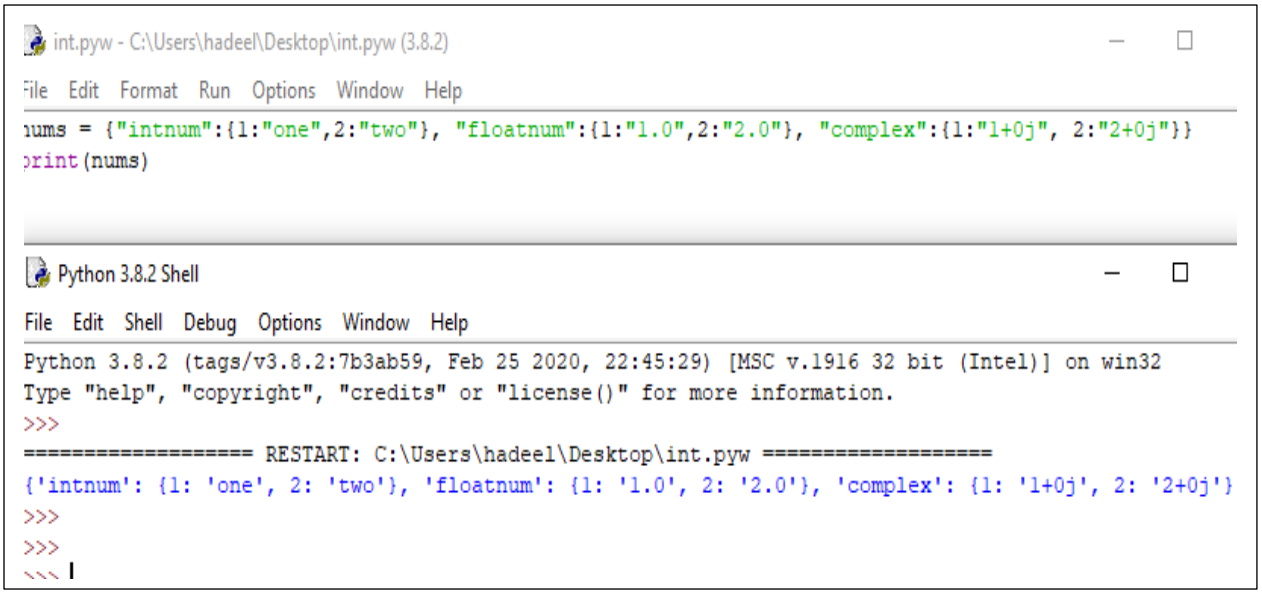
يمكننا اختبار ما اذا كان العنصر موجود داخل القاموس ام لا من خلال in وتسمى هذه الطريقة (Membership Test) وكما تعلمنا ان نتيجة in هي اما True او False:



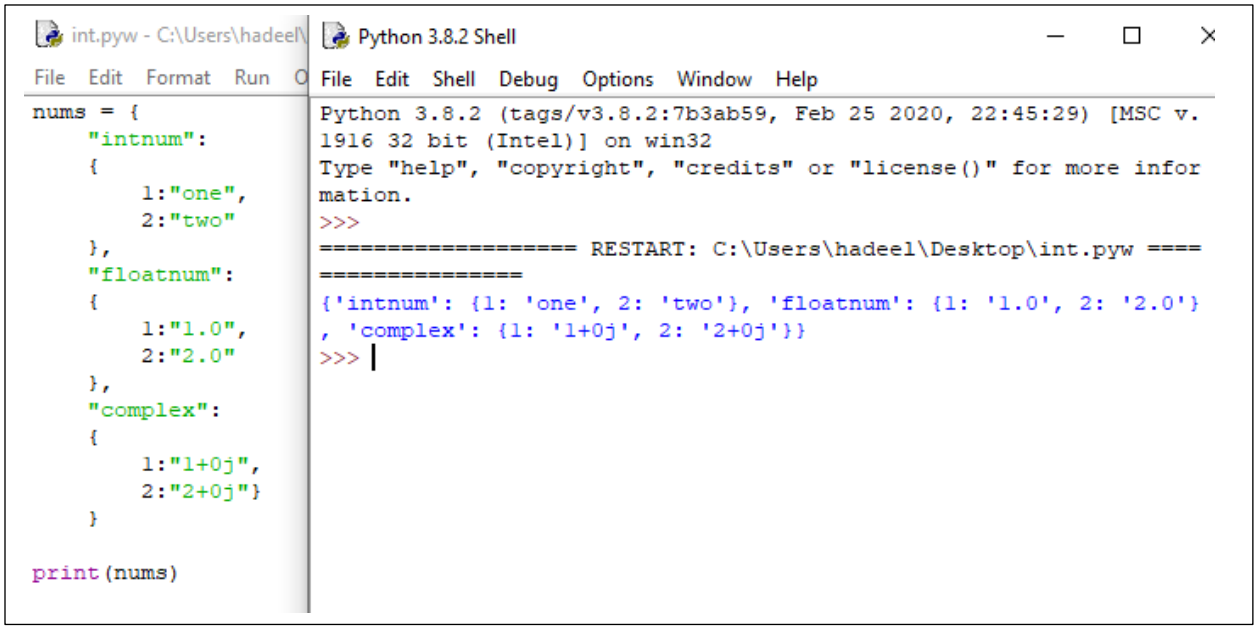
نلاحظ اعلاه انه تم الاعتماد على المفاتيح في معرفة ما اذا كان العنصر موجود اولا في القاموس ولا يتم الاعتماد ابدا على القيمة .

واخر ما نستطيع تعلمه في القواميس هي كيفية انشاء قواميس متداخلة(*Nested Dictionaries*) حيث يمكن أن يحتوي القاموس على العديد من القواميس داخل بمعنى اخر (قاموس كبير بداخله قواميس صغيرة) وهذه الحالة تسمى بالقواميس المتداخلة.

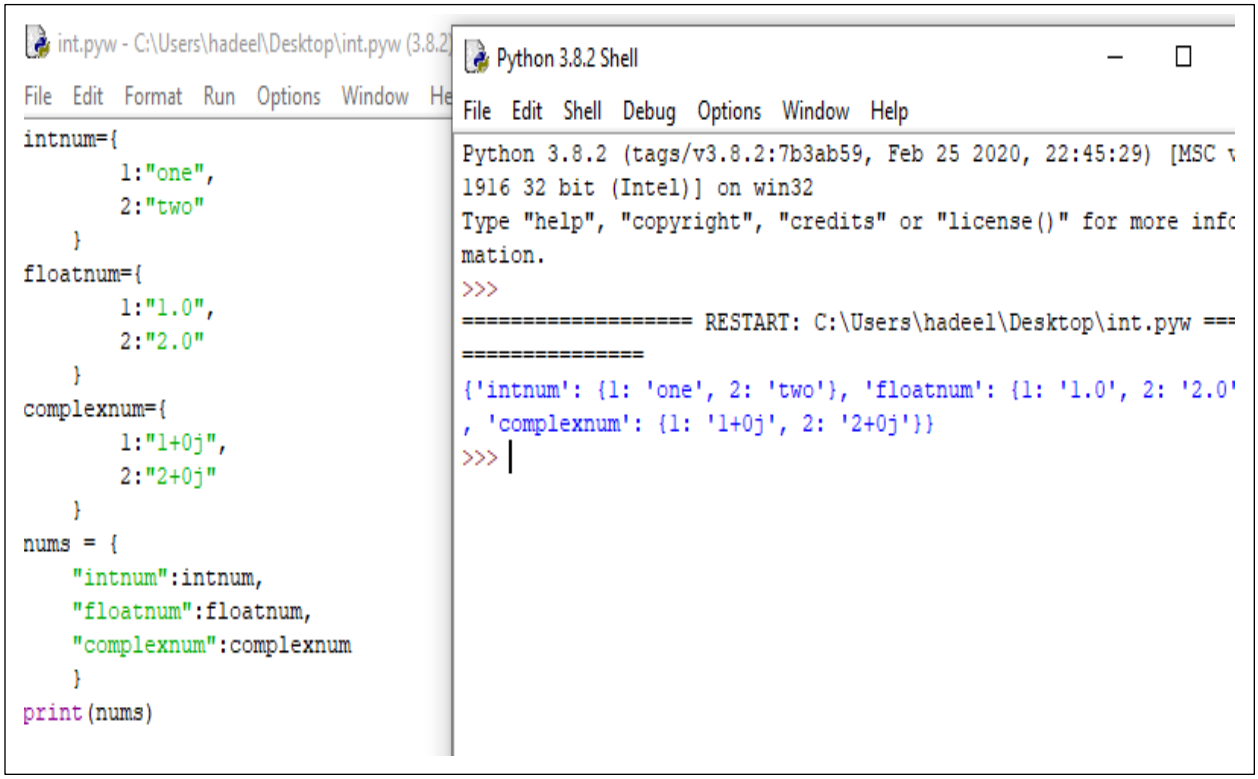
يمكن انشائها بطريقتين الاولى هي ان القاموس يحتوي بداخله على ثلاث قواميس كالتالي:



يمكن ترتيبها كالتالي بطريقة اوضح ويمكن قراءتها بسهولة :

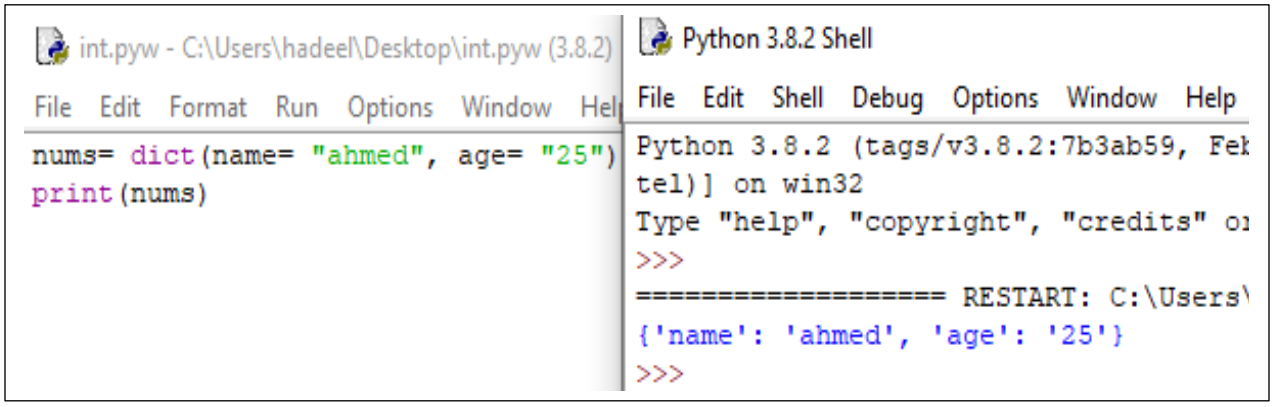


اعتقد ان فكرة القواميس المتداخلة اصبحت اكثر وضوحا. اما الطريقة الثانية دمج ثلاث قواميس موجودة في قاموس واحد كالمثال التالي:



ولكن بقي لدينا موضوع اخير في القواميس وهو Constructor للقاموس حيث يمكن استخدام Constructor لأنشاء قاموس جديد:

ملاحظة صغيرة في المثال ادناه يجب الاستغناء عن النقطتين المتعامدتين(:) واستبدالها بالفارزة(,)نلاحظ ان بعد التنفيذ شكل القاموس طبيعي بنقطتين متعامدتين.

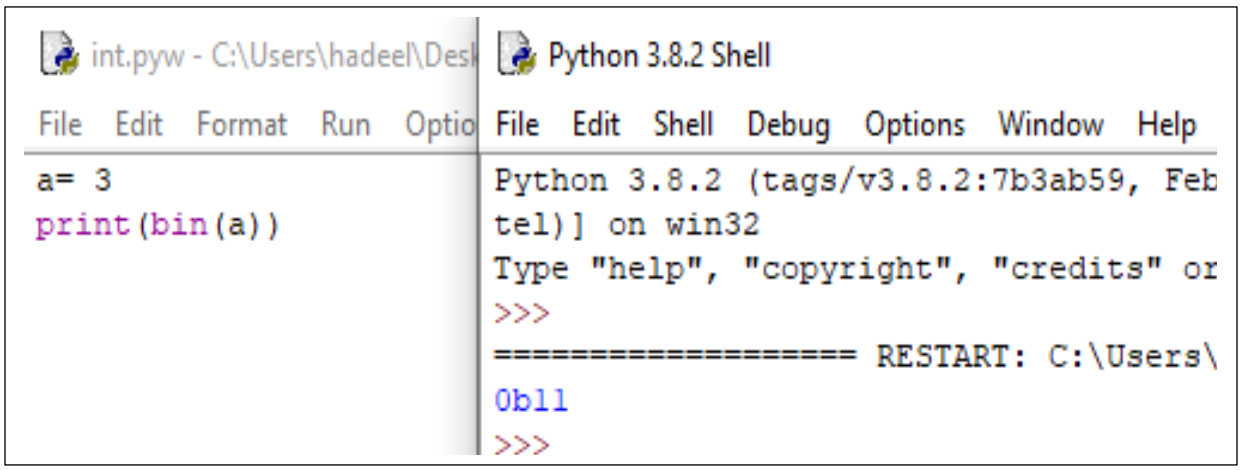


والان انتهينا من القاموس اتمنى ان يكون الشرح وافيا لكم

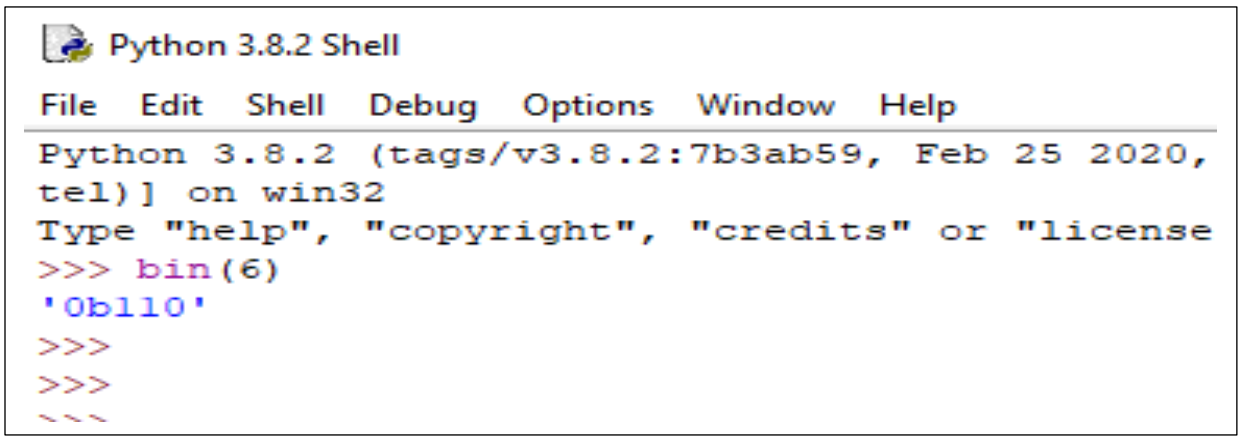
في اغلب المراجع والدورات يكتفون بهذا القدر من Data Types لكن يوجد ايضا من يضيفها الى انواع البيانات و لا ضير ان نستبين ما هو الثنائي binary ونتعرف كيفية تعامل البايثون معه.

٧. الثنائي Binary

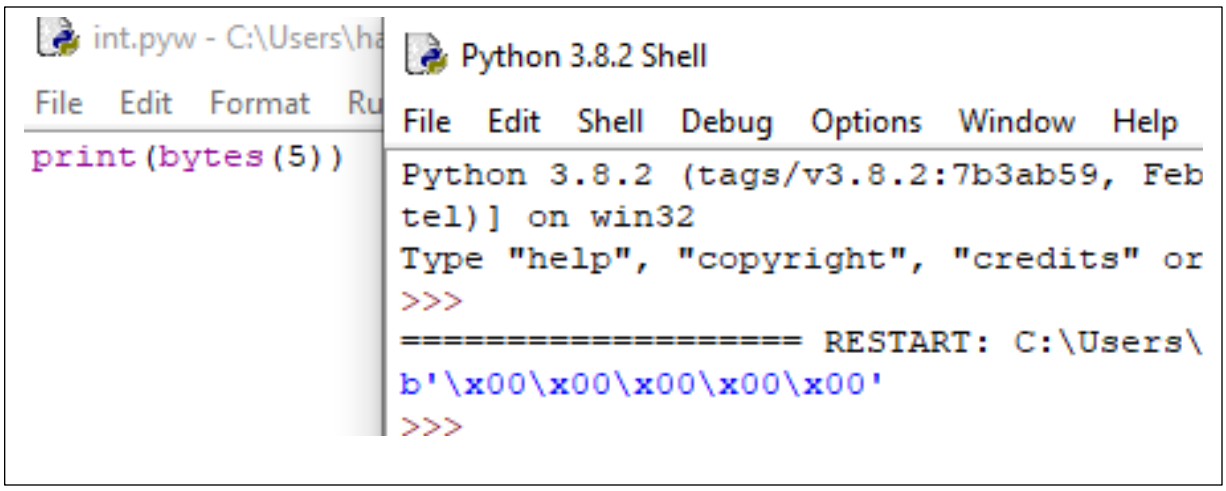
ان النظام الثنائي مهم جدا في اي لغة برمجية لذا فان البايثون تستخدم دالة bin() لتحويل الاعداد الى Binary وسناتي لشرح bytes و bytesarray بالتسلسل...لكن بداية نوضح دالة bin في مثال بسيط:



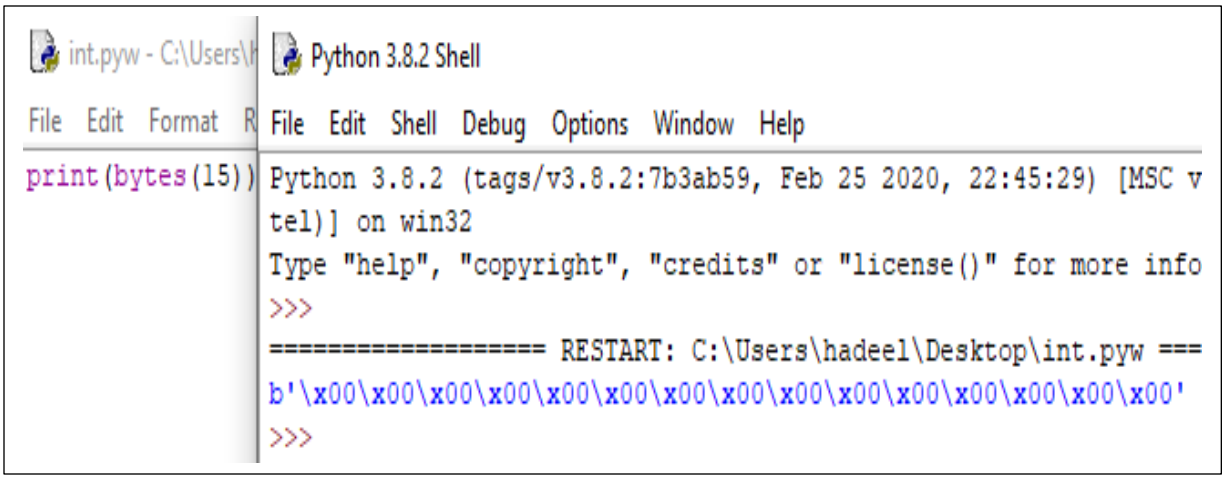
نلاحظ اعلاه انه تم تحويل الرقم ٣ الى نظام الثنائي وتم اضافة حرف b اختصارا (binary). مثال اخر:



والان لنبدأ ب (Bytes) هي عبارة عن bytes بايتات متسلسلة، مفردة وثابتة (immutable) اي انها غير قابلة للتغيير لذا فأنها تعطي النتيجة بسلسلة بمدى يبدا من 0 الى 256 اي (0 <=x < 256)، حيث يمكن تقديم bytes باستخدام الدالة bytes() :



نلاحظ في المثال اعلاه عند استخدم الدالة bytes() انها اعطت خمس بايتات لنغير الخمسة الى خمسة عشر سيكون كالاتي :

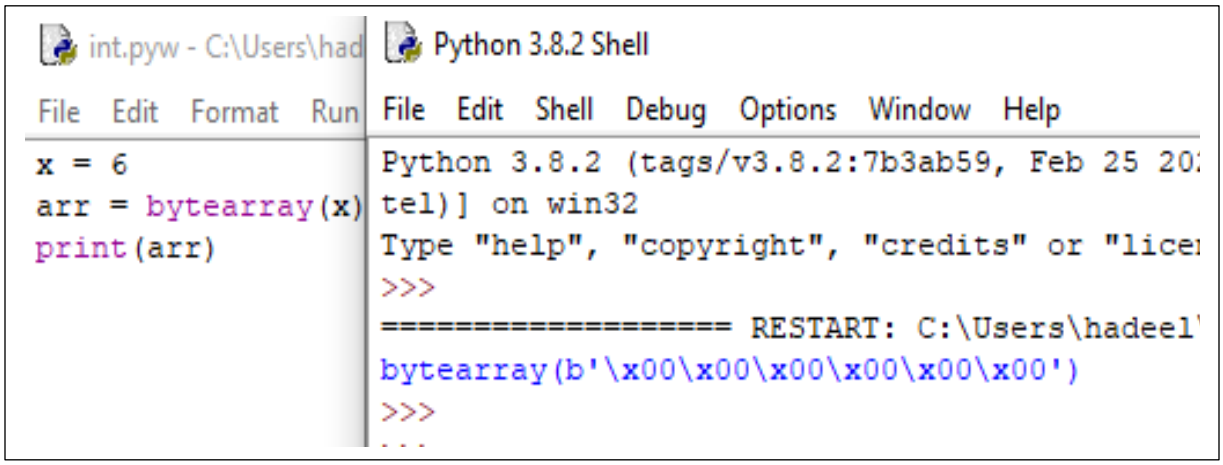


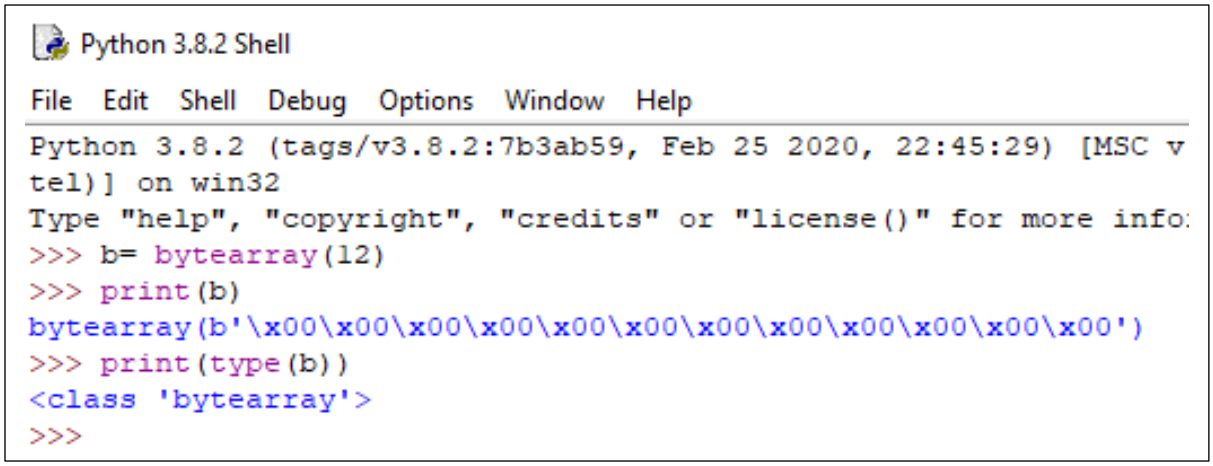
هناك العديد من الدوال التابعة لل bytes منها: bytes.fromhex() ,bytes.count() ,bytes.decode()

bytes.find() ,bytes.index() ,bytes.join() ,bytes.rfind() ..... الخ. ان البايتات bytes غالبا ما تكون صالحة للاستعمال مع البيانات ذات الترميز الشهير ASCII .

يمكن ان نمر بسرعة ايضا الى Bytesarray:

هي مشابهة للبايتات لكنها مختلفة بانها قابلة للتغيير وليست ثابتة كالبايتات حيث ان المثال ادناه يوضح هذه الدالة:





حيث ان هناك العديد من الدوال التابعة لها منها: (bytearray.fromhex)،(bytearray.hex) ، (bytearray.count)،(bytearray.find)،(bytearray.index).....الخ

الان انتهينا من انواع البيانات في البايثون

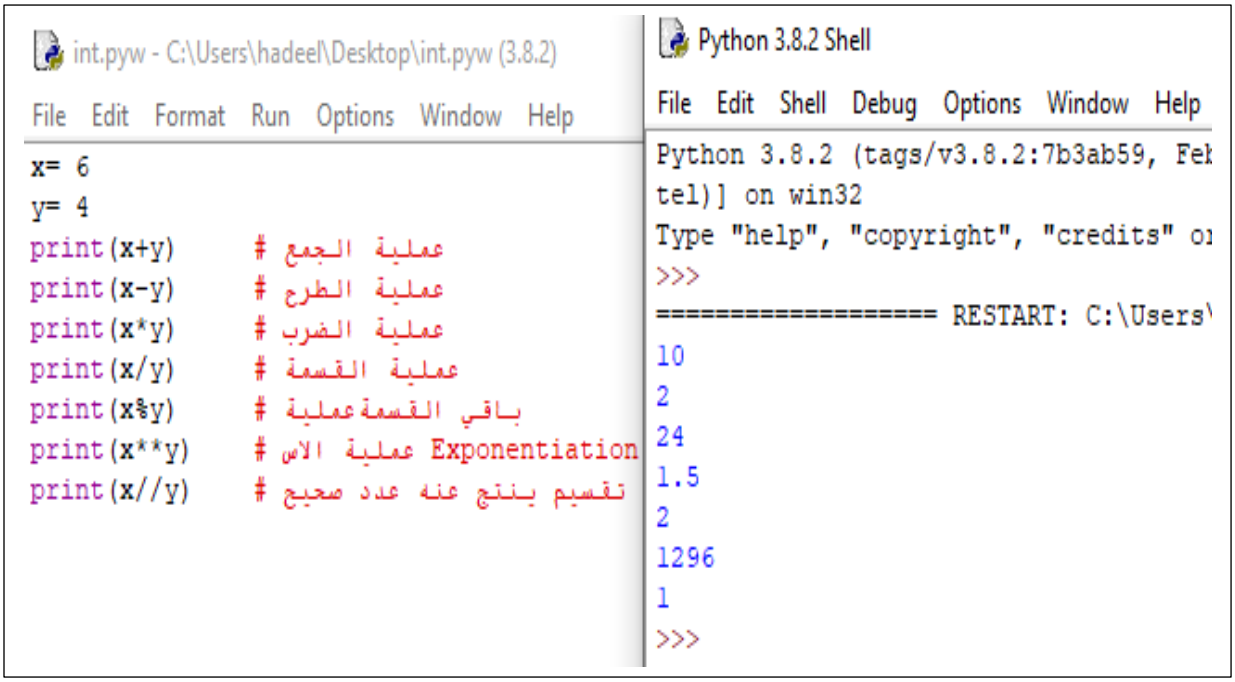
Python Operators

العمليات في البايثون

تعرف العمليات على انها رموز خاصة تقوم بإجراء العمليات الحسابية أو المنطقية. بحيث يجب ان نعلم ان القيمة التي يعمل عليها عامل التشغيل تسمى المعامل. وان العمليات في البايثون مهمة وضرورية جدا لذا يجب ان نتعرف على الانواع المختلفة وكيفية استخدامها في python

تقسم العمليات الى :

- العمليات الحسابية (Arithmetic operators): يتم استخدام العمليات الحسابية مع القيم(values) الرقمية لإجراء العمليات الحسابية الشائعة وسيكون المثال التالي موجز للعمليات الرياضية المعروفة :



في المثال اعلاه هناك سبعة انواع من العمليات الحسابية الاساسية وهي متواجدة في اغلب اللغات البرمجية .

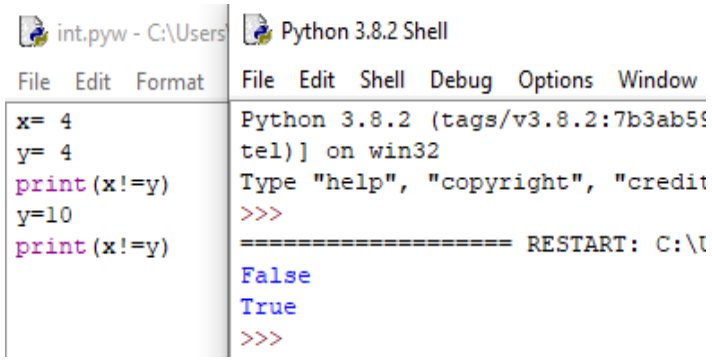
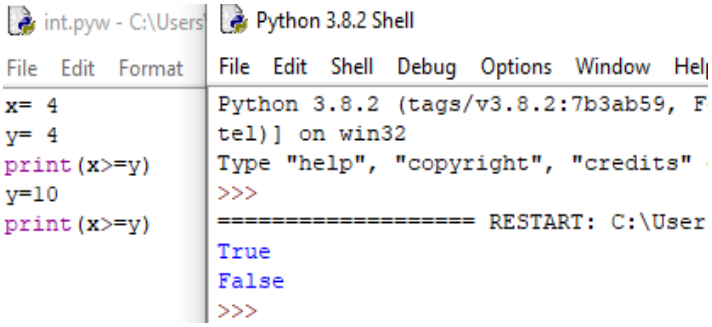
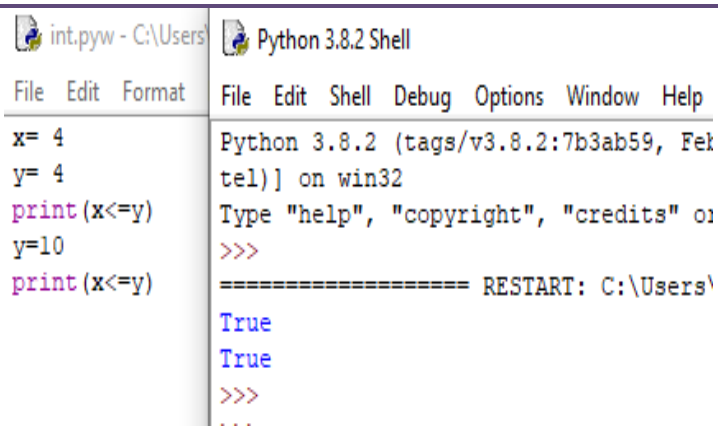
- **عمليات المقارنة(Comparison operators)**

المقارنة هي بين شيئين متماثلين وتكون النتيجة دائما بصح (True) او خطأ(False) بحسب الشرط .

باختصار ان عمليات المقارنة تستخدم عوامل المقارنة لمقارنة القيم

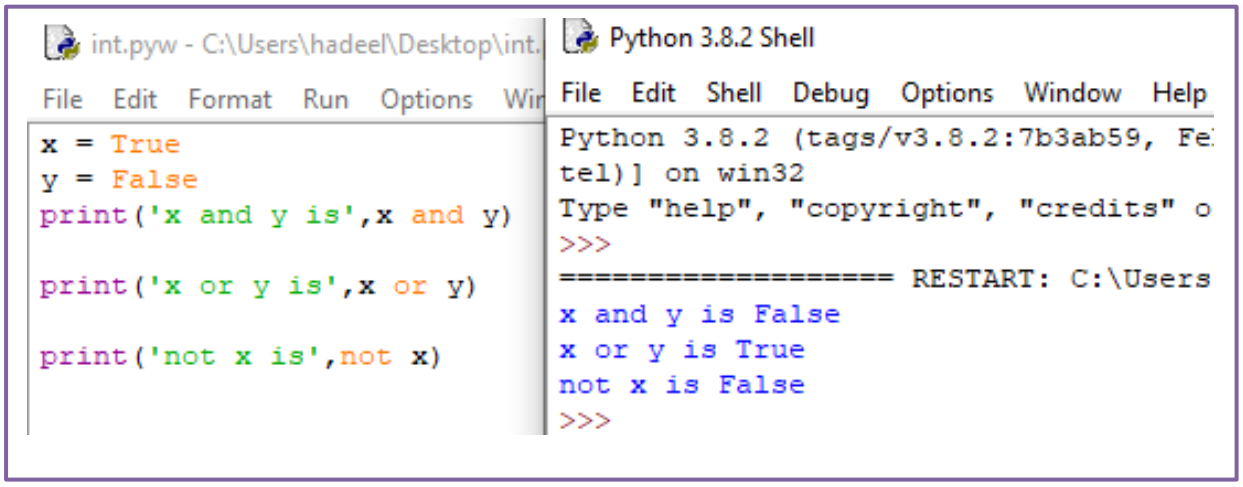
وفي الجدول ادناه العمليات(operators) وامثلة عن كيفية تمثيلها في البايثون :

العمليات operators	معنى ال operator	مثال في البايثون
>	أكبر من إذا كانت القيمة الأيسر أكبر من اليمين فالنتيجة True	int.pyw - C:\Users\hadeel File Edit Format Run x= 6 y= 4 print("x>y=", x>y) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:\Users\ x>y= True >>>
<	اصغر من إذا كانت القيمة الأيسر اصغر من اليمين فالنتيجة True	int.pyw - C:\Users\hadee File Edit Format Run x= 4 y= 6 print("x<y=", x<y) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:\Users\ x<y= True >>>
==	التساوي اذا كان المعاملين متساويين فان النتيجة True	int.pyw - C:\Use File Edit Format x= 4 y= 4 print(x==y) y=10 print(x==y) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:\Users\ True False >>>

	لا يساوي اذا كان المعاملين غير متساويين فان النتيجة True	!=
	أكبر او يساوي من إذا كانت القيمة الأيسر أكبر او تساوي القيمة من اليمين فالنتيجة True	>=
	اصغر او يساوي من إذا كانت القيمة الأيسر اصغر او تساوي القيمة من اليمين فالنتيجة True	<=

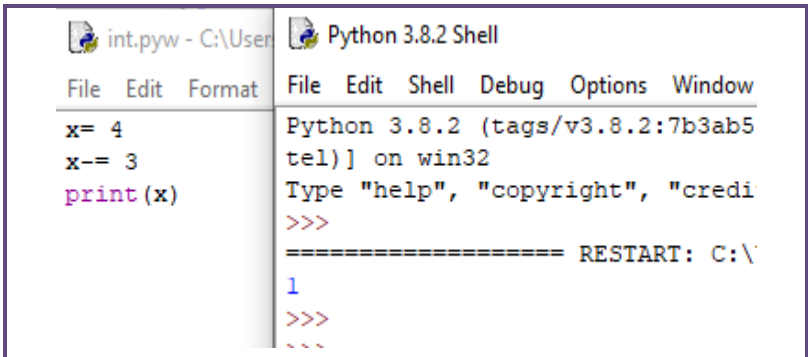
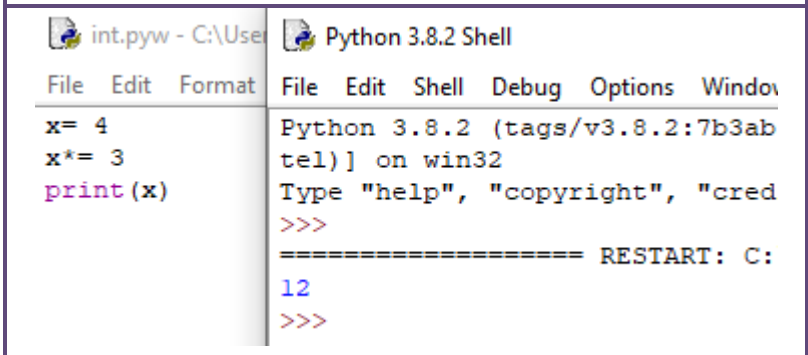
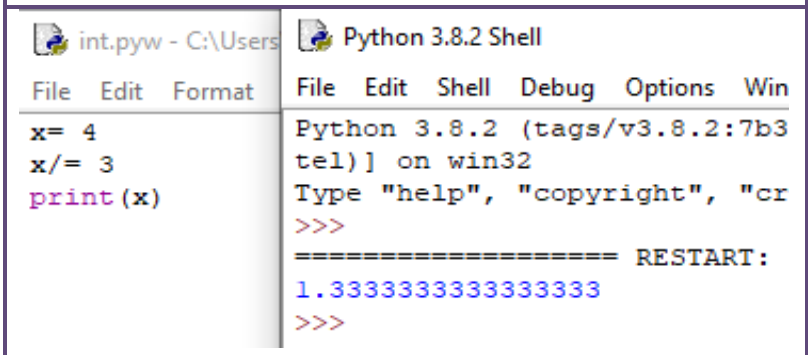
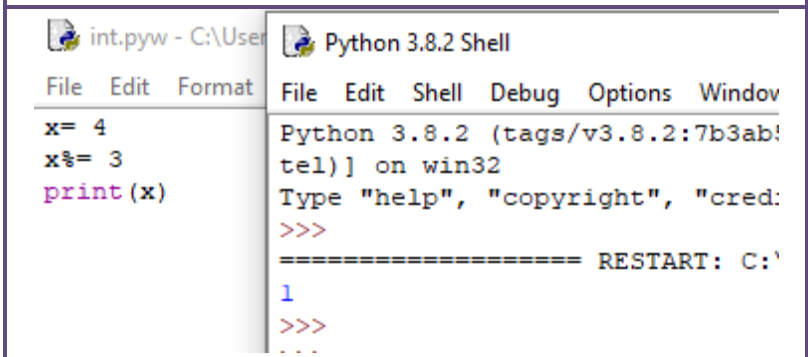
- **العمليات المنطقية(Logical operators):** يتم استخدام العمليات التشغيل المنطقية لدمج العبارات الشرطية. العمليات المنطقية هي : and , or , not ويمكن تمثيلها كالتالي:

العملية	عملها	مثال في البايثون
and	تكون النتيجة True اذا كان المعاملين True او الشرطين صحيحين	int.pyw - C:\Users\hadeel\Desktc File Edit Format Run Options x = 6 print(x > 4 and x < 10) Python 3.8.2 Shell File Edit Shell Debug Options Window Help Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb tel)] on win32 Type "help", "copyright", "credits" or >>> ========== RESTART: C:\Users\ True >>> >>>
or	تكون النتيجة True اذا كان احد المعاملين True او احد الشرطين صحيحين	int.pyw - C:\Users\hadeel\Des File Edit Format Run Optio x = 4 print(x > 3 or x < 3) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab59, tel)] on win32 Type "help", "copyright", "credits >>> ========== RESTART: C:\Us True >>>
not	عكس العملية بمعنى اذا كان المعامل False فالنتيجة True	int.pyw - C:\Users\hadeel\Desktop\ File Edit Format Run Options x = 4 print(not(x > 3 or x < 3)) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "credi >>> ========== RESTART: C:\ False >>>

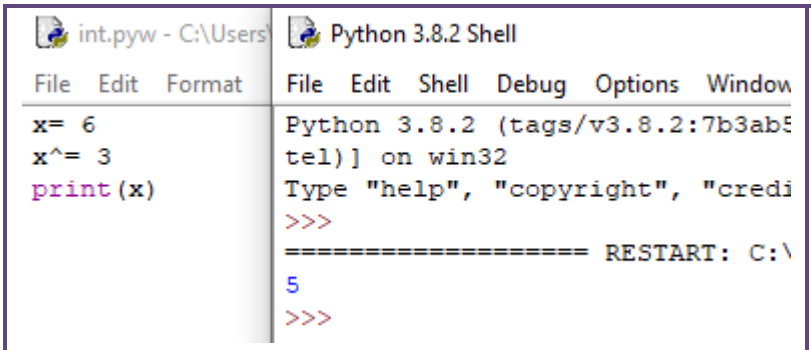
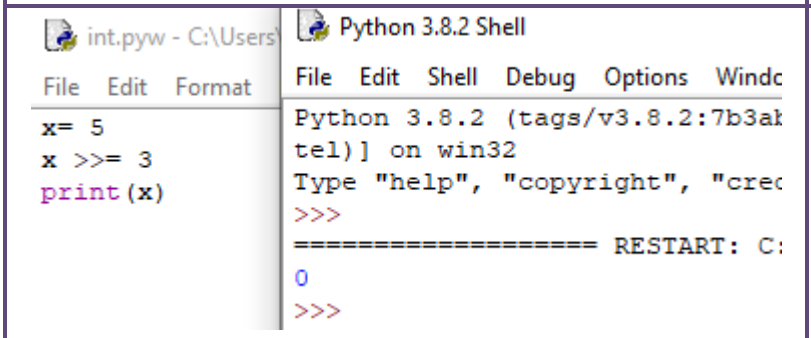
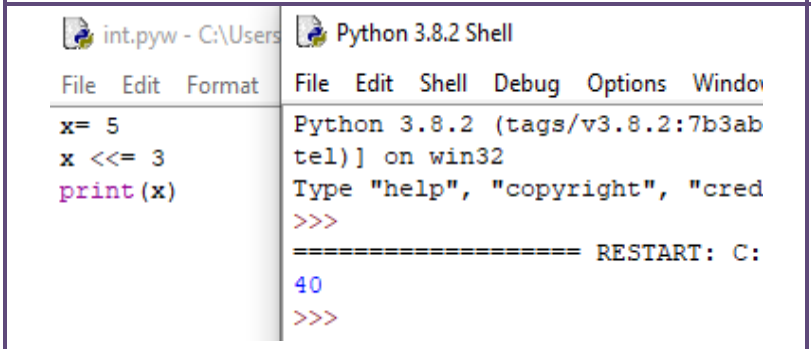


- العمليات لإسناد القيم (Assignment operators): تُستخدم لتعيين قيم للمتغيرات

العملية	عملها	مثال في البايثون
=	اسناد قيمة الى متغير	int.pyw - C:\Use File Edit Format x = 4 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab59 tel)] on win32 Type "help", "copyright", "credit >>> ========== RESTART: C:\U 4 >>>
+=	تمثل هذه العملية كالاتي: x = x + 3	int.pyw - C:\Use File Edit Format x= 4 x+= 3 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Windo Python 3.8.2 (tags/v3.8.2:7b3ab tel)] on win32 Type "help", "copyright", "crec >>> ========== RESTART: C: 7 >>>

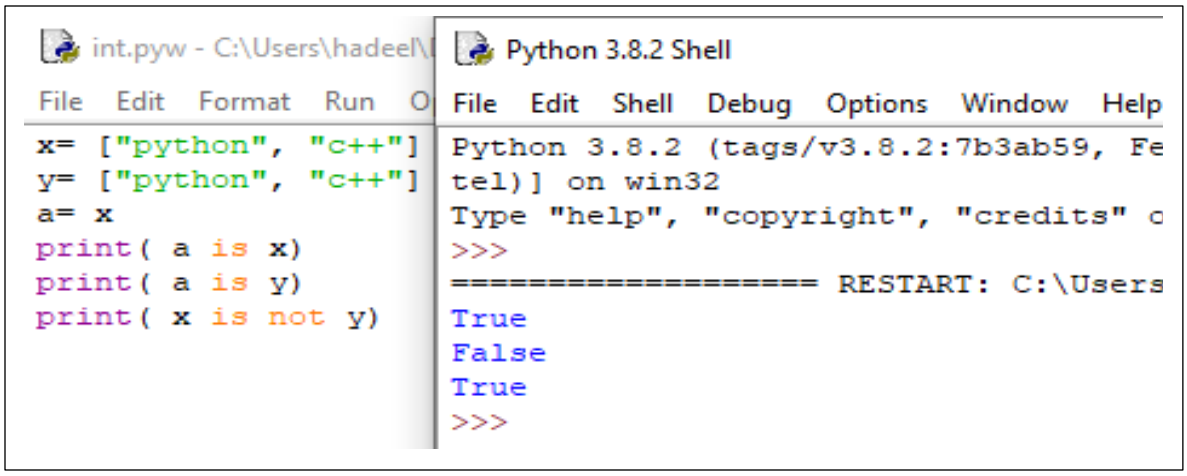
	تمثل هذه العملية كالاتي: x= x-3	-=
	تمثل هذه العملية كالاتي x= x*3	*=
	تمثل هذه العملية كالاتي x= x/3	/=
	تمثل هذه العملية كالاتي x= x%3	%=

int.pyw - C:\Users File Edit Format x= 4 x//= 3 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Windo Python 3.8.2 (tags/v3.8.2:7b3ab tel)] on win32 Type "help", "copyright", "crec >>> ========== RESTART: C: 1 >>>	تمثل هذه العملية كالاتي x= x//3	//=
int.pyw - C:\Users File Edit Format x= 4 x**= 3 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Windo Python 3.8.2 (tags/v3.8.2:7b3ab tel)] on win32 Type "help", "copyright", "cred >>> ========== RESTART: C: 64 >>>	تمثل هذه العملية كالاتي x= x**3	**=
int.pyw - C:\Users File Edit Format x= 5 x&= 3 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Windo Python 3.8.2 (tags/v3.8.2:7b3a tel)] on win32 Type "help", "copyright", "cre >>> ========== RESTART: C 1 >>>	تمثل هذه العملية كالاتي x= x&3	&=
int.pyw - C:\Users File Edit Format x= 5 x|= 3 print(x) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "credi >>> ========== RESTART: C:\ 7 >>>	تمثل هذه العملية كالاتي x= x | 3	|=

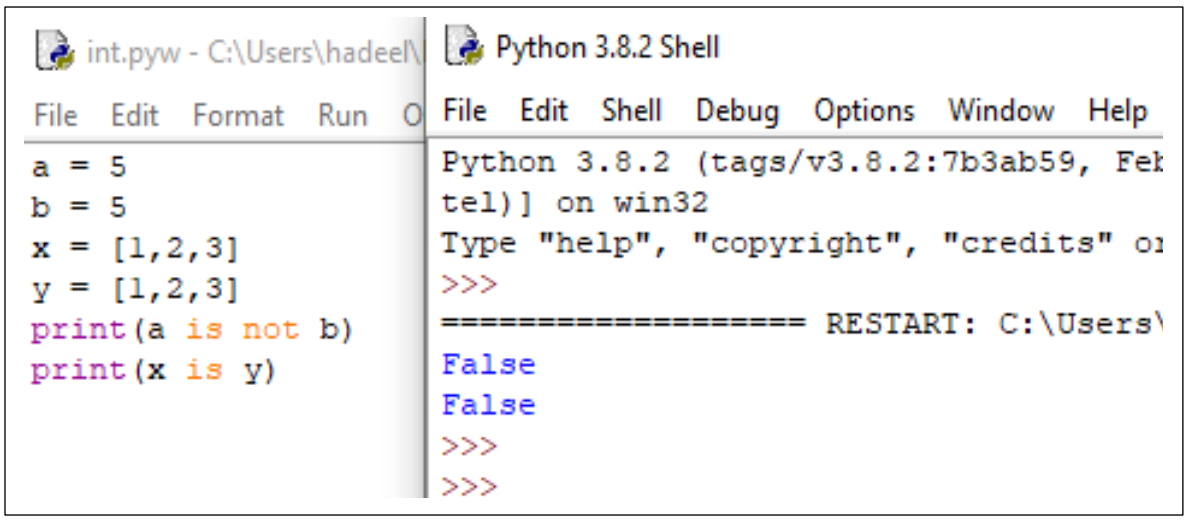
	تمثل هذه العملية كالاتي x= x^ 3	^=
	تمثل هذه العملية كالاتي x= x>>3	>>=
	تمثل هذه العملية كالاتي x= x<<3	<<=

- **العمليات (Identity Operators):** هذا النوع في العمليات يحتوي على (is) و (is not) فقط و تعمل للمقارنة وارجاع قيمة True او False, تعتمد تلك المقارنة على object

بحيث انها يجب ان تتساوى بالقيم وتتساوى بobject لتعطي نتيجة true. بمعنى ادق انه يتم استخدامها للتحقق من وجود قيمتين (أو متغيرات) على نفس الجزء من الذاكرة. وهذا لا يعني ان اي متغيران متساويان أنهما متطابقان. والمثال ادناه سيوضح عملية المقارنة:

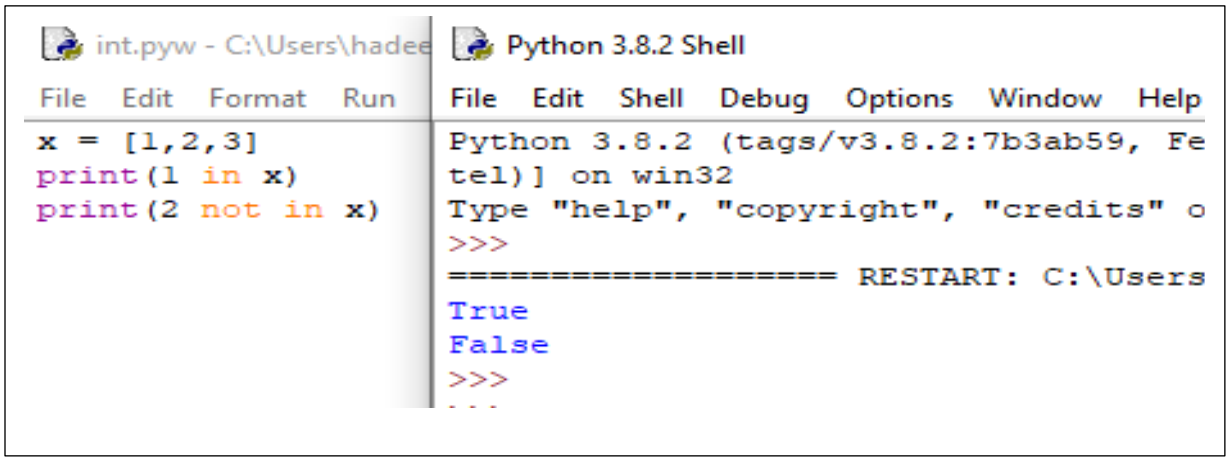


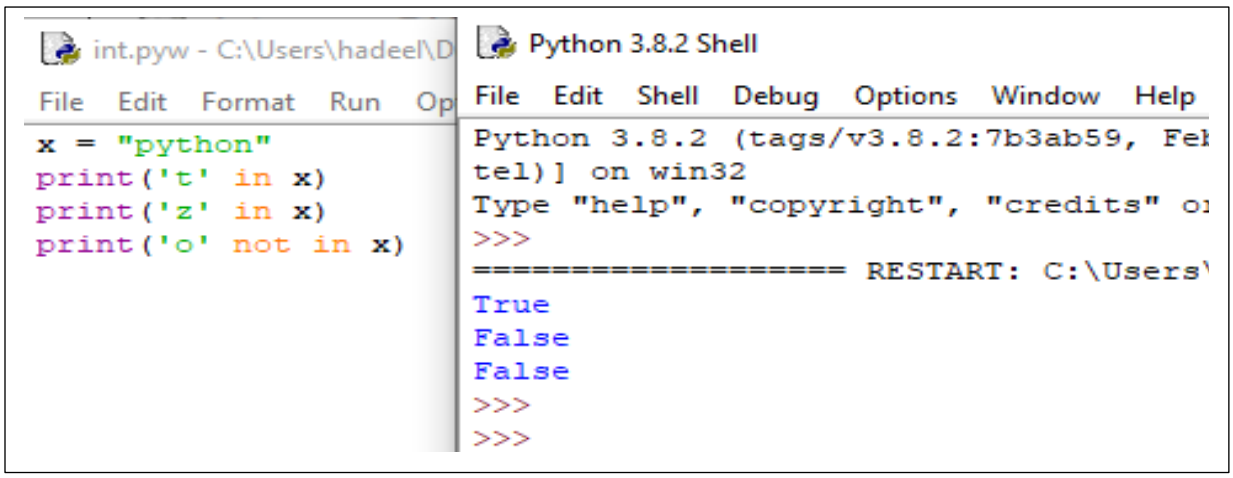
مثال اخر:

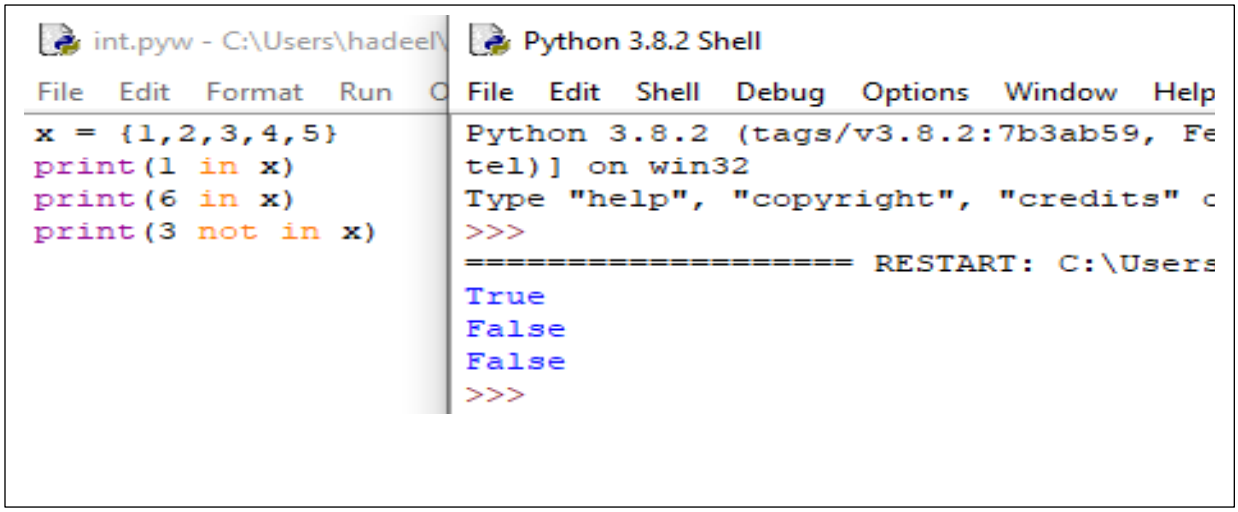


نلاحظ انه في الطباعة الاولى للمثال اعلاه اعطي قيمة خطأ لان الرقم خمسة في a ,b هو ذاته بنفس المكان في الذاكرة لذلك يجب ان تكون النتيجة True . اما في القائمة(list)فهي حالة مختلفة وان تشابهت العناصر داخل القائمتين الا ان لكل واحدة منهم حجز مختلف في الذاكرة

- عمليات العضوية (Membership Operators): وان كان التعريب للكلمة غير دقيق لكنه يوضح انها عمليات اختبار عضوية حيث تختبر هل هذا العنصر موجود في قائمة او حرف في جملة .... الخ


int.pyw - C:\Users\hadee
File Edit Format Run
x = [1,2,3]
print(1 in x)
print(2 not in x)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Fe
tel)] on win32
Type "help", "copyright", "credits" o
>>>
========== RESTART: C:\Users
True
False
>>>


int.pyw - C:\Users\hadeel\D
File Edit Format Run Op
x = "python"
print('t' in x)
print('z' in x)
print('o' not in x)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
True
False
False
>>>
>>>


int.pyw - C:\Users\hadeel\
File Edit Format Run O
x = {1,2,3,4,5}
print(1 in x)
print(6 in x)
print(3 not in x)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Fe
tel)] on win32
Type "help", "copyright", "credits" c
>>>
========== RESTART: C:\Users
True
False
False
>>>

- **عمليات (Bitwise operators):** تعمل على تحويل المعامل الى سلسلة من الارقام الثنائية (تستخدم لمقارنة الأرقام (الثنائية)) و هي كالتالي:

Bitwise عمليات	عملها	مثال
&	تحويل الارقام الى النظام الثنائي واجراء عملية and جميع الاحتمالات صفر ماعدا 1 ,1 تعطي 1	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window He a = 60 # 60 = 0011 1100 b = 13 # 13 = 0000 1101 c = 0 c = a & b # 12 = 0000 1100 print (" the value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "credi >>> ========== RESTART: C:\ the value of c is 12 >>> >>>
|	تحويل الارقام الى النظام الثنائي واجراء عملية or جميع الاحتمالات 1 ماعدا 0,0تعطي 0	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window He a = 60 # 60 = 0011 1100 b = 13 # 13 = 0000 1101 c = 0 c = a | b # 61 = 0011 1101 print (" the value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Windo Python 3.8.2 (tags/v3.8.2:7b3ab tel)] on win32 Type "help", "copyright", "cred >>> ========== RESTART: C: the value of c is 61 >>> >>>
^	تحويل الارقام الى النظام الثنائي واجراء عملية ^ تعطي الارقام الثنائية المتشابهة 0 والمختلفة 1	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window He a = 60 # 60 = 0011 1100 b = 13 # 13 = 0000 1101 c = 0 c = a ^ b # 49 = 0011 0001 print ("Value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Window Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "credi >>> ========== RESTART: C:\ Value of c is 49 >>> >>>

~	هذه الاشارة هي Not تعني الصفر واحد والواحد تحوله الى صفر	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window H a = 60 # 60 = 0011 1100 c = ~a #-61 = 1100 0011 print ("Value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Windov Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "cred: >>> ========== RESTART: C:\ Value of c is -61 >>>
>>	تسمى right shift يتم نقل قيمة المعاملات اليسرى إلى اليسار بعدد البتات التي يحددها المعامل الأيمن.	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window He a = 60 # 60 = 0011 1100 c= 0 c = a << 2 # 240 = 1111 0000 print ("Value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Windov Python 3.8.2 (tags/v3.8.2:7b3ab5 tel)] on win32 Type "help", "copyright", "credi >>> ========== RESTART: C:\ Value of c is 240 >>> >>>
<<	تسمى left shift يتم نقل قيمة المعاملات اليسرى لليمين بعدد البتات التي يحددها المعامل الأيمن.	int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2) File Edit Format Run Options Window H a = 60 # 60 = 0011 1100 c= 0 c = a >> 2 # 15 = 0000 1111 print ("Value of c is ", c) Python 3.8.2 Shell File Edit Shell Debug Options Windov Python 3.8.2 (tags/v3.8.2:7b3ab tel)] on win32 Type "help", "copyright", "cred >>> ========== RESTART: C: Value of c is 15 >>>

We finished Python Operators ☺

Making Choices and Decisions

صنع الاختيارات والقرارات

الان بعد ان تعلمنا اساسيات البايثون يمكن جعل البرامج اكثر ذكاء من خلال اتخاذ القرارات وتشمل هذه الفقرة كل من :

- If statement
- For loop
- While loop

تعرف ايضا ب(Conditionals and loops) هذه الشروط تعمل على السيطرة(control flow) على تدفق البيانات في البرنامج . بالإضافة إلى ذلك ، سنلقي نظرة أيضًا على المحاولة و الاستثناء وبيان ما يجب على البرنامج القيام به عند حدوث خطأ. اذا لنبدأ ب If statement

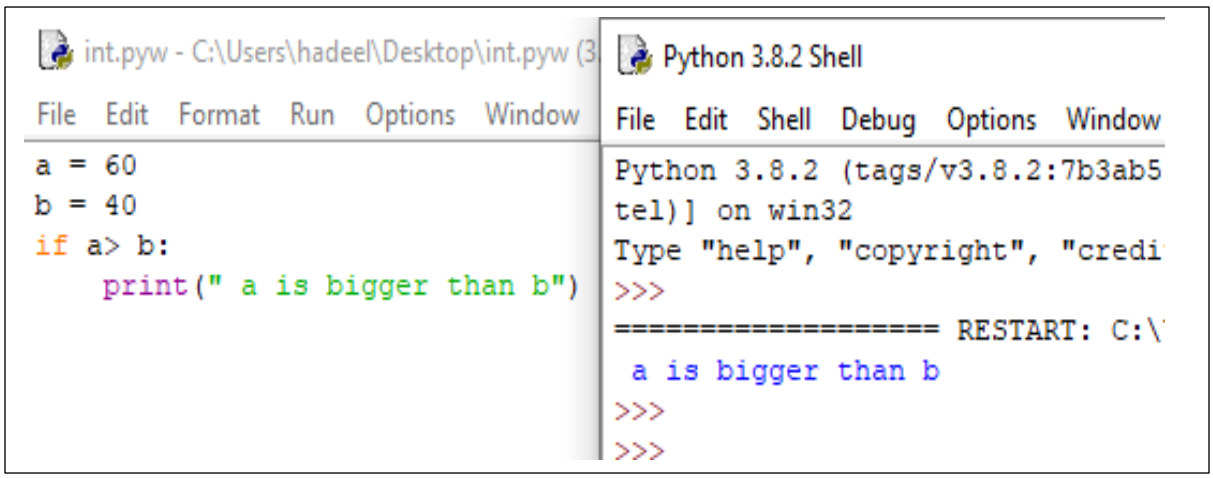
- If statement

لقد لاحظنا مسبقا ان البايثون تدعم الشروط المنطقية المعتادة من الرياضيات اثناء شرحنا لها في العمليات المنطقية. لكن الان نستطيع استخدام الشرط if ووضع عبارة اذا تحقق الشرط تنفذ تلك العبارة .

لنتعرف على تركيبة if statement في البايثون :

```
if test expression:
    Statement of if
elif test expression:
    Statement of elif
else:
   Statement of else
```

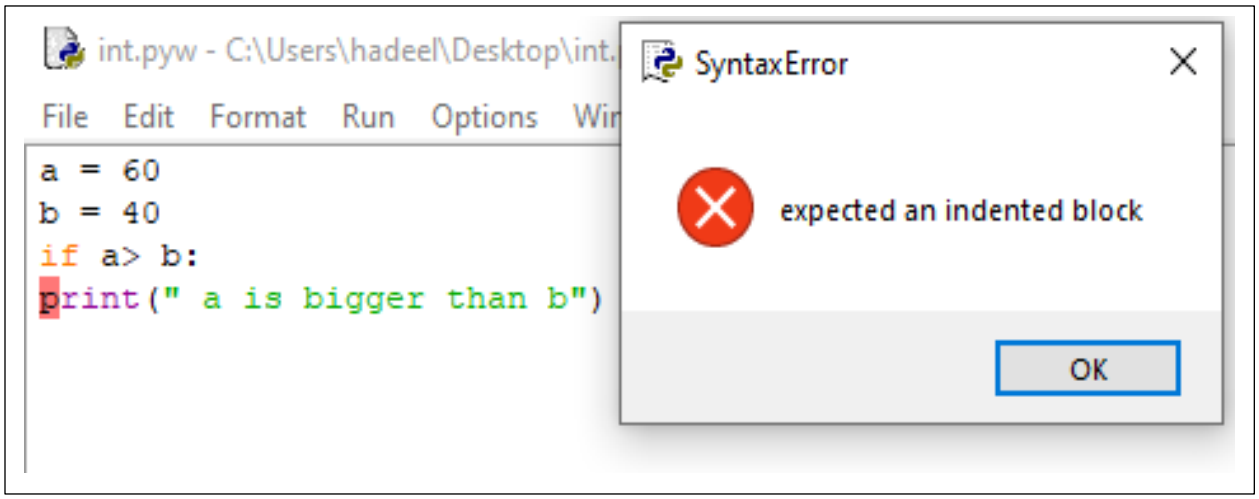
سنتعلم ايضا كيفية تمثيل if statement في البايثون خطو بخطوة :



في هذا المثال اعلاه ، لقد استخدمنا متغيرين وهما a و b ، يتم استخدامهما كجزء من if statement لاختبار ما إذا كانت a أكبر من b. بما أن 60 a ، و b هي 40 ، فإننا نعلم أن 60 أكبر من 40 ، لذا فإننا نطبع على الشاشة "a أكبر من b".

لكن لنلاحظ قليلا في المثال اعلاه ان هناك مسافة (whitespace) ما بعد ال if حيث ان تلك المسافة ما قبل عبارة الprint هي لتنفيذ شرط if . في بعض اللغات تكون ما بعد if اقواس المتعرجة {} ومابينهما العبارات التي نريد تنفيذها لكن في البايثون لا توجد اقواس وانما وضع مسافة فقط

لنجرب في حال لم نضع تلك المسافة ماذا سيحصل :



لم يتم تنفيذ الشرط لأنه لا توجد مسافة قبل الطباعة . لكن من جهة اخرى يمكن ان يكون statement(الطباعة) بجانب الشرط كالاتي:

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3.8.2)

File Edit Format Run Options Window Help

```
a = 40
b = 60
if b > a: print("b is greater than a")
```

Python 3.8.2 Shell

File Edit Shell Debug Options Wind

```
Python 3.8.2 (tags/v3.8.2:7b3a
tel)] on win32
Type "help", "copyright", "cre
>>>
========== RESTART: C
b is greater than a
>>>
```

والان في حال ان الشرط الاول لل if غير صحيح لذا يمكن ان نستخدم شرط اخر من خلال elif وهي طريقة البايثون لقول "إذا لم تكن الشروط السابقة صحيحة ، فجرب هذا الشرط".

int.pyw - C:\Users\hadeel\Desktop\int.pyw

File Edit Format Run Options Windo

```
a = 60
b = 40
if b > a:
  print("b is greater than a")
elif b < a:
  print("b is smaller than a")
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Fe
tel)] on win32
Type "help", "copyright", "credits" o
>>>
========== RESTART: C:\Users
b is smaller than a
>>>
>>>
```

وفي حال عدم وجود شرط صحيح يمكن تنفيذه ففي هذه الحالة نستخدم Else:

int.pyw - C:\Users\hadeel\Desktop\int.pyw (3

File Edit Format Run Options Window

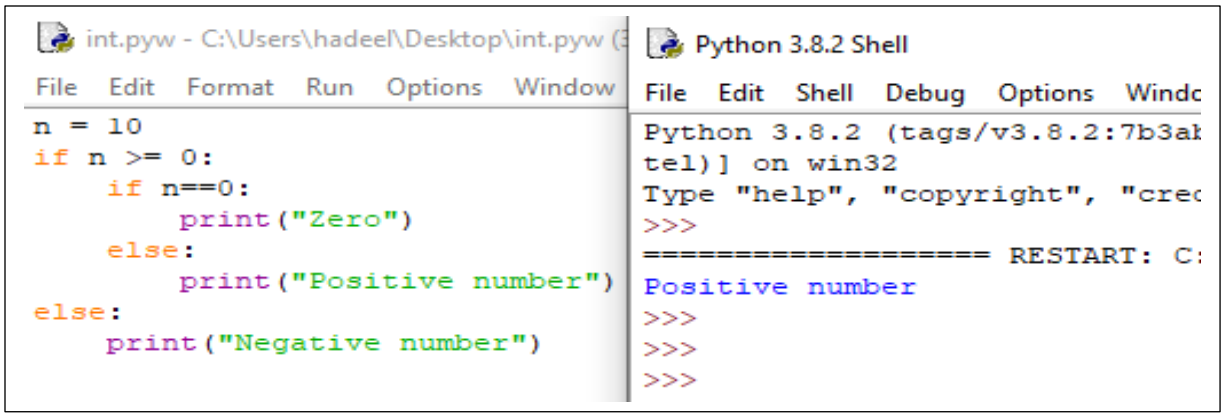
```
a = 60
b = 60
if b > a:
    print("b is greater than a")
elif b < a:
    print("b is smaller than a")
else:
    print("a and b are equal")
```

Python 3.8.2 Shell

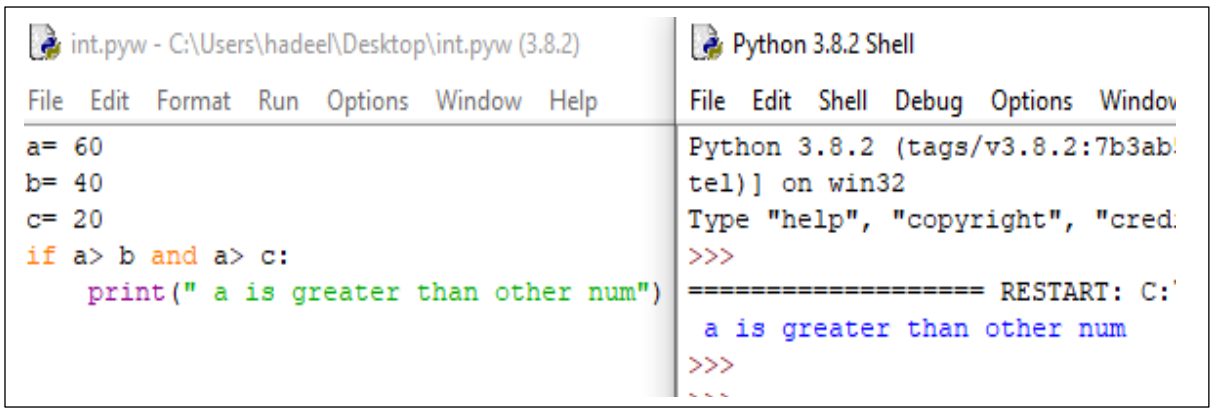
File Edit Shell Debug Options Windov

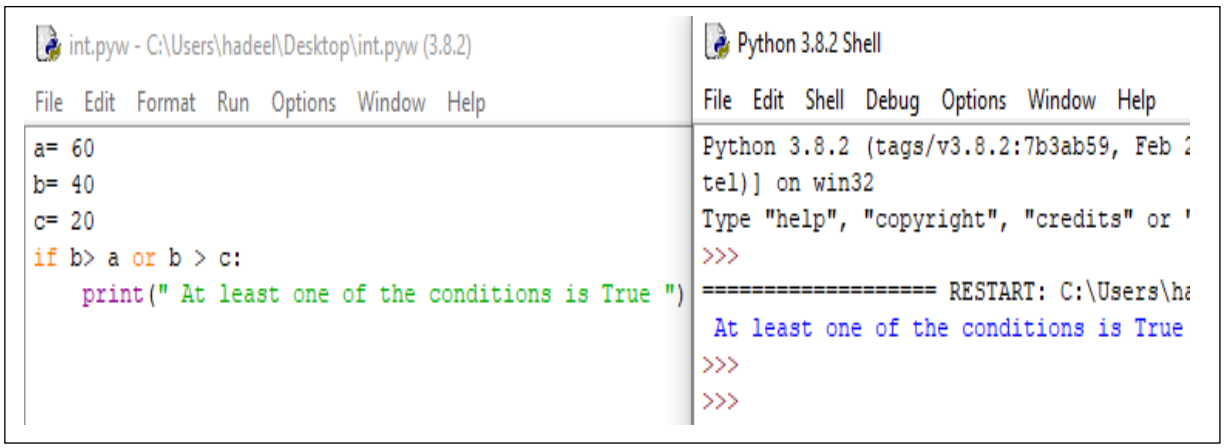
```
Python 3.8.2 (tags/v3.8.2:7b3ab
tel)] on win32
Type "help", "copyright", "cred
>>>
========== RESTART: C:
a and b are equal
>>>
>>>
>>>
```

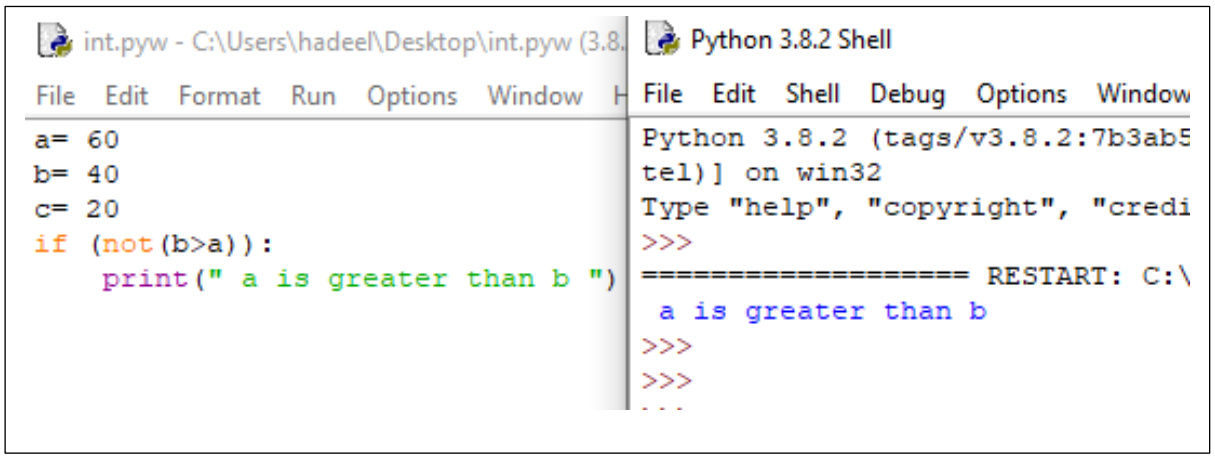
يوجد ايضا ما يعرف ب Nested If والتي يجب ان نذكرها وهي عبارة عن if statement بداخلها if statement اخرى :



يمكن ان تتضمن if statement العمليات المنطقية (logical operator) حيث انها تدمج مع العبارات الشرطية ومثال على ذلك:





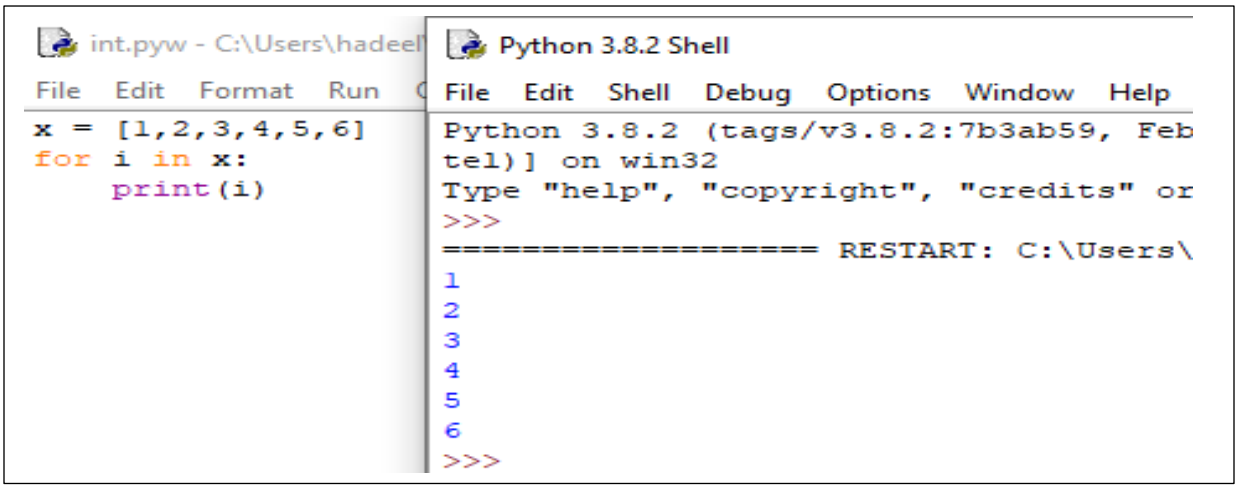


For Loops :-

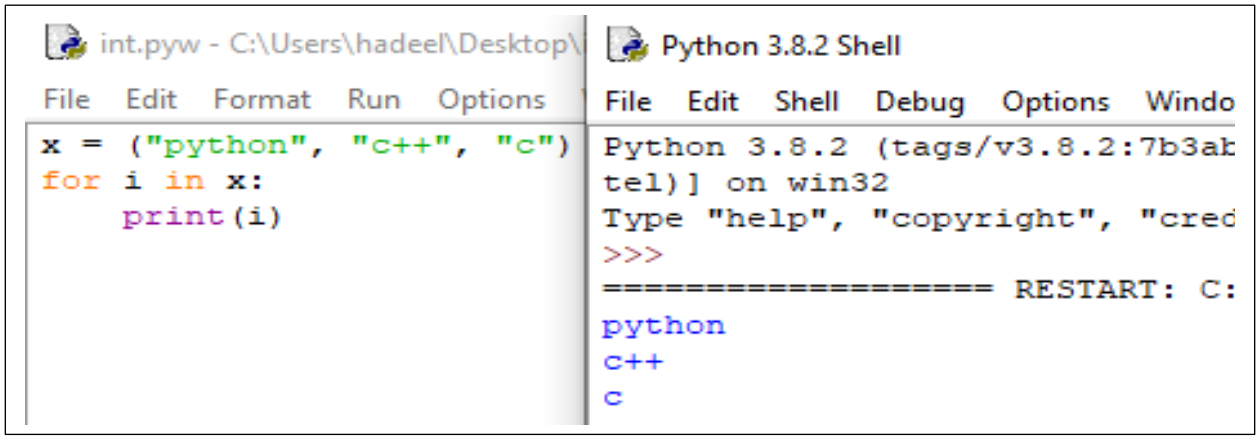
تُستخدم الحلقة التكرارية (for) في لغة البايثون للتكرار وذلك عبر تسلسل (قائمةlist ، مجموعةset ، جملةstring) أو اي بيانات أخرى قابلة للتكرار. لذا ان تركيبة for في البايثون كالاتي:

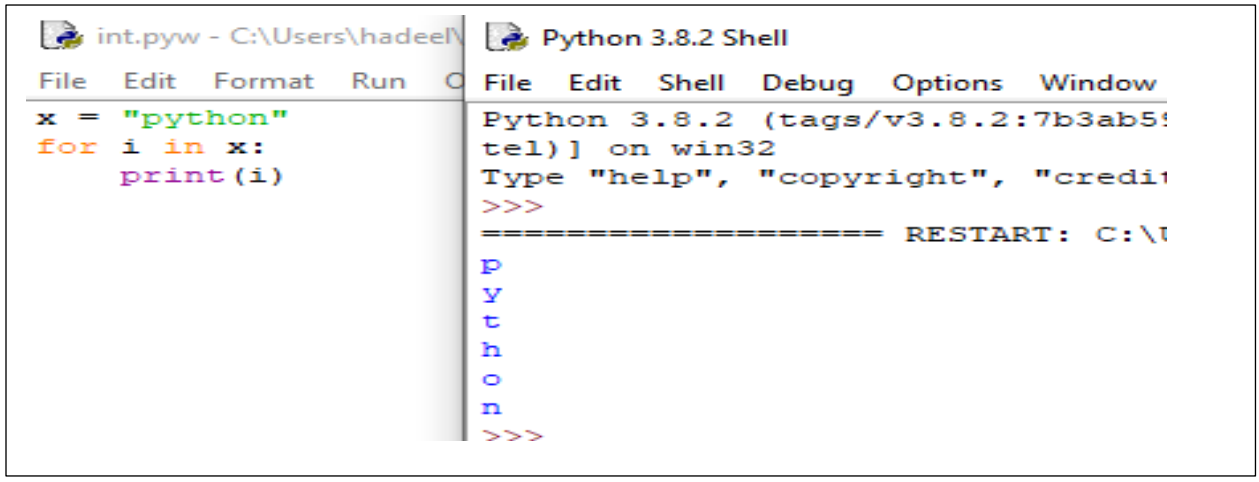
```
for value in sequence:
        statement of for
```

نلاحظ ان المتغير value يأخذ قيمة العنصر داخل المتسلسل في كل تكرار.حيث تستمر الحلقة التكرارية حتى نصل إلى العنصر الأخير في التسلسل .

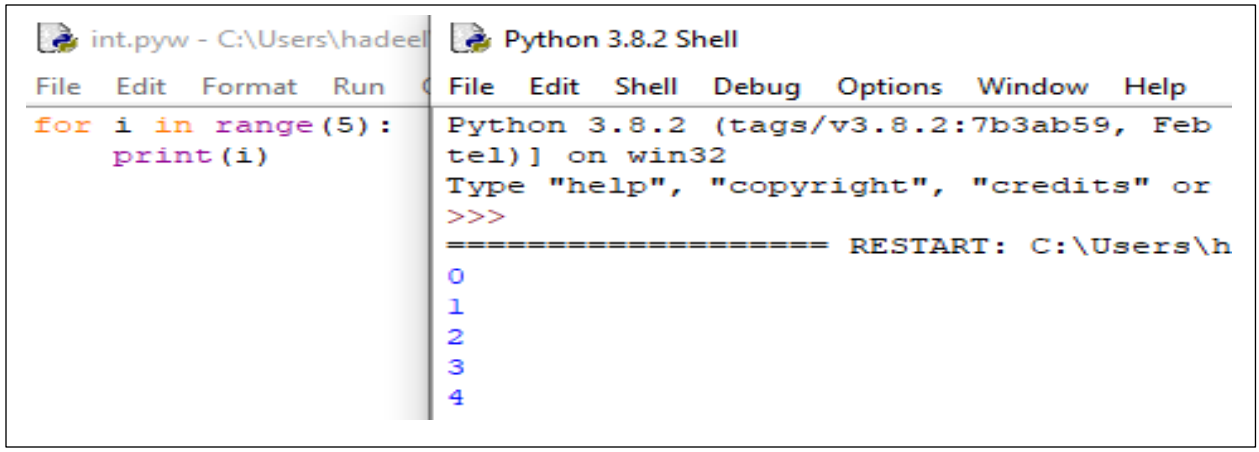


اعلاه نشاهد اننا لم نعرف المتغير i لان البايثون من اسهل اللغات الموجودة حاليا فانت لا تحتاج لتعريف المتغيرات قبل استخدامها . ان البرنامج عمل على طباعة جميع الارقام الموجودة بالقائمة وعندما وصل الى النهاية انتهى البرنامج .

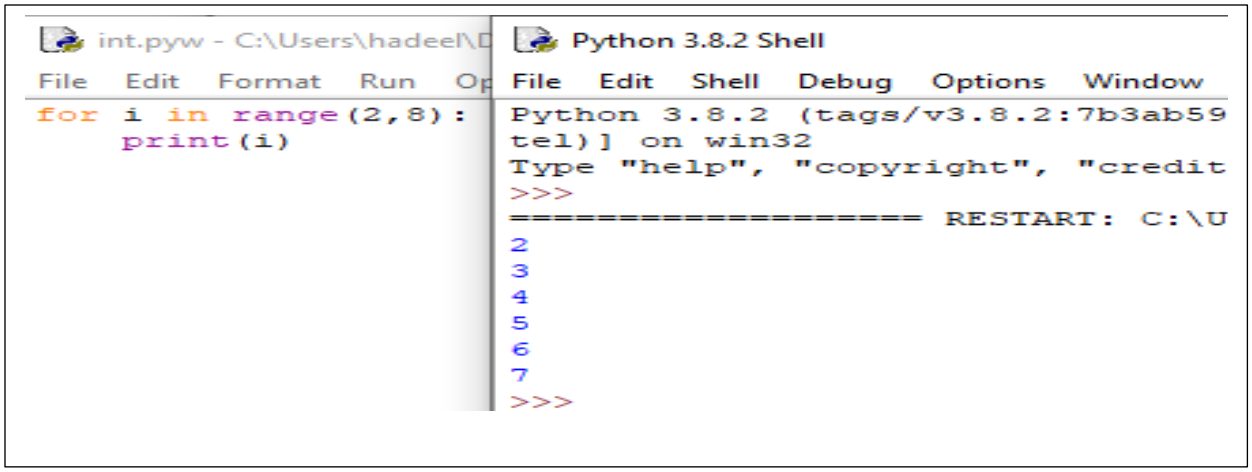




يمكن استخدام دالة المدى range () داخل الحلقة التكرارية حيث تُرجع الدالة range() سلسلة من الأرقام ، تبدأ من 0 بشكل افتراضي ، وتزداد بمقدار 1 (افتراضيًا) ، وتنتهي بالرقم المحدد ما بين قوسي الrange

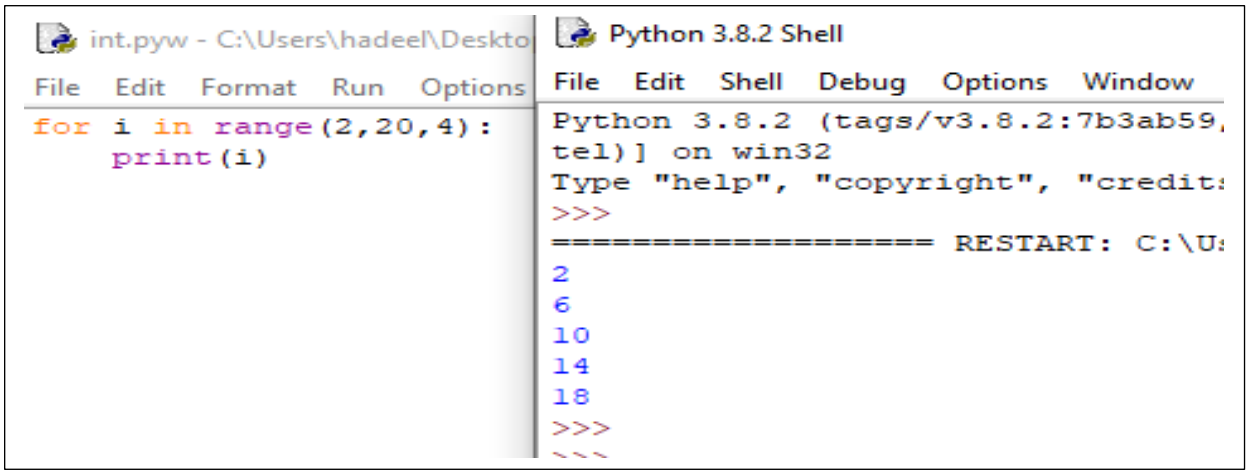


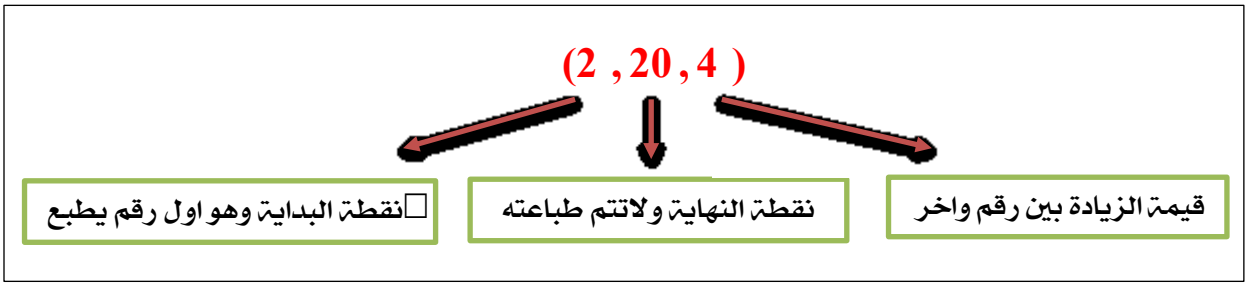
يمكن ان نحدد دالة المدى بقيمة بداية مثلا نجعلها تبدا من ٢ بدلا من الصفر وقيمة نهاية وهي ٨ :



في المثالين السابقين كانت قيمة الزيادة بين رقم والاخر هو 1 ولكن هل يمكن تغيير هذه القيمة ؟

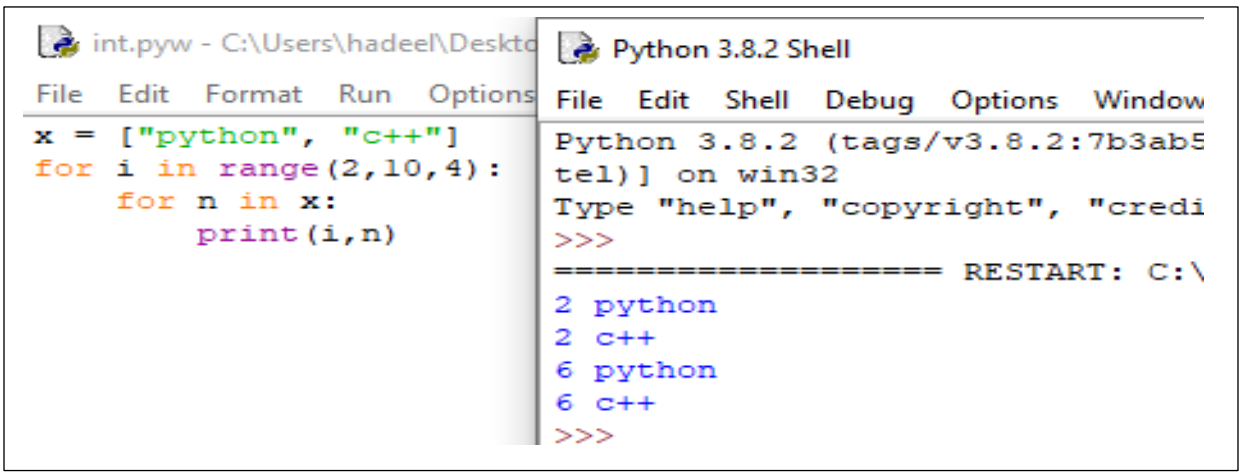
الجواب : نعم يمكن ذلك حيث نستطيع ان تكون قيمة الزيادة ٢ و ٣ او ٤ .... الخ باي قيمة داخل دالة المدى ويمكننا تمثيل ذلك كالاتي:





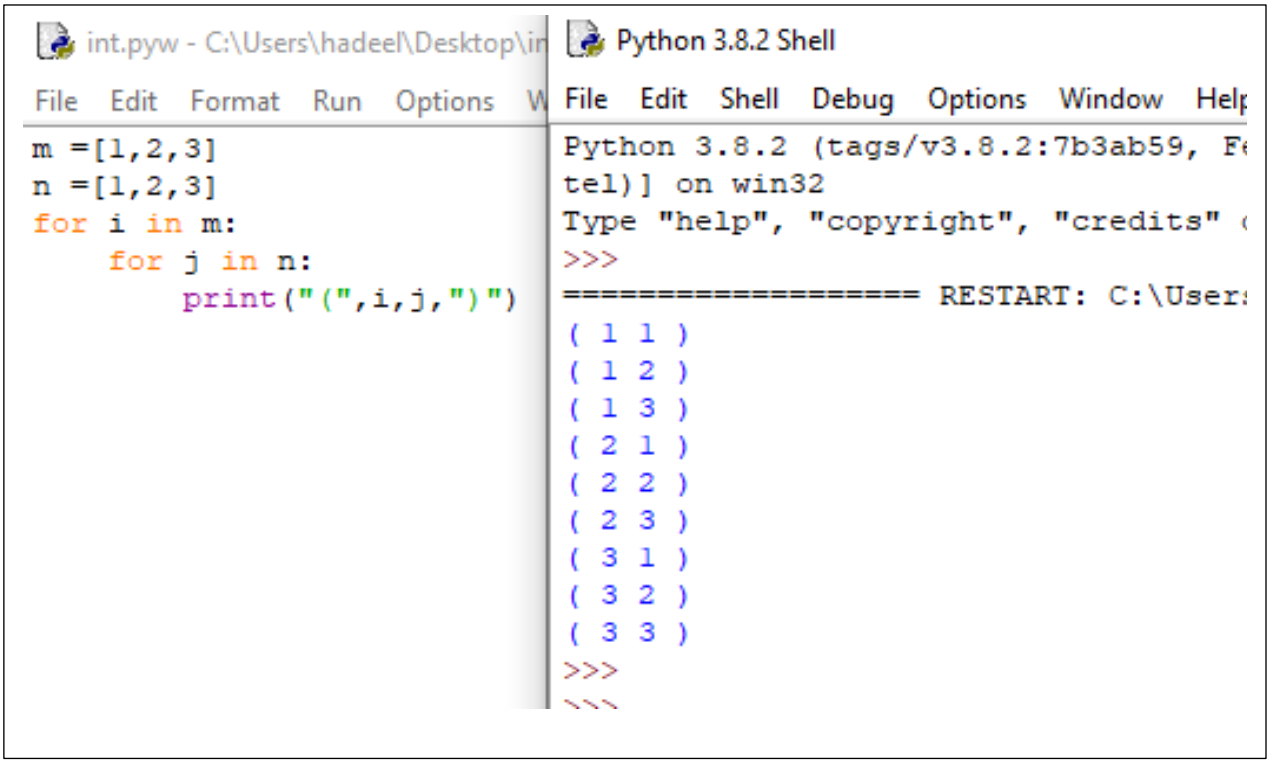
هناك ما يدعى بالتكرار المتداخل (Nested Loops) وهي عبارة عن for loop بداخلها for loop اخرى حيث تتم العملية من خلال تنفيذ loop الداخلية لكل iteration في loop الخارجية

ولسهولة فهم اسلوب عمل الحلقات التكرارية المتداخلة من خلال المثال ادناه:

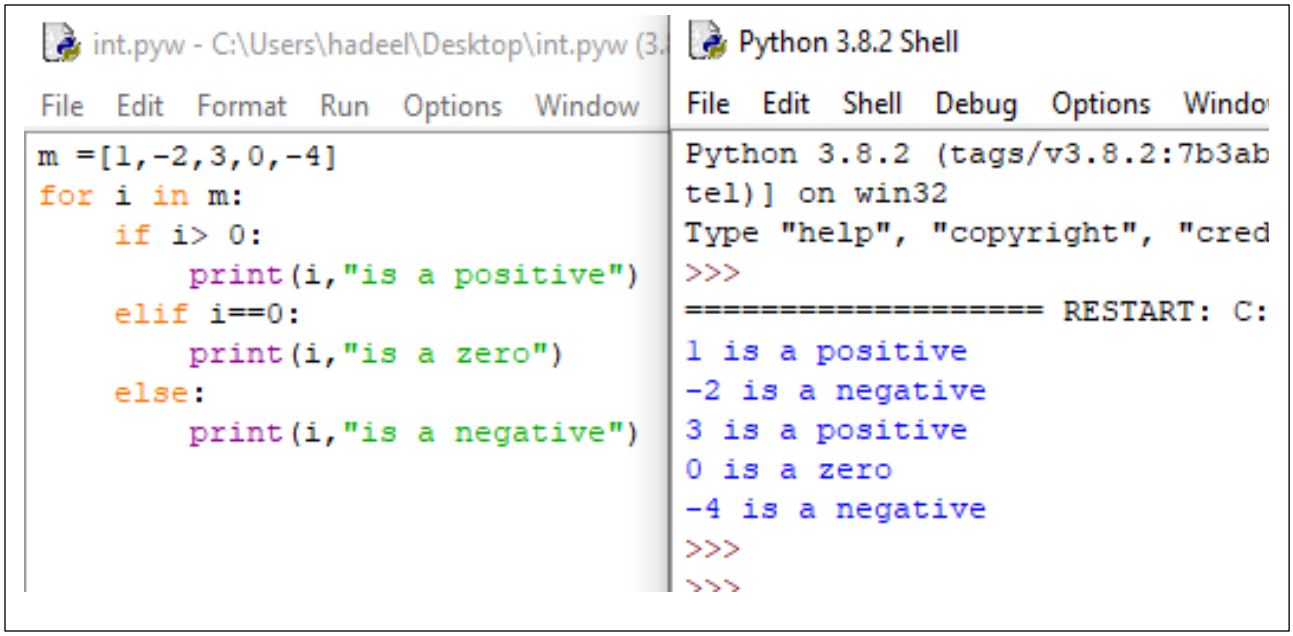


لقد تم تنفيذ البرنامج بالفعل من خلال تنفيذ اللوب الداخلي بالكامل لكل لوب خارجي اي تم تنفيذ القائمة بالكامل لكل رقم بالمدى.

مثال اخر لتوضيح تداخل الحلقات التكرارية :



يمكننا ان نضم if داخل for loop حيث ينفذ الشرط ان كان صحيح في كل دورة لfor



المسافات ضرورية جدا (كالتي بين ال for و if) يجب الانتباه لها لأنها قد تكون سبب لعدم تنفيذ برنامجك.

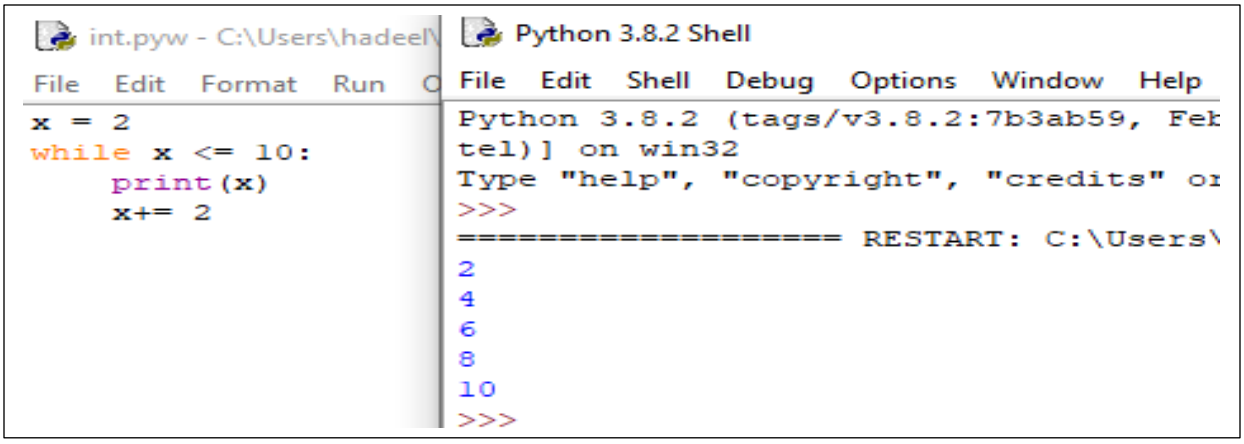
- **الحلقة التكرارية (While Loops):**

باستعمال حلقة while نستطيع تنفيذ مجموعة من العبارات طالما أن تعبير الاختبار(الشرط) صحيح.

ان التركيبة الاساسية لحلقة while كالاتي:

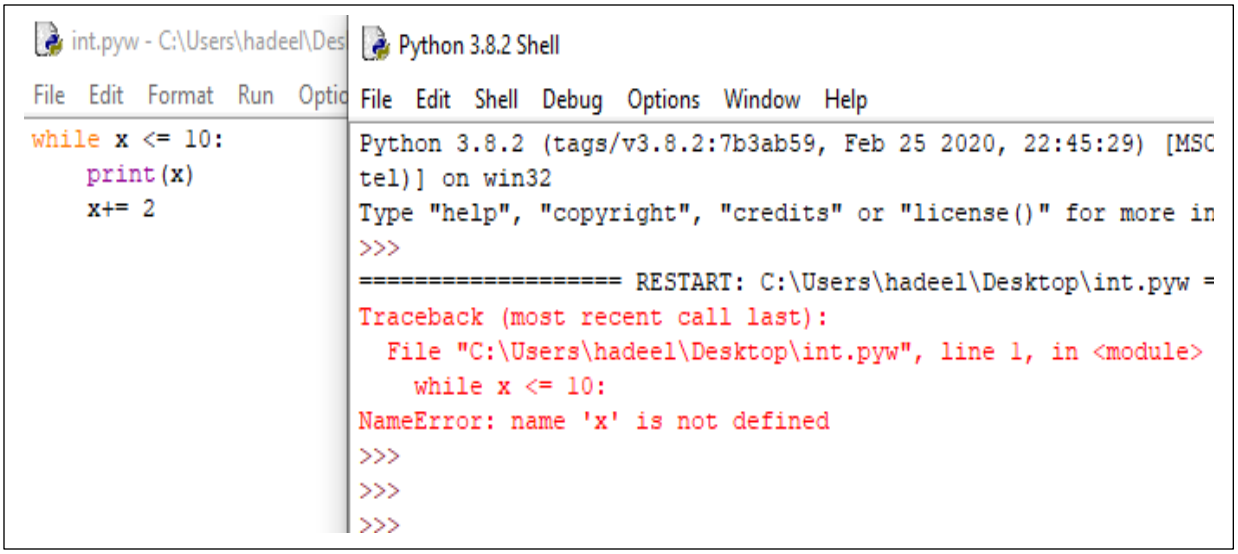
```
while test expression:
    Body of while
```

في (while loops) الحلقة التكرارية يتم فحص التعبير اولا وفي حال انه صحيح (true) تستخدم while وهكذا يستمر بالتنفيذ الى حين ان يتم تقييم التعبير بخطأ(false). هذه الحلقة التكرارية عادة ما تكون هناك مسافات يجب الانتباه اليها ايضا وهي مهمة جدا لضمان التنفيذ بدون اي اخطاء.



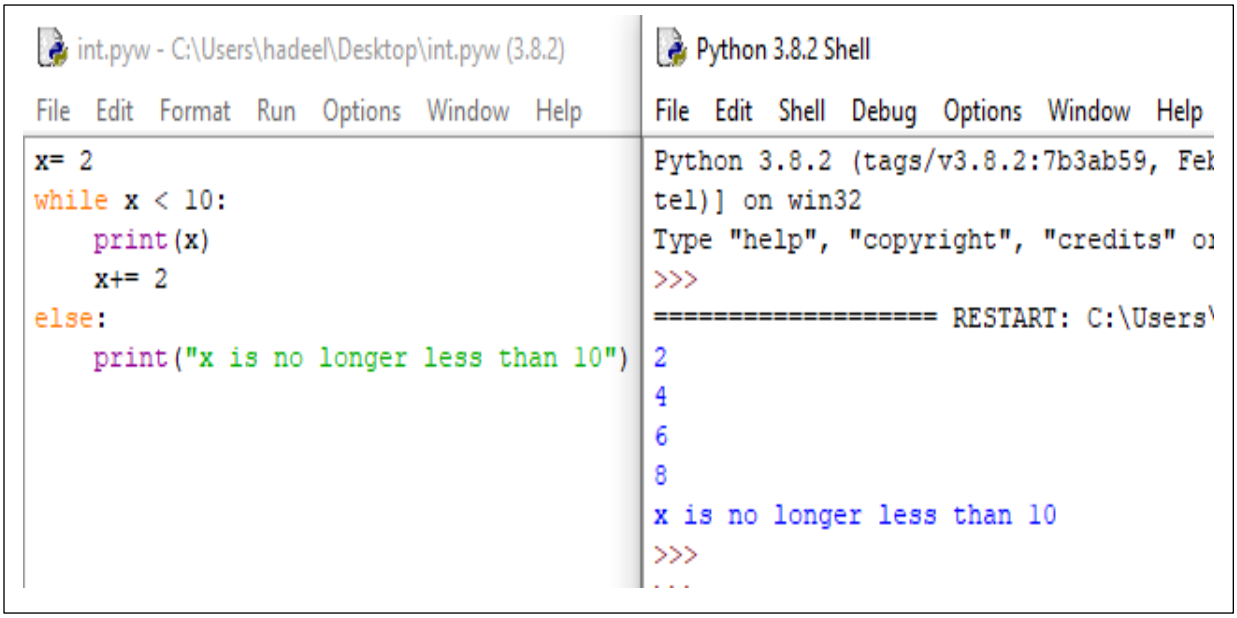
في while loop يجب ان نعطي قيمة بدائية للمتغير فضلا عن ذلك انها ستستمر بالتنفيذ الا ان يكتمل الشرط وهو ان x يجب ان يكون اصغر او يساوي ١٠ حينها سيتوقف البرنامج .

في حال عدم اعطاء قيمة للمتغير سيكون هناك خطا وسيظهر لك على شاشة التنفيذ :



While loop with else مصطلح جديد يجمع ما بين الحلقة التكرارية while مع else والتي تعودنا على استخدامها في if .

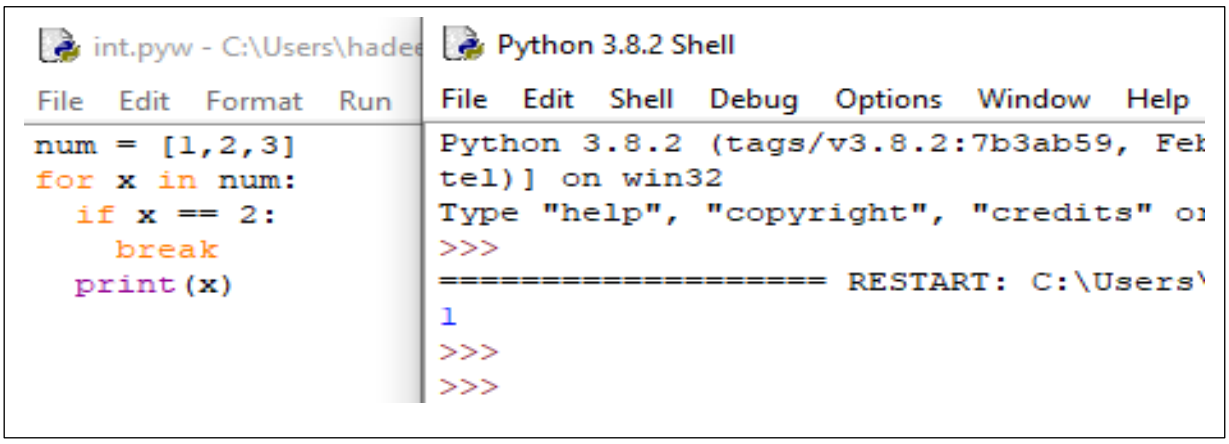
في الشرطية if استخدمنا else في حال كان الشرط غير صحيح نستعين بelse لكي تعطينا عبارة كالطباعة مثلا . اما هنا في while تستخدم بنفس القالب اي في حال ان الشرط غير صحيح يمكن استخدام else لتشغيل كتلة من التعليمات البرمجية مرة واحدة



والان انتهينا من الاختبارات والقرارات .

لكن قبل كل ان نختم موضوع الاختبارات والقرارات يجب ان نكتب لك ملاحظات مهمة ارجو الاستفادة منها:

- ان نهاية كل من if و for و while نقطتان متعامدتان (:) ويجب علينا ان لا ننساها ابدا لان نسيانها سيعطيك خطا في التنفيذ.
- المسافات ضرورية جدا حيث تم ذكرها اكثر من مرة لأهميتها لتنفيذ برنامجك بسهولة
- يمكن ان تتداخل ال for و while مع if الشرطية بسهولة
- لا توجد فارزة منقوطة اثناء كتابة الجمل البرمجية في البايثون على عكس C++
- هناك ما يدعى ب break و continue تستخدم في for و while واستخدامها سهل للغاية الاولى تستخدم لإيقاف الحلقة حتى لو كان الشرط صحيحًا اما continue فتستخدم لإيقاف التكرار الحالي ، والاستمرار مع التالي . ومثال ذلك :



int.pyw - C:\Users\hadee

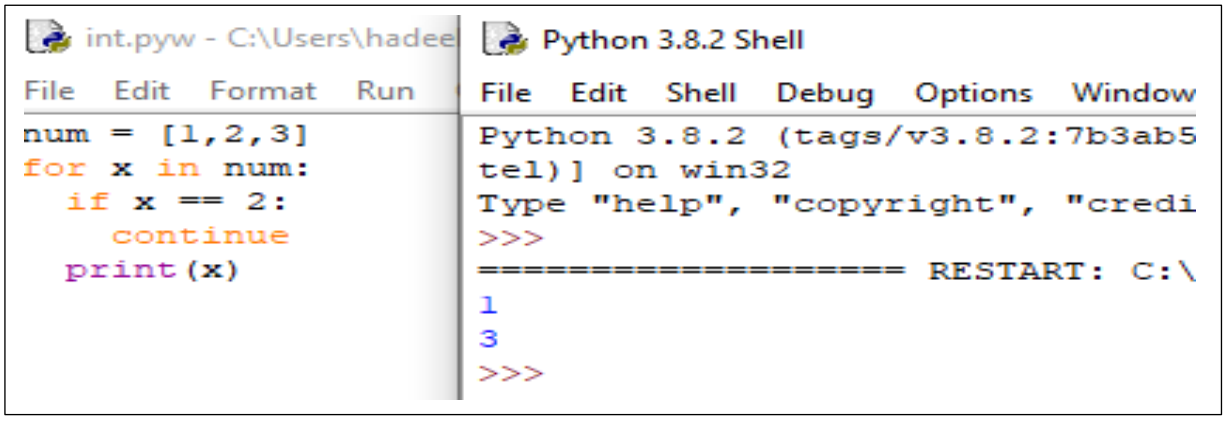
File Edit Format Run

```
num = [1,2,3]
for x in num:
  if x == 2:
    break
  print(x)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
1
>>>
>>>
```



int.pyw - C:\Users\hadee

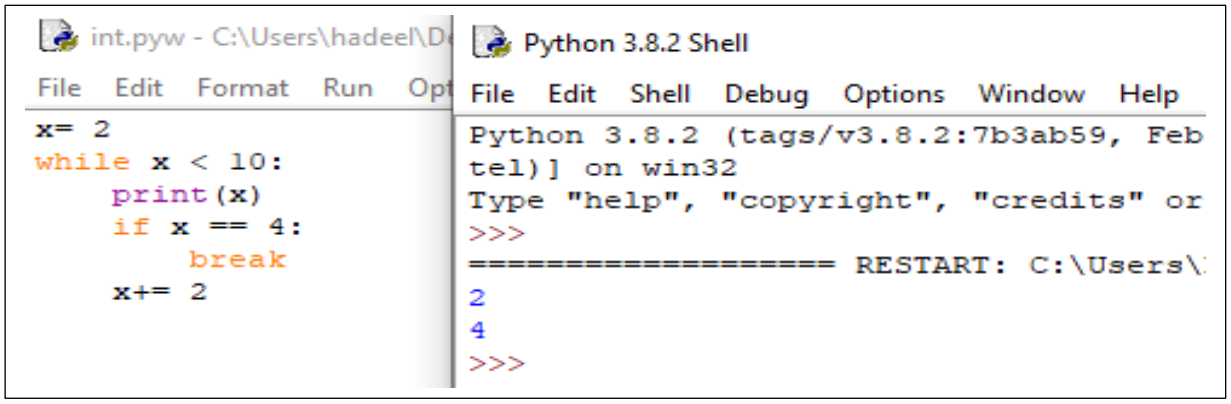
File Edit Format Run

```
num = [1,2,3]
for x in num:
  if x == 2:
    continue
  print(x)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window

```
Python 3.8.2 (tags/v3.8.2:7b3ab5
tel)] on win32
Type "help", "copyright", "credi
>>>
========== RESTART: C:\
1
3
>>>
```



int.pyw - C:\Users\hadeel\D

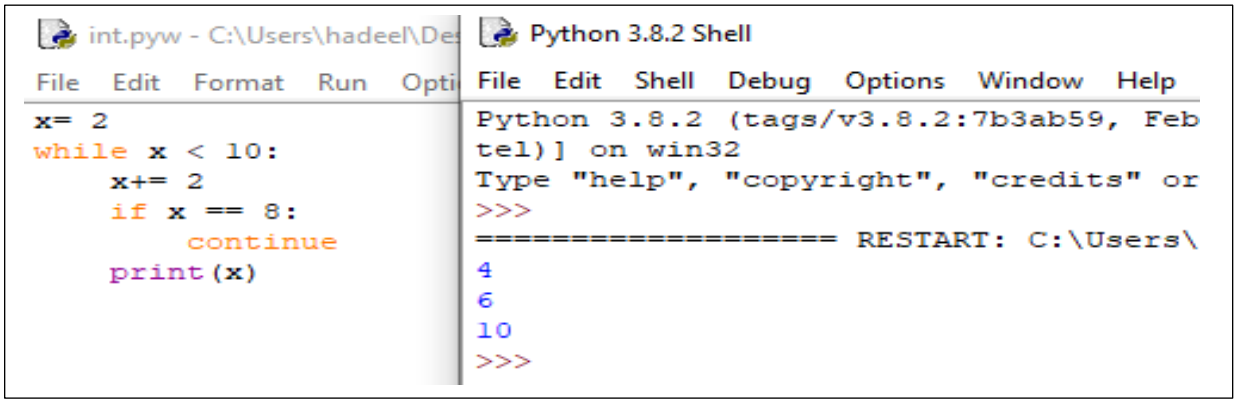
File Edit Format Run Opt

```
x= 2
while x < 10:
    print(x)
    if x == 4:
        break
    x+= 2
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
2
4
>>>
```



int.pyw - C:\Users\hadeel\De

File Edit Format Run Opti

```
x= 2
while x < 10:
    x+= 2
    if x == 8:
        continue
    print(x)
```

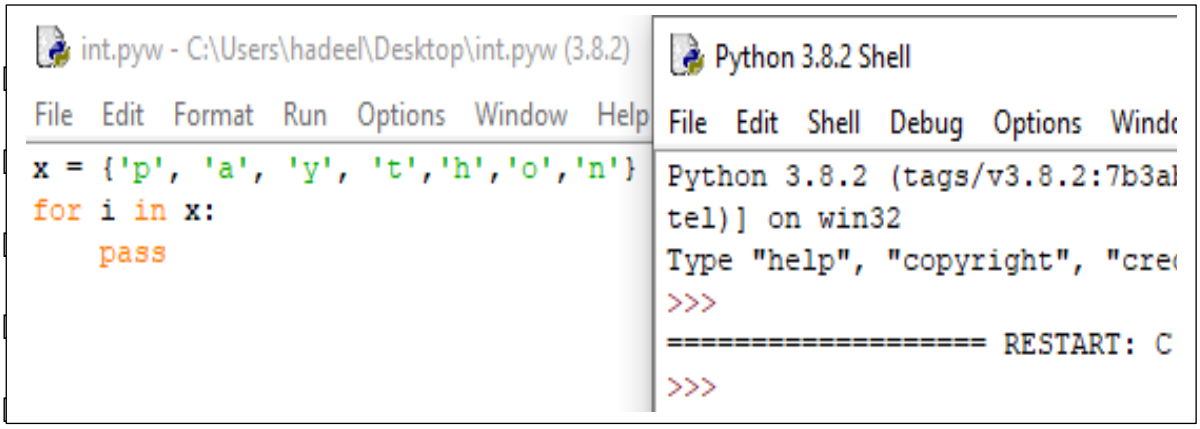
Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

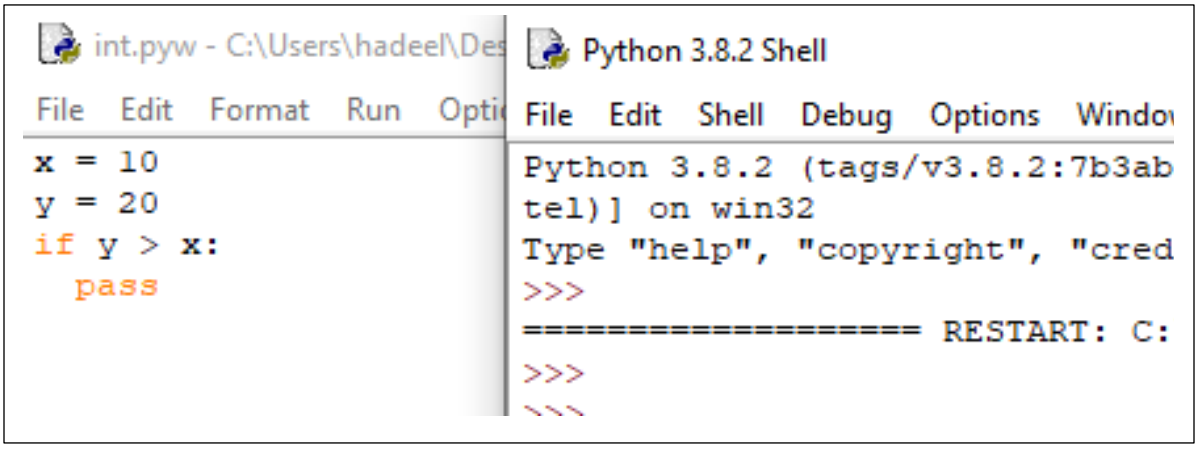
```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
========== RESTART: C:\Users\
4
6
10
>>>
```

- عبارة المرور (pass statement) : تستخدم عبارة pass لمرور عبارات لتجنب ظهور الاخطاء .

لننفذ pass ليكون فهم عملها اسهل للقارئ:



في برمجة البايثون عبارة pass تكون عبارة المرور عبارة خالية اي لا يوجد اي شيء لفعله لذا في المثال السابق فقط تم مرور جميع عناصر x (تم تنفيذ البرنامج بدون حدوث اي شيء)



- الملاحظة الاخيرة في حال تم الخطأ في احدى الخطوات واصبح لدينا infinite loop تكرار الى ما لانهاية نستخدم CTRL+C keys للخروج منه.

Python Try, Except

الاستثناءات في بايثون

الاستثناءات في لغة بايثون يكون من خلال تعبير(try ، except واخيرا finally) فكرة الاستثناء هي في حال وجد خطا برمجي يمكن استثنائه في محاولة واحدة او اثنان او ثلاثة حسب احتياجك الى ان نصل ل finally ، وذلك لأنه في حال عدم الاستثناء قد يؤدي تنفيذ تلك الشيفرة إلى التسبب في حدوث اخطاء برمجية وبالتالي ايقاف البرنامج ان تركيبة الاستثناءات كالتالي:

```
try:
do something
except:
do something else when an error occurs
```

tryexception.pyw - C:\Users\hadeel\Desktop\tryexception.pyw (3.8.2)

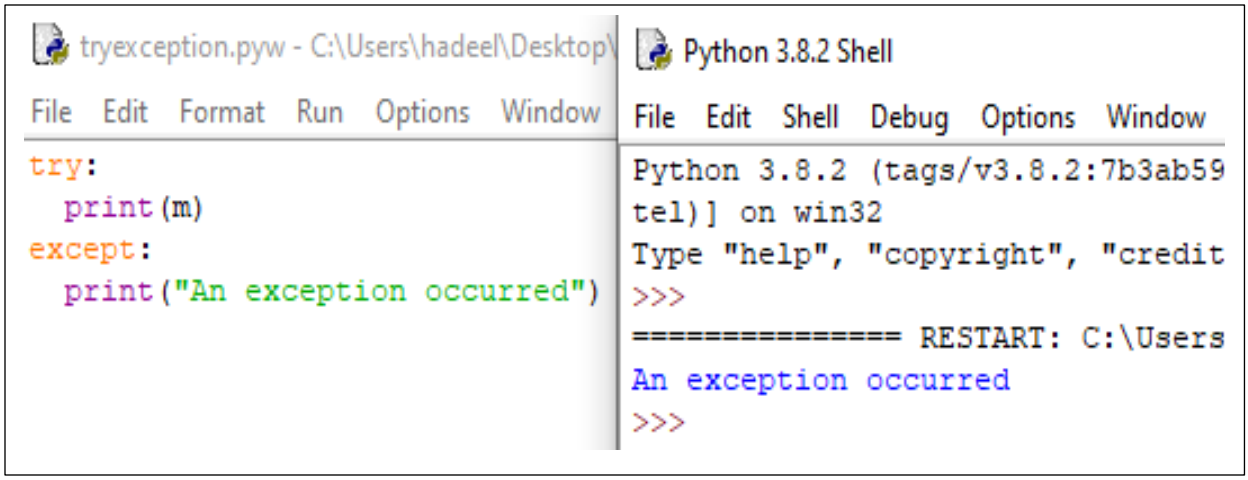
File Edit Format Run Options Window Help

```
while True:
    try:
        x= int(input("inter number int :  "))
        break
    except ValueError:
        print("that was no vaild int, try again please")
```

Python 3.8.2 Shell

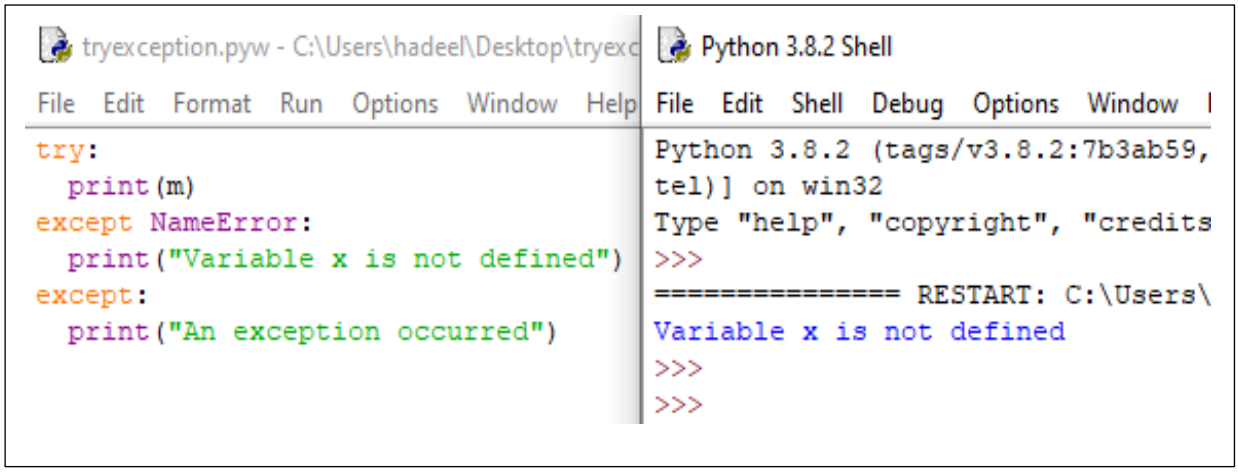
File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020,
tel)] on win32
Type "help", "copyright", "credits" or "license
>>>
============== RESTART: C:\Users\hadeel\Deskto
inter number int :  5.5
that was no vaild int, try again please
inter number int :  p
that was no vaild int, try again please
inter number int :  s
that was no vaild int, try again please
inter number int :  8
>>>
```



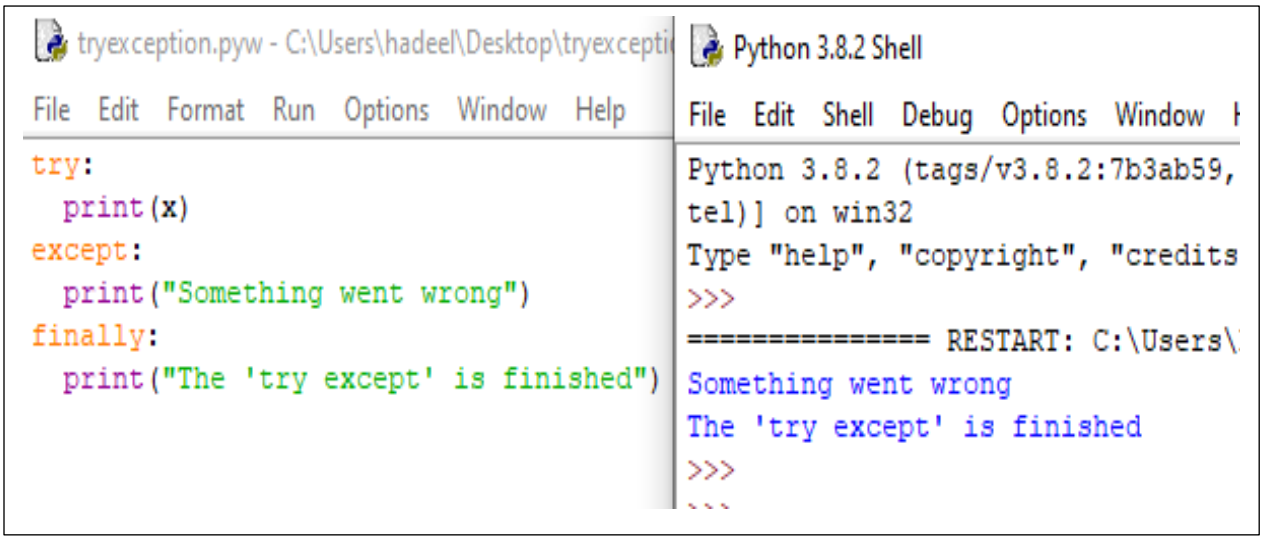
في هذا المثال ظهرت عبارة (except) بسبب ان (m) غير معرفة فكان من المفروض ان البرنامج يجب ان يظهر error لو لا وجود except .

في حال وجود اكثر من استثناء(except) حيث ان البرنامج سيقوم بطباعة عبارة واحدة اي يأخذ استثناء واحد فقط وفي المثال ادناه تم التعرف على ان الاسم خطا بسبب المتغير غير معرف سلفا في البداية .



هناك أنواع مختلفة من الاستثناءات منها : ZeroDivisionError و NameError و TypeError ، حيث تصف رسالة الخطأ نوع الاستثناء.

اما الfinally يتم تنفيذها بغض النظر عن try هل هي error or not(في حال تم تنفيذ try اوexcept فالنهاية (finally)تنفذ).



وملخص موضوع الاستثناءات كالاتي:

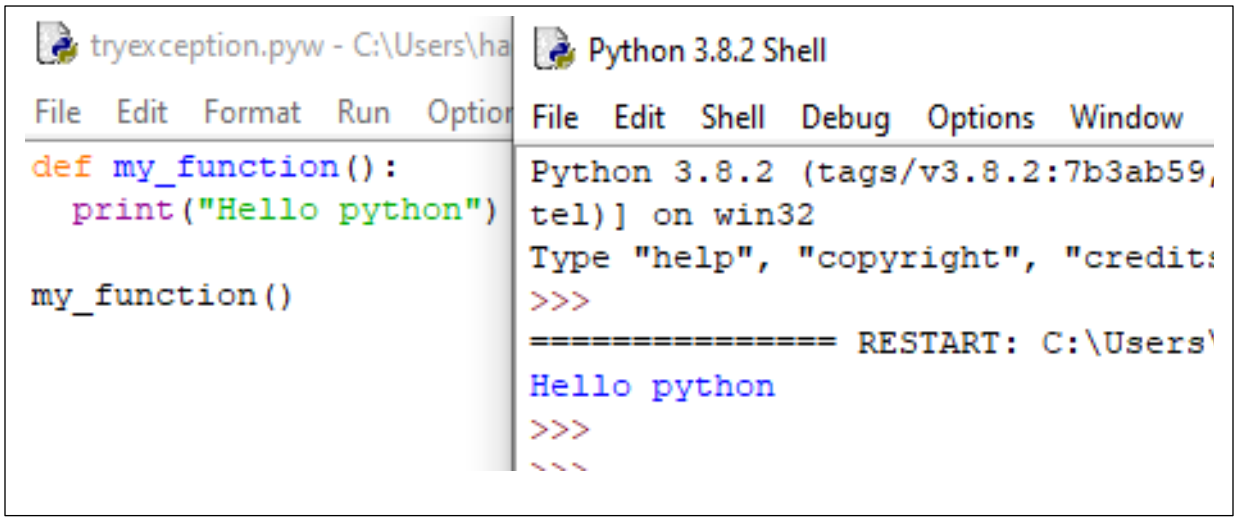
Tryهي المحاولة لاختبار تعليمات البرمجية معينة في حال ان فيها خطا سنستخدم except والتي تتيح لك استثناء معالجة الخطأ واخيرا finally تتيح لك تنفيذ التعليمات البرمجية ، بغض النظر عن نتيجة المحاولة او الاستثناء.

Python Functions

الدوال في البايثون

الدوال في البايثون هي مجموعة من العبارات البرمجية ذات صلة بالبرنامج وتؤدي مهمة معينة. ان الدوال تعمل على ترتيب البرنامج وتجعله اكثر تنظيما حيث تقسم البرنامج الى اجزاء منظمة لتجنب التكرار ويجعل الدالة قابلة لإعادة الاستخدام في حال احتياجنا لتنفيذها.

لإنشاء دالة نستخدم (def) لبداية تعريف الدالة :

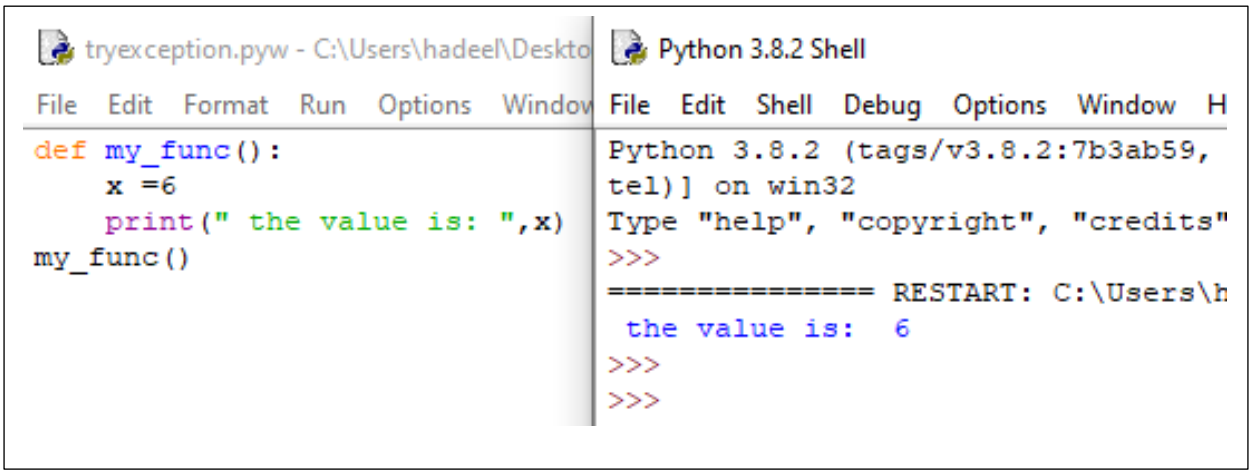


يحتوي المثال اعلاه على المكونات التالية:

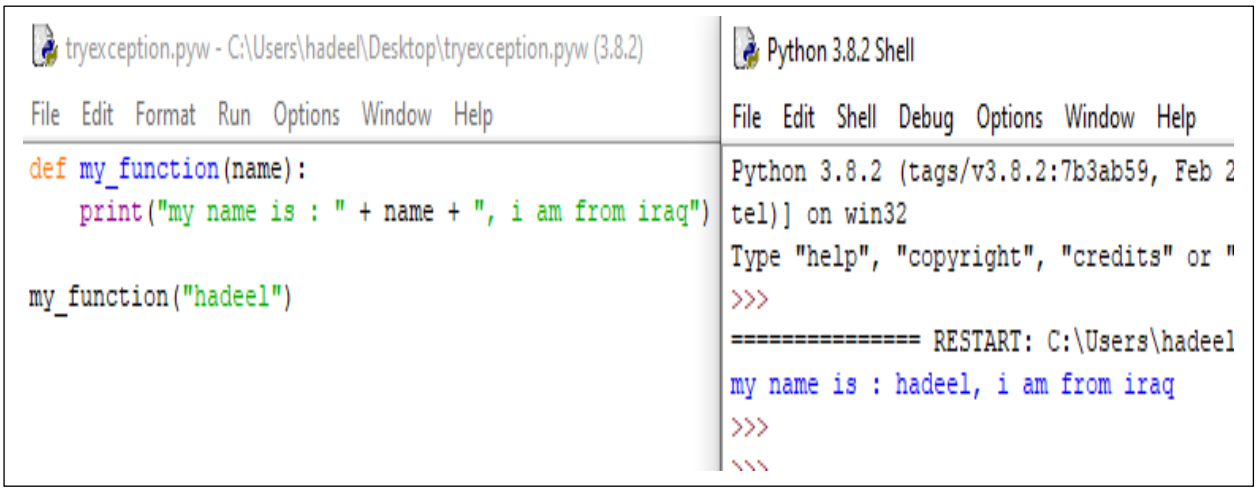
- الكلمة المفتاحية (def)التي تحدد بداية الدالة بعدها يجب ان يتبعها اسم تلك الدالة ثم الاقواس المدورة (). يكون داخل الاقواس المدورة ما يدعى ب(Parameter) ويقوم Parameter لتمرير قيم الى الدالة ثم توضع نقطتان (:) لوضع علامة على نهاية رأس الدالة.

- داخل هذه الدالة التي تم تعريفها مسبقا باسم my_function تأتي عبارات الstatements قد تكون واحدة كما في مثالنا اعلاه وهي مجرد طباعة فقط او اكثر من statements لكن يجب ان ننتبه الى المسافة البادئة وهي عادة ما تكون (٤ مسافات).
- واخيرا استدعاء الدالة حيث نستخدم اسم الدالة متبوعًا بأقواس ولا توجد مسافة بادئة لان الاستدعاء خارج الدالة.

مثال اخر عن الدالة:



لكن في حال تم وضع (Parameter) للدالة:

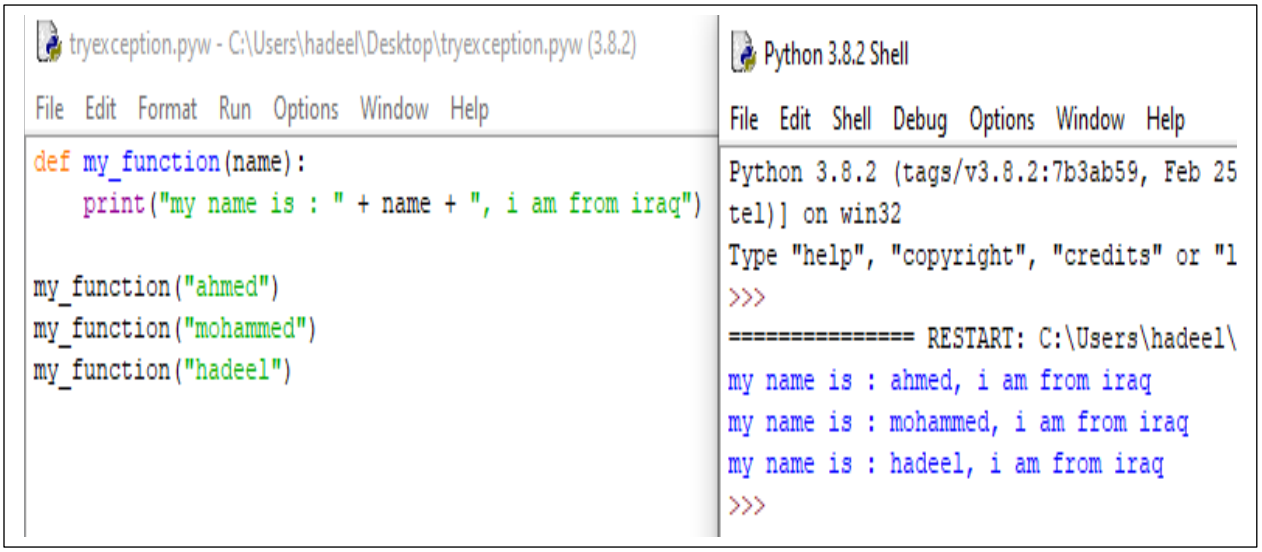


عند استدعاء الدالة نقوم بتمرير معلومات معينة من خلال الParameter حيث نلاحظ في المثال اعلاه تم استدعاء الدالة من خلال تمرير الاسم الأول ، والذي سيتم استخدامه داخل الدالة لطباعة الاسم داخل جملة الطباعة .

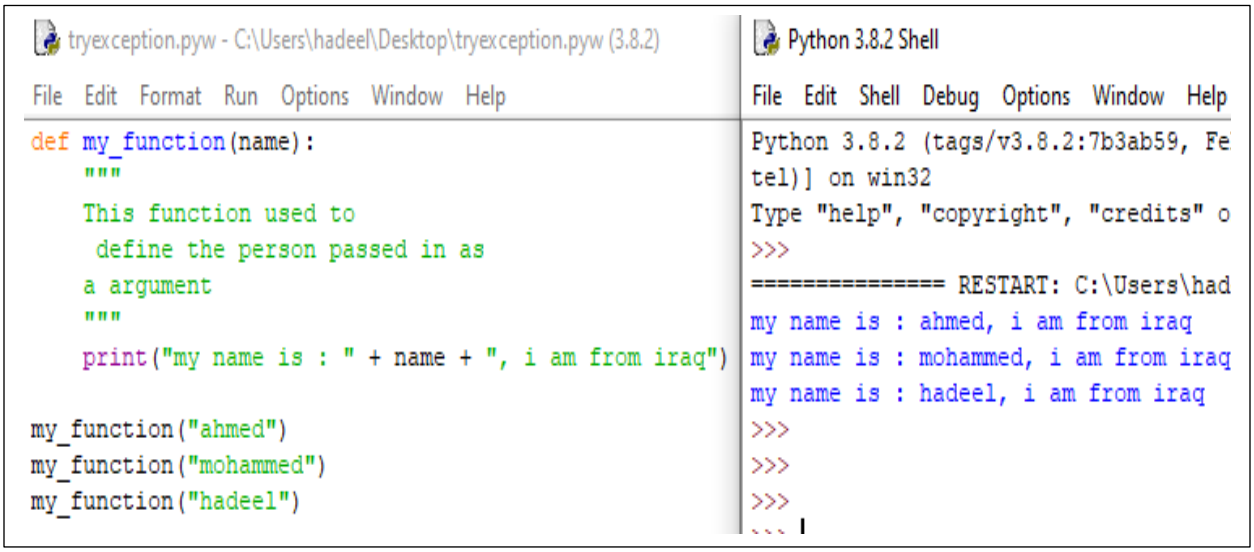
لكن هل يمكن استدعاء الدالة اكثر من مرة ؟ وهل نستطيع تمرير معلومات مختلفة من خلال Parameter ؟

الاجابة هي نعم يمكننا استدعاء الدالة باي وقت نحتاجها ويمكن تمرير معلومات مختلفة في Parameter كما التالي:

حيث سنستدعي الدالة اكثر من مرة بمعلومات مختلفة في Parameter لنفس الدالة السابقة



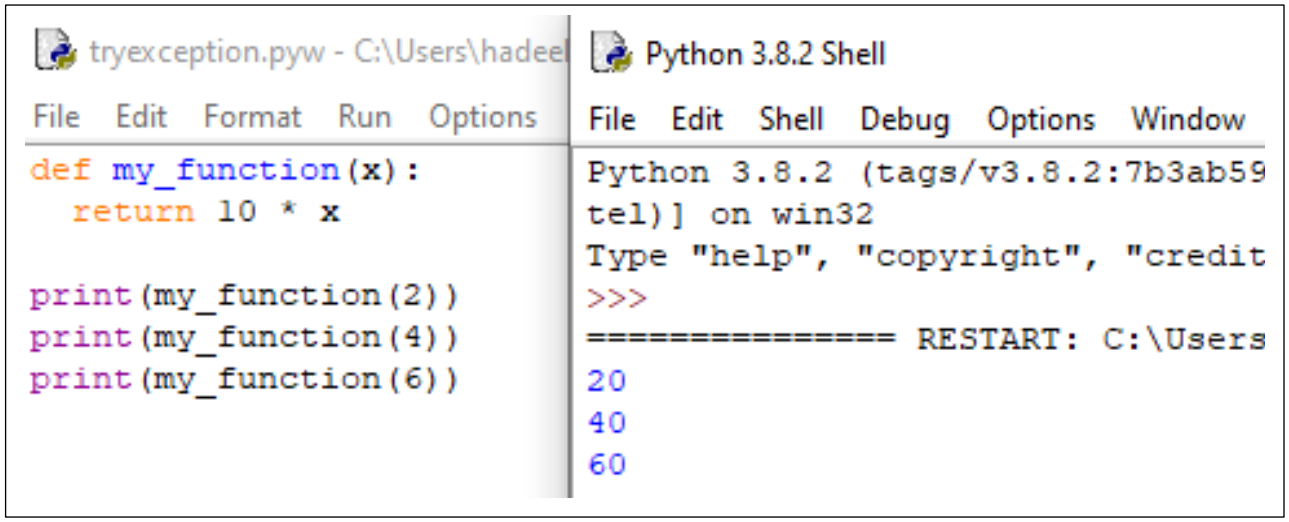
فضلا عن ذلك يمكن اضافة تعليقات توضيحية للمبرمج او المستخدم داخل الدالة :



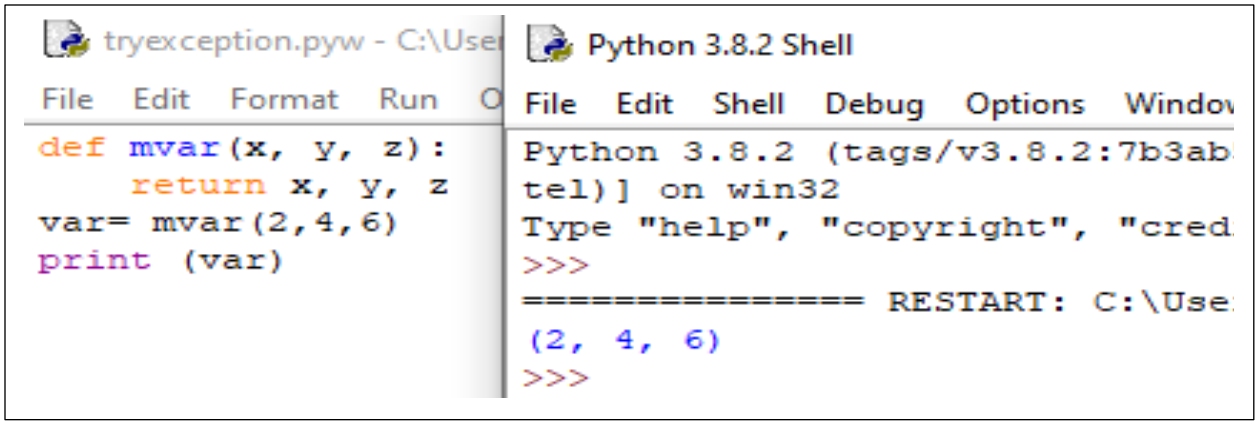
وكما تعلمنا مسبقا ان التعليقات لا تظهر في النتائج لأنها فقط تستخدم كتوضيح للمبرمج او المستخدم .

كما يمكن ان تحتوي الدالة على return وكما في اللغات البرمجية الاخرى حيث تعمل return على ارجاع قيمة معينة للدالة , مع الاخذ بنظر الاعتبار ان return قد تكون optional اي اختيارية وليست اجبارية لوضعها في كل دالة , سيكون تركيب الدالة مع return كالاتي:

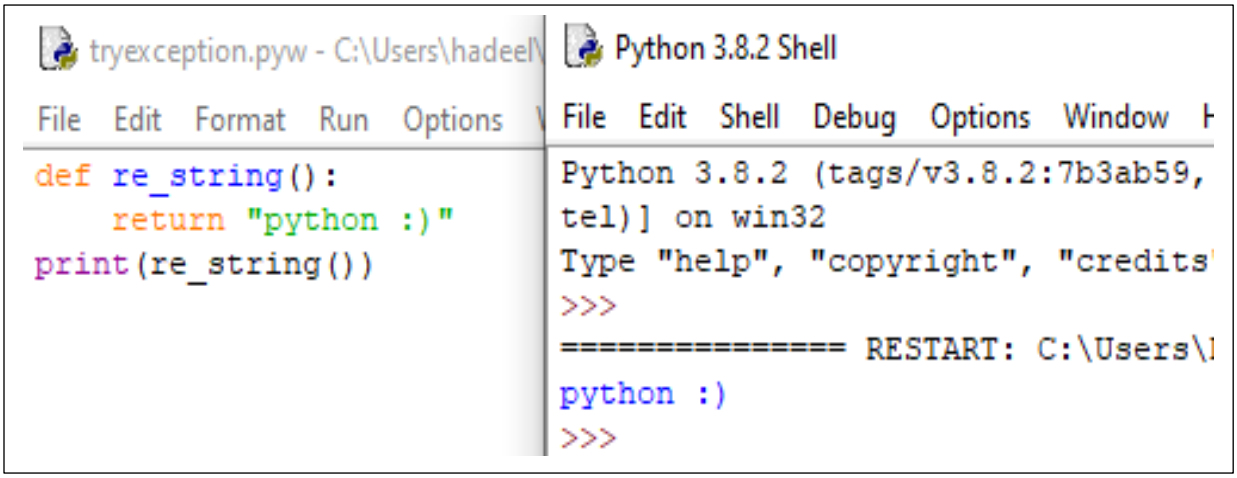
```
def function_name( parameters ):
   function_suite
  return [expression]
```



حيث نلاحظ ان الدالة سمحت بإرجاع قيمة باستخدامك عبارة الإرجاع لكن هل return تعمل على إرجاع قيمة واحدة ام يمكن ان تعمل على ارجاع اكثر من قيمة ؟ الجواب : كلا لأنه مهما فعلنا فسوف يكون الارجاع قيمة واحدة كالاتي:

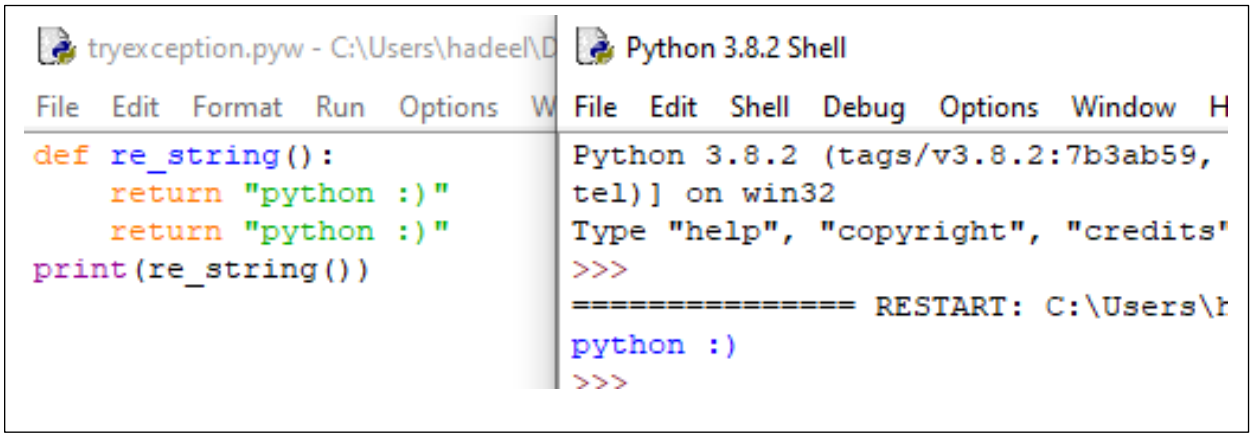


قام بإرجاع قيمة واحدة على شكل tuple على الرغم اننا ادخلنا قيم ثلاثة مختلفة , فضلا عن ذلك يمكن ارجاع جملة string وليس فقط ارقام وكما في المثال التالي:

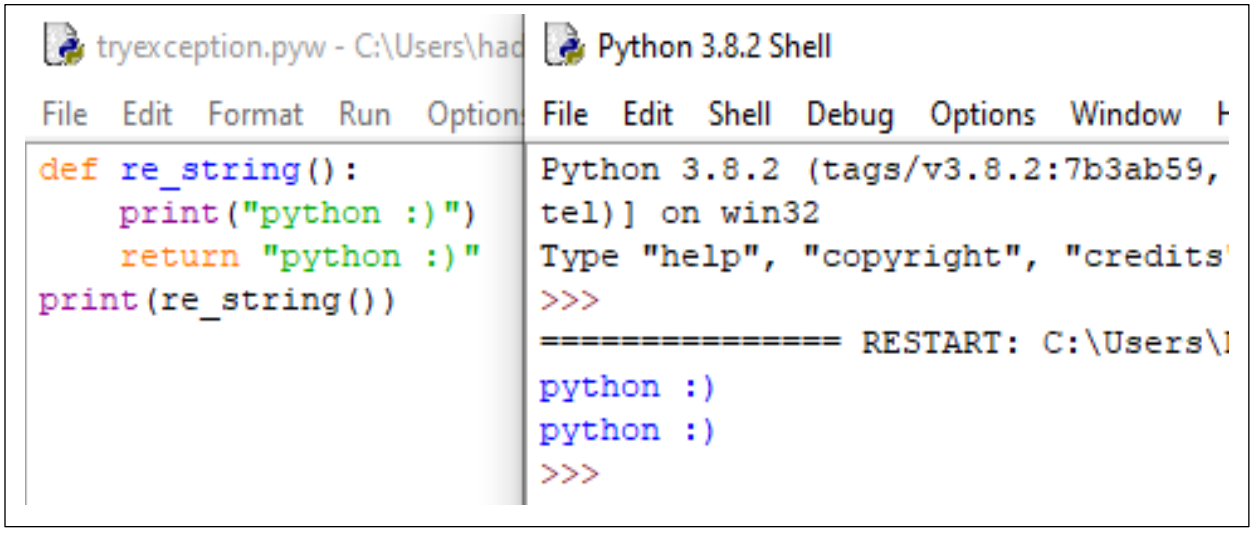


قد يتبادر الى ذهنك في حال قمنا بوضع return مرتين في داخل الدالة function هل يتم تنفيذ الاثنان معا ؟

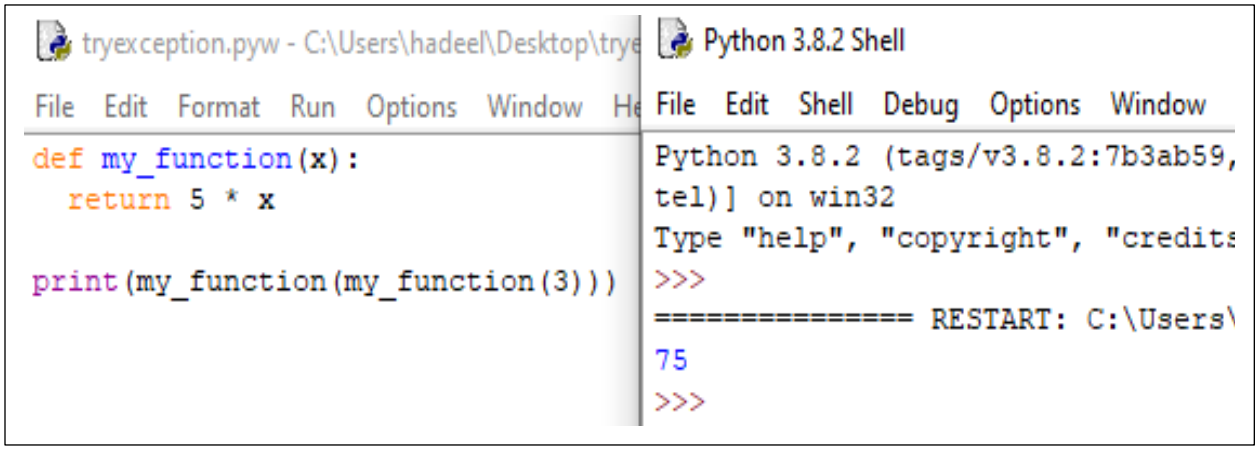
الحقيقة ان البايثون ستنفذ return واحدة فقط :



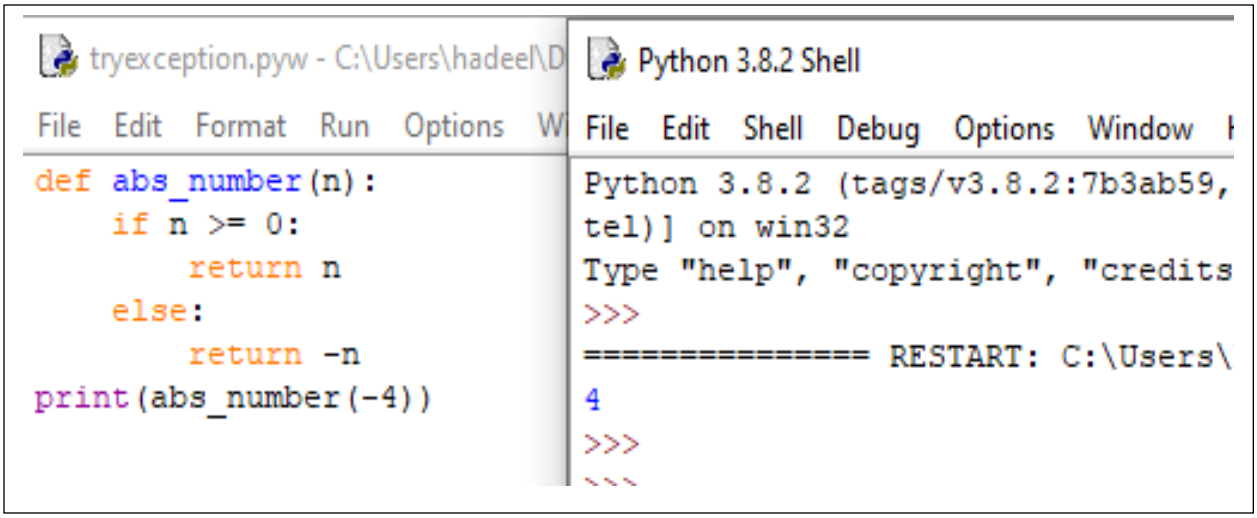
واخيرا يمكن ان تكون دالة الطباعة في الدالة مع وجود الارجاع (اي لا مانع من وجود الاثنين معا):



يمكن ايضا استدعاء الدالة لأكثر من مرة (اي متداخلة) حيث يمكن تنفيذها باي وقت نحتاج اليها كما في المثال ادناه:



ومن الطبيعي القول اننا نستطيع وضع for و if و while التي تم شرحها فيما سبق وفي ادناه بعض الامثلة لإضافتهن داخل ال function :



tryexception.pyw - C:\Users\hadeel\D

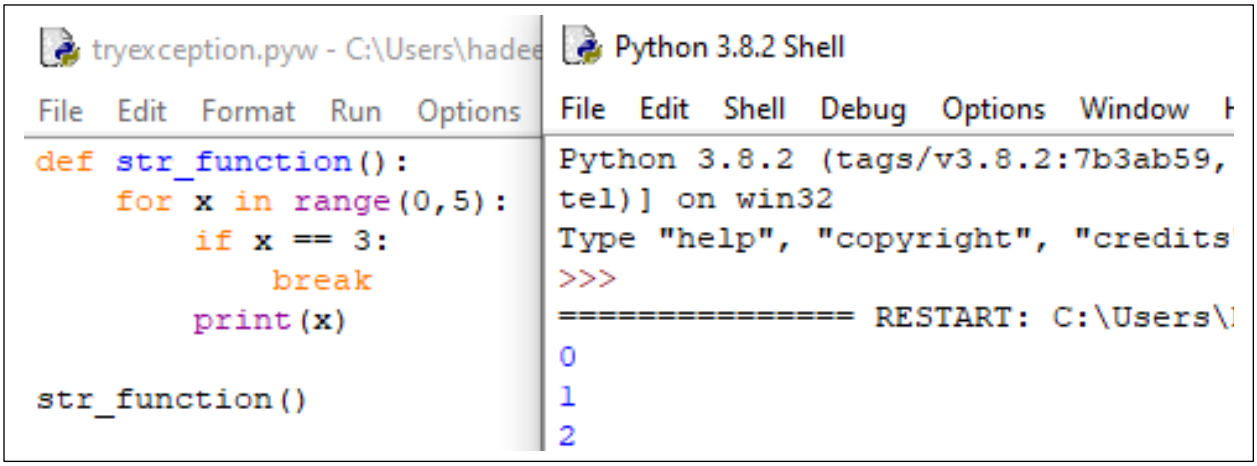
File Edit Format Run Options W

```
def abs_number(n):
    if n >= 0:
        return n
    else:
        return -n
print(abs_number(-4))
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window H

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits
>>>
========== RESTART: C:\Users\
4
>>>
```



tryexception.pyw - C:\Users\hadee

File Edit Format Run Options

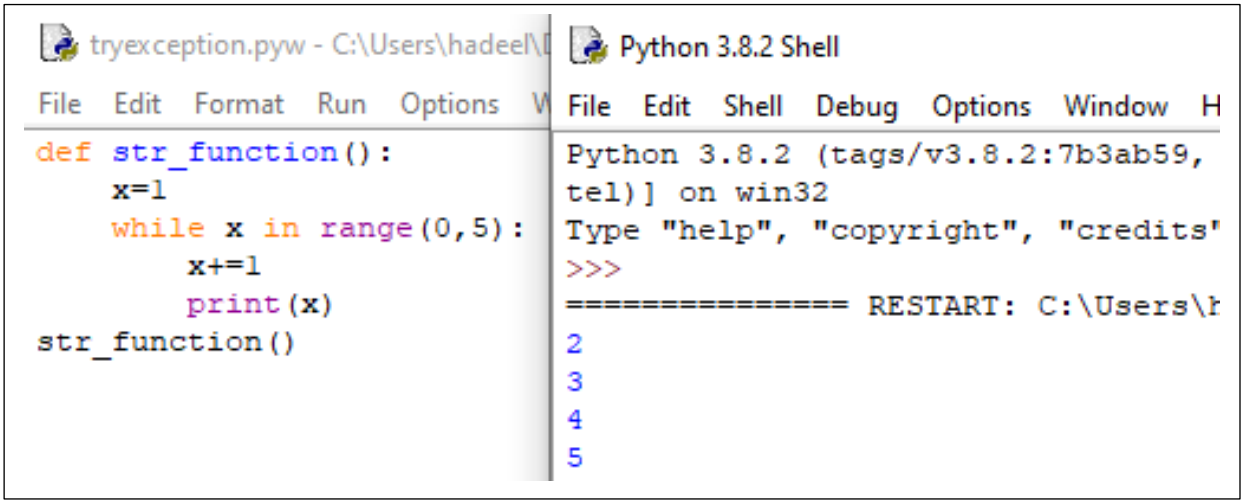
```
def str_function():
    for x in range(0,5):
        if x == 3:
            break
        print(x)

str_function()
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window H

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits
>>>
========== RESTART: C:\Users\
0
1
2
```



tryexception.pyw - C:\Users\hadeel\

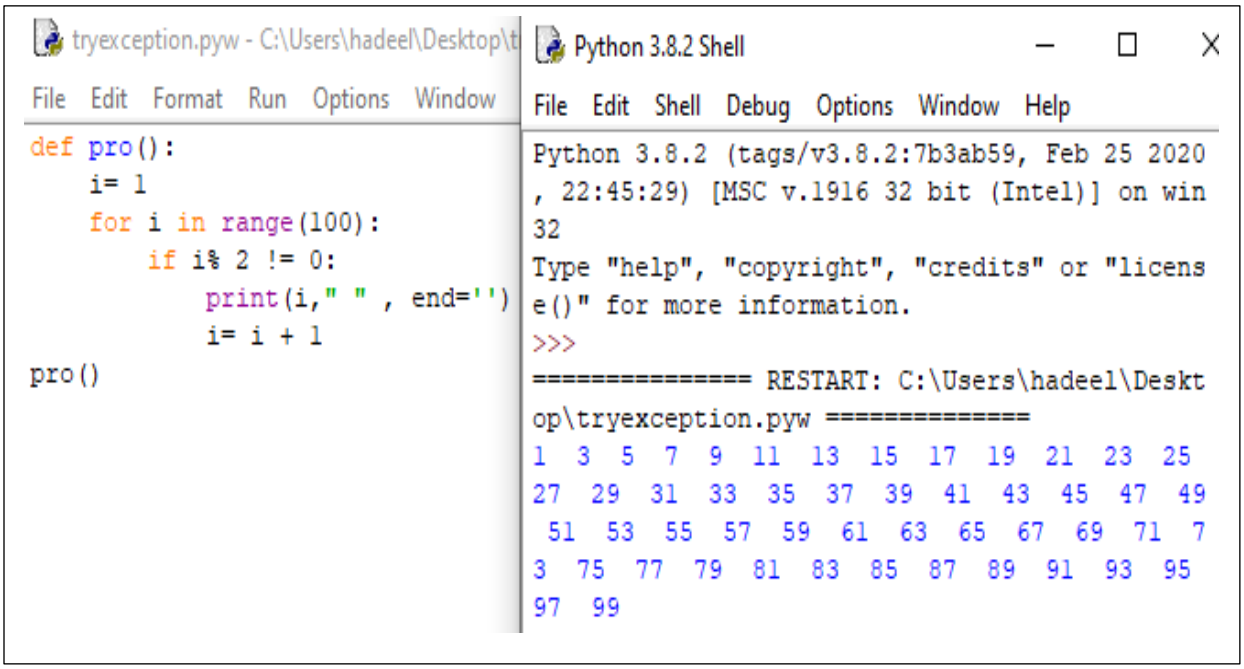
File Edit Format Run Options W

```
def str_function():
    x=1
    while x in range(0,5):
        x+=1
        print(x)
str_function()
```

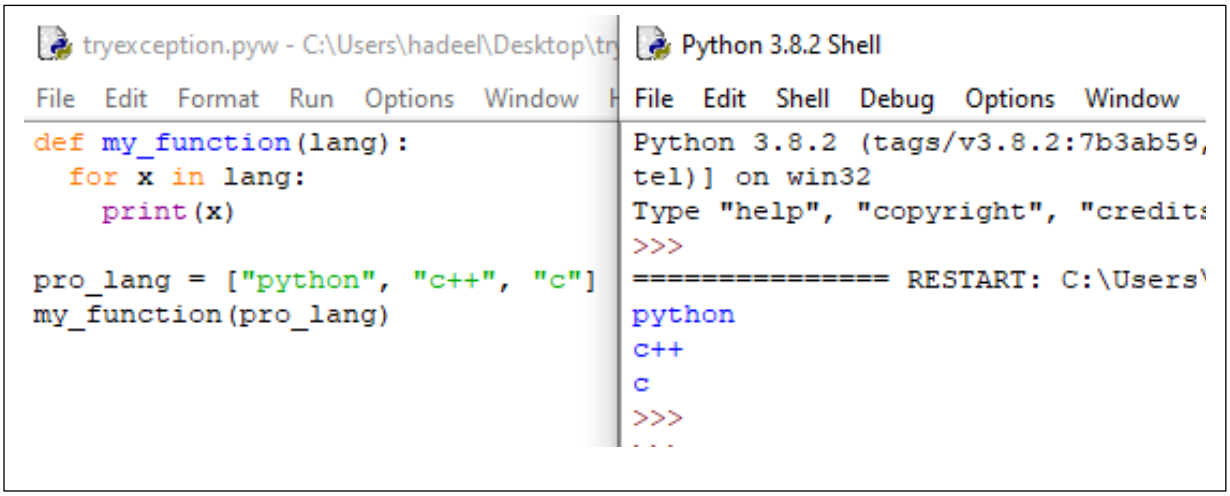
Python 3.8.2 Shell

File Edit Shell Debug Options Window H

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits"
>>>
========== RESTART: C:\Users\h
2
3
4
5
```



يمكننا ايضا ادراج أي نوع من أنواع البيانات إلى دالة (سلسلة ، رقم ، قائمة ، قاموس ، إلخ) ، وسيتم التعامل داخل الدالة function كالتالي:



الان استخدمنا القائمة list التي قمنا بإضافتها خارج الدالة وهي اللغات البرمجية ثم قمنا بإدراجها داخل الدالة ثم طباعتها بمعنى ان (pro_lang= lang) . اي في البايثون يمكن كتابة القائمة قبل الدالة function وفي الامثلة ادناه تمثيل ذلك حيث قمنا بكتابة القائمة قبل الدالة ثم طباعة ما نفعله في القائمة كما تم شرحه سابقا مثل طول القائمة او كيفية اضافة عنصر بنهاية القائمة وطباعة القائمة الجديدة بعد الاضافة :

tryexception.pyw - C:\Users\hadeel\Desktop\tryexcep

File Edit Format Run Options Window Help

```
my_list= [ "python","c++","c","java"]
def my_function(list):
    print(list)
    print(list[0])
    print(list[-1])
    print(list[3:5])
    print(list[:3])
    print(list[3:])
    print(len(list))
    list.append("z")
    print(list)

my_function(my_list)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>>
============== RESTART: C:\Users\hade
['python', 'c++', 'c', 'java']
python
java
['java']
['python', 'c++', 'c']
['java']
4
['python', 'c++', 'c', 'java', 'z']
>>>
>>>
```

tryexception.pyw - C:\Users\hadeel\Desktop\tryexception.pyw

File Edit Format Run Options Window Help

```
dic_function = {1:"one", 2:"two",3:"three"}
def my_function(collection):
        for key in collection:
            print(float(key))
my_function(dic_function)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits
>>>
============== RESTART: C:\Users\
1.0
2.0
3.0
>>>
```

tryexception.pyw - C:\Users\hadeel\Desktop\tryexception.pyw (3.8.2)

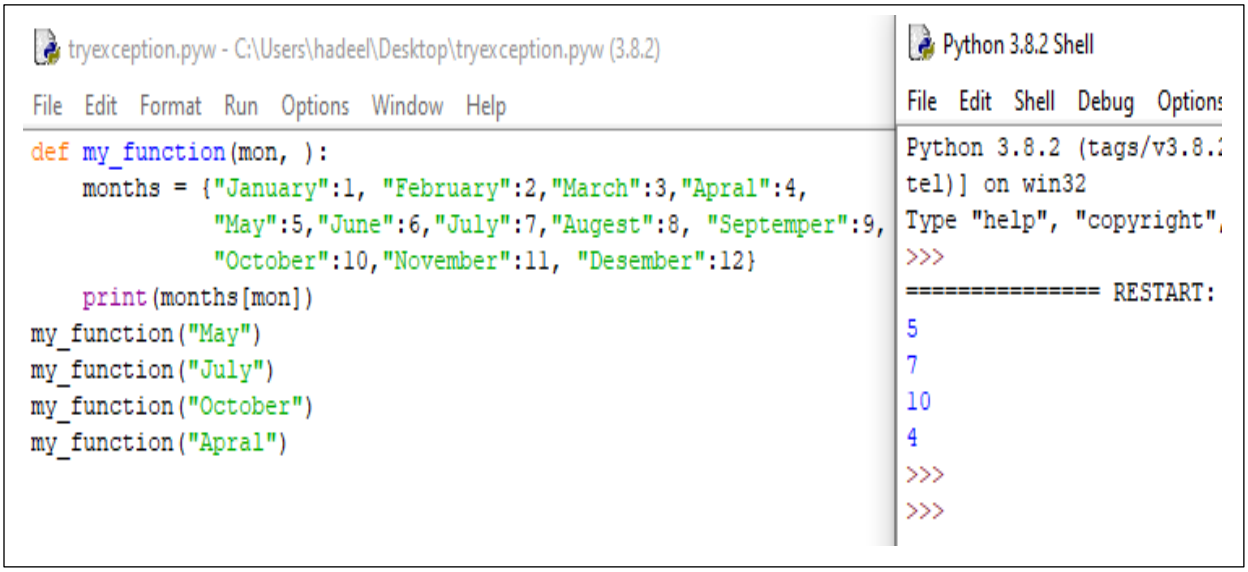
File Edit Format Run Options Window Help

```
def my_function(mon, ):
    months = {"January":1, "February":2,"March":3,"Apral":4,
              "May":5,"June":6,"July":7,"Augest":8, "Septemper":9,
              "October":10,"November":11, "Desember":12}
    print(months[mon])
my_function("May")
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits
>>>
============== RESTART: C:\Users\
5
>>>
>>>
```



tryexception.pyw - C:\Users\hadeel\Desktop\tryexception.pyw (3.8.2)

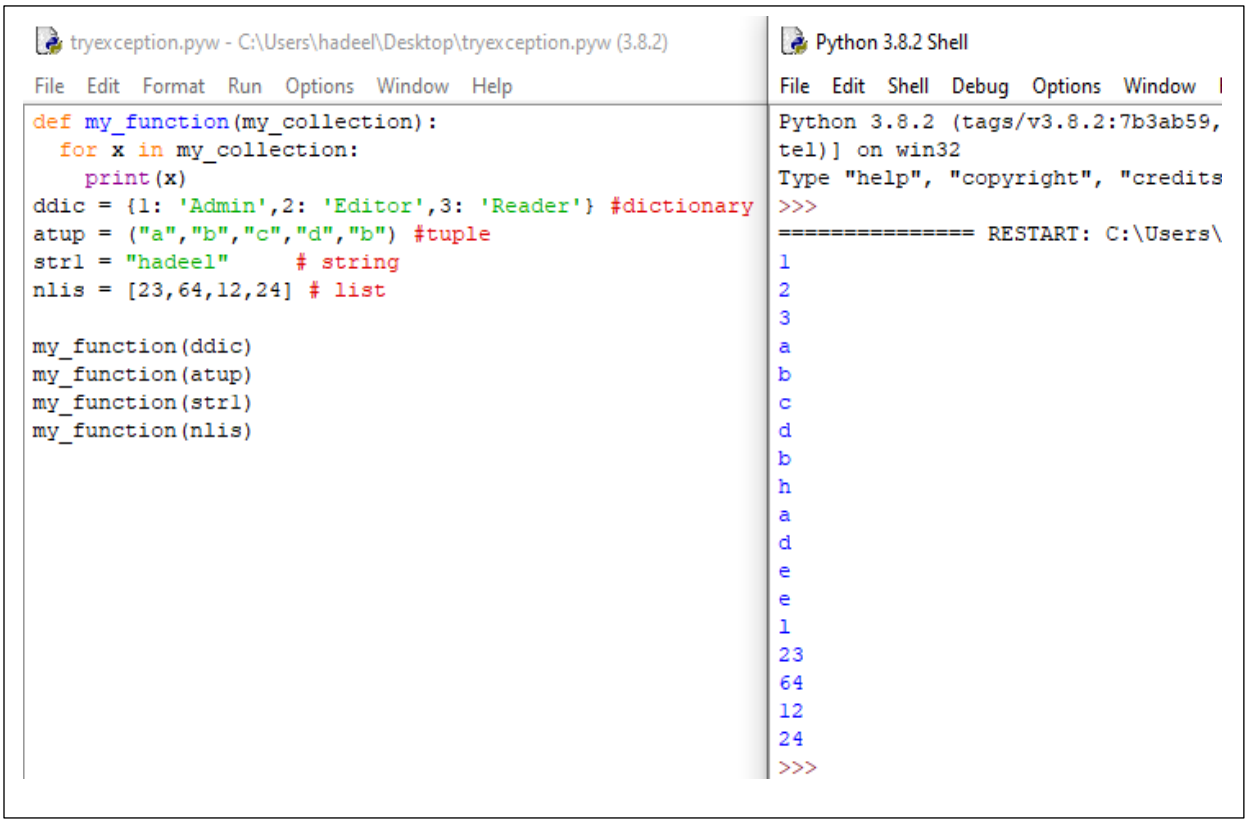
File Edit Format Run Options Window Help

```
def my_function(mon, ):
    months = {"January":1, "February":2,"March":3,"Apral":4,
              "May":5,"June":6,"July":7,"Augest":8, "Septemper":9,
              "October":10,"November":11, "Desember":12}
    print(months[mon])
my_function("May")
my_function("July")
my_function("October")
my_function("Apral")
```

Python 3.8.2 Shell

File Edit Shell Debug Options

```
Python 3.8.2 (tags/v3.8.2
tel)] on win32
Type "help", "copyright",
>>>
========== RESTART:
5
7
10
4
>>>
>>>
```



tryexception.pyw - C:\Users\hadeel\Desktop\tryexception.pyw (3.8.2)

File Edit Format Run Options Window Help

```
def my_function(my_collection):
  for x in my_collection:
    print(x)
ddic = {1: 'Admin',2: 'Editor',3: 'Reader'} #dictionary
atup = ("a","b","c","d","b") #tuple
strl = "hadeel"     # string
nlis = [23,64,12,24] # list

my_function(ddic)
my_function(atup)
my_function(strl)
my_function(nlis)
```

Python 3.8.2 Shell

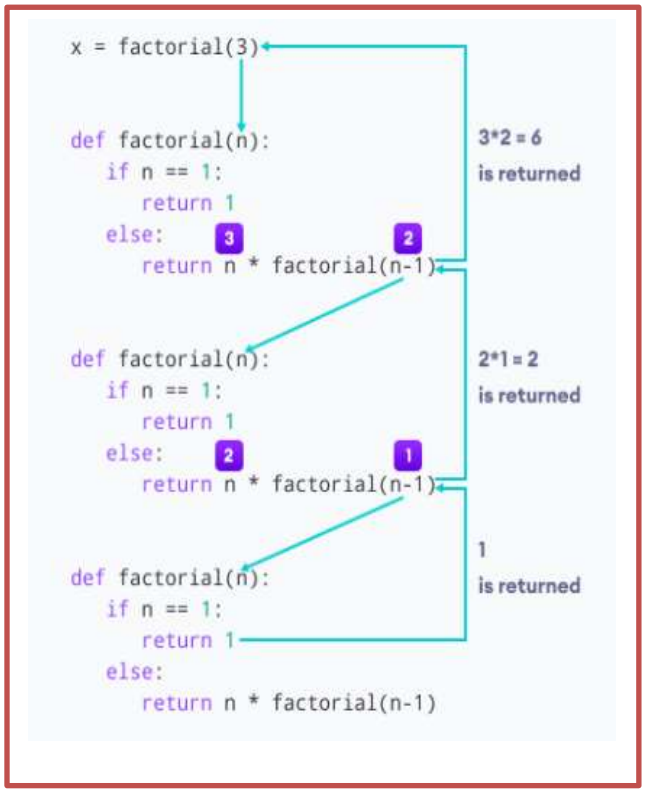
File Edit Shell Debug Options Window

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits
>>>
========== RESTART: C:\Users\
1
2
3
a
b
c
d
b
h
a
d
e
e
l
23
64
12
24
>>>
```

اما الان فيجب ان نوضح مفهوم مهم في اغلب اللغات البرمجية وهي ان الدالة تستدعي نفسها وهذا ما يعرف بالمصطلح Recursion(a function that calls itself) تكون تركيبة عملها كالاتي:

```
def recurse():
    .......
    recurse()
recurse()
```

التكرار له فائدة حيث أنه يمكننا تكرار البيانات للوصول إلى نتيجة. مع الواجب الحذر من التعامل مع تكرار البيانات. والمثال ادناه وهو عن factorial فمثلا fact(3) سيكون حسابه بالرياضيات 1*2*3 لنشرح طريقة ال factorial باستخدام التكرار بالخطوات التالية للسهولة:



اما برمجيا ستكون كالاتي :

python.pyw - C:/Users/hadeel/Desktop/python.pyw (3.8.2)

File Edit Format Run Options Window Help

```
def my_fact(n):
    if n == 1:
        return 1
    else:
        return (n * my_fact(n-1))

x = 6
print("The factorial of","(", x ,")", "is : ", my_fact(x))
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window H

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
tel)] on win32
Type "help", "copyright", "credits"
>>>
================== RESTART: C:/User
The factorial of ( 6 ) is :  720
>>>
>>>
```

مثال اخر عن تكرار الدالة لذاتها recursive Python function للجمع في حال ادخال رقم مثلا ٦ يكون جمعه ٦+٥+٤+٣+٢+١ فيكون الناتج ٢١ (هذا البرنامج شبيه لل factorial لكن هنا جمع وليس ضرب)

python.pyw - C:\Users\hadeel\Desktop\pyt

File Edit Format Run Options Window

```
def sumofn(n):
        if n==1:
                return 1
        return (n+sumofn(n-1))

print(sumofn(2))
print(sumofn(6))
print(sumofn(10))
```

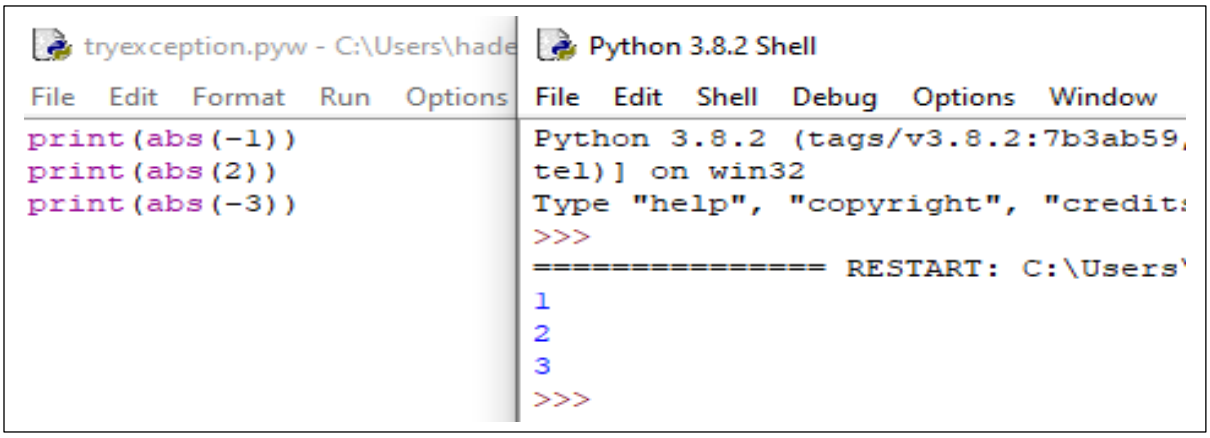
Python 3.8.2 Shell

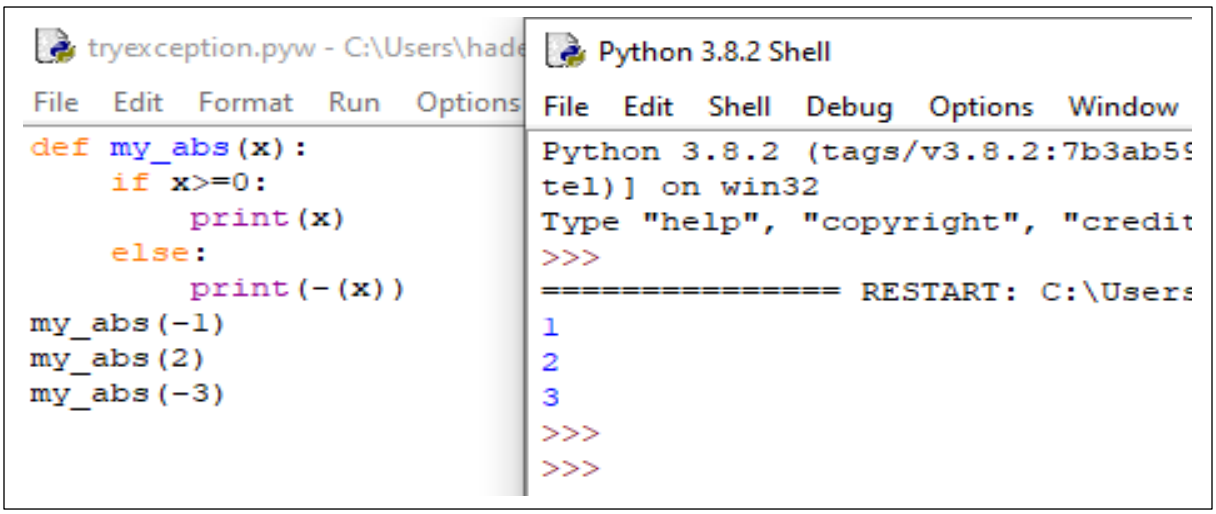
File Edit Shell Debug Options Window Hel

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, F
tel)] on win32
Type "help", "copyright", "credits"
>>>
================== RESTART: C:\Users
3
21
55
>>>
>>>
```

وملخص **Function** هي عبارة عن: كود بسيط يستخدم لتنفيذ مهمة معينة ويمكن استدعائه باي وقت وباي مكان داخل البرنامج. ومن خلال الشرح اعلاه يمكننا تقسيم الدالة (function) الى نوعان :

- الاول هو Built-in functions اي الدوال المبنية من قبل البايثون وابسط مثال لهذا النوع هي print() حيث يمكننا كتابة text داخل الدالة سيتم طباعته على الشاشة بمجرد التنفيذ. اضافة الى دوال جاهزة اخرى مثل append , abs(), min(), max(), sort() ..... الخ من الدوال.
- النوع الثاني هو User-defined functions هي الدوال التي تكتب من قبل المستخدم اي قوم ببناء الدالة بنفسه وسوف نوضح النوعين بمثال بسيط وهو العدد المطلق (اي يمكننا عمل مطلق لعدد معين من خلال دالة تم بنائها من قبل البايثون وهي abs() ومن جهة الاخرى يمكننا بناء دالة تعمل بالطريقة ذاتها تم أنشائها من قبل المستخدم والمثال ادناه سيوضح الفرق :





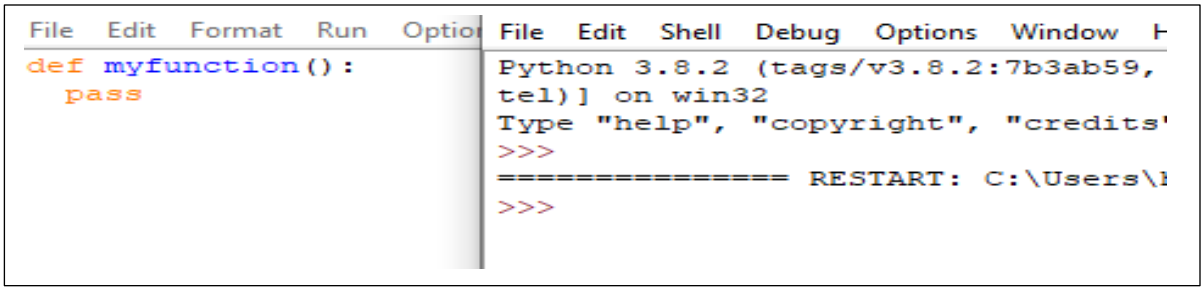
ملاحظات مهمة في الFunction

ملاحظة الاولى : هل ما نكتبه داخل اقواس الدالة هو Parameters ام Arguments ؟ في الحقيقة ان لهما نفس المعنى اي ببساطة انهما يستخدمان لتمرير المعلومات(information) داخل function لكن في حال ان اردنا تعريفها من وجهة نظر الدالة فيمكن القول ان :

- Parameter: هو المتغير(variable) المذكور داخل الأقواس في تعريف function.
- Argument: هي القيمة التي يتم إرسالها إلى الدالة عندما يتم استدعاؤها.

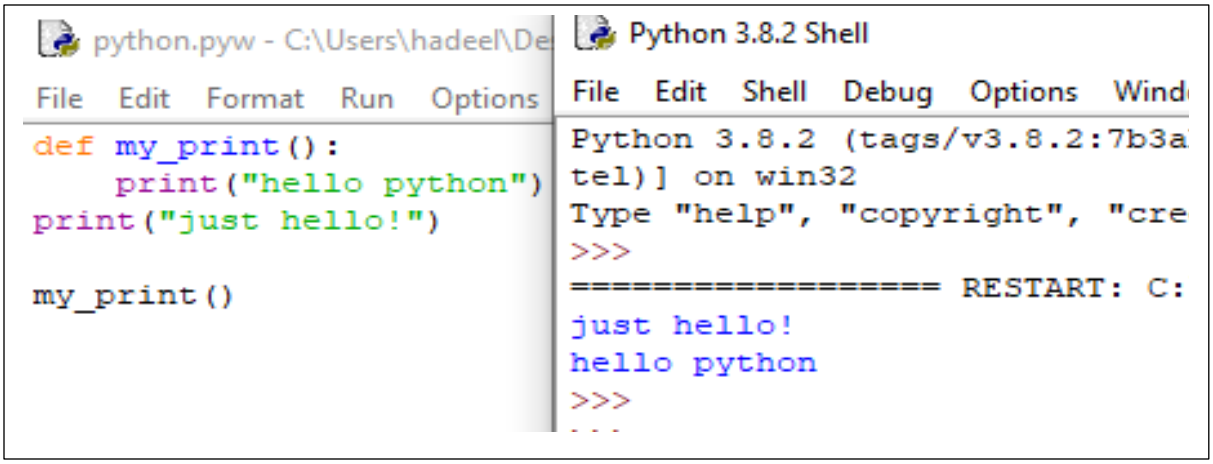
فلذا ارجوا التساهل معي في حال وجود Parameter بدلا من Argument او العكس في الكتاب لأنه تم اخذ المعنى العام .. والان من باب الامانة تم ذكر المعنيين .

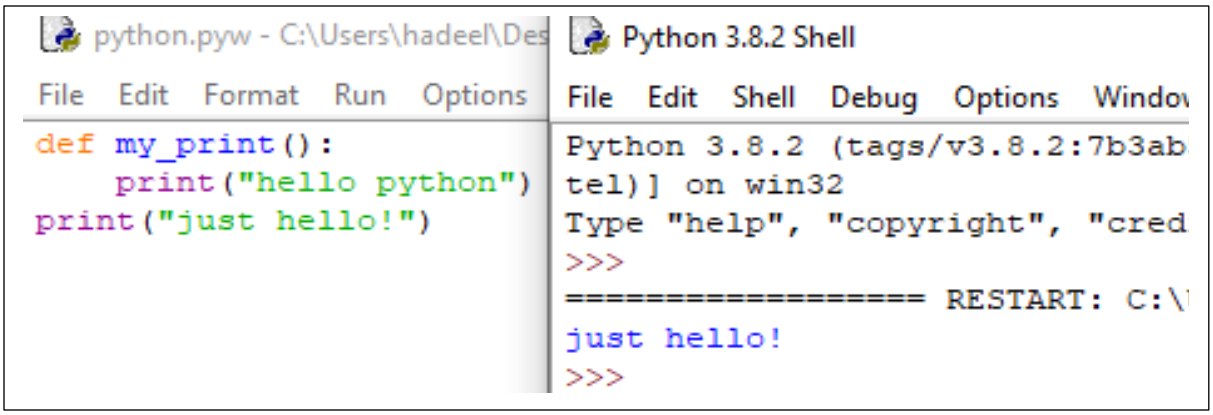
ملاحظة الثانية : هناك بعض function لا يوجد فيها محتوى (فارغة) ولضمان عدم حصول error خطأ يتم كتابة pass داخل الدالة .

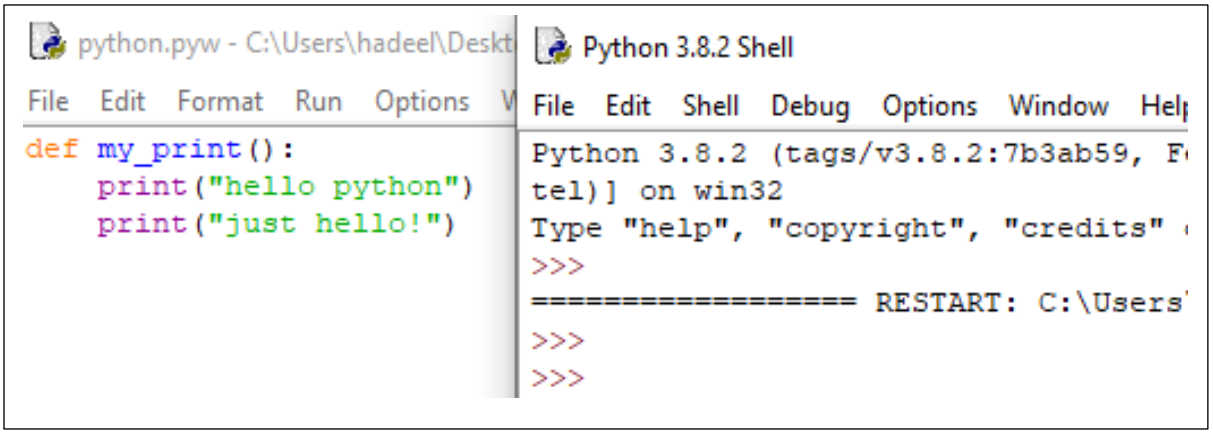


ملاحظة الثالثة : المسافات و المسافات ثم المسافات في الدوال وجودها مهم جدا في البايثون وهذه الملاحظة مكررة لكن لأهميتها نعيدها ونكررها حيث نشاهد في المثالين ادناه اهمية المسافة فالأول عند استدعاء function يتم طباعة function و دالة print() اما في المثال الثاني لم يتم استدعاء function نلاحظ انه تم تنفيذ البرنامج بدون error من خلال دالة الطباعة فقط حيث تم تنفيذها باعتبارها خارج عن الfunction.

لكن في حال تم اضافة مسافة (4 spaces) ولم يتم استدعاء function سوف لن يتم طباعة اي شيء على الشاشة كما في المثال الاخير :


python.pyw - C:\Users\hadeel\De
File Edit Format Run Options
def my_print():
print("hello python")
print("just hello!")
my_print()
Python 3.8.2 Shell
File Edit Shell Debug Options Wind
Python 3.8.2 (tags/v3.8.2:7b3a
tel)] on win32
Type "help", "copyright", "cre
>>>
========== RESTART: C:
just hello!
hello python
>>>


python.pyw - C:\Users\hadeel\Des
File Edit Format Run Options
def my_print():
print("hello python")
print("just hello!")
Python 3.8.2 Shell
File Edit Shell Debug Options Windo
Python 3.8.2 (tags/v3.8.2:7b3ab
tel)] on win32
Type "help", "copyright", "cred
>>>
========== RESTART: C:\
just hello!
>>>


python.pyw - C:\Users\hadeel\Deskt
File Edit Format Run Options
def my_print():
print("hello python")
print("just hello!")
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, F
tel)] on win32
Type "help", "copyright", "credits"
>>>
========== RESTART: C:\Users
>>>
>>>

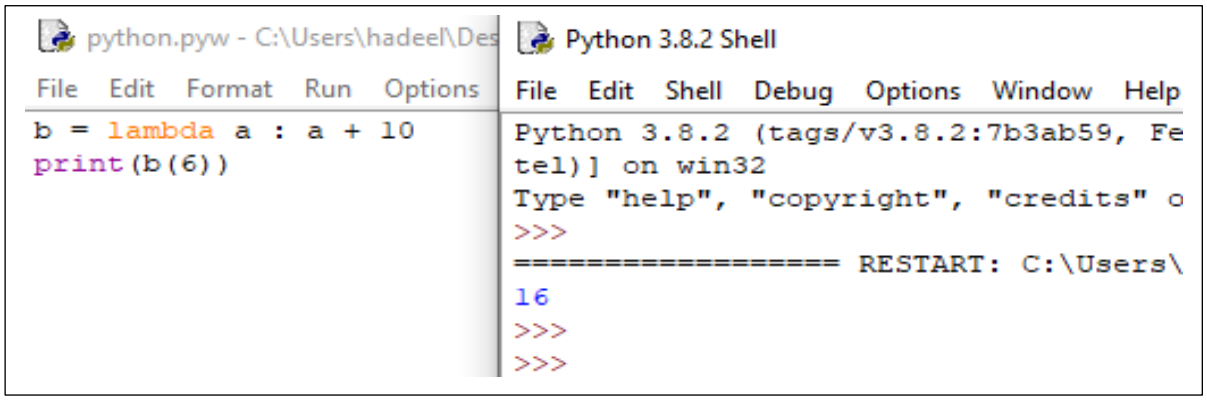
Lambda in Python

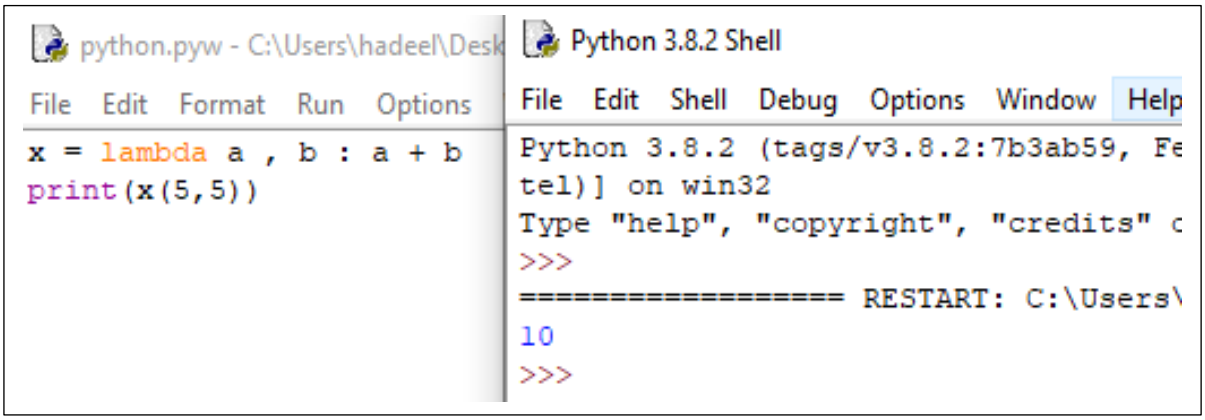
Lambda في البايثون

يمكن تعريفLambda بانها تعريف لدالة ليس لها اسم Anonymous Function , غالبا ما تكون هذه الدالة من سطر واحد وتعمل على ارجاع قيمة معينة عند استدعائها من خلال تعريف الfunction باستخدام كلمة lambda ولا نستخدم كلمة def التي تعودنا استخدامها في تعريف الدوال..

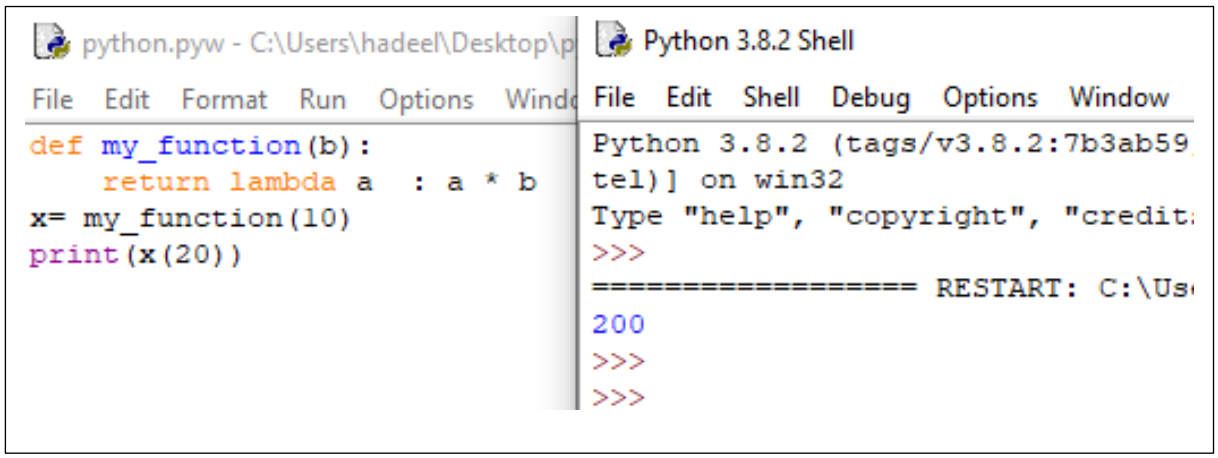
من مميزات lambda انها تحتوي على اي عدد من arguments لكن تحتوي على تعبير واحد(expression) فقط ويمكن تمثيل تركيبتها كالاتي :

```
lambda [arg1 [,arg2,.....arg n]]:expression
```





يمكن لهذه الدالة استخدامها بداخل دوال اخرى وهذه هي حدى ميزاتها ايضا :



نلاحظ اعلاه ان lambda تم دمجها او استخدامها بداخل دالة my_function .

هنالك العديد من المواضيع المتنوعة في البايثون منها Python Module وPython Class و – Python و Python Inheritance..... الخ. لكن قد يكون الافضل شرحها في المرحلة القادمة بعد يتم تعلم ما تم شرحه سابقا.

وفي النهاية اتمنى ان تكون هذه الصفحات القليلة

مفيدة لكم

والحمد لله رب العالمين والصلاة والسلام على سيد المرسلين محمد (صلى الله عليه وسلم)

Hadeel


Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb
tel)] on win32
Type "help", "copyright", "credits" or
>>> print(" تحياتي ..... لكم شكرا")
شكرا لكم ..... تحياتي
>>>
>>>
python