المحتويات

- ✓ التعامل مع الدالة ( )int
- ✓ العلاقة بين الكلاس والكائن
- ✓ فائدة الكلاس
- ✓ التعامل مع التاريخ والوقت
- ✓ التعامل مع الملفات
- ✓ دوال القراءة والكتابة في الملفات
- ✓ مدير الحزم pip

مقدمة عن لغة بايثون python

- لغة البرمجة بايثون لغة برمجية للأغراض العامة, تفاعلية, موجهة الغرض وعالية المستوى. تم إنشاء هذه اللغة على يد العالم جودي فان وزم بين عامي ١٩٨٥ – ١٩٩٠ . وكما هو الحال في لغة البرمجة بيرل فإن مصدر شيفرة لغة بايثون متوفر لدى حكومة الاتحاد الوطني (GNU) كشهادة عامة.
- مفتوحة المصدر وقابلة للتطوير. تعتبر لغة بايثون لغة تفسيرية متعددة الأغراض وتستخدم بشكل واسع في العديد من المجالات كبناء البرامج المستقلة باستخدام الواجهات الرسومية المعروفة وفي عمل برامج الويب، بالإضافة إلى استخدامها كلغة برمجة نصية للتحكم في أداء بعض من أشهر البرامج المعروفة أو في بناء برامج ملحقة لها. وبشكل عام يمكن استخدام بايثون لبرمجة البرامج البسيطة للمبتدئين ولإنجاز المشاريع الضخمة كأي لغة برمجية أخرى في نفس الوقت. غالباً ما يُنصح المبتدئين في ميدان البرمجة بتعلم هذه اللغة لأنها من بين أسرع اللغات البرمجية تعلماً.
- نشأت بايثون في مركز (Centrum Wiskunde & Informatica) **CWI** (مركز العلوم والحاسب الآلي) بأمستردام على يد جويدو فان رُزوم في أواخر الثمانينات من القرن المنصرم، وكان أول إعلان عنها في عام ١٩٩١. تم كتابة نواة اللغة بلغة C. أطلق فان رُزوم اسم "بايثون" على لغته تعبيرًا عن إعجابه بفِرقَة مسرحية هزلية شهيرة من بريطانيا، كانت تطلق على نفسها اسم مونتي بايثون بالإنجليزية(Monty Python).
- تتميز بايثون بمجتمعها النشط، كما أن لها الكثير من المكتبات البرمجية ذات الأغراض الخاصة والتي برمجها أشخاص من مجتمع هذه اللغة، مثلاً مكتبة PyGame التي توفر مجموعه من الوظائف من أجل برمجة الألعاب. ويمكن لبايثون التعامل مع العديد من أنواع قواعد البيانات مثل MySQL وغيره.
- اشتق اسم بايثون من الفرقة الكوميدية البريطانية مونتي بايثون نتيجة لتأثر فان روسم بمشاهدة عروضها أثناء عمله على تطوير اللغة.تظهر مونتي بايثون من حين لآخر في شيفرة بايثون البرمجية وثقافتها.
- تستخدم البادئة Py : للإشارة إلى أي شيء مُتعلِّق بهذه اللغة.

تطبيق Python في بعض الشركات المعروفة

- Google تم تطوير محرك تطبيقات Google و code.google.com و google earch وزاحف Google وإعلانات google ومشاريع أخرى في Python.
- CIA تم تطوير موقع CIA وكالة المخابرات المركزية باستخدام Python.
- ناسا تستخدم بايثون على نطاق واسع لتحليل البيانات والحسابات.
- YouTube أكبر موقع فيديو في العالم.
- Dropbox أكبر موقع ويب للتخزين عبر الإنترنت في الولايات المتحدة ، تم تنفيذه جميعًا في Python ، ويتعامل مع عمليات تحميل وتنزيلات تصل إلى مليار ملف يوميًا.
- Instagram أكبر شبكة اجتماعية لمشاركة الصور في الولايات المتحدة. تتم مشاركة أكثر من ٣٠ مليون صورة .
- Facebook يتم تنفيذ أكبر مكتبة أساسية من خلال بايثون .
- Redhat تم تطوير أداة إدارة الحزم yum في أكثر توزيعات Linux شيوعًا في العالم باستخدام بايثون.
- Douban (موقع ويب) يتم تطوير جميع أعمال الشركة تقريبًا من خلال بايثون .
- Zhihu أكبر منتدى للأسئلة والأجوبة في الصين ، تم تطويره بواسطة Python (Quora الأجنبية).
- بالإضافة إلى ما سبق ، فإن شركات مثل Sohu و Jinshan و Tencent و Shanda و Yiwangو Baidu و Alibaba و Taobao و Tudou و Sina و Guok كلها تستخدم Python لإكمال المهام المختلفة.

مميزات بايثون

١. سهلة التعلم: تحتوي لغة Python على عدد قليل نسبيًا من الكلمات الرئيسية وهيكل بسيط وقواعد محددة بوضوح، مما يسهل التعلم.
٢. سهولة القراءة: تعريف كود Python أكثر وضوحًا.
٣. سهولة الصيانة: يكمن نجاح Python في سهولة الحفاظ على شفرة المصدر الخاصة بها.
٤. مكتبة قياسية شاملة: من أكبر مزايا Python مكتبتها الغنية، والنظام الأساسي المشترك ، والمتوافقة مع UNIX و Windows و Macintosh.
٥. الوضع التفاعلي: دعم الوضع التفاعلي، يمكنك إدخال اللغة لتنفيذ التعليمات البرمجية والحصول على النتيجة من الجهاز الطرفي والاختبار التفاعلي وأجزاء رمز التصحيح.
٦. قابلية النقل: بناءً على خصائص المصدر المفتوح ، تم نقل Python (أي لجعلها تعمل) إلى العديد من الأنظمة الأساسية.
٧. قابل للتوسيع: إذا كنت بحاجة إلى رمز مفتاح يعمل بسرعة ، أو إذا كنت تريد كتابة بعض الخوارزميات التي لا تريد فتحها ، فيمكنك استخدام C أو C ++ لإكمال هذا الجزء من البرنامج ، ثم أطلق عليها من برنامج بايثون الخاص بك.
٨. قاعدة البيانات: توفر Python واجهات لجميع قواعد البيانات التجارية الرئيسية
٩. برمجة واجهة المستخدم الرسومية:تدعم Python واجهة المستخدم الرسومية التي يمكن إنشاؤها ونقلها إلى العديد من مكالمات النظام.
١٠. يمكن تضمينها :يمكنك تضمين Python في برنامج C / C ++ ، مما يسمح لمستخدمي برنامجك باكتساب قدرات "البرمجة النصية."

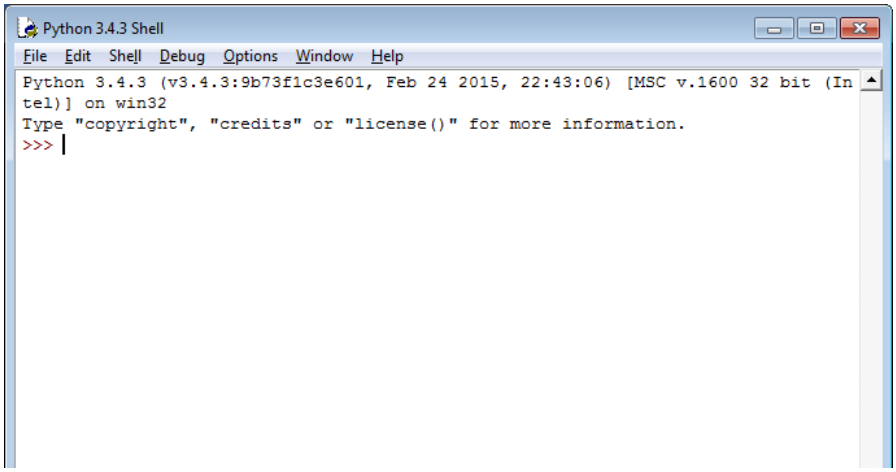
مجالات لغة python

- الذكاء الاصطناعي
- تعليم الآلة
- تحليل البيانات
- Visualization
- التتمة الصناعات البيانات
- desktop applications , Mob ,web
- امن المعلومات وغيرها الكثير

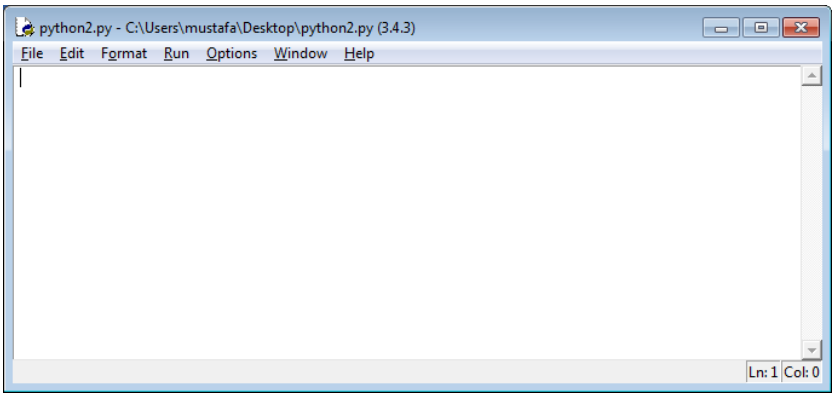
تشغيل برنامج Python

- ✓ نقوم بتنزيل النسخة المناسبة لنظام التشغيل(٣٢ او ٦٤) ونقوم بتنصيبها وهي سهلة التنصيب لا تحتاج أي معلومات مسبقة لإكمال تنصيبها.
- ✓ بعد اكتمال التنصيب نذهب الى قائمة **Start** نكتب في خانة البحث **idle** لتظهر لنا ضمن النتائج ال IDLE Python GUI فنقوم بفتحها لتظهر الواجهة التالية:

وهذه الواجهة تمثل الواجهة الرسومية لتطوير وتطبيق البرامج

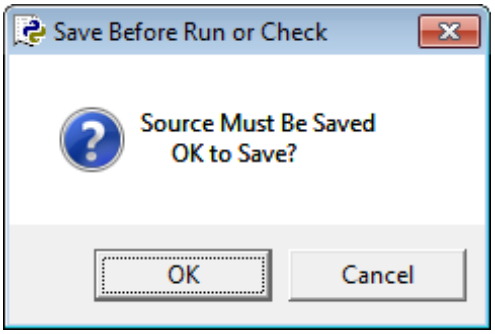


- الان نرى محرك الأوامر) >>> (وهو محرك الأوامر الخاص بلغة البايثون ويعني اننا مستعدون للبدء بكتابة برنامجنا الأول
- والذي سيكون كما في كل لغات البرمجة لطباعة عبارة (hello world) على الشاشة والذي يتكون في لغة بايثون من سطر واحد فقط على خلاف بقية لغات البرمجة الأخرى مثل السي بلس بلس والجافا
- من نافذة ال **shell** نفتح نافذة جديدة لكتابة البرامج كما في الصورة التالية :



- يمكن كتابة المزيد ولكننا هنا نحاول إعطاء مثال فقط والان لتنفيذ كل هذه الايعازات دفعة واحدة نقوم بالذهاب الى قائمة ثم اختيار run او F5 run module لتظهر نتيجة التنفيذ التالية

هنا يطلب منا حفظ الملف (save) حيث تتميز ملفات البايثون بعدم القابلية للتنفيذ بدون حفظ فننقر على(OK) ليتم حفظ تغييرات الملف وتنفيذه فيما بعد لتظهر النتائج التالية



أسلوب كتابة الكود في بايثون

Case Sensitivity تعني أن لغة البرمجة تميز بين الأحرف الكبيرة و الأحرف الصغيرة. بايثون تعامل الأسماء التي نستخدمها بتأني سواء كنا نضع هذه الأسماء للمتغيرات, الدوال, الكلاسات, الكائنات إلخ. مثال note و Note ليسوا شيئاً واحداً

إسم الكلاس: دائماً نبدأ إسم الكلاس بحرف كبير و في حال كان إسم الكلاس يتألف من أكثر من كلمة, نجعل أول حرف من كل كلمة كبيراً.

إسم المتغير: نستخدم الأحرف الصغيرة عند وضع أسماء للمتغيرات و في حال كان إسم المتغير يتألف من أكثر من كلمة نقوم بوضع (_) بين كل كلمتين.

إسم الدالة: نستخدم الأحرف الصغيرة عند وضع أسماء للدوال و في حال كان إسم الدالة يتألف من أكثر من كلمة نقوم بوضع (_) بين كل كلمتين.

التعليقات: نستخدم التعليقات لنضع ملاحظات حول الكود الذي كتبناه فقط لكي لا ننسى كيف برمجنا الكود في حال أردنا مراجعته أو التعديل عليه بعد وقت طويل.

التعليقات لا تؤثر إطلاقاً على الكود المكتوب الموضوع في البرنامج و يمكن وضع عدد غير محدود من التعليقات. لتضع تعليق ضع الرمز # ثم اكتب بعده ما شئت.

ملاحظة : أنت لست مجبراً على وضع تعليقات في برامجك. و لكن ينصح بوضع تعليقات دائماً حتى تساعد في فهم الكود الذي كتبته.

ملاحظة :إذا أردت كتابة أمر واحد على أكثر من سطر قم بوضع الرمز \ في نهاية كل سطر و هكذا سيفهم مترجم لغة بايثون أن الأمر يتألف من أكثر من سطر.

```
# هنا قمنا بتعريف ثلاث متغيرات
item_1 = 10
item_2 = 20
item_3 = 30
# الثلاث أسطر التالية عبارة عن أمر واحد
# إذا هنا سيتم جمع قيم المتغيرات item_1 و item_2 و item_3 و وضع الناتج في المتغير total
total = item_1 + \
      item_2 + \
     item_3
```

أ. قواعد إلزامية عند إعطاء الأسماء

١. الـ **Identifiers** يجب أن يبدأوا بحرف كبير بين A-Z أو حرف صغير بين a-z أو الشحطة.
٢. يمنع إستخدام أي كلمة من الكلمات المحجوزة .(Keywords)
٣. بايثون تطبق مبدأ الـ **Case Sensitive.**

ب. الكلمات المحجوزة في بايثون

جميع الكلمات التالية محجوزة للغة بايثون, أي لا يمكن إستخدامها كـ **Identifiers**.

raise	if	def	False
return	import	del	None
try	in	elif	True
while	is	else	and
with	lambda	except	as
yield	nonlocal	finally	assert
	not	for	break
	or	from	class
	pass	global	continue

مفهوم المتغيرات

المتغيرات (**variables**) عبارة عن أماكن يتم حجزها في الذاكرة بهدف تخزين بيانات فيها أثناء تشغيل البرنامج. في بايثون, المبرمج غير مسؤول عن تحديد أنواع المتغيرات التي يعرّفها في برنامجه. فعلياً, عندما تقوم بتعريف متغير و تضع فيه أي قيمة, سيقوم مفسّر لغة بايثون بتحديد نوع هذا المتغير بناءآ على القيمة التي أسندتها إليه بشكل تلقائي وقت التشغيل. في بايثون يجب إسناد قيمة إلى المتغير أثناء تعريفه.

مثال:

```
var = 5    # هنا قمنا بتعريف متغير إسمه var و قيمته 5
print(var)  # هنا قمنا بطباعة قيمة المتغير var
```

الناتج :

```
5
```

***في بايثون يمكن تعريف عدة متغيرات متساوية في القيمة في وقت واحد.

مثال:

```
x = y = z = 10 # 10 هنا قمنا بتعريف ثلاث متغيرات قيمتها

print('x = ', x) # x هنا قمنا بطباعة قيمة المتغير

print('y = ', y) # y هنا قمنا بطباعة قيمة المتغير

print('z = ', z) # z هنا قمنا بطباعة قيمة المتغير
```

الناتج:

```
x = 10

y = 10

z = 10
```

معرفة نوع المتغير

لمعرفة نوع أي متغير يمكنك إستخدام الدالة. Type ()

تذكر: نوع المتغير في بايثون غير ثابت لأنه يتغير بشكل تلقائي على حسب نوع القيمة التي يتم تخزينها فيه .

```
var = 10 # var هنا وضعنا رقم في المتغير

print(type(var)) # هنا طبعنا نوع قيمة var. لاحظ أن نوعها سيكون int لأنها عبارة عن رقم المتغير

var = 'harmash' # var هنا وضعنا نص في المتغير

print(type(var)) # هنا طبعنا نوع قيمة var. لاحظ أن نوعها سيكون str لأنها عبارة عن نص المتغير
```

الناتج :

```
<class 'int'>

<class 'str'>
```

العمليات الرياضية في بايثون

العملية	وصفها
+	الجمع
-	طرح
*	الضرب
/	القسمة
%	باقي القسمة
**	الأس

عمليات المقارنة في بايثون

العملية	وصفها
x>y	x اكبر من y
x<y	x أصغر من y
x==y	x يساوي y
x≠y	x لايساوي y
x ⩾ y	x أكبر من أو يساوي y
x ⩽ y	x أصغر من أو يساوي y

العمليات المنطقية في بايثون

العملية	الوصف
x and y	x و y، وتكون صحيحة إذا كان كلا المعاملين صحيح
x or y	x أو y، وتكون صحيحة إذا كان أحد المعاملين صحيح
not x	ليست x، وتكون صحيحة إذا كان المعامل غير صحيح

عمليات التعريف(is , is not) الناتج true or false

عمليات الإنتماء (in , not in) الناتج true or false

أنواع البيانات في بايثون

النوع	تعريفها	مثال
List	سلسلة مرتبة من العناصر	a=[1,2,'Aziz']
Tuple	سلسلة مرتبة من العناصر وهي غير قابلة للتغير	b=(1,2,'Aziz')
Strings	سلسلة من الأحرف او الكلمات	c="Aziz blog"
Set	سلسلة غير مرتبة من العناصر الغير المكررة	d={1,2,'Aziz'}

أنواع المتغيرات في بايثون

تنقسم أنواع المتغيرات في بايثون إلى **7** أنواع أساسية و هي:

- أرقام .(**Numbers**)
- نصوص .(**Strings**)
- منطقية .(**Booleans**)
- مصفوفات ليس لها حجم ثابت يقال لها **Lists**.
- مصفوفات حجمها و قيمها ثابتة, و غير قابلة للتغيير يقال لها **Tuples**.
- مصفوفات ليس لها حجم ثابت, و لا يمكن حذف قيمها, و يمكن إضافة قيم جديدة فيها يقال لها **Sets**.
- جداول تخزن البيانات فيها بطريقة مفايتح (**Keys**)و قيم (**Values**)يقال لها **Dictionaries**.

أ. الأرقام

عند تعريف متغير و تخزين رقم فيه, فإن مفسّر لغة بايثون سيقوم بشكل تلقائي بتحديد نوع هذا المتغير بناءآ على نوع القيمة الرقمية التي تم إسنادها إليه. فإذا وضعت فيه عدد صحيح, يصبح نوعه Int. و إذا وضعت فيه عدد عشري (أي يحتوي على فاصلة) يصبح نوعه float .

أنواع الأرقام في بايثون تنقسم إلى **3**أنواع كما في الجدول التالي.

النوع	إستخدامه	مثال
`int`	يستخدم لتخزين أعداد صحيحة.	`x = 3`
`float`	يستخدم لتخزين أعداد تحتوي على فاصلة عشرية.	`x = 1.5`
`complex`	يستخدم لتخزين أعداد مركبة (**Complex Number**) و التي غالباً ما يحتاجها المهندسون عند إجراء عمليات حاسبية معقدة. **ملاحظة:** هنا يجب وضع الحرف `J` أو `j` مباشرةً بعد العدد حتى يعرف مفسّر بايثون أنك تقصد عدد مركب و ليس عدد عادي.	`x = 4J`

ب. النصوص

لتعريف نص في بايثون نستخدم الرمز (') او الرمز (") اوالرمز (" ")

هل يوجد فرق بين هذه الرموز؟

بالنسبة للرمز (') والرمز (") فإنه لا يوجد أي فرق بينهما. ويمكن استخدام أي واحد منهما لتعريف نص يتكون من سطر واحد.

بالنسبة للرمز ''' و الرمز """ فإنه لا يوجد أي فرق بينهما. ويمكن استخدام أي واحد منهما لتعريف نص كبير يتألف من عدة أسطر.

في المثال التالي قمنا بتعريف ثلاث متغيرات تحتوي على قيم نصية. لاحظ أننا قمنا بتعريف كل متغير بواسطة رمز مختلف.

```
name = 'Mhamad'
job = "Programmer"
message = '''This string that will span across multiple lines. No need to use newline characters for the next lines.
The end of lines within this string is counted as a newline when printed.'''
# هنا قمنا بعرض قيم المتغيرات النصية بأسلوب مرتب
print('Name: ', name)
print('Job: ', job)
print('Message: ', message)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Name: Mhamad

Job: Programmer

Message: This string will span across multiple lines. No need to use newline
characters for the next lines.

The end of lines within this string is counted as a newline when printed.
```

ت. القيم المنطقية النوع bool

يستخدم في العادة عند وضع شروط منطقية أو لمعرفة ما إذا تم تنفيذ أمر معين بنجاح أم لا. عند إسناد القيمة True أو القيمة False إلى المتغير فإنه يصبح من النوع bool.

في الواقع القيمة True تساوي 1و القيمة False تساوي 0.

في بايثون يفضل إستخدام الصفر و الواحد بدلاً من إستخدام القيم المحجوزة True و False.

عند الفحص على قيمة المتغير أو على ما سترجعه الدالة.

في المثال التالي قمنا بتعريف متغير إسمه Check و أعطيناه القيمة True ثم إستخدمناه في وضع شرط.

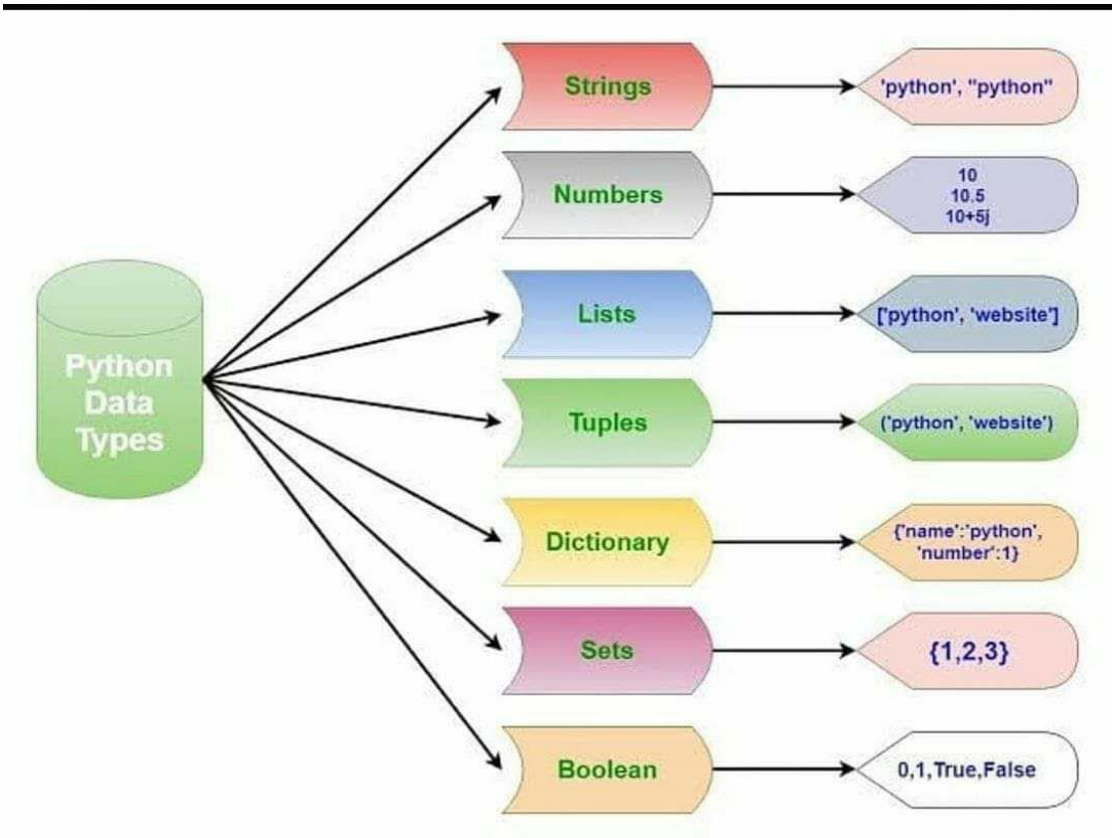
مثال

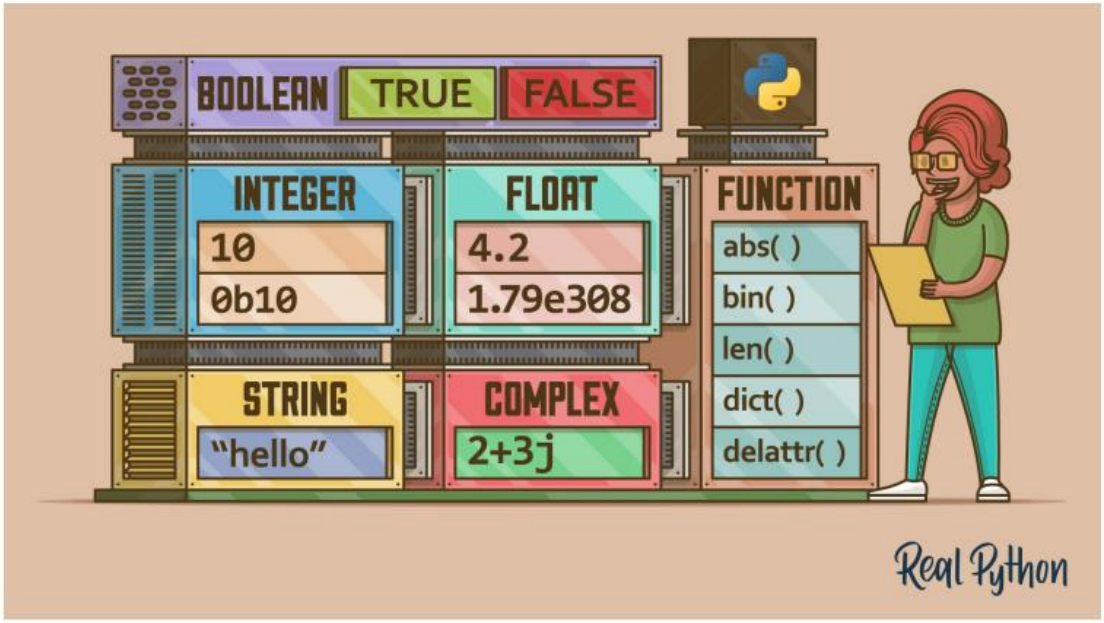
```
# هنا قمنا بتعريف إسمه check و قيمته True

check = True

# سيتم تنفيذ أمر الطباعة الموضوع هنا إذا كانت قيمة المتغير check تساوي True

if check == True:

    print('check = True')

# سيتم تنفيذ أمر الطباعة الموضوع هنا إذا لم تكن قيمة المتغير check تساوي True أي إذا كانت تساوي False

else:

    print('check = False')
```

سنحصل على النتيجة التالية عند التشغيل.

```
check = True
```

ملاحظة بدل الأمر :if check == True كان يمكنك كتابة :if check == 1 أو كتابة :if check فقط و الحصول على نفس النتيجة.


Python Data Types
Strings
'python', "python"
Numbers
10
10.5
10+5j
Lists
['python', 'website']
Tuples
('python', 'website')
Dictionary
{'name':'python', 'number':1}
Sets
{1,2,3}
Boolean
0,1,True,False


BOOLEAN
TRUE
FALSE
INTEGER
10
0b10
FLOAT
4.2
1.79e308
FUNCTION
abs( )
bin( )
len( )
dict( )
delattr( )
STRING
"hello"
COMPLEX
2+3j
Real Python

تخزين البيانات في List

الـ **List** عبارة عن مصفوفة حجمها غير ثابت و يمكنها تخزين قيم من مختلف الأنواع في وقت واحد. في بايثون نستخدم الرمز [ ] لتعريف مصفوفة أحادية (أي ذات بعد واحد) ليس لها حجم ثابت. في المثال التالي قمنا بتعريف **4** مصفوفات.

```
A = [] # هنا قمنا بتعريف مصفوفة فارغة
B = [10, 20, 30, 40, 50] # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة فقط
C = ['Mhamad', 'Samer', 'Abdullah'] # هنا قمنا بتعريف مصفوفة تحتوي على نصوص فقط
D = [1, 'two', 'three', 4] # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة و نصوص
```

في المثال التالي قمنا بتعريف مصفوفة تتألف من **4** عناصر, ثم قمنا بإعطائها **4** قيم, ثم قمنا بإضافة عنصر جديد عليها, ثم عرضنا قيمها و عدد عناصرها.

```
#هنا قمنا بتعريف مصفوفة من النصوص تتألف من ٤ عناصر
languages = [str] * 4
# هنا قمنا بوضع قيمة في كل عنصر فيها
languages[0] = 'Arabic'
languages[1] = 'French'
languages[2] = 'English'
languages[3] = 'Spanish'
# هنا قمنا بإضافة عنصر جديد على المصفوفة
languages.append('German')
# هنا قمنا بعرض قيم المصفوفة و عدد عناصرها
print('Stored languages:', languages)
print('Number of stored languages is:', len(languages))
```

سنحصل على النتيجة التالية عند التشغيل.

```
Stored languages: ['Arabic', 'French', 'English', 'Spanish', 'German']
Number of stored languages is: 5
```

تخزين البيانات في Tuple

الـ **Tuple** عبارة عن مصفوفة حجمها ثابت و قيمها غير قابلة للتغيير و يمكنها تخزين قيم من مختلف الأنواع في وقت واحد. في بايثون نستخدم الرمز ( ) لتعريف مصفوفة أحادية (أي ذات بعد واحد.)

في المثال التالي قمنا بتعريف **4** مصفوفات.

```
A = () # هنا قمنا بتعريف مصفوفة فارغة
B = (10, 20, 30, 40, 50) # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة فقط
C = ('Mhamad', 'Samer', 'Abdullah') # هنا قمنا بتعريف مصفوفة تحتوي على نصوص فقط
D = (1, 'two', 'three', 4) # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة و نصوص
```

في المثال التالي قمنا بتعريف مصفوفة تتألف من **4** عناصر ثابتة, ثم عرضنا قيمها و عدد عناصرها.

```
print(' A is :' ,A)
print(' B is:',B )
print('length C is: ' ,len(C))
```

سنحصل على النتيجة التالية عند التشغيل.

```
A is : 0
B is : 10 20 30 40 50
Length C is : 3
```

تخزين البيانات في Set

الـ **Set** عبارة عن مصفوفة ليس لها حجم ثابت و قيمها غير قابلة للتغيير و يمكنها تخزين قيم من مختلف الأنواع في وقت واحد. في بايثون نستخدم الرمز { } لتعريف مصفوفة أحادية (أي ذات بعد.

في مصفوفات الـ **Set** يتم تخزين البيانات بشكل عشوائي و ليس بالترتيب كما تم إدخالهم, و السبب في أنه في هذا النوع من المصفوفات لا يتم إعطاء رقم **Index** خاص لكل عنصر. لهذا السبب أيضاً, لا يمكنك الوصول لعنصر محدد في **Set** بشكل مباشر لأنه في الأساس لا يملك رقم **Index**.

في المثال التالي قمنا بتعريف **4** مصفوفات.

```
A = {} # هنا قمنا بتعريف مصفوفة فارغة
B = {10, 20, 30, 40, 50} # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة فقط
C = {'Mhamad', 'Samer', 'Abdullah'} # هنا قمنا بتعريف مصفوفة تحتوي على نصوص فقط
D = {1, 'two', 'three', 4} # هنا قمنا بتعريف مصفوفة تحتوي على أعداد صحيحة و نصوص
```

في المثال التالي قمنا بتعريف مصفوفة تتألف من **4** عناصر ثابتة, ثم عرضنا قيمها و عدد عناصرها.

```
# هنا قمنا بتعريف مصفوفة ليس لها نوع محدد و تتألف من ٤ عناصر
languages = {'Arabic', 'French', 'English', 'Spanish'}
# هنا قمنا بعرض قيم المصفوفة و عدد عناصرها
print('Stored languages:', languages)
print('Number of stored languages is:', len(languages))
```

سنحصل على النتيجة التالية عند التشغيل.

```
Stored languages: {'Arabic', 'English', 'French', 'Spanish'}
Number of stored languages is: 4
```

تخزين البيانات في Dictionary

عند إستخدام الـ **List** أو **Tuple** فإنك تتعامل مع عناصرهم من خلال أرقام الـ **Indices**. فكرة الـ **Dictionary** هي وضع مفتاح لكل قيمة. عندها تصل لقيمة كل عنصر موجود من خلال المفتاح الخاص فيه.

إذاً الـ **Dictionary** عبارة جدول تخزن فيه البيانات بطريقة مفاتيح (**Keys**) و قيم (**Values**). بالنسبة لنوع البيانات التي تخزنها بداخل الـ **Dictionary** فعندك الحرية في تخزين مفاتيح و قيم من أي نوع تريد. في بايثون نستخدم الرمز { } لتعريف **Dictionary**.

في المثال التالي قمنا بتعريف **Dictionary** يتألف من **5** عناصر, ثم عرضنا قيمة العنصر الثالث من خلال المفتاح الخاص فيه.

```
# هنا قمنا بتعريف dictionary يتألف من ٥ عناصر
dictionary = {
1: 'One',
2: 'Tow',
3: 'Three',
4: 'Four',
5: 'Five'
}
# هنا قمنا بعرض قيمة العنصر الذي يحمل المفتاح رقم ٣
print(dictionary[3])
```

سنحصل على النتيجة التالية عند التشغيل.

```
Three
```

مفهوم العوامل

العوامل (operators) عبارة عن رموز لها معنى محدد و يمكننا تقسيمها إلى 7 مجموعات أساسية هي:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Membership Operators
- Identity Operators

العوامل التي تستخدم في العمليات الحسابية (Arithmetic Operators)

إسم العامل	رمزه	مثال	شرح الكود
Assignment	=	a = b	أعطي a قيمة b
Addition	+	a + b	أضف قيمة b على قيمة a
Subtraction	-	a - b	إطرح قيمة b من قيمة a
Unary plus	+	+a	أضرب قيمة a بالعامل +
Unary minus	-	-a	أضرب قيمة a بالعامل -
Multiplication	*	a * b	أضرب قيمة a بقيمة b
Exponent	**	a ** b	ضاعف قيمة a بقيمة b
Division	/	a / b	أقسم قيمة a على قيمة b
Floor Divide	//	a // b	أقسم قيمة a على قيمة b و أرجع أقرب عدد صحيح للناتج لا يحتوي على فاصلة.
Modulo	%	a % b	للحصول على آخر رقم يبقى عندما نقسم قيمة a على قيمة b

العوامل التي تستخدم في المقارنات(Comparison Operators)

إسم العامل	رمزه	مثال	شرح الكود
Equal to	==	(a == b)	هل قيمة a تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True
Not equal to	!=	(a != b)	هل قيمة a لا تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True
Greater than	>	(a > b)	هل قيمة a أكبر من قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True
Less than	<	(a < b)	هل قيمة a أصغر من قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True
Greater than or Equal to	>=	(a >= b)	هل قيمة a أكبر أو تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True
Less than or Equal to	<=	(a <= b)	هل قيمة a أصغر أو تساوي قيمة b ؟ إذا كان الجواب نعم فإنها ترجع True

العوامل التي تستخدم في وضع شروط منطقية (Logical Operators)

إسم العامل	رمزه	مثال	شرح الكود
Logical AND	and	a and b	هل قيمة a و b تساويان True ؟ هنا يجب أن يتم تحقيق الشرطين ليرجع True
Logical OR	or	a or b	هل قيمة a أو b أو كلاهما تساويان True ؟ هنا يكفي أن يتم تحقيق شرط واحد من الشرطين ليرجع True
Logical NOT	not	not a	هل قيمة a لا تساوي True ؟ إذا كان الجواب نعم فإنها ترجع True

العوامل التي تستخدم لإعطاء قيم للمتغيرات (Assignment Operators)

إسم العامل	رمزه	مثال	شرح الكود
Basic Assignment	=	a = b	ضع قيمة b في a
Add AND Assignment	+=	a += b	أضف قيمة a على قيمة b و خزن الناتج في a
Susbtract AND Assignment	-=	a -= b	أنقص قيمة a من قيمة b و خزن الناتج في a
Multiply AND Assignment	*=	a *= b	أضرب قيمة a بقيمة b و خزن الناتج في a
Exponent AND Assignment	**=	a **= b	ضاعف قيمة a بقيمة b و خزن الناتج في a
Divide AND Assignment	/=	a /= b	أقسم قيمة a على قيمة b و خزن الناتج في a
Floor Divide AND Assignment	//=	a //= b	أقسم قيمة a على قيمة b و أرجع أقرب عدد صحيح للناتج
Modulo AND Assignment	%=	a %= b	أقسم قيمة a على قيمة b و خزن آخر رقم يبقى من عملية القسمة في a
Left shift AND Assignment	<<=	a <<= 2	أزح آخر إثنين bits و ضعهم في الأول ثم خزن الناتج في a
Right shift AND Assignment	>>=	a >>= 2	أزح أول اثنين bits و ضعهم في الآخر ثم خزن الناتج في a
Bitwise AND Assignment	&=	a &= b	أحسب ناتج جمع الـ bits المشتركة بين a و b و خزن الناتج في a
Bitwise exclusive OR and Assignment	^=	a ^= b	أحسب ناتج جمع الـ bits الغير مشتركة بين a و b و خزن الناتج في a
Bitwise inexclusive OR and Assignment	|=	a |= b	أحسب ناتج جمع الـ bits المشتركة و الغير مشتركة بين a و b و خزن الناتج في a

العوامل التي تستخدم للبحث في المصفوفات (Membership Operators)

إسم العامل	رمزه	مثال	شرح الكود
In	in	a in arr	هل قيمة المتغير a موجودة في المصفوفة arr ؟ إذا كان الجواب نعم فإنها ترجع True
Not In	not in	a not in arr	هل قيمة المتغير a غير موجودة في المصفوفة arr ؟ إذا كان الجواب نعم فإنها ترجع True

ادخال البيانات في بايثون

الدالة input

جعل المستخدم قادر على إدخال بيانات في البرنامج أثناء اشتغاله نستخدم الدالة input() في كل مرة تقوم فيها باستدعاء هذه الدالة يقوم مفسّر لغة بايثون بانتظارك لإدخال ما تريد من لوحة المفاتيح .(Keyboard)
بعد الإنتهاء من الإدخال و النقر على الزر enter سيتم إرجاع الشيء الذي قمت بإدخاله كنص في المكان الذي تم منه إستدعاء الدالة input( ).

عند استدعاء الدالة input ( ) فإنك حتى لو قمت بإدخال رقم فإنها سترجعه كنص. لذلك في حال كنت تريد من المستخدم أن يدخل رقم, سيكون عليك تحويل ما ترجعه الدالة لرقم.

امثلة على الدالة input ( )
مثال ١ برنامج لادخال الاسم وطباعة رسالة ترحيب

```
name = input("What's your name? ")
print("Nice to meet you", name)
```

الناتج

```
What's your name? mhamad

Nice to meet you mhamad
```

مثال ٢
إنشاء برنامج يطلب من المستخدم إدخال عددين صحيحين, ثم يعرضه له ناتج جمعهما. نحتاج الى استخدام دالة int لتحويل القيمة النصية الى قيمة رقمية

```
a = int(input("Enter a: "))
b = int(input("Enter b: "))
print('a + b =', a + b)
```

الناتج

```
Enter a: 5

Enter b: 7

a + b = 12
```

مثال:٣: انشاء برنامج يطلب من المستخدم إدخال عدد يمثل عدد عناصر مصفوفة, ثم يطلب منه إدخال قيمة لكل عنصر في المصفوفة.في الأخير يعرض له البرنامج كل قيمة قام بإدخالها فيها على سطر واحد.

```
n = int(input("Enter the length of the: "))
a= [0] * n
for i in range(len(aList)):
    aList[i] = int(input('Enter aList[' + str(i) + ']: '))
    print('acontain the following values:')
for val in aList:
```

```
    print(val, end=' ')
```

الناتج

.

```
Enter the length of the: 5
Enter aList[0]: 10
Enter aList[1]: 20
Enter aList[2]: 30
Enter aList[3]: 40
Enter aList[4]: 50
The contain the following values:
10 20 30 40 50
```

تعليمة الطباعة print

في لغة البرمجة بايثون، يتم استخدام تعليمة "print" لعرض مخرجات إلى وحدة الإخراج (مثل الشاشة). يمكن استخدامها لعرض نصوص، قيم متغيرات، وأكثر من ذلك. هناك بعض الأشكال الأساسية لاستخدام تعليمة الطباعة "print" في بايثون.

١. طباعة نص ثابت

```
print("Hello, World!")
```

٢. طباعة قيمة متغير

```
name = "Alice"
print("My name is", name)
```

٣. طباعة عدة متغيرات

```
Name="أحمد"
age = 25
print("اسم:", name, "العمر:", age)
```

٤. طباعة اكثر من قيمة مع فصلها بفواصل

```
age = 25
height = 170
print("My age is", age, "and my height is", height)
```

٥. طباعة بتنسيق معين باستخدام تنسيق السلسلة

```
name = "Bob"
age = 30
print("Name: {}, Age: {}".format(name, age))
# أو باستخدام الفهرسة:
print("Name: {0}, Age: {1}".format(name, age))
```

٦. طباعة باستخدام تنسيق f-string

```
name = "Charlie"
age = 22
print(f"My name is {name} and I am {age} years old.")
```

٧. طباعة دون ترتيب محدد

```
name = "David"
age = 28
print(f"My age is {age} and my name is {name=}")
```

٨. طباعة بناء على تنسيق نصي محدد

```
score = 90
print("نتيجتك هي: %d" % score)
```

٩. طباعة متعددة الاسطر باستخدام العلامة n/

```
print("سطر ١\nسطر ٢\nسطر ٣")
```

١٠. التحكم في الفاصل بين العناصر المطبوعة باستخدام sep

```
item1 = "تفاح"
item2 = "بانانا"
item3 = "كمثرى"
print(item1, item2, item3, sep=" - ")
```

١١. طباعة دون السطور التلقائية باستخدام end

```
print("سطر ١", end=" ")
print("سطر ٢", end=" ")
print("سطر ٣")
```

الشروط في بايثون If condition

الشروط (**conditions**) تستخدم لتحديد طريقة عمل البرنامج نسبةً للمتغيرات التي تطرأ على الكود.

كمثال بسيط, يمكنك بناء برنامج لمشاهدة الأفلام, عند الدخول إليه يطلب من المستخدم في البداية أن يدخل عمره لكي يقوم بعرض أقلام تناسب عمره.

يمكنك وضع العدد الذي تريده من الشروط في البرنامج الواحد, و تستطيع وضع الشروط بداخل بعضها البعض أيضاً.

طريقة وضع الشروط

if condition:

إذا كان الشرط صحيحاً نفذ هذا الكود

elif condition:

إذا كان الشرط صحيحاً نفذ هذا الكود

else:

إذا لم يتحقق أي شرط نفذ هذا الكود

إسم الجملة	دواعي الإستخدام
IF Statement	`if` في اللغة العربية تعني "إذا". و هي تستخدم فقط في حال كنت تريد تنفيذ كود معين حسب شرط معين.
Else Statement	`else` في اللغة العربية تعني "أي شيء آخر". و هي تستخدم فقط في حال كنا نريد تنفيذ كود معين في حال كانت نتيجة جميع الشروط التي قبلها تساوي `false` . يجب وضعها دائماً في الأخير, لأنها تستخدم في حال لم يتم تنفيذ أي جملة شرطية قبلها.
Else IF Statement	`elif` تستخدم إذا كنت تريد وضع أكثر من إحتمال (أي أكثر من شرط). جملة أو جمل الـ `elif` يوضعون في الوسط, أي بين الجملتين `if` و `else` .

امثلة على جمل الشرط

ادخال عدد والتأكد اذا كانت قيمته 5 يطبع الرقم يساوي ٥ والا يطبع الرقم لا يساوي ٥

```
S = int(input("Enter the number: "))
if S == 5:
   print('S is equal 5')
else:
   print('S is not equal 5')
```

سنحصل على الناتج التالي

```
S is equal 5
```

مثال: ادخال عدد والتأكد اذا كانت قيمته ٥ يطبع الرقم يساوي ٥ والا يطبع الرقم لا يساوي ٥

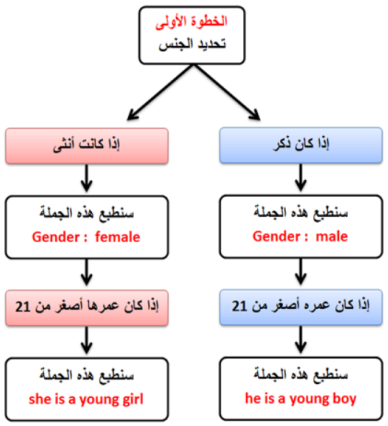
```
number = int(input("Enter the number: "))
if number == 1 :
        print('one')

elif number == 2 :
        print('two')

elif number == 3 :
        print('three')

elif number >= 4 :
        print('four or greater')

else :
        print('negative number')
```

سنحصل على الناتج

```
three
```

مثال على if المتداخلة

لنفترض نريد تحويل الصورة التالية الى برنامج




```
sex = 'female'
age = 14
if sex == 'male':
    print('Gender: male')
    if age <= 21:
        print('he is a young boy');
elif sex == 'female':
       print('Gender: female')
    if age <= 21:
        print('she is a young girl')
```

الناتج

```
Gender: female
she is a young girl
```

وضع اكثر من شرط داخل if

بإمكانك أن تضع أكثر من شرط بداخل جملة الشرط باستخدام العامل and او or.

العامل and يستخدم في حال كنت تريد تنفيذ كود معين إذا كان جواب جميع الشروط الموضوعة يساوي true

العامل or يستخدم في حال كنت تريد تنفيذ كود معين إذا كان جواب شرط واحد على الأقل يساوي true.

مثال : اذا كانت قيمة المتغير a بين ٠ و ٢٠ اطبع الجملة acceptable number

```
a = 14;

if a >= 0 and a <= 20:

   print("acceptable number")
```

الناتج :

acceptable number

مثال :

```
myNumber = 5
if myNumber > 2:
  print("myNumber Greater Than 2")
  if myNumber < 5:
    print("myNumber Less Than 5")
else:
  print("myNumber Less Than 2")
```

الحلقات في بايثون

مفهوم الحلقات في بايثون

نستخدم الحلقات (**Loops**) بهدف تكرار نفس الكود عدة مرات.

طريقة تنفيذ الأوامر و الحلقات في الذاكرة

الأوامر في العادة تتنفذ بتسلسل وراء بعضها, و لكن الحلقات تجعل سهم تنفيذ الأوامر يقف عندها فيقوم بتنفيذ الأوامر التي بداخلها عدة مرات, و بعد أن يخرج من الحلقة يعود و يكمل تنفيذ باقي الأوامر التي تليها, هناك نوعين من حلقات التكرار في بايثون

١. دوارة while : تستخدم للتكرار عبر كتلة من التعليمات البرمجية طالما ان تعبير الاختيار (الشرط) صحيح, نستخدم هذه الحلقة بشكل عام عندما لا نعرف عدد مرات تكرارها سابقاً

الصيغة العامة :

```
while condition :
    block
```

تبدأ التعليمة بكلمة while ثم الشرط وبعدها النقطتان (:)

Condition :يحدد الشرط (condition) ما اذا كان سيتم تنفيذ الكتلة ام لا

Block: عبارة عن تعليمة او مجموعة من التعليمات يتم تنفيذها طالما الشرط صحيح

٢. دوارة for: تستخدم دوارة for بطريقتين

أ. للتكرار عبر كتلة من التعليمات البرمجية باستخدام دالة النطاق

```
for val in range (begin , end , step ):
    Body of for
```

Val: متغير يأخذ قيمة العناصر داخل مدى الدالة لكل دورة

Begin: اول قيمة في المدى اذا محذوف تأخذ 0 كقيمة افتراضية

End:القيمة الاخيره في النطاق بمقدار 1 القيمة النهائية مطلوبة دائما ولا يجوز حذفها

Step: مقدار الزيادة او النقصان اذا كانت القيمة محذوفة تكون قيمة الزيادة افتراضية بمقدار 1.

ا. للتكرار عبر سلسلة (list,tupe,string, ect) تكرار السلسلة يسمى الاجتياز

```
for val in sequence
body of for
```

Val: متغير يأخذ العناصر داخل المتسلسلة لكل تكرار

الحلقات الموجودة في بايثون

إسم الحلقة	دواعي الإستخدام
For Loop	تستخدم الحلقة `for` للمرور على جميع عناصر السلسلة أو المصفوفة بسهولة بدون الحاجة لتعريف عداد و تحديد أين يبدأ و أين ينتهي. و تستخدم لتنفيذ الكود عدة مرات محددة.
While Loop	تستخدم الحلقة `while` لتنفيذ الكود عدة مرات غير محددة و يتوقف التنفيذ إذا تحقق شرط معين. لأن هذه الحلقة يتم توقيفها إذا تحقق الشرط الذي نضعه بين القوسين.

جمل التحكم في الحلقات في بايثون

جمل التحكم تعني **Control Statements** بالإنجليزية, و نستخدمهم للتحكم في سير تنفيذ الحلقات.

جملة التحكم	تعريفها
Break Statement	الجملة `break` تستخدم بشكل عام لإيقاف الحلقة في حال تحقق شرط معين. ثم تنتقل للكود الذي يليها في البرنامج.
Continue Statement	الجملة `continue` تستخدم بشكل عام لإيقاف الدورة الحالية في الحلقة و الإنتقال إلى الدورة التالية فيها في حال تحقق شرط معين.

مثال: اكتب برنامج بلغة بايثون باستخدام حلقة while لطباعة جميع الأرقام من 1إلى 10واستخدام الجملة break لجعل الحلقة تتوقف عندما تصبح قيمة العداد counter تساوي 5.

```
counter = 1

while counter <= 10:

    print(counter)

    if counter == 5:

        break

    counter += 1

print('The loop was stopped when counter =', counter)
```

الناتج

```
1
2
3
4
5
The loop was stopped when counter = 5
```

نفس المثال تم تطبيقه على for

```
for n in range(1,11):

    print(n)

    if n == 5:

        Break

print('The loop was stopped when n =', n)
```

الناتج

```
1
2
3
4
5
The loop was stopped when n = 5
```

مثال اكتب برنامج بلغة بايثون باستخدام حلقة for لطباعة جميع الأرقام من ١ الى ١٠ باستخدام continues لجعل الحلقة تتجاوز كل دورة تكون فيها قيمة المتغير n عبارة عن عدد مزدوج.

```
for n in range(1,11)
   if n % 2 == 0:
      continue
   print(n)
```

الناتج

```
1
3
5
7
9
```

تمارين توضح الفرق بين for و while

Example (1): Write Python Program to print numbers between 0 and 99.

while	for
i = 0 while i < 100: print(i) i = i+1	for i in range (100): print(i)

Example (2): Write Python Program to print odd numbers between 0 and 99.

while	for
i = 1 while i <= 99: print(i) i = i+2	for i in range (1,100,2): print(i)

Example (3): Write Python Program to print even numbers between 0 and 99.

while	for
i = 0 while i <= 99: print(i) i = i+2	for i in range (0,100,2): print(i)

Example (4): Write Python Program to read n numbers and print the largest number of them.

while	for

n = int(input("Enter n: "))
x = int(input("Enter x: "))
m= x
i = 1
while i < n:
  x = int(input("Enter x: "))
  if x > m:
  m=x
  i = i+1
print("The max is", m)
	n = int(input("Enter n: "))
	x = int(input("Enter x: "))
	m= x
	for i in range (1, n):
	  x = int(input("Enter x: "))
	  if x > m:
	  m=x
	print("The max is", m)

الحلقات الدورانية المتداخلة

هي عبارة عن دوارة داخل دوارة أخرى. الدوارة الداخلية تنفذ مرة واحده لكل تكرار في الدوارة الخارجية

Example (1): Write Python Program to find following series:

$$\sum_{i=1}^{5}\sum_{j=1}^{8} i+j$$

while	for

i,s=1,0
while i<=5:
  j=1
  while j<=8:
  s=s+ i+j
  j=j+1
  i=i+1
print("sum is",s)
	s=0
	for i in range(1,6):
	  for j in range(1,9):
	  s=s+ i+j
	  print("sum is",s)

Example (2): Write Python Program to find following series:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{i}{j}$$

While	for

n = int(input("enter n "))
m = int(input("enter m "))
 	
i,s=1,1
while i<=n:
  j=1
  while j<=m:
  s=s+ (i/j)
  j=j+1
  i=i+1
print("sum is",s)
	n = int(input("enter n "))
	m = int(input("enter m "))
	s=1
	for i in range(1,n+1):
	  for j in range(1,m+1):
	  s=s+ (i/j)
	  print("sum is",s)

Example (3): Write Python Program to find following pattern:

```
* * *
* * *
* * *
```

```
for i in range(1,4):
    for j in range(1,4):
        print("*",end="")
    print()
```

Example (4): Write Python Program to find following pattern:

```
111
222
333
```

```
for i in range(1,4):
    for j in range(1,4):
        print(i,end="")
    print()
```

Example (5): Write Python Program to find following pattern:

```
123
123
123
```

```
for i in range(1,4):
    for j in range(1,4):
        print(j,end="")
    print()
```

Example: Python break statement inside loop	
while	for

i=1
while i<5:
  if i == 3:
  break
  print(i)
  i=i+1
	for i in range(1,5):
	  if i == 3:
	  break
	  print(i)

Example: Python break statement inside nested loop	
while	for

i=1
while i<5:
  if i == 3:
  break
  j=1
  while j<5:
  print(i, end="")
  j=j+1
  print()
  i=i+1
	for i in range(1,5):
	  if i == 3:
	  break
	  for j in range(1,5):
	  print(i, end="")
	  print()

output	
1111 2222	

واجب :

١. اكتب برنامج بلغة بايثون لايجاد مفكوك العدد X
٢. اكتب برنامج بلغة بايثون لطباعة الاعداد من 5000 الى 1 ثم طباعة مجموعها
٣. اكتب برنامج بلغة بايثون لطباعة مربع الاعداد من 1 الى 10
٤. اكتب برنامج لقراءة عدد والتحقق اذا كان العدد اولي او غير اولي
٥. اكتب برنامج بلغة بايثون لقراءة n من الاعداد وطباعة كم عدد فردي وزوجي في المدخلات
٦. اكتب برنامج لايجاد مجموع الاعداد 3,9,27,81,243
٧. اكتب برنامج بلغة بايثون لايجاد مخرج المدى

range(-5,5) , range(1,-1,1) , range(2,11,2)

٨. اكتب برنامج بلغة بايثون لطباعة النمط

4444
3333
2222
1111

٩. اكتب برنامج بلغة بايثون لطباعة النمط

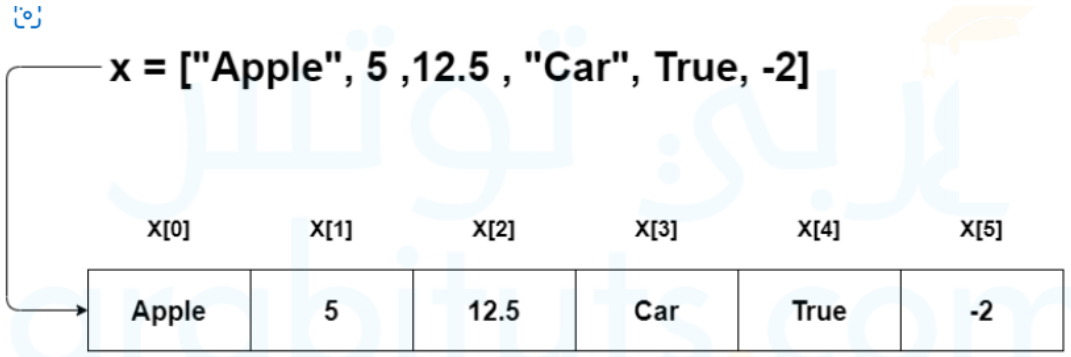
1
2 3
4 5 6
7 8 9 10

١٠. اكتب مخرجات الكود التالي بعد تنفيذه بلغة بايثون

```
i = 1
while i<=6:
   if i%3 == 0:
      break
   print(i)
   i=i+1
```

القوائم lists

هو اكثر أنواع البيانات استخداما وهو يستخدم لحفظ مجموعة من العناصر تحت مسمى واحد ونستطيع الوصول للعناصر عن طريق استخدام تسلسها ترتيبها index في القائمة . في بايثون كما العديد من لغات البرمجة يبدأ التسلسل من الرقم صفر الى ما لا نهاية والصورة التالية توضح ذلك



التعريف

نقوم بتعريف القوائم بوضع العناصر بين [ ] ويفصل بينهم بالفاصلة " ,"

ممكن ان تحتوي القائمة على اعداد صحيحة او نصوص او كلاهما

```
numbers = [10, 20, 30, 40, 50]

names = ['Rami', 'Sara', 'Nada']

data = [1, 'Mhamad', 'Harmush', 1500]
```

الوصول لعناصر الـ List

للوصول لأي عنصر في الـ list سواء للحصول على قيمته أو تغييرها فإننا نستخدم رقم الـ **Index** الخاص بالعنصر .في بايثون, يمكنك الوصول للعناصر الموجودة في الـ list بطريقتين:

- في حال أردت الوصول لعناصر الـ list من اليسار إلى اليمين, أي من أول عنصر تم إدخاله, فإن الـ **Index** لأول عنصر سيكون .**0**
- في حال أردت الوصول لعناصر الـ list من اليمين إلى اليسار, أي من آخر عنصر تم إدخاله, فإن الـ **Index** لآخر عنصر سيكون .**-1**

مثال:

```
# وضعنا فيه قيم نصية تمثل أسماء أشخاص names إسمه هنا قمنا بتعريف
names = ['Rami', 'Sara', 'Nada', 'Mhamad', 'Salem']
print(names[0])    # namesهنا قمنا بعرض قيمة أول عنصر موجود في الكائن
print(names[1])    # namesهنا قمنا بعرض قيمة ثاني عنصر موجود في الكائن
print(names[-1])    # names هنا قمنا بعرض قيمة آخر عنصر موجود في الكائن

print(names[-2])    # names هنا قمنا بعرض قيمة ما قبل آخر عنصر موجود في الكائن
```

الناتج

```
Rami
Sara

Salem
Mhamad
```

مثال

```
# هنا قمنا بتعريف كائن إسمه names وضعنا فيه قيم نصية تمثل أسماء أشخاص

names = ['Rami', 'Sara', 'Nada', 'Mhamad', 'Salem']

# و من ثم سيتم طباعتها x في المتغير names في كل مرة سيتم وضع قيمة عنصر من عناصر
الكائن

for x in names:

   print(x)
```

الناتج

```
Rami
Sara
Nada
Mhamad
Salem
```

مثال

```
# هنا قمنا بتعريف إسمه numbers وضعنا فيه أعداد صحيحة

Numbers=[10,20,30,40,50]

 #numbersهنا قمنا بتعريف متغير سنستخدمه لحفظ ناتج جمع القيم الموجودة في الكائن

total = 0

# في كل مرة سيتم وضع قيمة عنصر x من عناصر الكائنnumbers  و من ثم سيتم إضافتها على قيمة المتغير في المتغير total

for x in numbers:

    total += x

# هنا قمنا بطباعة قيمة المتغير total  و التي ستساوي ناتج جمع القيم الموجودة في الكائن numbers

    print('Total sum is:', total)
```

الناتج

```
Total sum is: 150
```

تعديل العناصر

نستطيع تعديل العناصر باستخدام التسلسل

مثال:

```
X=[“apple”,5,12.5,”car”,True,-2]

X[0]=10

X[1]=”orange”

print(X)
```

الناتج

```
[10, orange ,12.5,”car”,True,-2]
```

الدوال /الطرق method

يوفر list مجموعة من الطرق التي نستطيع استخدامها

Method	Description
append()	Add an element to the end of the list
extend()	Add all elements of a list to the another list
insert()	Insert an item at the defined index
remove()	Removes an item from the list
pop()	Removes and returns an element at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of the number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse()	- Reverse the order of items in the list

مثال : شامل لجميع دوال list

```
m=[3,8,1,6,0,8,4]

print('m= ',m)

print(m.index(8))#يعيد موقع العنصر ٨

print(m.count(8))#يعيد عدد مرات تكرار الرقم ٨ في الكائن

m.sort()#يرتب الكائن تصاعدياً

print('m sort',m)

m.reverse()#يعكس طباعة الكائن من اليمين الى اليسار

print('m reverse ',m)

m.append(11)#يضيف العنصر ١١ الى نهاية الكائن
```

```
print('m append ',m)

m.remove(3)#يحذف القيمة ٣ من الكائن

print('m remove ',m)

m.insert(4,'zahraa')#اضافة عنصر في الموقع ٤

m.insert(len(m),'husssein')

print('m insert',m)

m.pop(4)#يحذف العنصر في الموقع ٤

print('m pop',m)

del m[-1]

print(m)

n=['a','b','c','d']

m.extend(n)

print('m extend n',m)

m.clear()#يحذف كل عناصر الكائن

print('m=',m)
```

الناتج

```
m=  [3, 8, 1, 6, 0, 8, 4]

1

2

m sort [0, 1, 3, 4, 6, 8, 8]

m reverse  [8, 8, 6, 4, 3, 1, 0]

m append  [8, 8, 6, 4, 3, 1, 0, 11]
```

m remove [8, 8, 6, 4, 1, 0, 11]

m insert [8, 8, 6, 4, 'zahraa', 1, 0, 11, 'husssein']

m pop [8, 8, 6, 4, 1, 0, 11, 'husssein']

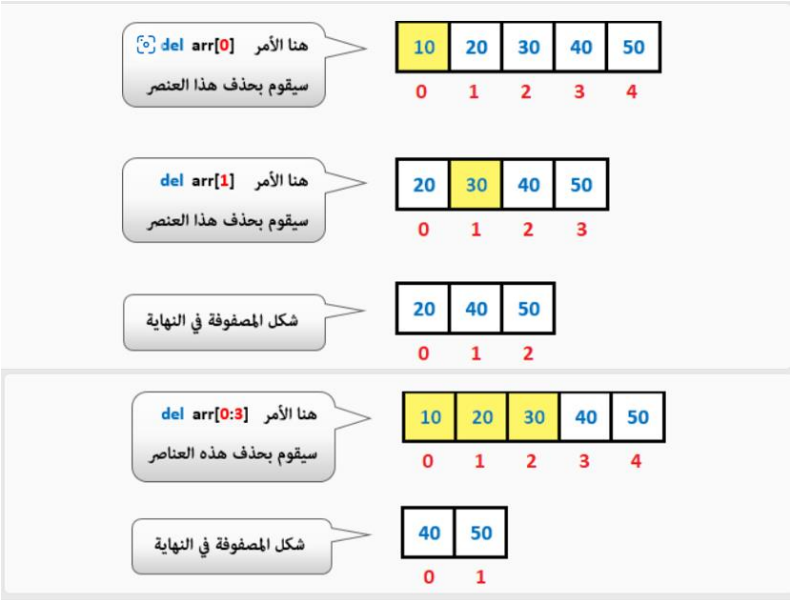
[8, 8, 6, 4, 1, 0, 11]

m extend n [8, 8, 6, 4, 1, 0, 11, 'a', 'b', 'c', 'd']

m= []

دالة **Del** تستخدم لحذف المصفوفة كما هي من الذاكرة او لحذف عناصر محدده منها . عند حذف أي عنصر فأن مترجم بايثون يقوم بإعادة ترتيب عناصره من جديد وتحديث رقم ال index الخاص بكل عنصر

الصور التالية توضح كيف يتم الغاء العنصر



مثال

وضعنا فيه أعداد صحيحة arr هنا قمنا بتعريف إسمه

arr=[10,20,30,40,50]

del arr[0] # هنا قمنا بحذف أول عنصر موجود في الكائنarr

del arr[1] # هنا قمنا بحذف ثاني عنصر موجود في الكائنarr

print(arr) # هنا قمنا بعرض ما يحتويه الكائنarr

del arr[0:3] # هنا قمنا بحذف أول ثلاث عناصر موجودة في الكائنarr

print(arr) # هنا قمنا بعرض ما يحتويه الكائنarr

arr2= arr[0: 3]

print(arr2) ارجاع جزء من الكائن

الناتج

[20, 40, 50]

[40, 50]

[10, 20, 30]

استخدام العوامل + و * و in مع ال list

العامل	دواعي الإستخدام
+	العامل + يستخدم لدمج list أو أكثر في list واحد.
*	العامل * يستخدم لتكرار قيمة معينة أو نوع بيانات معين ليمثل نوع القيم التي يمكن وضعها في عناصر الـ list كما فعلنا في بعض الأمثلة السابقة.
in	العامل in يستخدم للبحث في الـ list عن قيمة معينة أو للمرور على قيمه عند إستخدامه في الحلقة for كما فعلنا في بعض الأمثلة السابقة.

مثال

```
arr1 = [1, 2, 3]     # إسمه هنا قمنا بتعريف arr1 وضعنا فيه أعداد صحيحة
arr2 = [4, 5, 6]     # إسمه هنا قمنا بتعريف arr2 وضعنا فيه أعداد صحيحة
arr3 = arr1 + arr2   # إسمه هنا قمنا بتعريف arr3 وضعنا فيه قيم الكائنين arr1 و arr2
print(arr3)          # هنا قمنا بعرض ما يحتويه الكائ arr3
arr2 = ['python'] * 3     # إسمه هنا قمنا بتعريف arr وضعنا فيه ٣ عناصر قيمتهم النص 'python'
print(arr2)          # هنا قمنا بعرض ما يحتويه الكائن arr
```

الناتج

[1, 2, 3, 4, 5, 6]

['python', 'python', 'python']

مثال

```
arr = ['Mhamad', 'Rony', 'Rima', 'Sara']     # إسمه هنا  arr وضعنا فيه مجموعة قيم نصية قمنا بتعريف
x = 'Rima'                                   # هنا قمنا بتعريف متغير إسمه x وضعنا فيه نص
print('Is Rima in the?')
print(x in arr)                              # هل  x في الكائن arr. إذا تم إيجادها سيتم عرض True سيتم البحث عن قيمة
```

الناتج

Is Rima in the?

True

دوال جاهزة في بايثون للتعامل مع list

	إسم الدالة مع تعريفها
1	`len(list)` ترجع عدد صحيح يمثل عدد عناصر الـ `list` الذي نمرره لها عند إستدعاءها.
2	`min(list)` ترجع أصغر قيمة موجودة في الـ `list` الذي نمرره لها عند إستدعاءها.
3	`max(list)` ترجع أكبر قيمة في الـ `list` الذي نمرره لها عند إستدعاءها.
4	`list(sequence)` ترجع نسخة من أي كائن يحتوي مجموعة عناصر نمرره لها عند إستدعاءها ككائن من الكلاس الـ `list` .

مثال

```
alist=[1,2,3,4,5]
x=len(alist)
print('list length is: ',x)
print('minimum value is ',min(alist))
z=max(alist)
print('maximum value is ',z)
copylist=list(alist)
print('copy list ',copylist)
```

سنحصل على النتيجة التالية عند التشغيل.

```
list length is:  5
minimum value is  1
maximum value is  5
copy list  [1, 2, 3, 4, 5]
```

مفهوم الكلاس tuble

عبارة عن مصفوفة لها حجم ثابت, يمكنها تخزين قيم من مختلف الأنواع فـي وقت واحد و لا يمكن تبديل قيمها

طريقة تعريف tuble

نستخدم الرمز ( ) داخل هذا الرمز يمكنك تمرير القيم بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين. و يمكنك تحديد نوع و عدد العناصر التي تريد وضعها فيه فقط. يمكن ان تكون فارغة او تحتوي على ارقام فقط , نصوص فقط و ارقام ونصوص معاً.

```
A = (10,)
```

وضع الفاصلة بعد الرقم ١٠ ضروري للتميز بين متغير عادي او tuble, كما ان لا حاجة لوضع فارزة إضافية اذا كان اكثر من عنصر.

```
numbers = (10, 20, 30, 40, 50)

names = ('Rami', 'Sara', 'Nada')

data = (1, 'Mhamad', 'Harmush', 1500)
```

الوصول للعناصر

للوصول لاي عنصر بهدف الحصول على قيمته طبعاً فإننا نستخدم رقم الـ Indexالخاص بالعنصر.

يمكن الوصول بطريقتين

- في حال اردت الوصول من اليمين الى اليسار أي من أول عنصر تم إدخاله, فإن الـ Indexالأول عنصر سيكون 0.
- في حال اردت الوصول من اليسار الى اليمين أي من آخر عنصر تم إدخاله, فإن الـ Indexالآخر عنصر سيكون -1.

فائدة النوع tuple

- الوصول للعناصر اسرع من list
- قيمه تعتبر امنة حيث يمكن قراءتها فقط.
- في حال تستخدم النوع dictionary والذي يتم وضع البيانات فيه كجدول يتألف من مفاتيح وقيم فأنه يمكنك تمرير tupleتحتوي على ارقام ونصوص كمفاتيح له.

مثال :

في المثال التالي نقوم بتعريف tubleوضعنا فيه نصوص. بعدها قمنا بعرض قيمة أول و ثاني عنصر فيه .

```
# هنا قمنا بتعريف tuple إسمه names وضعنا فيه قيم نصية تمثل أسماء أشخاص
names = ('Rami', 'Sara', 'Nada', 'Mhamad', 'Salem')
print(names[0]) # هنا قمنا بعرض قيمة أول عنصر موجود في الكائن names
print(names[1]) # هنا قمنا بعرض قيمة ثاني عنصر موجود في الكائن names

print(names[-1]) # هنا قمنا بعرض قيمة آخر عنصر موجود في الكائن names
print(names[-2]) # هنا قمنا بعرض قيمة ما قبل آخر عنصر موجود في الكائن names

print('.....')

for x in names:
print(x)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Rami
Sara

Salem
Mhamad

.....

Rami
Sara
Nada
Mhamad
Salem
```

مثال :

في المثال التالي قمنا بتعريف tuble وضعنا فيه أعداد صحيحة. بعدها قمنا بحساب ناتج جمع جميع القيم الموضوعة فيه باستخدام الحلقة for.

```
# هنا قمنا بتعريف tuple إسمه numbers وضعنا فيه أعداد صحيحة
numbers = (10, 20, 30, 40, 50)
# هنا قمنا بتعريف متغير سنستخدمه لحفظ ناتج جمع القيم الموجودة في الكائن numbers
total = 0
```

```
# total في كل مرة سيتم numbers في المتغير x و من ثم سيتم إضافتها على قيمة المتغير
وضع قيمة عنصر من عناصر الكائن
for x in numbers:
    total += x
# numbers هنا قمنا بطباعة قيمة total و التي ستساوي ناتج جمع القيم الموجودة في الكائن
المتغير
print('Total sum is:', total)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Total sum is: 150
```

تجزئ ال tuble

المقصود بالتجزئه هو إرجاع جزء منه سواء لعرضه أو تخزينه

مثال :

```
arr = (10, 20, 30, 40, 50) # هنا قمنا بتعريف tuple إسمه arr وضعنا فيه أعداد صحيحة
print(arr[0: 3]) # هنا قمنا بعرض أول ثلاث عناصر موجودة في الكائن arr

                    #arr2 = arr[0: 3]  يمكن كتابته بصورة أخرى
```

سنحصل على النتيجة التالية عند التشغيل.

```
(10, 20, 30)
```

استخدام العوامل + و * و in و ال tuple

العامل	دواعي الإستخدام
+	العامل + يستخدم لدمج tuple أو أكثر في tuple واحد.
*	العامل * لوضع لتكرار قيمة معينة عدة مرات في الـ tuple .
in	العامل in يستخدم للبحث في الـ tuple عن قيمة معينة أو للمرور على قيمه عند إستخدامه في الحلقة for كما فعلنا في بعض الأمثلة السابقة.

مثال على العامل +

```
arr1 = (1, 2, 3) # هنا قمنا بتعريف tuple إسمه arr1 وضعنا فيه أعداد صحيحة
arr2 = (4, 5, 6) # هنا قمنا بتعريف tuple إسمه arr2 وضعنا فيه أعداد صحيحة
arr3 = arr1 + arr2 # هنا قمنا بتعريف tuple إسمه arr3 وضعنا فيه قيم الكائنين arr1 و arr2
print(arr3) # هنا قمنا بعرض ما يحتويه الكائن arr3
```

سنحصل على النتيجة التالية عند التشغيل.

(1, 2, 3, 4, 5, 6)

مثال على العامل * (لماذا وضعنا , بعد python ؟؟؟؟؟؟؟؟؟؟؟؟)

```
arr = ('python',) * 3 # 'python' هنا قمنا tuple إسمه arr وضعنا فيه ٣ عناصر قيمتهم النص بتعريف
print(arr) # arr هنا قمنا بعرض ما يحتويه الكائن
```

سنحصل على النتيجة التالية عند التشغيل.

('python', 'python', 'python')

مثال على العامل in

```
arr = ('Mhamad', 'Rony', 'Rima', 'Sara') # tuple إسمه arr وضعنا فيه مجموعة قيم نصية هنا قمنا بتعريف
x = 'Rima' # هنا قمنا بتعريف متغير إسمه x وضعنا فيه نص
print('Is Rima in the tuple?')
print(x in arr) # True هنا سيتم البحث عن قيمة x في الكائن arr. إذا تم إيجادها سيتم عرض
```

سنحصل على النتيجة التالية عند التشغيل.

Is Rima in the tuple?

True

دوال الكلاس ال tuble

	إسم الدالة مع تعريفها
1	`count(x)` تستخدم للبحث في ال `tuple` الذي قام باستدعائها عن قيمة معينة. ترجع عدد صحيح يمثل كم مرة تم إيجاد عنصر عنده نفس القيمة التي مررناها لها مكان البارميتر `x` .
2	`index(x[, start[, end]])` تبحث في ال `tuple` الذي قام باستدعائها عن **Index** أول عنصر يملك القيمة التي نمرره لها مكان البارامیتر `x` و ترجعه. في حال لم يتم العثور على القيمة المراد معرفة مكانها ترمي الإستثناء `ValueError` .

مثال ١

```
# هنا قمنا بتعريف tuple إسمه atuple وضعنا فيه مجموعة أعداد صحيحة
atuple = (1, 3, 8, 4, 3, 3, 7, 2, 3, 4, 3)
# found لترجع كم مرة يوجد القيمة '٣' في الكائن atuple و من ثم وضعنا الناتج في المتغير
count() هنا قمنا باستدعاء الدالة
found = atuple.count(3);
# found هنا قمنا بعرض القيمة الموجودة في المتغير
print('Number 3 exists', found, 'in the tuple')
```

سنحصل على النتيجة التالية عند التشغيل.

```
Number 3 exists 5 in the tuple
```

مثال ٢

```
# هنا قمنا بتعريف tuple إسمه atuple وضعنا فيه مجموعة أعداد صحيحة
atuple = (1, 2, 3, 4, 5)
# 3 هنا قمنا بعرض index أول عنصر يملك القيمة
print(atuple.index(3))
#هنا قمنا بعرض index أول عنصر يملك القيمة ٣ و بدأنا عملية البحث من ثاني عنصر موجود
في الكائنatuple
print(atuple.index(3, 1))
#هنا قمنا بعرض index أول عنصر يملك القيمة ٣ و بدأنا عملية البحث من ثاني عنصر موجود
في الكائن و حتى آخر عنصر موجود فيه atuple
print(atuple.index(3, 1, 5))
# عند التشغيل ValueError أول عنصر يملك القيمة ١٠. و بما أنه لا يوجد أي عنصر عنده هذه
هنا قمنا بعرض index القيمة سيظهر الخطأ
print(atuple.index(10))
```

سنحصل على النتيجة التالية عند التشغيل.

```
2
2
2
ValueError: 10 is not in tuple
```

دوال جاهزة في بايثون للتعامل مع الـ tuble

	إسم الدالة مع تعريفها
1	`len(tuple)` ترجع عدد صحيح يمثل عدد عناصر الـ `tuple` الذي نمرره لها عند إستدعائها.
2	`min(tuple)` ترجع أصغر قيمة موجودة في الـ `tuple` الذي نمرره لها عند إستدعائها.
3	`max(tuple)` ترجع أكبر قيمة في الـ `tuple` الذي نمرره لها عند إستدعائها.
4	`tuple(sequence)` ترجع نسخة من أي كائن يحتوي مجموعة عناصر نمرره لها عند إستدعائها ككائن من الكلاس الـ `tuple` .

مثال

```
# هنا قمنا بتعريف tuple إسمه atuple وضعنا فيه مجموعة أعداد صحيحة
atuple = (5, 2, 4, 6, 3)
# min() هنا قمنا بعرض اصغر قيمة في الكائن atuple و التي سترجعها الدالة
print('Minimum value is:', min(atuple))

print('Maximum value is:', max(atuple))
```

سنحصل على النتيجة التالية عند التشغيل.

```
Minimum value is: 2

Maximum value is: 6
```

مثال ٤

```
# هنا قمنا بتعريف set إسمه aSet وضعنا فيه مجموعة أعداد صحيحة
aSet = {1, 2, 3, 4, 5}
# atuple لترجع نسخة منه  tuple. بعدها قمنا بتخزين كائن الـ tuple الذي سترجعه في الكائن
هنا قمنا باستدعاء الدالة tuple() على الكائن aSet ككائن من الكلاس
atuple = tuple(aSet)
# atuple هنا قمنا بعرض ما يحتويه الكائن
print(atuple)
```

سنحصل على النتيجة التالية عند التشغيل.

```
(1, 2, 3, 4, 5)
```

مفهوم الكلاس set

هو عبارة عن مصفوفة ليس لها حجم ثابت, يمكنها تخزين قيم من مختلف الأنواع في وقت واحد,و لا يمكن تبديل أو حذف قيمها بشكل مباشر .كما أنه لا يمكنها أن تحتوي على قيم مكررة. أي إذا وضعت فيها نفس القيمة مرتين فإنه سيتم تخزين قيمة واحدة فيها و ليس قيمتين.

النوع set لا يحافظ على الترتيب الذي تم فيه إدخال العناصر لأنه لا يضيف رقم **Index** لكل عنصر كما يفعل النوع list والنوع tuple.

لذلك لا تستغرب إذا قمت بتخزين مجموعة قيم بداخل set ثم حاولت عرضها. لأنك في كل مرة ستقوم فيها بتشغيل البرنامج من جديد ستتبدل أماكن القيم.

طريقة تعريف set

نستخدم الرمز { } بداخل هذا الرمز يمكنك تمرير القيم بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين. يمكن ان نضع داخلها اعداد صحيحة اونصوص او اعداد ونصوص معاً.

```
numbers = {10, 20, 30, 40, 50}

names = {'Rami', 'Sara', 'Nada'}

data = {1, 'Mhamad', 'Harmush', 1500}
```

مثال

```
# هنا قمنا بتعريف set إسمه names يحتوي على نصوص و لاحظ أننا قمنا بتكرار بعض القيم
names = {'Rami', 'Rami', 'Rami', 'Nada', 'Nada', 'Ahmad'}
# هنا قمنا بعرض ما names ( أي كما قمنا بتعريفه ) كما هو. لاحظ أنه لا يحتوي على قيم مكررة يحتويه الكائن
print(names)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل لأننا لا نعرف كيف سيتم ترتيب العناصر.

```
{'Ahmad', 'Rami', 'Nada'}
```

حذف ال set بواسطة del

```
# هنا قمنا بتعريف set إسمه arr وضعنا فيه أعداد صحيحة
arr = {10, 20, 30, 40, 50}
del arr # هنا قمنا بحذف الكائن arr كما هو من الذاكرة
print(arr) # هنا arr و الذي في الأصل قمنا بحذفه من الذاكرة لذلك سيظهر خطأ عند التشغيل حاولنا عرض ما يحتويه الكائن
```

سنحصل على النتيجة التالية عند التشغيل.

NameError: name 'arr' is not defined

البحث عن القيم بواسطة العامل in

مثال:

```
arr = {'Mhamad', 'Rony', 'Rima', 'Sara'} # هنا set إسمه arr وضعنا فيه مجموعة قيم نصية قمنا بتعريف
x = 'Rima' # هنا قمنا بتعريف متغير إسمه x وضعنا فيه نص
print('Is Rima in the set?')
print(x in arr) # هنا سيتم البحث عن قيمة x في الكائن arr. إذا تم إيجادها سيتم عرض True
```

سنحصل على النتيجة التالية عند التشغيل.

Is Rima in the set?

True

مثال

```
# هنا قمنا بتعريف set إسمه names وضعنا فيه قيم نصية تمثل أسماء أشخاص
names = {'Rami', 'Sara', 'Nada', 'Mhamad', 'Salem'}
# في كل مرة سيتم وضع قيمة عنصر من عناصر names في المتغير x و من ثم سيتم طباعتها الكائن
for x in names:
    print(x)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل لأننا لا نعرف كيف سيتم ترتيب العناصر.

Nada
Salem

```
Mhamad
Rami
Sara

num_rows = 4

num = 1

for i in range(num_rows):

    for j in range(i+1):

        print(num, end=" ")

        num += 1

    print()
```

دوال الكلاس set

	إسم الدالة مع تعريفها
1	`add(elem)` تستخدم لإضافة عنصر جديد في الـ `set` الذي قام باستدعائها. شاهد المثال »
2	`discard(elem)` تستخدم لحذف عنصر محدد من الـ `set` الذي قام باستدعائها. مكان البارامتر `elem` نمرر كائن قيمته تطابق قيمة العنصر الذي نريد حذفه. شاهد المثال »
3	`remove(elem)` تستخدم لحذف عنصر محدد من الـ `set` الذي قام باستدعائها. مكان البارامتر `elem` نمرر كائن قيمته تطابق قيمة العنصر الذي نريد حذفه. **ملاحظة:** الفرق بينها و بين الدالة `discard()` أنها ترمي الإستثناء `KeyError` في حال لم يتم إيجاد عنصر في الـ `set` يملك نفس قيمة الكائن الذي مررناه مكان البارامتر `elem` . شاهد المثال »
4	`clear()` تستخدم لحذف جميع عناصر الـ `set` الذي قام باستدعائها. شاهد المثال »
5	`pop()` تستخدم لإرجاع قيمة عنصر يتم إختياره بشكل عشوائي من الـ `set` الذي قام باستدعائها, بعدها يتم حذفه منه. شاهد المثال »
6	`copy()` ترجع نسخة من الـ `set` الذي قام باستدعائها. شاهد المثال »
7	`difference(*sets)` ترجع `set` جديد يحتوي على العناصر الموجودة في الـ `set` الذي قام باستدعائها و الغير موجودة في كل `set` نمرره لها مكان البارامتر `sets*` . شاهد المثال »
8	`difference_update(*sets)` تقارن الـ `set` الذي قام باستدعائها مع كل `set` نمرره لها مكان البارامتر `sets*` . بعدها تحذف من الـ `set` الذي قام باستدعائها العناصر المشتركة بينهم. شاهد المثال »
9	`intersection(*sets)` ترجع `set` جديد يحتوي على العناصر الموجودة في الـ `set` الذي قام باستدعائها و في كل `set` نمرره لها مكان البارامتر `sets*` . شاهد المثال »

10	`intersection_update(*sets)` تقارن الـ `set` الذي قام باستدعائها مع كل `set` نمرره لها مكان البارامتر `sets*` . بعدها تحذف من الـ `set` الذي قام باستدعائها العناصر الغير مشتركة بينهم. شاهد المثال »
11	`symmetric_difference(anotherSet)` ترجع `set` جديد يحتوي على العناصر الغير موجودة في كل من الـ `set` الذي قام باستدعائها و في الـ `set` الذي نمرره لها مكان البارامتر `anotherSet` . شاهد المثال »
12	`symmetric_difference_update(anotherSet)` تقارن الـ `set` الذي قام باستدعائها مع الـ `set` الذي نمرره لها مكان البارامتر `anotherSet` . بعدها تضع في الـ `set` الذي قام باستدعائها العناصر المشتركة بينهما فقط و تمسح أي عناصر آخرى كانت موجودة فيه. شاهد المثال »
13	`union(sets*)` ترجع `set` جديد يحتوي على العناصر الموجودة في الـ `set` الذي قام باستدعائها و العناصر الموجودة في كل `set` نمرره لها مكان البارامتر `sets*` . شاهد المثال »

	إسم الدالة مع تعريفها
1	`len(set)` ترجع عدد صحيح يمثل عدد عناصر الـ `set` الذي نمرره لها عند إستدعائها. شاهد المثال »
2	`min(set)` ترجع أصغر قيمة موجودة في الـ `set` الذي نمرره لها عند إستدعائها. شاهد المثال »
3	`max(set)` ترجع أكبر قيمة في الـ `set` الذي نمرره لها عند إستدعائها. شاهد المثال »
4	`set(sequence)` ترجع نسخة من أي كائن يحتوي مجموعة عناصر نمرره لها عند إستدعائها ككائن من الكلاس الـ `set` . شاهد المثال »

مثال: في هذا المثال تم تطبيق جميع دوال Set بأنشاء عدد من كائن set حسب الحاجة :

```
aSet = {'Apple', 'Banana', 'Mango'}
print('Before:', aSet) #عرض set الاصلية
aSet.add('Orange') #اضافة عنصر في نهاية القائمة
print('After: ', aSet) #عرض بعد اضافة عنصر
aSet.discard('Apple')#حذف العنصر الذي يملك القيمة بين القوسين
print('discard',aSet)
aSet.remove('Mango')#الذي يملك القيمة بين القوسينحذف العنصر
print('remove',aSet)
aSet.clear() #حذف الكائن بالكامل
print('aSet clear :', aSet)
print('.....')
aSet = {10, 20, 30, 40, 50}
print('Returned element pop:', aSet.pop())#استخراج اخر عنصر موجود في الكائن
print('Remaining elements:', aSet)#طباعة باقي العناصر
print('.....')
set1 = {1, 2, 3, 4, 5}
set2 = set1.copy()#نسخ عناصر الكائن في الكائن set2
print('set1 contains:', set1)
print('set2 copy:', set2)
print('.....')
set1 = {1, 2, 3, 4, 5}
set2 = {1, 2, 3}
newSet = set1.difference(set2)#انشاء نسخه من العناصر الموجوده في Set1 والغير موجوده في set2 ووضعها في الكائن newset
print('newSet difference:', newSet)
print('.....')
set1 = {1, 2, 3, 4, 5}
set2 = {1, 2, 3}
```

```
set1.difference_update(set2)#حذف العناصر الموجوده في الكائنين وعرض الباقي في Set

print('set1 difference_update:', set1)

print('.....')

set1 = {1, 2, 3, 4, 5}

set2 = {1, 2, 3}

newSet = set1.intersection(set2)#العناصر المشتركة بين الكائنين

print('newSet intersection:', newSet)

print('.....')

set1 = {1, 2, 3, 4, 5}

set2 = {1, 2, 3}

set1.intersection_update(set2)#حذف العناصر الغير موجوده في الكائنين

print('set1 intersection_update:', set1)

print('.....')

set1 = {1, 2, 3, 4, 5}

set2 = {1, 2, 3}

newSet = set1.symmetric_difference(set2)# انشاء نسخه من العناصر الغير موجوده في
الكائنين

print('newSet symmetric_difference:', newSet)

print('.....')

set1 = {1, 2, 3, 4, 5}

set2 = {1, 2, 3}

set1.symmetric_difference_update(set2)

print('set1 symmetric_difference_update:', set1)

set1 = {1, 2, 3}

set2 = {1, 2, 3, 4, 5}

newSet = set1.union(set2)# استرجاع جميع عناصر الكائنين بدون تكرار

print('newSet union :', newSet)

aSet = {1, 2, 3, 4, 5}

print('Array length is:', len(aSet))
```

```
print('Minimum value is:', min(aSet))
print('Maximum value is:', max(aSet))
alist= [1, 2, 3, 4, 5]
aSet = set(alist)# استرجاع عناصر set على شكل list
print('Set as List ',aSet)
```

الناتج

```
Before: {'Banana', 'Apple', 'Mango'}
After:  {'Orange', 'Banana', 'Apple', 'Mango'}
discard {'Orange', 'Banana', 'Mango'}
remove {'Orange', 'Banana'}
aSet clear : set()
.....
Returned element pop: 50
Remaining elements: {20, 40, 10, 30}
.....
set1 contains: {1, 2, 3, 4, 5}
set2 copy: {1, 2, 3, 4, 5}
.....
newSet difference: {4, 5}
.....
set1 difference_update: {4, 5}
.....
newSet intersection: {1, 2, 3}
.....
set1 intersection_update: {1, 2, 3}
.....
```

newSet symmetric_difference: {4, 5}

.....

set1 symmetric_difference_update: {4, 5}

newSet union : {1, 2, 3, 4, 5}

Array length is: 5

Minimum value is: 1

Maximum value is: 5

Set as List {1, 2, 3, 4, 5}

مفهوم الكلاس Dict

كلمة Dict هي اختصار ل dictionaryوالتي تعني القاموس.

Dict عبارة عن جدول يتألف من عامودين, الأول يحتوي المفاتيح (**Keys**) و الثاني يحتوي القيم (**Values**) الخاصة بكل عنصر. كل عنصر يتم إضافته في dict جب إعطاؤه قيمتين. الأولى تمثل المفتاح (**Key**) و الثانية تمثل قيمته (**Value**). المفاتيح تستخدم للوصول إلى القيم, لهذا لا يمكن وجود عنصرين في dict عندهم نفس المفتاح. إذاً, كل **Key**موضوع يسمح لك بالوصول لقيمة واحدة من القيم الموجودة في ال dict.

طريقة تعريف dict

لتعريف dict نستخدم الرمز { } داخل هذا الرمز يمكنك تمرير العناصر بشكل مباشر له بشرط وضع فاصلة بين كل عنصرين.

لا تنسى أن كل عنصر يجب أن يملك قيمتين, الأولى تمثل المفتاح و الثانية تمثل القيمة. بين كل مفتاح و قيمة نضع الرمز : .

عناصر ال dict يمكن ان تكون فارغة وهنا لا فرق بينها وبين set او تكون ارقام , نصوص او الاثنين معاً سواء للقيم او للمفاتيح كما سنلاحظ في الأمثلة التالية :

مثال

```
# هنا قمنا بتعريف dict يتألف من ثلاث عناصر, إسمه data
data = {
```

```
    1: 'Admin',
    2: 'Editor',
    3: 'Reader'
}
Print(data)
# هنا قمنا بعرض ما يحتويه الكائن data ( أي كما قمنا بتعريفه ) كما هو
```

{1: 'Admin', 2: 'Editor', 3: 'Reader'}

مثال

```
# هنا قمنا بتعريف dict يتألف من ثلاث عناصر, إسمه data
data = {
    'id': 1,
    'name': 'Mhamad',
    'mobile': 70123456
}
Print(data1)
# هنا قمنا بعرض ما يحتويه الكائن data ( أي كما قمنا بتعريفه ) كما هو
```

{'id': 1, 'name': 'Mhamad', 'mobile': 70123456}

الوصول لقيم عناصر ال dict

للوصول لاي عنصر في ال dict سواء للحصول على قيمته أو تغييرها أو حذفها فإننا نستخدم المفتاح الخاص بالعنصر.

مثال

```
# هنا dict يتألف من ثلاث عناصر وعرض قيمة العنصر الذي يملك المفتاح رقم ١, إسمه data قمنا بتعريف
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# هنا قمنا بطباعة قيمة العنصر الذي يملك المفتاح رقم 1
print(data[1])
```

سنحصل على النتيجة التالية عند التشغيل.

```
Admin
```

مثال تم استخدام دالة get لعرض قيمة العنصر الذي يمكل المفتاح رقم ١

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر,إسمه
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# هنا قمنا بطباعة قيمة العنصر الذي يملك المفتاح رقم 1
print(data.get(1))
```

سنحصل على النتيجة التالية عند التشغيل.

```
Admin
```

مثال : استخدام الحلقة for لعرض جميع المفاتيح الموضوعه في dict

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر, إسمه
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# في كل مرة سيتم وضع مفتاح عنصر من عناصر data في المتغير key و من ثم سيتم طباعته
الكائن
for key in data:
print(key)
```

سنحصل على النتيجة التالية عند التشغيل.

```
1
2
3
```

إضافة عناصر في ال dict

لاضافة عنصر جديد في ال dict فإننا نمرر مفتاح جديد و من بعدها نضع القيمة التي يساويها.

مثال

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر, إسمه
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# 'Author' هنا قمنا بإضافة عنصر جديد في الكائن data, مفتاحه الرقم ٤, و قيمته النص
data[4] = 'Author'
# data هنا قمنا بعرض ما أصبح يحتويه الكائن
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{1: 'Admin', 2: 'Editor', 3: 'Reader', 4: 'Author'}
```

تبديل قيم عناصر ال dict

لتبديل قيمة أي عنصر في ال dict فإننا نمرر مفتاح العنصر الذي نريد تبديل قيمته و من بعدها نضع القيمة الجديدة.

مثال :

```
# data يتألف من ثلاث عناصر قمنا بتبديل قيمة العنصر الذي يمكل مفتاح يساوي ٣, إسمه dict
هنا قمنا بتعريف
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# 'Subscriber' هنا قمنا بتبديل قيمة العنصر الذي يملك المفتاح  data. وضعنا بدلاً منها النص
رقم ٣ في الكائن
data[3] = 'Subscriber'
# data هنا قمنا بعرض ما أصبح يحتويه الكائن
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{1: 'Admin', 2: 'Editor', 3: 'Subscriber'}
```

حذف عناصر ال dict بواسطة Del

مثال

```
# data هنا قمنا بتعريف dict يتألف من ثلاث عناصر وحذف عنصرين منه, إسمه
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# data هنا قمنا بحذف العنصر الذي يملك المفتاح رقم ٣ في الكائن
del data[3]
# data هنا قمنا بعرض ما أصبح يحتويه الكائن
print(data)
```

سنحصل على النتيجة التالية عند التشغيل.

```
{1: 'Admin', 2: 'Editor'}
```

معرفة عدد عناصر ال dict

لمعرفة عدد عناصر كائن ال dict نستخدم الدالة ( )len

مثال

```
# data هنا قمنا بتعريف dict إسمهlenيتألف من ثلاث عناصر, قمنا بطباعة عدد عناصره بواسطة الدالة
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
# len() هنا قمنا بعرض عدد عناصر الكائن data الذي سترجعه الدالة
print(len(data))
```

سنحصل على النتيجة التالية عند التشغيل.

```
3
```

مثال شامل لجميع ما سبق

```
data = {
    1: 'Admin',
    2: 'Editor',
    3: 'Reader'
}
print(data)
print(data[1]) # استرجاع العنصر الذي يمتلك المفتاح رقم ١
print(data.get(1)) #استرجاع العنصر باستخدام الدالة  الذي يمتلك المفتاح رقم ١
for key in data: # استخدام الدوارة لطباعة جميع عناصر المفتاح
    print(key)
data[4] = 'Author' # إضافة عنصر الى الكائن
print(data)
data[3] = 'Subscriber' # استبدال العنصر في المفتاح ٣ بالقيمة المعطاة
print(data)
del data[3]# حذف العنصر الذي يملك المفتاح رقم ٣
print(data)
print(len(data))# إيجاد طول الكائن
data1 = {
    'id': 1,
    'name': 'Mhamad',
    'mobile': 70123456
}
print(data1)
```

دوال الكلاس dict

	إسم الدالة مع تعريفها
1	`clear()` تستخدم لحذف جميع عناصر الـ `dict` الذي قام باستدعائها. شاهد المثال »
2	`copy()` ترجع نسخة من الـ `dict` الذي قام باستدعائها. شاهد المثال »
3	`popitem()` ترجع آخر عنصر تم إضافته في كائن الـ `dict` الذي قام باستدعائها و من ثم تحذفه منه. في حال لم يتم العثور على القيمة المراد حذفها ترمي الإستثناء `KeyError` . شاهد المثال »
4	`pop(key[, default])` تبحث في كائن الـ `dict` الذي قام باستدعائها عن العنصر الذي يملك نفس المفتاح الذي نمرره لها مكان البارامتر `key` و من ثم تحذفه منه. مكان البارامتر `default` يمكنك تمرير قيمة إفتراضية يتم إرجاعها فقط في حال لم يتم العثور على عنصر يملك المفتاح الذي مررناه لها مكان البارامتر `key` . في حال لم يتم العثور على المفتاح المراد حذف العنصر الذي يملكه و لم تمرر لها قيمة مكان البارامتر `default` ترمي الإستثناء `KeyError` . شاهد المثال »
5	`get(key[, default])` تبحث في كائن الـ `dict` الذي قام باستدعائها عن قيمة المفتاح الذي نمرره لها مكان البارامتر `key` و ترجعها. مكان البارامتر `default` يمكنك تمرير قيمة إفتراضية يتم إرجاعها فقط في حال لم يتم العثور على عنصر يملك المفتاح الذي مررناه لها مكان البارامتر `key` . في حال لم يتم العثور على المفتاح المراد حذف العنصر الذي يملكه و لم تمرر لها قيمة مكان البارامتر `default` فإنها ترجع `None` . شاهد المثال »

6	`keys()` ترجع مصفوفة نوعها `dict_keys` فيها جميع المفاتيح الموجودة في كائن الـ `dict` الذي قام باستدعائها. **ملاحظة:** مصفوفة المفاتيح التي يتم إرجاعها, هي ليست نسخة عن المفاتيح الموجودة في كائن الـ `dict` بل هي نفسها. و بالتالي أي تعديل يتم إجراؤه على المفاتيح من كائن الـ `dict` ينعكس على مصفوفة المفاتيح و العكس كذلك. شاهد المثال »
7	`values()` ترجع مصفوفة نوعها `dict_keys` فيها جميع القيم الموجودة في كائن الـ `dict` الذي قام باستدعائها. **ملاحظة:** مصفوفة القيم التي يتم إرجاعها, هي ليست نسخة عن القيم الموجودة في كائن الـ `dict` بل هي نفسها. و بالتالي أي تعديل يتم إجراؤه على القيم من كائن الـ `dict` ينعكس على مصفوفة القيم و العكس كذلك. شاهد المثال »
8	`items()` ترجع مصفوفة نوعها `dict_items` فيها جميع العناصر الموجودة في كائن الـ `dict` الذي قام باستدعائها. **ملاحظة:** مصفوفة العناصر التي يتم إرجاعها, هي ليست نسخة عن العناصر الموجودة في كائن الـ `dict` بل هي نفسها. و بالتالي أي تعديل يتم إجراؤه على العناصر من كائن الـ `dict` ينعكس على مصفوفة العناصر و العكس كذلك. شاهد المثال »
9	`fromkeys(seq[, value])` تستخدم لإنشاء `dict` جديد مبني على قيم مصفوفة نمررها له. عند إستدعائها نمرر لها كائن يمثل مصفوفة من أي نوع مكان البارامتر `seq` , عناصر هذا الكائن يتم وضعها كمفاتيح في كائن الـ `dict` الذي سيتم إرجاعه. مكان البارامتر `value` يمكنك تمرير القيمة الإفتراضية التي تريد وضعها لجميع عناصر كائن الـ `dict` الذي سيتم إرجاعه. و في حال لم تقم بتمرير قيمة إفتراضية مكان البارامتر `value` ستكون جميع قيمه تساوي `None` . شاهد المثال »
10	`setdefault(key[, default])` تستخدم للحصول على قيمة مفتاح موجود في كائن الـ `dict` الذي قام باستدعائها, أو لإضافة عنصر جديد فيه مع إرجاع قيمته أيضاً. مكان البارامتر `key` نمرر مفتاح العنصر الذي إن كان موجوداً في كائن الـ `dict` سيتم إرجاع قيمته فقط, و إن لم يكن موجوداً سيتم إضافته و إرجاع قيمته. مكان البارامتر `default` يمكنك تمرير القيمة الإفتراضية التي تريد وضعها للعنصر في حال تم إضافته لأنه إفتراضياً إن تم إضافة عنصر جديد ستكون قيمته `None` . شاهد المثال »
11	`update([other])` تستخدم لتحديث قيم عناصر الـ `dict` الذي قام باستدعائها على أساس المفاتيح الموجودة فيه. مكان البارامتر `other` يمكننا أن نمرر كائن `dict` فيه عنصر واحد أو مجموعة العناصر التي نريد تحديث قيمها أو إضافتها في كائن الـ `dict` الذي قام باستدعائها. أي عنصر نمرره مكان البارامتر `other` يملك مفتاح غير موجود في كائن الـ `dict` الذي قام باستدعائها سيتم إضافته فيه. شاهد المثال »

مثال شامل لجميع دوال dic

```
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
print('dic data1 ',data)
data2 = data.copy()
print('dict2 copy:', data2)
print('Returned element:', data.popitem())
print('Remaining elements from popitem:', data)
print('Returned element:', data.pop(2))
print('Remaining elements form pop:', data)
print('Returned value:', data.get(2))
keys = data.keys()
print(keys)
values = data.values()
print(values)
items = data.items()
print(items)
aTuple = (1, 2, 3, 4)
aDict = dict.fromkeys(aTuple)
print(aDict)
print('Returned value:', data.setdefault(3))
data.update({3: 'Subscriber'})
print(data)
```

الناتج

```
data = {
1: 'Admin',
2: 'Editor',
3: 'Reader'
}
print('dic data1 ',data)
data2 = data.copy()# قمنا بنسخ عناصر من كائن الى كائن اخر
print('dict2 copy:', data2)
print('Returned element:', data.popitem())# استخراج اخر عنصر في الكائن
print('Remaining elements from popitem:', data)
print('Returned element:', data.pop(2)) # استخراج العنصر الذي يملك المفتاح رقم ٢
print('Remaining elements form pop:', data)
print('Returned value:', data.get(2))# الحصول على قيمة العنصر الذي يملك المفتاح رقم ٢
keys = data.keys()# استرجاع مفاتيح الكائن
print(keys)# طباعة المفاتيح
values = data.values()# استرجاع قيم الكائن فقط بدون مفاتيح
print(values)
items = data.items()# استرجاع عناصر ومفاتيح الكائن
print(items)
print('Returned value:', data.setdefault(3))# الحصول على قيمة العنصر الذي يملك المفتاح رقم ٣
data.update({3: 'Subscriber'})# تحديث قيمة العنصر الذي يملك المفتاح رقم ٣
print(data)
data.clear() # حذف جميع العناصر الموجودة في الكائن
print('data contains:', data)
```

الناتج

```
dic data1  {1: 'Admin', 2: 'Editor', 3: 'Reader'}
dict2 copy: {1: 'Admin', 2: 'Editor', 3: 'Reader'}
Returned element: (3, 'Reader')
Remaining elements from popitem: {1: 'Admin', 2: 'Editor'}
Returned element: Editor
Remaining elements form pop: {1: 'Admin'}
Returned value: None
dict_keys([1])
dict_values(['Admin'])
dict_items([(1, 'Admin')])
Returned value: None
{1: 'Admin', 3: 'Subscriber'}
data after clear  {}
```

مفهوم الدوال

الدالة function عبارة عن مجموعة أوامر مجمعة في مكان واحد و تتنفذ عندما نقوم باستدعائها.

بايثون تحتوي على مجموعة كبيرة جداً من الدوال الجاهزة و التي سبق أن إستخدامنا مثل print & min & max

ملاحظة : الدوال الجاهزة في بايثون, يقال لها **Built-in Functions**.

الدوال التي يقوم المبرمج بتعريفها, يقال لها **User-defined Functions**.

تعريف دوال جديدة

الشكل الأساسي الذي يجب إتباعه عند تعريف أي دالة في بايثون هو التالي:

```
def functionname():
    function_suite
```

def تعني انك تعرف دالة جديدة

Ffunctionname**:** نضع مكانها الإسم الذي نعطيه للدالة, و الذي من خلاله يمكننا استدعاءها.

( ) : بداخل القوسين يمكنك وضع باراميترات و يجب أن تضع (:) مباشرةً بعد القوسين و من ثم تنزل على سطر جديد لتبدأ بكتابة الأوامر التي ستتنفذ عند إستدعاء الدالة.

Function_suite: تعني الأوامر التي سنضعها في الدالة و التي ستتنفذ عند إستدعائها.

ملاحظة : يجب وضع **4** مسافات فارغة قبل الأوامر التي ستضعها في الدالة حتى يعرف مفسّر لغة بايثون أن هذه الاوامر موجودة بداخل الدالة. للترتيب ولكتابة الكود كما يفعل باقي المبرمجين, قم بإضافة سطرين فارغين بعد تعريف الدالة.

```
# هنا قمنا بتعريف دالة إسمها my_function
def my_function():
    print('My first function is called')
# هنا قمنا باستدعاء الدالة my_function حتى يتنفذ الأمر الموضوع فيها
my_function()
```

سنحصل على النتيجة التالية عند التشغيل.

```
My first function is called
```

جميع المعلومات التي يمكن ذكرها عند تعريف دالة جديدة

قبل قليل تكلمنا عن الأشياء الأساسية التي يجب أن تكون متوفرة عند تعريف أي دالة. الآن, عليك معرفة أنه يمكنك وضع مزيد من التفاصيل بشرط أن تضيفها ضمن ترتيب محدد و ليس شرطاً أن تضيفها كلها.

```
def functionname(parameters):
 """ function_docstring """
function_suite
  return [expression]
```

def تعني انك تعرف دالة جديدة

Parameter: المقصود بها البارميترات التي نمررها لها عند استدعائها

functionname: نضع مكانها الإسم الذي نعطيه للدالة, و الذي من خلاله يمكننا استدعاءها.

function_docstring : نضع مكانها نص الهدف منه تفسير ما تفعله الدالة بشكل مختصر (وضع التفسير هو أمر إختياري)

Function_suite: تعني الأوامر التي سنضعها في الدالة و التي ستتنفذ عند إستدعائها.

return [expression] : نضع مكانها ما يمكن أن ترجعه الدالة في المكان الذي تم إستدعاءها منه)إرجاع قيمة هو أمر إختياري.(

مثال :

```
# تحتوي على بارامتر واحد. عند إستدعائها نمرر لها إسم, فتطبع جملة ترحيب بإسم الشخص
هنا قمنا بتعريف دالة إسمها greeting الذي نمرره لها
def greeting(name):
""" This function print hello message based on the specified name """
print('Hello '+name+', welcome to our company.')
# user هنا قمنا بتخزين إسم الشخص الذي سنمرره للدالة في المتغير
user = 'Ahmad'
# و تمرير إسم الشخص الذي قمنا بتخزينه في المتغير user حتى تطبع رسالة ترحيب له
greeting() هنا قمنا باستدعاء الدالة
greeting(user)
```

سنحصل على النتيجة التالية عند التشغيل.

```
Hello Ahmad, welcome to our company.
```

مثال

```
# تحتوي على بارامتر واحد. عند إستدعائها نمرر لها إسم, فتطبع جملة ترحيب بإسم الشخص الذي نمرره لها greeting هنا قمنا بتعريف دالة إسمها
def greeting(name):
""" This function print hello message based on the specified name """
print('Hello '+name+', welcome to our company.')
# greeting() هنا قمنا بطباعة الشرح المرفق بالدالة
print(greeting.__doc__)
```

سنحصل على النتيجة التالية عند التشغيل.

This function print hello message based on the specified name

مثال:

```
# عند إستدعائها يمكنك تمرير قيمة language و يمكنك عدم تمرير قيمة لأنه أصلاً يملك قيمة هنا قمنا بتعريف دالة إسمها print_language. لها مكان البارامتر
def print_language(language='English'):
print('Your language is:', language)
# 'English' بدون تمرير قيمة مكان البارامتر language و بالتالي ستظل قيمته هنا قمنا باستدعاء الدالة print_language()
print_language()
# 'Arabic' مع تمرير القيمة 'Arabic' للبارامتر language و بالتالي ستصبح قيمته هنا قمنا باستدعاء الدالة print_language()
print_language('Arabic')
```

سنحصل على النتيجة التالية عند التشغيل.

Your language is: English

Your language is: Arabic

بناء دوال تقبل عدد غير محدد من القيم عند إستدعائها

في بعض الأحيان قد تحتاج إلى بناء دالة تعالج عدد غير محدد من القيم عند استدعائها. أي مهما كان عدد القيم التي ستمررها لها فإنها يجب أن تعالجهم كلهم.

لبناء دالة يمكن تمرير عدد غير محدد من القيم لها عند إستدعائها عليك تجهيز بارامتر واحد لهذا الأمر.

نضع (*) قبل إسم البارامیتر و عندها سیفهم مفسّر لغة بایثون أنه یمكنك تمریر عدد غیر محدد من القیم لهذا البارامیتر. عندها سیتم تجمیع كل القیم التي تمررها للدالة بداخل tuple مما یسهل التعامل معها .

```
# عند إستدعائها یمكننا تمریر عدد غیر محدد من القیم لها. بعدها ستقوم بطباعة القیم التي مررناها لها print_args هنا قمنا بتعریف دالة إسمها
def print_args(*args):
for e in args:
print(e)
# هنا قمنا بإستدعاء الدالة print_args() مع تمریر ١٠ قیم لها
print_args(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

سنحصل على النتیجة التالیة عند التشغیل.

```
1
2
3
4
5
6
7
8
9
10
```

مثال

```
# هنا قمنا بتعریف متغیر إسمه x قیمته تساوي ١
x = 1
# هنا قمنا بتعریف دالة إسمها test تقوم بتغیر قیمة x الذي تم تعریفه خارجها
def test():
global x
x = 5
# global حتى تقوم x الذي قمنا بتعریفه في الأساس خارجها و الذي وصلنا إلیه بواسطة الكلمة
هنا قمنا باستدعاء الدالة test() بتغییر قیمة المتغیر
test()
# هنا قمنا بطباعة قیمة x الموجودة في خارج الدالة. لاحظ أنها بقیت كما هي
print('Global x =', x)
```

سنحصل على النتیجة التالیة عند التشغیل.

```
Global x = 5
```

عناصر الواجهات الرسومية الأساسية

لبرمجة الواجهات مجموعتها الخاصة من المصطلحات البرمجية، وأكثر المصطلحات استخدامًا فيها هي:

النافذة window

جزء من الشاشة يتحكم فيه التطبيق، وتكون النوافذ مربعةً في العادة، لكن قد تسمح بعض البيئات الرسومية بأشكال أخرى، وقد تحتوي النوافذ على نوافذ أخرى داخلها، ويُعامل كل متحكم رسومي GUI control على أنه نافذة بذاته.

المتحكم Control

كائن رسومي يُستخدم للتحكم في التطبيق، وتحتوي المتحكمات على خصائص properties ، وتولّد أحداثًا events ، وتستجيب عادةً لكائنات على مستوى التطبيق، حيث تُرفَق الأحداث بتوابع الكائن الموافق corresponding object ، فإذا وقع الحدث نفّذ الكائن أحد توابعه، وتوفر الواجهة الرسومية آليات لربط الأحداث بالتوابع.

الودجِت Widget

متحكم مقيَّد عادةً بالمتحكمات المرئية، إذ يمكن ربط بعض المتحكمات -مثل المؤقتات timers- بنافذة ما، لكن دون أن تكون مرئيةً، أما الودجات فهي فئة مرئية فرعية من المتحكمات، ويمكن للمستخدم أو المبرمج أن يعدل فيها.

Tkinter

تعد مكتبة Tkinter -وتنطق T-K-Inter اختصار لكلمة Tk Interface أي واجهات TK- الأشهر والأكثر استخدامًا، ومن حسن الحظ أنها تأتي مدمجة مع لغة البايثون النسخة ٣، فلا حاجة لتثبيت أي مكون إضافي لاستخدامها، فهي مكتبة لتطوير الواجهات الرسومية تحوي مجموعة أدوات لعناصر واجهة المستخدم وهي مفتوحة المصدر تستخدمها عدة لغات لتطوير الواجهات لأنظمة ويندوز وماك ويونكس.

تعد المكتبة Tkinter في بايثون اختيارًا جيدًا لإنشاء الواجهات الرسومية لعدة أسباب أهمها أنها سهلة التعلم، ويستخدم فيها القليل جدًا من التعليمات البرمجية لإنشاء تطبيق سطح مكتب يعمل بشكل ممتاز، ونستطيع تشغيلها على مختلف أنظمة التشغيل، وكما قلنا

مسبقًا لا نحتاج أي مجهودٍ في تثبيتها فهي تأتي مع البايثون بشكل افتراضي. كل هذه المميزات تجعلها نقطة انطلاق قوية للمبتدئين والمتوسطين لتعلم الواجهات الرسومية في البايثون.

سنسلط الضوء في هذه السلسلة على المكونات الأساسية في مكتبة Tkinter لصنع واجهة مستخدم رسومية للشفرات البرمجية في بايثون

تعد عملية إنشاء واجهة رسومية باستخدام Tkinter مهمة سهلة تتضمن مجموعة من الخطوات:

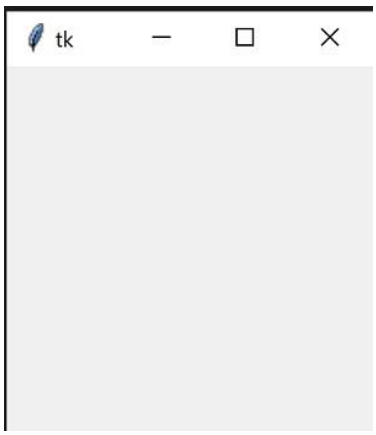
- استيراد الوحدة.Tkinter
- إنشاء النافذة الرئيسية (التي ستحتوي على جميع العناصر الرسومية.(
- إضافة أي عدد من المكونات التي تحتاجها إلى النافذة الرئيسية.
- تفعيل دوال الاستجابة لمكوناتك الرسومية.ص٢

```
import tkinter as TK

main_window = TK.Tk()
```

- بعد ذلك نقوم باستدعاء الدالة mainloop والتي تقوم بعمل استمرار لانهائي لظهور النافذة بانتظار بانتظار أن يتفاعل معها المستخدم أو يغلقها، ويكون ذلك بإضافة السطر التالي:

```
import tkinter as TK

main_window = TK.Tk()
main_window.mainloop()
```

- عند تشغيل هذه الشفرة ستظهر لنا نافذة بالشكل التالي:



- النافذة الرئيسية
- يمكننا إعادة كتابة الشفرة باستخدام الصيغة الثانية لاستدعاء المكتبة كالتالي:

```
from tkinter import *

main_window = Tk()
main_window.mainloop()
```

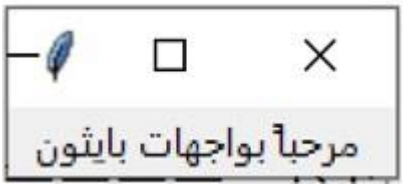
- يستدعي السطر الأول جميع محتويات المكتبة Tkinter من أصناف ودوال وثوابت لتكون من ضمن برنامجك، لذلك نستخدم اسم المصنف فقط دون الحاجة إلى استخدام اسم المكتبة قبل اسم المصنف كما في السابق، وعند تشغيل الشفرة سنحصل على نفس النتيجة للشفرة بالصيغة الأولى.
- لنقم الآن بإضافة عنصر داخل نافذتنا الرئيسية لنكتب فيه النص المراد إظهاره، لعمل ذلك نحتاج أن ننشئ كائنًا (نختار له اسمًا) من نوع المصنف Label يحتاج هذا الكائن على الأقل إلى تحديد المكان الذي يجب أن يوضع فيه بالإضافة إلى النص الذي تريد كتابته، فتكتب الشفرة على النحو التالي:

```
L=Label(main_window, text="مرحباً بواجهات بايثون ")
```

- بعد ذلك نقوم باستخدام الدالة pack والتي تقوم بوضع الكائن على نافذتنا الرئيسية لتكون الشفرة كاملةً على النحو التالي:

```
from tkinter import *
main_window = Tk()

L=Label(main_window, text="مرحباً بواجهات بايثون ")
L.pack()

main_window.mainloop()
```

- عند تشغيل البرنامج سنحصل على الشكل التالي:



- نافذة بنص
- **خصائص النافذة الرئيسية**

- هناك الكثير من الخصائص التي يمكن تنسيقها للنافذة لترتيب مظهرها من أبعاد ولون خلفية وحدود ونحوه، الفقرات التالية تشرح أهم الخصائص:
- عنوان النافذة
- نضيف عنوانًا لنافذتنا الرئيسية باستخدام الدالة title مع الكائن الذي قمنا بتعريفه لها، بطبيعة الأمر سنقوم بإرسال النص الذي نريد كتابته كعنوان للدالة title ونضيفه إلى الشفرة السابقة في السطر الثالث بهذه الطريقة:

```
from tkinter import *
main_window = Tk()
main_window.title("أول واجهة رسومية ")
L=Label(main_window, text="مرحباً بواجهات بايثون ")
L.pack()
main_window.mainloop()
```

- فنحصل على الشكل التالي:

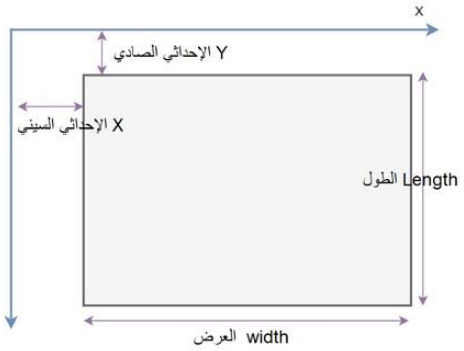
- نافذة بعنوان
- وعلى عكس العملية السابقة بإمكاننا استرجاع عنوان أي نافذة باستخدام القيمة المرجعة من نفس الدالة title ولكن بدون إرسال أي شي لها على النحو التالي:

```
the_title=main_window.title()
```

- وبناءً على مثالنا السابق يفترض أن تكون قيمة المتغير the_title هي جملة "أول واجهة رسومية."

```
the_title=main_window.title()
```

- وبناءً على مثالنا السابق يفترض أن تكون قيمة المتغير the_title هي جملة "أول واجهة رسومية."
- حجم ومكان ظهور النافذة
- نتحكم بحجم النافذة وكذلك مكان ظهورها باستخدام الدالة geometry وفيها نحتاج لتحديد الطول والعرض بالبكسل، وكذلك الإحداثي السيني والصادي للمكان المراد ظهور النافذة فيه في شاشة العرض، ويوضح الشكل التالي التفاصيل بالنسبة للشاشة:



- احداثيات النافذة
- الطريقة العامة لاستخدام الدالة على النحو التالي:

```
main_window.geometry('WidthxLength+x(horizontal)+y(vertical)')
```

- يجب مراعاة أن التفاصيل تعطى على صيغة سلسلة نصية أي أنها محاطة بعلامات التنصيص.
- لنحدد طولًا وعرضًا لنافذتنا السابقة، لنفرض أننا نرغب أن تكون بشكل مربع بطول وعرض ٤٠٠ بكسل، وأن تظهر مبتعدة عن الجانب الأيسر من الشاشة بمقدار ٢٠٠ بكسل ومن أعلى الشاشة بمقدار ٣٠٠ بكسل فتكتب الشفرة على النحو التالي:

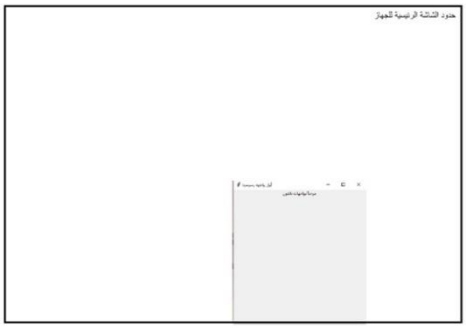
```
main_window.geometry('400x400+200+300')
```

- لنفترض أنك تريد أن تُظهر نافذة برنامجك في وسط الشاشة تمامًا، لفعل ذلك تحتاج أن تحصل على طول وعرض شاشة العرض في جهاز الحاسوب لتحقيق ذلك نستخدم الدالة winfo_screenwidth للحصول على عرض الشاشة، والدالة winfo_screenheight لنحصل على طولها، يتم استدعائها باستخدام النافذة الرئيسية على النحو التالي:

```
screen_width = main_window.winfo_screenwidth()
```

```
screen_height = main_window.winfo_screenheight()
```

- للحصول على إحداثيات نقطة منتصف الشاشة نقسم الطول والعرض على ٢، فنحصل على النقطة (ارتفاع الشاشة/٢ ،عرض الشاشة/٢)، وعند كتابة هذه النقطة في الدالة سيكون موضع ظهور النافذة في الجزء السفلي الأيمن من الشاشة، حيث أن بداية رسم النافذة يكون من النقطة العلوية اليسرى، الشكل التالي يوضح المقصود:



- موضع النافذة في الشاشة
- لذلك سنقوم بوضع منتصف نافذتنا الرئيسية في منتصف شاشة العرض الأساسية فتكون الصياغة العامة للإحداثي السيني على النحو التالي:

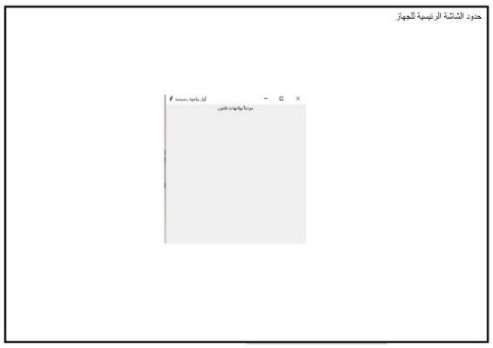
```
main_window.winfo_screenwidth()//2 - main_window_width//2
```

- قمنا باستخدام القسمة الصحيحة للحصول على عددٍ صحيح كامل بلا فواصل، وعلى نفس النهج نحصل على الإحداثي الصادي.
- يجب مراعاة أن المتغيرات التي أنشئت لحساب الإحداثيات عددية النوع بينما الدالة geometry تأخذ سلسلة نصية لذلك أضفنا تنسيق لنحول الأعداد إلى نص. سنُظهر الآن نافذتنا ذات الأبعاد ٤٠٠*٤٠٠ بكسل طولًا وعرضًا في منتصف الشاشة، فتكون الشفرة كاملة على النحو التالي:

```
from tkinter import *

main_window = Tk()
main_window.title("أول واجهة رسومية ")

screen_width = main_window.winfo_screenwidth()
screen_height = main_window.winfo_screenheight()

middle_width= screen_width //2 - 400//2
middle_hight= screen_height//2 - 400//2
main_window.geometry(f'400x400+{middle_width}+{middle_hight}')

L=Label(main_window, text="مرحباً بواجهات بايثون ")
```

```
L.pack()

main_window.mainloop()
```

- ويظهر الشكل التالي نتيجة الشفرة عند تشغيلها:



- موضع النافذة في الشاشة

لتعديل على حجم النافذة

يُسمح للمستخدم بشكل افتراضي أن يقوم بتعديل حجم النافذة (الطول والعرض) باستخدام الفأرة، وتستطيع أن تمنع ذلك باستخدام الدالةresizable(width,height) ، حيث نعطيها قيمة خطأ False لكلٍ من الطول والعرض فيمنع ذلك تعديلهم من المستخدم، تكتب كما يلي:

```
main_window.resizable(False, False)
```

بإمكانك ترك النافذة كما في الإعدادات الإفتراضية قابلة للتعديل ويمكنك أن تضبط .. أقصى طول وعرض تسمح للنافذة أن تتوسع له وأقل طول وعرض يمكن أن تنكمش له باستخدام الدالتين maxsize و minsize على النحو التالي:

```
main_window.minsize(min_width, min_height)
main_window.maxsize(min_height, max_height)
```

تغيير نوع وحجم خط عناصر النافذة

نستطيع تغيير خط العناصر على النافذة الرئيسية باستخدام الدالةoption_add ، ونعطيها نوع الخط وحجم الخط الذي نرغب بعرضه وسيطبق على جميع العناصر التي توضع على النافذة، لنحدد حجم الخط 20 ونوعه Times مثلًا سنكتب السطر التالي:

```
main_window.option_add('*Font', 'Times 20')
```

وتكون النتيجة:

نوع وحجم خط النافذة للحصول على قائمة بأسماء الخطوط المتوفرة بإمكاننا تشغيل الشفرة التالية وسيخرج لنا قائمة بالخطوط:

```
from tkinter import Tk, font
root = Tk()
print(font.families())
```

تحسين دقة النافذة

من شكل النافذة السابق قد تبدو لك جملة (مرحباً بواجهات بايثون) ضبابية قليلًا وأقل وضوحًا من عنوان النافذة، لنحسن دقة عناصر النافذة نضيف لبرنامجنا أسطر الشيفرة التالية:

```
from ctypes import windll
windll.shcore.SetProcessDpiAwareness(1)
```

فنحصل على النتيجة في الشكل التالي:

دقة النافذة

الاطار Frame

نوع من الودجات يُستخدم لدمج ودجات أخرى معًا، ويُستخدم عادةً لتمثيل النافذة ككل، ويمكن دمج إطارات أخرى فيه. جميع الكائنات التالية الذكر تستخدم تقريبا نفس الخصائص (اغلبها اختياري) مثل خصائص اللون (للخلفية والخط)والحجم (الطول والعرض) ونوع الخط والموقع جميع الخصائص تقع تحت بند الأنماط.

الصيغة العامة

```
from tkinter import * ws = Tk() ws.geometry('300x200') ws.title('PythonGuides')

frame = Frame(ws, bg='red', bd=10, relief=FLAT, bd=10, height=300, width=300)
frame.pack()

ws.mainloop()
```

bd : borderwidth

bg: borderwidth

هناك ٥ أنواع من relief

1) Flat 2) sunken 3) Raised 4) Groove 5) Ridge

Grid frames

يتم استخدام الشبكة لوضع عنصر واجهة المستخدم في تنسيق الصف والعمود. الصف والعمود هي الوسيطات اللازمة.

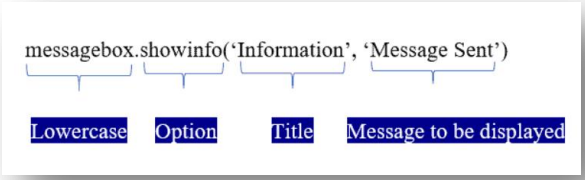
```
frame1 = Frame(ws, padx=5, pady=5)

frame1.grid(row=0, column=1)
```

Tkinter Messagebox

يُستخدم لعرض الرسائل المنبثقة. لبدء استخدام مربع الرسائل ، قم باستيراد صندوق رسائل مكتبة في.Python يوفر Messagebox بشكل أساسي ٦ أنواع من رسائل المطالبة مثل showinfo () و showerror () و showwarning () و askquestion () و askokcancel () و askyesno () و askretyrcancel ().

الصيغة العامة

```
from tkinter import messagebox

messagebox.option('title', 'message_to_be_displayed')
```



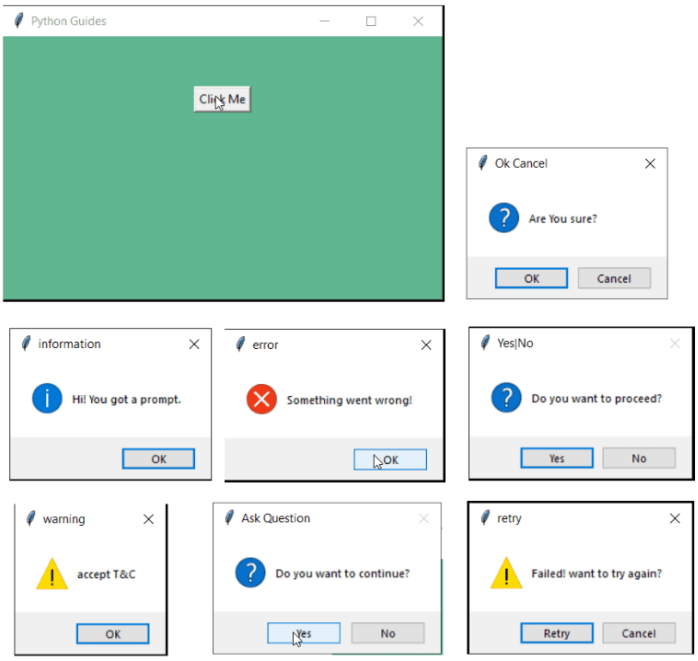
مثال يوضح جميع حالات messagebox

```
from tkinter import*
from tkinter import messagebox
ws = Tk()
ws.title('Python Guides('
ws.geometry('300x200('
ws.config(bg='#5FB691('
def msg1:()
    messagebox.showinfo('information', 'Hi! You got a prompt('.
    messagebox.showerror('error', 'Something went wrong('!
    messagebox.showwarning('warning', 'accept T&C('
    messagebox.askquestion('Ask Question', 'Do you want to continue('?
    messagebox.askokcancel('Ok Cancel', 'Are You sure('?
    messagebox.askyesno('Yes|No', 'Do you want to proceed('?
```

messagebox.askretrycancel('retry', 'Failed! want to try again('?

Button(ws, text='Click Me', command=msg1).pack(pady=50(

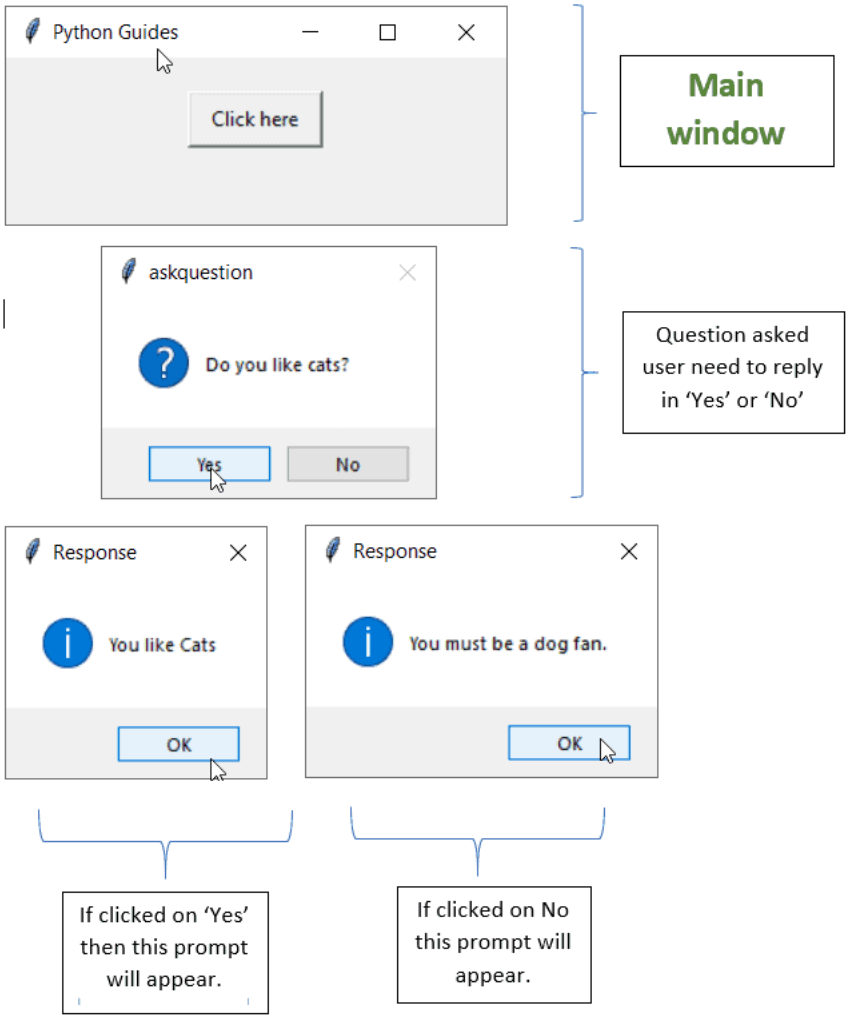
ws.mainloop()



مثال

```
from tkinter import*
from tkinter import messagebox
ws = Tk()
ws.title('Python Guides('
ws.geometry('300x100('

def askMe:()
    res = messagebox.askquestion('askquestion', 'Do you like cats('?
    if res == 'yes:'
        messagebox.showinfo('Response', 'You like Cats('
    elif res == 'no:'
        messagebox.showinfo('Response', 'You must be a dog fan('.
    else:
        messagebox.showwarning('error', 'Something went wrong('!
```

```
Button(ws, text='Click here', padx=10, pady=5, command=askMe).pack(pady=20(
ws.mainloop()
```



Button wedges

يحتوي الزر على وظيفة يتم تشغيلها عند الضغط عليه.

Button(ws, text="any text", command=function/method).pack()

Button style

يُستخدم النمط لتوفير مظهر محسّن للأزرار

from tkinter import * from tkinter.ttk import * ws = Tk() ws.title('PythonGuides') ws.geometry('200x200') st = Style() st.configure('W.TButton', background='#345', foreground='black', font=('Arial', 14)) Button(ws, text='Smash Me', style='W.TButton', command=None).pack() ws.mainloop()

Output

Button position

يوجد ٣ مديرين للتخطيط Pack :، Grid، Place.

Pack : تُستخدم لمحاذاة عنصر واجهة المستخدم في وسط الإطار.

Grid : تستخدم طريقة الصف والعمود لوضع الأدوات.

Place : يتم استخدام لوضع عنصر واجهة المستخدم في أي تنسيق يتم توفيره كـ x & y

Button(ws, text="any text", command=function/method).pack() or grid(row=value, column=value) or place(x=value, y=value)

from tkinter import *

ws = Tk() ws.title("PythonGuide")

ws.geometry('200x250') Button(ws, text="Click", command =None).pack()

or

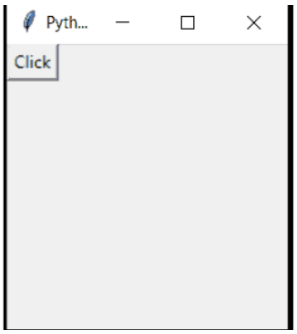
Button(ws,text="Click",command=None).grid(row=0, column=0)

Or

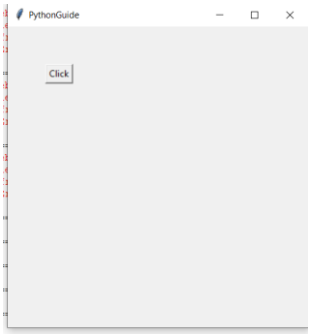
Button(ws, text="Click", command=None).place(x=50, y=50)

ws.mainloop()

Button pack



button grid



button place

خصائص Button

Button(ws, text='Smash Me!', height=10, width=20, bg='#567', fg='White').pack(pady=10)

Button(ws, text="font", font=('arial bold', 18), borderwidth= value, relief ="type ofborder", padx=10, pady=5, justify=CENTE) .pack()

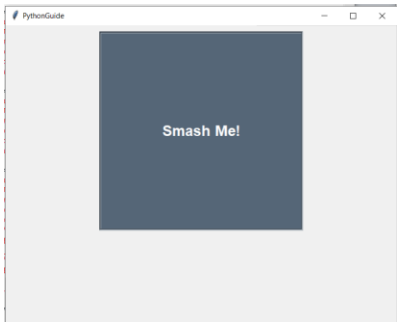
Justify: (center ,left, right)

Padx,pady

يضيف padx مساحة فارغة عموديًا

يضيف pady مساحة فارغة أفقيًا

إذا تم استخدامها داخل الزر ، فسيضيفون مساحة إضافية داخل المربع



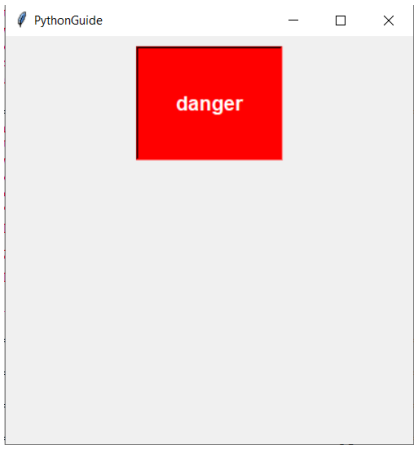
Button(ws, text="Smash Me!", height=10, width=20, bg='#567', fg='White', font=('arial bold', 18), borderwidth =3, relief ="groove", padx=10, pady=5, justify='right').pack(pady=10)

Label

التسمية تعني ببساطة النص على الشاشة. يمكن أن تكون تعليمات أو معلومات. الملصقات هي الأداة المستخدمة على نطاق واسع وهي أمر في جميع أدوات ولغات دعم واجهة المستخدم الرسومية. تُستخدم الملصقات أيضًا لعرض الصور والرموز.

Label(ws, text="any text here", font=('font-name & weight', 'font-size', .place(x=coordinate_value, y=coordinate_value, borderwidth=value, relief="typeofborder", bg="blue", fg="#000", height=5, width=10, padx=10, pady=5, justify=CENTER, anchor=' ')

Anchor =(n:north ,s: south ,e: east ,w: weast ,ne**:** North East, NW : North West, SE:south east, se:south east)



Label(ws, text='danger', bg='red', fg='white', font=('Arial bold', 14), relief='sunken', cursor='pirate', height=4, width=10, padx=10, pady=10, justify= 'center') .pack (padx =10, pady=10, anchor='n')

Tkinter Entry

تستخدم لادخال مدخلات المستخدم من حيث السلاسل أحادية السطر. هو صندوق من سطر واحد حيث يمكن للمستخدم كتابة شيء ما. للسماح بإدخال عدة أسطر ، نقوم بزيادة ارتفاع أداة الإدخال.

هناك طريقتان للادخال البيانات:

1. Entry.insert

- **Insert يستعمل لادخال البيانات او الكلمات**
- **End يحدد أنه يجب إدخال الحرف التالي مباشرة بعد الحرف الأخير**

```
from tkinter import *

 ws = Tk() info

_Tf = Entry(ws) info_Tf.insert(END, 'You email here')

 info_Tf.pack()

ws.mainloop()
```

2. Textvariable يستخدم لتقديم قيمة من خلال متغير, يمكن أن تكون القيمة عددًا صحيحًا أو سلسلة

- **لعدد صحيح: يتم استخدام الكلمة الأساسية IntVar ()**
- **لـ String: يتم استخدام الكلمة الأساسية StringVar ()**

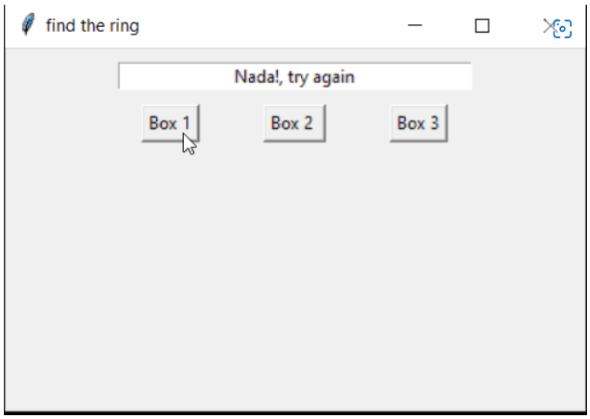
```
from tkinter import *

ws = Tk()

name = StringVar(ws, value='not available')

 nameTf = Entry(ws, textvariable=name).pack()

 ws.mainloop()
```

Entry insert textvariabale

Set text

يتم استخدام نص المجموعة لتعيين النص في مربع الإدخال في Python tkinter. يتم استخدامه مع الوظيفة ويلعب دور وضع النص في مربع الإدخال.

```
from tkinter import *

 def chooseOne(res):

            userInput.delete(0, END)

           userInput.insert(0, res)

           return

 ws = Tk()

ws.title("find the ring")

 ws.geometry("400x250")

frame = Frame(ws)

userInput = Entry(frame, width=40, justify=CENTER) userInput.grid(row=0, columnspan=3, padx=5, pady= 10)

Button(frame,text="Box 1",command=lambda:chooseOne("Nada!, try again")).grid(row=1, column=0)

Button(frame,text="Box 2 ",command=lambda:chooseOne("Great! You found the ring")).grid(row=1, column=1)

Button(frame,text="Box 3",command=lambda:chooseOne("Nada! try again")).grid(row=1, column=2) frame.pack()

 ws.mainloop()
```



Tkinter check button

زر الاختيار هو مربع به علامة تحديد عند النقر فوقه.تختفي علامة الاختيار عند كل نقرة حتى إذا لم يتم تحديدها بالفعل.

الحصول على قيمة من مربع الاختيار checkbutton

يتم استخدام الحصول على القيمة لسحب القيمة من زر الاختيار الذي تم تعيينه أثناء إنشاء زر الاختيار . يمكن أن تكون هذه القيمة عددًا صحيحًا أو سلسلة أو قيمة منطقية.

() IntVar يستخدم للحصول على قيم عدد صحيح

() StringVar يستخدم للحصول على قيم سلسلة

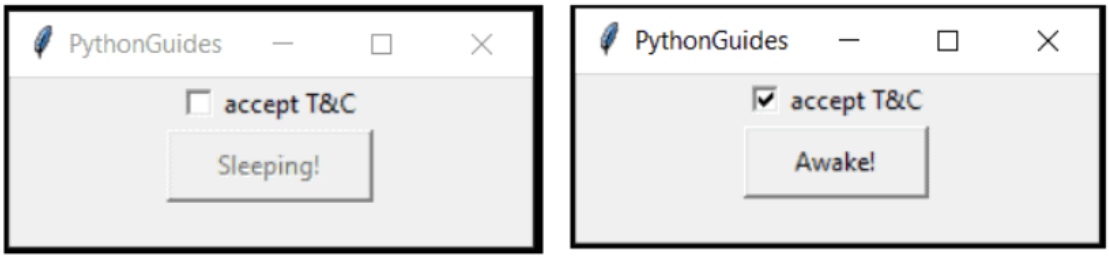
()BooleanVar يستخدم للحصول على قيم منطقية ، مثل True أو False.

```
from tkinter import *

 ws = Tk() ws.title('PythonGuides')

ws.geometry('200x80')

def isChecked():

        if cb.get() == 1: btn['state'] = NORMAL btn.configure(text='Awake!')
```

elif cb.get() == 0: btn['state'] = DISABLED btn.configure(text='Sleeping!')

else: messagebox.showerror('PythonGuides', 'Something went wrong!') cb = IntVar() Checkbutton(ws, text="accept T&C", variable=cb, onvalue=1, offvalue=0, command= isChecked).pack() btn = Button(ws, text='Sleeping!', state=DISABLED, padx=20, pady=5) btn.pack()

ws.mainloop()



Out put

مثال شامل للموضوع (للاطلاع فقط)

from tkinter import *

ws = Tk() ws.title('Email System')

ws.geometry('940x500')

ws.config(bg='#f7ef38') variable = StringVar() gender = ('Male', 'Female', 'Other') variable.set(gender[0])

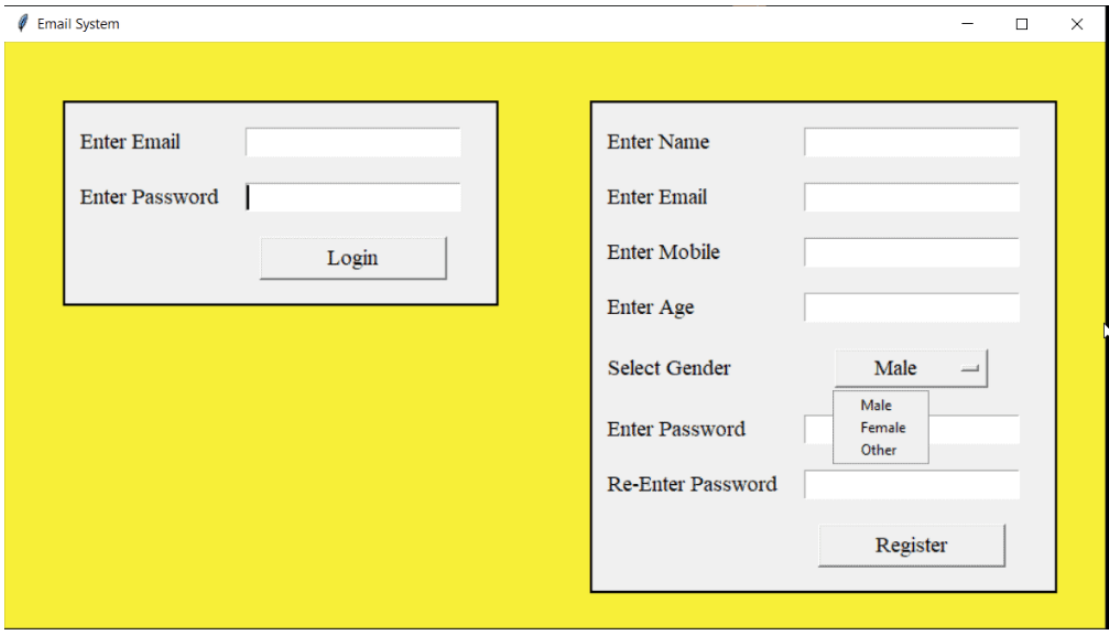
widgets

left_frame = Frame(ws, bd=2, relief=SOLID, padx=10, pady=10)

Label(left_frame, text="Enter Email", font=('Times', 14)).grid(row=0, column=0, sticky=W, pady=10)

```
Label(left_frame, text="Enter Password", font=('Times', 14)).grid(row=1, column=0, pady=10)
log_em = Entry(left_frame, font=('Times', 14))

 log_pw = Entry(left_frame, font=('Times', 14))

login_btn = Button(left_frame, width=15, text='Login', font=('Times', 14), command=None)
right_frame = Frame(ws, bd=2, relief=SOLID, padx=10, pady=10)

Label(right_frame, text="Enter Name", font=('Times', 14)).grid(row=0, column=0, sticky=W,
pady=10)

Label(right_frame, text="Enter Email", font=('Times', 14)).grid(row=1, column=0, sticky=W,
pady=10)

 Label(right_frame, text="Enter Mobile", font=('Times', 14)).grid(row=2, column=0, sticky=W,
pady=10)

 Label(right_frame, text="Enter Age", font=('Times', 14)).grid(row=3, column=0, sticky=W,
pady=10)

 Label(right_frame, text="Select Gender", font=('Times', 14)).grid(row=4, column=0, sticky=W,
pady=10)

Label(right_frame, text="Enter Password", font=('Times', 14)).grid(row=5, column=0, sticky=W,
pady=10)

Label(right_frame, text="Re-Enter Password", font=('Times', 14)).grid(row=6, column=0,
sticky=W, pady=10) reg_na = Entry(right_frame, font=('Times', 14))

 reg_em = Entry(right_frame, font=('Times', 14))

reg_mo = Entry(right_frame, font=('Times', 14))
```

```
reg_ag = Entry(right_frame, font=('Times', 14))

reg_ge = OptionMenu(right_frame, variable, *gender)

reg_ge.config(width=10, font=('Times', 14))

reg_pw = Entry(right_frame, font=('Times', 14))

re_pw = Entry(right_frame, font=('Times', 14))

reg_btn = Button(right_frame, width=15, text='Register', font=('Times', 14), command=None)
# widgets placement

 log_em.grid(row=0, column=1, pady=10, padx=20) log_pw.grid(row=1, column=1, pady=10,
padx=20) login_btn.grid(row=2, column=1, pady=10, padx=20)

left_frame.place(x=50, y=50) reg_na.grid(row=0, column=1, pady=10, padx=20)
reg_em.grid(row=1, column=1, pady=10, padx=20)

 reg_mo.grid(row=2, column=1, pady=10, padx=20) reg_ag.grid(row=3, column=1, pady=10,
padx=20)

 reg_ge.grid(row=4, column=1, pady=10, padx=20) reg_pw.grid(row=5, column=1, pady=10,
padx=20)

re_pw.grid(row=6, column=1, pady=10, padx=20) reg_btn.grid(row=7, column=1, pady=10,
padx=20)

right_frame.place(x=500, y=50)

# infinite loop

 ws.mainloop()
```



مفهوم الـ Module

كلمة **Module** يقال لها **موديول** في العربية, و تعني ملف بايثون عادي يحتوي على مجموعة متغيرات, دوال و كلاسات يمكنك تضمينها في برنامجك. إذاً أي ملف إمتداده py يمثل موديل في بايثون.

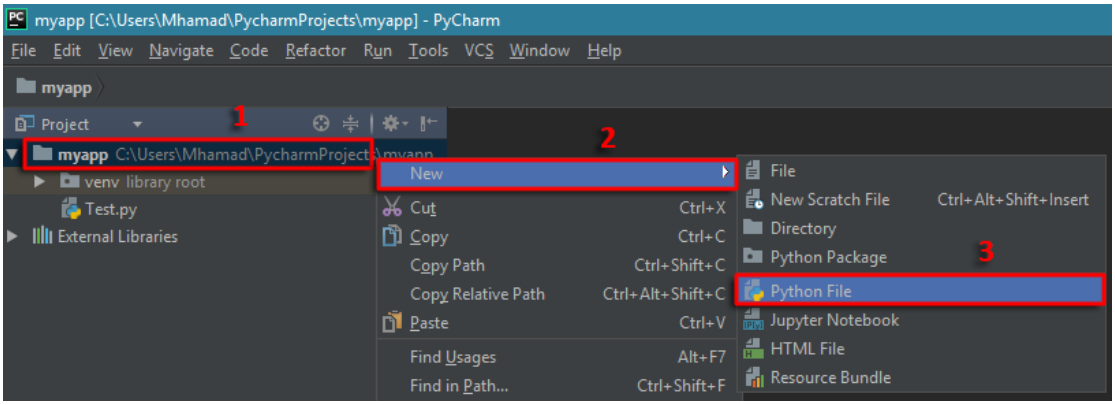
فكرة الـ **Module** هي جعل الكود متاح لإعادة الإستخدام, حيث أن وضع الكود في ملف خاص يتيح لك نقله و إستخدامه في برامجك كلما إحتجت إليه.

بالإضافة ذلك, فإنها تساعدك في تنظيم مشاريعك, فمثلاً في حال كنت تبني برنامج, موقع أو لعبة إلخ.. فإنك ستجد أن تقسيم المشروع الواحد إلى مجموعة **Modules** أمر ضروري جداً لأنه سيسهل عليك كثيراً تطويره و صيانته في المستقبل.

طريقة إنشاء Module

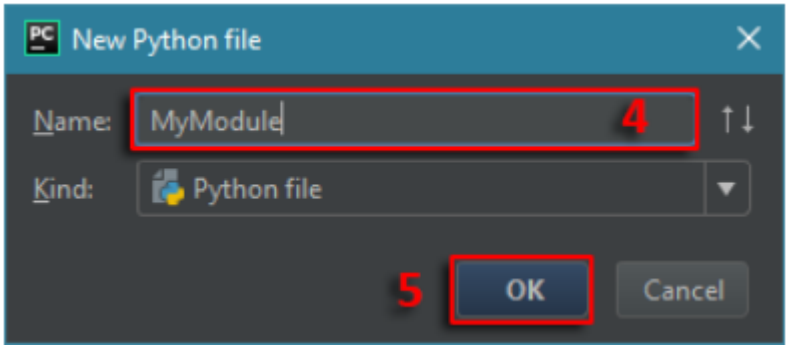
لإنشاء **Module** جديد في أي مشروع إتبع الخطوات التالية:

١. أنقر بزر الماوس الأيمن على إسم المشروع الذي ستنشئ فيه الموديول.
٢. مرر الماوس فوق كلمة **New**.
٣. أنقر على **Python File**.



٤. قم بإعطاء الملف أي إسم مثل MyModule

٥. أنقر **OK** حتى يتم إنشاء ملف بايثون بهذا الإسم في المشروع.



لآن, لاحظ أنه تم إنشاء موديول جديد بداخل نفس المشروع إسمه

مفهوم الكلاس

الكلاس عبارة عن حاوية تستطيع أن تضع بداخلها متغيرات, مصفوفات, دوال إلخ..

لتعريف كلاس جديد نكتب class ثم نضع له إسم, ثم نضع نقطتين.

في المثال التالي قمنا بإنشاء كلاس إسمه my class وضعنا فيه متغير اسمه x

```
class MyClass:
  x = 3
```

مفهوم الكائن

الكائن عبارة عن نسخة من الكلاس. لإنشاء نسخة من كلاس معين, نقوم بتعريف متغير قيمته تساوي إسم الكلاس, ثم نضع قوسين.

```
# هنا قمنا بتعريف كلاس إسمه MyClass
class MyClass:
x = 0
# هنا قمنا بإنشاء كائن من الكلاس MyClass إسمه obj
obj = MyClass()
# هنا قمنا بتغيير قيمة المتغير x الموجود في الكائن obj
obj.x = 10
# هنا قمنا بعرض قيمة المتغير x الموجود في الكائن obj
print('obj.x =', obj.x)
```

سنحصل على النتيجة التالية عند التشغيل.

```
obj.x = 10
```

مفهوم الخصائص

أي متغير تقوم بتعريفه بشكل مباشر في الكلاس يقال له خاصيّة (propriety) السبب في ذلك أن كل نسخة تنشئها من الكلاس ستملك نسختها الخاصة من هذا المتغير.

مثال

```
# هنا قمنا بتعريف كلاس إسمه MyClass
class MyClass:
x = 0
# هنا قمنا بإنشاء كائنين من الكلاس MyClass, الأول إسمه o1 و الثاني إسمه o2
o1 = MyClass()
o2 = MyClass()
# هنا جعلنا قمنا بتغيير قيمة x الموجود في الكائن o1 و قيمة x الموجود في الكائن o2
o1.x = 10
o2.x = 20
# هنا قمنا بعرض قيمة المتغير x الموجود في الكائن o1 و قيمة x الموجود في الكائن o2
print('o1.x =', o1.x)
print('o2.x =', o2.x)
```

سنحصل على النتيجة التالية عند التشغيل.

```
o1.x = 10
o2.x = 20
```

نلاحظ أن كل كائن أنشأناه من الكلاس myclass يملك نسخه خاصه فيه من المتغير x

الكلمة self

وضع الكلمة self كأول باراميتر في الدالة يجعل مفسّر لغة بايثون قادر على الوصول إلى الخصائص الموجودة في نفس الكلاس. أي عند وضع هذه الكلمة كبارامتير في الدالة, تصبح هذه الكلمة بمثابة مؤشر للكلاس نفسه مما يجعلك قادر على الوصول إلى أي شيء تم تعريفه بداخل لكلاس عن طريقها.

في المثال التالي قمنا بإنشاء كلاس إسمه comparator وضعنا فيه دالة اسمها print_max فيها باراميترين فقط (a,b)

```
# Salary هنا قمنا بتعريف كلاس إسمه
class Comparator:
# هنا قمنا بتعريف دالة تأخذ قيمتين عند إستدعائها و a و b. بعدها تطبع القيمة الأكبر بينهما تخزنهما في الباراميترين
def print_max(self, a, b):
if a > b:
print(a, 'is bigger')
elif a < b:
print(b, 'is bigger')
else:
print('They are equal')
# comparator إسمه Comparator هنا قمنا بإنشاء كائن من الكلاس
comparator = Comparator()
# هنا قمنا باستدعاء الدالة print_max() و تمرير قيمتين لها حتى تطبع قيمة العدد الأكبر بينهما
comparator.print_max(2, 6)
```

سنحصل على النتيجة التالية عند التشغيل.

```
6 is bigger
```

التعامل مع الدالة __int__()

يجب ان نقوم بتعريف الدالة مثل أي دالة تقوم بتعريفها. بين أقواس الدالة يمكنك مباشرةً تمرير أسماء الخصائص التي تريد أن وضعها في الكلاس و تريد إعطاءها قيم أولية مباشرةً عند إنشاء كائنات من الكلاس.

في المثال التالي قمنا بإنشاء كلاس إسمه class يحتوي على دالة اسمها __int__ فيها باراميترين فقط (name , age) و بالتالي عند إنشاء كائن من هذا الكلاس, سصبح هذين الباراميترين عبارة عن خصائص لهذا الكائن.

```
# هنا قمنا بتعريف كلاس إسمه Salary
class Person:
# هنا قمنا بتعريف الدالة __init__() و وضعنا فيها باراميترين ( age و name )
# سيتم وضعهما كخصائص في Person لأنهما موضوعين كباراميترات في الدالة __init__() تذكر أن name و age الكلاس
def __init__(self, name, age):
self.name = name
self.age = age
# هنا قمنا بتعريف دالة print_info() مهمتها طباعة قيم خصائص الكلاس Person بشكل مرتب إسمها
def print_info(self):
print('Name:', self.name)
print('Age:', self.age)
print('-----------------')
# هنا قمنا بإنشاء كائنين من الكلاس Person, الأول إسمه p1 و الثاني إسمه p2
p1 = Person('Ahmad', 24)
p2 = Person('Maria', 19)
# هنا قمنا باستدعاء print_info() من الكائنين p1 و p2 حتى تطبع قيم خصائصهم بشكل مرتب الدالة
p1.print_info()
p2.print_info()
```

سنحصل على النتيجة التالية عند التشغيل.

```
Name:                                   Ahmad
Age:                                    24
-----------------
Name:                                   Maria
Age:                                    19
-----------------
```

العلاقة بين الكلاس و الكائن

الفكرة الأساسية من الكلاس هي تجهيز الشكل العام لحفظ البيانات و توفير طرق سهلة الإستخدام للوصول إلى هذه البيانات و التعامل معها بسلاسة.

إذاً الكلاس بطبيعته لا يحفظ أي معلومة, لهذا يقال له نسخة خام (**Blue Print**).

الفكرة الأساسية من الكائن هي إنشاء نسخة مطابقة من الكلاس و إدخال البيانات التي تريد فيها مع إحترام أي شروط موضعة في الكلاس الأساسي.

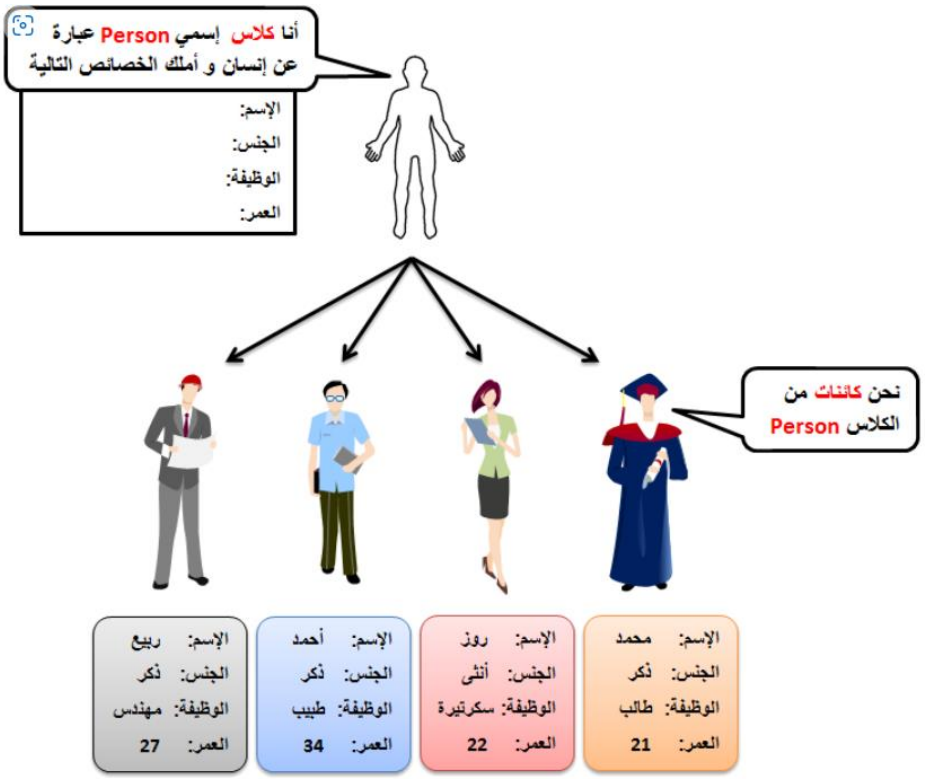
إذاً لا يمكن إنشاء كائن بدون كلاس لأن الكائن بطبيعته يمثل نسخ من كلاس محدد.

فائدة الكلاس

- الخصائص التي يجب أن يمتكلها كل كائن, يتم تعريفها مرة واحدة في الكلاس الأساسي و ليس لكل كائن على حدة.
- إذا أردت إضافة, حذف أو تعديل خصائص الكائنات, نعدّل في الكلاس الأساسي فقط لأن الكائنات هي نسخة مطابقة للكلاس.
- الكلاس يمكن وضعه في ملف خاص و هذا الأمر سيساعدك كثيراً في المستقبل عند ترتيب كود المشروع - الذي قد يتكون من عشرات الكلاسات - بطريقة سهلة المراجعة و التطوير.
- القدرة على تجميع الكائنات و تناقلها دفعة واحدة سواء لتخزينها في قاعدة بيانات أو لنقلها بين شاشة و أخرى الخ..

هذه الأشياء سنشرحها بتفصيل في مستوى متقدم.

الآن, إذا كنت تنوي إنشاء برنامج بسيط لحفظ معلومات عدد غير محدد من الأشخاص. و كل شخص عنده إسم, جنس, عمر و وظيفة. ماذا ستفعل؟!
بكل بساطة الحل هو أن تنشئ كلاس واحد فقط يمثل شخص, و تضع فيه الأشياء الأساسية التي تريدها أن تكون موجودة عند كل شخص. ثم تنشئ منه كائنات قدر ما شئت, و عندها يصبح كل كائن من هذا الكلاس عبارة عن شخص له معلوماته الخاصة كما في الصورة التالية.



كما تلاحظ قمنا بإنشاء كلاس يحتوي على المعلومات الأساسية التي نريد تعبئتها لكل شخص. بعدها قمنا بإنشاء **4** كائنات (أي **4** أشخاص), ثم قمنا بإدخال معلومات خاصة لكل كائن فيهم.

الآن في حال قمت بإضافة أي متغير أو دالة جديدة في الكلاس person فإن أي كائن من هذا الكلاس سيملك نسخة من الشيء الجديد الذي أضفته. و في حال قمت بتعديل كود معين في الكلاس person فأيضاً سيتم تعديل هذا الكود عند جميع الكائنات من هذا الكلاس.

Person.py

```
# هنا قمنا بتعريف كلاس إسمه Person
class Person:
# و وضعنا فيها ٤ بارامترات ( name ,gender ,job و age ) و أعطيناهم None كقيمة إفتراضية
# هنا قمنا بتعريف الدالة __init__()
# سيتم وضعهم كخصائص في Person لأنهم موضوعين كبارامترات في الدالة __init__()
# تذكر أن name و job و gender و age الكلاس
def __init__(self, name=None, gender=None, job=None, age=None):
self.name = name
self.gender = gender
self.job = job
self.age = age
# هنا قمنا بتعريف دالة print_info() مهمتها طباعة قيم خصائص الكلاس Person بشكل مرتب إسمها
```

```
def print_info(self):
print('Name:', self.name)
print('Gender:', self.gender)
print('Job:', self.job)
print('Age:', self.age)
print('-----------------')
```

Test.py

```
# Person هنا قمنا بتضمين الكلاس Person الموجود في الموديول
from Person import Person
# هنا قمنا بإنشاء ٤ Person إسمهم p1, p2, p3 و p4 مع تبديل قيم خصائصهم الإفتراضية كائنات من الكلاس
p1 = Person('Mhamad', 'Male', 'Programmer', 21)
p2 = Person('Rose', 'Female', 'Secretary', 22)
p3 = Person('Ahmad', 'Male', 'Doctor', 34)
p4 = Person('Rabih', 'Male', 'Engineer', 27)
# هنا قمنا print_info() من الكائنات p1, p2, p3 و p4 حتى تطبع قيم خصائصهم بشكل مرتب باستدعاء الدالة
p1.print_info()
p2.print_info()
p3.print_info()
p4.print_info()
```

سنحصل على النتيجة التالية عند تشغيل الموديول test

.

```
Name:                                   Mhamad
Gender:                                 Male
Job:                                    Programmer
Age:                                    21
-----------------
Name:                                   Rose
Gender:                                 Female
Job:                                    Secretary
Age:                                    22
-----------------
Name:                                   Ahmad
Gender:                                 Male
Job:                                    Doctor
Age:                                    34
-----------------
Name:                                   Rabih
Gender:                                 Male
Job:                                    Engineer
Age:                                    27
-----------------
```

التعامل مع التاريخ و الوقت

مقدمة

بايثون تحتوي على أكثر من موديول جاهز للتعامل مع التاريخ, الوقت و التقويم بكل سهولة.

الموديل datetime

الموديل datetime يحتوي على مجموعة كلاسات فيها دوال جاهزة للتعامل مع التاريخ و الوقت.

date : يحتوي على مجموعة دوال خاصة للتعامل مع التاريخ.

time: يحتوي على مجموعة دوال خاصة للتعامل مع الوقت.

datetime: يحتوي على مجموعة دوال خاصة للتعامل مع التاريخ و الوقت.

timedelta: يحتوي على مجموعة دوال خاصة لحساب الفرق بدقة بين تاريخ و آخر.

timezone: يحتوي على مجموعة دوال خاصة لحساب فرق التوقيت بين تاريخ و آخر على حسب المنطقة الزمنية لكل تاريخ.

مثال :

```
# هنا قمنا بتضمين كل محتوى الموديول datetime
import datetime
# هنا قمنا بتخزين التاريخ و الوقت الحالي ككائن من الكلاس datetime في الكائن dt
الذي سترجعه الدالة
dt = datetime.datetime.now()
# هنا قمنا بعرض قيمة الكائن dt
print(dt)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل.

```
2018-12-01 09:13:09.598797
```

الكلاس datetime يحتوي على

class datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=**None**, *, fold=0)

اذا عند انشاء كائن datetime يمكنك مباشرة ان تدخل تاريخ ووقت فيه

فعلياً, أنت مجبر على إدخال قيمة مكان البارميترات year , month , day لأنه لم يتم إعطائهم قيم إفتراضية.

بالنسبة للباراميترات الأخرى فيمكنك تحديد قيمهم الإفتراضية أو عدم تحديدها لأنه تم إعطائهم قيم إفتراضية.

القيم التي يمكنك تمريرها للبارميترات هي التالية:

مثال

```
# هنا قمنا بتضمين كل محتوى الموديول datetime
import datetime
# هنا قمنا بإنشاء كائن من الكلاس datetime يمثل تاريخ محدد و قمنا بتخزينه في الكائن dt
dt = datetime.datetime(2012, 4, 5)
# هنا قمنا بعرض قيمة الكائن dt
print(dt)
```

سنحصل على نتيجة تشبه النتيجة التالية عند التشغيل.

```
2012-04-05 00:00:00
```

التعامل مع الملفات

معالجة الملفات

التعامل مع الملفات أو معالجة الملفات (**Files Handling**) يقصد منها إجراء عملية ما على الملفات على مختلف انواعها (مثل **txt** - **jpg** - **mp4**).

الدالة ( ) open

هذه الدالة هي من الدوال الجاهزة في بايثون و هي تستخدم لإنشاء ملف جديد أو لفتح الملف الذي سيتم التعامل معه.

في حال تم إنشاء الملف بشكل صحيح أو تم فتح الملف بشكل صحيح ترجع file يتيح لك التعامل معه. في حال لم تستطع إنشاء الملف أو الوصول إليه ترمي إستثناء.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None)
```

file نمرر نص يمثل إسم الملف الذي سيتم إنشاؤه أو التعامل معه.

mode هو باراميتر إختياري, نمرر مكانه حرف)أو أكثر (يمثل كيف سنتعامل مع الملف, مثل: هل تنوي القراءة منه أو الكتابة فيها إلخ..

Buffering هو باراميتر إختياري, يمكنك أن تمرر مكانه رقم يحدد كيف سيتم تخزين الأحرف بشكل مؤقت في الذاكرة أثناء الكتابة أو القراءة من الملف.

Encoding هو باراميتر إختياري, يمكنك أن تمرر مكانه إسم الترميز الذي يجب استخدامه عند التعامل مع الملف.

error هو باراميتر إختياري, يمكنك أن تمرر مكانه كلمة لتحدد كيف سيتم التعامل مع الأخطاء التي قد تحدث عند التعامل مع الملف.

newline و باراميتر إختياري, يمكنك أن تمرر مكانه الرمز الذي يمثل نهاية كل سطر في الملف و الذي يجعل النص الذي يوضع بعضه ينزل على سطر جديد.

أهم بارامتر إختياري في هذه الدالة هو البارامتر mode لأنه كما سبق و قلنا أن الحرف الذي نمرره مكانه يحدد الهدف من فتح الملف.

في الجدول التالي وضعنا كل الحروف التي يمكن تمريرها مكان هذا البارامتر.

الحرف	معناه
'r'	يعتبر إختصار للكلمة **Read**, و هو يستخدم لفتح الملف من أجل القراءة منه. كما أنه الـ **Mode** الإفتراضي للملف الذي تفتحه.
'w'	يعتبر إختصار للكلمة **Write**, و هو يستخدم لفتح الملف من أجل الكتابة فيه. و في حال لم يكن الملف المراد الكتابة فيه موجوداً أصلاً, سيتم إنشاؤه. **ملاحظة:** هذا الحرف يقوم بحذف النص الذي كان موجوداً في الملف في حال لم يكن فارغاً.
'a'	يعتبر إختصار للكلمة **Append**, و هو يستخدم لفتح الملف من أجل الكتابة في آخره. أي لإضافة نص جديد على النص الموجود في الملف. و في حال لم يكن الملف المراد الكتابة فيه موجوداً أصلاً, سيتم إنشاؤه.
'x'	يعتبر إختصار للكلمة **Create**, و هو يستخدم لإنشاء ملف جديد فقط في حال لم يكن موجوداً.
't'	يعتبر إختصار للكلمة **Text**, و هو يستخدم لتحديد أن محتوى الملف عبارة عن نص عادي. كما أنه الـ **Mode** الإفتراضي للملف الذي تفتحه.
'b'	يعتبر إختصار للكلمة **Binary**, و هو يستخدم لتحديد أن محتوى الملف عبارة عن **Binary**, أي أحرف لا يمكن أن يفهمها الإنسان العادي. هذا الـ **Mode** نستخدمه عند التعامل مع الملفات الغير نصية مثل الصور, الفيدوهات, التسجيلات الصوتية إلخ.. **ملاحظة:** المثال سيعلمك طريقة إنشاء نسخة من أي ملف.
'+'	يعتبر إختصار للكلمتين **Read & Write**, و هو يستخدم لفتح الملف مع إمكانية القراءة منه و الكتابة فيه في نفس الوقت.

الأحرف المذكورة في الجدول يمكن دمجها مع بعضها, أي يمكنك تحديد أكثر من Modeفي وقت واحد.
فمثلاً يمكنك كتابة 'wb' من أجل فتح ملف جديد و وضع فيه نص نوعه Binaryكما نفعل في حال أردنا نسخ صورة على سبيل المثال.

مثال بإنشاء ملف نصي جديد إسمه demo.txt في نفس البرنامج وبعدها قمنا بكتابة سطر داخله
python is an easy language to learn

```
# هنا قمنا 'demo.txt' و وضعنا الرمز 'w' لكي يتم إنشاء الملف و لنستطيع الكتابة فيه أيضاً
بإنشاء كائن يشير لملف إسمه
```

```
opened_file = open('demo.txt', 'w')
# هنا قمنا باستدعاء الدالة write() من الكائن opened_file للكتابة في الملف الذي يشير إليه
opened_file.write('Python is an easy language to learn.')
# هنا قمنا  close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة باستدعاء الدالة
opened_file.close()
```

بعد تشغيل الملف test.py سيتم إنشاء ملف إسمه demo.txt في نفس المشروع الذي نعمل فيه و بداخله النص التالي.

Python is an easy language to learn.

دوال القراءة والكتابة في الملفات

	إسم الدالة مع تعريفها
1	`write(string)` تستخدم للكتابة في الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها. مكان البارامتر `string` نمرر النص الذي نريد أن يتم كتابته في الملف. شاهد المثال »
2	`writelines(aList)` تستخدم لكتابة مجموعة نصوص مخزنة في `list` في الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها. مكان البارامتر `lines` نمرر كائن `aList` فيه مجموعة النصوص التي نريد أن يتم كتابتها بنفس الترتيب في الملف. شاهد المثال »
3	`read(n = -1)` تستخدم للقراءة من الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها. إذا قمت باستدعائها و لم تمرر لها أي رقم, سترجع كل النص الموجود في الملف دفعة واحد. `n` هو بارامیتر إختياري يمكنك أن تمرر مكانه رقم يمثل عدد الأحرف التي تريد قراءتها من الملف في حال لم ترد أن تقرأ كل محتوى الملف دفعة واحدة. مع الإشارة إلى أنك في كل مرة تقوم فيها باستدعائها ستعطيك الأحرف التالية الموجودة في الملف. شاهد المثال »
4	`readline(limits = -1)` تستخدم للقراءة سطراً سطراً من الكائن الذي يمثل الملف المفتوح الذي قام باستدعائها. إذا قمت باستدعائها و لم تمرر لها أي رقم, سترجع السطر التالي الموجود في الملف. `n` هو بارامیتر إختياري يمكنك أن تمرر مكانه رقم يمثل عدد الأحرف التي تريد قراءتها من السطر التالي في الملف في حال لم ترد أن تقرأ كل محتوى السطر دفعة واحدة. مع الإشارة إلى أنك في كل مرة تقوم فيها باستدعائها ستعطيك الأحرف الموجودة حتى نهاية السطر الحالي في الملف.
8	`close()` تستخدم لإغلاق الإتصال مع الملف و تنظيف الذاكرة من كل ما له علاقة بهذا الملف. **ملاحظة:** في حال قمت بفتح الملف بالأساس بواسطة الجملة `with` فلا داعي لإغلاق الملف لأنها تقوم بإغلاقه بشكل تلقائي عنك. شاهد المثال »

مثال١:

```
# و وضعنا الرمز 'w' للإشارة إلى أننا سنستخدم هذا الكائن لكتابة نص جديد في الملف 'demo.txt' هنا قمنا بإنشاء كائن يشير لملف إسمه
opened_file = open('demo.txt', 'w')
# هنا قمنا باستدعاء write() من الكائن opened_file لكتابة نص جديد في الملف الذي يشير إليه الدالة
opened_file.write('This new text will replace the old text.')
# هنا قمنا close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة باستدعاء الدالة
opened_file.close()
```

بعد تشغيل الملف test.py سيتم إنشاء ملف إسمه demo.txt

في نفس المشروع الذي نعمل فيه و بداخله النص التالي.

This new text will replace the old text.

مثال ٢

```
# و وضعنا الرمز 'r' للإشارة إلى أننا سنستخدم هذا الكائن لقراءة النص الموجود في الملف 'demo.txt' هنا قمنا بإنشاء كائن يشير لملف إسمه
opened_file = open('demo.txt', 'r')
# الذي يشير للملف المفتوح حتى ترجع كل النص الموجود فيه, بعدها قمنا بطباعة النص الذي سترجعه opened_file من الكائن read() هنا قمنا باستدعاء الدالة
print(opened_file.read())
# هنا قمنا close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة باستدعاء الدالة
opened_file.close()
```

بعد تشغيل الملف test.py سيتم طباعة كل النص الموجود في الملف demo.txt

الذي إفترضنا أننا أنشأناه في نفس المشروع الذي نعمل فيه.

Python is an easy language to learn.

--> هنا افترضنا أن هذا النص كان موجوداً أصلاً في الملف

مثال ٣

```
# و وضعنا الرمز 'r' للإشارة إلى أننا سنستخدم هذا الكائن لقراءة النص الموجود في الملف 'demo.txt' هنا قمنا بإنشاء كائن يشير لملف إسمه
opened_file = open('demo.txt', 'r')
# الذي يشير للملف المفتوح حتى ترجع كل النص الموجود فيه, بعدها قمنا بطباعة النص الذي ستررجعه opened_file من الكائن read() هنا قمنا باستدعاء الدالة
print(opened_file.read())
# هنا قمنا close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة باستدعاء الدالة
opened_file.close()
```

بعد تشغيل الملف Test.py سيتم طباعة كل النص الموجود في الملف demo.txt

الذي إفترضنا أننا أنشأناه في نفس المشروع الذي نعمل فيه.

```
Python is an easy language to learn.
```

<-- هنا افترضنا أن هذا النص كان موجوداً أصلاً في الملف

تخزين احرف عربية

إذا حاولت كتابة أحرف عربية في ملف و لم تحدد أن نوع الترميز المستخدم عند التعامل مع الملف هو **utf-8** سيظهر أمامك الخطأ ('charmap' codec can't encode)

و إذا حاولت قراءة نص عربي موجود في ملف و لم تحدد نوع الترميز ستلاحظ أن النص يظهر بشكل غير مفهوم كالتالي (Ø§Ù„Ø³Ù„Ø§Ù… Ø¹Ù„ÙŠÙƒÙ…)

مثال

في المثال التالي قمنا بإنشاء ملف إسمه ARABIC.txt قمنا بتخزين نص عربي فيه و بعدها قمنا بقراءة النص الموجود فيه.

```
# يشير لملف جديد إسمه 'arabic.txt' سيتم إنشاؤه في نفس المشروع بجانب الملف 'Test.py' هنا قمنا بإنشاء كائن إسمه opened_file
# حتى يتم إنشاء الملف و يكون لدينا القدرة على 'utf-8' لنستطيع التعامل مع الأحرف العربية وضعنا الرمز 'w+' الكتابة و القراءة منه في نفس الوقت. كما أننا حددنا نوع الترميز هو
opened_file = open('arabic.txt', 'w+', encoding='utf-8')
# من opened_file لكتابة نص جديد في الملف الذي يشير إليه و لاحظ أننا أدخلنا نص عربي هنا قمنا باستدعاء الدالة write() الكائن
```

```
opened_file.write('السلام عليكم و رحمة الله و بركاته')
# هنا قمنا باستدعاء الدالة seek() للعودة لأول الملف
opened_file.seek(0,0)
# الذي يشير للملف المفتوح حتى ترجع كل النص الموجود فيه, بعدها قمنا بطباعة النص الذي سترجعه opened_file هنا قمنا باستدعاء الدالة read() من الكائن
print(opened_file.read())
# هنا قمنا close() من الكائن opened_file لإغلاق الإتصال مع الملف المفتوح في الذاكرة باستدعاء الدالة
opened_file.close()
```

بعد تشغيل الملف Test.py سيتم إنشاء الملف arabic.txt و تخزين النص العربي بداخله. أيضاً سيتم طباعة النص العربي الموجود في الأساس بداخل الملف كالتالي.

السلام عليكم و رحمة الله و بركاته

مدير الحزم pip

مفهوم الـ PIP

في أي مشروع تبنيه بلغة بايثون ستقوم في الغالب باستخدام كلاسات و دوال جاهزة قام ببنائها مطورون آخرون حتى لا تضيع وقتك في بناء كل شيء من الصفر. في هذا الموضوع سنتعلم كيف نقوم بتضمين أكواد جاهزة في المشاريع بكل سهولة بالإعتماد على أداة إسمها **PIP** بالإضافة إلى فعل ذلك بشكل مباشر من برنامج **PyCharm**.

الآن, بما أنك شخص واحد من ضمن ملايين الأشخاص الذين قرروا إستخدام لغة بايثون في بناء مشاريعهم, فهذا يعني أن مطورين كثر قاموا حتماً ببناء أكواد جاهزة و نشرها مجاناً حتى يستفيد منها باقي المطورين مع الإستمرار في تحديث هذه الأكواد حتى تظل تتوافق مع التحديثات المستمرة للغة بايثون و لمعالجة أي ثغرات في هذه الأكواد.

المطور الذي يعمل بطريقة إحترافية, يقوم في العادة ببناء الموديول بنفس الطريقة التي تعلمتها سابقاً. أي يقوم بإنشاء ملف بايثون و بداخله يقوم بتعريف الكلاسات و الدوال إلخ.. و طبعاً لا شيء يمنع المطور من أن يقوم بإنشاء أكثر من موديول, أو أن يقوم بتضمين موديول آخر في الموديول الذي يقوم بتطويره.

عندما يقوم المطور بتجميع الملفات التي يعدّها في مجلد واحد لجعلها قابلة للتحميل و التضمين بسهولة فإن هذا المجلد يقال له حزمة (**Package**).

الأداة **PIP** تساعدك في تحميل و تضمين أي حزمة تريد في مشاريعك بكل سهولة بدل أن تضطر إلى أن تبحث عنهم بنفسك في النت.

ملاحظة :

إبتداءآ من الإصدار **3.4** من لغة بايثون, أصبحت الأداة **PIP** يتم تنصيبها عنك بشكل تلقائي أثناء تنصيب لغة بايثون.أي إذا كنت تستخدم هذا الإصدار أو إصدار أعلى من لغة بايثون, فلا حاجة لأن تقوم بتحميلها, بل و بإمكانك إستخدامها فوراً.

في حال كنت تستخدم إصدار قديم من لغة بايثون, فهذا هو الموقع الرسمي pypa.io لتحميل هذه الأداة.

في هذا الموقع أيضاً ستجد الخطوات التي يجب إتباعها لتحميلها و تنصيبها بشكل صحيح على الحاسوب.

الموقع الرسمي لتحميل الحزم

في هذا الموقع pypi.org تجد جميع الحزم التي يمكنك تنصيبها بواسطة الأداة **PIP**.

طريقة معرفة ما إن كانت الأداة PIP منصبة على الحاسوب

للتأكد ما إن كانت الأداة **PIP** منصبة على الحاسوب يمكنك محاولة عرض إصدار النسخة المنصبة منها كالتالي:

١. قم بفتح موجه الاوامر.
٢. أكتب الأمر python -m pip –version
٣. ثم انقر على enter

تمت بعون الله

تمنياتي لكم بالتوفيق