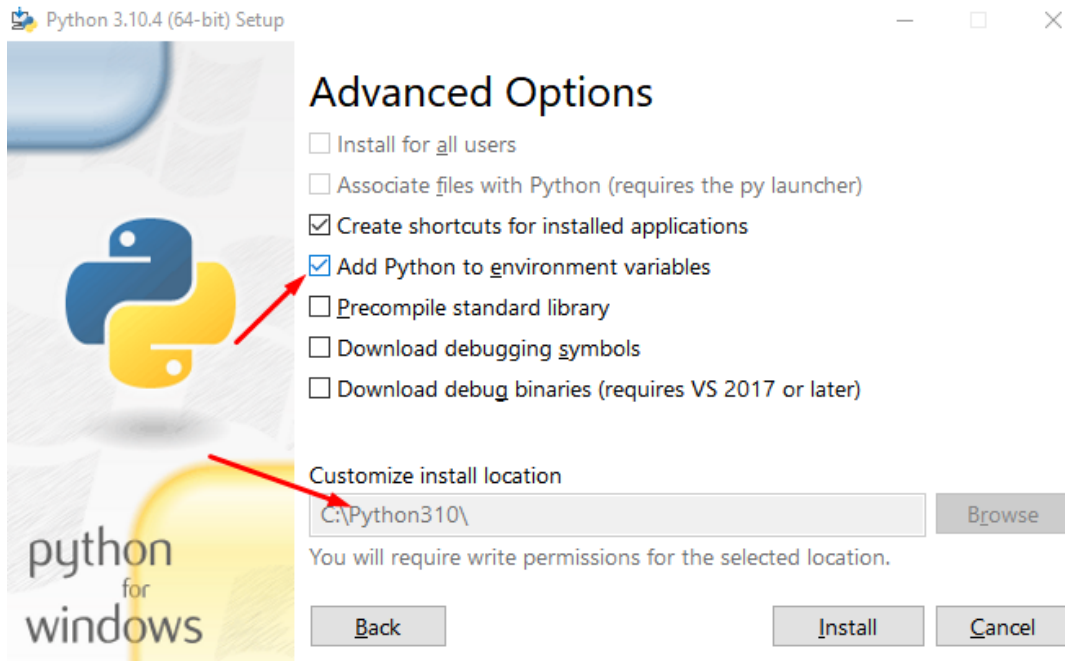


نوتة 1 أساسيات البايثون

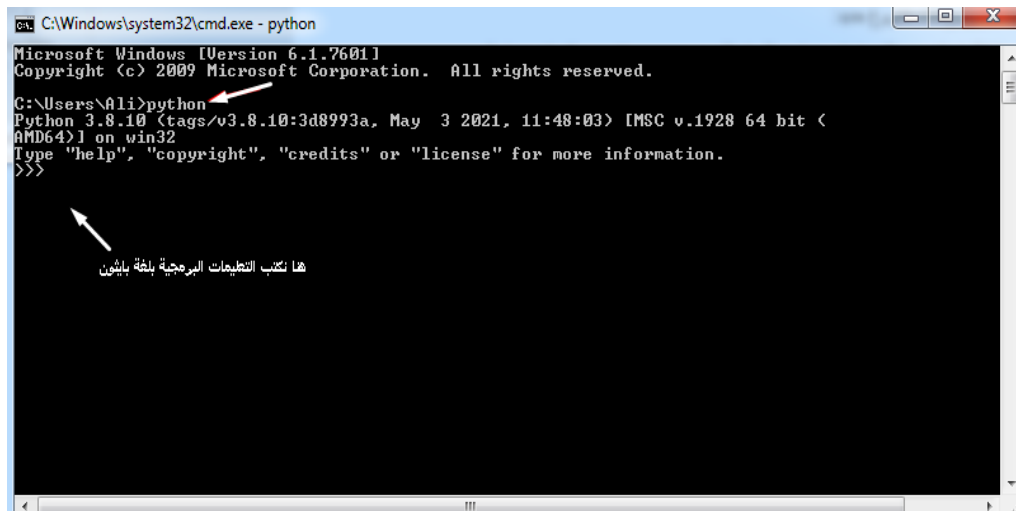
مقدمة لشرح وتثبيت بيئة بايثون وإنشاء مشروع جديد:

الشرح سيتم على التجميعية الأساسية python 3.10 والتي يتم تنصيبها من الموقع الرسمي python.org، يمكن تنصيبها على مختلف أنظمة التشغيل كون الكود في لغة بايثون لا يتم ترجمته مباشرة إلى لغة الآلة. (يمكن استخدام النسخة 3.8 أو إصدار أعلى..)

ملاحظة: أثناء التنصيب يجب اختيار add python to environment variables لكي يتمكن نظام التشغيل من التعرف على الملف التنفيذي للبايثون وبالتالي إمكانية تشغيل البايثون من موجه الأوامر.



- للتأكد أنه تم التنصيب بنجاح ننتقل إلى موجه الأوامر ونكتب python ثم نضغط enter فتظهر نسخة البايثون.



- لتنفيذ ملف بايثون اسمه hello.py (امتداد ملف البايثون هو py) موجود في مجلد اسمه project في القرص E باستخدام موجه الأوامر نكتب الصيغة [python E:\project\hello.py]

```
C:\Users\Ali>python E:/project/hello.py
hello world
```

- يمكن كتابة التعليمات بشكل مباشر وتنفيذها في موجه الأوامر أو في windows powershell وذلك بكتابة python في موجه الأوامر فيظهر رقم البايثون والبادئة >>> التي يمكن الكتابة بعدها وكل تعليمة تنفذ فور كتابتها والضغط على enter.
- كما يمكن التعامل مع بيئة IDLE التي تأتي افتراضياً مع البايثون يمكن الكتابة بلغة بايثون ضمنها. توجد بيئات تطوير شاملة يتم التطوير باستخدام بايثون من خلالها مثل pycharm وغيرها، وبيئات تفاعلية مثل jupyter notebook.
- يتم تنزيل jupyter notebook عن طريق كتابة السطر pip install jupyter وبعد الانتهاء من التنزيل يتم تشغيل الـ jupyter عن طريق كتابة jupyter notebook في موجه الأوامر حيث يتم انشاء web server ويتنصت على بورت معين وبعدها يتم فتح متصفح الويب لتظهر عليه واجهة الـ jupyter.

ملاحظات:

- يتم تنزيل أي مكتبة في بايثون عبر كتابة التعليمة التالية في موجه الأوامر
اسم المكتبة
pip install
- يتم تحديث مكتبة معينة في بايثون عن طريق كتابة التعليمة
اسم المكتبة
pip install --upgrade

- في بعض الحالات يجب علينا ترقية حزمة الـ pip المستخدمة لتثبيت المكتبات إلى إصدار أحدث فعندها نقوم بكتابة التعليمة `python -m pip install --upgrade pip` ضمن موجه الأوامر.
- البايثون يعمل على مختلف أنظمة التشغيل لأن الكود لا يترجم مباشرة إلى لغة الآلة بل يترجم في البداية إلى لغة وسيطية `python Byte-Code` وبعدها يقوم `Python Virtual Machine (PVM)` بتفسير الـ `Byte-Code` إلى لغة الآلة حسب نظام التشغيل.
- توجد توزيعية `Jython` التي تختلف عن `Python` فقط بوجود `JVM (Java Virtual Machine)` وذلك للاستفادة من البلوكات المكتوبة بلغة `Java` ضمن الكود المكتوب بلغة `Python`.
- يتم استخدام التعليمة `pip list` في موجه الأوامر لعرض الحزم أو المكتبات المثبتة، والتعليمة `pip list > aa.txt` لحفظ أسماء الحزم المثبتة وإصداراتها في ملف اسمه `aa.txt` بصيغة `aa.txt`.
- لبناء تطبيق بايثون نقوم بإنشاء مجلد جديد لهذا التطبيق ونضع فيه الحزم اللازمة لهذا التطبيق فقط والمفسر الخاص بهذا التطبيق والإصدارات الخاصة بهذا التطبيق وكذلك لبناء تطبيق آخر سيكون له مجلده الخاص الذي يحوي حزمه الخاصة أيضاً.... وللقيام بهذا الشيء نحتاج إلى أداة تدعى `virtual env` نقوم بتثبيتها باستخدام التعليمة `pip install virtualenv`.

مثال: إنشاء مجلد للمشروع في القرص `C` باستخدام موجه الأوامر.

نفتح موجه الأوامر ونكتب `cd..` ثم نضغط `enter` ثم `cd..` ثم `enter` فيصبح المسار في موجه الأوامر `C:\>` بعدها نكتب `mkdir project` لإنشاء مجلد مشروع اسمه `project` بعدها نفتح هذا المجلد بكتابة التعليمة `cd project` ثم `enter` فيصبح المسار `C:\project>`

```

C:\Windows\system32\cmd.exe
C:\Users\Ali>pip list > a.txt
C:\Users\Ali>cd..
C:\Users>cd..
C:\>mkdir project
C:\>cd project
C:\project>

```

كان يمكن إنشاء مجلد بالطريقة العادية ثم كتابة مسار المجلد في موجه الأوامر بعد الأمر `cd`

```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Ali>cd C:\project
C:\project>_

```

- نقوم بإنشاء بيئة افتراضية ضمن هذا المجلد وذلك بكتابة التعليمة
`python -m virtualenv plenv` وبالتالي يصبح لدينا في المجلد `project` بيئة افتراضية
اسمها `plenv` تحوي الحزم الخاصة بهذا التطبيق او المشروع والتي تدعى (site package).

```

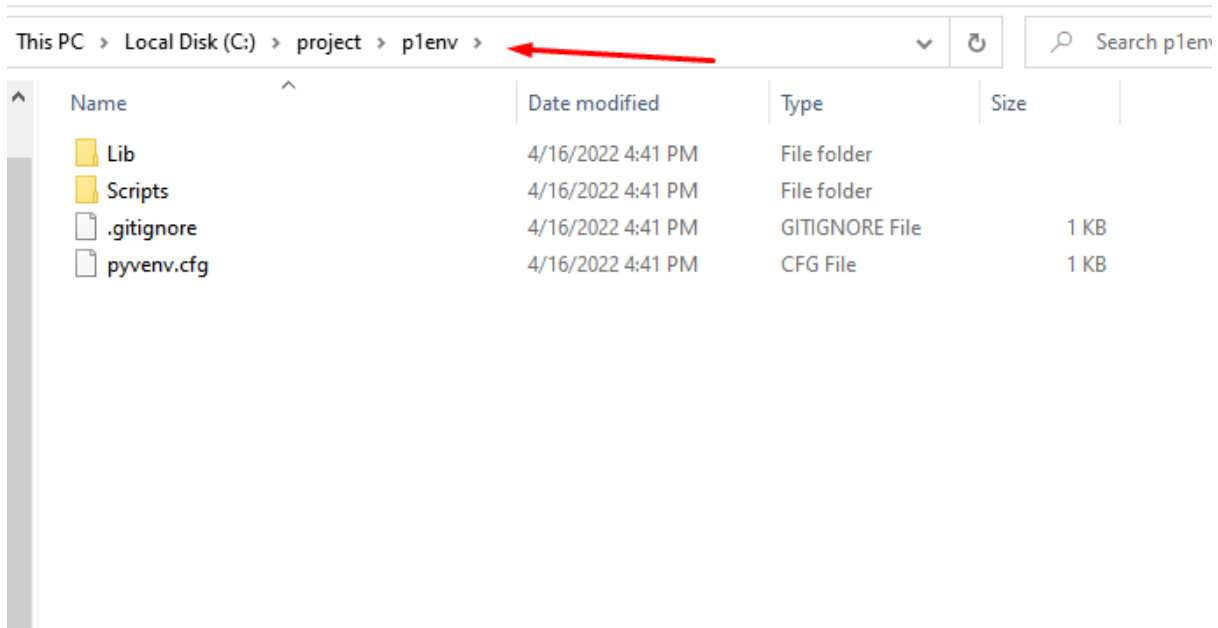
C:\Users\Ali>cd ..
C:\Users>cd ..
C:\>mkdir project
C:\>cd project
C:\project>python -m virtualenv plenv
created virtual environment CPython3.10.4.final.0-64 in 842ms
creator CPython3Windows(dest=C:\project\plenv, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\
pData\Local\pypa\virtualenv)
added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\project>dir
Volume in drive C has no label.
Volume Serial Number is CE0D-BAED

Directory of C:\project

04/16/2022  04:41 PM  <DIR>          .
04/16/2022  04:41 PM  <DIR>          ..
04/16/2022  04:41 PM  <DIR>          plenv
               0 File(s)                0 bytes
               3 Dir(s) 66,144,243,712 bytes free

```



- لتفعيل هذه البيئة (أي يصبح عملنا على هذه البيئة وليس على البيئة الأساسية وبالتالي أي حزمة أو مكتبة نقوم بتثبيتها تكون مرئية ضمن هذه البيئة الافتراضية فقط) نقوم بتشغيل الملف activate.bat الموجود في المجلد Scripts وذلك بكتابة مساره في موجه الأوامر C:\project\p1env\scripts\activate.bat ثم نضغط enter فيتم تفعيل البيئة الافتراضية ويظهر على نافذة موجه الأوامر قبل المسار اسم هذه البيئة على الشكل التالي c:\.....\p1env) وبعدها نقوم بتنصيب الحزم وكتابة الكود الخاص بهذا المشروع فقط.

- عند الانتهاء نقوم بإلغاء تفعيل هذه البيئة والعودة للعمل ضمن البيئة الأساسية عن طريق تشغيل الملف deactivate.bat بنفس الطريقة deactivate.bat C:\project\p1env\scripts\

```
Microsoft Windows [Version 10.0.19044.1620]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ali>C:\project\p1env\scripts\activate.bat

(p1env) C:\Users\Ali>pip list
Package      Version
-----
pip          22.0.4
setuptools   62.1.0
wheel        0.37.1

المكتبات الموجودة ضمن البيئة الافتراضية التي قمنا بإنشائها

(p1env) C:\Users\Ali>C:\project\p1env\scripts\deactivate.bat
C:\Users\Ali>
C:\Users\Ali>pip list
Package      Version
-----
abs1-py      1.0.0
argon2-cffi  21.3.0
argon2-cffi-bindings  21.2.0
asttokens    2.0.5
astunparse   1.6.3
attrs        21.4.0
backcall     0.2.0
beautifulsoup4  4.10.0
bleach       4.1.0
cachetools   5.0.0
certifi      2021.10.8
cffi         1.15.0
charset-normalizer  2.0.12

المكتبات الموجودة ضمن بيئة بايثون الاساسية
```

أساسيات البرمجة بلغة بايثون:

لغة بايثون هي أبسط لغة برمجة وتتبع هذه البساطة من الفلسفة الخاصة لمخترعها ويمكن عرض الأفكار الخاص به عن طريق كتابة التعليمة `import this`.

```
Command Prompt - python
Microsoft Windows [Version 10.0.19044.1620]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ali>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

تسهل بايثون الكثير بالتعامل مع البرمجة بشكل عام ومع برمجة الويب والشبكات بشكل خاص حيث تمكن التعامل مع الشبكات بعدة طبقات من النموذج المرجعي وتأمين تجريد بمستوى عالٍ لأغلب الاحتياجات في عالم الشبكات وتأمين الوصول والتخاطب مع الطبقات الدنيا (طبقة الداتا لينك والطبقة الفيزيائية) وتعمل مع مختلف البروتوكولات (UDP, TCP, MTP, FTP,).

تدعم بايثون البرمجة غرضية التوجه Object Oriented.

لا تحتوي على أقواس البلوكات {...} حيث التعليمات التي تنتمي لبلوك محدد تبعد نفس البعد عن بداية البلوك ولا تحتوي فواصل منقوطة.

مثال:

```
In [2]: s="welcome to python"
for i in s:
    i=i*2
    if i=='d':
        i=i*4
        print('level 3')
    print('level 2')
print('level 1')
```

يوجد لدينا بلوكين متداخلين هما بلوك الـ `if` ضمن بلوك الـ `for` حيث يعبر عن كل بلوك بإزاحة بمقدار 4 فراغات أو بالضغط على زر الـ `TAB`، حسب المثال يعبر الخط المنقط عن البلوك حيث أول بلوك هو بلوك حلقة `for` وجميع التعليمات التي تبعد مسافة 4 فراغات بعد تعليمة الـ `for` تنتمي إلى هذه الحلقة وبالتالي تعليمة `if` تنتمي إلى بلوك `for` وتقع تعليماتها على بعد 4 فراغات من بداية تعليمة `if` أي على بعد 8 فراغات من بداية `for` وهكذا ...

البايثون حساسة لحالة الأحرف، على سبيل المثال إذا أردنا استخدام التعبير المنطقي `True` ضمن الكود فيجب كتابتها بالشكل `True` وليس `....true`.

المعاملات الحسابية في بايثون شبيهة باللغات البرمجية الأخرى مع اختلاف في القوة (الأس) حيث نستخدم المعامل `**` وأيضاً معامل القسمة / يعطي ناتج من نوع `float` و // يعطي ناتج من نوع `int`.

أولويات العمليات الحسابية: أقواس — الرفع إلى قوة — ضرب، قسمة، باقي قسمة — جمع وطرح. ويجب الأخذ بالحسبان الترتيب من اليسار إلى اليمين عند تساوي الأولويات.

المعاملات المنطقية هي (أكبر >) و (أصغر <) و (أكبر أو يساوي >=) و (أصغر أو يساوي <=) و (يساوي ==) و (لا يساوي !=) وهذا المعاملات تعيد قيمتين فقط هما إما `True` أو `False`.

```
In [65]: print(5>3)
True
```

```
In [66]: print(5==3)
False
```

مثال:

العمليات الحسابية

```
In [11]: print('1+1=',1+1)
          print('2-5=',2-5)
          print('10*3=',10*3)
          print('5/2=',5/2)
          print('5//2=',5//2)
          print('4/2=',4/2)
          print('4//2=',4//2)
          print('4**2=',4**2)
          print('9%2=',9%2)
```

```
1+1= 2
2-5= -3
10*3= 30
5/2= 2.5
5//2= 2
4/2= 2.0
4//2= 2
4**2= 16
9%2= 1
```

أولوية العمليات الحسابية

```
In [10]: print(5+5*2+5+2**3+2*(5-2))
```

```
34
```

أنماط البيانات

أنماط البيانات الأساسية هي numbers و list و set و tuple و dictionary و String

يوجد نوعين أساسيين لأنماط البيانات في بايثون هما: mutable و immutable.

Mutable: يمكن تغيير قيمتها مثل List و Dictionary و Set.

Immutable: لا يمكن تغيير قيمتها مثل Numbers و String و Tuple.

يتم تعريف المتغيرات بذكر اسمها والقيمة فقط دون ذكر نوع المتغير ولمعرفة نوع المتغير نستخدم التابع type وهو تابع built in أي موجود بشكل افتراضي في البايثون.

مثال:

```
In [3]: x=5
         type(x)
```

```
Out[3]: int
```

```
In [4]: x='ali'
         type(x)
```

```
Out[4]: str
```

```
In [5]: x=1.3
         type(x)
```

```
Out[5]: float
```


- لمعرفة التوابع المكتوبة مسبقاً في البايثون (built in) نستخدم التعليمة (dir(__builtins__)) ولمعرفة التوابع الخاصة بنمط بيانات محدد نستخدم التعليمة (dir(نوع المعطيات)) أو (اسم المتغير من نوع معطيات محدد) dir(int) مثلاً:
- البايثون dynamic data type أي يمكن إسناد قيم من أنواع مختلفة لنفس اسم المتغير.
- لمعرفة عنوان الذاكرة لمتغير أو object يمكن استخدام التعليمة (اسم المتغير).id.

Numbers:

لها أنواع مختلفة مثل integer و float و complex وبعض التوابع التي نستخدمها مع الـ numbers:

التابع	الاستخدام
abs()	القيمة المطلقة أو الطويلة
int()	التحويل إلى عدد صحيح integer أي حذف الجزء العشري
float()	التحويل إلى عدد عشري float أي إضافة جزء عشري (.0)
round()	التقريب إلى أقرب عدد صحيح ويمكن أخذ بارمترين هما العدد العشري ورقم الخانة العشرية المراد التقريب بالاعتماد عليها

مثال:

numbers

```
In [54]: a=-5
          b=3.562
          c=2+4j
          print('abs(a)=',abs(a))
          print('int(b)=',int(b))
          print('float(a)=',float(a))
          print('round(b)=',round(b))
          print('round(b,0)=',round(b,0))
          print('round(b,1)=',round(b,1))
          print('round(b,2)=',round(b,2))
          print('round(b,3)=',round(b,3))
          print('round(b,4)=',round(b,4))

          abs(a)= 5
          int(b)= 3
          float(a)= -5.0
          round(b)= 4
          round(b,0)= 4.0
          round(b,1)= 3.6
          round(b,2)= 3.56
          round(b,3)= 3.562
          round(b,4)= 3.562
```

In []:

:String

لا يمكن تغيير قيمتها immutable وهي عبارة عن سلسلة محارف لكل محرف index (فهرس) وتبدأ الـ index من الصفر وليس من الواحد. يمكن التعبير عنها بعلامات الاقتباس ' ' أو " " ولأجل النزول بمقدار سطر نستخدم الرمز \n أو يمكن تعريف فقرة تحوي أكثر من سطر دون استخدام \n وذلك بكتابة النص ضمن 3 علامات اقتباس """ TEXT """ وعند إظهارها على الخرج ستظهر كترتيب أسطر، يمكن الوصول إلى محرف في الـ string عن طريق الـ index الخاص به كما هو موضح في المثال.

- الفهرسة في بايثون أو الـ indexing يمكن أن تتبع ترتيب موجب من 0 إلى عدد معين حيث الصفر يمثل فهرس أول عنصر في الـ string أو ترتيب سالب حيث يكون فهرس آخر عنصر هو -1 والعنصر الذي قبله -2 وهكذا

مثال:

```
In [60]: s='i am immutable and you cannot change me'
print('s[0]=' ,s[0])
print('s[-1]=' ,s[-1])
print('s[-2]=' ,s[-2])
print('s[5]=' ,s[5])
```

```
s[0]= i
s[-1]= e
s[-2]= m
s[5]= i
```

```
In [ ]:
```

String

```
In [55]: s='i am immutable and you cannot change me'
print(s[0])
s[0]='ali'
```

i

```
-----
TypeError                                 Traceback (most recent call last)
Input In [55], in <cell line: 3>()
      1 s='i am immutable and you cannot change me'
      2 print(s[0])
----> 3 s[0]='ali'

TypeError: 'str' object does not support item assignment
```

immutable

```
In [58]: m=""" hello,
my name is mohammad"""
a="hello,\n my name is ali"
print(m)
print(a)
```

```
hello,
my name is mohammad
hello,
my name is ali
```

بعض التوابع التي يمكن استخدامها مع المتغيرات من نوع String:

بفرض عرفنا متغير `s='python'`

اسم التابع	مثال	الخرج او النتيجة	الشرح
<code>len()</code>	<code>len(s)</code>	6	عدد المحارف ضمن السلسلة المحرفية
<code>upper()</code>	<code>s.upper()</code>	"PYTHON"	تحويل جميع الحروف الى احرف كبيرة
<code>lower()</code>	<code>s.lower()</code>	"python"	تحويل جميع الحروف إلى أحرف صغيرة
<code>count()</code>	<code>s.count('th')</code>	1	حساب عدد مرات تكرار محرف أو مجموعة محارف متتالية
<code>capitalize()</code>	<code>"coDE".capitalize()</code>	"Code"	تحويل الخرف الاول الى حرف كبير والبقية الى احرف صغيرة
<code>title()</code>	<code>"beN hur".title()</code>	"Ben Hur"	تحويل او حرف من كل كلمة في العبارة إلى حرف كبير والبقية احرف صغيرة
<code>rstrip()</code>	<code>" ab ".rstrip()</code>	" ab"	حذف الفراغات على يمين السلسلة المحرفية
<code>lstrip()</code>	<code>" ab ".lstrip()</code>	"ab "	حذف الفراغات على يسار السلسلة المحرفية

strip()	“ ab ”. strip()	“ab”	حذف الفراغات على يمين ويسار السلسلة المحرفية
---------	-----------------	------	--

List:

يمكن تغيير قيمته mutable والمتغير من نوع list يحوي بداخله على متغيرات قد تكون من نفس النوع أو من أنواع مختلفةint, float, string, list.... وتكون مرتبة أي لكل عنصر فهرس يدل عليه. يتم تعريف الـ list باستخدام الأقواس المربعة [].

مثال:

List

```
[71]: l=['sara','karam',5,6.3,[1,3,'ali']]
print('l[0]=' ,l[0])
print('l[3]=' ,l[3])
print('l[4]=' ,l[4])
print('l[-1]=' ,l[-1])

l[0]= sara
l[3]= 6.3
l[4]= [1, 3, 'ali']
l[-1]= [1, 3, 'ali']
```

بعض التتابع المستخدمة مع الـ List:

بفرض تم تعريف المتغيرات التالية:

```
t=["ali",1997,"mohammad"]
```

```
nums=[4,2,11,6]
```

```
words=["spam","ni"]
```

التابع	مثال	النتيجة (الخرج)	الشرح
len()	len(words)	2	عدد العناصر في القائمة
max()	max(nums)	11	القيمة العظمى (يجب أن تكون جميع العناصر من نفس النوع)
min()	min(nums)	2	القيمة الصغرى (يجب أن تكون العناصر من نفس النوع)
Sum()	sum(nums)	23	مجموع العناصر ويجب أن تكون العناصر أعداد
count()	nums.count(11)	1	عدد مرات تكرار العنصر وهنا عدد مرات تكرار العدد 11
index()	nums.index(11)	2	فهرس العنصر (أول ظهور من اليسار)
reverse()	words.reverse()	["ni","spam"]	عكس ترتيب العناصر
clear()	t.clear()	[]	جعل القائمة فارغة
append()	nums.append(7)	[4,2,11,6,7]	إضافة عنصر إلى آخر الـ list
extend()	nums.extend([3,5])	[4,2,11,6,3,5]	إضافة عناصر الـ list إلى آخر الـ list الأصلية
del()	del t[-1]	["ali",1997]	حذف عنصر ذو فهرس محدد
remove()	nums.remove(2)	[4,11,6]	حذف أول ظهور للعنصر المحدد
insert()	words.insert(1,"wink")	["spam","wink","ni"]	إضافة عنصر جديد قبل العنصر المذكور فهرسه

+	['a',1]+[2,'b']	['a',1,2,'b']	دمج قائمتين وتحويلهما إلى قائمة واحدة
*	[0]*3	[0,0,0]	تكرار عنصر في قائمة

التابع `sort()` يقوم بترتيب عناصر الـ `list` فإذا كانت العناصر أعداد يرتبها تصاعدياً وإذا كانت `String` وأعداد يرتب الأعداد في البداية والـ `String` بعدها حسب ترتيب أول حرف. ومن أهم التوابع أيضاً هما التابعين `split()` و `join()`:

`split()`: يحول الـ `string` إلى `list`

`join()`: يحول الـ `list` إلى `string`

يمكن الاستفادة من هذه التوابع عند إنشاء `client-server` حيث عند إرسال أكثر من رسالة بينهما فيمكن تجميع الـ `String` في `list` واحدة وإرسالها مرة واحدة (سيتم الشرح في القسم الثاني من المقرر)

مثال:

```
In [37]: s1='a,b,c'
s2='hello word'
l1=s1.split(',')
l2=s2.split(' ')
print(l1)
print(l2)
l3=s1.split()
l4=s2.split('o')
print(l3)
print(l4)

['a', 'b', 'c']
['hello', 'word']
['a,b,c']
['hell', ' w', 'rd']
```

Split

```
In [38]: l=['a','b','c']
s=' '.join(l)
s2='*'.join(l)
s3='123'.join(l)
print(s)
print(s2)
print(s3)

a b c
a*b*c
a123b123c
```

join

بعض العمليات على القوائم (list):

list slicing: هي عملية اقتطاع جزء من الـ list ونستخدم لذلك المعامل [:] حيث يمكننا من تحديد البداية والنهاية للاقتطاع أو البداية فقط أو النهاية فقط.

list1[m:n] يتم اقتطاع العناصر من العنصر ذو الفهرس m إلى العنصر ذو الفهرس n-1

list1[m:] يتم اقتطاع العناصر من العنصر ذو الفهرس m إلى آخر عنصر في الـ list

list1[:n] يتم اقتطاع العناصر من أول عنصر إلى العنصر ذو الفهرس n-1

list1[:] يتم اقتطاع جميع العناصر من أول عنصر إلى آخر عنصر وتفيد في عملية النسخ.

مثال:

```
In [41]: list1=['a','b','c','d','e','f']
print('list1[1:3]=' ,list1[1:3])
print('list1[-4:-2]=' ,list1[-4:-2])
print('list1[:4]=' ,list1[:4])
print('list1[4:]=' ,list1[4:])
del list1[1:3]
print('new list is ',list1)
print('list1[2:len(list1)]=' ,list1[2:len(list1)])
print('list1[1:3][1]=' ,list1[1:3][1])
print('list1[3:2]=' ,list1[3:2])

list1[1:3]= ['b', 'c']
list1[-4:-2]= ['c', 'd']
list1[:4]= ['a', 'b', 'c', 'd']
list1[4:] = ['e', 'f']
new list is  ['a', 'd', 'e', 'f']
list1[2:len(list1)]= ['e', 'f']
list1[1:3][1]= e
list1[3:2]= []
```

نسخ الـ list:

لنسخ الـ list لا يمكن استخدام المعامل = لأن المتغير الجديد الذي أسندنا له المتغير الأصلي والمتغير الأصلي سوف يدلان (يؤشران) على نفس المكان في الذاكرة وأي تغيير على أحدهما يغير الثاني أيضاً.

لذلك نستخدم تابع النسخ `list()` أو نستخدم `list slicing` لنسخ `list` موجودة لدينا.

مثال:

الطريقة الخاطئة: نلاحظ عنوان الذاكرة للمتغير `l1` والمتغير `l2` هو نفسه وبالتالي المعامل = قام بجعل المتغير `l2` يؤشر إلى نفس خانة الذاكرة الخاصة بالمتغير `l1`.

```
In [43]: l1=['a','b','c']
          l2=l1
          print('عنوان المتغير 11 في الذاكرة',id(l1))
          print('عنوان المتغير 12 في الذاكرة',id(l2))

          l2[0]='x'
          print(l1)
```

عنوان المتغير 11 في الذاكرة 1322454051072
عنوان المتغير 12 في الذاكرة 1322454051072
['x', 'b', 'c']

الطريقة الصحيحة: نلاحظ اختلاف عنوان الذاكرة الخاص بكل من `l1` و `l2`.

```
In [44]: l1=['a','b','c']
          l2=list(l1)
          print('عنوان المتغير 11 في الذاكرة',id(l1))
          print('عنوان المتغير 12 في الذاكرة',id(l2))

          l2[0]='x'
          print(l1)
```

عنوان المتغير 11 في الذاكرة 1322454115520
عنوان المتغير 12 في الذاكرة 1322454099200
['a', 'b', 'c']

1

```
In [45]: l1=['a','b','c']
          l2=l1[:]
          print('عنوان المتغير 11 في الذاكرة',id(l1))
          print('عنوان المتغير 12 في الذاكرة',id(l2))

          l2[0]='x'
          print(l1)
```

عنوان المتغير 11 في الذاكرة 1322454102400
عنوان المتغير 12 في الذاكرة 1322454093952
['a', 'b', 'c']

2

- في حال كانت الـ `list` متداخلة (أي `list` ضمن `list`) وأردنا الوصول إلى عنصر ضمن الـ `list` الأم فنستخدم لذلك فهرس واحد `[]` وقد يكون العنصر قيمة واحدة أو `list` أو `string` وللوصول

إلى عنصر ضمن الـ list الداخلية على سبيل المثال نستخدم فهرسين [] حيث الفهرس الذي يقع على اليسار أي أول فهرس هو الـ list الداخلية والفهرس الذي يليه هو فهرس العنصر الذي يقع ضمن الـ list الداخلية ونفس الأمر فيما يخص الـ string حيث يمكننا الوصول إلى محرف ضمن الـ string الموجودة ضمن الـ list.

مثال:

```
i=[3,[5,2,['first','second']],9,10]
```

```
In [50]: i=[3,[5,2,['first','second']],9,10]
print('i[1]=',i[1])
print('i[1][2]=',i[1][2])
print('i[1][2][0]=',i[1][2][0])
print('i[1][2][0][0]=',i[1][2][0][0])
print('i[-1]=',i[-1])

i[1]= [5, 2, ['first', 'second']]
i[1][2]= ['first', 'second']
i[1][2][0]= first
i[1][2][0][0]= f
i[-1]= 10
```

:tuple

عبارة عن نمط معطيات يشبه الـ list مع فرق أنه من نوع immutable أي لا يمكن تغيير قيمته بعد إنشائه ويتم تعريف الـ tuple عن طريق الأقواس المنحنية ().

:set

نمط بيانات أيضاً من نوع mutable ويتم تعريفه باستخدام الأقواس المتعرجة {} وتكون عناصر الـ set غير مرتبة أي العناصر ليس لها فهرس وأيضاً الـ Set لا تقبل عناصر مكررة.

:dictionary

أهم أنماط البيانات في بايثون وعناصرها مكونة من أزواج key:value حيث الـ key يمكن تشبيهه بالـ index ويمكن أن يكون الـ key و value من أي نوع بيانات وأقواس الـ dictionary أيضاً متعرجة.

على سبيل المثال: `d={'first':'ali','second':'sara',0:[2,'a']}`

يمكن الوصول إلى الـ value عن طريق الـ key أي `d[key]=value` مثلاً `d['first']` يعطي 'ali'

```
In [53]: d={'first':'ali','second':'sara',0:[2,'a']}  
         d['first']|
```

```
Out[53]: 'ali'
```

التابع `keys()` يعطي الـ keys الموجودة ضمن المتغير من نوع dictionary والتابع `values` يعطي الـ values الموجودة ضمن المتغير من نوع dictionary.

```
In [54]: d={'first':'ali','second':'sara',0:[2,'a']}  
         d.keys()
```

```
Out[54]: dict_keys(['first', 'second', 0])
```

لإظهار الـ الأزواج key مع value نستخدم التابع `items()`

```
In [55]: d={'first':'ali','second':'sara',0:[2,'a']}  
         d.items()
```

```
Out[55]: dict_items([('first', 'ali'), ('second', 'sara'), (0, [2, 'a'])])
```

أما في حال طباعة المتغير `d` بتعليمة الطباعة بدون استخدام التتابع الخاصة بالـ dictionary سيكون الخرج بالشكل التالي:

```
In [56]: d={'first':'ali','second':'sara',0:[2,'a']}  
print(d)
```

```
{'first': 'ali', 'second': 'sara', 0: [2, 'a']}
```

لاختبار إذا كانت value محددة موجودة ضمن متغير من نوع dictionary نستخدم الصيغة التالية:

'value' in اسم المتغير.values()

التي تعيد قيمة منطقية True أو False

```
In [58]: d={'first':'ali','second':'sara',0:[2,'a']}  
'ali' in d.values()
```

```
Out[58]: True
```

```
In [61]: d={'first':'ali','second':'sara',0:[2,'a']}  
'ahmad' in d.values()
```

```
Out[61]: False
```

لاختبار إذا كان key محدد موجود ضمن متغير من نوع dictionary نستخدم الصيغة التالية:

'key' in اسم المتغير.keys()

التي تعيد قيمة منطقية True أو False

```
In [63]: d={'first':'ali','second':'sara',0:[2,'a']}  
0 in d.keys()
```

```
Out[63]: True
```

```
In [64]: d={'first':'ali','second':'sara',0:[2,'a']}  
'0' in d.keys()
```

```
Out[64]: False
```

```
In [65]: d={'first':'ali','second':'sara',0:[2,'a']}
         'ali' in d.keys()
```

Out[65]: False

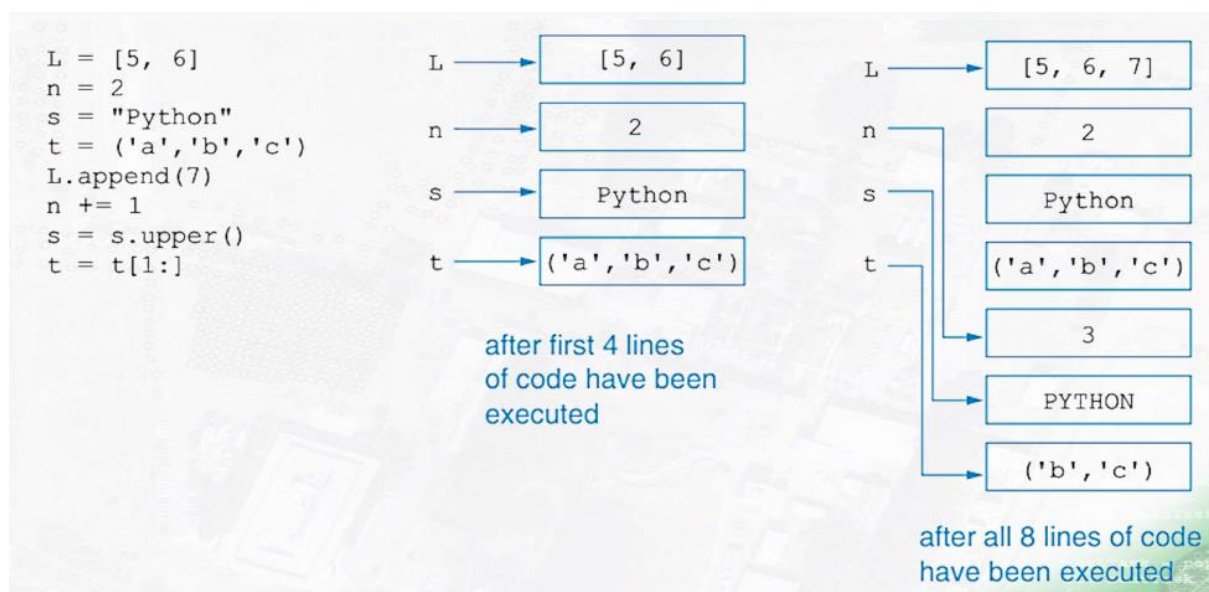
- ملاحظة: الـ slicing تم شرحه من أجل نوع المعطيات list ولكن يمكن تطبيق الـ slicing على جميع أنواع المعطيات التي تتعامل مع الفهارس مثل string و tuple.

مثال:

```
In [5]: t=(1,2,3,'first','second')
        s='this is a string'
        l=[1,2,3,4,5,'a']
        print('tuple slicing example is t[2:4]=',t[2:4])
        print('string slicing example is s[1:4]=',s[1:4])
        print('list slicing example is l[2:4]=',l[2:4])
```

tuple slicing example is t[2:4]= (3, 'first')
string slicing example is s[1:4]= his
list slicing example is l[2:4]= [3, 4]

مثال: بخصوص mutable و immutable



في هذا المثال قمنا بإنشاء 4 متغيرات: L من نوع list و n من نوع int و s من نوع string و t من نوع tuple. السطور البرمجة التالية هي عمليات تغيير على قيم المتغيرات.

L تبقى كما هي كونها من نوع list ويمكن تغيير قيمتها مع المحافظة على موقعها في الذاكرة (mutable) أما s و n و t عند تغيير قيمتها يتم بناء متغير بنفس الاسم ولكن في موقع مختلف من الذاكرة والموقع القديم يتم حذفه بواسطة أداة تدعى garbage collector.

```
In [10]: L=[5,6]
n=2
s="python"
t=('a','b','c')
print("المواقع القديمة في الذاكرة", 'L= ',id(L),'n= ',id(n),'s= ',id(s),'t=',id(t))
L.append(7)
n+=1
s=s.upper()
t=t[1:]
print("المواقع الجديدة في الذاكرة", 'L= ',id(L),'n= ',id(n),'s= ',id(s),'t=',id(t))
```

L=	2830443036480	n=	2832479944976	s=	2832528021488	t=	2830442675008	المواقع القديمة في الذاكرة
L=	2830443036480	n=	2832479945008	s=	2830442945200	t=	2830442819840	المواقع الجديدة في الذاكرة

التابعين input() و print()

التابع input() يستخدم للسماح للمستخدم بإدخال بيانات إلى كود البايثون، حيث يتم إيقاف تنفيذ الكود عند هذه التعليمة ولا يتم تنفيذ أي تعليمة بعدها إلا بعد قيام المستخدم بإدخال قيمة، والقيمة التي يعيدها هذا التابع هي من نوع String ويمكن أيضاً وضع رسالة ضمن قوسين التابع لتظهر على الشاشة.

مثال:

```
In [11]: s=input('enter youe name: ')
print(s)
print(type(s))
```

```
enter youe name: ali
ali
<class 'str'>
```

في حال كنا نريد إدخال رقم معين والقيام بعملية حسابية على هذا الرقم سنقوم بتحويل القيمة المدخلة من نوع string إلى قيمة عددية ولتكن integer وذلك باستخدام التابع int().

مثال:

```
In [12]: a=input('enter number: ')
          print(a)
          print(type(a))
          n=int(a)
          print(n)
          print(type(n))
```

```
enter number: 2
2
<class 'str'>
2
<class 'int'>
```

مثال:

```
In [13]: a=input('enter first number: ')
          b=input('enter second number: ')
          c=a+b
          print(c)
```

```
enter first number: 3
enter second number: 2
32
```



```
In [14]: a=int(input('enter first number: '))
          b=int(input('enter second number: '))
          c=a+b
          print(c)
```

```
enter first number: 3
enter second number: 2
5
```



التابع print() يقوم بطباعة متغير من نوع معين على الشاشة وبعد الطباعة يقوم بجعل مؤشر الخرج يقوم بالنزول سطر. من الممكن أن يأخذ هذا التابع عدد غير محدود من البارمترات تفصل بينها فاصلة عادية. وهناك بعض البارمترات المعرفة مسبقاً ضمنه مثل end= و sep= سنتطرق لها في الامثلة.

مثال:

```
In [24]: a=21
s='a#@'
print('hello','hi',22,'hello word',a,'name',s)
print('السطر الثاني')
print()
print('السطر الرابع')
```

hello hi 22 hello word 21 name a#@

السطر الثاني

السطر الرابع

مثال:

البارمتر `end=""` يجعل نهاية العبارة المطبوعة ليست سطر جديد وبالتالي تعليمة الطباعة التالية لتعليمة الطباعة المستخدم بها هذا البارمتر ستطبع العبارة على نفس السطر وليس في سطر جديد ويمكن تحديد الفاصل بين العبارات برمز محدد نضع ضمن الـ `' '`.

```
In [25]: print('ali',end='- ')
print('ali',end='*')
print('ali')
print('mohammad')
print('mohammad')
print('mohammad')
```

ali-ali*ali

mohammad

mohammad

mohammad

مثال:

البارمتر `sep=""` يحد نوع الفاصل بين كل بارمتر يتم طباعته باستخدام تعليمة الطباعة وفي الحالة الافتراضية أي بدون استخدام هذا البارمتر يكون الفاصل عبارة عن فراغ واحد.

```
In [29]: print('hello', 'word', 'python')
print('hello', 'word', 'python', sep='')
print('hello', 'word', 'python', sep='*')
print('hello', 'word', 'python', sep='\t')
print('hello', 'word', 'python', sep='----')
```

```
hello word python
helloworldpython
hello*word*python
hello    word    python
hello----word----python
```

التعليمات الشرطية:

التعليمة if لها البنية التالية:

```
if      شرط:
    تعليمات
    ...
    نهاية البلوك
```

في حال تحقق الشرط (أي في حال قيمة منطقية True) يتم تنفيذ التعليمات الخاصة بـ if أي الموجودة ضمن بلوك if.

مثال:

```
In [32]: x=int(input('enter number: '))
if x>5:
    print(x,'larger than ',5 )
if x<5:
    print(x,'lower than ',5 )
```

```
enter number: 3
3 lower than 5
```


مثال: if متداخلة

```
In [33]: x=4
         if x<10:
             if x<7:
                 print(x,' is between 0 and 7 ' )
             if x<5:
                 print(x,' is between 0 and 5 ' )

4 is between 0 and 7
4 is between 0 and 5
```

التعليمة if-else : لها البنية التالية:

```
if شرط:
    تعليمات
...
else:
    ...
    تعليمات
```

في حال تحقق الشرط يتم تنفيذ التعليمات الخاصة بـ if وبعد انتهائها يكمل البرنامج التنفيذ دون تنفيذ تعليمات else أما في حال عدم تحقق الشرط فيتم تنفيذ تعليمات else فقط دون تنفيذ تعليمات if.

مثال:

```
In [34]: x=10
         if x>5:
             print(x,'larger than 5')
         else:
             print(x,'lower than 5')
         y=2
         if y>5:
             print(y,'larger than 5')
         else:
             print(y,'lower than 5')

10 larger than 5
2 lower than 5
```

تعليلة if-else المركبة: يتم إختبار أكثر من شرط وعند عدم تحقق جميع الشروط يتم تنفيذ تعليمات خاصة بـ else.

مثال:

```
In [35]: x=10
         if x>15:
             print(1)
         elif x>12:
             print(2)
         elif x>10:
             print(3)
         else:
             print(4)
```

4

```
In [36]: x=14
         if x>15:
             print(1)
         elif x>12:
             print(2)
         elif x>10:
             print(3)
         else:
             print(4)
```

2

بعض التوابع التي تعيد قيمة منطقية:

- isinstance(variable,type) يعيد True إذا كان المتغير variable من نفس النوع المحدد type
- isdigit() يعيد True إذا كانت جميع المحارف ضمن الـ string عبارة عن أرقام.
- isalpha() يعيد True إذا كانت جميع المحارف ضمن الـ string عبارة عن حروف.
- isalnum() يعيد True إذا كانت جميع المحارف ضمن الـ string عبارة عن حروف و أرقام فقط.
- islower() يعيد True إذا كانت جميع المحارف ضمن الـ string عبارة عن حروف صغيرة.
- isupper() يعيد True إذا كانت جميع المحارف ضمن الـ string عبارة عن حروف كبيرة.
- startswith() يأخذ بارمتر محرف ويختبر فيما إذا كانت السلسلة المحرفية تبدأ بهذا المحرف.
- endswith() يأخذ بارمتر محرف ويختبر فيما إذا كانت السلسلة المحرفية تنتهي بهذا المحرف.

مثال:

```
In [41]: x=5
s='python'
print('isinstance(x,int)',isinstance(x,int))
print('isinstance(x,float)',isinstance(x,float))
print()
print('s.isdigit()',s.isdigit())
print('s.isalpha()',s.isalpha())
print()
print('s.isalnum()',s.isalnum())
print('s.islower()',s.islower())
print('s.isupper()',s.isupper())
print()
print('s.isupper()',s.isupper())
print('s.startswith(p)',s.startswith('p'))
print('s.endswith(p)',s.endswith('p'))

isinstance(x,int) True
isinstance(x,float) False

s.isdigit() False
s.isalpha() True

s.isalnum() True
s.islower() True
s.isupper() False

s.isupper() False
s.startswith(p) True
s.endswith(p) False
```

مثال:

```
In [42]: x=50
if isinstance(x,int):
    if x>=60:
        print('passed')
    else:
        print('field')
else:
    print('enter integer number')

field
```

الحلقات التكرارية:

يوجد لدينا حلقتين تكراريتين هما `while` و `for` وبشكل عام وظيفة الحلقات هي تكرار مجموعة تعليمات عدد محدد او غير محدد من المرات وذلك بالاعتماد على شرط محدد بمعنى انه طالما تكون نتيجة الشرط `True` يتم تنفيذ التعليمات ضمن الحلقة وعندما يختل الشرط أي يصبح `False` ينتهي تنفيذ تعليمات الحلقة.

حلقة `while`: لها البنية التالية:

```
while    شرط:
    تعليمات
    ...
```

حلقة `for` لها البنية التالية:

```
for    شرط:
    تعليمات
    ...
    ..
```

الشرط في حلقة `for` يكون من الصيغة : `for .. in ...:`

مثال:

```
In [43]: n=1
         while n < 6:
             print(n, end=" ")
             n+=1
```

1 2 3 4 5

مثال:

```
for i in 'network programming language':
    print(i,end=',')
n,e,t,w,o,r,k, ,p,r,o,g,r,a,m,m,i,n,g, ,l,a,n,g,u,a,g,e,
for i in [5,7,15,3]:
    print(i,end=',')
5,7,15,3,
for i in (5,2,9):
    print(i,end=',')
5,2,9,
for i in range(5,500,50):
    print(i,end='_')
5_55_105_155_205_255_305_355_405_455_
```

التابع `range()` يعيد مجموعة قيم عددية، يأخذ ثلاث بارمترات هي أول قيمة (قيمتها الافتراضية صفر) وآخر قيمة (نرسل له n فتكون آخر قيمة هي $n-1$) والخطوة (قيمتها الافتراضية واحد) وفي حال أرسلنا له بارمترين يكونان أول قيمة وآخر قيمة والخطوة تبقى على قيمتها الافتراضية أما في حال أرسلنا له قيمة واحدة فيعتبرها آخر قيمة وتكون أول قيمة هي الصفر. يستخدم هذا التابع بشكل عام مع حلقات `for`.

مثال:

```
In [51]: for i in range(5):
          print(i)
0
1
2
3
4

In [52]: for i in range(5):
          print(i,end=" ")
0 1 2 3 4

In [54]: for i in range(2,5):
          print(i,end=" ")
2 3 4

In [58]: for i in range(0,5,2):
          print(i,end=" ")
0 2 4
```

في حال أردنا مقاطعة تسلسل التكرار في الحلقات نستخدم لذلك التعليمتين break و continue.

تعليمية break: عند تنفيذ هذه التعليمية يتم الخروج من الحلقة حتى وإن لم يخل الشرط.

تعليمية continue: عند تنفيذ هذه التعليمية يتم تجاهل التعليمات ضمن الحلقة والتي تقع بعد هذه التعليمية والانتقال إلى التكرار التالي للحلقة.

مثال:

```
In [61]: for i in range(7):
          print(i,end=" ")
          if i==5:
              break
```

0 1 2 3 4 5

```
In [62]: for i in range(7):
          print(i,end=" ")
          if i==6:
              continue
```

0 1 2 3 4 5 6

التعليمات المنطقية في بايثون هي and, or, not وسنعرض مثال من المحاضرات:

Example 6 Logical Operators Suppose the variable n has value 4 and the variable $answ$ has value "Y". Determine whether each of the following conditions evaluates to **True** or **False**.

- (a) $(2 < n)$ and $(n < 6)$
- (b) $(2 < n)$ or $(n == 6)$
- (c) not $(n < 6)$
- (d) $(answ == "Y")$ or $(answ == "y")$
- (e) $(answ == "Y")$ and $(answ == "y")$
- (f) not $(answ == "y")$
- (g) $((2 < n)$ and $(n == 5 + 1))$ or $(answ == "No")$
- (h) $((n == 2)$ and $(n == 7))$ or $(answ == "Y")$
- (i) $(n == 2)$ and $((n == 7)$ or $(answ == "Y"))$

SOLUTION

SOLUTION

- (a) **True**, because the conditions $(2 < 4)$ and $(4 < 6)$ are both true.
- (b) **True**, because the condition $(2 < 4)$ is true. The fact that the condition $(4 == 6)$ is false does not affect the conclusion. The only requirement is that at least one of the two conditions be true.
- (c) **False**, because $(4 < 6)$ is true.
- (d) **True**, because the first condition becomes $("Y" == "Y")$ when the value of $answ$ is substituted for $answ$.
- (e) **False**, because the second condition is false. Actually, this compound condition is false for any value of $answ$.
- (f) **True**, because $("Y" == "y")$ is false.
- (g) **False**. In this logical expression, the compound condition $((2 < n)$ and $(n == 5 + 1))$ and the simple condition $(answ == "No")$ are joined by the logical operator or. Because both these conditions are false, the total condition is false.

بفرض لدينا شرط أول اسمه cond1 وشرط ثاني cond2 نعرض يمكننا عرض بعض الحيل المنطقية المكافئة لبعضها والتي قد تساعدنا في بعض المشاكل البرمجية:

$\text{not}(\text{cond1 and cond2}) \quad \longleftrightarrow \quad \text{not}(\text{cond1}) \text{ or } \text{not}(\text{cond2})$

$\text{not}(\text{cond1 or cond2}) \quad \longleftrightarrow \quad \text{not}(\text{cond1}) \text{ and } \text{not}(\text{cond2})$

list comprehension: إحدى الطرق لتبسيط كتابة الكود هي عمليات إنشاء عناصر الـ list بتعبير رياضي أو طريقة معينة وذلك بالاستفادة من الحلقات والتعليمات الشرطية بطريقة بسيطة.

مثال: في حال أردنا إنشاء list مكونة من أعداد من 0 إلى 30 يمكننا تعريف list بالطريقة التالية:

```
In [139]: l1=[x for x in range(31)]  
print(l1)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

وفي حال كنا نريد الأعداد الزوجية فقط من 0 لـ 30 نضيف شرط if:

```
In [140]: l2=[x for x in range(31) if x%2==0]  
print(l2)
```

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]

وفي حال أردنا نسبة مئوية لمجموعة أعداد في متغير من نوع list معرف مسبقاً:

```
In [142]: l=[4323,4000,2000,1500]  
l1=[x*0.01 for x in l]  
print(l1)
```

[43.230000000000004, 40.0, 20.0, 15.0]

أما إذا أردنا مثلاً النسبة المئوية للأعداد الأكبر من 2000:

```
In [143]: l=[4323,4000,2000,1500]  
l1=[x*0.01 for x in l if x>2000]  
print(l1)
```

[43.230000000000004, 40.0]

بالمقارنة مع الطريقة التقليدية نجد أننا حصلنا على نفس النتيجة بسطر واحد مقارنة بـ 4 أسطر. الطريقة التقليدية:

```
In [149]: l=[4323,4000,2000,1500]
l1=[]
for i in l:
    if i>2000:
        l1.append(i*0.01)
print(l1)

[43.230000000000004, 40.0]
```

يمكننا أيضاً إنشاء متغير من نوع dictionary وذلك بالاستفادة من dictionary comprehension.

أمثلة:

1- جدول الضرب من 1 إلى 5.

```
In [71]: for m in range(1,6):
          for n in range(1,6):
              print(m,'X',n,'=', m*n, end="\t")
          print()
```

1 X 1 = 1	1 X 2 = 2	1 X 3 = 3	1 X 4 = 4	1 X 5 = 5
2 X 1 = 2	2 X 2 = 4	2 X 3 = 6	2 X 4 = 8	2 X 5 = 10
3 X 1 = 3	3 X 2 = 6	3 X 3 = 9	3 X 4 = 12	3 X 5 = 15
4 X 1 = 4	4 X 2 = 8	4 X 3 = 12	4 X 4 = 16	4 X 5 = 20
5 X 1 = 5	5 X 2 = 10	5 X 3 = 15	5 X 4 = 20	5 X 5 = 25

لدينا حلقتي for الأولى تعرف لديها المتغير m الذي يأخذ القيم من التابع range حيث تكون أول قيمة هي 1.

الحلقة الثانية تقع ضمن الحلقة الأولى وبالتالي يتم تنفيذها ولديها المتغير n الذي يأخذ أيضاً القيم من التابع range ويتم تنفيذ تعليمة الطباعة ضمن الحلقة الثانية والعودة إلى التكرار الثاني حيث يأخذ المتغير n القيمة التالية تلقائياً من التابع range التي تساوي 2 وهكذا حتى يتم التكرار خمس مرات وتكون قيمة n أصبح تساوي 5 وعندها يتم الانتهاء من الحلقة الداخلية ويتم تنفيذ التعليمة التالية ضمن الحلقة الأولى والتي هي تعليمة print ومهمتها هنا هي النزول بمقدار سطر في الخرج وبعدها ينتهي أول تكرار للحلقة

الأولى وتأخذ m القيمة 2 من التابع range ويتم تنفيذ الحلقة الثانية مرة أخرى بمقدار 5 تكرارات ضمن التكرار الثاني للحلقة الأولى وهكذا حتى انتهاء تنفيذ البرنامج وبالتالي تم تكرار الحلقة الأولى بمقدار خمس مرات وفي كل مرة يتم تكرار الحلقة الداخلية بمقدار 5 مرات.

2- يراد إنشاء متغير اسمه months يحوي أسماء الأشهر وطباعة أول ثلاثة حروف من كل شهر.

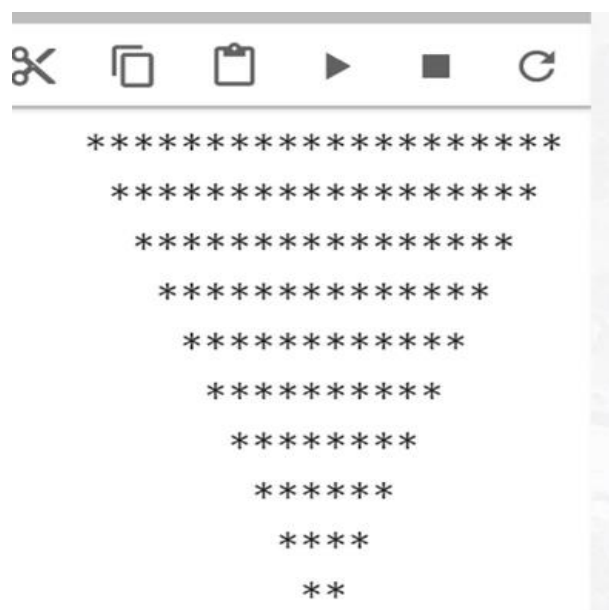
```
In [76]: m=[]
months=["January","February","March","April","May",
        ,"June","July","August","september","October","November","December"]
for i in range(len(months)):
    m.append(months[i][0:3])
print(m)

['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'sep', 'Oct', 'Nov',
'Dec']
```

هنا في حلقة for سنستخدم i كفهرس وبالتالي سنحتاج لجعل i تأخذ قيم من 0 إلى 11 كون طول المتغير months هو 12.

عرفنا list فارغة اسمها m لوضع أول 3 أحرف من كل شهر ضمنها باستخدام التابع append().

3- مطلوب طباعة الخرج التالي:



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

الحل:

```
In [113]: s="*" * 20  
print("\t\t",s)  
for i in range(1,len(s)):  
    print("\t\t", " "*i+s[i:-i])  
  
*****  
      *****  
        *****  
          *****  
            *****  
              *****  
                *****  
                  *****  
                    ****  
                      ***  
                        **
```

4- برنامج استعلام عن علامات مواد وإضافة علامات إلى المواد.

```
course1={'karam':70,'sara':50,'rana':84,'sam':45}
course2={'karam':90,'sara':80,'rana':60,'sam':30}
course3={'karam':72,'sara':61,'rana':82,'sam':90}
l=[course1,course2,course3]
while True:
    avg=0
    ty=input('are you student or teacher, please enter s or t: ')
    if ty=='s':
        sname=input('enter your name: ')
        for d in l:
            if sname in d.keys():
                print(d[sname])
                avg+=d[sname]
            else:
                print('don't match')
        if avg:
            print("your average rate is ", avg/len(l))
        s=input('do you want to continue as for yes enter y: ')
        if s!='y':
            break
    elif ty=='t':
        c=input('select course, 1 or 2 or 3: \n')
        if c=='1':
            sname=input('enter student name: \n')
            course1[sname]=int(input('enter the mark: \n'))
        elif c=='2':
```

```

    sname=input('enter student name: \n')
    course2[sname]=int(input('enter the mark: \n'))
elif c=='3':
    sname=input('enter student name: \n')
    course3[sname]=int(input('enter the mark: \n'))

else:
    print('dont match, try again \n')
s=input('do you want to continue as for yes enter y: ')
if s!='y':
    break
else:
    print('error input, try again\n')

```

الخرج:

```

are you student or teacher, please enter s or t: s
enter your name: ali
dont match
dont match
dont match
do you want to continue as for yes enter y: y
are you student or teacher, please enter s or t: t
select course, 1 or 2 or 3:
2
enter student name:
ali
enter the mark:
100
do you want to continue as for yes enter y: y
are you student or teacher, please enter s or t: s
enter your name: ali
dont match
100
dont match
your average rate is 33.333333333333336
do you want to continue as for yes enter y: y
are you student or teacher, please enter s or t: karam
error input, try again

are you student or teacher, please enter s or t: s
enter your name: sara
50
80
61
your average rate is 63.666666666666664
do you want to continue as for yes enter y: n

```

قمنا بإنشاء 3 متغيرات من نوع dictionary كل متغير يمثل علامات الطلاب في مادة محددة ويحوي كل متغير اسم الطالب كـ key وعلامة الطالب كـ value وبعدها قمنا بوضع هذه المتغيرات ضمن متغير list من نوع list سنوضح لاحقاً لماذا...

قمنا بإنشاء حلقة while لا نهائية من أجل عدم توقف البرنامج عندما ينتهي المستخدم من وظيفة واحدة وتنتهي الحلقة عندما يدخل المستخدم أي شيء باستثناء حرف y وذلك في المرحلة التي يقوم بها البرنامج بالسؤال عن المتابعة أو التوقف.

قمنا بإنشاء متغير avg وأسندنا القيمة 0 وذلك لجعله عداد يراكم علامات الطالب من أجل حساب معدله النهائي.

تظهر رسالة لتحديد المستخدم إن كان طالب أو معلم وقمنا بتخزين القيمة التي يدخلها المستخدم في متغير اسمه ty وقمنا باختبار إذا على هذه القيمة.

في حال كان المستخدم طالب تظهر رسالة له من أجل إدخال اسمه وحفظه في متغير sname وبعدها نتحقق من وجود اسم الطالب في كل المقررات.

```
sname=input('enter your name: ')
for d in l:
    if sname in d.keys():
        print(d[sname])
        avg+=d[sname]
    else:
        print('don't match')
```

قمنا في بداية البرنامج بإسناد العلامات إلى متغير l وبالتالي في أول تنفيذ لحلقة for ستكون قيمة d هي نفسها المتغير course1 من نوع dictionary ثم نقوم بالتحقق من وجود اسم الطالب ضمن الـ keys الخاصة بالمتغير d في حال اسم الطالب موجود نقوم بإظهار العلامة وإضافة العلامة إلى avg ويتم التكرار 3 مرات.

في النهاية يظهر معدل الطالب على الشاشة. وبعدها يظهر سؤال للطالب إذا أراد المتابعة أم التوقف وفي حال التوقف تنتفذ تعليمة break.

في حال كان المستخدم معلم تظهر رسالة لإدخال رقم المقرر المطلوب إضافة علامة الطالب إليه. وبعدها يقوم بإدخال اسم الطالب والعلامة الخاصة به.

5- dictionary comprehension: المطلوب إنشاء متغير من نوع dictionary يحوي قيم عبارة

عن العدد ومربعه للأعداد من 10 إلى 20 بطريقتين. الطريقة الأولى: dictionary comprehension والطريقة الثانية: الطريقة التقليدية.

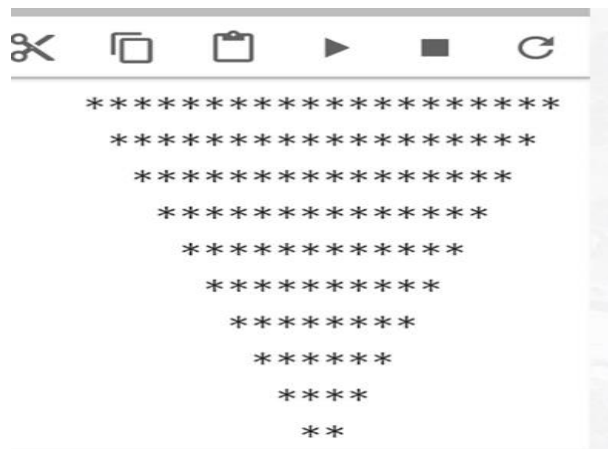
```
In [144]: d={x:x**2 for x in range(10,21)}
print(d)
```

```
{10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225, 16: 256, 17: 289, 18: 324, 19: 361, 20: 400}
```

```
In [145]: d={}
for i in range(10,21):
    d[i]=i**2
print(d)
```

```
{10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225, 16: 256, 17: 289, 18: 324, 19: 361, 20: 400}
```

6- المطلوب الخرج التالي باستخدام list comprehension.



الحل:

```
In [154]: s="*" * 20
l=[" " * i + s[i:-i] for i in range(len(s))]
for i in l:
    print(i)
```

```

*****
 ****
  ***
   **
    *
   **
  ***
 ****
 *****

```