

البرمجة بلغة بايثون Python Programming Language

السلام عليكم اخوتي الكرام

بعد ان شرحنا الجزء الأول من دورة البرمجة بلغة البايثون في ١٢ درساً في الجزء الأول من هذا الكتاب والذي يمكن

تنزيله من الرابط التالي: <http://www.kutub.info/library/book/18993>

نأتي اليوم الى شرح بقية الدروس من هذه الدورة في الجزء الثاني من الكتاب والذي اعتمدنا فيه على منهج كورس

البرمجة بلغة بايثون من جامعة ميشيفان في الولايات المتحدة الامريكية فتابعوا معنا:

ملاحظة: للمزيد عن لغة بايثون وكل ما يخص الحاسوب من شبكات وبرمجة وصيانة وغيرها تفضلوا بزيارتنا على

الرابط التالي:

[/https://mustafasadiq0.wordpress.com](https://mustafasadiq0.wordpress.com)

السلاسل الرمزية (Strings) في لغة بايثون

سبق ان تم التطرق الى اساسيات التعامل مع السلاسل الرمزية للغة بايثون في الدرس الخامس الذي يمكن زيارته

من [هنا](#).

اما اليوم فسنتكلم بشيء من التفصيل عن كيفية التعامل مع السلاسل الرمزية لأهميتها في التعامل مع النصوص في

الملفات النصية وحتى في ملفات الوسائط المتعددة وكبداية بسيطة تعرف السلسلة الرمزية (String) في كل لغات

البرمجة على انها سلسلة من الحروف او الأرقام او الرموز الخاصة محصورة بين علامتي اقتباس مفردة او مزدوجة

(" "، ' ').

يمكن تحويل السلاسل الرمزية الى ارقام صحيحة او عشرية باستخدام الدالتين `int()` and `float()`

على التوالي كما ذكرنا في الدرس الخامس ويمكن دمج (concatenate) سلسلتين رمزيتين بوضع علامة (+) بينهما وكما موضح في المثال التالي:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> first_name="ahmed"
>>> second_name="ali"
>>> third_name="helal"
>>> full_name=first_name + ' '+second_name+ ' '+third_name
>>> print (full_name)
ahmed ali helal
>>>
```

وكذلك المثال التالي لقابلية التحويل:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x='12345'
>>> y='12345'
>>> z=int(x)
>>> w=int(y)
>>> x+y
'1234512345'
>>> z+w
24690
>>> |
```

وكما نلاحظ هنا فقط تم التعامل مع (x,y) على انها سلاسل رمزية وقامت علامة (+) بأرفاق الثانية في نهاية الأولى في حين تم التعامل مع (z,w) على انها اعداد صحيحة وقامت علامة الجمع (+) بطباعة ناتج جمعهما. كذلك يمكن قراءة السلاسل الرمزية من دالة الادخال (input) ثم تحويلها وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=input("enter a number")
enter a number123.456
>>> y=float(x)
>>> x*x
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x*x
TypeError: can't multiply sequence by non-int of type 'str'
>>> y*y
15241.383936
>>> |
Ln: 13 Col: 4
```

وكما نلاحظ هنا فقد قمنا بإدخال سلسلة رمزية تحت مسمى (x) ثم تحويلها الى عدد عشري واعطيناه اسم متغير (y) ثم لاحظنا ان استخدام الضرب (*) بين ال (x) ونفسه اعطى رسالة خطأ لأنهما سلاسل رمزية لا يمكن ضربها ولكنه اعطى ناتج الضرب للقيمة (y*y) لأنهما عددان عشريان يمكن ضربهما وهكذا.

الى هنا يتضح لنا سهولة التعامل مع السلاسل الرمزية كوحدة واحدة وكيفية تحويلها الى اعداد صحيحة او عشرية ولكن الاستخدام الأهم للسلاسل الرمزية يتضمن الدخول الى داخل السلسلة الرمزية والتعامل مع مكوناتها كوحدات مستقلة ويتم ذلك باعتبارها مصفوفة وكما في ادناه:

Hello

0	1	2	3	4
-5	-4	-3	-2	-1

حيث يمكننا ترقيم الرموز بداخل السلسلة الرمزية بدءاً من الصفر من اليسار الى اليمين او من السالب واحد من اليمين الى اليسار وكما موضح في المثال ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="my name is mustafa"
>>> x[1]
'y'
>>> x[0]
'm'
>>> x[2]
' '
>>> len(x)
18
>>> x[17]
'a'
>>> x[18]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x[18]
IndexError: string index out of range
>>> |
```

وهنا من الواضح اننا قمنا بإدخال سلسلة رمزية تحت اسم متغير (x) ثم محاولة طباعة الرمز الثاني (x[1]) والرمز الأول (x[0]) وهكذا لبقية الرموز.

كذلك استطعنا معرفة طول السلسلة الرمزية (عدد رموزها) من الدالة المعرفة بداخل لغة بايثون len(x) ورغم ان طول السلسلة ١٨ الا ان الرمز (x[18]) غير موجود لأننا بدأنا الترقيم من الصفر أي ان اخر عنصر في سلسلة طولها ١٨ هو (x[17]).

كذلك يمكن استخدام عبارات الشرط والتكرار مع السلاسل الرمزية وكما في ادناه:

```
3.py - C:\Users\mustafa\Desktop\3.py Python 3.4.3 Shell
File Edit Format Run Options File Edit Shell Debug Options Window Help
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print (index, letter)
    index = index + 1
print(fruit+fruit)

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015) on win32
Type "copyright", "credits" or "license()" :
>>> ===== RESTARTING SHELL =====
>>>
0 b
1 a
2 n
3 a
4 n
5 a
bananabanana
>>>
```

حيث تم استخدام أداة (while) وبالاستعانة بدالة طول السلسلة لعرض عناصر السلسلة مع فهرس (index) كل منها.

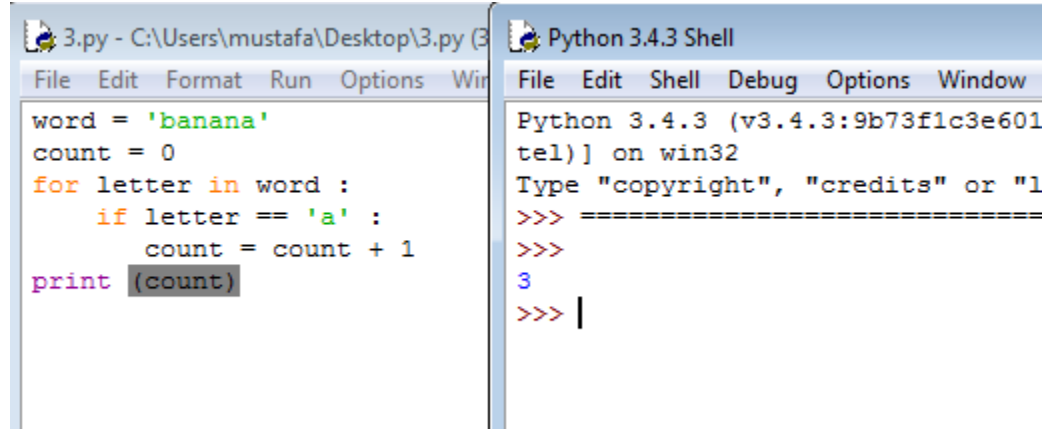
يمكن انجاز نفس العمل باستخدام أداة (for) وكما في ادناه:

```
3.py - C:\Users\mustafa\Desktop\3.py Python 3.4.3 Shell
File Edit Format Run Options File Edit Shell Debug Options Window Help
fruit = 'banana'
count=0
for letter in fruit:
    print (count,letter)
    count=count+1

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015) on win32
Type "copyright", "credits" or "license()" :
>>> ===== RESTARTING SHELL =====
>>>
0 b
1 a
2 n
3 a
4 n
5 a
>>> |
```

وكما نلاحظ فإن القيام بذلك بهذه الطريقة أسهل بكثير مما كان مع (while). كما نلاحظ هنا اننا لم نحتاج الى استخدام دالة طول السلسلة الرمزية (len()) لأننا قمنا بحساب طول السلسلة باستخدام اللوب وال (count).

كذلك يمكن استخدام الحلقات التكرارية والشروط لحساب عدد مرات تكرار رمز معين في السلسلة وكما في المثال التالي:

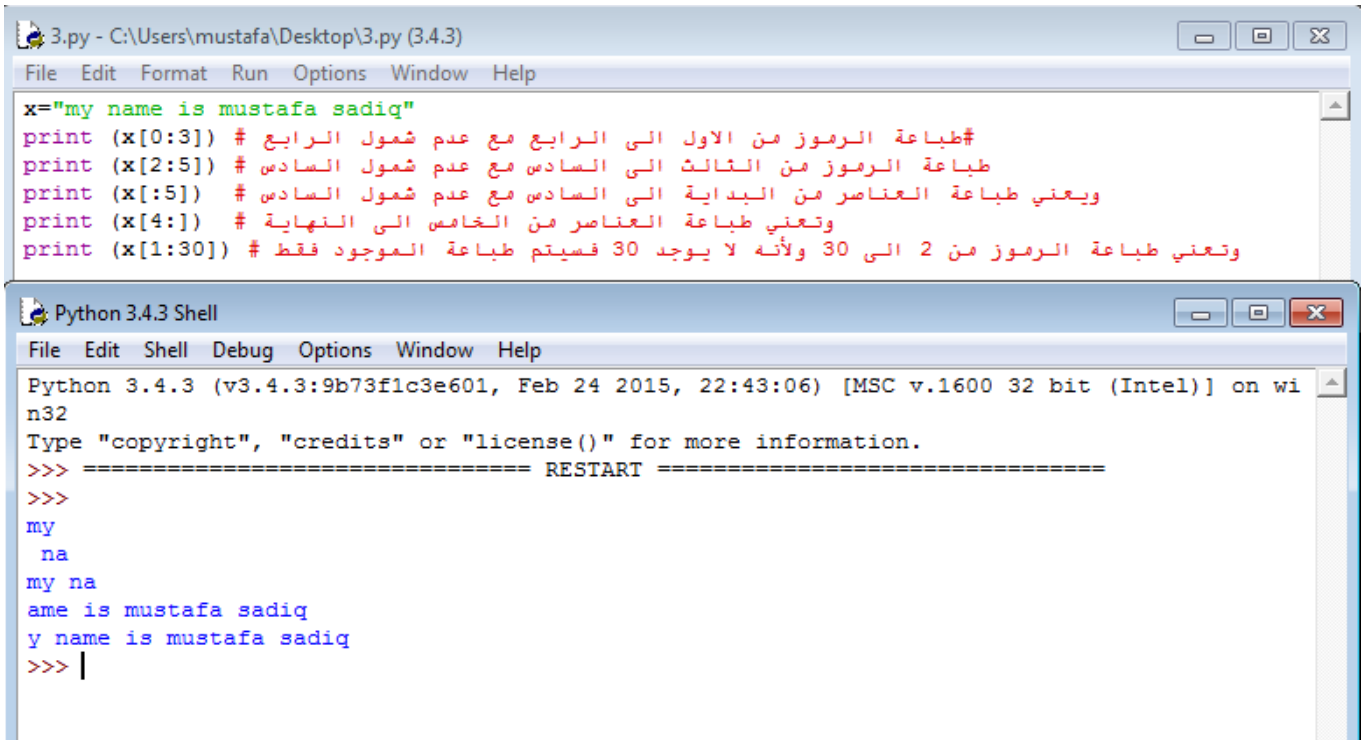


```
3.py - C:\Users\mustafa\Desktop\3.py (3
File Edit Format Run Options Win
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print (count)

Python 3.4.3 Shell
File Edit Shell Debug Options Window
Python 3.4.3 (v3.4.3:9b73f1c3e601
tel)] on win32
Type "copyright", "credits" or "l
>>> =====
>>>
3
>>> |
```

حيث قام الكود أعلاه بحساب عدد مرات ورود الرمز (a) في السلسلة الرمزية.

من الأمور المهمة الأخرى حول السلاسل الرمزية والتي تم التطرق لها بشكل مختصر في الدرس الخامس هي تشريح او تقطيع السلاسل الرمزية (string slicing) وذلك بطباعة جزء من السلسلة الرمزية يبدأ برقم معين وينتهي برقم معين وكما في المثال ادناه:



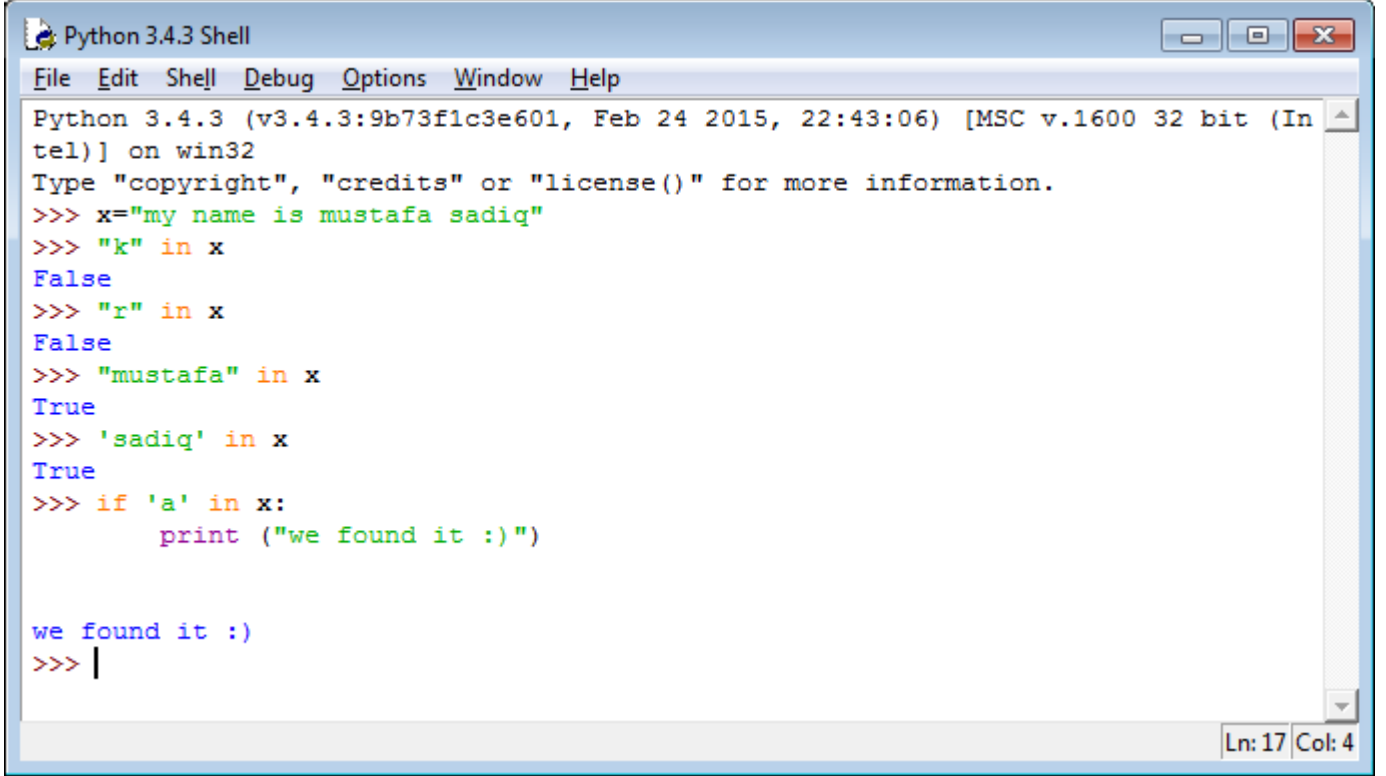
```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x="my name is mustafa sadiq"
print (x[0:3]) # طباعة الرموز من الاول الى الرابع مع عدم شعول الرابع
print (x[2:5]) # طباعة الرموز من الثالث الى السادس مع عدم شعول السادس
print (x[:5]) # يعني طباعة العناصر من البداية الى السادس مع عدم شعول السادس
print (x[4:]) # وتعني طباعة العناصر من الخامس الى النهاية
print (x[1:30]) # وتعني طباعة الرموز من 2 الى 30 ولأنه لا يوجد 30 فسيتم طباعة الموجود فقط

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on wi
n32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
my
na
my na
ame is mustafa sadiq
y name is mustafa sadiq
>>> |
```

استخدام (in) كدالة منطقية:

ذكرنا سابقاً ان عبارة (in) المحفوظة في لغة بايثون لها استخدامات كثيرة مع عبارات التكرار والشروط وهنا يمكن

استخدامها ايضاً كعبارة اختبار منطقية كما في الأمثلة التالية:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="my name is mustafa sadig"
>>> "k" in x
False
>>> "r" in x
False
>>> "mustafa" in x
True
>>> 'sadiq' in x
True
>>> if 'a' in x:
    print ("we found it :)")

we found it :)
>>> |
```

مكتبة دوال السلاسل الرمزية:

هناك الكثير من الدوال المحفوظة في لغة بايثون للتعامل مع السلاسل الرمزية وسنتطرق الى بعضها في الأمثلة

التالية:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (I
ntel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="this file is full of IMPORTANT DATA, so please: READ it CAREFULLY"
>>> y=x.lower()
>>> y
'this file is full of important data, so please: read it carefully'
>>> z=y.upper()
>>> z
'THIS FILE IS FULL OF IMPORTANT DATA, SO PLEASE: READ IT CAREFULLY'
>>> dir(x)
['_add_', '_class_', '_contains_', '_delattr_', '_dir_', '_doc_', '_
_eq_', '_format_', '_ge_', '_getattribute_', '_getitem_', '_getnewar
gs_', '_gt_', '_hash_', '_init_', '_iter_', '_le_', '_len_', '_lt
_', '_mod_', '_mul_', '_ne_', '_new_', '_reduce_', '_reduce_ex_',
'_repr_', '_rmod_', '_rmul_', '_setattr_', '_sizeof_', '_str_', '_
subclasshook_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswi
th', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha
', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable
', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketr
ans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit
', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>> type(x)
<class 'str'>
>>> type(y)
<class 'str'>
>>> type(22)
<class 'int'>
>>> type(22.44)
<class 'float'>
>>> |
```

ويمكن الاطلاع على تفاصيل أكثر ودوال أكثر عن السلاسل الرمزية على الرابط التالي:

<https://docs.python.org/2/library/stdtypes.html#string-methods>

البحث بداخل السلاسل الرمزية باستخدام دالة (find()):

من اهم الدوال المحفوظة في لغة بايثون للتعامل مع السلاسل الرمزية هي دالة البحث (find()) والتي يمكن

استخدامها للعثور على رمز او مجموعة رموز وتعيد قيمة فهرس (index) اول ظهور لتلك المجموعة او الرمز

كقيمة عددية وفي حالة عدم إيجاد الرمز المطلوب تقوم بأرجاع قيمة (-1) وكما في ادناه:


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="tell me you name in order to call you with it"
>>> y=x.find("r")
>>> y
21
>>> y*2
42
>>> z=x.find("ali")
>>> z
-1
>>> z*y
-21
>>> r=x.find("you")
>>> r
8
>>>
Ln: 17 Col: 4
```

البحث والاستبدال باستخدام دالة (replace()):

يمكن استخدام دالة (replace()) لإيجاد رمز معين او مجموعة رموز واستبدالها برمز او مجموعة رموز وكما في

ادنناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="this string is good example to replace things"
>>> y=x.replace("is", "are")
>>> y
'thare string are good example to replace things'
>>> z=x.replace("good", "bad")
>>> z
'this string is bad example to replace things'
>>> k=x.replace("example", "ex.")
>>> k
'this string is good ex. to replace things'
>>> t=x.replace("toto","atat")
>>> t
'this string is good example to replace things'
>>>
```

حيث لا يشترط كما لاحظنا ان تكون السلاسل التي نستبدلها بنفس الطول.

حذف المسافات الفارغة من بداية او نهاية السلسلة الرمزية:

كما هي عادة الكتابة باستخدام لوحة المفاتيح، تحتوي الكثير من الكلمات والجمل على مسافات فارغة قبل وبعد

الكلمات ولحذفها نستخدم الدوال التالية:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="    hello world    "
>>> y=x.lstrip()
>>> y
'hello world    '
>>> z=x.rstrip()
>>> z
'    hello world'
>>> w=x.strip()
>>> w
'hello world'
>>> |
```

ومن الجميل ان نلاحظ ان هذه الدوال لا تتلاعب بالفراغات بين الكلمات في الجملة الواحدة مما لا يؤثر على المعنى.

دالة (startswith())

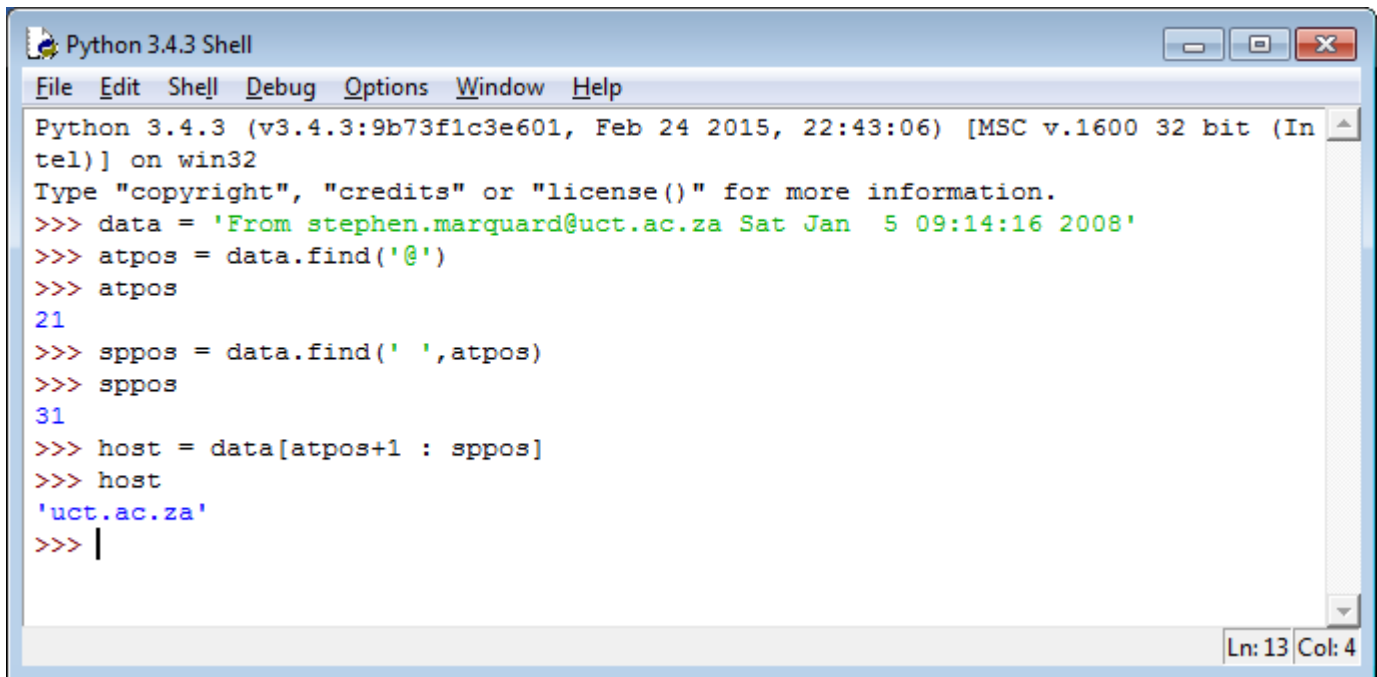
وتعمل كدالة اختبار منطقية (ترجع قيمة true or false) وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x="please, tell me your name"
>>> y=x.startswith("p")
>>> y
True
>>> z=x.startswith("please")
>>> z
True
>>> w=x.startswith("hello")
>>> w
False
>>> r=x.startswith("P")
>>> r
False
>>> |
```

حيث يتم اسناد قيمة صح (true) او خطأ (false) اعتماداً على مقارنة القيمة المعطاة داخل دالة (Startswith()) مع الرمز او مجموعة الرموز التي تبدأ بها السلسلة الرمزية الاصلية. بخصوص الحالة الأخيرة اعادت الدالة خطأ بالرغم من انها تبدأ بالحرف (p) لأن لغة بايثون حساسة لحالة الحروف (letter case sensitive) أي انها تفرق بين الحرف الكبير والصغير لذا يرجى الانتباه.

استخلاص جزء من السلسلة الرمزية يبدأ برمز معين وينتهي برمز معين آخر:

يمكن عمل ذلك بأتباع الخطوات التالية:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> atpos
21
>>> sppos = data.find(' ', atpos)
>>> sppos
31
>>> host = data[atpos+1 : sppos]
>>> host
'uct.ac.za'
>>> |
```

وهو مهم جداً في التعامل مع السلاسل الرمزية لمواقع الانترنت وكما سنرى لاحقاً ان شاء الله. واخيراً اليكم مثال محلول يوضح بعضاً من خصائص السلاسل الرمزية وكيفية التعامل معها:

6.5 Write code using find() and string slicing (see section 6.10) to extract the number at the end of the line below. Convert the extracted value to a floating point number and print it out.

Check Code

Reset Code



Exit

Grade updated on server.

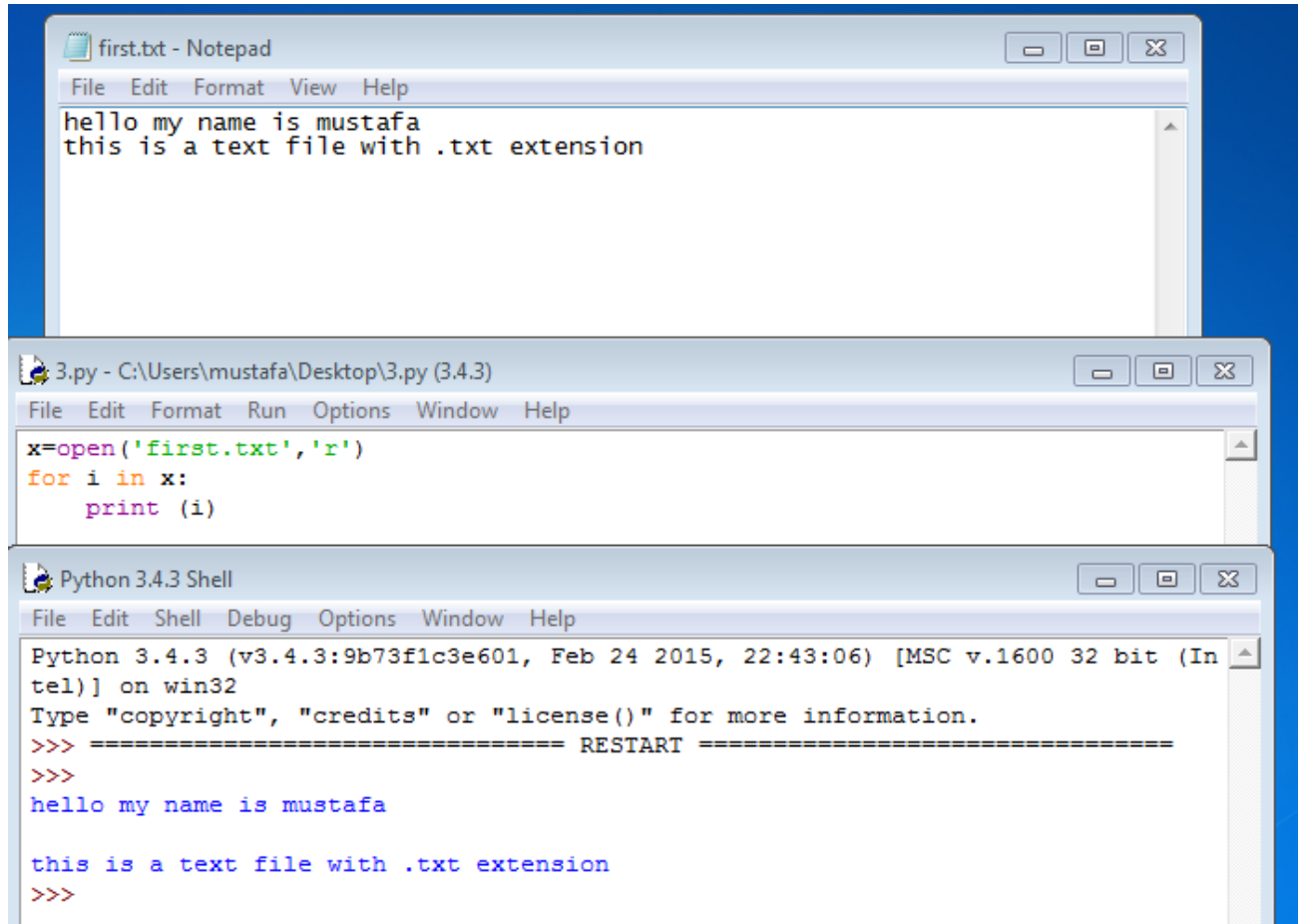
```
1 text = "X-DSPAM-Confidence: 0.8475";
2 beg=text.find('0')
3 number=text[beg:]
4 number1=float(number)
5 print number1
```

التعامل مع الملفات Files

كل ما تعاملنا معه لحد الان هو بيانات مؤقتة مخزونة في الذاكرة الرئيسية (RAM) ولم نصل بعد الى التعامل مع ملفات او بيانات مخزونة مسبقاً في القرص الصلب لاستدعائها الى المعالج لغرض معالجتها وإعادة خزنها وهو ما سنتعامل معه اليوم ان شاء الله.

الملفات: هي تجمع من البيانات المرتبطة ببعضها البعض في مكان واحد (او عدة مكانات موزعة) في الذاكرة. وسنركز في درس اليوم على التعامل مع الملفات النصية (text files) لقراءة محتوياتها والتنقل بينها والتعديل عليها او النقاط شيء معين منها للاستفادة منه في مكان اخر وكما سنرى:

قبل التعامل مع محتويات ملف نصي معين يجب ان نقوم بقراءته ويتم ذلك باستخدام الدالة: Open() ويشترط في الملف الذي نريد قراءته ان يكون مخزون في نفس مكان ملف البرنامج الذي نعمل عليه ، أي اننا ان قمنا بأنشاء ملف جديد بامتداد (.py) وقمنا بفتحه بأحد المفسرات للتعامل معه وقمنا باستدعاء الدالة (open()) فيجب ان يكون الملف الذي نريد فتحه موجود في نفس مكان خزن ملف ال (.py) وكما في المثال ادناه:



```
first.txt - Notepad
File Edit Format View Help
hello my name is mustafa
this is a text file with .txt extension

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first.txt','r')
for i in x:
    print (i)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
hello my name is mustafa

this is a text file with .txt extension
>>>
```

هنا نشرح البرنامج وكما في ادناه:

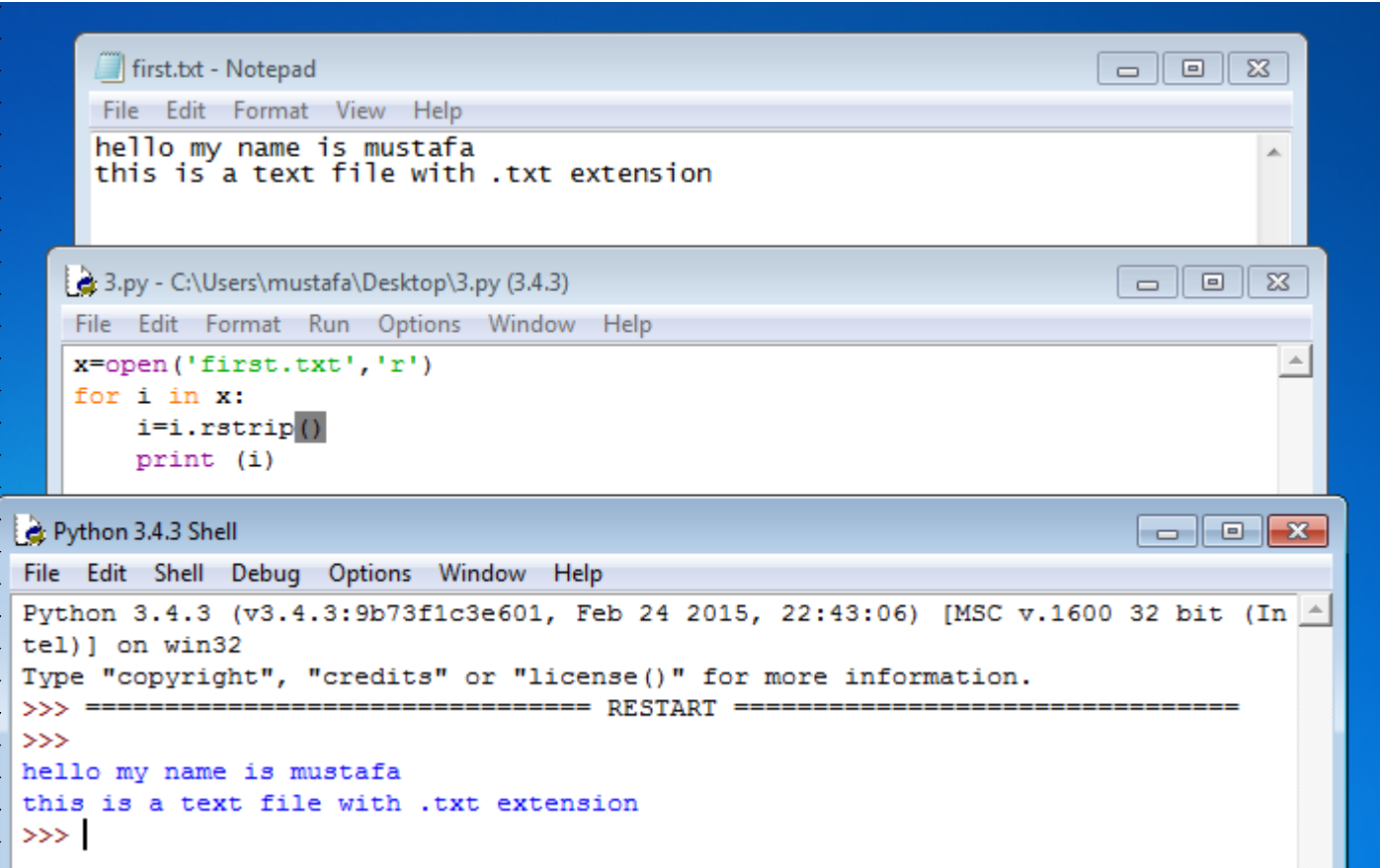
السطر الأول: اسناد ما في الملف (first.txt) الى متغير اسمه (x) مع إعطاء نمط الفتح (mode) الحرف (r) ويعني قراءة ملف (read) بدون التعديل عليه واما إذا أردنا فتحه والكتابة بداخله فنستخدم الحرف (w) من كلمة كتابة (write) وجدير بالذكر ان النمط (mode) اختياري أي اننا يمكن ان نكتب عبارة (open()) بدون عبارة الاستدعاء للنمط. ملاحظة: اسم الملف وامتداده والنمط يجب ان تكون محصورة بعلامات اقتباس مفردة او مزدوجة.

السطر الثاني: عبارة تكرار for للتنقل بين مكونات الملف المحفوظ الان في المتغير (x).

السطر الثالث: طباعة مكونات الملف وكما هو ظاهر أعلاه.

ملاحظة مهمة جداً: نلاحظ ان الملف المطبوع يختلف عن الملف الموجود في الملف النصي بوجود سطر فارغ بين السطرين في الملف والسبب في ذلك ان كل سطر في الملفات يتم استدعائه وبعده عبارة (\n) أي النزول الى سطر جديد

ولأن اخر رمز في أي سطر هو (\n) ايضاً فيحصل قفز سطرين أي ترك سطر فارغ في الوسط وللتخلص من هذه المشكلة فقط نقوم باستخدام دالة (rstrip()) وكما في المثال ادناه:



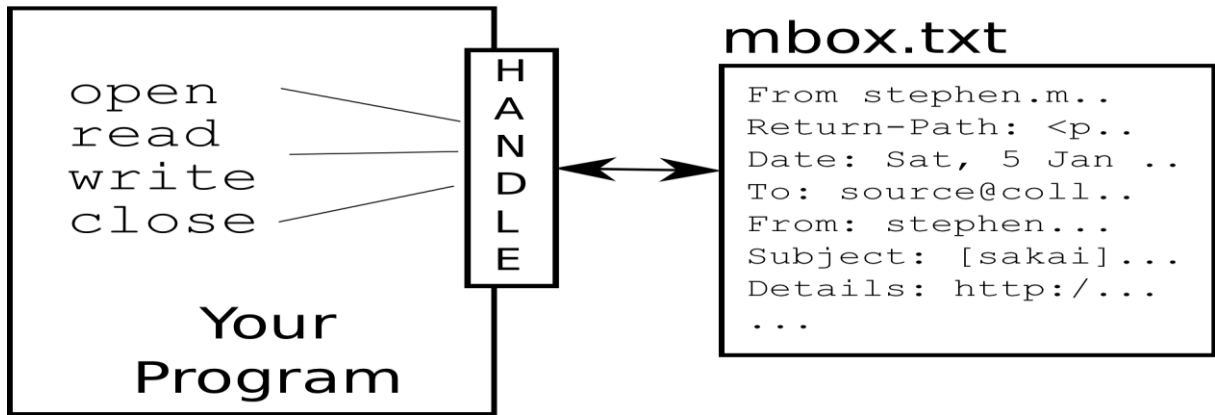
The image shows a screenshot of a Windows desktop environment. At the top, there is a Notepad window titled 'first.txt - Notepad' containing the text: 'hello my name is mustafa' followed by a blank line, and then 'this is a text file with .txt extension'. Below this is a Python 3.4.3 IDE window titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)' with the following code:

```
x=open('first.txt','r')
for i in x:
    i=i.rstrip()
print (i)
```

 At the bottom is a 'Python 3.4.3 Shell' window showing the execution of the script. The output is:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
hello my name is mustafa
this is a text file with .txt extension
>>> |
```

وهنا نلاحظ اننا استدعينا دالة قص الفراغات الى اليمين (rstrip()) التي تحدثنا عنها في الدرس السابق لتخليصنا من السطر الفارغ بين السطرين بل وكل الاسطر الفارغة في الملف (إذا كان يحتوي على الكثير من الاسطر). ملاحظة: ان دالة (open()) هي ليست لحفظ الملف كله في متغير وانما لفتح نافذة بين الذاكرة الثانوية (الهارد) والذاكرة الرئيسية (الرام) عن طريق المتغير وكما في الرسم التوضيحي التالي:



وكما يوضحه المثال ادناه:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first.txt','r')
print(x)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
<_io.TextIOWrapper name='first.txt' mode='r' encoding='cp1252'>
>>> |

```

هنا نرى ان طباعة المتغير (x) لا تعطينا محتويات الملف وانما شيء يسمى (handler) وهو مفتاح او قناة توصيل بين الذاكرة الرئيسية والثانوية ومن هنا نعلم ان الطريقة الوحيدة للوصول الى محتويات الملف باستخدام دالة (for) كما ذكرنا في المثال الأول.

في حالة محاولة فتح ملف غير موجود سنحصل على رسالة خطأ كما في ادناه:


```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first1.txt','r')
print (x)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Traceback (most recent call last):
  File "C:\Users\mustafa\Desktop\3.py", line 1, in <module>
    x=open('first1.txt','r')
FileNotFoundError: [Errno 2] No such file or directory: 'first1.txt'
>>> |
```

مرة أخرى نعود الى الرمز سطر جديد (\n) فهو يعتبر رمزاً خاصاً في لغة بايثون وكما يوضحه المثال التالي:

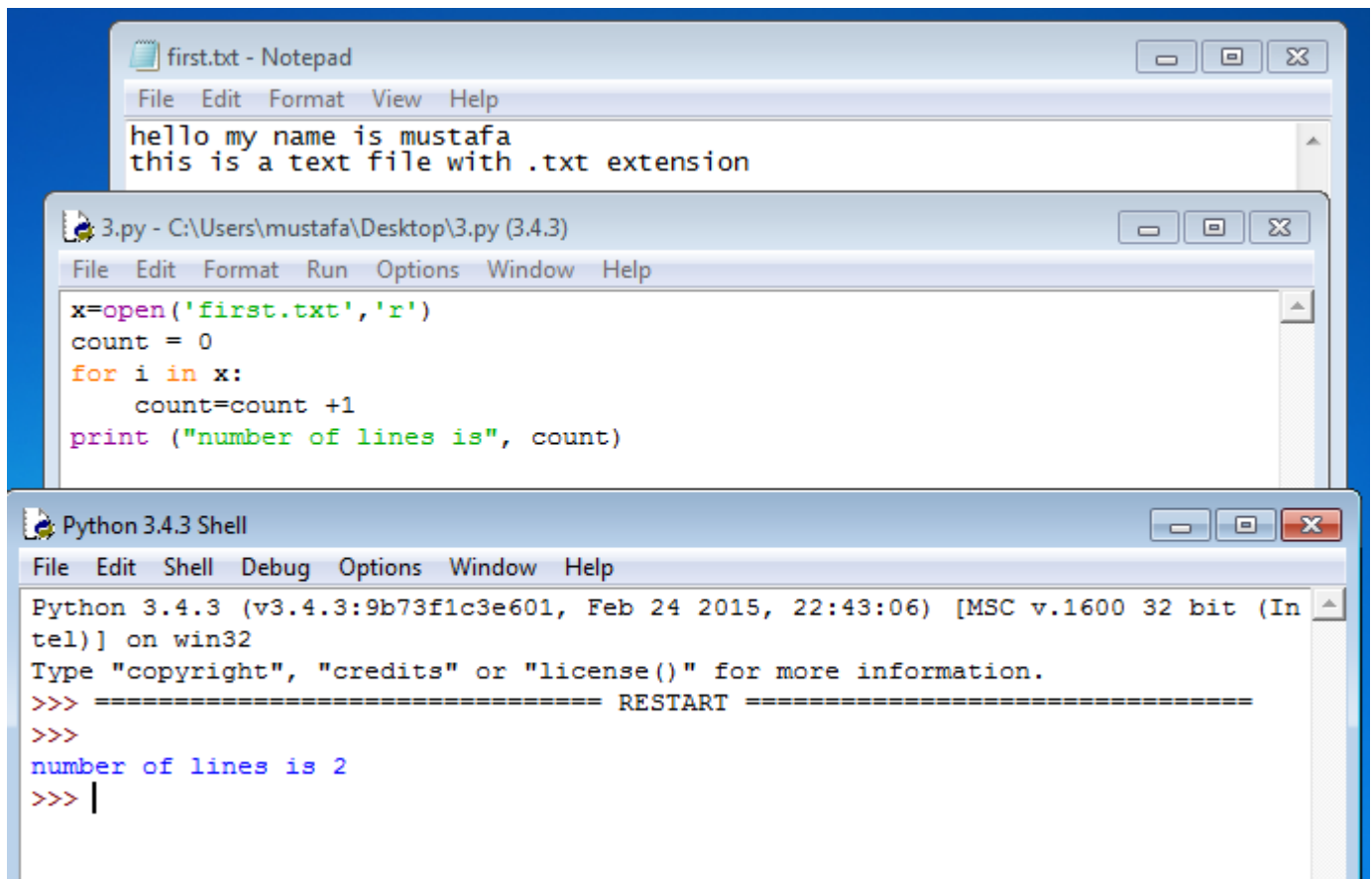
```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x='hello\nworld'
>>> print (x)
hello
world
>>> y="x\nz"
>>> y
'x\nz'
>>> print (y)
x
z
>>> len(y)
3
>>> |
```

هنا نلاحظ ان الرمز (\n) على الرغم من كونه موجود بداخل علامات اقتباس الا ان اللغة عرفت انه خاص بالنزول الى سطر جديد وهذا ما حدث عند استخدام عبارة الطباعة وكذلك حين استعلمنا عن طول (length) السلسلة الرمزية (y) قام بأخبارنا ان طولها ٣ اي انه تعامل مع (\n) على انه رمز واحد فيجب الانتباه.

ولتوضيح المعنى أكثر تصور ان كل ملف هو مجموعة من الاسطر وان كل سطر يحتوي في نهايته على رمز (\n) وهذا ما تراه لغة بايثون في الملفات النصية:

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008\nReturn-Path: <postmaster@collab.sakaiproject.org>\nDate: Sat, 5 Jan 2008 09:12:18 -0500\nTo: source@collab.sakaiproject.org\nFrom: stephen.marquard@uct.ac.za\nSubject: [sakai] svn commit: r39772 - content/branches/\n\n\nDetails: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

لحساب عدد الاسطر في ملف معين نتبع الخطوات في المثال ادناه:



```
first.txt - Notepad
File Edit Format View Help
hello my name is mustafa
this is a text file with .txt extension

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first.txt','r')
count = 0
for i in x:
    count=count +1
print ("number of lines is", count)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
number of lines is 2
>>> |
```

والان لقراءة ملف كامل نستخدم دالة (read()) وكما في ادناه:

```
*3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)*
File Edit Format Run Options Window Help
x=open('first.txt','r')
y=x.read() # read the entire file
print (len(y)) # print the length of the file = number of characters
print (y[2:]) # print the file starting from the third character

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
65
llo my name is mustafa
this is a text file with .txt extension
>>> |
```

وكما هو موضح في التعليقات امام كل سطر برمجي.

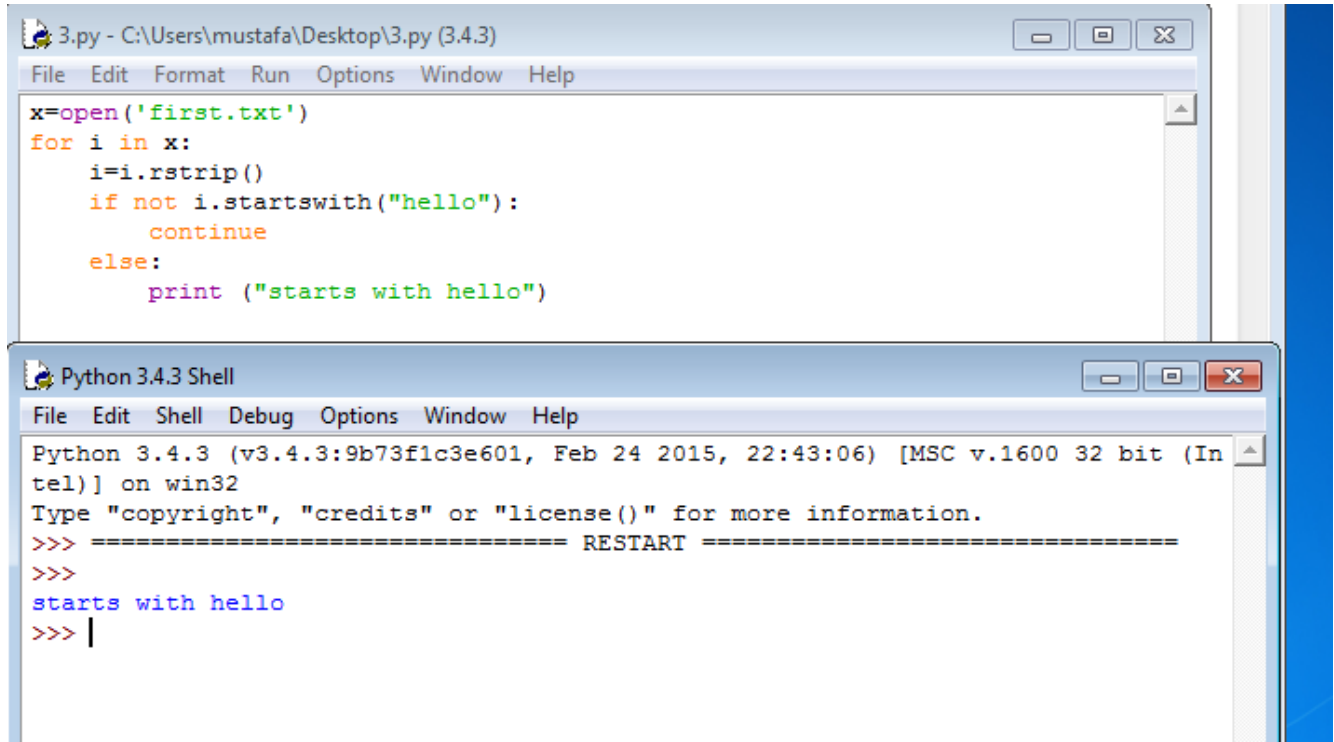
اما للبحث في داخل ملف معين فيمكن استخدام التقنية التالية:

```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first.txt')
for i in x:
    if i.startswith("hello"):
        print (i)
    else:
        print (" not there")

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
hello my name is mustafa

not there
>>> |
```

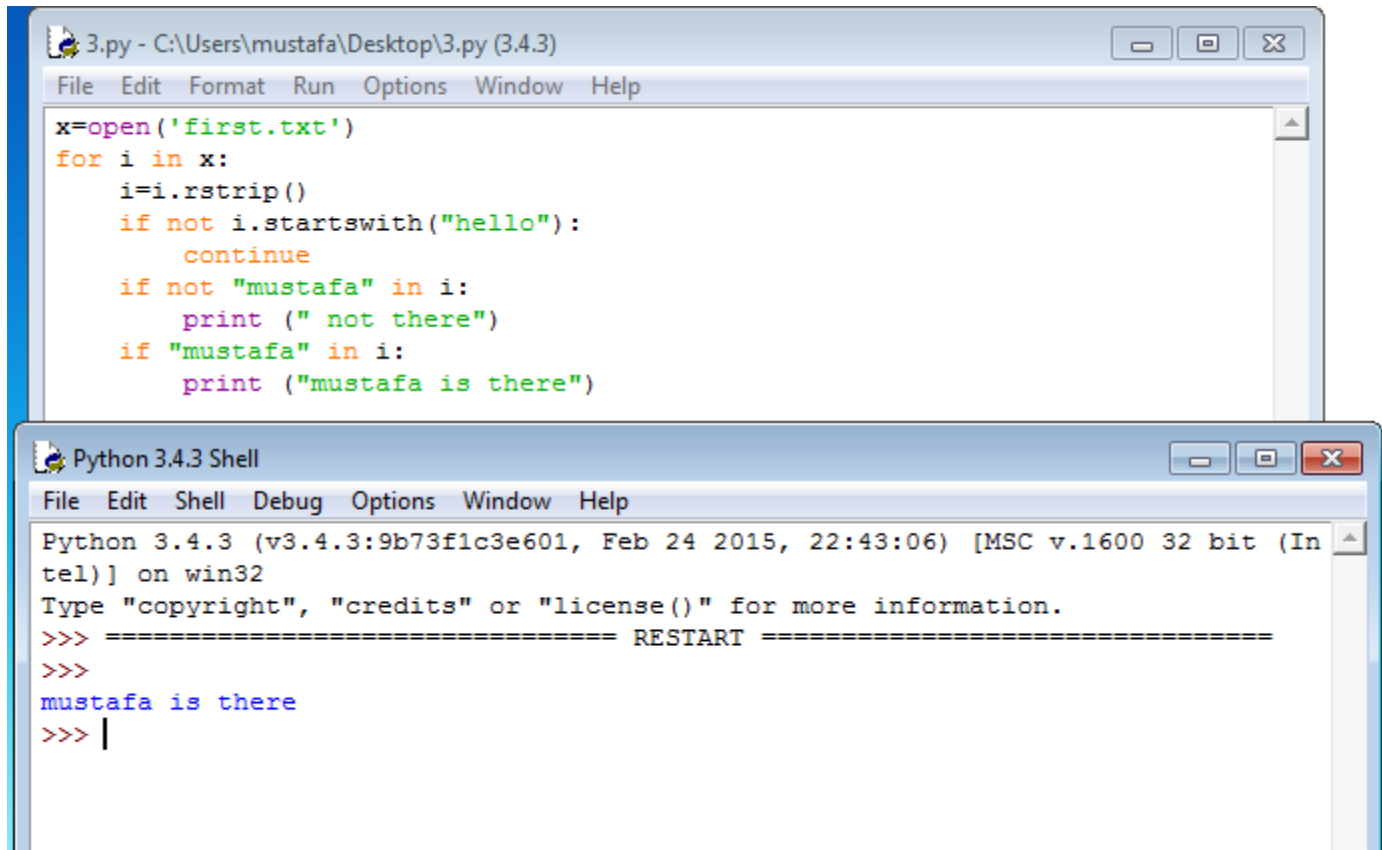
حيث قمنا باستخدام دالة (startswith()) لطباعة السطر الذي يبدأ برمز معين او كلمة معينة او مجموعة رموز.
كذلك يمكن استخدام التقنية التالية للاستمرار في الحلقة التكرارية في حالة عدم توفر شرط معين او في حالة تحققه وكما
في المثال التالي:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
x=open('first.txt')
for i in x:
    i=i.rstrip()
    if not i.startswith("hello"):
        continue
    else:
        print ("starts with hello")

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
starts with hello
>>> |
```

حيث ان الشرط هنا انه ان كان السطر لا يبدأ بعبارة (hello) فأكمل الحلقة التكرارية والا فأطبع عبارة (starts with)
(hello). كذلك يمكن البحث بداخل كل سطر من الملف باستخدام عبارة (in) وكما في ادناه:



The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
x=open('first.txt')
for i in x:
    i=i.rstrip()
    if not i.startswith("hello"):
        continue
    if not "mustafa" in i:
        print (" not there")
    if "mustafa" in i:
        print ("mustafa is there")
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution output:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
mustafa is there
>>> |
```

لحد الان كنا نتعامل مع ملف قمنا بإدخال اسمه من ضمن البرنامج وهذا ليس الوضع عادة حيث نريد ان نعطي للمستخدم قابلية قراءة أي ملف يعرف اسمه ويقوم بتنفيذ أوامر معينة عليه وكما في ادناه:

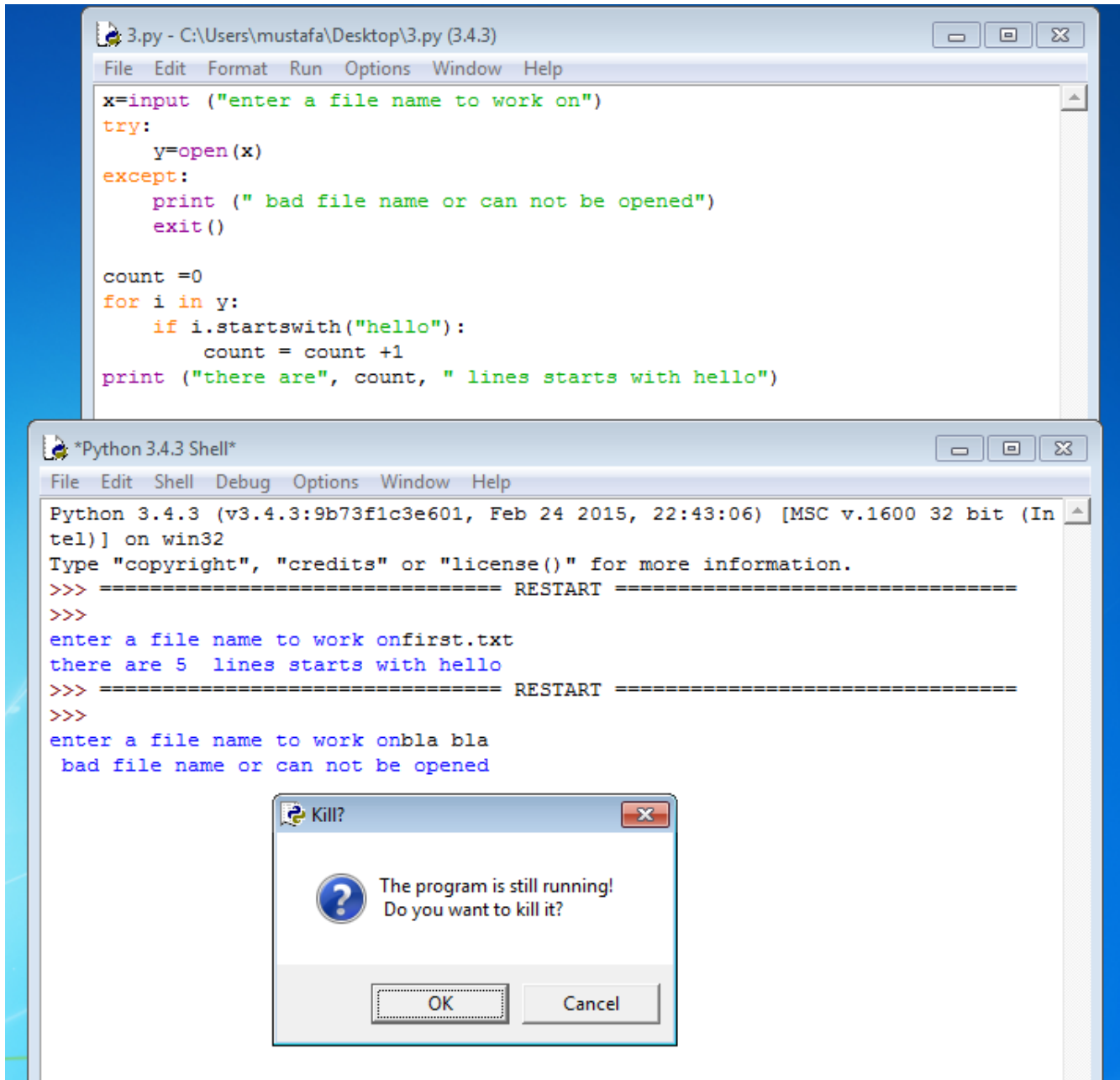
The image shows a screenshot of a Windows desktop environment. In the background, there are two windows: 'first.txt - Notepad' and '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)'. The Notepad window contains the following text: 'hello my name is mustafa', 'this is a text file with .txt extension', 'hello again', 'how are you', 'now also hello :)', 'hello', 'hello', 'haha', 'hello'. The Python script window contains the following code: 'x=input ("enter a file name to work on")', 'y=open(x)', 'count =0', 'for i in y:', 'if i.startswith("hello"):', 'count = count +1', 'print ("there are", count, " lines starts with hello")'. In the foreground, there is a 'Python 3.4.3 Shell' window. The shell displays the following output: 'Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32', 'Type "copyright", "credits" or "license()" for more information.', '==== RESTART ====', '>>>', 'enter a file name to work onfirst.txt', 'there are 5 lines starts with hello', '>>> |'.

وهنا قمنا بإدخال اسم الملف والذي يظهر اسمه ومحتوياته في الزاوية العليا اليسرى من الصورة أعلاه والبرنامج الى

اليمين والتنفيذ الى الأسفل ☺

وكما في كل البرامج السابقة يجب ان نأخذ بنظر الاعتبار ان المستخدم قد يقوم بإدخال اسم ملف خاطئ او بدون امتداد

وهنا يجب ان نحسب حساب معالجة هذا الموقف وكما في ادناه:



حيث استخدمنا عبارة (try exception) لطباعة رسالة خطأ للمستخدم وانهاء التنفيذ ويمكن عدم انهاء التنفيذ والقيام

بأي عمل اخر بحسب رغبتنا وما يريده المستخدم النهائي منا.

واخيراً ارفق لكم مجموعة من الأمثلة المحولة للتعامل مع الملفات ومحتوياتها عسى ان تنفعكم ☺

7.1 Write a program that prompts for a file name, then opens that file and reads through the file, and print the contents of the file in upper case. Use the file **words.txt** to produce the output below.

You can download the sample data at <http://www.pythonlearn.com/code/words.txt>

Check Code

Reset Code



Exit

Grade updated on server.

```
1 # Use words.txt as the file name
2 fname = raw_input("Enter file name: ")
3 fh = open('words.txt')
4 for lines in fh:
5     up=lines.upper()
6     up1=up.rstrip()
7     print up1
```


7.2 Write a program that prompts for a file name, then opens that file and reads through the file, looking for lines of the form:

```
X-DSPAM-Confidence:    0.8475
```

Count these lines and extract the floating point values from each of the lines and compute the average of those values and produce an output as shown below.

You can download the sample data at <http://www.pythonlearn.com/code/mbox-short.txt> when you are testing below enter **mbox-short.txt** as the file name.

Check Code

Reset Code



Exit

Grade updated on server.

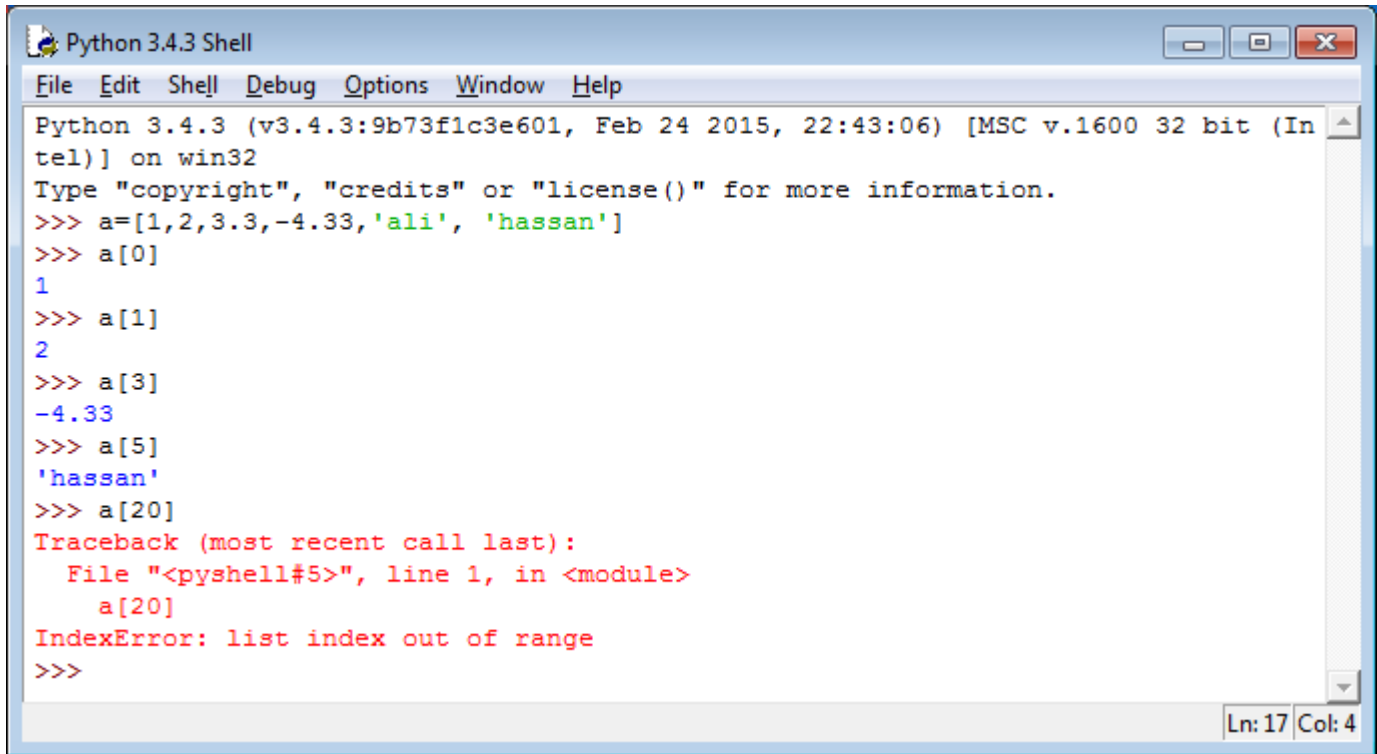
```
1 # Use the file name mbox-short.txt as the file name
2 fname = raw_input("Enter file name: ")
3 fh = open(fname)
4 count=0
5 sum1=0
6 for line in fh:
7     if not line.startswith("X-DSPAM-Confidence:") : continue
8     beg=line.find('0')
9     number=line[beg:]
10    number1=float(number)
11    count = count+1
12    sum1=sum1+number1
13 print "Average spam confidence:", sum1/count
14
```

كما في كل درس أتمنى ان تكون المادة المشروحة واضحة وانا حاضر لأي سؤال او استفسار وأتمنى التفاعل والمساعدة

في النشر لهذه الدروس لتعميم الفائدة والله الموفق لكل خير.

القوائم في لغة بايثون Lists

القوائم (lists) هي تجمع لعدد من البيانات في مكان واحد وتحت اسم واحد وهي تشبه الى حد كبير المصفوفات أحادية البعد في لغات البرمجة الأخرى وقد تم التطرق لها والتعامل معها منذ بداية الدورة والدروس الأولى وميزتها اننا لا نحتاج الى تعريفها بل نستخدمها مباشرة ولا يشترط بعناصرها ان تكون من نوع واحد فقد يكون بعضها اعداد صحيحة والأخرى عشرية والبعض الاخر رموز او سلاسل رمزية وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3.3,-4.33,'ali', 'hassan']
>>> a[0]
1
>>> a[1]
2
>>> a[3]
-4.33
>>> a[5]
'hassan'
>>> a[20]
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    a[20]
IndexError: list index out of range
>>>
```

وكما تلاحظون فما كتبناه هنا ليس جديداً في شيء فنحن قد تعاملنا مع مثل ذلك في اغلب الدروس السابقة وعرفنا كيف نقوم بإدخال القائمة محصورة بقوسين من النوع الكبير [] وتفصل بين عناصرها فارزة ولو كان ضمن عناصرها رموز او سلاسل رمزية فيجب حصرها بعلامات اقتباس مفردة او مزدوجة. وعرفنا أيضاً كيف يمكننا الوصول الى أي عنصر في القائمة وكما في أعلاه. واخيراً عرفنا ان استدعاء عنصر خارج نطاق القائمة يسبب ظهور رسالة خطأ (traceback).

كما ذكرنا فإن القائمة هي تجمع لعدة متغيرات في مكان واحد واما تعريف متغير مفرد فليس قائمة وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=1 # not a list
>>> y=2+4 # not a list
>>> z=x+y #not a list
>>> w=[x,y,z]
>>> w
[1, 6, 7]
>>> |
Ln: 9 Col: 4
```

وهنا نلاحظ ان (x,y,z) هي متغيرات ولكن (w) هي قائمة وهكذا.

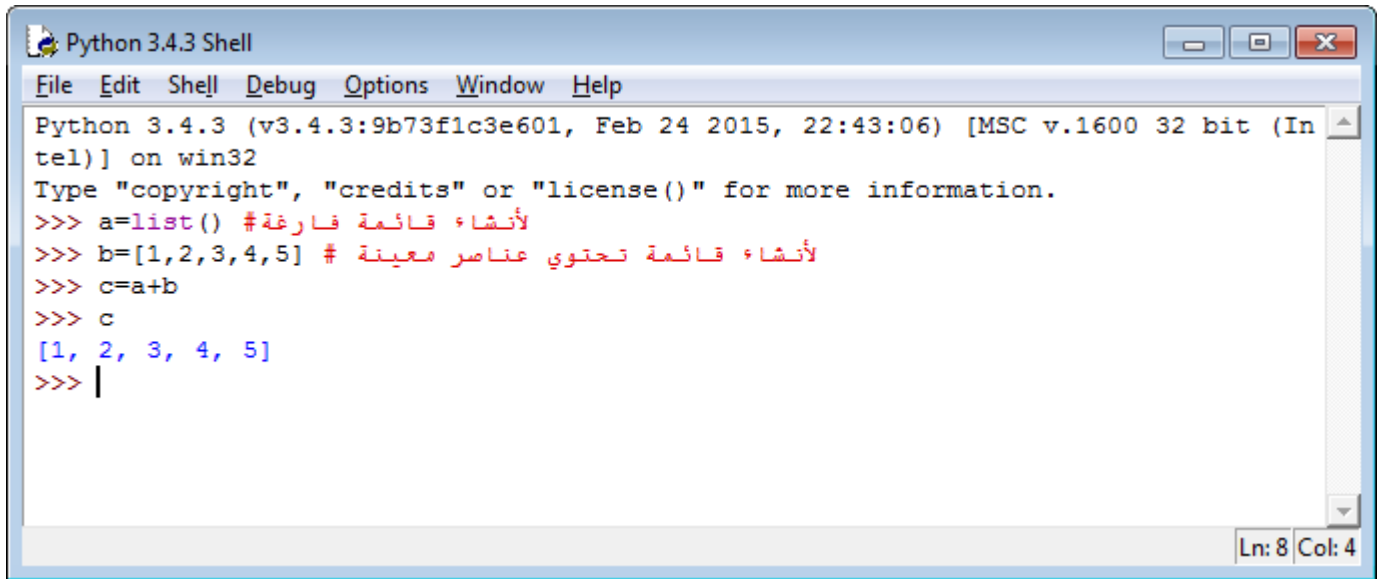
القوائم في لغة بايثون قد تكون فارغة وقد تحتوي عدد محدد او غير محدد من العناصر وقد تحتوي قوائم فرعية (sub-lists) بداخلها وكما يوضحه المثال التالي:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x=[]
>>> y=[1,2,3]
>>> z=[1,2,[4,5], 'ali', 3.3, -5.6, ['abc', 'axt', 2]]
>>> z
[1, 2, [4, 5], 'ali', 3.3, -5.6, ['abc', 'axt', 2]]
>>> w=y+z
>>> w
[1, 2, 3, 1, 2, [4, 5], 'ali', 3.3, -5.6, ['abc', 'axt', 2]]
>>> |
Ln: 11 Col: 4
```

وهنا نلاحظ ان (x) قائمة فارغة و(y) ليست فارغة و (z) قائمة تحتوي ٧ عناصر بعضها متغيرات او ثوابت وبعضها قوائم فرعية. واخيراً يمكن استخدام اشارة الجمع (+) لأرفاق (append) القوائم بعضها ببعض حيث أرفقنا القائمة (z) في نهاية القائمة (y) ووضعنا الكل في القائمة (w).

انشاء القوائم:

سبق ان تعلمنا كيفية انشاء قائمة او التعامل معها واليكم بعض الطرق لذلك:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=list() # أنشاء قائمة فارغة
>>> b=[1,2,3,4,5] # أنشاء قائمة تحتوي عناصر معينة
>>> c=a+b
>>> c
[1, 2, 3, 4, 5]
>>> |
```

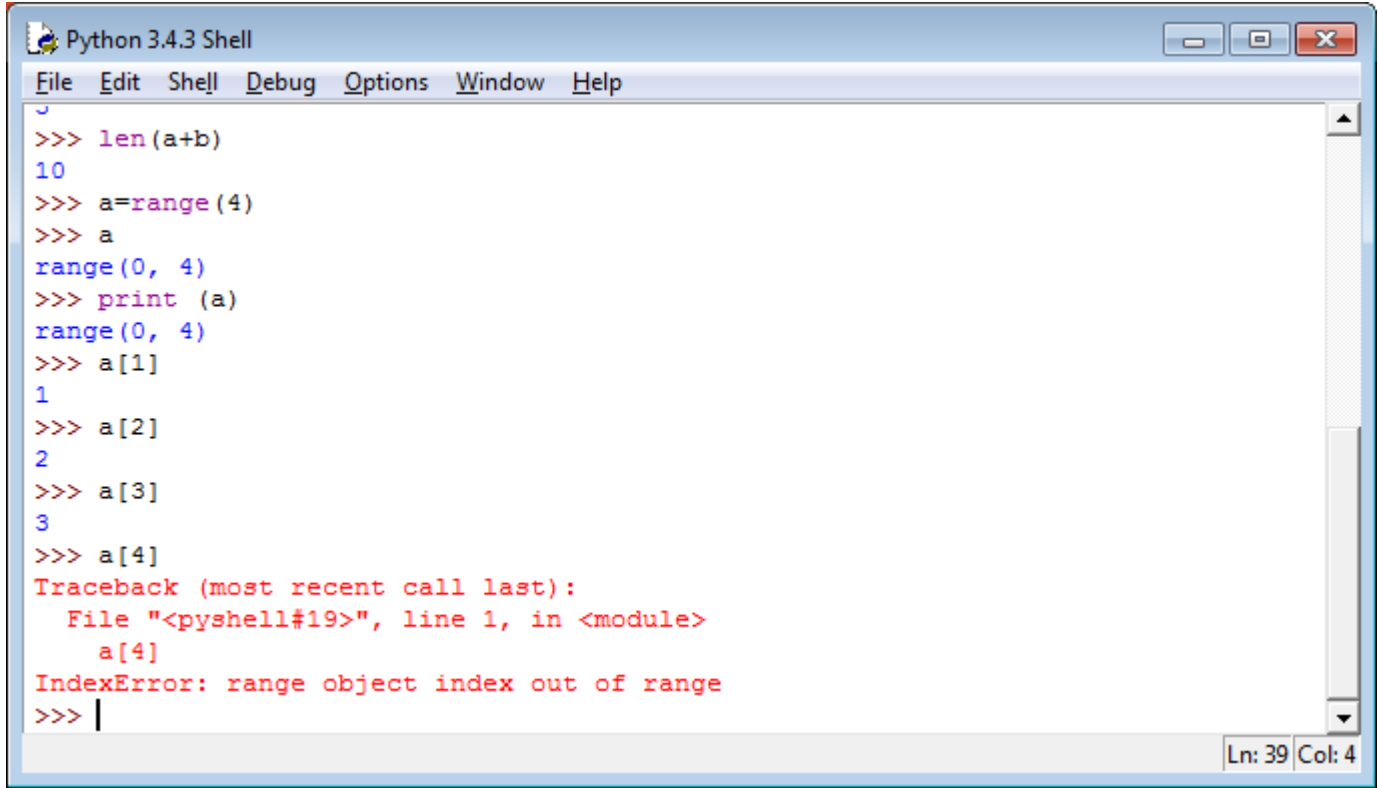
الامر الاخر المهم حول القوائم هو الفرق المهم بين قوائم الرموز وقوائم الأرقام حيث يمكن تغيير محتويات قوائم الأرقام ولكن لا يمكن التلاعب بمحتويات قوائم الرموز في البايتون (2.7) ولكن كل ذلك ممكن في البايتون (3) وما بعده وكما يوضحه المثال التالي:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5]
>>> a[2]=3.5
>>> a
[1, 2, 3.5, 4, 5]
>>> a[1]=a[1]+a[2]
>>> a
[1, 5.5, 3.5, 4, 5]
>>> b=['a','b','c','ali','ahmad']
>>> b[3]='s'
>>> b
['a', 'b', 'c', 's', 'ahmad']
>>> b[1]=b[2]+b[3]
>>> b
['a', 'cs', 'c', 's', 'ahmad']
>>>
Ln: 17 Col: 4
```

ايضاً لمعرفة طول القائمة نستخدم دالة (len()) وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> a
[1, 2, 3.5, 4, 5]
>>> a[1]=a[1]+a[2]
>>> a
[1, 5.5, 3.5, 4, 5]
>>> b=['a','b','c','ali','ahmad']
>>> b[3]='s'
>>> b
['a', 'b', 'c', 's', 'ahmad']
>>> b[1]=b[2]+b[3]
>>> b
['a', 'cs', 'c', 's', 'ahmad']
>>> len(a)
5
>>> len(b)
5
>>> len(a+b)
10
>>> |
Ln: 23 Col: 4
```

من أدوات انشاء القوائم التلقائية هي الدالة (range()) التي تستخدم لإنشاء قوائم تحتوي عدد من الرموز يساوي العدد المكتوب بداخل الدالة وكما في المثال ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> len(a+b)
10
>>> a=range(4)
>>> a
range(0, 4)
>>> print(a)
range(0, 4)
>>> a[1]
1
>>> a[2]
2
>>> a[3]
3
>>> a[4]
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    a[4]
IndexError: range object index out of range
>>> |
```

Ln: 39 Col: 4

كما ذكرنا سابقاً ايضاً يمكن ايضاً تشريح (slicing) القوائم بطباعة جزء او أجزاء منها وكما في المثال ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5,6,7,8,9,0]
>>> a[1:3]
[2, 3]
>>> a[:5]
[1, 2, 3, 4, 5]
>>> a[6:]
[7, 8, 9, 0]
>>> a[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> a[3:40]
[4, 5, 6, 7, 8, 9, 0]
>>> |
```

وقد تم سابقاً شرح معنى كل من الايعازات أعلاه.

دوال القوائم:

كما ذكرنا سابقاً فأحد أبرز الدوال المستخدمة مع القوائم هي دالة (list()) لإنشاء دالة فارغة واما دالة (type()) فنستخدم لإرجاع نوع القائمة او المتغير كما رأينا سابقاً واخيراً هناك دالة المجلد (dir()) والتي تعرض كل الدوال المتوفرة لنوع معين من المتغيرات او القوائم وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> a[3:40]
[4, 5, 6, 7, 8, 9, 0]
>>> aaa=list()
>>> type(a)
<class 'list'>
>>> type(aaa)
<class 'list'>
>>> dir(aaa)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'_doc_', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__',
'_gt_', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le_',
'_len_', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_e
x_', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__s
izeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'ex
tend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Ln: 21 Col: 4

هنا قد يتساءل البعض عن كيفية الاستفادة من دالة (list()) والتي تبني قائمة فارغة؟ والجواب يمكن معرفته من المثال

التالي:


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=list()
>>> a.append(1)
>>> a
[1]
>>> a.append('hello')
>>> a
[1, 'hello']
>>> a.append(333,44.6,'ali')
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    a.append(333,44.6,'ali')
TypeError: append() takes exactly one argument (3 given)
>>> a.append(22.3)
>>> a
[1, 'hello', 22.3]
>>> a+a
[1, 'hello', 22.3, 1, 'hello', 22.3]
>>> |
```

استخدام أدوات (in, not in) لاختبار وجود او عدم وجود عنصر معين في القائمة: يوضحه المثال التالي:

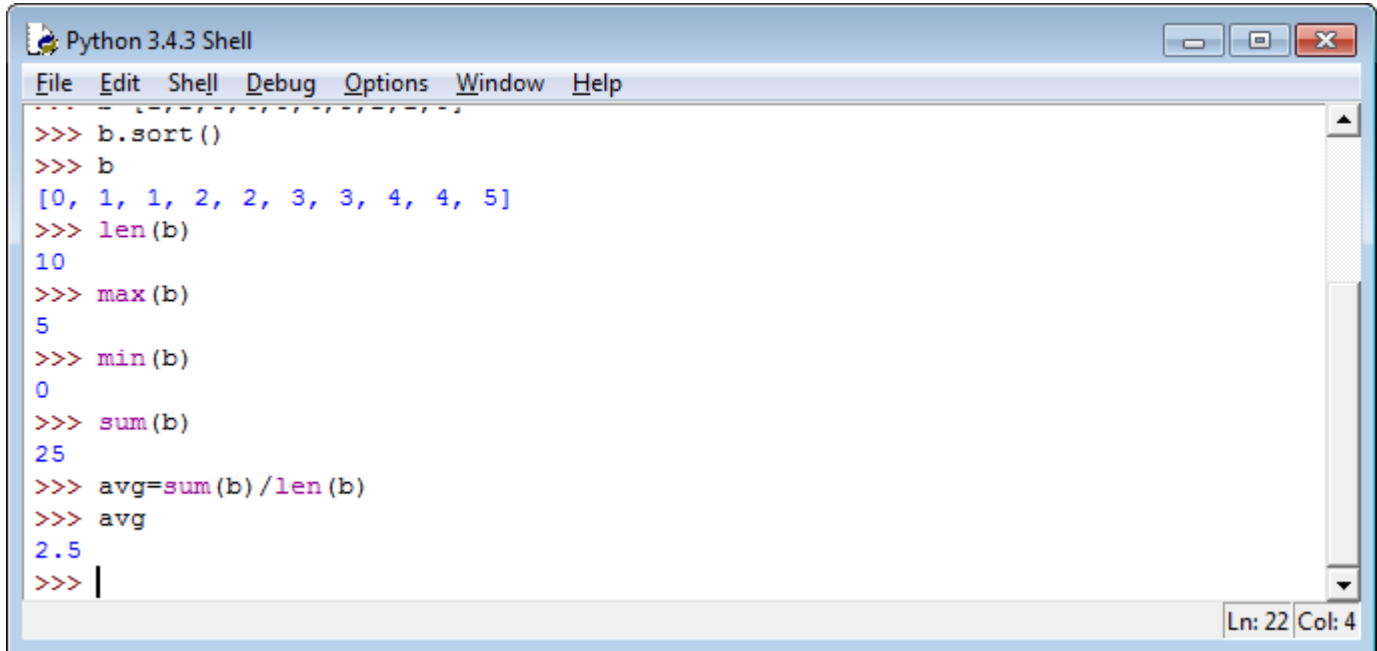
```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,'ali',2,'ajmad',33.3]
>>> i in a
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    i in a
NameError: name 'i' is not defined
>>> 1 in a
True
>>> ali in a
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    ali in a
NameError: name 'ali' is not defined
>>> 'ali' in a
True
>>> 33.3 not in a
False
>>> 33.3 in a
True
>>>
```

كيفية ترتيب قائمة اجدياً للرموز والسلاسل الرمزية ومن الصغير للكبير بالنسبة للأرقام:

باستخدام دالة (sort()) وكما في ادناه:

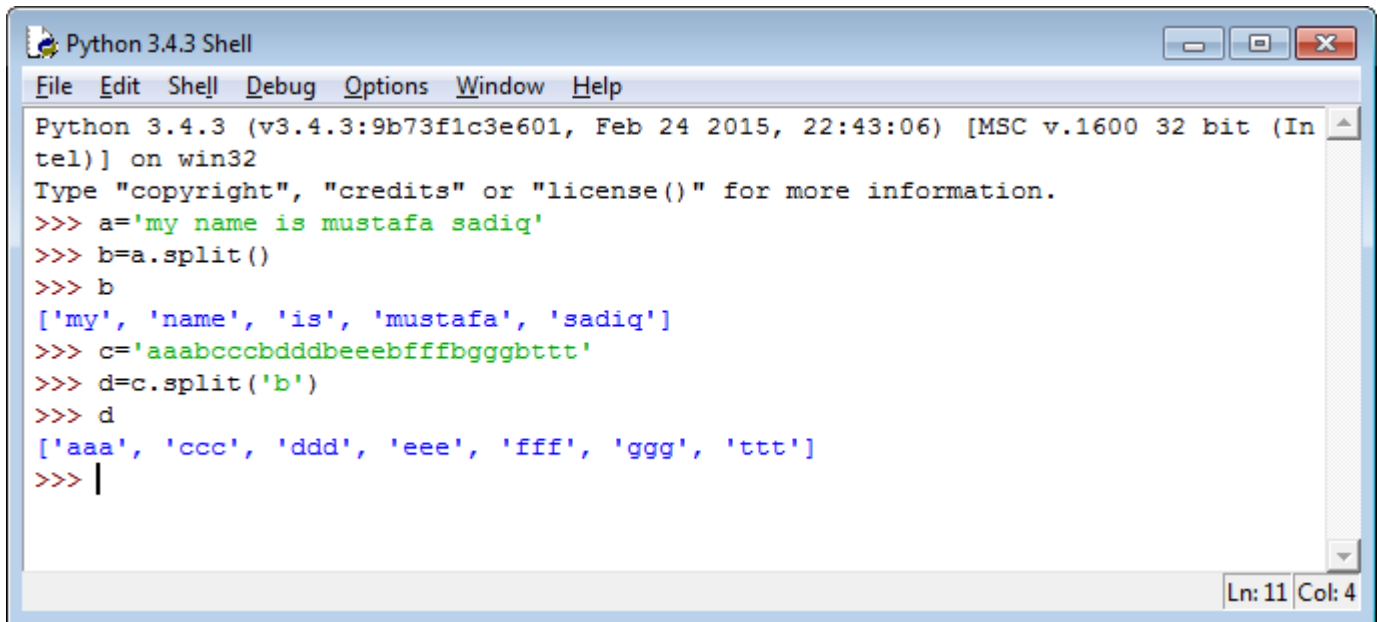
```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=['ahmad', 'zaid', 'bilal', 'suha', 'mustafa']
>>> a.sort()
>>> a
['ahmad', 'bilal', 'mustafa', 'suha', 'zaid']
>>> b=[1,2,3,4,5,4,3,2,1,0]
>>> b.sort()
>>> b
[0, 1, 1, 2, 2, 3, 3, 4, 4, 5]
>>> |
```

بعض الدوال المحفوظة في لغة بايثون للتعامل مع القوائم:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> b.sort()
>>> b
[0, 1, 1, 2, 2, 3, 3, 4, 4, 5]
>>> len(b)
10
>>> max(b)
5
>>> min(b)
0
>>> sum(b)
25
>>> avg=sum(b)/len(b)
>>> avg
2.5
>>> |
Ln: 22 Col: 4
```

تجزئة السلاسل الرمزية الى عناصرها وتحويلها الى قوائم:
ويتم ذلك باستخدام دالة (`split()`) وكما يوضحها المثال التالي:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a='my name is mustafa sadiq'
>>> b=a.split()
>>> b
['my', 'name', 'is', 'mustafa', 'sadiq']
>>> c='aaabcccbbdddbeeebfffbgggbttt'
>>> d=c.split('b')
>>> d
['aaa', 'ccc', 'ddd', 'eee', 'fff', 'ggg', 'ttt']
>>> |
Ln: 11 Col: 4
```

هنا قمنا بتعريف متغير اسمه (a) ونوعه سلسلة رمزية ثم قمنا بتجزئته (split()) ووضع الأجزاء على شكل قائمة في المتغير (b) ثم قمنا بعمل نفس الشيء مع ال (c) وال (d) الا ان الفرق الوحيد اننا جزئنا ال (a) على أساس المسافات الفارغة (spaces) وهو ما يحصل حين نترك دالة (Split) فارغة واما مع ال (c) فقد قمنا بالتجزئة اعتماداً على الحرف (b) حيث يتم تجزئة السلسلة الرمزية الى مجموعة سلاسل او مجموعة عناصر في قائمة عند كل ذكر للحرف (b) وكما شاهدنا أعلاه.

ملاحظة: عند ترك دالة (split) فارغة فإن أي عدد من الفراغات (spaces) بين الحروف والكلمات يعتبر مسافة واحدة ويتم التجزئة على أساسها فيجب الانتباه.

واخيراً اليكم مثال عن كيفية التعامل مع أجزاء النصوص والسلاسل الرمزية وكيفية الوصول الى ادق التفاصيل باستخدام دوال القوائم والتجزئة:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> word=line.split()
>>> print (word)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>> email=word[1]
>>> email
'stephen.marquard@uct.ac.za'
>>> extension=email.split('@')
>>> extension
['stephen.marquard', 'uct.ac.za']
>>> extension[1]
'uct.ac.za'
>>> |
```

وكما هي عادتنا نختم درسنا بمجموعة من الأمثلة المحلولة للفائدة.

8.4 Open the file **romeo.txt** and read it line by line. For each line, split the line into a list of words using the **split()** function. The program should build a list of words. For each word on each line check to see if the word is already in the list and if not append it to the list. When the program completes, sort and print the resulting words in alphabetical order. You can download the sample data at <http://www.pythonlearn.com/code/romeo.txt>

Check Code

Reset Code



Exit

Grade updated on server.

```
1 fname = raw_input("Enter file name: ")
2 fh = open(fname)
3 lst = list()
4 for line in fh:
5     words=line.split()
6     for word in words:
7         if word not in lst:
8             lst.append(word)
9         else: continue
10 lst.sort()
11 print lst
12
```

8.5 Open the file **mbox-short.txt** and read it line by line. When you find a line that starts with 'From ' like the following line:

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

You will parse the From line using `split()` and print out the second word in the line (i.e. the entire address of the person who sent the message). Then print out a count at the end.

Hint: make sure not to include the lines that start with 'From:'.

You can download the sample data at <http://www.pythonlearn.com/code/mbox-short.txt>

Check Code

Reset Code



Exit

Grade updated on server.

```
1 fname = raw_input("Enter file name: ")
2 if len(fname) < 1 : fname = "mbox-short.txt"
3
4 fh = open(fname)
5 count = 0
6 for line in fh:
7     if line.startswith("From:"): continue
8     if line.startswith("From"):
9         address=line.split()
10        count=count+1
11        print address[1]
12
13 print "There were", count, "lines in the file with From as the first wor
14
```

المجلدات او القواميس فى لغة بايثون Dictionaries

وهي أحد أنواع البيانات او مجاميع البيانات (collections) والتي تسمح بجمع مجموعة غير مرتبة من الأسماء والقيم (values and keys) ووضعها تحت اسم واحد وهذا هو فرقها الرئيسي عن القوائم التي تتميز بترتيب عناصرها وهي أكثر مرونة من القوائم التي سبق شرحها ويمكن التعبير عن المجلد بالصورة التوضيحية التالية:



حيث ان المجلد يشبه الحقيبة التي تحتوي مجموعة من الأشياء وبأعداد مختلفة والهدف الرئيسي من إيجاد المجلدات في لغة بايثون لتسهيل عملية حساب عدد مرات تكرار عنصر معين (رمز او رقم او سلسلة رمزية) في ملف او سطر من ملف نصي.

ويمكن تعريف المجلد كما في المثال ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> classroom=dict()
>>> classroom['desks']=20
>>> classroom['fans']=4
>>> classroom['students']=40
>>> classroom['teacher']=1
>>> classroom['blackboard']=2
>>> classroom
{'students': 40, 'desks': 20, 'blackboard': 2, 'teacher': 1, 'fans': 4}
>>> classroom['teacher']
1
>>> classroom['desks']
20
>>> classroom['pens']
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    classroom['pens']
KeyError: 'pens'
>>> |
```

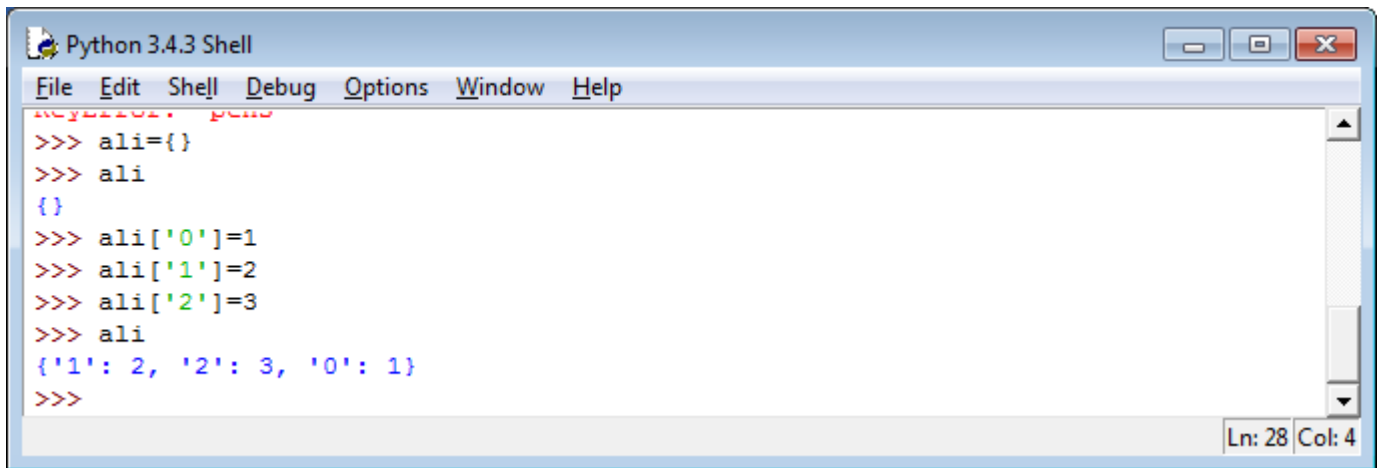
وهنا عرفنا مجلد اسمه (classroom) باستخدام دالة (dict()) وقمنا بإضافة عناصره واحداً بعد الآخر ثم حين طلبنا طباعته جاءت النتيجة كما هو متوقع بترتيب مغاير لما تم إدخاله وهو امر يختلف عما اعتدنا عليه في القوائم وبعدها ومن مميزات المجلدات عملها كقواعد بيانات للغة بايثون تمكننا من الوصول الى أي عنصر بمعرفة مفتاحه (the key) ليرجع لنا القيمة المحفوظة بداخله (the value) وحين طلبنا قيمة لمفتاح غير معرف جاءت رسالة الخطأ (traceback) لتخبرنا ان المفتاح (pens) غير معرف.

والان لمعرفة الفرق بين القوائم والمجلدات أكثر لاحظوا المقارنة التالية:

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print lst
[21, 183]
>>> lst[0] = 23
>>> print lst
[23, 183]

>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print ddd
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print ddd
{'course': 182, 'age': 23}
```

والان لأنشاء قائمة فارغة عرفنا اننا نستطيع فعل ذلك باستخدام دالة (list()) كما في الصورة أعلاه ولأنشاء مجلد فارغ فقط نستخدم الدالة (dict()) ولكن يمكن أيضاً انشاء مجلد فارغ باستخدام اقواس المجموعة فارغة وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> ali={}
>>> ali
{}
>>> ali['0']=1
>>> ali['1']=2
>>> ali['2']=3
>>> ali
{'1': 2, '2': 3, '0': 1}
>>>
```

كما ذكرنا قبل قليل فإن اهم تطبيق للمجلدات معرفة عدد مرات تكرار رقم او حرف او سلسلة رمزية في سطر او ملف نصي وهذا هو ما سنركز عليه في شرحنا لهذا الدرس علماً ان المجلدات يمكن ان تستخدم في امكان أخرى عديدة وحسب حاجة المبرمج.

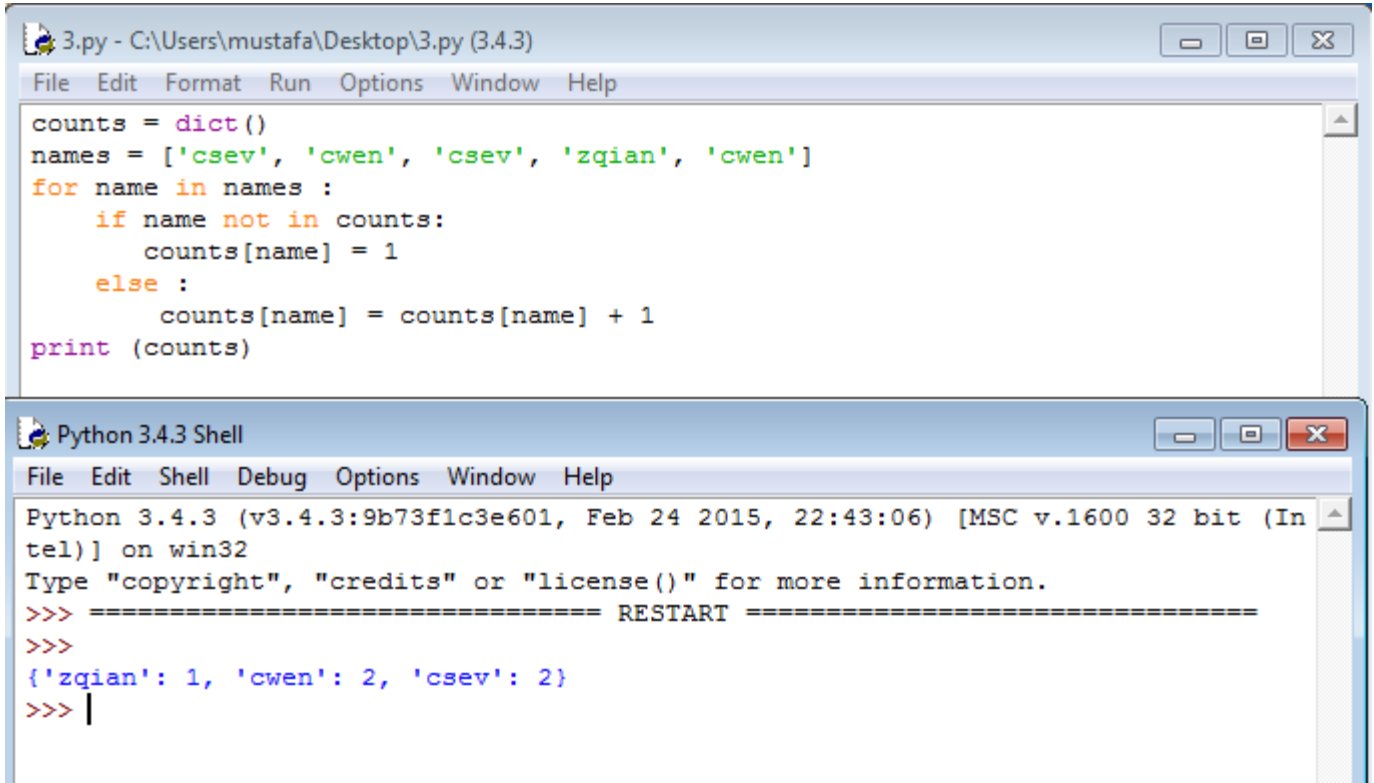
والان إذا ان هناك قائمة من الأسماء وارادنا معرفة أي اسم تكرر أكثر من الأسماء الأخرى فكل ما علينا فعله هو انشاء مجلد وتسميه مفاتيح مشابهة للأسماء واطافة ١ الى قيمة الاسم كلما تكرر ذكره وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> names=dict()
>>> names['ali']=1
>>> names['mustafa']=1
>>> names
{'mustafa': 1, 'ali': 1}
>>> names['huda']=1
>>> names['ali']=names['ali']+1
|
SyntaxError: EOL while scanning string literal
>>> names['ali']=names['ali']+1
>>> names['huda']=names['huda']+2
>>> names
{'huda': 3, 'mustafa': 1, 'ali': 2}
>>>
Ln: 10 Col: 0
```

يمكن أيضاً استخدام دالة (in) كدالة منطقية لمعرفة هل يوجد مفتاح معين في المجلد او لا وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> names['ali']=names['ali']+1
>>> names['huda']=names['huda']+2
>>> names
{'huda': 3, 'mustafa': 1, 'ali': 2}
>>> "ali" in names
True
>>> 'huda' not in names
False
>>> 'huda' in names
True
>>> 'roa' in names
False
>>> |
Ln: 24 Col: 4
```

والان نعود الى مثالنا عن عدد مرات تكرار اسم معين. حيث لاحظنا في الحالة السابقة اننا نقوم بإضافة الأسماء وعدد مرات تكرارها يدوياً كل مرة وهذه طريقة غير مجدية برمجياً خصوصاً ان كنا نتعامل مع ملف يحتوي الالف الكلمات والمفاتيح والقيم وهنا نستخدم التركيب التالي:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print (counts)

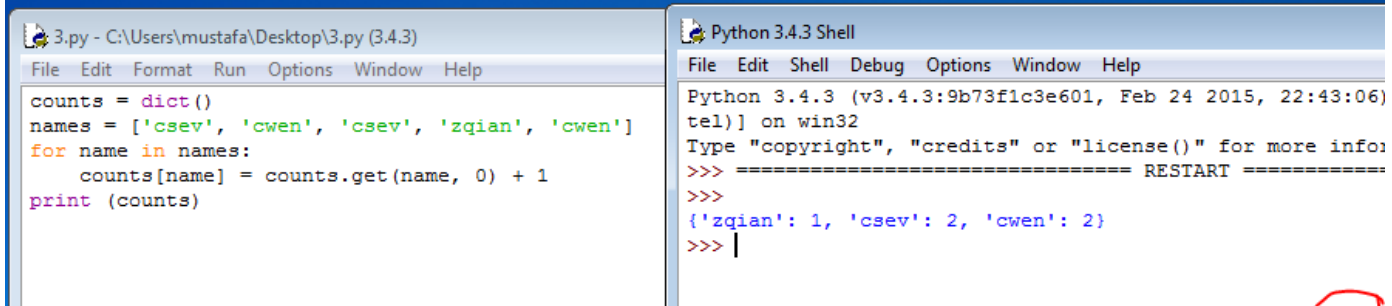
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
{'zqian': 1, 'cwen': 2, 'csev': 2}
>>> |
```

من المثال أعلاه نلاحظ اننا نستطيع استخدام دالة (for) للتنقل بين عناصر المجلد كما في القوائم تماماً مستخدمين اسمها وسيقوم المفسر باستخدام قيم المفاتيح (keys) للتنقل بين عناصر المجلد وهنا قلنا للمفسر انه لكل اسم (name) وهو اسم متغير عشوائي موجود في مجلد الأسماء: ان كان غير موجود في قائمة الاعداد السابقة فأجعل قيمته ١ والا فأضف الى قيمته السابقة ١ وبذلك نكون قد قمنا بعد كل الأسماء ومعرفة عدد مرات تكرار كل منها.

والان لأن السيناريو أعلاه يتكرر بكثرة عند استخدام المجلدات فقد تم بناء دالة خاصة فقط تقوم بكل ما ذكر أعلاه واسم هذه الدالة (get()) وتقوم هذه الدالة بما يلي:

إذا كان المفتاح موجوداً في القائمة المعدة مسبقاً تقوم بإضافة واحد له.

إذا كان المفتاح غير موجود في القائمة نقوم بأسناد قيمة افتراضية (default value) له ونحن من نقوم بإعطاء القيمة الافتراضية وبحسب الحاجة وكما في ادناه:



The screenshot shows a Python IDE with two windows. The left window is titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)' and contains the following code:

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names:
    counts[name] = counts.get(name, 0) + 1
print (counts)
```

The right window is titled 'Python 3.4.3 Shell' and shows the output of the code:

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06)
tel) on win32
Type "copyright", "credits" or "license()" for more info
>>> ===== RESTART =====
>>>
{'zqian': 1, 'csev': 2, 'cwen': 2}
>>> |
```

code result ☺

حيث تقوم الدالة هنا بعد الأسماء وعدد مرات تكرار كل منها وتضيف القيمة الافتراضية للاسم الغير مذكور سابقاً (0).

والان الصيغة العامة لحساب عدد مرات تكرار اسم معين او سلسلة رمزية معينة في المثال التالي:

The image shows two windows from a Python IDE. The top window, titled '3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)', contains the following Python code:

```
counts = dict()
print ('Enter a line of text:')
line =input('')

words = line.split()

print ('Words:', words)

print ('Counting...')
for word in words:
    counts[word] = counts.get(word,0) + 1
print ('Counts', counts)
```

The bottom window, titled 'Python 3.4.3 Shell', shows the execution of the script. It displays the prompt 'Enter a line of text:', the user input 'i am mustafa sadiq from iraq and I am happy to study python and teach it to you', the resulting list of words, and the final dictionary of word counts.

```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Enter a line of text:
i am mustafa sadiq from iraq and I am happy to study python and teach it to you
Words: ['i', 'am', 'mustafa', 'sadiq', 'from', 'iraq', 'and', 'I', 'am', 'happy', 'to', 'study', 'python', 'and', 'teach', 'it', 'to', 'you']
Counting...
Counts {'am': 2, 'you': 1, 'iraq': 1, 'python': 1, 'and': 2, 'to': 2, 'mustafa': 1, 'from': 1, 'sadiq': 1, 'teach': 1, 'i': 1, 'it': 1, 'I': 1, 'happy': 1, 'study': 1}
>>> |
```

والان لنشرح البرنامج خطوة بخطوة:

السطر الأول: انشاء مجلد فارغ

السطر الثاني: طباعة عبارة على الشاشة تطلب من المستخدم ادخال سطر من الكلمات

السطر الثالث: استخدام عبارة الادخال للسماح للمستخدم بإدخال سطر البيانات.

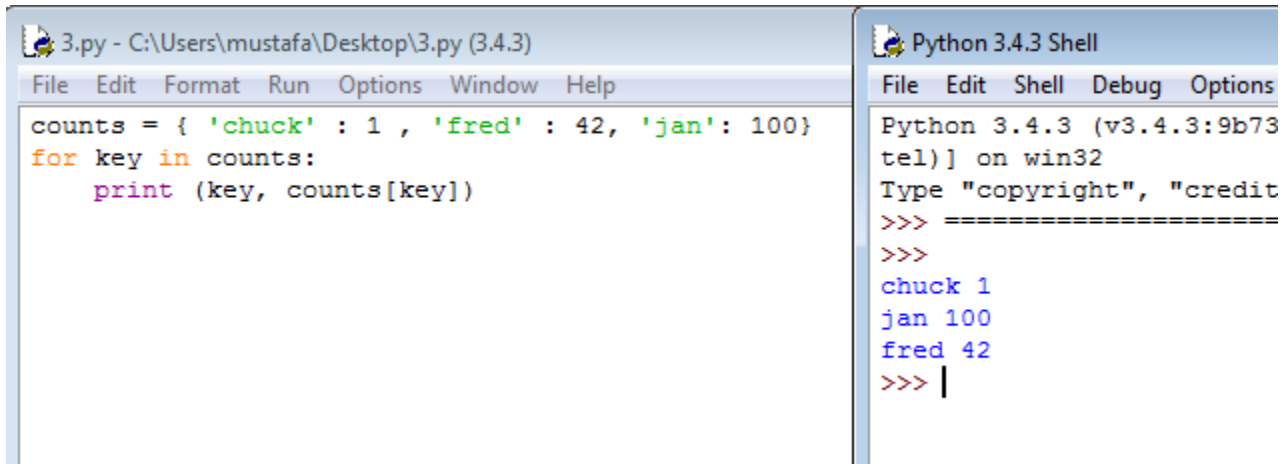
السطر الرابع: استخدام عبارة تجزئة السطر الى كلمات (split()).

السطر الخامس: طباعة قائمة الكلمات التي تكون السطر المدخل.

السطر السادس: طباعة عبارة (counting ...).

السطر السابع والثامن والتاسع: عملية العد كما ذكرناها سابقاً وطباعة النتائج

وكما ذكرنا سابقاً فإن عبارة (for) يمكن ان تستخدم للتنقل بين مكونات المجلد وكما في ادناه:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for key in counts:
    print (key, counts[key])

Python 3.4.3 Shell
File Edit Shell Debug Options
Python 3.4.3 (v3.4.3:9b73
tel) on win32
Type "copyright", "credit
>>> =====
>>>
chuck 1
jan 100
fred 42
>>> |
```

واما لاسترجاع مكونات المجلد كقوائم منفصلة للمفاتيح (keys) او القيم (values) او الأزواج المرتبة للمفاتيح والقيم

(items) نستخدم العبارات التالية:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> names=dict()
>>> names["ali"]=5
>>> names["hassan"]=6
>>> names["hussain"]=7
>>> names["khalid"]=22
>>> names
{'hassan': 6, 'hussain': 7, 'ali': 5, 'khalid': 22}
>>> names1=names.keys()
>>> names1
dict_keys(['hassan', 'hussain', 'ali', 'khalid'])
>>> names2=names.values()
>>> names2
dict_values([6, 7, 5, 22])
>>> names3=names.items()
>>> names3
dict_items([('hassan', 6), ('hussain', 7), ('ali', 5), ('khalid', 22)])
>>>
```

كذلك من الأمور المميزة في المجلدات إمكانية استخدام عبارة (for) المزدوجة بمتغيرين ليكون أحدهما معبراً عن المفاتيح والآخر عن القيم وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
>>> a=dict()
>>> a={"I":1, "you":2, "he":3, "she":3, "it":3, "we":1, "they":3}
>>> for i,j in a.items():
    print (i,j)

they 3
it 3
you 2
I 1
she 3
we 1
he 3
>>> |
```

واخيراً وكملخص لكل ما درسنا لحد الان أتمنى التركيز على المثال التالي حيث يشرح كيفية توظيف كل الدوال والمميزات التي شرحناها لحد الان عن السلاسل الرمزية والملفات وأدوات الادخال والإخراج والقوائم والمجلدات لقراءة ملف وتجزئة محتوياته ثم التعامل مع كل منها على حدة لحساب عدد مرات تكرار كل كلمة ثم طباعة الكلمة التي ذكرت أكثر عدد من المرات مع عدد مرات التكرار فتابعوا:

9.4 Write a program to read through the **mbox-short.txt** and figure out who has the sent the greatest number of mail messages. The program looks for 'From ' lines and takes the second word of those lines as the person who sent the mail. The program creates a Python dictionary that maps the sender's mail address to a count of the number of times they appear in the file. After the dictionary is produced, the program reads through the dictionary using a maximum loop to find the most prolific committer.

Check Code

Reset Code



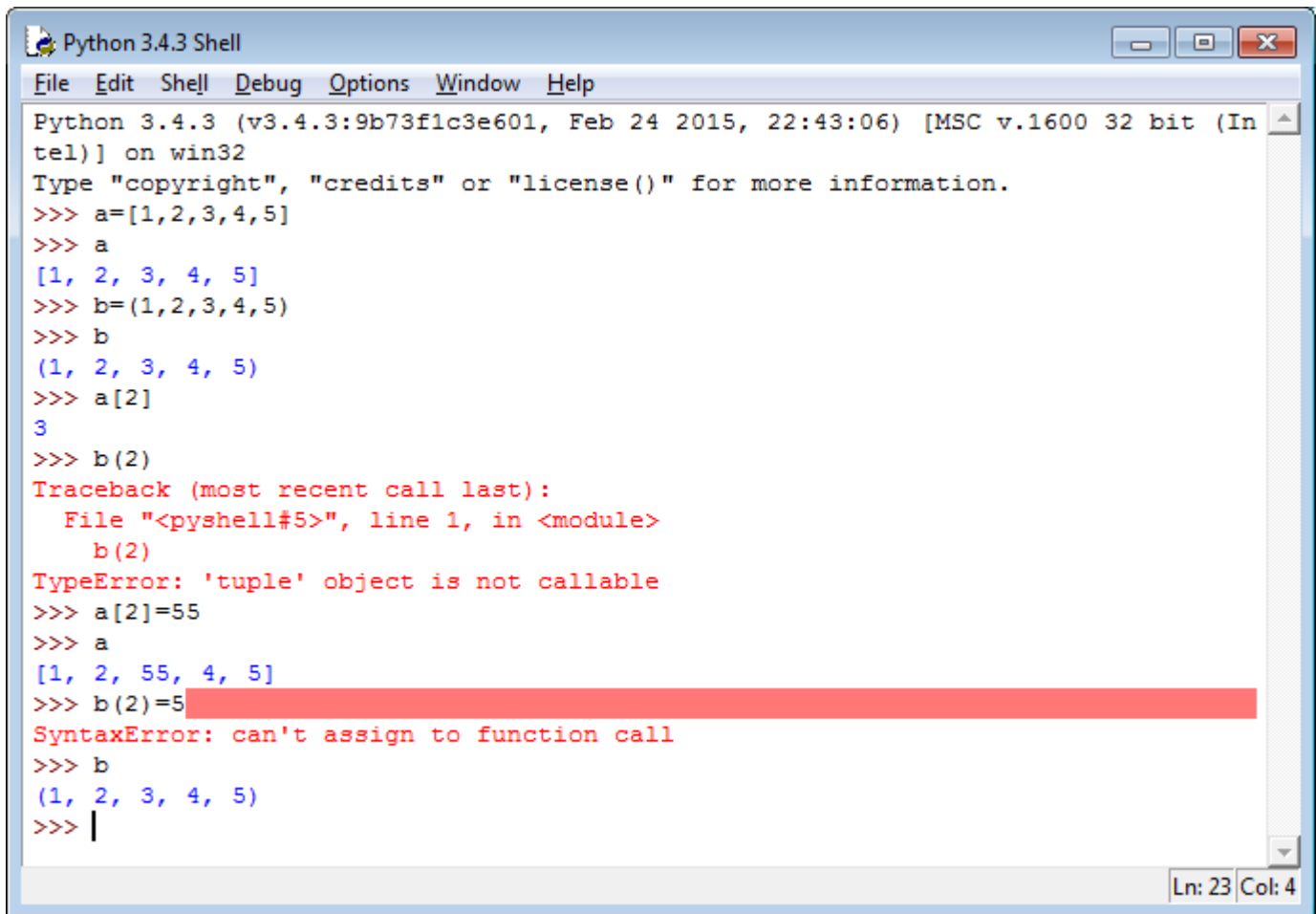
Exit

Grade updated on server.

```
1 name = raw_input("Enter file:")
2 if len(name) < 1 : name = "mbox-short.txt"
3 handle = open(name)
4 mail=dict()
5 for line in handle:
6     if line.startswith('From '):
7         address=line.split()
8         address1=address[1]
9         mail[address1]=mail.get(address1,0)+1
10 bigcount=None
11 bigaddress=None
12 for line, count in mail.items():
13     if bigcount is None or count > bigcount:
14         bigaddress=address1
15         bigcount=count
16 print bigaddress, bigcount
```

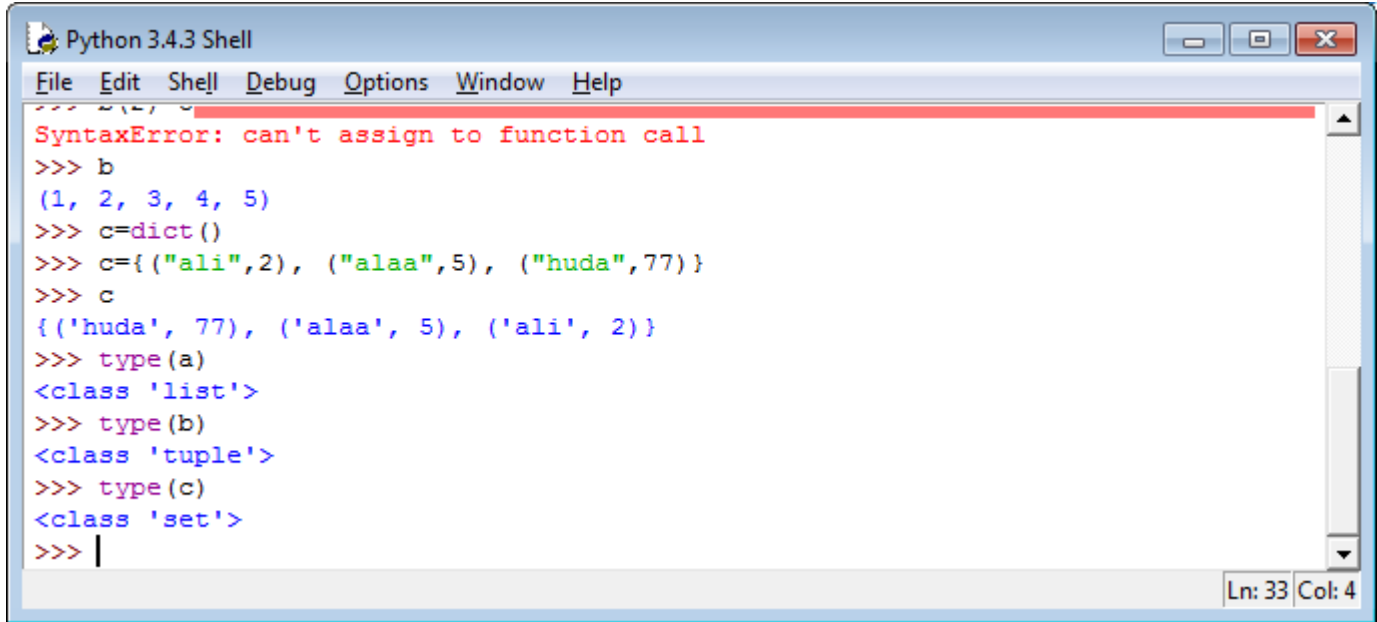

الصفوف او المجاميع Tuples

نصل اليوم الى شرح النوع الثالث من أنواع مجاميع البيانات في لغة البايثون بعد ان شرحنا القوائم (lists) والمجلدات او القواميس (directories) وهذا النوع يسمى الصفوف او المجاميع (Tuples). وهذا النوع من البيانات يشبه الى حد كبير القوائم في كونه يتكون من مجموعة من العناصر الا ان العناصر هنا تكون محصورة بين قوسين صغيرين () بدلاً من الاقواس الكبيرة للقوائم []. والميزة الأهم للصفوف ان عناصرها غير قابلة للتعديل (immutable) ولذا تستخدم عادة بشكل كفوء كبديل للقوائم والمجلدات في حالة الحاجة الى التعامل مع قوائم مؤقتة (Temporary lists) بدون الحاجة الى تغيير مكوناتها حيث انها تأخذ مساحة اقل من الذاكرة ووقتاً اقل في المعالجة من قبل وحدة المعالجة المركزية (CPU) ويمكن التعامل معها في لغة بايثون كما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5]
>>> a
[1, 2, 3, 4, 5]
>>> b=(1,2,3,4,5)
>>> b
(1, 2, 3, 4, 5)
>>> a[2]
3
>>> b(2)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b(2)
TypeError: 'tuple' object is not callable
>>> a[2]=55
>>> a
[1, 2, 55, 4, 5]
>>> b(2)=5
SyntaxError: can't assign to function call
>>> b
(1, 2, 3, 4, 5)
>>> |
```

وهنا نتضح الفوارق الرئيسية بين القوائم والصفوف حيث رأينا ان (a) هي قائمة يمكن استدعاء عناصرها وتغييرها في حين لا يسمح الصف (b) باستدعاء عناصره او تغييرها. كذلك لمعرفة نوع المجاميع التي نتعامل معها نستخدم دالة (type) وكما في ادناه:



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
SyntaxError: can't assign to function call
>>> b
(1, 2, 3, 4, 5)
>>> c=dict()
>>> c={"ali",2}, ("alaa",5), ("huda",77)}
>>> c
{('huda', 77), ('alaa', 5), ('ali', 2)}
>>> type(a)
<class 'list'>
>>> type(b)
<class 'tuple'>
>>> type(c)
<class 'set'>
>>> |
Ln: 33 Col: 4
```

يمكن التعامل مع الصفوف كمتغيرات تماماً كما تعاملنا مع القوائم والمجندات بإدخالها ضمن عبارات الشروط والتكرار وكل ما تعلمناه سابقاً من أدوات الإدخال والإخراج وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=(1,2,3,4,5)
>>> for i in a:
    print(i*i)

1
4
9
16
25
>>> a+a
(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
>>> for k in a:
    if k>3:
        print ("ok")
    else: print (" not ok")

not ok
not ok
not ok
ok
ok
>>> |
```

من مميزات القوائم الأخرى انها وكما قلنا غير قابلة لتغيير عناصرها ولذا فهي غير قابلة للترتيب او الإضافة او الحذف
كما هو ممكن في القوائم وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3,4,5] # a is list
>>> b=(1,2,3,4,5) #b is a tuple
>>> max(a)
5
>>> max(b)
5
>>> a.append(6)
>>> a
[1, 2, 3, 4, 5, 6]
>>> b.append(6)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    b.append(6)
AttributeError: 'tuple' object has no attribute 'append'
>>> b
(1, 2, 3, 4, 5)
>>> a.reverse()
>>> a
[6, 5, 4, 3, 2, 1]
>>> b.reverse()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    b.reverse()
AttributeError: 'tuple' object has no attribute 'reverse'
>>> b
(1, 2, 3, 4, 5)
>>> c=[1,4,7,9,5,3,1]
>>> c.sort()
>>> c
[1, 1, 3, 4, 5, 7, 9]
>>> b.sort()
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    b.sort()
AttributeError: 'tuple' object has no attribute 'sort'
>>> b
(1, 2, 3, 4, 5)
>>>
```

ولمعرفة الدوال التي تعمل مع الصفوف نستخدم دالة (dir()) كما في القوائم والمجلدات وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=[1,2,3]
>>> b=(4,5,6)
>>> c=dict()
>>> c={(a,1),(b,2),(c,4)}
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    c={(a,1),(b,2),(c,4)}
TypeError: unhashable type: 'dict'
>>> c={('a',1),('b',2),('c',4)}
>>> dir(a)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>> dir(b)
['_add_', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
>>> dir(c)
['_and_', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__iand_', '__init__', '__ior_', '__isub_', '__iter__', '__ixor__', '__le__', '__len_', '__lt__', '__ne_', '__new__', '__or_', '__rand_', '__reduce__', '__reduce_ex_', '__repr__', '__ror_', '__rsub_', '__rxor_', '__setattr__', '__sizeof_', '__str__', '__sub_', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
>>> |
```

من مميزات الصفوف ايضاً قابلية الاسناد المزدوج والأكثر من المزدوج (اسناد عنصرين او أكثر الى قيمتين او أكثر في نفس الوقت) وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a,b=5,6
>>> a
5
>>> b
6
>>> c=(a,b)
>>> c
(5, 6)
>>> c+c+c
(5, 6, 5, 6, 5, 6)
>>> d=c*3
>>> d
(5, 6, 5, 6, 5, 6)
>>> (r,t)=(33,"tool")
>>> r
33
>>> t
'tool'
>>> c+r+t
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    c+r+t
TypeError: can only concatenate tuple (not "int") to tuple
>>> c+r
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    c+r
TypeError: can only concatenate tuple (not "int") to tuple
>>> r+t
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    r+t
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> |
```

وهنا كما لاحظنا استطعنا ان نسند عنصرين وهما (a,b) الى قيم (5,6) في ايعاز واحد كما رأينا كيف يمكن تنفيذ ايعازات الجمع والضرب على الصفوف حيث لا يتم الجمع ولا الضرب وانما ارفاق او تكرار الصف اكثر من مرة

وبحسب اليعاز. واخيراً رأينا كيف انه لا يمكن إضافة عناصر رمزية الى الصفوف ولا يمكن ارفاق سلسلة رمزية مع

متغيرات صحيحة وهي أمور بديهية الان لمن تابع دورتنا من البداية ☺

والان للتنقل بين القواميس (dictionaries) والصفوف (tuples) نستخدم ايعاز (.items()) ليحول لنا المجلد او

القاموس الى سلسلة من الأزواج المرتبة (tuples) وكما في ادناه:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=dict()
>>> b=[1,2,3,4,5]
>>> for i in b:
    a[i]=i*i

>>> a
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> for (k,v) in a.item():
    print (k,v)

Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    for (k,v) in a.item():
AttributeError: 'dict' object has no attribute 'item'
>>> for (k,v) in a.items():
    print (k,v)

1 1
2 4
3 9
4 16
5 25
>>> tuples=a.items()
>>> print tuples
SyntaxError: Missing parentheses in call to 'print'
>>> print (tuples)
dict_items([(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)])
>>> |
```

والان من مميزات الصفوف قابلية المقارنة بينها حيث ان كان الصف يتكون من ٤ عناصر وارادنا المقارنة فيتم مقارنة العنصر الأول من الصف الأول بالعنصر الأول من الصف الثاني فأن تساويا فيتم الانتقال الى مقارنة العنصر الثاني من الصف الأول بالعنصر الثاني من الصف الثاني وهكذا وكما توضحه الأمثلة التالية:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=(1,2,3)
>>> b=(3,2,1)
>>> a>b
False
>>> a<b
True
>>> a<=b
True
>>> ("jonas", "hadi")<("Khalid","terry")
False
>>> ("ABC","DEF")=("abc","def")
SyntaxError: can't assign to literal
>>> ("ABC","DEF")==("abc","def")
False
>>> ("ABC","DEF")>=("abc","def")
False
>>> ("ABC","DEF")<("abc","def")
True
>>> ("a2","aa","33")<("a2","aa","34")
True
>>> |
```

من المميزات الهامة للصفوف هو إمكانية ترتيب عناصرها باستخدام دالة (sorted ()) وكما في ادناه:


```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=dict()
>>> a={'a':2,'b':3,'c':33,'g':22}
>>> a.items()
dict_items([('g', 22), ('a', 2), ('b', 3), ('c', 33)])
>>> b=sorted(a.items())
>>> b
[('a', 2), ('b', 3), ('c', 33), ('g', 22)]
>>> a['d']=44
>>> a
{'g': 22, 'a': 2, 'b': 3, 'd': 44, 'c': 33}
>>> print(sorted(a.items()))
[('a', 2), ('b', 3), ('c', 33), ('d', 44), ('g', 22)]
>>> |
```

والان نلاحظ انه في المثال أعلاه قمنا بترتيب عناصر القاموس اعتماداً على المفاتيح (keys) واما لو أردنا الترتيب بحسب القيم (values) فنقوم بالتالي:

```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a={'a':22,'r':11,'c':3,'z':44}
>>> temp=list()
>>> for k,v in a.items():
    temp.append((v,k))

>>> temp
[(44, 'z'), (3, 'c'), (22, 'a'), (11, 'r')]
>>> temp.sort()
>>> temp
[(3, 'c'), (11, 'r'), (22, 'a'), (44, 'z')]
>>> temp.sort(reverse=True) # reverse sort
>>> temp
[(44, 'z'), (22, 'a'), (11, 'r'), (3, 'c')]
>>> |
```

واخيراً اليكم بعض الأمثلة التي تجمع الكثير من المبادئ التي تعلمناها لحد الان:

المثال الأول: فتح ملف وقراءة محتوياته ثم حساب عدد مرات تكرار كل كلمة وطباعة الكلمات العشرة الأكثر تكراراً وعدد مرات تكرار كل منها:

```
*3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)*
File Edit Format Run Options Window Help
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0 ) + 1

lst = list()
for key, val in counts.items():
    lst.append( (val, key) )

lst.sort(reverse=True)

for val, key in lst[:10] :
    print (key, val)
```

Ln: 15 Col: 20

المثال الثاني: قراءة ملف بريد الكتروني ومعرفة عدد الرسائل التي تم استلامها في كل ساعة واستخراج الساعات من السطور التي تبدأ بكلمة (From) وتجزئة تلك الاسطر الى حد الساعة ثم حساب عدد الرسائل لكل ساعة وترتيبها وطباعتها جميعاً:

10.2 Write a program to read through the **mbox-short.txt** and figure out the distribution by hour of the day for each of the messages. You can pull the hour out from the 'From ' line by finding the time and then splitting the string a second time using a colon.

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

Once you have accumulated the counts for each hour, print out the counts, sorted by hour as shown below. Note that the autograder does not have support for the sorted() function.

Check Code

Reset Code



Exit

Grade updated on server.

```
1 name = raw_input("Enter file:")
2 if len(name) < 1 : name = "mbox-short.txt"
3 handle = open(name)
4 count=dict()
5 for line in handle:
6     if line.startswith("From "):
7         time=line.split()
8         hours=time[5].split(":")
9         count[hours[0]]=count.get(hours[0],0)+1
10 lst=list()
11 for key, val in count.items():
12     lst.append((key, val))
13 lst.sort()
14 for key,val in lst:
15     print key, val
```

التعابير القياسية (Regular Expressions)

ربما لاحظتم احبتي الكرام ان تركيزنا في هذه الدورة لحد الان هو على كيفية استخدام لغة بايثون لاستخلاص معلومات مفيدة من ملفات بيانات نصية وهو ما ركز عليه الكتاب المنهجي لهذه الدورة (Python of Informatics) ولا بد ان اغلبكم يعلم ان تطبيقات بايثون لا تقتصر على هذا وانما على برمجة تطبيقات الحاسوب وبرمجة مواقع الانترنت وبرمجة الأجهزة المستخدمة في السيطرة والتحكم والكثير الكثير من المجالات الأخرى ولكن هدفنا هنا هو دورة اساسيات اللغة البايثون للمبتدئين وحتى المحترفين في مجال المعلوماتية وقد تعلمنا لحد الان الكثير من الطرق للبحث عن بيانات معينة في سطر او ملف واستخلاص ما نريد وطباعته او معالجته بطريقة معينة.

كل ما في الامر ان هناك طرق مختصرة أكثر وأسرع لتحقيق ما كنا نكتبه في ١٠ او أكثر من الاسطر البرمجية في

سطر واحد او سطرين!

نعم يمكن كل ذلك باستخدام رموز برمجية معقدة الى حد ما تنتمي الى مدرسة البرمجة القديمة (old school programming) حين كانت البرمجة معقدة للغاية ومليئة بالرموز والاختصار لتقليل حجم البرامج وزمن تنفيذها حين كانت سرعة المعالجات محدودة والذاكر صغيرة.

على كل حال ينصح خبراء البرمجة بتعلم هذه الطرق المختصرة للبرمجة لمن يريد الحرص على صغر البرامج وسرعة تنفيذها ولا داعي للقلق في حالة عدم فهمها فقد تعلمنا (وستعلم ان شاء الله) طرق أكثر بساطة في تحقيق كل ما تحققه الأدوات التالية:

```
^      Matches the beginning of a line
$      Matches the end of the line
.      Matches any character
\s     Matches whitespace
\S     Matches any non-whitespace character
*      Repeats a character zero or more times
*?     Repeats a character zero or more times (non-greedy)
+      Repeats a character one or more times
+?     Repeats a character one or more times (non-greedy)
[aeiou] Matches a single character in the listed set
[^XYZ] Matches a single character not in the listed set
[a-z0-9] The set of characters can include a range
(      Indicates where string extraction is to start
)      Indicates where string extraction is to end
```

والان لنبدأ بالأمر أولاً بأول:

قبل استخدام أي من هذه الأدوات لا بد من استدعاء مكتبة خاصة بها ويتم ذلك بوضع عبارة (import re) في بداية البرنامج.

أهمية التعبيرات القياسية وخصائصها:

- انها فعالة جداً ومختصرة ومشفرة الى حد كبير.

- انها ممتعة حالما تتعلم كيفية استخدامها.

- تعتبر التعبيرات القياسية لغة بحد ذاتها وتتشترك الكثير من خصائصها مع العديد من لغات البرمجة الأخرى.
 - تسمى هذه اللغة الخاصة لغة برمجة رموز التأشير (marker characters) وهي أحد أدوات المدرسة القديمة (old school) في البرمجة.
 - قبل استخدام هذه التعبيرات يجب استيراد مكتبتها (import re).
 - تستطيع هذه التعبيرات استخدام دوال (re.search()) لمطابقة سلاسل رمزية مع رموز معينة يتم إدخالها وهي تشبه إلى حد كبير دالة (find()) إلا أنها أكثر كفاءة كما سنرى.
 - تستطيع هذه التعبيرات استخدام دالة (re.findall()) لاستخلاص جزء من سلسلة رمزية تطابق مدخلاتنا وهي تشبه عمل كل من دالة البحث (find()) والتشريح للسلاسل الرمزية (slicing).
- والآن قبل الخوض في إعطاء أمثلة للمقارنة بين شكل البرنامج عند استخدام التعبيرات القياسية وبدونها، لا بد من شرح معنى كل منها فتعالوا معنا:

التعبير	الاستخدام
^	مطابقة بداية السطر.
\$	مطابقة نهاية السطر.
.	مطابقة أي رمز في سلسلة رموز.
\s	مطابقة المسافات (الفراغات) او whitespaces
\S	مطابقة الرموز التي ليست فراغات (non-whitespaces)
*	كما في البحث يستخدم للدلالة على (anything) لأي عدد من الرموز من صفر إلى أي عدد آخر.
*?	نفس الرمز السابق إلا أنه يسمى التكرار غير الطماع (non greedy) أي أنه يتوقف عند أو مطابقة.
+	مثل عمل ال * تماماً.

+	مثل عمل * تماماً.
[abcd]	مطابقة رمز واحد من هذه الرموز مع المدخلات المعطاة. يمكن وضع أي رمز هنا.
[^abcd]	مطابقة رمز واحد ليس في هذه القائمة وايضاً يمكن وضع أي رمز هنا (حرف صغير او كبير او رموز خاصة او ارقام).
[0-9]	هذه الاقواس يمكن ان تحتوي مدى من الرموز بينها علامة الشرطة (-) ولأي سلسلة من الرموز او الأرقام المتتالية.
[a-z]	
[A-Z]	
()	اقواس البداية والنهاية للاستخلاص حيث يتم ارجاع ما بداخلها فقط من ما ترجمه بقية الأدوات من مطابقة او اختلاف او غيرها.

والان الى الأمثلة حيث سنتناول مجموعة من الأمثلة التي تحدثنا عنها في الدروس السابقة ولكن هذه المرة سنأخذها مرة مع التعبيرات القياسية ومرة بدونها (وهو ما شرحناه لحد الان).

المثال الأول:

```

import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print line

```

البرنامج الى جهة اليسار هو ما تعلمناه سابقاً من فتح (قراءة) ملف نصي ثم استخدام (for) للتنقل بين اسطره وإلغاء الفراغات الى يمين (نهاية) كل سطر ثم البحث عن الاسطر التي تبدأ بكلمة (From:) لطباعتها.

اما البرنامج الى اليمين فهو يقوم بالمثل بالضبط ولكن باستخدام التعبيرات القياسية حيث استخدمنا دالة (re.search ()) وفي داخلها وضعنا (line, 'Form: ') وتعني ابحت باستخدام التعبيرات القياسية (التي لم نستخدم أي منها هنا!) عن أي كلمة (From:) لطباعة سطرها.

المثال الثاني:

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:') :
        print line
```

```
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print line
```

هنا نفس البرنامج مع تحديد أكثر بالبحث عن السطر الذي يبدأ بكلمة (From) وليس أي كلمة (from) في أي مكان من السطر والبرنامج الى اليسار يفترض انه مفهوم واما البرنامج الى اليمين ففيه العبارة التالية التي تستحق الشرح:

if re.search('^From:', line) :

حيث قلنا للمترجم هنا: اذا كانت نتيجة البحث عن (السطر الذي يبدأ (^) بعبارة (From:)) صحيحة (True). بعدها السطر الأخير الذي يطلب طباعة السطر.

وهنا لاحظنا كيف استخدمنا اول تعبير قياسي (^) للمقارنة ببداية السلسلة الرمزية.

المثال الثالث:

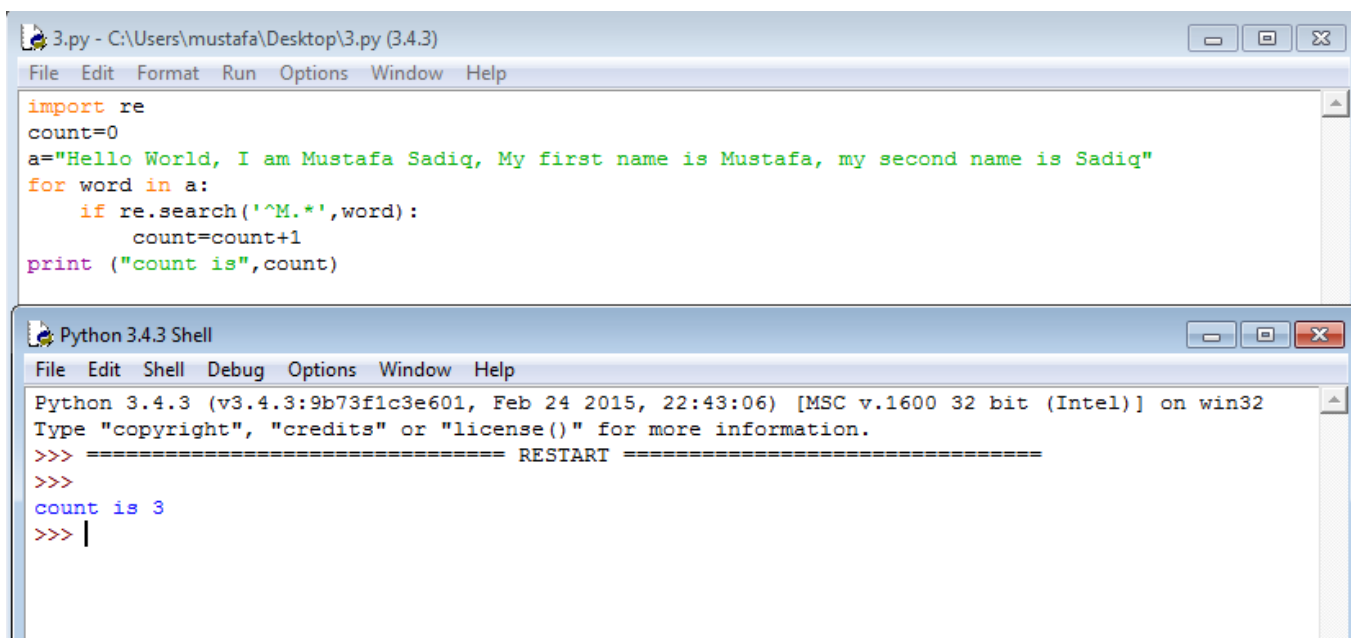
ذكرنا ان النقطة (.) تعني مطابقة أي رمز وعلامة (*) تعني أي عدد من الرموز وعند اجتماعهما معاً يعطيان البحث والمطابقة بعداً جديداً يسمى احياناً (wild card) وكما في المثال ادناه:


```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

^X.*:

وهنا معنى التركيبة الى اليمين قم بالبحث والمطابقة لكل سلسلة رمزية تبدأ (^) بحرف (X) كبير ثم بعده أي رمز (.).
او مجموعة من الرموز (*) ومثال ذلك كما في ادناه:

المثال الرابع:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
count=0
a="Hello World, I am Mustafa Sadiq, My first name is Mustafa, my second name is Sadiq"
for word in a:
    if re.search('^M.*',word):
        count=count+1
print ("count is",count)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
count is 3
>>> |
```

والان لنشرح المثال أعلاه:

السطر الأول: استيراد لمكتبة التعابير القياسية.

السطر الثاني: تعريف لمتغير اسمه (count) ومنحه قيمة أولية مقدارها صفر.

السطر الثالث: تعريف لمتغير اسمه (a) وهو عبارة عن سلسلة رمزية واضحة باللون الأخضر.

السطر الرابع: عبارة (for) للتنقل بين مكونات السلسلة الرمزية حيث يكون كل مجموعة رموز يفصل بينها وبين ما قبلها وما بعدها فراغ كعنصر واحد من عناصر السلسلة الرمزية نسميها (word).

السطر الخامس: شرط للبحث باستخدام التعابير القياسية نصه يقول: ابحث عن التراكيب التي تبدأ بحرف (M) كبير وبعده مجموعة من الرموز في كل جزء (word) من السلسلة (a) وفي حالة كون الشرط صحيح سيتم تطبيق السطر السادس.

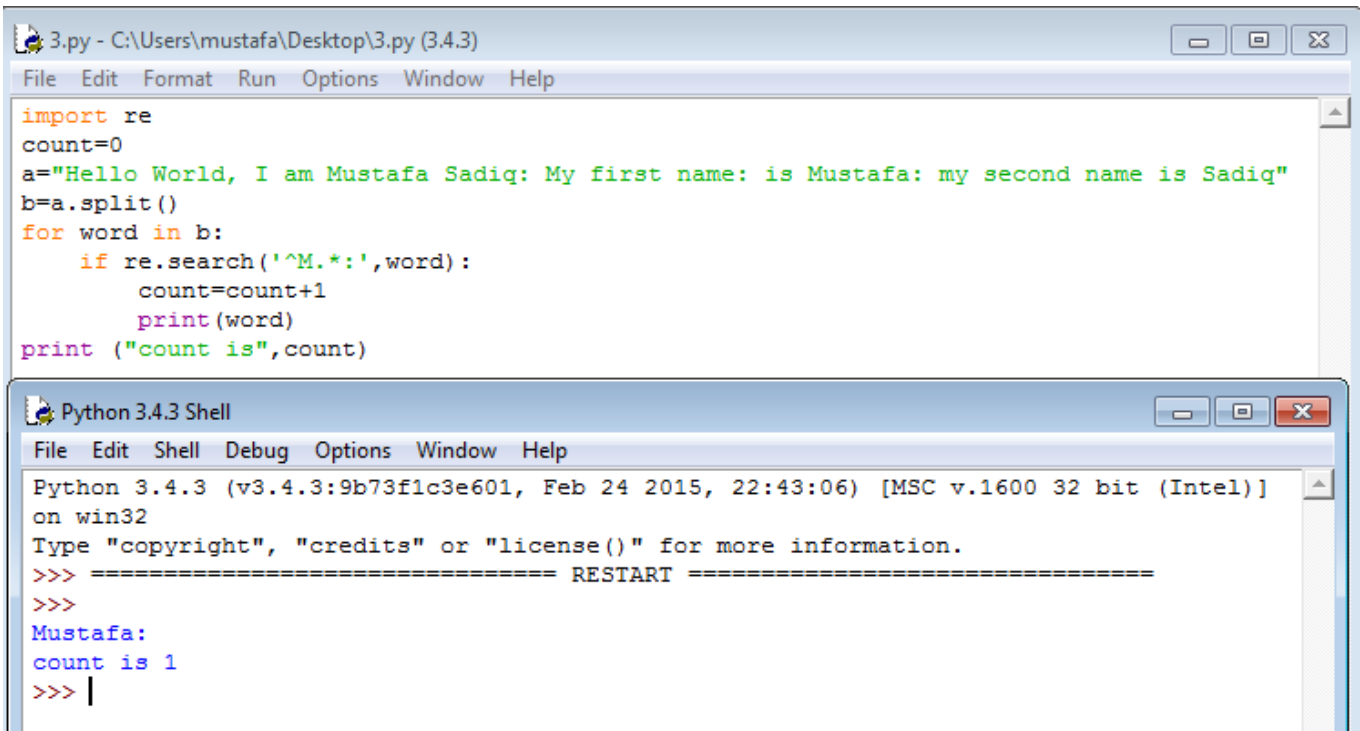
السطر السادس: قم بزيادة عدد ال (Count) بمقدار واحد.

السطر السابع: طباعة عبارة (count is) ثم عدد مرات تكرار ذلك التركيب الذي كنا نبحث عنه.

نتائج التنفيذ: ظهر لنا ان ناتج التنفيذ هو ٣ وهي الكلمات (Mustafa, My, Mustafa). والتي يبدأ كل منها بحرف (M) كبير وبعده مجموعة رموز غير محددة.

ملاحظة: بعض الأحيان البيانات تكون غير مرتبة او (Clean) كما نتوقع فيجب ان نحسب حساب كل الاحتمالات ونضيق نطاق البحث باستخدام أدوات أخرى من ضمن التعابير القياسية وكما في المثال ادناه:

المثال الخامس:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
count=0
a="Hello World, I am Mustafa Sadiq: My first name: is Mustafa: my second name is Sadiq"
b=a.split()
for word in b:
    if re.search('^M.*:',word):
        count=count+1
        print(word)
print ("count is",count)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Mustafa:
count is 1
>>> |
```

هنا قمنا بنفس خطوات المثال أعلاه غير اننا فقط اضفنا الى شرط المطابقة العبارة التالية:

ابحث عن الكلمات التي تبدأ بالحرف (M) ثم أي مجموعة من الحروف ثم الرمز (:). وهنا نجد ان المطابقة كانت رمزاً واحداً فقط.

كذلك نستطيع عمل المثل كما في المثال التالي:

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks
```

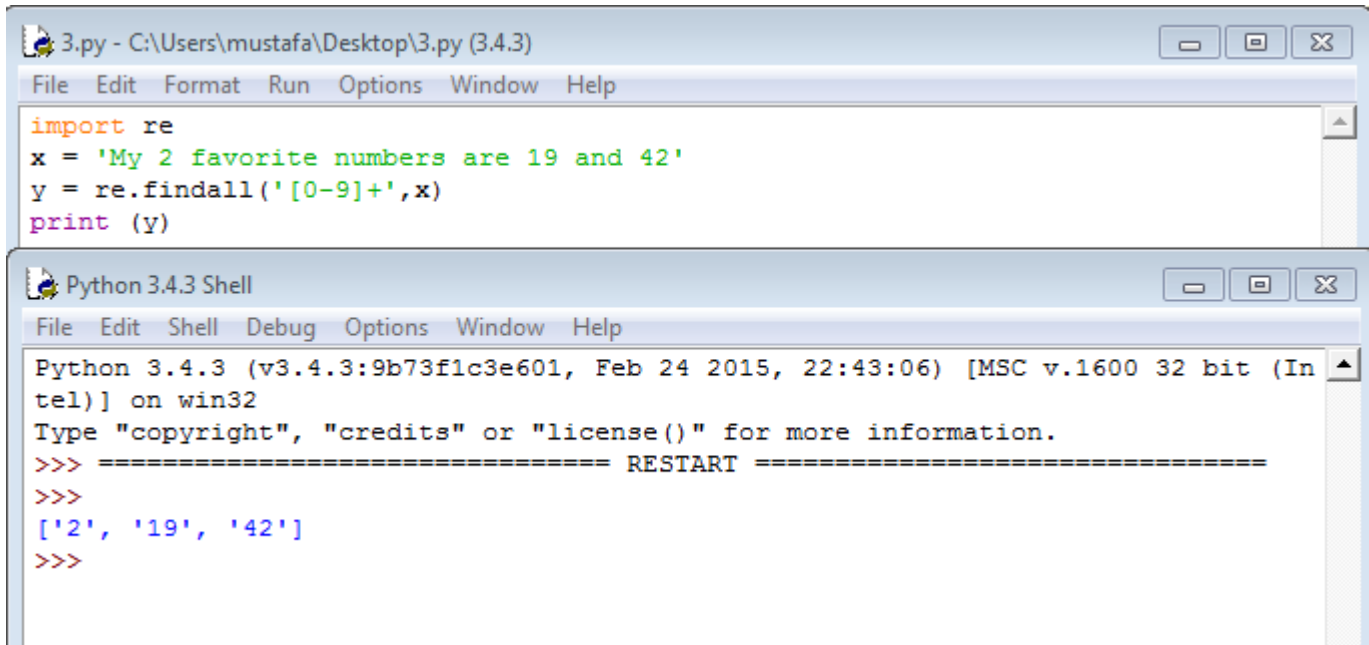
طابق بداية السطر
رمز واحد او اكثر
X- \ S+ :
طابق الرموز التي ليست فراغات non-whitespace

حيث قلنا للمفسر هنا: قم بمطابقة الكلمات (او السلاسل الرمزية) التي تبدأ بالحرف الكبير (X) ثم بعده مجموعة رموز ليست فراغات ثم رمز (:). ونرى في الجانب الايسر ان اول عبارتين طابقتا المطلوب ولكن العبارة الثالثة لم تعد مطابقة لأن بين ال (X) وال (:). توجد مجموعة فراغات.

التعابير القياسية ٢

بعد ان تحدثنا عن مقدمة التعامل مع التعابير القياسية (Regular Expressions) ومميزاتها وبعض الأمثلة عنها في الدرس السابق، نأتي اليوم الى اكمال حديثنا عنها بشرح المزيد من الأمثلة للمقارنة بين مميزات استخدامها والبرمجة بدونها وكما كرنا سابقاً يبقى خيار استخدامها او تركها للمبرمج مع التأكيد على أهميتها لتقليل حجم البرنامج وتسريع التنفيذ فتابعوا معنا:

المثال الأول: استرجاع الأرقام فقط:

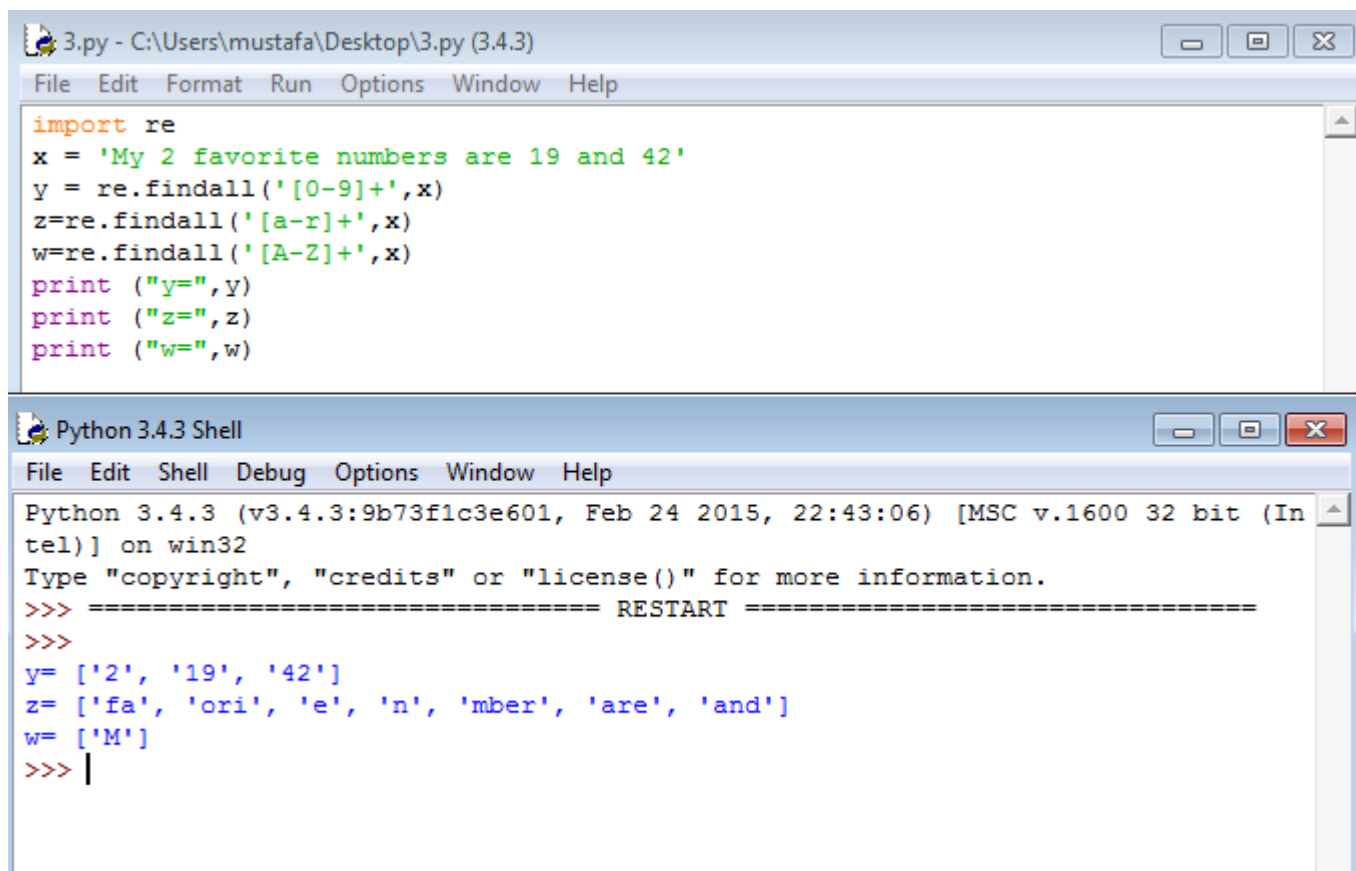


```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
x = 'My 2 favorite numbers are 19 and 42'
y = re.findall('[0-9]+',x)
print (y)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['2', '19', '42']
>>>
```

هذا المثال يتكون من ثلاث أسطر برمجية فقط في الأول قمنا باستيراد مكتبة التعابير القياسية التي لا يمكن ان تعمل الا باستيراد مكتبتها والسطر الثاني ادخال لقيمة المتغير (x) واما السطر الثالث فهو تعريف للمتغير (y) واسناد قيمة (او قيم) له هي عبارة عن ناتج تنفيذ عبارة البحث ضمن المتغير (x) عن أي سلسلة رقمية تحتوي الأرقام من الصفر الى التسعة ويمكن البحث عن أي سلسلة رقمية او رمزية وكما في ادناه:

المثال الثاني: استرجاع قيم رمزية لمجموعة حروف معينة فقط:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

import re
x = 'My 2 favorite numbers are 19 and 42'
y = re.findall('[0-9]+',x)
z=re.findall('[a-r]+',x)
w=re.findall('[A-Z]+',x)
print ("y=",y)
print ("z=",z)
print ("w=",w)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
y= ['2', '19', '42']
z= ['fa', 'ori', 'e', 'n', 'mber', 'are', 'and']
w= ['M']
>>> |
```

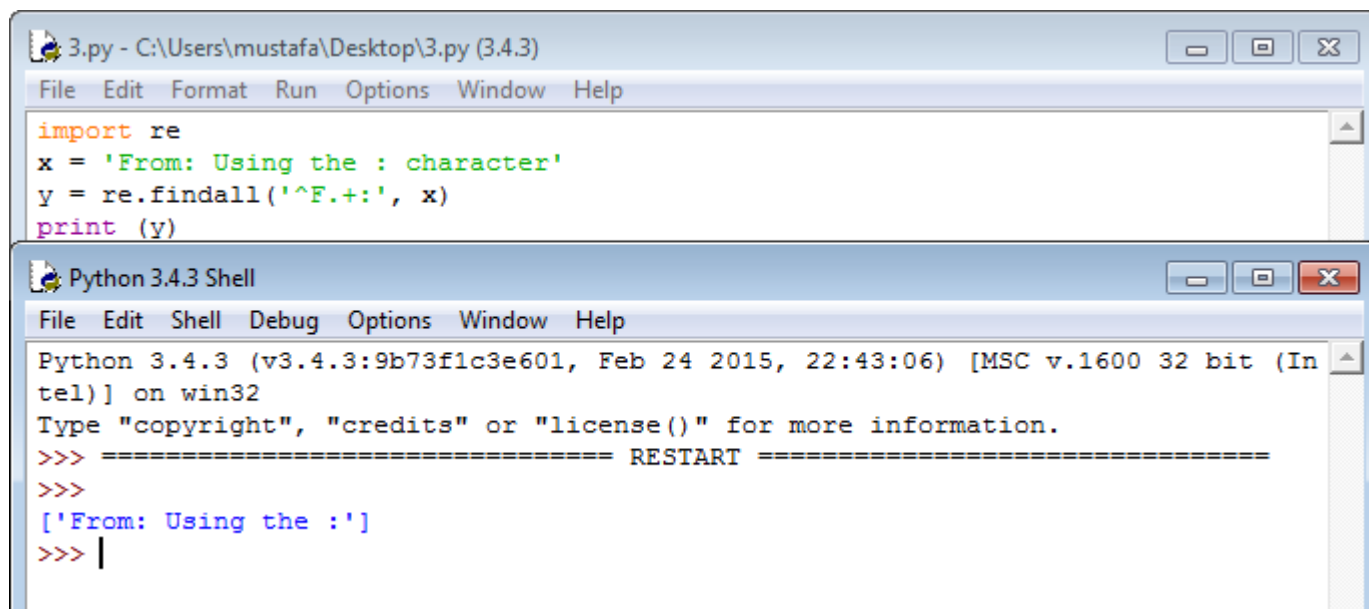
وكما شرحنا في أعلاه:

Y تقوم بإرجاع الأرقام

Z تقوم بإرجاع السلاسل الرمزية التي تحتوي الحروف الصغيرة من a إلى r علماً أن ال r من ضمنها.

W تقوم بإرجاع السلاسل الرمزية التي تتكون من حروف كبيرة فقط.

المثال الثالث:

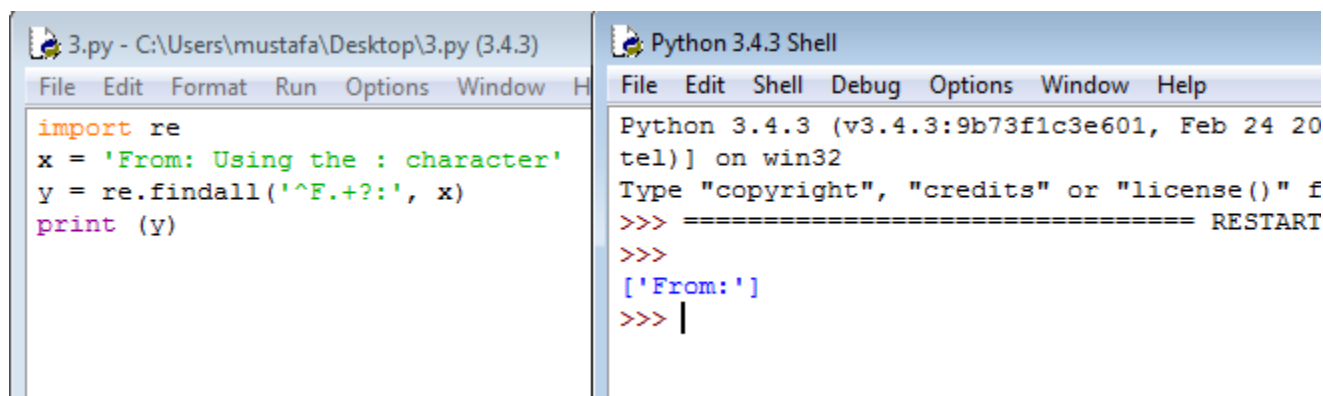


```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
x = 'From: Using the : character'
y = re.findall('^F.+:', x)
print (y)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['From: Using the :']
>>> |
```

هنا قلنا للمفسر قم بطباعة قيمة المتغير (y) على انها أي سلسلة رمزية تبدأ بالحرف الكبير (F) ثم بعده مجموعة رموز وصولاً الى النقطتين (:). ولكن لأن الرمزين (+ و *) يسميان أدوات المطابقة الطماعا (greedy matching tools) فهما يقومان بالتعدي الى ابعد نقطة مطابقة للشروط ولذلك لم تتوقف المطابقة عند اول نقطتين (:). وانما تجاوزتها الى اخر نقطتين. ولتلافي هذه المشكلة نتابع المثال التالي:

المثال الرابع: المطابقة غير الطماعا (non-greedy matching):



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window H
import re
x = 'From: Using the : character'
y = re.findall('^F.+?:', x)
print (y)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 20
tel)] on win32
Type "copyright", "credits" or "license()" f
>>> ===== RESTART =====
>>>
['From:']
>>> |
```

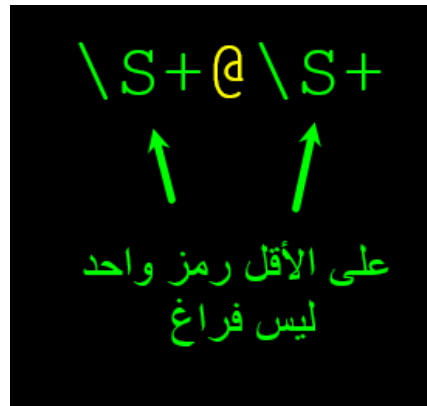
هنا قمنا بكتابة نفس الكود البرمجي وكل ما غيرناه هو إضافة (?) الى التركيب القياسي لنجعله يتوقف عند اول نقطة مطابقة لنهاية السلسلة المطلوبة وهكذا حصل.

المثال الخامس: تركيز المطابقة حول رمز واحد:

```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
x = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('\S+@\S+', x)
print (y)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['stephen.marquard@uct.ac.za']
>>> |
```

هنا قلنا للمفسر قم بالبحث عن الرمز (@) وقبله على الأقل رمز واحد ليس فراغ (non-whitespace) وبعده على الأقل رمز واحد ليس فراغ (non-whitespace) وقم بوضعه داخل المتغير (y) ثم اطبعه.



المثال السادس: لزيادة التركيز أكثر في نتائج المطابقة نستخدم الاقواس () والتي تحدد بداية ونهاية الجزء المراد ارجاعه كما في ادناه:

```

3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help

import re
x = 'From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('\S+@\S+', x)
print ("y=",y)
z=re.findall('^From:.*? (\S+@\S+)',x)
print ("z=",z)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help

Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
y= ['stephen.marquard@uct.ac.za']
z= ['stephen.marquard@uct.ac.za']
>>> |

```

وهنا نلاحظ ان ال (z) على الرغم من انها تحتوي على الكثير من الرموز للمطابقة بداخلها الا ان ما ارجعته عبارة الطباعة هو فقط ما كان محصوراً بين القوسين () وهو المطلوب لتضييق نطاق النتائج المرجعة.

المثال السابع:

```

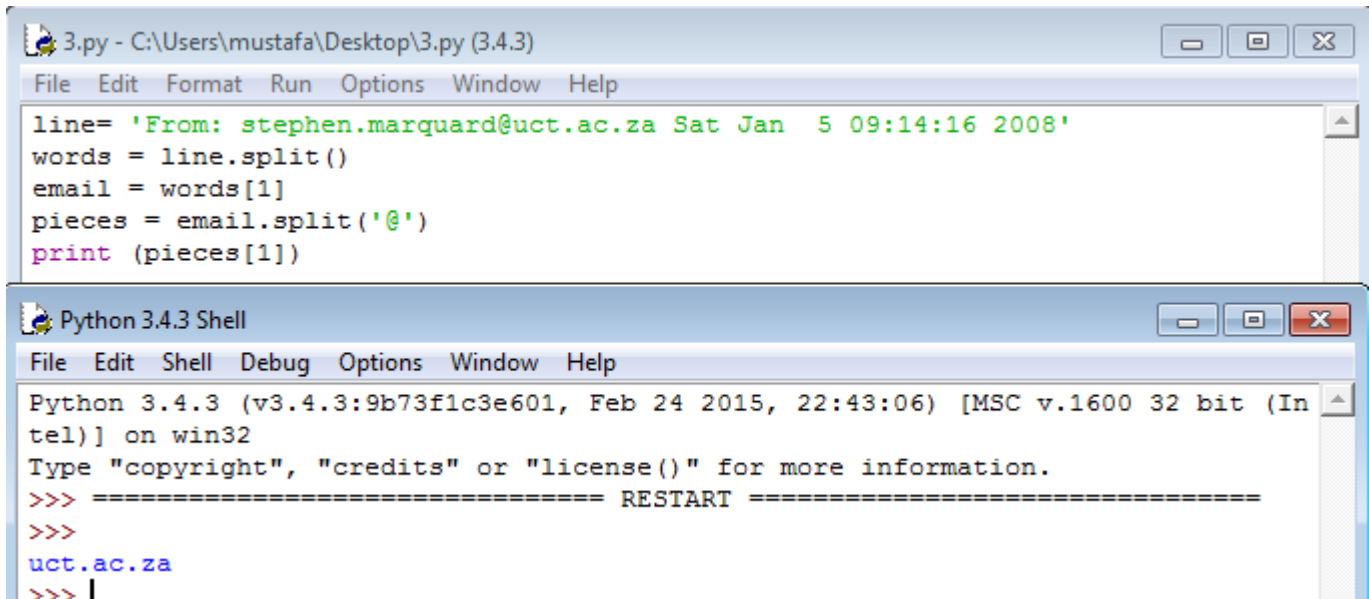
                21           31
                ↓           ↓
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print atpos
21
>>> spos = data.find(' ', atpos)
>>> print spos
31
>>> host = data[atpos+1 : spos]
>>> print host
uct.ac.za

```

كما وضعنا سابقاً يمكن استخدام دالة البحث (find()) لإيجاد رمز معين بدون التعابير القياسية كما في هذا المثال:

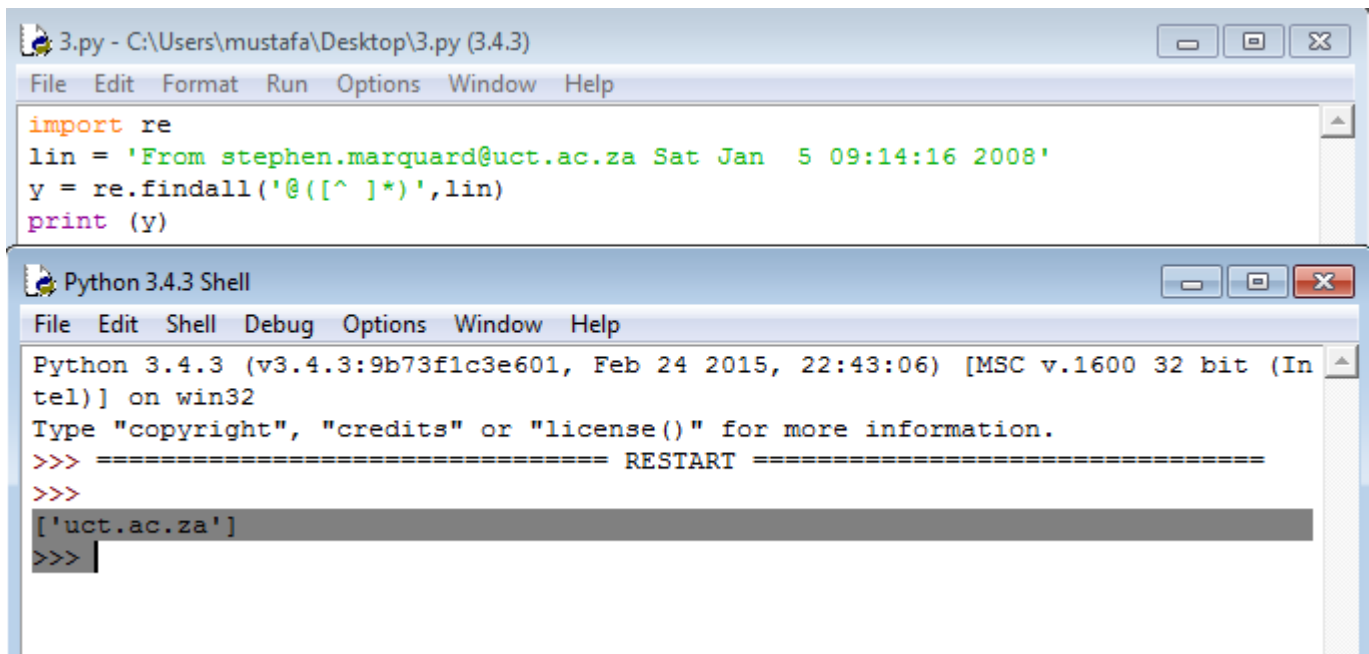
المثال الثامن: البحث بطريقة التجزئة المزدوجة (double split): وقد سبق التطرق الى هذه الطريقة وشرحها سابقاً



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
line= 'From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
words = line.split()
email = words[1]
pieces = email.split('@')
print (pieces[1])

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
uct.ac.za
>>> |
```

المثال التاسع: لتطبيق نفس المطلوب في المثال السابق باستخدام التعابير القياسية:



```
3.py - C:\Users\mustafa\Desktop\3.py (3.4.3)
File Edit Format Run Options Window Help
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('@([^\ ]*)',lin)
print (y)

Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
['uct.ac.za']
>>> |
```

وللتوضيح لاحظ الشرح التالي:

'@([]*)'

ابحث داخل السلسلة حتى تجد الرمز @ ثم ارجع ما بعده حتى الفراغ

'@([]*)'

أي عدد من الرموز

طابق الرموز الغير فارغة non space

'@([]*)'

ارجع فقط ما بين القوسين الصغيرين

المثال العاشر: خيارات أكثر للبحث والمطابقة:

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*@([ ]*)',lin)
print y
['uct.ac.za']
```

'^From .*@([]*)'

'From' ابحث عن السلسلة الرمزية , ابدأ من بداية السطر

'^From .*@([]*)'

ابحث عن الرمز @ , اترك مجموعة من الرموز

'^From .*@ ([^]*)'

ابدأ المطابقة

'^From .*@ ([^]*)'

أي عدد منها 😊

طابق الرموز الغير فارغة

'^From .*@ ([^]*)'

أوقف المطابقة

الى هنا احبتي الكرام ينتهي الجزء الثاني من دورتنا المستمرة ان شاء الله وقد تناولنا في هذه الدورة كل ما يتعلق بأدوات الادخال والإخراج واساسيات البرمجة بلغة بايثون ومعالجة البيانات الرقمية والنصية بكل الطرق الممكنة. سيكون تركيزنا في الأجزاء القادمة من هذه الدورة ان شاء الله على البرمجة للتطبيقات التفاعلية (interactive applications) للحاسوب والهواتف النقالة ان شاء الله وربما نصل الى مرحلة البرمجة للتطبيقات الموزعة عبر الانترنت (الشبكات) وبرمجة مواقع الانترنت بلغة بايثون فتابعوا معنا كل جديد على مدونة مصطفى صادق العلمية على الرابط التالي:

[/ https://mustafasadiq0.wordpress.com](https://mustafasadiq0.wordpress.com)

مصطفى صادق لطيف 😊