Προγραμματισμός με τη γλώσσα python

Alexandros Kanterakis (mailto:kantale@ics.forth.gr) kantale@ics.forth.gr (mailto:kantale@ics.forth.gr)

Διάλεξη 5η, Τρίτη 12 Νοεμβρίου 2019

Παραλειπόμενα

Όταν γράφουμε: `a=b` , τότε στην `a` αποθηκεύεται η τιμή της `b` . Όταν όμως γράφουμε σκέτο: `b` τότε η τιμή της `b` αποθηκεύεται αυτόματα στη μεταβλητή: `_` :

In [1]:
```
a=42
print (a)
```

```
42
```

In [2]:
```
52
```

Out[2]: 52

In [3]:
```
_
```

Out[3]: 52

Προσοχή! Η extend και η append δεν επιστρέφουν τίποτα:

In [4]:
```
a = [1,2,3]
print (a.append(4))
```

```
None
```

In [5]:
```
print (a)
```

```
[1, 2, 3, 4]
```

In [6]:
```
a = [1,2,3]
print (a.extend([4]))
```

```
None
```

In [7]:
```
print (a)
```

```
[1, 2, 3, 4]
```

In [8]:
```
print (a)
```

```
[1, 2, 3, 4]
```

Γενικότερος κανόνας: Όταν γράφουμε: `XXX.YYY()` αν η μέθοδος `YYY` μεταβάλει το `XXX` τότε αυτή η μέδοδος δεν επιστρέφει τίποτα. Διαφορετικά επιστρέφει κάτι.

Δηλαδή οι παρακάτω μέθοδοι **δεν** επιστρέφουν `None`

In [9]:
```
"abcdfdfgsdfgsfg".count("f")
```

Out[9]: 4

In [10]:
```
a = "abcdfdfgsdfgsfg".count("f")
print (a)
```

```
4
```

Όταν γράφουμε `a = ....` τότε το a μεταβάλεται

In [11]:
```
c = ["kati asxeto"]
print (c)
a = [1,2,3] # Το a δεν μεταβάλεται
b = [4,5,6] # Το b  δεν μεταβάλεται
c = a+b # Μόνο το c μεταβάλεται
print (a)
print (b)
print (c)
```

```
['kati asxeto']
[1, 2, 3]
[4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

break και continue

Με την εντολή `break` "σταματάμε" την επανάληψη της for. Όταν ο υπολογιστής "δει" τη `break` τότε βγαίνει τελείως από τη `for` :

In [12]:
```
for i in range(1,10):
    print (i)
    if i>5:
        break # Get out from for !!!
```

```
1
2
3
4
5
6
```

Με την εντολή `continue` σταματάμε για αυτό και μόνο το κομμάτι της επανάληψης. Ο υπολογιστής "συνεχίζει" (continue..) με την επόμενη επανάληψη:

In [13]:
```
for i in range(1,10):
    if i == 5: # ΔΕΝ ΤΥΠΩΝΕΙ ΤΟ 5
        continue
    print (i) # TO PRINT **DEN** EINAI MESA STHN IF! (einai mesa sth for)
```

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

Προσοχή! ότι υπάρχει κάτω (και στο ίδιο indentation) από τη `continue` και τη `break` αγνοείται!

In [14]:
```
for i in range(1,10):
    if i == 5:
        continue
        print (i) #  TO PRINT EINAI MESA STHN IF!
```

η `break` και η `continue` μπορεί να χρησιμοποιηθεί και στη `while` .

In [15]:
```
a=1
while True:
    print (a)
    if a>5:
        break
    a +=1
```

```
1
2
3
4
5
6
```

In [16]:
```
a=1
while True:
    print (a)
    a +=1
    if a>5:
        break
```

```
1
2
3
4
5
```

In [17]:
```
a=1
counter =0
while True:
    counter += 1
    #print (a)
    a +=5
    if a>20:
        break
print ("Loop counter:",counter)
```

```
Loop counter: 4
```

In [18]:
```
a=1
counter =0
while True:
    counter += 1
    #print (a)
    if a>20:
        break
    a +=5
print ("Loop counter:",counter)
```

```
Loop counter: 5
```

Το απόλυτο ενός αριθμού:

In [19]:
```
abs(-3)
```

Out[19]: 3

In [20]:
```
abs(3)
```

Out[20]: 3

Ternary operator

με το ternary operator (https://en.wikipedia.org/wiki/%3F:) μπορούμε να γράψουμε μία `if ... else ...` σε μία εντολή. Η δομή είναι:

```
a = EXPRESSION_IF_THE_CONDITION_IS_TRUE if CONDITION else EXPRESSION_IF_TH
E_CONDITION_IS_FALSE
```

π.χ.:

In [21]:
```
a = 1 if 5<3 else 8
print (a)
```

```
8
```

In [22]:
```
a = 1 if 3<5 else 8
print (a)
```

```
1
```

Μπορούμε να συνθέσουμε πολλούς ternary operators μαζί:

In [23]:
```
a = (1 if 3<5 else 6) if 5>6 else (6 if 4>1 else 7)
print (a)
```

```
6
```

Αν βγάλω τις παρενθέσεις θα έχει διαφορετική τιμή η `a`. (Γιατί;)

In [24]:
```
a = 1 if 3<5 else 6 if 5>6 else 6 if 4>1 else 7
print (a)
```

```
1
```

Η συνάρτηση `input`

Με τη συνάρτηση `input` μπορούμε να πούμε στον χρήστη να εισάγει μία τιμή:

In [26]:
```
name = input('What is your name? ')
```

```
What is your name? Alex
```

In [27]:
```
print (name)
```

```
Alex
```

Η συνάρτηση `del`

Με την `del` μπορούμε να "σβήνουμε" τιμές από δομές δεδομένων:

In [28]:
```
a = [1,2,3]
print (a)
```

```
[1, 2, 3]
```

In [29]:
```
del a[1]
print (a)
```

```
[1, 3]
```

In [30]:
```
a = {'a':2, 'b':3, 'c': 4}
print (a)
```

```
{'a': 2, 'b': 3, 'c': 4}
```

In [31]:
```
del a['b']
print (a)
```

```
{'a': 2, 'c': 4}
```

In [32]:
```
a=3
print (a)
```

```
3
```

In [33]:
```
del a
print (a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-6a9b8a3d8df8> in <module>()
      1 del a
----> 2 print (a)

NameError: name 'a' is not defined
```

Μπορούμε να μετατρέψουμε μία λίστα με dictionary αν αποτελείται από υπολίστες όπου το κάθε στοιχείο της υπολίστας έχει δύο τιμές. Σε αυτή τη περίπτωση το πρώτο στοιχείο της υπολίστας είναι το κλειδί και το δεύτερο η τιμή:

In [34]:
```
a = [['name', 'alex'], ['age', 30]]
dict(a)
```

Out[34]:
```
{'name': 'alex', 'age': 30}
```

Παραλειπόμενα

assert

Με την `assert` ελέγχουνε αν "όλα πάνε καλά" σε κάποιο σημείο του προγράμματος:

In [83]:
```
assert True # Ολα καλα
```

In [85]:
```
assert False # Πρόβλημα!
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-85-c54b44bc6e13> in <module>()
----> 1 assert False # Πρόβλημα!

AssertionError:
```

Την `assert` τη χρησιμοποιούμε σαν quality control. π.χ. πριν από μία διαίρεση:

```
assert d != 0
result = 10/d

assert d>=0
result = math.sqrt(d)
```

lambda functions

Οι lambda functions είναι ειδικές συναρτήσεις που έχουν τις παρακάτω ιδιότητες:

- Δεν έχουν όνομα (https://en.wikipedia.org/wiki/Anonymous_function).
- Περιέχουν μόνο αυτό που κάνουν return. Δηλαδή δεν μπορούν να έχουν πάνω από μία γραμμές.

In [35]:
```
f = lambda x : x/2
f(10)
```

Out[35]:
```
5.0
```

Το παραπάνω είναι ισοδύναμο με:

Προσέχτε ότι μπορούμε να παραλείψουμε τελείως το γράμμα `f` που είναι το όνομα της συνάρτησης:

In [36]:
```
(lambda x : x/2)(10)
```

Out[36]:
```
5.0
```

Οι lambda συναρτήσεις έχουν ακριβώς τον ίδιο τύπο με τις "κανονικές" συναρτήσεις:

In [37]:
```
type(lambda x:x)
```

Out[37]:
```
function
```

Γιατί είναι χρήσιμες οι συναρτήσεις lambda; Πολλές φορές χρειάζεται να χρησιμοποιήσουμε μία συνάρτηση που κάνει κάτι απλό ή πρόκειται να τη χρησιμοποιήσουμε μία φορά, οπότε δεν υπάρχει λόγος να της δώσουμε όνομα ως μεταβλητή. Αυτό συμβαίνει συχνά όταν θέλουμε να δώσουμε μία συνάρτηση σαν όρισμα σε μία άλλη συνάρτηση, ή σε συναρτήσεις που επιστρέφουν συναρτήσεις ή σε συναρτήσεις που είναι μέσα σε λίστες και dictionaries.

Π.χ:

In [38]:
```
a = {
    'accurate': lambda x:x/3, # Δεκαδική διαίρεση με 3
    'not_accurate': lambda x:x//3 # Ακέραια διαίρεση με 3
}
print(a['accurate'](55))
print (a['not_accurate'](55))
```

```
18.333333333333332
18
```

Η συνάρτηση `map`

Η `map` "περνάει" όλα τα στοιχεία μιας λίστας από μία συνάρτηση.

In [39]:
```
def f(x):
    return x/10

print (list(map(f, [1,2,3,4,5])))
```

```
[0.1, 0.2, 0.3, 0.4, 0.5]
```

ή αλλιώς:

In [40]:
```
print (list(map(lambda x:x/10, [1,2,3,4,5])))
```

```
[0.1, 0.2, 0.3, 0.4, 0.5]
```

Τα δύο παραπάνω είναι ισοδύναμα με:

In [41]:
```
[f(x) for x in [1,2,3,4]]
```

Out[41]:
```
[0.1, 0.2, 0.3, 0.4]
```

In [42]:
```
[(lambda x:x/10)(x) for x in [1,2,3,4,5]]
```

Out[42]:
```
[0.1, 0.2, 0.3, 0.4, 0.5]
```

Ένα παράδειγμα:

Έστω ότι θέλουμε ότι βρούμε το συμπληρωματικό μιας ακολουθίας DNA. Δηλαδή αν η ακουλοθία μας είναι η "ACAGT" τότε η συμπληρωματική της είναι: "TGTCA".

Μπορούμε να φτιάξουμε μία συνάρτηση που επιστρέφει το συμπληρωματικό ενός νουκλεοτιδίου:

In [43]:
```
def invert_1(n):
    if n == 'A':
        return 'T'
    if n == 'T':
        return 'A'
    if n == 'C':
        return 'G'
    if n == 'G':
        return 'C'
```

Μια άλλη υλοποίηση της invert:

In [44]:
```
def invert_2(n):
    invert_dictionary = {
        'A': 'T',
        'T': 'A',
        'C': 'G',
        'G': 'C'
    }

    return invert_dictionary[n]
```

Υλοποίηση της invert με lambda:

In [45]:
```
invert_dictionary = {
        'A': 'T',
        'T': 'A',
        'C': 'G',
        'G': 'C'
    }
invert_3 = lambda x : invert_dictionary[x]
```

Ή αλλιώς:

In [46]:
```
invert_4 = lambda x : {'A':'T', 'T':'A', 'C':'G', 'G':'C'}[x]
```

Μπορούμε να χρησιμοποιήσουμε και ternary operator:

In [47]:
```
invert_5 = lambda x : 'A' if x == 'T' else 'T' if x == 'A' else 'C' if x=='G' else
```

Η `invert_1` , `invert_2` , `invert_3` , `invert_4` και `invert_5` είναι ισοδύναμες. Τώρα μπορούμε να χρησιμοποιήσουμε οποιαδίποτε από αυτές σε ένα string.

In [48]:
```
a = 'CGATACCGCTATTCGCTATCGATCGAACGAT'
```

In [49]:
```
#Method 1
''.join([invert_1(x) for x in a])
```

Out[49]: 'GCTATGGCGATAAGCGATAGCTAGCTTGCTA'

In [50]:
```
#Method 2
''.join(map(invert_2, a))
```

Out[50]: 'GCTATGGCGATAAGCGATAGCTAGCTTGCTA'

In [51]:
```
# Method 3
result = ''
for x in a:
    result += invert_3(x)
print (result)
```

```
GCTATGGCGATAAGCGATAGCTAGCTTGCTA
```

In [52]:
```
# Method 4
sequence_inverter = lambda x : ''.join(map(invert_5, x))
sequence_inverter(a)
```

Out[52]: 'GCTATGGCGATAAGCGATAGCTAGCTTGCTA'

Μία άλλη προσέγγιση είναι να χρησιμοποιήσουμε τη `replace` . Φτιάχνουμε μία συνάρτηση η οποία αντιστρέφει ένα ζευγάρι από γραμματα (π.χ: `letter_1` , `letter_2`) σε ένα string. Προσέχτε ότι δεν μπορούμε να κάνουμε:

```
string.replace(letter_1, letter_2).replace(letter_2, letter_1)
```

(γιατί;)

Πρέπει να χρησιμοποιήσουμε ένα άσχετο string (στη περίπτωσή μας το `@`), και να αντικαθαστίσουμε το `letter_1` με το `@` , στη συνέχεια το `letter_2` με το `letter_1` και στο τέλος το `@` με το `letter_2`

In [53]:
```
def swap_letters(string, letter_1, letter_2):
    return string.replace(letter_1, '@').replace(letter_2, letter_1).replace('@',
```

Τώρα μπορούμε να φτιάξουμε μία συνάρτηση που εναλλάσει τα `A` με `T` (και το αντίστροφο) και τα `C` με `G` και το αντίστροφο:

In [54]:
```
def sequence_inverter_2(sequence):
    invert_AT = swap_letters(sequence, 'A', 'T')
    invert_ATCG = swap_letters(invert_AT, 'C', 'G')

    return invert_ATCG

sequence_inverter_2(a)
```

Out[54]:
```
'GCTATGGCGATAAGCGATAGCTAGCTTGCTA'
```

φυσικά μπορούμε να κάνουμε το κάνουμε και με lambda:

In [55]:
```
sequence_inverter_3 = lambda x : swap_letters(swap_letters(x, 'A', 'T'), 'C', 'G')
print (sequence_inverter_3(a))
```

```
GCTATGGCGATAAGCGATAGCTAGCTTGCTA
```

Και ένας τελευταίος τρόπος χρησιμοποιώντας μόνο lambda και τη `replace` :

In [57]:
```
sequence_inverter_4 = lambda x : x.replace('A', '@').replace('T', 'A').replace('@'
sequence_inverter_4(a)
```

Out[57]:
```
'GCTATGGCGATAAGCGATAGCTAGCTTGCTA'
```

Generators

Οι generators (γεννήτριες) είναι δομές παρόμοιες με τις λίστες με τη διαφορά, ότι στη μνήμη του υπολογιστή δεν κρατάει όλες τις τιμές της λίστας, αλλά τον κώδικα που χρειάζεται για την αναπαραγωγή τους.

Παρόλο που δεν το έχουμε τονίσει, οι συναρτήσεις `range` , `enumerate` , `items` , `map` που έχουμε παρουσιάσει επιτρέφουν generator.

Για να καταλάβουμε καλύτερα τους generators ας χρησιμοποιήσουμε ένα ανθρώπινο παράδειγμα. Ας υποθέσουμε οτι κάποιος σας λέει: απομνημόνευσε τους παρακάτω 100 αριθμούς τηλεφώνου με τη σειρά που στους δίνω. Αφού το κάνεις αυτό, πες τον πρώτο αριθμό, μετά τον δεύτερο, κτλ..

Αυτό θα απαιτούσε τεράστια προσπάθεια (και μνήμη) από εσάς.

Ας υποθέσουμε τώρα ότι κάποιος σας λέει: απομνημόνευσε όλους τους ζυγούς αριθμούς από το 2 μέχρι το 1000. Αφού το κάνεις αυτό πες μου τον πρώτο αριθμό, μετά τον δεύτερο κτλ..

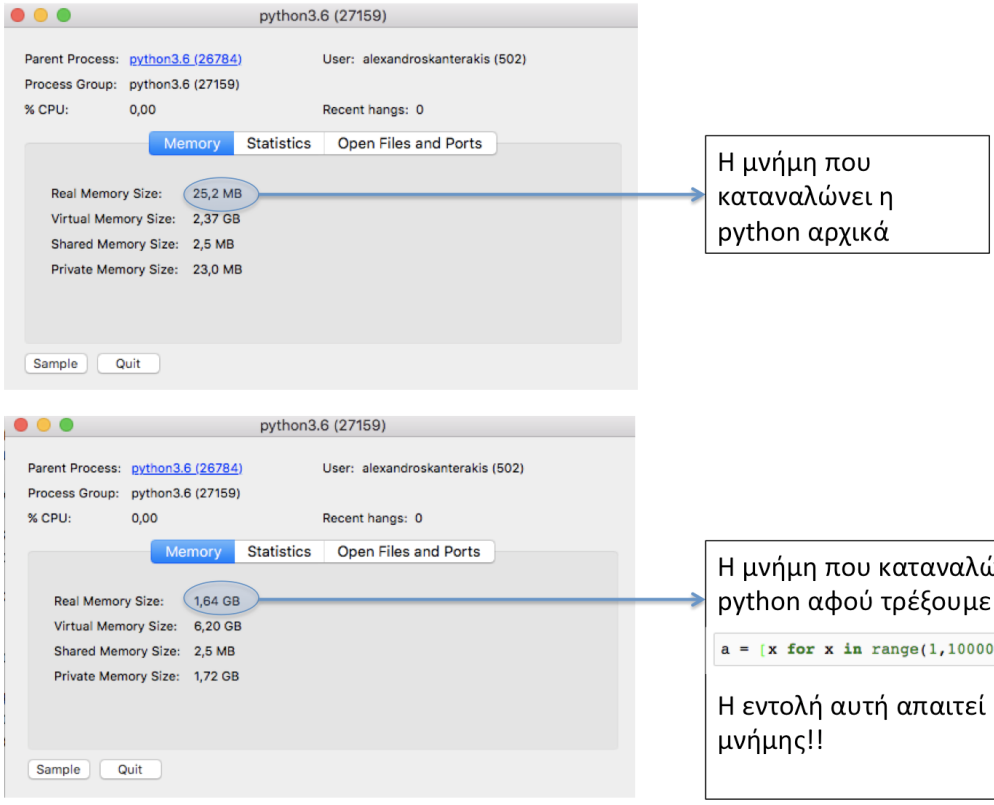
Δεν χρειάζεται ιδιαίτερος κόπος για αυτό. Απλά πρέπει να θυμάσαι που πρέπει να αρχίσεις (το 2), να τελειώσεις (το 1000), σε ποιο σημείο είσαι τώρα και με ποιο τρόπο βρίσκεις το επόμενο. Αυτό ακριβώς κάνουν οι generators. Είναι χρήσιμες σε περιπτώσεις που το επόμενο στοιχείο μιας λίστας μπορεί να υπολογιστεί και όχι να ανακληθεί από τη μνήμη.

Έτσι όταν λέμε:

```
range(1,1000000)
```

Δεν υπάρχει λόγος να αποθηκεύσουμε 1.000.000 τιμές στη μνήμη του υπολογιστή. Ας το δούμε στη πράξη. Αν τρέξουμε:

```
a = [x for x in range(1,100000000)]
```



βλέπουμε ότι η python χρειάζεται 1.5GB μνήμης. Αν χρειάζεται να παίρνουμε αυθαίρετα τιμές από τη λίστα `a` τότε δεν μπορούμε να κάνουμε αλλιώς. Διαφορετικά αν χρειάζεται απλά να παίρνουμε τη μία τιμή μετά την άλλη τότε μπορούμε να γλυτώσουμε αυτή τη μνήμη γράφοντας:

```
for x in range(1, 1000000):
    ...
```

Μπορούμε να φτιάξουμε τον δικό μας generator με δύο τρόπους: Ο πρώτος είναι μέσω generator comprehension και ο δεύτερος είναι μέσω συναρτήσεων που αντί να κάνουν `return` κάνουν `yield` .

Τα generator comprehension είναι ακριβώς όπως και τα list comprehensions (και τα dictionary, set comprehension) απλά χρησημοποιούμε τις παρενθέσεις:

In [58]:
```
my_generator = (x for x in range(1,10))
type(my_generator)
```

Out[58]:
```
generator
```

Ας πάρουμε τη πρώτη τιμή του generator:

In [59]:
```
next(my_generator)
```

Out[59]:
```
1
```

Ας πάρουμε την επόμενη:

In [60]:
```
next(my_generator)
```

Out[60]:
```
2
```

... κτλ.. Μπορούμε να πάρουμε τις υπόλοιπες τιμές του generator:

In [61]:
```
for x in my_generator:
    print (x)
```

```
3
4
5
6
7
8
9
```

Βλέπουμε ότι ο generator "κρατάει" σε ποια τιμή ήμασταν και υπολογίζει την επόμενη. Όταν τελειώσουν οι τιμές του generator τότε δεν μπορούμε να πάρουμε άλλη:

In [62]:
```
next(my_generator)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-62-beb7403f481d> in <module>()
----> 1 next(my_generator)

StopIteration:
```

Ο δεύτερος τρόπος να φτιάξουμε generator είναι να δηλώσουμε μία συνάρτηση με τη `def` και αντί για `return` να κάνουμε `yield` (https://www.youtube.com/watch?v=H2KYzeT_lGY):

In [63]:
```
def f():
    yield "first"
    yield "second"
    yield "third"
```

Προσοχή η `f` είναι μία συνάρτηση που φτιάχνει έναν generator! Η `f` δεν είναι generator!

In [64]:
```
type(f)
```

Out[64]:
```
function
```

In [65]:
```
my_generator = f()
type(my_generator)
```

Out[65]:
```
generator
```

Το my_generator είναι generator. Ας τον "τρέξουμε":

In [66]:
```
next(my_generator)
```

Out[66]:
```
'first'
```

In [67]:
```
next(my_generator)
```

Out[67]:
```
'second'
```

In [68]:
```
next(my_generator)
```

Out[68]:
```
'third'
```

Ή αλλιώς:

In [69]:
```
for x in f():
    print (x)
```

```
first
second
third
```

Files

Έστω ότι στο ίδιο directory που τρέχουμε τη python (ή το jupyter notebook) έχουμε το παρακάτω αρχείο με το όνομα: `test.txt`

```
aaa
bbb
ccc
```

Το αρχείο αυτό μπορώ να το ανοίξω με:

In [70]:
```
%%bash
echo "aaa" >> test.txt
echo "bbb" >> test.txt
echo "ccc" >> test.txt
```

In [71]:
```
!cat test.txt
```

```
aaa
bbb
ccc
```

In [72]:
```
f = open('test.txt')
```

Αφού το ανοίξω μπορώ να διαβάσω όλα τα περιεχόμενά του:

In [73]:
```
a= f.read()
print (a)
```

```
aaa
bbb
ccc
```

Αν επιχειρήσω να ξαναδιαβάσω το αρχείο τότε θα μου επιστρέψει ένα άδειο string:

In [74]:
```
f.read()
```

Out[74]: ''

Όταν διαβάζουμε ένα αρχείο με τη python και φτάνουμε στο τέλος, τότε η python δεν συνεχίζει να το διαβάζει από την αρχή. Αντίθετα επιτρέφει ένα άδειο string.

Για να ξαναδιαβάσουμε ένα αρχείο πρέπει να το κλείσουμε:

In [75]:
```
f.close()
```

Και να το ξανα-ανοίξουμε:

In [76]:
```
f = open('test.txt')
a= f.read()
print (a)
```

```
aaa
bbb
ccc
```

Μπορούμε να διαβάσουμε μία μόνο γραμμή από το αρχείο:

In [77]:
```
f = open('test.txt')
line = f.readline()
print (line)
```

```
aaa
```

Παρατηρείστε ότι η `line` περιέχει και το enter (`'\n'`) που υπάρχει στο τέλος της γραμμής:

In [78]:
```
line
```

Out[78]: 'aaa\n'

Αν καλέσουμε τη `readline` ξανά, τότε θα διαβάσει την επόμενη γραμμή:

In [79]:
```
f.readline()
```

Out[79]: 'bbb\n'

In [80]:
```
f.readline()
```

Out[80]: 'ccc\n'

Αν φτάσουμε στο τέλος τότε η `f.readline()` επιστρέφει το άδειο string:

In [81]:
```
f.readline()
```

Out[81]:
```
''
```

Επίσης μπορούμε να κάνουμε iterate (δηλαδή να εφαρμόσουμε τη `for`) σε ένα αρχείο που έχουμε ανοίξει:

In [82]:
```
f = open('test.txt')
for line in f:
    print (line)
```

```
aaa

bbb

ccc
```

In [83]:
```
f = open('test.txt')
for line in f:
    print (line.replace('\n', ''))
```

```
aaa
bbb
ccc
```

Επειδή παρατηρήθηκε ότι πολλές φορές οι προγραμματιστές ξεχνάγαν να κλείσουν ένα αρχείο η python συνιστά να χρησιμοποιούμε τη `with open() as f:` :

In [84]:
```
with open('test.txt') as f:
    for l in f:
        print (l.replace('\n', ''))
```

```
aaa
bbb
ccc
```

Όσες φορές και να τρέξω τις παραπάνω εντολές παράγεται το ίδιο αποτέλεσμα.

Μπορώ με αυτόν τον τρόπο να ανοίξω παραπάνω από ένα αρχίο με το ίδιο `with` . Ας υποθέσουμε ότι έχουμε τα αρχεία: test_1.txt, test_2.txt:

In [86]:
```
!cp test.txt test_1.txt
!cp test.txt test_2.txt
```

In [87]:
```
with open('test_1.txt') as f1, open('test_2.txt') as f2:
    for l in f1:
        print (l)

    for l in f2:
        print (l)
```

```
aaa

bbb

ccc

aaa

bbb

ccc
```

Παράδειγμα

Σε αυτό το link (https://pastebin.com/dl/Wq2PpbZy) υπάρχει ένα αρχείο με το πολύ γνωστό IRIS dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set). Κατεβάστε το και σώστε το στο ίδιο directory με τη python. **ΠΡΟΣΟΧΗ!** Αυτό το αρχείο δεν πρέπει να είναι το ίδιο με το αυθεντικό γιατί περιέχει μόνο 149 δεδομένα (αντί για 150) που έχει το αυθεντικό. Για τους σκοπούς του παραδείγματος αυτού όμως αυτό δεν μας επηρεάζει.

Το όνομα του αρχείο είναι: `Wq2PpbZy.txt` .

Ας παίξουμε μαζί του!

Αρχικά το ανοίγουμε:

In [91]:
```
! wget -O Wq2PpbZy.txt https://pastebin.com/raw/Wq2PpbZy
```

```
--2019-11-06 14:25:17--  https://pastebin.com/raw/Wq2PpbZy (https://pastebin.com/raw/Wq2PpbZy)
Resolving pastebin.com (pastebin.com)... 104.22.3.84, 104.22.2.84
Connecting to pastebin.com (pastebin.com)|104.22.3.84|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'Wq2PpbZy.txt'

Wq2PpbZy.txt            [ <=>                ]   4.91K  --.-KB/s    in 0s

2019-11-06 14:25:18 (14.8 MB/s) - 'Wq2PpbZy.txt' saved [5027]
```

In [92]:
```
with open('Wq2PpbZy.txt') as f:
    data = f.read()
print(data)
```

```
Sepal length    Sepal width     Petal length    Petal width     Species
5.2     3.5     1.4     0.2     I. setosa
4.9     3.0     1.4     0.2     I. setosa
4.7     3.2     1.3     0.2     I. setosa
4.6     3.1     1.5     0.2     I. setosa
5.0     3.6     1.4     0.3     I. setosa
5.4     3.9     1.7     0.4     I. setosa
4.6     3.4     1.4     0.3     I. setosa
5.0     3.4     1.5     0.2     I. setosa
4.4     2.9     1.4     0.2     I. setosa
4.9     3.1     1.5     0.1     I. setosa
5.4     3.7     1.5     0.2     I. setosa
4.8     3.4     1.6     0.2     I. setosa
4.8     3.0     1.4     0.1     I. setosa
4.3     3.0     1.1     0.1     I. setosa
5.8     4.0     1.2     0.2     I. setosa
5.7     4.4     1.5     0.4     I. setosa
5.4     3.9     1.3     0.4     I. setosa
5.1     3.5     1.4     0.3     I. setosa
```

Μπορούμε να πάρουμε μία λίστα με όλες τις γραμμές:

In [93]:
```
data = data.split('\n')
data
```

Out[93]:
```
['Sepal length \tSepal width \tPetal length \tPetal width \tSpecies',
 '5.2 \t3.5 \t1.4 \t0.2 \tI. setosa',
 '4.9 \t3.0 \t1.4 \t0.2 \tI. setosa',
 '4.7 \t3.2 \t1.3 \t0.2 \tI. setosa',
 '4.6 \t3.1 \t1.5 \t0.2 \tI. setosa',
 '5.0 \t3.6 \t1.4 \t0.3 \tI. setosa',
 '5.4 \t3.9 \t1.7 \t0.4 \tI. setosa',
 '4.6 \t3.4 \t1.4 \t0.3 \tI. setosa',
 '5.0 \t3.4 \t1.5 \t0.2 \tI. setosa',
 '4.4 \t2.9 \t1.4 \t0.2 \tI. setosa',
 '4.9 \t3.1 \t1.5 \t0.1 \tI. setosa',
 '5.4 \t3.7 \t1.5 \t0.2 \tI. setosa',
 '4.8 \t3.4 \t1.6 \t0.2 \tI. setosa',
 '4.8 \t3.0 \t1.4 \t0.1 \tI. setosa',
 '4.3 \t3.0 \t1.1 \t0.1 \tI. setosa',
 '5.8 \t4.0 \t1.2 \t0.2 \tI. setosa',
 '5.7 \t4.4 \t1.5 \t0.4 \tI. setosa',
 '5.4 \t3.9 \t1.3 \t0.4 \tI. setosa',
 '5.1 \t3.5 \t1.4 \t0.3 \tI. setosa',
```

Φαίνεται ότι το αρχείο διαχωρίζει τις κολόνες με tabs. Ας το κάνουμε split με βάση τα tabs:

In [94]:
```
data = [x.split('\t') for x in data]
data
```

Out[94]:
```
[['Sepal length ', 'Sepal width ', 'Petal length ', 'Petal width ', 'Species'],
 ['5.2 ', '3.5 ', '1.4 ', '0.2 ', 'I. setosa'],
 ['4.9 ', '3.0 ', '1.4 ', '0.2 ', 'I. setosa'],
 ['4.7 ', '3.2 ', '1.3 ', '0.2 ', 'I. setosa'],
 ['4.6 ', '3.1 ', '1.5 ', '0.2 ', 'I. setosa'],
 ['5.0 ', '3.6 ', '1.4 ', '0.3 ', 'I. setosa'],
 ['5.4 ', '3.9 ', '1.7 ', '0.4 ', 'I. setosa'],
 ['4.6 ', '3.4 ', '1.4 ', '0.3 ', 'I. setosa'],
 ['5.0 ', '3.4 ', '1.5 ', '0.2 ', 'I. setosa'],
 ['4.4 ', '2.9 ', '1.4 ', '0.2 ', 'I. setosa'],
 ['4.9 ', '3.1 ', '1.5 ', '0.1 ', 'I. setosa'],
 ['5.4 ', '3.7 ', '1.5 ', '0.2 ', 'I. setosa'],
 ['4.8 ', '3.4 ', '1.6 ', '0.2 ', 'I. setosa'],
 ['4.8 ', '3.0 ', '1.4 ', '0.1 ', 'I. setosa'],
 ['4.3 ', '3.0 ', '1.1 ', '0.1 ', 'I. setosa'],
 ['5.8 ', '4.0 ', '1.2 ', '0.2 ', 'I. setosa'],
 ['5.7 ', '4.4 ', '1.5 ', '0.4 ', 'I. setosa'],
 ['5.4 ', '3.9 ', '1.3 ', '0.4 ', 'I. setosa'],
 ['5.1 ', '3.5 ', '1.4 ', '0.3 ', 'I. setosa'],
```

Παρατηρούμε ότι υπάρχουν spaces (κενά) στο τέλος κάποιων strings. Ας τα "περάσουμε" όλα από ένα `strip` :

In [95]:
```
data = [[y.strip() for y in x] for x in data]
data
```

Out[95]:
```
[['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'Species'],
 ['5.2', '3.5', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.0', '1.4', '0.2', 'I. setosa'],
 ['4.7', '3.2', '1.3', '0.2', 'I. setosa'],
 ['4.6', '3.1', '1.5', '0.2', 'I. setosa'],
 ['5.0', '3.6', '1.4', '0.3', 'I. setosa'],
 ['5.4', '3.9', '1.7', '0.4', 'I. setosa'],
 ['4.6', '3.4', '1.4', '0.3', 'I. setosa'],
 ['5.0', '3.4', '1.5', '0.2', 'I. setosa'],
 ['4.4', '2.9', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.1', '1.5', '0.1', 'I. setosa'],
 ['5.4', '3.7', '1.5', '0.2', 'I. setosa'],
 ['4.8', '3.4', '1.6', '0.2', 'I. setosa'],
 ['4.8', '3.0', '1.4', '0.1', 'I. setosa'],
 ['4.3', '3.0', '1.1', '0.1', 'I. setosa'],
 ['5.8', '4.0', '1.2', '0.2', 'I. setosa'],
 ['5.7', '4.4', '1.5', '0.4', 'I. setosa'],
 ['5.4', '3.9', '1.3', '0.4', 'I. setosa'],
 ['5.1', '3.5', '1.4', '0.3', 'I. setosa'],
```

Φαίνεται ότι το πρώτο στοιχείο της λίστας είναι το header. Ας το πάρουμε:

In [96]:
```
header = data[0]
header
```

Out[96]:
```
['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'Species']
```

Επίσης φαίνεται ότι όλα τα υπόλοιπα στοιχεία (εκτός το πρώτο) είναι τα δεδομένα. Ας πάρουμε μόνο αυτά:

In [97]:
```
data = data[1:]
data
```

Out[97]:
```
[['5.2', '3.5', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.0', '1.4', '0.2', 'I. setosa'],
 ['4.7', '3.2', '1.3', '0.2', 'I. setosa'],
 ['4.6', '3.1', '1.5', '0.2', 'I. setosa'],
 ['5.0', '3.6', '1.4', '0.3', 'I. setosa'],
 ['5.4', '3.9', '1.7', '0.4', 'I. setosa'],
 ['4.6', '3.4', '1.4', '0.3', 'I. setosa'],
 ['5.0', '3.4', '1.5', '0.2', 'I. setosa'],
 ['4.4', '2.9', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.1', '1.5', '0.1', 'I. setosa'],
 ['5.4', '3.7', '1.5', '0.2', 'I. setosa'],
 ['4.8', '3.4', '1.6', '0.2', 'I. setosa'],
 ['4.8', '3.0', '1.4', '0.1', 'I. setosa'],
 ['4.3', '3.0', '1.1', '0.1', 'I. setosa'],
 ['5.8', '4.0', '1.2', '0.2', 'I. setosa'],
 ['5.7', '4.4', '1.5', '0.4', 'I. setosa'],
 ['5.4', '3.9', '1.3', '0.4', 'I. setosa'],
 ['5.1', '3.5', '1.4', '0.3', 'I. setosa'],
 ['5.7', '3.8', '1.7', '0.3', 'I. setosa'],
```

Παρατηρώ ότι οι τιμές στις 4 πρώτες στήλες είναι σε string. Τις μετατρέπουμε σε float:

In [98]:
```
data = [list(map(float, x[:4])) + [x[-1]]  for x in data]
data
```

Out[98]:
```
[[5.2, 3.5, 1.4, 0.2, 'I. setosa'],
 [4.9, 3.0, 1.4, 0.2, 'I. setosa'],
 [4.7, 3.2, 1.3, 0.2, 'I. setosa'],
 [4.6, 3.1, 1.5, 0.2, 'I. setosa'],
 [5.0, 3.6, 1.4, 0.3, 'I. setosa'],
 [5.4, 3.9, 1.7, 0.4, 'I. setosa'],
 [4.6, 3.4, 1.4, 0.3, 'I. setosa'],
 [5.0, 3.4, 1.5, 0.2, 'I. setosa'],
 [4.4, 2.9, 1.4, 0.2, 'I. setosa'],
 [4.9, 3.1, 1.5, 0.1, 'I. setosa'],
 [5.4, 3.7, 1.5, 0.2, 'I. setosa'],
 [4.8, 3.4, 1.6, 0.2, 'I. setosa'],
 [4.8, 3.0, 1.4, 0.1, 'I. setosa'],
 [4.3, 3.0, 1.1, 0.1, 'I. setosa'],
 [5.8, 4.0, 1.2, 0.2, 'I. setosa'],
 [5.7, 4.4, 1.5, 0.4, 'I. setosa'],
 [5.4, 3.9, 1.3, 0.4, 'I. setosa'],
 [5.1, 3.5, 1.4, 0.3, 'I. setosa'],
 [5.7, 3.8, 1.7, 0.3, 'I. setosa'],
```

Πόσα συνολικά species υπάρχουν;

In [99]:
```
species = set([x[-1] for x in data])
species
```

Out[99]:
```
{'I. setosa', 'I. versicolor', 'I. virginica'}
```

Πόσα data έχουμε για κάθε species;

In [100]:
```
b = [(x, sum([y[-1]==x for y in data])) for x in species]
b
```

Out[100]:
```
[('I. virginica', 49), ('I. setosa', 50), ('I. versicolor', 50)]
```

Αυτό μπορούμε να το μετατρέψουμε και σε dictionary

In [101]:
```
dict(b)
```

Out[101]:
```
{'I. virginica': 49, 'I. setosa': 50, 'I. versicolor': 50}
```

Ποιος είναι ο μέσος όρος του "Sepal Length" για όλα τα data;

In [102]:
```
def average(x):
    return sum(x)/len(x)
```

In [103]:
```
average([x[header.index('Sepal length')] for x in data])
```

Out[103]:
```
5.831543624161076
```

Ποιος είναι ο μέσος όρος του "Sepal length" για κάθε species ξεχωριστά;

In [104]:
```
[(x, average([y[0] for y in data if y[-1] == x])) for x in species]
```

Out[104]:
```
[('I. virginica', 6.565306122448979),
 ('I. setosa', 5.007999999999999),
 ('I. versicolor', 5.936)]
```

Πόσα "I. setosa" έχουν Petal length > 1.5 ;

In [105]:
```
sum([x[2] > 1.5 and x[-1] == 'I. setosa' for x in data])
```

Out[105]:
```
13
```

ή αλλιώς:

In [106]:
```
[x[2] > 1.5 and x[-1] == 'I. setosa' for x in data].count(True)
```

Out[106]:
```
13
```

Ή αλλιώς:

In [107]:
```
len([None for x in data if x[2] > 1.5 and x[-1] == 'I. setosa'])
```

Out[107]:
```
13
```

Ποια είναι αυτά;

In [108]:
```
[x for x in data if x[2] > 1.5 and x[-1] == 'I. setosa']
```

Out[108]:
```
[[5.4, 3.9, 1.7, 0.4, 'I. setosa'],
 [4.8, 3.4, 1.6, 0.2, 'I. setosa'],
 [5.7, 3.8, 1.7, 0.3, 'I. setosa'],
 [5.4, 3.4, 1.7, 0.2, 'I. setosa'],
 [5.1, 3.3, 1.7, 0.5, 'I. setosa'],
 [4.8, 3.4, 1.9, 0.2, 'I. setosa'],
 [5.0, 3.0, 1.6, 0.2, 'I. setosa'],
 [5.0, 3.4, 1.6, 0.4, 'I. setosa'],
 [4.7, 3.2, 1.6, 0.2, 'I. setosa'],
 [4.8, 3.1, 1.6, 0.2, 'I. setosa'],
 [5.0, 3.5, 1.6, 0.6, 'I. setosa'],
 [5.1, 3.8, 1.9, 0.4, 'I. setosa'],
 [5.1, 3.8, 1.6, 0.2, 'I. setosa']]
```

Ποια είναι τα indexes των παραπάνω (I. setosa που έχουν Petal length > 1.5)

In [109]:
```
[i for i,x in enumerate(data) if x[2]>1.5 and x[-1] == 'I. setosa']
```

Out[109]:
```
[5, 11, 18, 20, 23, 24, 25, 26, 29, 30, 43, 44, 46]
```

Ποιο είναι το minimum Sepal length για όλα;

In [110]:
```
min(data, key=lambda x:x[header.index('Sepal length')])
```

Out[110]:
```
[4.3, 3.0, 1.1, 0.1, 'I. setosa']
```

Που είναι το mimimum Sepal Length για όλα τα είδη;

In [111]:
```
[min([(y[0], y) for y in data if y[-1] == x]) for x in species]
```

Out[111]:
```
[(4.9, [4.9, 2.5, 4.5, 1.7, 'I. virginica']),
 (4.3, [4.3, 3.0, 1.1, 0.1, 'I. setosa']),
 (4.9, [4.9, 2.4, 3.3, 1.0, 'I. versicolor'])]
```

Ποιο είναι το index των παραπάνω;

In [112]:
```
[min([(y[0], y, i) for i,y in enumerate(data) if y[-1] == x]) for x in species]
```

Out[112]:
```
[(4.9, [4.9, 2.5, 4.5, 1.7, 'I. virginica'], 106),
 (4.3, [4.3, 3.0, 1.1, 0.1, 'I. setosa'], 13),
 (4.9, [4.9, 2.4, 3.3, 1.0, 'I. versicolor'], 57)]
```

Ποιο είναι το εύρος (δηλαδή το μικρότερο και το μεγαλύτερο) του "Sepal Length" για κάθε είδος ξεχωριστά;

In [113]:
```
# Επιστρέφει το εύρος μίας λίστας
def my_range(l):
    return min(l), max(l)

my_range([4,5,6,3,4,7,8,9])
```

Out[113]:
```
(3, 9)
```

In [114]:
```
[ (my_range([y[0] for y in data if y[-1] ==x]), x) for x in species]
```

Out[114]:
```
[((4.9, 7.9), 'I. virginica'),
 ((4.3, 5.8), 'I. setosa'),
 ((4.9, 7.0), 'I. versicolor')]
```

Ας βάλουμε και τα indexes των μικρότερων και μεγαλύτερων:

In [115]:
```
# Επιστρέφει το index του μεγαλύτερου στοιχείου
def max_index(l):
    return max(range(len(l)), key=lambda x : l[x])

max_index([3,6,5,8,7])
```

Out[115]:
```
3
```

In [116]:
```
# Επιστρέφει το index του μικρότερου στοιχείου
def min_index(l):
    return min(range(len(l)), key=lambda x : l[x])

min_index([3,6,5,8,7])
```

Out[116]:
```
0
```

In [117]:
```
def my_range_2(l):
    return 'min:{} min_index:{} max:{} max_index: {}'.format(
        min(l), min_index(l), max(l), max_index(l)
    )
```

In [118]:
```
[ (my_range_2([y[0] for y in data if y[-1] ==x]), x) for x in species]
```

Out[118]:
```
[('min:4.9 min_index:6 max:7.9 max_index: 31', 'I. virginica'),
 ('min:4.3 min_index:13 max:5.8 max_index: 14', 'I. setosa'),
 ('min:4.9 min_index:7 max:7.0 max_index: 0', 'I. versicolor')]
```

Ποιο dataset έχει το μεγαλύτερο εμβαδό sepal. Ως εμβαδό ορίζουμε το γινόμε Sepal width*Sepal length

In [119]:
```
max([(x[0]*x[1], x, i) for i,x in enumerate(data)])
```

Out[119]:
```
(30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131)
```

Ποιο dataset έχει το μεγαλύτερο εμβαδό για κάθε είδος;

Φτιάχνουμε μία συνάρτηση η οποία παίρνει μία λίστα από data και υπολογίζει ποιο από αυτα έχει το μεγαλύτερο εμβαδό:

In [120]:
```
f = lambda d : max([(x[0]*x[1], x, data.index(x)) for x in d])
```

π.χ: από όλα τα data, το μεγαλύτερο εμβαδό το έχει:

In [121]:
```
f(data)
```

Out[121]:
```
(30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131)
```

Το εφαρμόζουμε για κάθε είδος ξεχωριστά:

In [122]:
```
[f([y for y in data if y[-1] == x]) for x in species]
```

Out[122]:
```
[(30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131),
 (25.080000000000002, [5.7, 4.4, 1.5, 0.4, 'I. setosa'], 15),
 (22.400000000000002, [7.0, 3.2, 4.7, 1.4, 'I. versicolor'], 50)]
```

In [ ]:
```
```

Δημιουργία αρχείων

Την προηγούμενη φορά είπαμε πως ανοίγουμε αρχεία:

In [87]:
```
f=open('test.txt')
f.read()
f.close()
```

Μπορούμε να δημιουργήσουμε ένα καινούργιο αρχείο:

In [88]:
```
f=open('results.txt', 'w')
```

Προσέχτε το `'w'` .

Προσοχή!!! Αν το αρχείο `results.txt` υπάρχει ήδη τότε το διαγράφει!

Μπορούμε να κλείσουμε ένα αρχείο με τον ίδιο τρόπο:

In [90]:
```
f.close()
```

Σε ένα αρχείο αποθηκεύουμε μόνο string μεταβλητές με την `f.write()` :

In [92]:
```
f = open('results.txt', 'w')
f.write('Hello world\n')
f.close()
```

Επίσης μπορούμε να χρησιμοποιήσουμε την `with` :

In [93]:
```
a = "Hello world!"
with open('results.txt', 'w') as f:
    f.write(a)
    f.write('\n')
```

Μπορούμε επίσης να χρησιμοποιήσουμε πολλές `open` μέσα στη `with` για να ανοίξουμε πολλά αρχεία. Για παράδειγμα η παρακάτω συνάρτηση αντιγράφει ένα αρχείο σε ένα άλλο:

In [95]:
```
def my_copy(from_file, to_file):
    with open(from_file) as f_from, open(to_file, 'w') as f_to:
        data = f_from.read()
        f_to.write(data)

my_copy('results.txt', 'new_results.txt')
```

Εκτέλεση μίας εντολής απο το jupyter:

Γράφοντας τον χαρακτήρα `!` στην αρχή ενός κελιού στο jupyter, εκτελείται η εντολή που περιέχει:

In [97]:
```
! ls results.txt new_results.txt
```

```
new_results.txt results.txt
```

In [98]:
```
! uptime
```

```
16:40  up 41 days,  6:09, 3 users, load averages: 1.44 1.58 1.59
```

Η συνάρτηση `zip`

Με τη `zip` μπορούμε να 'ενώσουμε΄ δύο λίστες σε μία λίστα από υπολίστες:

In [99]:
```
list(zip([1,2,3], ['a', 'b', 'c']))
```

Out[99]:
```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Αυτό είναι χρήσιμο όταν θέλουμε να φτιάξουμε ένα dictionary όπου σε μία λίστα έχουμε τα κλειδιά και σε μία άλλη τις τιμές:

In [100]:
```
dict(zip([1,2,3], ['a', 'b', 'c']))
```

Out[100]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

Flatening

Με το παρακάτω "κόλπο" μπορούμε να κάνουμε μία λίστα από λίστες , μία απλή λίστα:

In [102]:
```
a=[[1,2], [3,4], [5,6]]
[y for x in a for y in x]
```

Out[102]:
```
[1, 2, 3, 4, 5, 6]
```

Για να κάνουμε το αντίθετο:

In [104]:
```
a = [1,2,3,4,5,6]
[[a[x], a[x+1]] for x in range(0, len(a), 2)]
```

Out[104]:
```
[[1, 2], [3, 4], [5, 6]]
```