Οι μέθοδοι `split` και `join`

Αν θέλουμε να "σπάσουμε" ένα string σε μία λίστα από πολλά string τότε μπορούμε να χρησιμοποιήσουμε τη `split`

In [57]:
```
"a+b+c".split('+')
```

Out[57]:
```
['a', 'b', 'c']
```

In [58]:
```
"hello world".split(' ')
```

Out[58]:
```
['hello', 'world']
```

In [59]:
```
"I like to move it move it".split('move')
```

Out[59]:
```
['I like to ', ' it ', ' it']
```

In [60]:
```
a = '''
ἄνδρα μοι ἔννεπε, μοῦσα, πολύτροπον, ὃς μάλα πολλὰ
πλάγχθη, ἐπεὶ Τροίης ἱερὸν πτολίεθρον ἔπερσεν·
πολλῶν δ' ἀνθρώπων ἴδεν ἄστεα καὶ νόον ἔγνω,
πολλὰ δ' ὅ γ' ἐν πόντῳ πάθεν ἄλγεα ὃν κατὰ θυμόν,
ἀρνύμενος ἥν τε ψυχὴν καὶ νόστον ἑταίρων.
ἀλλ' οὐδ' ὣς ἑτάρους ἐρρύσατο, ἱέμενός περ·
αὐτῶν γὰρ σφετέρῃσιν ἀτασθαλίῃσιν ὄλοντο,
νήπιοι, οἳ κατὰ βοῦς Ὑπερίονος Ἠελίοιο
ἤσθιον· αὐτὰρ ὁ τοῖσιν ἀφείλετο νόστιμον ἦμαρ.
'''
a.split('\n')
```

Out[60]:
```
['',
 'ἄνδρα μοι ἔννεπε, μοῦσα, πολύτροπον, ὃς μάλα πολλὰ',
 'πλάγχθη, ἐπεὶ Τροίης ἱερὸν πτολίεθρον ἔπερσεν·',
 'πολλῶν δ' ἀνθρώπων ἴδεν ἄστεα καὶ νόον ἔγνω,',
 'πολλὰ δ' ὅ γ' ἐν πόντῳ πάθεν ἄλγεα ὃν κατὰ θυμόν,',
 'ἀρνύμενος ἥν τε ψυχὴν καὶ νόστον ἑταίρων.',
 'ἀλλ' οὐδ' ὣς ἑτάρους ἐρρύσατο, ἱέμενός περ·',
 'αὐτῶν γὰρ σφετέρῃσιν ἀτασθαλίῃσιν ὄλοντο,',
 'νήπιοι, οἳ κατὰ βοῦς Ὑπερίονος Ἠελίοιο',
 'ἤσθιον· αὐτὰρ ὁ τοῖσιν ἀφείλετο νόστιμον ἦμαρ.',
 '']
```

Αν η `split` δεν έχει κάποιο όρισμα, τότε αφαιρεί όλα τα κενά (και τα tabs και τα enters) μεταξύ των λέξεων σε ένα string:

In [61]:
```
"hello                world".split()
```

Out[61]:
```
['hello', 'world']
```

Η μέθοδος `join` κάνει το αντίθετο. Παίρνει μία λίστα από strings και τα εννώνει σε ένα string:

In [62]:
```
'+'.join(['a','b','c'])
```

Out[62]:
```
'a+b+c'
```

In [64]:
```
' '.join(['hello', 'world'])
```

Out[64]:
```
'hello world'
```

In [65]:
```
print ('\n'.join(['line 1', 'line 2']))
```

```
line 1
line 2
```

Οι συναρτήσεις all και any

Η `all` επιστρέφει `True` αν **όλα** τα στοιχεία μιας λίστας είναι `True`

In [49]:
```
all([True, True, True])
```

Out[49]:
```
True
```

In [50]:
```
all([True, False, True])
```

Out[50]:
```
False
```

In [51]:
```
all([3,4,5,4,5])
```

Out[51]:
```
True
```

In [52]:
```
all([3,4,5,'',4,5])
```

Out[52]:
```
False
```

Η `any` μας επιστρέφει `True` αν έστω και ένα στοιχείο της λίστας είναι `True` :

In [53]:
```
any([False, False, False])
```

Out[53]:
```
False
```

In [54]:
```
any([False, False, False, "mitsos"])
```

Out[54]:
```
True
```

Προσοχή Η άδεια λίστα έχει `all` `True` και `any` `False` :

In [55]:
```
all([])
```

Out[55]:
```
True
```

In [56]:
```
any([])
```

Out[56]:
```
False
```

In [ ]:
```

```

Η συνάρτηση range

Η συνάρτηση `range` δημιουργεί κάτι* που αναπαραστάει μία αριθμητική ακολουθία.

* Αυτό το "κάτι" ονομάζεται generator και θα το μάθουμε στην επόμενη διάλεξη

In [22]:
```
range(1,10)
```

Out[22]:
```
range(1, 10)
```

Το σημαντικό είναι ότι αν του εφαρμόσουμε τη `list` τότε μας επιστρέφει μία λίστα:

In [25]:
```
list(range(10)) # Από το 0 έως το 10 (χωρίς το 10)
```

Out[25]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [26]:
```
list(range(5,10)) # Από το 5 έως το 10 (χωρίς το 10)
```

Out[26]:
```
[5, 6, 7, 8, 9]
```

In [27]:
```
list(range(1,11,2)) # Από το 1 έως το 11 (χωρίς το 11) με βήμα 2
```

Out[27]:
```
[1, 3, 5, 7, 9]
```

Η αριθμητική πρόοδος μπορεί να είναι και "ανάποδη"

In [28]:
```
list(range(11,1,-2))
```

Out[28]:
```
[11, 9, 7, 5, 3]
```

In [29]:
```
list(range(10,1,-1))
```

Out[29]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

Η `list(range(...))` επιστρέφει μία λίστα που μπορούμε να κάνουμε πράξεις κανονικά όπως έχουμε μάθει:

In [31]:
```
a = list(range(100, 120, 5)) + ["mitsos"]
print (a)
```

```
[100, 105, 110, 115, 'mitsos']
```

Γιατί στη python όποτε βλέπουμε το `"XYZ"[a:b]` , `[1,2,3][a:b]` , `range(a,b)` αυτό σημαίνει από το `a` μέχρι το `b` **ΧΩΡΙΣ ΤΟ `b`** ; Αυτή η ιστορία ξεκινάει από πολύυυυυ παλιά. Το βασικό είναι ότι όταν λέμε σε έναν υπολογιστή "θέλω Ν" στοιχεία, τότε με βάση μιας παλιάς σύμβασης, το πρώτο στοιχείο έχει δείκτη (index) 0, το δεύτερο 1, κτλ. Οπότε όταν λέμε `range(10)` ο υπολογιστής φτιάχνει τη λίστα από το 0 μέχρι και το 9. Όταν λέμε range(5,7) τότε στην ουσία ζητάμε από τον υπολογιστή μία λίστα με 2 στοιχεία (7-5). Το πρώτο σύμφωνα με την ίδια σύμβαση θα είναι "από εκεί που ξεκινάει η αρίθμηση" δηλαδή το 5. Αφού η λίστα πρέπει να έχει 2 στοιχεία τότε το δεύτερο θα είναι το επόμενο δηλαδή το 6. Αυτή η αρίθμηση μας "βολεύει" και για κάποιους μαθηματικούς υπολογισμούς. Κάποια επιπλέον παραδείγματα:

In [32]:
```
list(range(3,10,2))
```

Out[32]:
```
[3, 5, 7, 9]
```

In [33]:
```
list(range(3,11,2))
```

Out[33]: [3, 5, 7, 9]

In [34]:
```
list(range(3,12,2))
```

Out[34]: [3, 5, 7, 9, 11]

In [35]:
```
list(range(10)) # list(range(0,10))
```

Out[35]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [36]:
```
list(range(5,7)) # list(range(a,b)) # b-a
```

Out[36]: [5, 6]

Η μέθοδος `zip`

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

In [70]:
```
list(zip([1,2,3], ['a', 'b', 'c']))
```

Out[70]: [(1, 'a'), (2, 'b'), (3, 'c')]

Η μέθοδος `enumerate`

Η μέθοδος `enumerate` παίρνει μία λίστα και δημιουργεί μία άλλη λίστα η οποία περιέχει και τις θέσεις (indexes) και τα στοιχεία της πρώτης λίστας:

In [109…
```
a = ["python", "mitsos", "Crete"]
print (list(enumerate(a)))
```

```
[(0, 'python'), (1, 'mitsos'), (2, 'Crete')]
```

Μπορείτε να κάνετε ότι και η `enumerate` αν συνδοιάσετε τη `range` με τη `zip` :

In [110…
```
a = ["python", "mitsos", "Crete"]
print (list(zip(range(len(a)),a)))
```

```
[(0, 'python'), (1, 'mitsos'), (2, 'Crete')]
```

In [ ]:
```

```

Η εντολή `for`

Με τον εντολή `for` μπορούμε να επαναλάβουμε εντολές για κάθε στοιχεία μιας λίστας

In [37]:
```
for x in [1,4,6]:
    print (x)
```

```
1
4
6
```

In [38]:
```
for x in [1,4,6]:
    print ("The number is:", x)
```

```
The number is: 1
The number is: 4
The number is: 6
```

In [39]:
```
for x in range(1,10):
    print ("The number is:", x)
```

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
```

In [40]:
```
for i in range(1,10,3):
    print ("Hello:", i)
```

```
Hello: 1
Hello: 4
Hello: 7
```

Αν οι εντολές που θέλουμε να επαναλάβουμε είναι πάνω από 1 τότε είναι **ΥΠΟΧΡΕΩΤΙΚΟ** να τις βάλουμε στην επόμενη γραμμή και πιο μέσα. Αυτό ονομάζεται υποχρεωτικό indentation ή αλλιώς κανόνας off-side!

In [41]:
```
for i in range(1,10,3):
    print ("command A:", i)
    print ("command B:", i)
```

```
command A: 1
command B: 1
command A: 4
command B: 4
command A: 7
command B: 7
```

Αν υπάρχει `for` μέσα στη `for` τότε πρέπει οι επόμενες γραμμές να μπούν ακόμα πιο μέσα:

In [42]:
```
for i in range(1,5):
    for j in range(1,5):
        print (i,j)
```

```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
```

Παράδειγμα: Ο πίνακας πολλαπλασιασμού:

In [43]:
```
for i in range(1,11):
    for j in range(1,11):
        print ("{} X {} = {}".format(i,j,i*j))
    print ("=" * 10)
```

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
1 X 6 = 6
1 X 7 = 7
1 X 8 = 8
1 X 9 = 9
1 X 10 = 10
==========
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
==========
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
==========
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
==========
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
==========
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
```

```
6 X 9 = 54
6 X 10 = 60
==========
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
==========
8 X 1 = 8
8 X 2 = 16
8 X 3 = 24
8 X 4 = 32
8 X 5 = 40
8 X 6 = 48
8 X 7 = 56
8 X 8 = 64
8 X 9 = 72
8 X 10 = 80
==========
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90
==========
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
==========
```

Επίσης μπορούμε να κάνουμε επανάληψη χρησιμοποιώντας string αντί για λίστα:

In [44]:
```
for letter in "python":
    print (letter)
```

```
p
y
t
h
o
n
```

Αν μία λίστα έχει υπο-λίστες με παραπάνω από 1 στοιχεία τότε μπορούμε να χρησιμοποιήσουμε παραπάνω απο 1 μεταβλητές στη `for` :

In [45]:
```
a = [[2, "Crete"], [3, "Cyprus"], [4, "Majiorca"]]
for x, y in a:
    print ("Number: {} Island: {}".format(x,y))
```

```
Number: 2 Island: Crete
Number: 3 Island: Cyprus
Number: 4 Island: Majiorca
```

Φυσικά το ίδιο μπορεί να γίνει αν έχει υπο-λίστες με 3 στοιχεία κτλ..

In [46]:
```
a = [[1,2,3], ["a", "b", "c"]]
for x,y,z in a:
    print ("{} {} {}".format(x,y,z))
```

```
1 2 3
a b c
```

Η `if` μπορεί να είναι μέσα σε μία `for` :

In [47]:
```
for i in range(1,10):
    if i>5:
        print (i)
```

```
6
7
8
9
```

In [ ]:
```
for i in range(1,10):
    if i>=5:
        print (i)
```

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 [67]:
```
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 [68]:
```
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 [69]:
```
for i in range(1,10):
    if i == 5:
        continue
        print (i) #  TO PRINT EINAI MESA STHN IF!
```

In [ ]:

In [ ]:

List Comprehensions

Τα List Comprehensions είναι ένας μηχανισμός για να δημιουργήσουμε μία λίστα από κάποια άλλη λίστα. Η επίσημη περιγραφή βρίσκεται εδώ. Η γενικότερη μορφή είναι:

```
a = [ΕΚΦΡΑΣΗ for ΜΕΤΑΒΛΗΤΗ in LIST]
```

Το οποίο είναι ισοδύναμο με:

```
a = []
for ΜΕΤΑΒΛΗΤΗ in LIST:
    a.append(ΕΚΦΡΑΣΗ)
```

Παράδειγμα:

In [1]:
```
a = [1,2,3]
```

In [2]:
```
b = [x+1 for x in a]
print (b)
```

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

Το παραπάνω είναι ισοδύναμο με:

In [3]:
```
a = [1,2,3]
b = []
for x in a:
    b.append(x+1)
print (b)
```

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

Κάποια άλλα παραδείγματα:

In [5]:
```
a = ["a", "b", "c"]
["hello: " + x for x in a]
```

Out[5]:
```
['hello: a', 'hello: b', 'hello: c']
```

In [6]:
```
a = ["a", "b", "c"]
b  = [x * 3 for x in a]
print (b)
```

```
['aaa', 'bbb', 'ccc']
```

In [8]:
```
a = [1,2,3,4,5,6]
[x/2 for x in a]
```

Out[8]:
```
[0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
```

Επίσης μπορούμε να χρησημοποιήσουμε την `if` στα list comprehensions. Η γενικότερη μορφή είναι:

```
a = [ΕΚΦΡΑΣΗ for ΜΕΤΑΒΛΗΤΗ in LIST if ΛΟΓΙΚΗ_ΕΚΦΡΑΣΗ]
```

Το οποίο είναι ισοδύναμε με:

```
a = []
for ΜΕΤΑΒΛΗΤΗ in LIST:
    if ΛΟΓΙΚΗ_ΕΚΦΡΑΣΗ:
        a.append(ΕΚΦΡΑΣΗ)
```

Παραδείγματα:

In [10]:
```
a = [1,2,3,4,5,6]
[x/2 for x in a if x>4]
```

Out[10]:
```
[2.5, 3.0]
```

Αυτό είναι ισοδύναμο με:

In [12]:
```
a = [1,2,3,4,5,6]
b = []
for x in a:
    if x>4:
        b.append(x/2)
print (b)
```

```
[2.5, 3.0]
```

Άλλο ένα παράδειγμα. Έστω η λίστα:

In [13]:
```
a = [1,2,3,4,5,4,3,5,6,7,8,7,6,5,5,4]
```

Ποιές είναι όλες οι θέσεις που έχουν τι τιμή 4;

Πρώτος τρόπος:

In [14]:
```
[i for i, x in enumerate(a) if x==4]
```

Out[14]:
```
[3, 5, 15]
```

Δεύτερος τρόπος:

In [16]:
```
a = [1,2,3,4,5,4,3,5,6,7,8,7,6,5,5,4]
[x for x in range(len(a)) if a[x] == 4]
```

Out[16]:
```
[3, 5, 15]
```

In [18]:
```
list(range(len(a)))
```

Out[18]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Επίσης ένα list comprehension μπορεί να έχει παραπάνω από 1 `for` :

In [19]:
```
a = [1,2,3]
b = ["a", "b", "c"]
["{}{}".format(x,y) for x in a for y in b]
```

Out[19]:
```
['1a', '1b', '1c', '2a', '2b', '2c', '3a', '3b', '3c']
```

Αυτό είναι ισοδύναμο με:

In [20]:
```
c = []
for x in a:
    for y in b:
        c.append("{}{}".format(x,y))
print (c)
```

```
['1a', '1b', '1c', '2a', '2b', '2c', '3a', '3b', '3c']
```

Χρησιμοποιώντας list comprehension μπορούμε να παράξουμε ένα string που περιέχει όλη τη προπαίδεια:

In [21]:
```
print ('\n'.join(["{} X {} = {}".format(x,y,x*y) for x in range(1,11) for y in
```

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
1 X 6 = 6
1 X 7 = 7
1 X 8 = 8
1 X 9 = 9
1 X 10 = 10
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
```

```
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
6 X 9 = 54
6 X 10 = 60
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
8 X 1 = 8
8 X 2 = 16
8 X 3 = 24
8 X 4 = 32
8 X 5 = 40
8 X 6 = 48
8 X 7 = 56
8 X 8 = 64
8 X 9 = 72
8 X 10 = 80
9 X 1 = 9
9 X 2 = 18
9 X 3 = 27
9 X 4 = 36
9 X 5 = 45
9 X 6 = 54
9 X 7 = 63
9 X 8 = 72
9 X 9 = 81
9 X 10 = 90
10 X 1 = 10
10 X 2 = 20
10 X 3 = 30
10 X 4 = 40
10 X 5 = 50
10 X 6 = 60
10 X 7 = 70
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

Μερικά παραδείγματα χρήσης των παραπάνω

1. Από μία λίστα πάρε μόνο αυτά που έχουν μία ιδιότητα

π.χ. Από μία λίστα πάρε μόνο τους μονούς αριθμούς

In [74]:
```
a = [1,2,3,4,5,6,7,8,9,10]
def f(x):
    return x%2==1
```

In [75]:
```
# 1ος τρόπος
b = list(filter(f,a))
print (b)
```

```
[1, 3, 5, 7, 9]
```

In [76]:
```
# 2ος τρόπος
b = []
for x in a:
    if f(x):
        b.append(x)
print (b)
```

```
[1, 3, 5, 7, 9]
```

In [77]:
```
# 3ος τρόπος
b = []
for x in a:
    if not f(x):
        continue

    b.append(x)
print (b)
```

```
[1, 3, 5, 7, 9]
```

In [78]:
```
# 4ος τρόπος
b = [x for x in a if f(x)]
print (b)
```

```
[1, 3, 5, 7, 9]
```

2. Από μία λίστα μέτρα αυτά που έχουν μία ιδιότητα

Πόσοι είναι οι μονοί αριθμοί;

In [83]:
```
# 1ος τρόπος
c = len(list(filter(f,a)))
print(c)
```

```
5
```

In [84]:
```
# 2ος τρόπος
c = sum(map(f,a))
print (c)
```

```
5
```

In [85]:
```
# 3ος τρόπος
c = 0
for x in a:
    if f(x):
        c += 1
print (c)
```

```
5
```

In [86]:
```
# 4ος τρόπος
c = 0
for x in a:
    if not f(x):
        continue
    c += 1
print (c)
```

```
5
```

In [89]:
```
# 5ος τρόπος
c = len([None for x in a if f(x)])
print (c)
```

```
5
```

In [90]:
```
# 6ος τρόπος
c = sum(f(x) for x in a)
print (c)
```

```
5
```

3. Εφάρμοσε μία συνάρτηση σε όλα τα στοιχεία μίας λίστας

π.χ δεκαπλασίασε όλα τα στοιχεια

In [91]:
```
a = [1,2,3,4,5,6,7,8,9,10]
def f(x):
    return x*10
```

In [92]:
```
# 1ος τρόπος
b = list(map(f,a))
print (b)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

In [93]:
```
# 2ος τρόπος
b = []
for x in a:
    b.append(f(x))
print (b)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

In [94]:
```
# 3ος τρόπος
b = [f(x) for x in a]
print (b)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

4. Ένας συνδυασμός όλων των παραπάνω

Βρες το άθροισμα του γινόμενου επί 10 των μονών αριθμών μίας λίστας.

In [95]:
```
a = [1,2,3,4,5,6,7,8,9,10]
def f(x):
    return x%2==1
def g(x):
    return x*10
```

In [97]:
```
# 1ος τρόπος
c = sum(map(g, filter(f, a)))
print (c)
```

```
250
```

In [98]:
```
# 2ος τρόπος
c = 0
for x in a:
    if f(x):
        c += g(x)
print (c)
```

```
250
```

In [99]:
```
# 3ος τρόπος
c = 0
for x in a:
    if not f(x):
        continue
    c += g(x)
print (c)
```

```
250
```

In [102…
```
# 4ος τρόπος
c = sum(g(x) for x in a if f(x))
print (c)
```

```
250
```

5. Συνδοιάζοντας πάνω από μία λίστες

Έχουμε δύο λίστες με το ίδιο μέγεθος α,β. Σε κάθε θέση των ληστών έχουμε από μία μετρήση. Η κάθε μέτρηση έχει 2 "τιμές". Για παράδειγμα:

Οι πόλεις:

In [103…
```
cities = ['Heraklion', 'Athens', 'Thessaloniki']
```

και οι **αντίστοιχοι** πληθυσμοί τους:

In [104…
```
pop = [200_000, 4_000_000, 500_000]
```

Βρες τις πόλεις που έχουν πληθυσμό κάτω από 1.000.000:

In [105…
```
# 1ος τρόπος
solution = []
for city, p in zip(cities, pop):
    if p < 1_000_000:
        solution.append(city)
print (solution)
```
['Heraklion', 'Thessaloniki']

In [106…
```
# 2ος τρόπος
solution = []
for i, p in enumerate(pop):
    if p < 1_000_000:
        solution.append(cities[i])
print (solution)
```
['Heraklion', 'Thessaloniki']

In [107…
```
# 3ος τρόπος
solution = [city for city, p in zip(cities, pop) if p<1_000_000]
print (solution)
```
['Heraklion', 'Thessaloniki']

In [108…
```
# 4ος τρόπος
solution = [cities[i] for i,p in enumerate(pop) if p<1_000_000]
print (solution)
```
['Heraklion', 'Thessaloniki']

In [114…
```
# 5ος τρόπος (πολύ άσχημος!)
def f(x):
    return x[1]<1_000_000

def g(x):
    return x[0]

def h(x):
    return cities[x]

solution = list(map(h, map(g, filter(f, enumerate(pop)))))
print (solution)
```
['Heraklion', 'Thessaloniki']

6. Κάνε μία λίστα με λίστες, μία επίπεδη (flat) λίστα

Έστω η λίστα:

In [115…
```
a = [ [1,2], ["a", "b"], [True, False] ]
```

Φτιάξε μία λίστα που θα έχει όλα τα στοιχεία της a σε ένα επίπεδο (χωρίς υπολίστες)

In [118…
```
# 1ος τρόπος
b = []
for x in a:
    b.extend(x)
print (b)
```
```
[1, 2, 'a', 'b', True, False]
```

In [119…
```
# 2ος τρόπος
b = []
for x in a:
    for y in x:
        b.append(y)
print (b)
```
```
[1, 2, 'a', 'b', True, False]
```

In [117…
```
# 2ος τρόπος (hot!)
b = [y for x in a for y in x]
print (b)
```
```
[1, 2, 'a', 'b', True, False]
```

7. Ξεζιπάρισμα!

Έστω η λίστα:

In [120…
```
a = [ [1, "a"], [2, "b"], [3, "c"]]
```

Φτιάξτε δύο λίστες k,l οι ποίες θα έχουν τα πρώτα και δεύτερα στοιχεία των υπολοιστών της a αντίστοιχα!

In [121…
```
# 1ος τρόπος
k = []
l = []
for x in a:
    k.append(x[0])
    l.append(x[1])
print (k)
print (l)
```
```
[1, 2, 3]
['a', 'b', 'c']
```

In [122…
```
# 2ος τρόπος:
k = []
l = []
for x,y in a:
    k.append(x)
    l.append(y)
print (k)
print (l)
```
```
[1, 2, 3]
['a', 'b', 'c']
```

In [123…
```
# 3ος τρόπος
k = [x[0] for x in a]
l = [x[1] for x in a]
print (k)
print (l)
```

```
[1, 2, 3]
['a', 'b', 'c']
```

8. Έλεγχος αν υπάρχει ένα στοιχείο σε μία λίστα.

υπάρχει το 3 στη λίστα `a = [1,2,3,4,5,6,7,8,9,10]` ;

In [124…
```
a = [1,2,3,4,5,6,7,8,9,10]
```

In [125…
```
# 1ος τρόπος
b = 3 in a
print (b)
```

```
True
```

In [126…
```
# 2ος τρόπος
b = False
for x in a:
    if x==3:
        b = True
        break
print (b)
```

```
True
```

In [127…
```
# 3ος τρόπος
b = True
for x in a:
    if x==3:
        break
else:
    b = False
print (b)
```

```
True
```

In [ ]: