Λίστες

Οι λίστες είναι ένα μία βασική έννοια της επιστήμης υπολογιστών.

Στην ουσία είναι μια διατεταγμένη σειρά από δεδομένα. Διατεταγμένο = κάθε στοιχείο έχει τη θέση του (1η, 2η, ...)

In [13]:
```
a = [1,2,3,4]
print (a)
```

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

Μία λίστα μπορεί να έχει στοιχεία διαφορετικού τύπου (αριθμοί, δεκαδικά, strings, ...)

In [14]:
```
a = [1,2,3,"mitsos", 5, 7.77777778]
print (a)
```

```
[1, 2, 3, 'mitsos', 5, 7.77777778]
```

Η προσπέλαση των στοιχείων μίας λίστας γίνεται ακριβώς όπως και με τα strings:

In [15]:
```
a[0] # Το πρώτο στοιχείο
```

Out[15]:
```
1
```

In [16]:
```
a[0:3] #  Όλα τα στοιχεία από το πρώτο μέχρι το τέταρτο (χωρίς το τέταρτο)
```

Out[16]:
```
[1, 2, 3]
```

In [17]:
```
a[-1] # Το τελευταίο στοιχείο
```

Out[17]:
```
7.77777778
```

In [18]:
```
a[-2:] # Το προτελευταίο στοιχείο
```

Out[18]:
```
[5, 7.77777778]
```

Ομοίως, μπορούμε να χρησιμοποιήσουμε τα διαστήματα. Έστω:

In [19]:
```
b = [1,2,3,4,5,6]
```

In [20]:
```
b[2:5:2] # Από το 3ο μέχρι το 6ο (χωρίς να πάρουμε ΚΑΙ το 6ο), με βήμα 2
```

Out[20]:
```
[3, 5]
```

In [21]:
```
b[::2] # Από την αρχή μέχρι ΚΑΙ το τέλος με βήμα 2
```

Out[21]:
```
[1, 3, 5]
```

In [22]:
```
b[:3] # Από την αρχή μέχρι το 4ο στοιχείο (χωρίς να πάρουμε ΚΑΙ το 4ο)
```

Out[22]:
```
[1, 2, 3]
```

In [23]:
```
# Όλα τα παρακάτω είναι ισοδύναμα
print (b)
print (b[:])
print (b[::])
print (b[::1])
print (b[0:])
print (b[0::])
print (b[0::1])
print (b[:len(b)])
print (b[:len(b):])
print (b[0:len(b)])
print (b[0:len(b):1])
```

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

In [24]:
```
b[-1:0:-1] # Από το τέλος μέχρι την αρχή (χωρίς να πάρουμε ΚΑΙ την αρχή)
```

Out[24]:
```
[6, 5, 4, 3, 2]
```

In [25]:
```
b[-1::-1] # Από το τέλος μέχρι την αρχή (παίρνουμε και την αρχή)
```

Out[25]:
```
[6, 5, 4, 3, 2, 1]
```

In [26]:
```
b[::-1] # Αυτό είναι ισοδύναμε με το παραπάνω
```

Out[26]:
```
[6, 5, 4, 3, 2, 1]
```

Όπως και τα strings έτσι και στις λίστες μπορούμε να εφαρμόσουμε τις `len` , `count` , `index` .

In [27]:
```
a = [1,2,3,"mitsos", 5, 7.77777778]
```

In [28]:
```
len(a) # Το πλήθος όλων των στοιχείων της λίστας
```

Out[28]:
```
6
```

In [29]:
```
a.count(1) # Πόσες φορές υπάρχει το 1 μέσα στη λίστα;
```

Out[29]:
```
1
```

In [30]:
```
a.count(55) # Πόσες φορές υπάρχει το 55 μέσα στη λίστα;
```

Out[30]:
```
0
```

In [31]:
```
a.index("mitsos") # Σε ποια θέση της λίστας εμφανίζεται το "mitsos"?
```

Out[31]: 3

In [32]:
```
a.index(4) # Σε ποια θέση της λίστας εμφανίζεται το 4;
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-e644608c73eb> in <module>
----> 1 a.index(4) # Σε ποια θέση της λίστας εμφανίζεται το 4;

ValueError: 4 is not in list
```

Μία λίστα μπορεί να έχει μέσα άλλες λίστες!

In [130…
```
a = [1,2, [3,4,5], 6, 7]
```

In [34]:
```
len(a)
```

Out[34]: 5

In [35]:
```
a[2]
```

Out[35]: [3, 4, 5]

In [36]:
```
a[1]
```

Out[36]: 2

In [131…
```
a[2][1]
```

Out[131… 4

Μπορούμε να αλλάξουμε τα περιεχόμενα οποιουδήποτε στοιχείου μίας λίστας:

In [132…
```
a = [1,2,3,4,5]
a[2] = 8
print (a)
```

```
[1, 2, 8, 4, 5]
```

In [133…
```
a[3] = ['Mitsos', 'Kwstas']
print (a)
```

```
[1, 2, 8, ['Mitsos', 'Kwstas'], 5]
```

Προσοχή! αυτό δεν επιτρέπεται στα strings:

In [134…
```
a = 'Mitsos'
a[2] = 'k'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-134-2320b677b6ef> in <module>
      1 a = 'Mitsos'
----> 2 a[2] = 'k'

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

Αν θέλετε να διαβάσετετε περισσότερα σχετικά με το γιατί ισχύει αυτό κάντε google: 'Why

are Python Strings Immutable?'

Υπάρχει επίσης η άδεια λίστα: []

In [38]:
```
a = []
print (len(a))
```

```
0
```

Μπορούμε να γράψουμε τις λίστες με πολλούς τρόπους:

In [39]:
```
# Τα παρακάτω είναι ισοδύναμα:
a = [1,2,3]

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

Προσοχή! Δεν υπάρχει πρόβλημα αν βάλουμε ένα κόμμα στο τέλος μίας λίστας!

In [40]:
```
# Αυτά τα δύο είναι ισοδύναμα:
print ([1,2,3])
print ([1,2,3,])
```

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

Οπότε αν βάλουμε ένα κόμμα στο τέλος δεν υπάρχει πρόβλημα. Υπάρχει όμως αν ΔΕΝ βάλουμε στη μέση:

In [41]:
```
a = [
    'aaaa',
    'bbbb' # PROSOXH! auto einai 1 string (me to apo katw)
    'cccc'
]
print (len(a))
```

```
2
```

Μία λίστα μπορεί να έχει μία λίστα, που έχει μία λίστα που έχει..

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

```
1
```

In [43]:
```
a = [[[[[[[[[[[[[[[[[[]]]]]]]]]]]]]]]]]
print (len(a))
```

```
1
```

In [44]:
```
a = [1,2,3,[],4]
```

In [45]:
```
len(a)
```

Out[45]:
```
5
```

In [46]:
```
a=3
b = [a,a,a+1, a/2]
print (b)
```

```
[3, 3, 4, 1.5]
```

Μπορούμε να προσθέσουμε δύο λίστες:

In [48]:
```
[1,2,3] + ["mitsos", "a"]
```

Out[48]:
```
[1, 2, 3, 'mitsos', 'a']
```

Μπορούμε να πολλαπλασιάσουμε μία λίστα με έναν αριθμό:

In [49]:
```
[1,2,3] *4
```

Out[49]:
```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Δεν μπορούμε να πολλαπλασιάσουμε ή να αφαιρέσουμε δύο λίστες!

In [50]:
```
[1,2,3] * [5,6]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-4f81883a1dc4> in <module>
----> 1 [1,2,3] * [5,6]

TypeError: can't multiply sequence by non-int of type 'list'
```

In [51]:
```
[1,2,3] - ["mitsos", "a"]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-d5954fea0119> in <module>
----> 1 [1,2,3] - ["mitsos", "a"]

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Η Μέθοδος `list` κάνει ότι μπορεί να μετατρέψει κάτι σε λίστα:

In [52]:
```
list("mitsos")
```

Out[52]:
```
['m', 'i', 't', 's', 'o', 's']
```

In [53]:
```
list([1,2,3]) # Δεν κάνει τίποτα
```

Out[53]:
```
[1, 2, 3]
```

Δεν μπορούν να μετατραπούν τα πάντα σε λίστα:

In [54]:
```
list(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-0c7f5cd48ec1> in <module>
----> 1 list(5)

TypeError: 'int' object is not iterable
```

Μία λίστα είναι πάντα True, εκτώς αν είναι άδεια:

In [57]:
```
if [1,2,3]:
    print ("not empty")
```

```
not empty
```

In [56]:
```
if []:
    print ("not empty")
else:
    print ("is empty!")
```

```
is empty!
```

Σύγκριση strings

Όταν εφαρμόζονται σε strings, τότε τα συγκρίνουμε αλφαριθμητικά (λεκτικά). Ποιο μικρό θεωρείται αυτό που σε μία ταξινόμηση, παίρνει τη μικρότερη θέση:

In [1]:
```
'ab' < 'fg'
```

Out[1]: True

In [2]:
```
'ab' < 'b'
```

Out[2]: True

In [3]:
```
'ab' < 'ac'
```

Out[3]: True

In [4]:
```
'ab' < 'a'
```

Out[4]: False

Το άδειο string έχει τη πιο μικρή δυνατή τιμή

In [5]:
```
'' < '0'
```

Out[5]: True

In [6]:
```
"A" < "a"
```

Out[6]: True

In [7]:
```
"05456745674" < "5"
```

Out[7]: True

In [8]:
```
'8' < '09'
```

Out[8]: False

Ο τελεστής in

Αυτός ο τελεστής ελέγχει αν υπάρχει "κάτι" "κάπου"

In [9]:
```
'rak' in 'Heraklion'
```

Out[9]: True

In [10]:
```
'raki' in 'Heraklion'
```

Out[10]: False

In [11]:
```
'h' in 'Heraklion'
```

Out[11]: False

In [12]:
```
'H' in 'Heraklion'
```

Out[12]: True

In [58]:
```
1 in [1,2,3]
```

Out[58]: True

In [59]:
```
[1,2] in [1,2,3]
```

Out[59]: False

In [60]:
```
[1,2] in [1, [1,2], 3]
```

Out[60]: True

In [61]:
```
False in [1, True-True]
```

Out[61]: True

In [63]:
```
None in [3, None, 4]
```

Out[63]: True

In [65]:
```
'ra' in ['Heraklion']
```

Out[65]: False

In [94]:
```
[] in [1, [], 2]
```

Out[94]: True

map και filter

Μπορούμε να εφαρμόσουμε μία συνάρτηση σε όλα τα στοιχεία μία λίστας με τη συνάρτηση `map` :

In [97]:
```
def f(x):
    return x+1

a = [4,5,6]

list(map(f,a))
```

Out[97]: [5, 6, 7]

Μπορούμε να πάρουμε ένα υποσύνολο των στοιχείων μία λίστας τα οποία έχουν μία ιδιότητα με τη συνάρτηση `filter`. Η filter πρέπει να επιστρέφει κάτι που μπορεί να αποτιμηθεί ως `True` ή `False`.

In [100…
```
def is_even(x):
    # Επέστρεψε True / False ανάλογα αν το χ είναι άρτιο ή όχι.
    return x%2==0

a = [1,2,3,4,5,6,7,8,9]
list(filter(is_even,a))
```

Out[100… [2, 4, 6, 8]

In [101…
```
def is_first_vowel(x):
    # Επέστρεψε True / False ανάλογα αν το χ ξεκινάει (πρώτο γράμμα) από φωνήε
    return x[0].lower() in 'αειουω'

a = ['Ηράκλειο', 'Θεσσαλονίκη', 'Αθήνα']
list(filter(is_first_vowel,a))
```

Out[101… ['Ηράκλειο', 'Αθήνα']

Πράξεις πάνω σε λίστες

Σε λίστες μπορούμε να κάνουμε τις παρακάτω πράξεις:

In [67]:
```
sum([2,3,4]) # Το άθροισμα όλων των στοιχείων της λίστας:
```

Out[67]: 9

Προσοχή Η `sum` πρέπει να έχει μόνο `int` ή `float`

In [68]:
```
sum(['a', 'b'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-0ca1e4efb8fe> in <module>
----> 1 sum(['a', 'b'])

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [69]:
```
min([3,5,4]) # Το μικρότερο στοιχείο
```

Out[69]: 3

In [70]:
```
max(['heraklion', 'patras', 'athens']) # Το μεγαλύτερο στοιχείο
```

Out[70]: 'patras'

In [71]:
```
max(['heraklion', 'patras', 't'])
```

Out[71]: 't'

Μία ιδιότητα που έχουν οι `max` και η `min` είναι ότι αν η λίστα περιέχει υπολίστες τότε ψάχνουν το μικρότερο στοιχείο στο πρώτο στοιχείο της υπολίστας. Αν υπάρχουν δύο ή περισσότερες υπολίστες με το ίδιο μικρότερο πρώτο στοιχείο τότε ψάχνουν στο δεύτερο κτλ.:

In [72]:
```
min([[5, "b"], [7, "t"], [6, 'r'], [5, 'a']])
```

Out[72]: [5, 'a']

Και αυτό γιατί:

In [74]:
```
[5, 'a'] < [5, 'b']
```

Out[74]: True

Πρόσθεση και αφαίρεση στοιχείων σε μία λίστα

Θυμόμαστε ότι μπορούμε να προσθέσουμε δύο λίστες μεταξύ τους:

In [103…
```
[1,2,3] + ['Μίτσος', 7.8]
```

Out[103… [1, 2, 3, 'Μίτσος', 7.8]

Μπορούμε να χρησιμοποιήσουμε αυτή την ιδιότητα για να προσθέτουμε στοιχεία σε μία λίστα σε οποιοδήποτε σημείο:

In [109…
```
# Πρόσθεση στο τέλος:
a = [1,2,3]
a = a + ['Μήτσος']
print (a)
```

[1, 2, 3, 'Μήτσος']

Θυμόμαστε ότι το `a = a + b` είναι ισοδύναμο με το `a += b` :

In [111…
```
# Πρόσθεση στο τέλος:
a = [1,2,3]
a += ['Μήτσος']
print (a)
```

[1, 2, 3, 'Μήτσος']

In [106…
```
# Πρόσθεση στην αρχή:
a = [1,2,3]
a = ['Μήτσος'] + a
print (a)
```

['Μήτσος', 1, 2, 3]

In [107…
```
# Πρόσθεση σε ένα οποιαδήποτε σημείο (χρησιμοποιούμε slicing):
a = [1,2,3]
a = a[:2] + ['Μήτσος'] + a[2:]
print (a)
```

```
[1, 2, 'Μήτσος', 3]
```

Μπορούμε αντί για αυτά να χρησιμοποιήσουμε τις συναρτήσεις append, extend και insert:

In [112…
```
a = [1,2,3]
a.append('Mitsos') # ισοδύναμο με το a += ['Mitsos']
print (a)
```

```
[1, 2, 3, 'Mitsos']
```

In [114…
```
a = [1,2,3]
a.extend(['Mitsos', 7.8]) # ισοδύναμο με το a += ['Mitsos', 7.8]
print (a)
```

```
[1, 2, 3, 'Mitsos', 7.8]
```

In [116…
```
a = [1,2,3]
a.insert(2, 'Mitsos') # ισοδύναμε με το a = a[:2] + ['Mitsos'] + a[2:]
print (a)
```

```
[1, 2, 'Mitsos', 3]
```

Για να αφαιρέσουμε ένα στοιχείο μπορούμε να χρησιμοποιήσουμε πάλι slicing:

In [119…
```
a = [1, 2, 'Mitsos', 3]
a = a[:2] + a[3:]
print(a)
```

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

Μπορούμε όμως και να χρησιμοποιήσουμε τη `del` :

In [120…
```
a = [1, 2, 'Mitsos', 3]
del a[2]
print (a)
```

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

Ένας άλλος τρόπος είναι απλά να κάνουμε filter:

In [123…
```
a = [1, 2, 'Mitsos', 3]

def remove_mitsos(x):
    return x != 'Mitsos'

a=list(filter(remove_mitsos, a))
print(a)
```

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

Sorting

Με την εντολή `sorted` μπορούμε να ταξινομήσουμε μία λίστα:

In [75]:
```
a = [3,4,5,3,2,1]
```

In [76]:
```
sorted(a)
```

Out[76]:
```
[1, 2, 3, 3, 4, 5]
```

Προσοχή! η `sorted` ΔΕΝ αλλάζει τη λίστα. Αποθηκεύει το αποτέλεσμα σε μία άλλη μεταβλητή:

In [77]:
```
a
```

Out[77]:
```
[3, 4, 5, 3, 2, 1]
```

In [78]:
```
b = sorted(a)
```

In [127…
```
b
```

Out[127…
```
[1, 2, 3, 3, 4, 5]
```

αν θέλουμε να αλλάξει η λίστα μας (sorting in place) χρησιμοποιούμε τη συνάρτηση sort:

In [128…
```
a = [3, 4, 5, 3, 2, 1]
a.sort()
print (a) # Η a άλλαξε!
```

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

Μπορούμε να ταξινομήσουμε μόνο λίστες που έχουν τον ίδιο τύπο δεδομένων:

In [80]:
```
sorted(["b", "a", "c"])
```

Out[80]:
```
['a', 'b', 'c']
```

In [81]:
```
sorted(["b", "a", 100, "c"])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-b245b7eeb5df> in <module>
----> 1 sorted(["b", "a", 100, "c"])

TypeError: '<' not supported between instances of 'int' and 'str'
```

Μπορούμε να ταξινομήσουμε από το μεγαλύτερο προς το μικρότερο:

In [82]:
```
sorted([3,4,5,2,3,4,5,2,1], reverse=True)
```

Out[82]:
```
[5, 5, 4, 4, 3, 3, 2, 2, 1]
```

Όπως και με τη `min` και τη `max`, αν αυτό που ταξινομούμε είναι λίστα από λίστες (ή tuple), τότε ελέγχει πρώτα το πρώτο στοιχείο της υπολίστας. Αν είναι ίσο, τότε ελέγχει το δεύτερο κτλ:

In [83]:
```
a = [
    ["mitsos", 50],
    ['gianni', 40],
    ['gianni', 30]
]
sorted(a)
```

Out[83]:
```
[['gianni', 30], ['gianni', 40], ['mitsos', 50]]
```

Στο παραπάνω παράδειγμα το `['gianni', 30]` είναι μικρότερο από το `['gianni', 40]` :

In [84]:
```
['gianni', 30] < ['gianni', 40]
```

Out[84]:
```
True
```

Πολλές φορές θέλουμε να ταξινομήσουμε μία λίστα που περιέχει υπολίστες αλλά θέλουμε η ταξινόμηση να γίνει όχι με βάση το πρώτο στοιχείο αλλά με βάση μία δική μας συνάρτηση. Π.χ. Έστω η λίστα:

In [85]:
```
a = [["gianni", 30, 20000], ["mitsos", 50, 4000], ["anna", 60, 100000]]
```

Ας υποθέσουμε ότι θέλουμε να ταξινομήσουμε τα στοιχεία της λίστας με βάση το τρίτο στοιχείο τους (20000, 4000, 100000). Προσέχτε ότι αν τρέξουμε τη `sorted` τότε δεν θα μας επιστρέψει αυτό που θέλουμε:

In [86]:
```
sorted(a)
```

Out[86]:
```
[['anna', 60, 100000], ['gianni', 30, 20000], ['mitsos', 50, 4000]]
```

Εμείς θέλουμε το στοιχείο που έχει το 4000 να βγει πρώτο μετά το στοιχείο που έχει το 20000 να βγει δεύτερο και το στοιχείο που έχει το 100000 να βγει τελευταίο.

Σε αυτή τη περίπτωση μπορούμε να φτιάξουμε μία συνάρτηση η οποία όταν παίρνει ως όρισμα κάποιο στοιχείο μιας λίστας να μας επιστρέφει την τιμή μέσω της οποίας θα γίνει η ταξινόμηση:

In [87]:
```
def sort_according_to_this(x):
    return x[2]
```

Αν τώρα βάλω σε αυτή τη συνάρτηση ένα στοιχείο της λίστας θα μου επιστρέψει το τρίτο στοιχείο του, το οποίο είναι και αυτό που θέλω να βασιστεί η ταξινόμηση:

In [88]:
```
sort_according_to_this(a[0])
```

Out[88]:
```
20000
```

In [89]:
```
sort_according_to_this(a[1])
```

Out[89]:
```
4000
```

In [90]:
```
sort_according_to_this(a[2])
```

Out[90]:
```
100000
```

Τώρα μπορώ να περάσω ως όρισμα τη συνάρτηση `sort_according_to_this` στη `sorted` και να ταξινομήσει τη λίστα `a` με βάση το τρίτο στοιχείο του κάθε στοιχείου της:

In [91]:
```
sorted(a, key=sort_according_to_this)
```

Out[91]:
```
[['mitsos', 50, 4000], ['gianni', 30, 20000], ['anna', 60, 100000]]
```

Ένα άλλο παράδειγμα. Έστω η λίστα:

In [124…
```
a = ["heraklion", "patras", "thessaloniki", "athens"]
```

Η παρακάτω εντολή ταξινομεί τη συνάρτηση με βάσει το μήκος των strings:

In [93]:
```
sorted(a, key=len)
```

Out[93]:
```
['patras', 'athens', 'heraklion', 'thessaloniki']
```

Οι συναρτήσεις `max` και `len` επίσης υποστηρίζουν τη key=...:

In [125…
```
min(a,key=len) # H polh me to mikrotero onoma
```

Out[125…
```
'patras'
```

In [126…
```
max(a,key=len) # H polh me to megalutero onoma
```

Out[126…
```
'thessaloniki'
```

In [ ]:
```

```