Προγραμματισμός με τη γλώσσα python

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

Διάλεξη 1η, Παρασκευή 17 Οκτωβρίου 2019

Αντί εισαγωγής

Όλες οι διαλέξεις θα διατίθονται σε μορφή jupyter notebooks (http://jupyter.org/). Το Jupyter είναι ένα περιβάλλον που σου επιτρέπει να γράφεις python και να βλέπεις άμεσα τα αποτελέσματα των εντολών στον browser του υπολογιστή σου. Μπορείτε να σώσετε την ανάλυση σε ένα αρχείο, να το στείλετε mail κτλ.

Ένα jupyter notebook αποτελείται από κελιά. Κάθε κελί μπορεί να περιέχει είτε κώδικα python (επιτρέπονται και άλλες γλώσσες) είτε markdown (https://en.wikipedia.org/wiki/Markdown). Το markdown είναι μια συλλογή από συμβάσεις για να εισάγουμε μορφοποίηση σε ένα αρχείο κειμένου. Π.χ. αν στο markdown γράγουμε μία λέξη ανάμεσα σε 2 αστεράκια (π.χ.: `**Αλέξανδρος**`) τότε αυτή θα εμφανιστεί ως bold (έντονη) δηλαδή έτσι: **Αλέξανδρος**. Πλήρη λίστα με όλες τις markdown συμβάσεις (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet).

Επίσης μπορείτε να φορτώσετε κάποιο notebook που θα σας στείλουν στον browser σας. Ακόμα καλύτερα μπορείτε να αποθηκεύσετε ένα notebook στο Internet δωρεάν! Ως gist (https://gist.github.com/). Θα δούμε πως θα το κάνουμε αυτό στην επόμενη διάλεξη.

print

```
In [104]: print ('kati')
```

```
kati
```

Μπορούμε να χρησιμοποιήσουμε μονά (') ή διπλά αυτάκια (")

```
In [10]: print ("hello")
```

```
hello
```

(θα επανέλθουμε αργότερα)

Σχόλια

Σε οποιαδήποτε γραμμή οτιδήποτε ακολουθεί τον χαρακτήρα `#` θεωρείται σχόλιο και αγνοείται:

```
In [255]: # Αυτό είναι ένα σχόλιο
          print ('Αυτό δεν είναι σχόλιο') # Αυτό όμως είναι!
```

```
Αυτό δεν είναι σχόλιο
```

Μαθηματικές πράξεις:

In [11]:
```
3+2
```

Out[11]: 5

Δεκαδική διαίρεση:

In [12]:
```
3/2
```

Out[12]: 1.5

Ακέραια διαίρεση:

In [13]:
```
3//2
```

Out[13]: 1

Προσοχή!

In [189]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-189-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

Αλφαριθμητικά (ή αλλιώς: strings)

In [14]:
```
"mitsos"
```

Out[14]: 'mitsos'

Μπορούμε να προσθέσουμε δύο strings:

In [15]:
```
'a' + 'b'
```

Out[15]: 'ab'

Μπορούμε να πολλαπλασιάσουμε string με ακέραιο:

In [16]:
```
'a' * 10
```

Out[16]: 'aaaaaaaaaa'

Υπάρχει και το άδειο string

In [238]:
```
''
```

Out[238]: ''

Η `len` επιστρέφει το μέγεθος ενός string

In [228]:
```
len("abcdefg")
```

Out[228]: 7

In [239]:
```
len('')
```

Out[239]: 0

Η `count` μας επιστρέφει πόσες φορές υπάρχει ένα string μέσα σε ένα άλλο string.

In [230]:
```
"zabarakatranemia".count('a')
```

Out[230]: 6

In [231]:
```
"zabarakatranemia".count('ra')
```

Out[231]: 2

In [232]:
```
"zabarakatranemia".count('c')
```

Out[232]: 0

Η `index` μας επιστρέφει σε ποιο σημείο συναντάμε ΠΡΩΤΗ φορά κάποιο string μέσα σε ένα άλλο string.

In [234]:
```
"zabarakatranemia".index('anemia')
```

Out[234]: 10

In [236]:
```
"zabarakatranemia".index('ra') # Το "ra" υπάρχει δύο φορές αλλά επιστρέφει τη θέση της πρώτης.
```

Out[236]: 4

Αν δεν υπάρχει τότε πετάει λάθος!

In [237]:
```
"zabarakatranemia".index('c')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-237-1515cc1d7dbe> in <module>()
----> 1 "zabarakatranemia".index('c')

ValueError: substring not found
```

Προσοχή! Δύο strings το ένα δίπλα στο ένα θεωρούνται ένα!

In [271]:
```
"Hello" "world"
```

Out[271]: 'Helloworld'

Ένα string μπορεί να έχει χαρακτήρες σε οποιαδίποτε γλώσσα!

In [275]:
```
a = "σε οποιαδήποτε γλώσσα ذا كيور (بالإنجليزية: The Cure) هي فرقة روك إنجليزية
" ، تم تكوينها في كرولي، غرب ساسكس عام 1976. واجهت الفرقة عِدة تغيرات؛ مع
print (a)
```

```
σε οποιαδήποτε γλώσσα ذا كيور (بالإنجليزية: The Cure) هي فرقة روك إنجليزية، تم
تكوينها في كرولي، غرب ساسكس عام 1976. واجهت الفرقة عِدة تغيرات؛ مع
```

Ναι, τα emoji (https://unicode.org/emoji/charts/full-emoji-list.html) συμπεριλαμβάνονται:

In [277]:
```
print ("\U0001F621")
```

😡

Μεταβλητές

Όταν γράφουμε `a=3` τότε αποθηκεύουμε στη μνήμη του υπολογιστή την τιμή 3. Μπορούμε όποτε θέλουμε μετά να αναφερθούμε σε αυτή την τιμή χρησμοποιώντας το όνομα της μεταβλητής

In [17]:
```
a=3
```

In [18]:
```
a
```

Out[18]: 3

In [19]:
```
a+a
```

Out[19]: 6

In [20]:
```
a = 'mitsos'
```

In [21]:
```
a
```

Out[21]: 'mitsos'

Οι δεκαδικές μεταβλητές έχουν συγκεκριμένη ακρίβεια:

In [22]:
```
a=5.55545345345234578234659823468237456928374659283746592837465928374659238746592837456
928347659283745692834765298734 56
```

In [23]:
```
a
```

Out[23]: 5.555453453452346

Οι ακέραιες μεταβλητές μπορούν να είναι όσο μεγάλες θέλουμε!

In [24]:
```
a = 29843756938475629384765293874562938745629384765293874659238475692387456923874569238
746592387465923874659283746 5
```

In [25]:
```
a
```

Out[25]: 29843756938475629384765293874562938745629384765293874659238475692387456923874569238746
592387465923874659283746 5

Μπορούμε να κάνουμε διάφορες πράξεις μεταξύ μεταβλητών:

In [27]:
```
a=3
b=4
print (a+b)
```

```
7
```

Τα κενα δεν έχουν σημασία (αρκεί όλες οι γραμμές να ξεκινάνε από το ίδιο κενό).

Όλα τα παρακάτω είναι ισοδύναμα:

In [240]:
```
a=3
a = 3
a                                   =                                   3
```

Στα παρακάτω όμως υπάρχει λάθος!

In [242]:
```
a=3
 a=3 # Αυτό ξεκινάει με ένα κενό πιο μέσα!
```

```
  File "<ipython-input-242-fc87d32c6889>", line 2
    a=3 # Αυτό ξεκινάει με ένα κενό πιο μέσα!
    ^
IndentationError: unexpected indent
```

Με την print μπορούμε να τυπώνουμε τις τιμές πολλών μεταβλητών:

In [28]:
```
a="the answer is"
b=7
```

In [29]:
```
print (a,b)
```

```
the answer is 7
```

In [30]:
```
print ("this answer is", b)
```

```
this answer is 7
```

Αν μέσα σε ένα string βάλουμε το `{}` τότε μπορούμε να βάλουμε μία μεταβλητή σε αυτή τη θέση του string. Για να το κάνουμε αυτό χρησιμοποιούμε τη μέθοδο `format` .

In [31]:
```
c = "answer is {}".format(b)
print (c)
```

```
answer is 7
```

Μπορούμε να βάλουμε πάνω από ένα `{}` σε ένα string:

In [192]:
```
a = 'James'
b = 'Bond'
print ('My name is {}. {} {}.'.format(b, a, b))
```

```
My name is Bond. James Bond.
```

Όλα κεφαλαία:

```
In [32]: "abcde".upper()
```

```
Out[32]: 'ABCDE'
```

Όλα μικρά:

```
In [33]: "ABCDE".lower()
```

```
Out[33]: 'abcde'
```

Αντικατάσταση κάποιου κομματιού του string με ένα άλλο:

```
In [34]: "hello world".replace('l', "QQQ")
```

```
Out[34]: 'heQQQQQQo worQQQd'
```

Indexing

Στα strings (όπως και στις λίστες όπως θα δούμε παρακάτω), μπορούμε να παραπάνω ένα υποσύνολό τους χρησιμοποιώντας το `[]` . Αυτή η δυνατότητα ονομάζεται indexing.

```
In [194]: print ("hello")
```

```
          hello
```

Προσοχή! Η αρίθμηση ξεκινάει από το 0!

```
In [195]: "hello"[0]
```

```
Out[195]: 'h'
```

```
In [196]: "hello"[1]
```

```
Out[196]: 'e'
```

Προσοχή! Η αρίθμηση δεν πρέπει να ξεπεράσει το μέγεθος του string!

```
In [197]: "hello"[100]
```

```
          ---------------------------------------------------------------------------
          IndexError                                Traceback (most recent call last)
          <ipython-input-197-e6ccf1afaf71> in <module>()
          ----> 1 "hello"[100]

          IndexError: string index out of range
```

Το index (ή αλλιώς η "αρίθμηση") μπορεί να πάρει πάρει και αρνητικές τιμές! το `-1` είναι το τελευταίο στοιχείο. Το `-2` το προτελευταίο κτλ..

```
In [199]: "hello"[-1]
```

```
Out[199]: 'o'
```

In [200]:
```
"hello"[-2]
```

Out[200]:
```
'l'
```

In [201]:
```
"hello"[-100]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-201-552ff00ad524> in <module>()
----> 1 "hello"[-100]

IndexError: string index out of range
```

Indexing spaces

Μπορούμε να πάρουμε κάποια ένα υποσύνολο ενώς string με βάση τα διαστήματα που ορίζουμε στο `[]`

In [202]:
```
"hello"[1:3]
```

Out[202]:
```
'el'
```

Όταν γράφουμε `[a:b]` εννοούμε "ξεκίνα από το α-στό στοιχείο (η αρίθμηση ξεκινάει από 0!) και σταμάτο στο β-στό στοιχείο, ΧΩΡΙΣ ΟΜΩΣ ΝΑ ΠΑΡΕΙΣ ΑΥΤΟ!!"

In [204]:
```
"hello"[1:4]
```

Out[204]:
```
'ell'
```

Αν θέλουμε να πάρουμε ένα υποσύνολο που ξεκινάει από την αρχή του string τότε μπορούμε να γράψου είτε `[0:b]` είτε `[:b]`

In [206]:
```
"hello"[0:2]
```

Out[206]:
```
'he'
```

In [207]:
```
"hello"[:2]
```

Out[207]:
```
'he'
```

Αν θέλουμε ένα υποσύνολο που τελειώνει στο τέλος του string τότε μπορούμε να γράψουμε `[a:]`

In [208]:
```
"hello"[2:]
```

Out[208]:
```
'llo'
```

Indexing spaces with steps

Μπορούμε να χρησημοποιήσουμε για indexing το `[a:b:c]` . Αυτό σημαίνει: πήγαινε από το a-στο στο b-στο (χωρίς να πάρεις το b-στο!) με βήμα: c

In [212]:
```
"abcdefgij"[1:7:2]
```

Out[212]:
```
'bdf'
```

```
In [213]: "abcdefgij"[1:7:3]
```

```
Out[213]: 'be'
```

Αν παραλείψουμε το πρώτο στοιχείο τότε από default βάζει το 0

```
In [215]: "abcdefgij"[:7:3]
```

```
Out[215]: 'adg'
```

Αν παραλείψουμε το δεύτερο τότε από default βάζει το τέλος του string

```
In [216]: "abcdefgij"[1::3]
```

```
Out[216]: 'bei'
```

Μπορούμε να παραλείψουμε και τα δύο οπότε θα πάρει από την αρχή μέχρι το τέλος του string

```
In [217]: "abcdefgij"[::3]
```

```
Out[217]: 'adg'
```

Αν παραλείψουμε το τρίτο τότε από default βάζει το 1

```
In [218]: "abcdefgij"[1:7:]
```

```
Out[218]: 'bcdefg'
```

Το c δεν μπορεί να είναι 0!

```
In [219]: "abcdefgij"[1:7:0]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-219-96e6dd4da4bc> in <module>()
----> 1 "abcdefgij"[1:7:0]

ValueError: slice step cannot be zero
```

Αρνητικά indexing steps.

Το βήμα `c` μπορεί να είναι αρνητικό!

```
In [220]: "abcdefgij"[7:1:-1]
```

```
Out[220]: 'igfedc'
```

```
In [221]: "abcdefgij"[7:1:-2]
```

```
Out[221]: 'ifd'
```

```
In [222]: "abcdefgij"[7::-2]
```

```
Out[222]: 'ifdb'
```

```
In [223]: "abcdefgij"[::-2]
Out[223]: 'jgeca'

In [224]: "abcdefgij"[::-1] # Reverse a string!
Out[224]: 'jigfedcba'
```

Χρήσιμο όταν έχουμε cDNA !

```
In [225]: "ACGT"[::-1]
Out[225]: 'TGCA'
```

Φυσικά μπορεί να μπει και κάποια μεταβλητή σε αυτά.

```
In [45]: a=3
         "abcde"[0:a]
Out[45]: 'abc'
```

Special Characters

Έχουμε πει ότι με τα μονά ή διπλά "αυτάκια" μπορούμε να δηλώσουμε ένα string. Τι γίνεται όμως όταν θέλουμε να βάλουμε μέσα ένα string ένα μονό ή διπλό αυτάκι; Μπορούμε να χρησιμοποιήσουμε το `\` ή αλλιώς backslash:

```
In [243]: print("mitsos")
          mitsos

In [244]: print("My name is \"mitsos\"")
          My name is "mitsos"

In [245]: print('My name is "mitsos"')
          My name is "mitsos"

In [248]: print ('My name is \'Mitsos\'')
          My name is 'Mitsos'

In [249]: print ("My name is 'Mitsos'")
          My name is 'Mitsos'
```

Υπάρχουν επίσης και οι παρακάτω ειδικοί χαρακτήρες:

- Νέα γραμμή: `\n` (n = New line)
- Tab: `\t`

```
In [250]: print("Line 1\nLine 2")
          Line 1
          Line 2
```

In [251]:
```
print ("Col 1\tCol2")
```

```
Col 1	Col2
```

Σε περίπτωση που θέλουμε να γράψουμε ένα μεγάλος string που έχει μέσα πολλούς ειδικούς χαρακτήρε (αυτάκια, new lines, κτλ..) μπορούμε να χρησημοποιείσουμε τα τριπλά μονά ή διπλα αυτάκια:

In [252]:
```
a = '''
"Be realistic - demand the impossible!"
    Soyez réalistes, demandez l'impossible! - Anonymous graffiti, Paris 1968
'''
print(a)
```

```
"Be realistic - demand the impossible!"
    Soyez réalistes, demandez l'impossible! - Anonymous graffiti, Paris 1968
```

Η πράξη: +=

Έστω ότι έχουμε μία μεταβλητή που έχει την τιμή 3:

In [135]:
```
a = 3
print (a)
```

```
3
```

Πως μπορούμε να της αυξήσουμε τη τιμή κατά 1;

In [136]:
```
a = a + 1
print (a)
```

```
4
```

Το `a=a+1` χρησημοποιείται πολύ συχνά (στην ουσία κάθε φορά που "μετράμε" κάτι). Οπότε μπορούμε να το γράψουμε και σαν: `a += 1` . Παρομοίως μπορούμε να γράψουμε και `a += 4`

In [137]:
```
a += 4 # a = a + 4
print (a)
```

```
8
```

Το ίδιο μπορεί να γίνει με όλες τις άλλες πράξεις. Π.χ. το `a -= 1` είναι ισοδύναμε με `a = a - 1` .

In [138]:
```
a -= 1
print (a)
```

```
7
```

Αφού η πράξη της πρόσθεσης επιτρέπεται μεταξύ λίστες, μπορούμε να χρησιμοποιήσουμε το `+=` για να προσθέσουμε στοιχεία σε μία λίστα:

In [281]:
```
a = [1,2,3]
a += [5,6] # Αυτό είναι ισοδύναμο με: a = a + [5,6]
print (a)
```

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

In [165]:
```
a = [1,2,3,4]
a += [0,1]
print (a)
```

```
[1, 2, 3, 4, 0, 1]
```

Ένας άλλον τρόπος να εισάγουμε νέα στοιχεία σε μία λίστα είναι μέσω της `extend` . Η `extend` είναι ισοδύναμη με το `+=` :

In [295]:
```
a = [1,2,3,4]
a.extend([0,1]) # equivalent with a += [0,1]
print (a)
```

```
[1, 2, 3, 4, 0, 1]
```

Η `extend` δέχεται ως όρισμα μόνο λίστα!

In [298]:
```
a = [1,2,3,4,5]
a.extend(8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-298-396500df017c> in <module>()
      1 a = [1,2,3,4,5]
----> 2 a.extend(8)

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

Αν θελουμε να βάλουμε ΕΝΑ στοιχείο στη λίστα μπορούμε να χρησιμοποιήσουμε την `append` . Η `a.append(b)` είναι ισοδύναμη με: `a += [b]` ή `a = a + [b]`

In [296]:
```
a = [1,2,3,4]
a.append([0,1])
print (a)
```

```
[1, 2, 3, 4, [0, 1]]
```

In [297]:
```
a = [1,2,3,4,5]
a.append(8)
print (a)
```

```
[1, 2, 3, 4, 5, 8]
```

Λογικές μεταβλητές

Μέχρι στιγμής έχουμε μάθει τις αριθμητικές (`a=3.2`) αλφαριθμητικές (`a="mitsos"`) και λίστες (`a=[1,2,3]`) μεταβλητές. Υπάρχουν ακόμα οι λογικές μεταβλητές (https://el.wikipedia.org/wiki/%CE%86%CE%BB%CE%B3%CE%B5%CE%B2%CF%81%CE%B1_%CE%9C%CF%80%CE%BF%CF%85%CE%BB). Οι τιμές που μπορούν να πάρουν είναι: `True` ή `False` προσοχή πρέπει να είναι κεφαλαίο το πρώτο γράμμα).

In [2]:
```
a = True
a = False
```

Η πράξη `and` είναι `True` αν και οι δύο μεταβλητές είναι `True` :

In [3]:
```
True and True
```

Out[3]: True

In [4]:
```
True and False
```

Out[4]: False

In [5]:
```
False and True
```

Out[5]: False

In [6]:
```
False and False
```

Out[6]: False

Η πράξη `or` είναι `True` αν έστω μία από τις δύο μεταβλητές είναι `True` :

In [7]:
```
True or True
```

Out[7]: True

In [8]:
```
True or False
```

Out[8]: True

In [9]:
```
False or True
```

Out[9]: True

In [10]:
```
False or False
```

Out[10]: False

Η πράξη `not` αντιστρέφει μία λογική έκφραση:

In [11]:
```
not False
```

Out[11]: True

In [12]:
```
not True
```

Out[12]: False

Μπορούμε να συνθέτουμε πολύπλοκες λογικές εκφράσεις:

In [13]:
```
False and (True or False)
```

Out[13]: False

In [14]:
```
True and (True or False)
```

Out[14]:
```
True
```

Συνδοιασμός μεταβλητών διαφορετικών τύπων

float και int μας κάνει float:

In [15]:
```
3+0.0
```

Out[15]:
```
3.0
```

In [16]:
```
0 + 0.0
```

Out[16]:
```
0.0
```

Η διαίρεση έχει πάντα αποτέλεσμα float:

In [17]:
```
5/2
```

Out[17]:
```
2.5
```

In [18]:
```
6/2
```

Out[18]:
```
3.0
```

float/int και string δεν επιτρέπεται

In [19]:
```
4.5 + "μίτσος"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-835a49c7937c> in <module>()
----> 1 4.5 + "μίτσος"

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

όταν αναμειγνύουμε float/int με boolean τότε το True αντιστοιχεί με 1 και το False με 0:

In [21]:
```
4 + True
```

Out[21]:
```
5
```

In [22]:
```
4 * False
```

Out[22]:
```
0
```

In [23]:
```
6 / True
```

Out[23]:
```
6.0
```

Μπορούμε να κάνουμε και το εξής:

In [24]:
```
'Μήτσος' * True # είναι το ίδιο με 'Μήτσος' * 1
```

Out[24]: 'Μήτσος'

In [26]:
```
'Μήτσος' * False #  είναι το ίδιο με 'Μήτσος' * 0
```

Out[26]: ''

Μπορούμε να προσθέσουμε True/False μεταβλητές μεταξύ τους!

Και γενικότερα μπορούμε να κάνουμε οποιαδήποτε μαθηματική πράξη

In [27]:
```
True + True
```

Out[27]: 2

In [28]:
```
True + False + True
```

Out[28]: 2

In [29]:
```
(True + False) / (True + True)
```

Out[29]: 0.5

In [30]:
```
True * True * True * True * True * True
```

Out[30]: 1

In [31]:
```
True * True * True * True * False * True
```

Out[31]: 0

Οι τελεστές `and` και `or` με μεταβλητές που ΔΕΝ είναι boolean

Θυμάστε τους τελεστές `and` και `or` . Π.χ:

In [65]:
```
True and False
```

Out[65]: False

Τι θα γίνει αν τους χρησιμοποιήσω με μεταβλητές (ή σταθερές) που ΔΕΝ είναι boolean;

Αν κάνω `A and B and C and ... and Z` θα μου επιστρέψει τη πρώτη έκφραση που είναι False. Αν δεν υπάρχει καμία που να είναι False, θα μου επιστρέψει τη τελευταία:

In [69]:
```
5 and '' and 'Μήτσος'
```

Out[69]: ''

In [72]:
```
5 and 'Μήτσος' and 0.0
```

Out[72]: 0.0

In [73]:
```
5 and 'Μήτσος' and 3.2
```

Out[73]: 3.2

Γιατί όμως γίνεται αυτό; Γιατί όταν σε μία έκφραση A and B and C το B είναι False, τότε δεν έχει νόημα να δούμε τι τιμή είναι το C. Είτε το C είναι True, είτε False, το αποτέλεσμα θα είναι False. Οπότε στην ουσία η python επιστρέφει τη τιμή της έκφρασης που αποτίμησε τελευταία.

Αυτή η τεχνική ονομάζεται [short-circuit evaluation (https://en.wikipedia.org/wiki/Short-circuit_evaluation)]

Ομοίως α κάνουμε: `A or B or C or ... or Z` . Θα επθστρέχει τη πρώτη τιμή που είναι True. Αν δεν υπάρχει καμία που να είναι True, τότε θα επιστρέψει τη τελευταία:

```
In [75]: 0 or 5.3 or 'Μήτσος'
```

```
Out[75]: 5.3
```

```
In [76]: 0 or 5.3 or ''
```

```
Out[76]: 5.3
```

```
In [77]: 0 or False or ''
```

```
Out[77]: ''
```

Μετατροπή τύπων

Υπάρχουν ειδικές συναρτήσεις για να μετατρέπουμε μεταβλητές από έναν τύπο στον άλλο:

- `int` μετατράπει σε ακέραιο
- `float` μετατρέπει σε δεκαδικό
- `bool` μετατρέπει σε δυαδικό
- `str` μετατρέπει σε αλφαριθμητικό

Μερικά παραδείγματα:

```
In [32]: int('42')
```

```
Out[32]: 42
```

```
In [33]: int('42.4')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-c0c93863b08a> in <module>()
----> 1 int('42.4')

ValueError: invalid literal for int() with base 10: '42.4'
```

```
In [34]: int(42.4)
```

```
Out[34]: 42
```

```
In [35]: int(True)
```

```
Out[35]: 1
```

In [36]:
```
int(False)
```

Out[36]:
```
0
```

In [37]:
```
int(42)
```

Out[37]:
```
42
```

In [39]:
```
int('mitsos')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-39-24e8b5b4a1dd> in <module>()
----> 1 int('mitsos')

ValueError: invalid literal for int() with base 10: 'mitsos'
```

In [40]:
```
int('                                42')
```

Out[40]:
```
42
```

In [41]:
```
int('42                                  ')
```

Out[41]:
```
42
```

In [42]:
```
int('               42                    ')
```

Out[42]:
```
42
```

In [43]:
```
float('3.4')
```

Out[43]:
```
3.4
```

In [44]:
```
float('3')
```

Out[44]:
```
3.0
```

In [45]:
```
float('')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-45-45d756431581> in <module>()
----> 1 float('')

ValueError: could not convert string to float:
```

In [46]:
```
float('mitsos')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-a78f2c30f998> in <module>()
----> 1 float('mitsos')

ValueError: could not convert string to float: 'mitsos'
```

In [47]:
```
float('3.4             ')
```

Out[47]:
```
3.4
```

In [48]:
```
float('   3.4           ')
```

Out[48]:
```
3.4
```

In [49]:
```
float('               3.4')
```
Out[49]: 3.4

In [50]:
```
float(3)
```
Out[50]: 3.0

In [51]:
```
float(3.4)
```
Out[51]: 3.4

In [52]:
```
float(True)
```
Out[52]: 1.0

In [53]:
```
float(False)
```
Out[53]: 0.0

In [55]:
```
bool(2)
```
Out[55]: True

In [56]:
```
bool(0)
```
Out[56]: False

In [57]:
```
bool(3.3)
```
Out[57]: True

In [58]:
```
bool(0.0)
```
Out[58]: False

In [59]:
```
bool(0.000000000001)
```
Out[59]: True

In [60]:
```
bool('mitsos')
```
Out[60]: True

In [61]:
```
bool('')
```
Out[61]: False

In [62]:
```
bool(' ')
```
Out[62]: True

In [63]:
```
bool(True)
```
Out[63]: True

In [64]:
```
bool(False)
```
Out[64]: False

Βοήθεια και οδηγίες

Η python περιέχει κάποιες βασικές οδηγίες και βοήθεια:

In [300]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

In [301]:
```
help("".count)
```

```
Help on built-in function count:

count(...) method of builtins.str instance
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are
    interpreted as in slice notation.
```

In [303]:
```
help([].append)
```

```
Help on built-in function append:

append(...) method of builtins.list instance
    L.append(object) -> None -- append object to end
```