Προγραμματισμός με τη γλώσσα python

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

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/).

Μία πρώτη γεύση

print

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

```
kati
```

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

```
In [1]: print ("hello")
```

```
hello
```

```
In [9]: print ("""hello""")
```

```
hello
```

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

```
hello
```

Μπορούμε να βάλουμε ένα "Enter" μέσα σε ένα string με "\n":

In [4]:
```
print ("hello\nworld")
```

```
hello
world
```

Ομοίως μπορούμε να βάλουμε μονά ή διπλά αυτάκια σε ένα string. Ανάλογα με το τι χρησιμοποιούμε για να δηλώσουμε ένα string (μονά ή διπλά αυτάκια) θα πρέπει να χρησιμοποιήσουμε \' ή \":

In [5]:
```
print ("his master's voice")
```

```
his master's voice
```

In [6]:
```
print ('his master\'s voice')
```

```
his master's voice
```

In [7]:
```
print (" I am \"fear\"  ")
```

```
 I am "fear"
```

In [8]:
```
print ('i am "fear"')
```

```
i am "fear"
```

Αν χρησιμούμε τριπλά αυτάκια μπορούμε να έχουμε πολλές γραμμές σε ένα string (multiline strings):

In [12]:
```
print ("""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.""")
```

```
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.
```

(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:

Η python μπορεί να κάνεις πράξεις με ακέραιους με οποιοδήποτε πλήθος αριθμών:

In [13]:
```
2432847023984750293467209834752034986 7 * 23457345872983561938475639
8456
```

Out[13]:
```
5706813409766902302233897564852365380972748955782671565995978605352
```

Έχουμε τις κλασσικές πράξεις: πρόσθεση, αφαίρεση, πολλαπλασιασμός διαίρεση:

In [15]:
```
3+2
```

Out[15]: 5

In [16]:
```
3-2
```

Out[16]: 1

In [17]:
```
3*2
```

Out[17]: 6

Δεκαδική διαίρεση:

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

Out[12]: 1.5

Ακέραια διαίρεση:

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

Out[13]: 1

Προσοχή!

In [18]:
```
1/0
```

```
---------------------------------------------------------------------
---------
ZeroDivisionError                         Traceback (most recent c
all last)
<ipython-input-18-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

Έχουμε επίσης κάποιες επιπλέον πράξεις:

Το υπόλοιπο της διαίρεσης:

```
In [19]: 15 % 4
```

```
Out[19]: 3
```

Δηλαδη 15 = 3*4 + **3**

Το εκθετικό:

```
In [20]: 4**2
```

```
Out[20]: 16
```

Προσοχή: Το εκθετικό ΔΕΝ είναι \^:

```
In [21]: 4 ^ 2
```

```
Out[21]: 6
```

Το ^ είναι μια άλλη πράξη που ονομάζεται XOR (https://en.wikipedia.org/wiki/Exclusive_or) και δεν θα μας απασχολήσει σε αυτό το μάθημα.

Στη python κάθε πράξη έχει μία προτεραιότητα. Για παράδειγμα οι πολλαπλασιασμοί και διαρέσεις εκτελούνται **πριν** τις προσθέσεις και αφαιρέσεις:

```
In [22]: 10+6/2
```

```
Out[22]: 13.0
```

Σημείωση: Ποτέ δεν "εμπιστευόμαστε" τη προτεραιότητα πράξεων! Πολλές φορές δεν είναι και τόσο ξεκάθαρο. Χρησιμοποιούμε παρενθέσεις:

```
In [23]: 10+(6/2)
```

```
Out[23]: 13.0
```

```
In [24]: (10+6)/2
```

```
Out[24]: 8.0
```

Αν μία πράξη σε οποιοδήποτε σημείο της έχει τη διαίρεση **ή** συμμετέχει κάποιος δεκαδικός αριθμός, το αποτέλεσμα είναι δεκαδικό, αλλιώς είναι ακέραιος:

```
In [25]: 2/2
```

```
Out[25]: 1.0
```

```
In [26]: 2*2
```

```
Out[26]: 4
```

```
In [27]: 2 + 2.0
```

```
Out[27]: 4.0
```

```
In [48]: 2+2
```

```
Out[48]: 4
```

Η python έχει μία:

- συνώνυμη λέξη για το 1: το `True`
- συνώνυμη λέξη για το 0: το `False`

```
In [49]: True + 1
```

```
Out[49]: 2
```

```
In [50]: False + 1
```

```
Out[50]: 1
```

Τα `False` και `True` θα τα δούμε λίγο αργότερα!

Παρόλο που οι ακέραιοι μπορούν να έχουν απεριοριστό πλήθος από ψηφία, οι δεκαδικοί αριθμοί έχουν συγκεκριμένη ακρίβεια:

```
In [156]: 5.123456789012345678901234567
```

```
Out[156]: 5.123456789012345
```

Γιατί γίνεται αυτό; Έναν ακέραιο, όσο μεγάλος και να είναι, μπορούμε να τον αναπαραστήσουμε στη μνήμη χωρίς να χάνουμε ακρίβεια. Για κάποιους δεκαδικούς όμως είναι απλά αδύνατο να έχουμε άπειρη ακρίβεια. Για παράδειγμα:

```
In [157]: 1/3
```

```
Out[157]: 0.3333333333333333
```

Το `1/3` έχει άπειρα δεκαδικά ψηφία! Πως θα το αποθηκεύσουμε αυτό στη μνήμη; Η λύση είναι να αποθηκεύουμε ένα συγκεκριμένο πλήθος από δεκαδικά ψηφία. Ευτυχώς υπάρχει ένα διεθνές πρότυπο IEEE-754 (https://en.wikipedia.org/wiki/IEEE_754) το οποίο καθορίζει με ποιο τρόπο αποθηκεύουμε τους δεκαδικούς αριθμούς. Παρόλα αυτά μπορείτε να καθοδηγήσετε τη python (https://docs.python.org/3/library/decimal.html) να μη χρησιμοποιεί αυτό το πρότυπο και να χειρίζεστε δεκαδικούς με όση ακρίβεια θέλετε (θυσιάζοντας λίγο τη μνήμη και τη ταχύτητα των υπολογισμών).

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 άλλο θεωρούνται ένα!

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

```
Out[271]: 'Helloworld'
```

Ένα string μπορεί να έχει χαρακτήρες σε οποιαδίποτε γλώσσα!

```
In [29]: a = "σε οποιαδίποτε γλώσσα ذا كيور (بالإنجليزية: The Cure) هي فرقة ر وك إنجليزية، تم تكوينها في كرولي، غرب ساسكس عام 1976. واجهت الفرقة عِدة تغيرات؛ مع "
print (a)
```

```
σε οποιαδίποτε γλώσσα ذا كيور (بالإنجليزية: The Cure) هي فرقة روك إ نجليزية، 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")
```

😡

Μετά από τα παραπάνω που μας έβαλαν σιγά σιγά στη python, μπορούμε να μιλήσουμε λίγο πιο θεωρητικά:

Τελεστές

Οι τελεστες είναι σύμβολα ή δεσμευμένες λέξεις με τους οποίους εφαρμόζουμε βασικές πράξεις σε διάφορες εκφράσεις. Για περισσότερα μπορείτε να διαβάσετε εδώ: https://en.wikipedia.org/wiki/Operator_(computer_programming (https://en.wikipedia.org/wiki/Operator_(computer_programming))

Μερικοί από τους πιο βασικους τελεστές που υποστηρίζει η python είναι:

- +
- -
- /
- //
- *
- %
- **
- <
- >
- <=
- >=
- !=
- ==
- and
- or
- not
- in
- is

Ο τελεστής +

Οι πράξεις που επιστρέπονται με τον τελεστή '+' είναι:

```
In [32]: 3+2
Out[32]: 5
```

```
In [33]: 3+2.0
Out[33]: 5.0
```

In [34]:
```
3+0.0
```
Out[34]: 3.0

In [35]:
```
'ab' + 'cde'
```
Out[35]: 'abcde'

In [36]:
```
True + True + False
```
Out[36]: 2

In [37]:
```
True + 2
```
Out[37]: 3

In [38]:
```
True + 0.0
```
Out[38]: 1.0

In [39]:
```
[1,2,3] + [4,5,6] # Λίστες θα το δούμε αργότερα
```
Out[39]: [1, 2, 3, 4, 5, 6]

Ο τελεστής -

Οι πράξεις που επιτρέπονται με τον τελεστή '-' είναι:

In [40]:
```
3-2
```
Out[40]: 1

In [41]:
```
3-7
```
Out[41]: -4

In [42]:
```
4-6.0
```
Out[42]: -2.0

In [43]:
```
True - True
```
Out[43]: 0

In [44]:
```
True - 6.6
```
Out[44]: -5.6

Ο τελεστής *

Οι πράξεις που επιτρέπονται με τον τελεστή '*' είναι:

In [45]:
```
6*7
```

Out[45]: 42

In [46]:
```
6.6*2
```

Out[46]: 13.2

In [47]:
```
True * 2
```

Out[47]: 2

In [51]:
```
True * False
```

Out[51]: 0

In [52]:
```
True * 2.3
```

Out[52]: 2.3

In [53]:
```
6 * 'hello'
```

Out[53]: 'hellohellohellohellohellohello'

In [55]:
```
[1,2,3] * 5  # Λίστες, θα τα δούμε αργότερα..
```

Out[55]: [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

Ο τελεστής '/'

Οι πράξεις που επιτρέπονται με τον τελεστή '/' είναι:

In [56]:
```
4/5
```

Out[56]: 0.8

ΠΡΟΣΟΧΗ!!

In [57]:
```
5/0
```

```
---------------------------------------------------------------------
---------
ZeroDivisionError                         Traceback (most recent c
all last)
<ipython-input-57-0106664d39e8> in <module>()
----> 1 5/0

ZeroDivisionError: division by zero
```

In [58]:
```
True/True
```

Out[58]: 1.0

In [59]:
```
6/3
```

Out[59]: 2.0

In [60]:
```
6.5/3
```

Out[60]: 2.1666666666666665

Ο τελεστής '//'

Αυτός ο τελεστής μας δίνει το αποτέλεσμα της ακέραιας διαίρεσης

In [62]:
```
5//2
```

Out[62]: 2

In [63]:
```
11//3
```

Out[63]: 3

In [64]:
```
6.5 // 2.1
```

Out[64]: 3.0

In [65]:
```
True // 2
```

Out[65]: 0

Ο τελεστής %

Αυτός ο τελεστής μας δίνει το υπόλοιπο της ακέραιας διαίρεσης

In [66]:
```
5%2
```

Out[66]: 1

In [67]:
```
5.2 % 2
```

Out[67]: 1.2000000000000002

In [68]:
```
True % 2
```

Out[68]: 1

Ο τελεστής % χρησιμοποιείται (όχι και τόσο συχνά) για να βάλουμε strings μέσα σε strings

In [70]:
```
'my name is %s nice to meet you' % "mitsos"
```

Out[70]: 'my name is mitsos nice to meet you'

Μπορείτε να βρείτε εδώ (https://python-reference.readthedocs.io/en/latest/docs/str/formatting.html) για το πως μπορείτε να χρησιμοποιείσετε αυτόν τον τελεστή για strings με περισσότερα παραδείγματα

Ο τελεστής **

Αυτός ο τελεστής μας επιστρέφει το εκθετικό a^b

```
In [71]: 3**2
```

```
Out[71]: 9
```

```
In [72]: 3.2**2.3
```

```
Out[72]: 14.515932837559118
```

```
In [73]: True ** 2
```

```
Out[73]: 1
```

Λογικοί Τελεστές

Είδαμε πριν τις σταθερές `True` και `False` . Ποιος είναι ο σκοπός τους; Μέχρι στιγμής είδαμε τελεστές οι οποίοι μπορούν να παράγουν αριθμούς. Για την παράδειγμα ο τελεστής `+` μπορεί να παράξει έναν οποιοδήποτε αριθμό. Υπάρχει όμως μία ομάδα τελεστών που μπορούν να παράξουν 2 μόνο διαφορετικές τιμές. Τις τιμές `True` και `False` . Οι τελεστές αυτές είναι:

- Οι τελεστές σύγκρισης <, >, <=, >=
- Οι τελεστές ισότητας ==, !=
- Οι τελεστές and, or, not
- Οι τελεστές in, is (θα τους δούμε αργότερα)

Αυτοί οι τελεστές επιστρέφουν **πάντα** True ή False σε **ότι** και αν τους εφαρμόσουμε!

Ο α < β ελέγχει αν το α είναι μικρότερο από το β:

```
In [74]: 2<3
```

```
Out[74]: True
```

```
In [76]: 3<2
```

```
Out[76]: False
```

```
In [77]: 3<3
```

```
Out[77]: False
```

Ο α <= β ελέγχει α είναι μικρότερο ή ίσο με το β:

```
In [78]: 2<=3
Out[78]: True

In [79]: 3<=2
Out[79]: False

In [80]: 3<=3
Out[80]: True
```

Ο α > β ελέγχει αν το α είναι μεγαλύτερο από το β:

```
In [82]: 3 > 2
Out[82]: True

In [85]: 2 > 3
Out[85]: False

In [86]: 3 > 3
Out[86]: False
```

Ο α >= β ελέγχει αν το α είναι μεγαλύτερο ή ίσο με β:

```
In [87]: 3 >= 2
Out[87]: True

In [88]: 2 >= 3
Out[88]: False

In [89]: 3 >= 3
Out[89]: True
```

Ο τελέστής α == β ελέγχει αν το α είναι ίσο με το β

```
In [90]: 3==2
Out[90]: False

In [91]: 3==3
Out[91]: True

In [92]: 3==3.0
Out[92]: True
```

In [93]:
```
"3" == 3
```
Out[93]: False

In [114]:
```
16**0.5 == 4
```
Out[114]: True

In [115]:
```
'mitsos' == 'mits' + 'os'
```
Out[115]: True

In [116]:
```
3 == 6/2
```
Out[116]: True

In [117]:
```
True == True or False
```
Out[117]: True

In [118]:
```
3 == True + True + True + False
```
Out[118]: True

In [119]:
```
3 == 'mits' + 'os'
```
Out[119]: False

In [121]:
```
[1,2,3] == [1,2,3] # Λίστες, θα τα δούμε αργότερα
```
Out[121]: True

In [122]:
```
[1,2,3] == [2,1,3]
```
Out[122]: False

Ο τελεστής α != β αν το α **δεν** είναι ίσο με το β

In [95]:
```
3 != 2
```
Out[95]: True

In [108]:
```
2 != 2
```
Out[108]: False

In [109]:
```
2 != 2.0
```
Out[109]: False

In [110]:
```
1 != True
```
Out[110]: False

```
In [111]: 'hello' != ' hello '
Out[111]: True

In [112]: '' != ''
Out[112]: False

In [113]: '' != ' '
Out[113]: True
```

Μπορούμε επίσης να χρησιμοποιήσουμε τους ίδιους τελεστές <, >, <=, >= παραπάνω από μία φορά:

```
In [97]: 2<3<4
Out[97]: True

In [98]: 2<3<3
Out[98]: False
```

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

```
In [99]: 'ab' < 'fg'
Out[99]: True

In [100]: 'ab' < 'b'
Out[100]: True

In [101]: 'ab' < 'ac'
Out[101]: True

In [102]: 'ab' < 'a'
Out[102]: False
```

Το άδειο string έχει τη πιο μικρή δυνατή τιμή

```
In [103]: '' < '0'
Out[103]: True

In [104]: "A" < "a"
Out[104]: True
```

In [105]:
```
"05456745674" < "5"
```

Out[105]: True

In [106]:
```
'8' < '09'
```

Out[106]: False

Πολλές φορές θέλουμε να πάρουμε μία απόφαση ανάλογα με τα αν 2 ή παραπάνω λογικές τιμές είναι αληθής. Για παράδειγμα:

Θα βγω έξω αν κάνει καλό καιρό **και** δεν έχω δουλειά.

Θα βγω έξω αν κάνει καλό καιρό **ή** δεν έχω δουλεία.

Έτσι έχουμε δύο επειπλέον λογικούς τελεστές: `and` , `or` .

Ο α and β είναι `True` αν τα α και τα β είναι `True` , αν ένα από τα α,β είναι `False` είναι `False` (ή και τα δύο), τότε το αποτέλεσμα είναι `False` :

In [123]:
```
True and True
```

Out[123]: True

In [124]:
```
True and False
```

Out[124]: False

In [125]:
```
False and True
```

Out[125]: False

In [126]:
```
False and False
```

Out[126]: False

In [127]:
```
(1==1) and (2==3-1)
```

Out[127]: True

In [128]:
```
(1==1) and (2==3)
```

Out[128]: False

In [129]:
```
(1>=1) and (2<=2)
```

Out[129]: True

Ομοίως το αποτέλεσμα της πράξη α or β είναι `True` αν ένα από τα α,β (ή και τα δύο) είναι `True` , διαφορετικά είναι `False` :

In [130]:
```
True or True
```

Out[130]: True

In [131]:
```
True or False
```

Out[131]: True

In [132]:
```
False or True
```

Out[132]: True

In [133]:
```
False or False
```

Out[133]: False

In [134]:
```
1==2 or 1<=1
```

Out[134]: True

In [136]:
```
1>2 or 2<1
```

Out[136]: False

In [137]:
```
0==1 or True
```

Out[137]: True

Τέλος, υπάρχει ο τελεστής `not` . Αυτός ο τελεστής έχειτην ιδαιτερότητα ότι εφαρμόζεται σε μία μόνο τιμή. Το `not` α, έχει σαν αποτέλεσμα `False` αν το α είναι `True` και `True` αν το α είναι `False` :

In [138]:
```
not True
```

Out[138]: False

In [144]:
```
not False
```

Out[144]: True

In [151]:
```
not 0
```

Out[151]: True

In [152]:
```
not 0.0000000001
```

Out[152]: False

In [146]:
```
not ''
```

Out[146]: True

In [147]:
```
not 1
```

Out[147]: False

In [148]:
```
not ' '
```
Out[148]: False

In [140]:
```
not 3==4
```
Out[140]: True

In [141]:
```
not 3==3
```
Out[141]: False

In [142]:
```
not "mitsos"=="Mitsos"
```
Out[142]: True

In [149]:
```
not "mitsos" == "mitsos"
```
Out[149]: False

Fun fact: Ο υπολογιστής που έχετε και κάνει όλα τα απίστευτα πράγματα που κάνει στην ουσία μπορεί να κάνει μία και μόνο μία πράξη: `not (a and b)` . Αυτή η πράξη ονομάζεται NAND. Για παράδειγμα όταν ο υπολογιστής κάνει μία μαθηματική πράξη (π.χ. `14.2 * 51.1`), "σπάει" τη πράξη αυτή σε πράξεις NAND. Δηλαδή ο επεξεργαστής έχει δισεκατομύρια κυκλώματα που κάνουν αυτή και μόνο αυτή τη πράξη. Είναι όμως οργανωμένα έτσι ώστε όταν συνδυαστούν με τον σωστό τρόπο να κάνουν όλες τις αριθμητικές πράξεις! Περισσότερα (https://en.wikipedia.org/wiki/NAND_logic). Ακόμα και όταν κάνει πιο πολύπλοκα πράγμα (streaming, παίζει ένα παιχνίδι, ελέγχει έναν πυρηνικό αντιδραστήρα, καθοδηγεί ένα διαστημόπλοιο) πάλι σπάει όλες τις πράξεις που χρειάζονται σε πράξεις NAND.

Επιστροφή στα strings!

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 [173]:
```
"hello world".replace('l', "QQQ")
```
Out[173]: 'heQQQQQQo worQQQd'

Μπορούμε να αφαιρέσουμε τα κενά από το τέλος και από την αρχή:

```
In [176]: "    hello    "
Out[176]: '    hello    '

In [178]: "    hello    ".strip()
Out[178]: 'hello'

In [179]: "+++hello+++".strip('+')
Out[179]: 'hello'

In [180]: not "     "
Out[180]: False

In [182]: not "     ".strip()
Out[182]: True
```

Έλεγχος αν ένα string αρχίζει με ένα άλλο string:

```
In [183]: "heraklio".startswith('her')
Out[183]: True
```

Έλεγχος αν ένα string τελειώνει με ένα άλλο string:

```
In [184]: "alex".endswith("lex")
Out[184]: True
```

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 c
all 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 c
all 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 c
all 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 [172]:
```
print( '''
"Be realistic - demand the impossible!"
    Soyez réalistes, demandez l'impossible! - Anonymous graffiti, P
aris 1968
''')
```

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

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 c
all 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]: ''
```

Έλεγχος του τύπου μιας τιμής

Η συνάρτηση `type` επιστρέφει ένα string το οποίο περιέχει τον τύπο μίας τιμής:

In [162]:
```
type(2)
```

Out[162]:
```
int
```

In [163]:
```
type(2.0)
```

Out[163]:
```
float
```

In [164]:
```
type('')
```

Out[164]:
```
str
```

In [165]:
```
type('mitsos')
```

Out[165]:
```
str
```

In [166]:
```
type(True)
```

Out[166]:
```
bool
```

In [168]:
```
type(1==2)
```

Out[168]:
```
bool
```

In [169]:
```
type(1+2)
```

Out[169]:
```
int
```

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 c
all 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 c
all 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 c
all last)
<ipython-input-45-45d756431581> in <module>()
----> 1 float('')

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

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

```
---------------------------------------------------------------------
---------
ValueError                                Traceback (most recent c
all 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

Όλα αυτά είναι πολλά! Πως θα τα θυμάμαι;

Δεν χρειάζεστε να θυμάστε πολλά.. Έχετε πάντα το google.. αν ρωτήσετε "how to ... in python" συνήθως το πρώτο αποτέλεσμα θα έχει μία πολύ καλή απάντηση! Πρόσφατα ο δημιουργός της python (https://en.wikipedia.org/wiki/Guido_van_Rossum) είπε ότι χρησιμοποιεί ο ίδιος το google για να βρίσκει πως θα κάνει μερικά πράγματα στη.. python.

Παρόλα αυτά 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.
```

Δοκιμάστε και:

In [171]:
```
?len
```