Προγραμματισμός με τη γλώσσα python

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

Διάλεξη 6η, Τρίτη 19 Νοεμβρίου 2019

import

Με την `import` μπορούμε να "βάλουμε" στο περιβάλλον που δουλεύουμε κώδικα από ένα άλλο αρχείο. Ας υποθέσουμε ότι έχουμε το αρχείο `a.py` το οποίο έχει τον παρακάτω κώδικα:

```
# File: a.py

def f():
    print ("hello")

def g():
    print ("world")
```

Τότε μπορούμε να κάνουμε `import` τις συναρτήσεις αυτές σε ένα άλλο αρχείο:

In [1]:
```
from a import f # Κάνω import μόνο την f
f()
```

```
hello
```

In [2]:
```
from a import f,g # Κάνω import την f και τη g
f()
g()
```

```
hello
world
```

In [3]:
```
from a import * # Κάνω import όλες τις συναρτήσεις / μεταβλητές / ... που υπάρχουν στο a.py
f()
g()
```

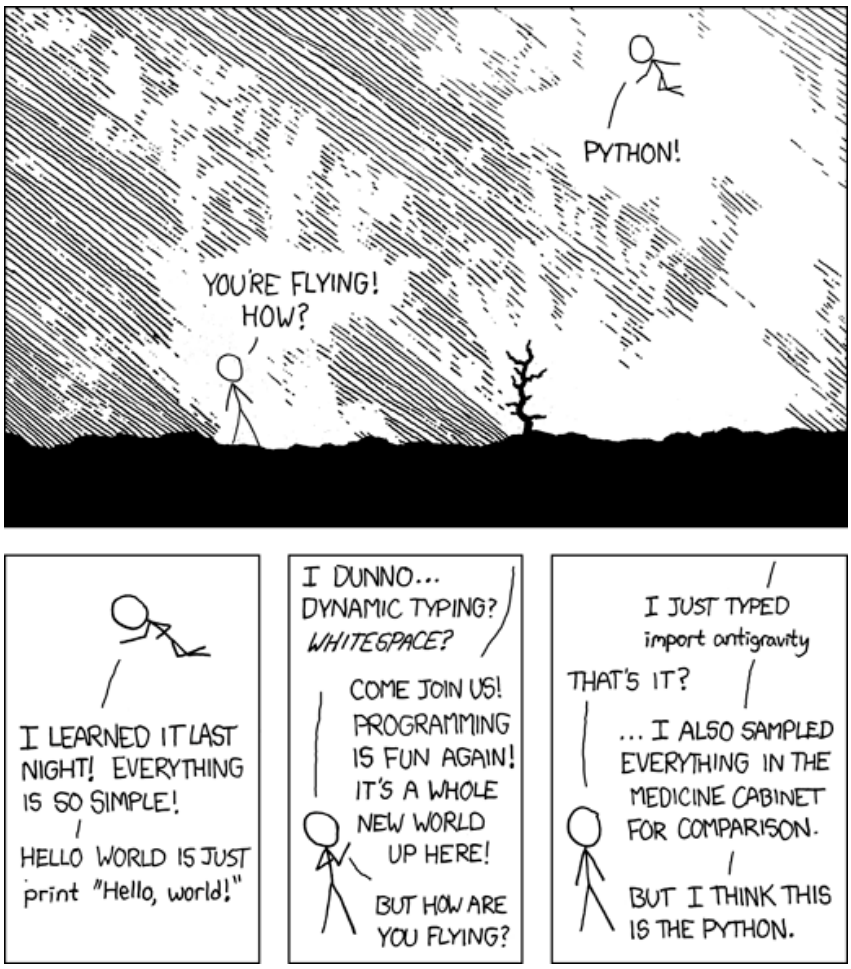
```
hello
world
```

Μπορώ να κάνω `import` το `a` και να χρησιμοποιώ το `a.f()` όταν θέλω να καλέσω μία συνάρτηση:

In [4]:
```
import a
a.f()
a.g()
```

```
hello
world
```

Η python είναι μία γλώσσα "batteries included (https://en.wikipedia.org/wiki/Batteries_Included)". Το οποίο σημαίνει ότι έχει πάρα πολλές βιβλιοθήκες (libraries) με χρήσιμες συναρτήσεις (https://docs.python.org/3/library/index.html). Τις συναρτήσεις αυτές τις βάζουμε με την import



πηγή (https://xkcd.com/353/)

Δοκιμάστε:

```
import antigravity
```

Κάποια παραδείγματα από τις βιβλιοθήκες της python

```
In [5]: import random
```

```
In [6]: random.random() # τυχαίοι αριθμοί από το 0 μέχρι το 1
Out[6]: 0.16162891645386535
```

```
In [7]: random.randint(1,10) # τυχαίοι ακέραιοι από το 1 μέχρι το 10
Out[7]: 10
```

```
In [8]: random.choice(['Ηράκλειο', 'Άμστερνταμ', 'Βερολίνο']) # Διαλέγει ένα στη τύχη
Out[8]: 'Ηράκλειο'
```

In [9]:
```
random.sample(['Ηράκλειο', 'Αμστερνταμ', 'Βερολίνο'],2) # Διαλέγει 2 στη τύχη
```

Out[9]:
```
['Βερολίνο', 'Ηράκλειο']
```

Η βιβλιοθήκη itertools περιέχει πολλές χρήσιμες συναρτήσεις για τον χειρισμό λιστών:

In [10]:
```
import itertools
```

In [11]:
```
list(itertools.combinations([1,2,3,4],2)) # Συνδοιασμοί ανά 2
```

Out[11]:
```
[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
```

In [12]:
```
# Πόσες στήλες (δάρια) υπάρχουν στον λόττο;
sum((1 for i in itertools.combinations(range(49), 6)))
```

Out[12]:
```
13983816
```

In [13]:
```
# Πόσες στήλες υπάρχουν στο Τζόκερ;
sum((1 for i in itertools.combinations(range(45), 5))) * 20
```

Out[13]:
```
24435180
```

Με τη βιβλιοθήκη `sys` μπορείτε να έχετε πρόσβαση στις παραμέτρους της γραμμής εντολών. Φτιάξτε το παρακάτω αρχείο pyhon με το όνομα `test.py`

```
import sys

print (sys.argv)
```

Και τρέξτε το από το command line βάζοντας κάποιες παραμέτρους:

In [14]:
```
!python test.py XXX YYY ZZZ
```

```
['test.py', 'XXX', 'YYY', 'ZZZ']
```

Με αυτόν τον τρόπο μπορεί ένας χρήστης να βάζει παραμέτρους στη γραμμή εντολών. Π.χ. μπορεί να δηλώνει ποιο είναι το input αρχείο που θέλει να επεξεργαστείτε.

Με τη βιβλιοθήκη `os` μπορείτε να τρέξετε ένα πρόγραμμα από τη γραμμή εντολών μέσα από τη python:

In [15]:
```
import os
os.system('cp test.py new_test.py') # Αντιγράφει (cp) το αρχείο test.py στο new
_test.py
```

Out[15]:
```
0
```

In [16]:
```
!cat new_test.py
```

```
import sys

print (sys.argv)
```

Επίσης, μπορούμε νε ελέγξουμε αν ένα αρχείο υπάρχει:

In [17]:
```
os.path.isfile('new_test.py')
```

Out[17]:
```
True
```

In [19]:
```
os.path.isfile('xyz')
```

Out[19]:
```
False
```

Regular Expressions

Τα Regular Expressions (https://en.wikipedia.org/wiki/Regular_expression) (ή αλλιώς regexp για συντομία) είναι μια βασική ιδέα στην επιστήμη υπολογιστών (υπάρχουν απο το 1956..). Είναι στην ουσία μία νέα γλώσσα με την οποία μπορείς να δηλώσεις κάποια patterns μέσα σε ένα string. Ειδικοί αλγόριθμοι αναλαμβάνουν να εντοπίσουν αυτά τα patterns με πολύ μεγάλη ταχύτητα. Τα regexp υλοποιούνται στη python στη βιβλιοθήκη `re` :

In [20]:
```
import re # Regular Expression
```

Ας υποθέσουμε ότι έχουμε ένα string:

In [21]:
```
s = 'abc 123 a1 1a'
```

Έστω ότι θέλουμε να αναζητήσουμε ένα γράμμα (π.χ. το 'c')

In [22]:
```
result = re.search(r'c', s)
print(result)
```

```
<re.Match object; span=(2, 3), match='c'>
```

Αυτό μας ενημερώνει ότι έχει βρει κάπου αυτό που του ζητήσαμε. Μπορούμε να το επιβεβαιώσουμε με το `group()`

In [23]:
```
result.group()
```

Out[23]:
```
'c'
```

Αν δεν είχε βρει τίποτα τότε θα επιστρέφει `None`

In [24]:
```
result = re.search(r'd', s) # Το 'd' δεν υπάρχει
print (result)
```

```
None
```

Το 'bc' υπάρχει:

In [25]:
```
result = re.search(r'bc', s)
print(result)
```

```
<re.Match object; span=(1, 3), match='bc'>
```

Το 'cd' δεν υπάρχει:

In [26]:
```
result = re.search(r'cd', s)
print(result)
```

```
None
```

Με τις αγκύλες `[]` μπορούμε να δηλώσουμε ότι ψάχνουμε έναν χαρακτήρα που να ανοίκει σε ένα σετ από χαρακτήρες. Π.χ:

In [27]:
```
a = 'analyze'
b = 'analyse'

result = re.search(r'analy[sz]e', a)
print (result.group())

result = re.search(r'analy[sz]e', b)
print (result.group())
```

```
analyze
analyse
```

Στις αγκύλες μπορούμε να δηλώσουμε ranges από χαρακτήρες:

In [28]:
```
a = 'I am using python 3'
b = 'I am using python 2'

result = re.search(r'python [0-9]', a)
print(result.group())

result = re.search(r'python [0-9]', b)
print(result.group())
```

```
python 3
python 2
```

Τι γίνεται όταν ψάχνουμε "παραπάνω από ένα"; Τότε μπορούμε να χρησιμοποιήσουμε το '+'

In [29]:
```
a = 'My number is 69123456789'
result = re.search(r'[0-9]+', a)
print(result.group())
```

```
69123456789
```

Με το '?' δηλώνουμε: 'κανένα ή 1'

In [30]:
```
a = 'My number is +69123456789'
result = re.search(r'[+]?[0-9]+', a)
print(result.group())

a = 'My number is 69123456789'
result = re.search(r'[+]?[0-9]+', a)
print(result.group())
```

```
+69123456789
69123456789
```

Με το `'*'` μπορούμε να δηλώσουμε "κανένα, 1 ή παραπάνω"

In [31]:
```
a = 'My number is 69123456789' # Κανένα +
result = re.search(r'[+]*[0-9]+', a)
print(result.group())

a = 'My number is +69123456789'
result = re.search(r'[+]*[0-9]+', a) # Ένα +
print(result.group())

a = 'My number is +++++++++69123456789' # Πολλά +
result = re.search(r'[+]*[0-9]+', a)
print(result.group())
```

```
69123456789
+69123456789
+++++++++69123456789
```

Αντί να γράφουμε `[0-9]` για να δηλώνουμε όλους τους αριθμούς και `[a-zA-Z]` γινα δηλώνουμε όλα τα γράμματα μπορούμε να χρησιμοποιήσουμε τα εξής:

`\d` είναι το ίδιο με: `[0-9]`

`\w` είναι το ίδιο με: `[a-zA-Z0-9_]`

In [32]:
```
a = 'my number is 693232314'
result = re.search(r'[\d]+',a)
print (result.group())

a = 'My favorite gene is TPMT'
result = re.search(r'My favorite gene is [\w]+',a)
print (result.group())
```

```
693232314
My favorite gene is TPMT
```

Το `[\s]` κάνει match το [whitespace (https://en.wikipedia.org/wiki/Whitespace_character)](https://en.wikipedia.org/wiki/Whitespace_character)

In [33]:
```
a = 'my number is:                  33333'
result = re.search(r'[\s]+[\d]+',a)
print (result.group())
```

```
                  33333
```

Αν ο πρώτος χαρακτήρας στις αγκύλες είναι το ^ τότε σημαίνει όλους εκτός αυτούς τους χαρακτήρες:

In [34]:
```
a = 'angelopoulos angelopoulou '

result = re.search(r'[\w]+[^s] ', a) # Τα ονόματα που δεν τελειώνουν σε "s"
print(result.group())
```

```
angelopoulou
```

Τα `+`, `?` και `*` προσπαθούν να κάνουν match όσο το δυνατόν περισσότερους χαρακτήρες (greedy match). Πολλές φορές θέλουμε να κάνουν match όσο το δυνατόν λιγότερο. Για να το κάνουμε αυτό χρησιμοποιούμε τα `+?`, `??` και `*?` αντίστοιχα. Π.χ:

In [35]:
```
a = 'I bought 3 bananas I bought 5 bananas I bought 10 bananas' # Σταματάει στο τελευταίο bananas
result = re.search(r'I bought [\w ]+ bananas', a)
print(result.group())

a = 'I bought 3 bananas I bought 5 bananas I bought 10 bananas' # Σταματάσει στο πρώτο bananas
result = re.search(r'I bought [\w ]+? bananas', a)
print(result.group())
```

```
I bought 3 bananas I bought 5 bananas I bought 10 bananas
I bought 3 bananas
```

Το ^ (έξω από αγκύλες) δηλώνει την αρχή του string:

In [36]:
```
a = '1234 abc'
b = 'abc 123'

# Ποια από αυτά τα string ξεκινάει με αριθμό;
for x in [a,b]:
    if re.search(r'^[\d]', x):
        print ('String "{}" starts with a number'.format(x))
```

```
String "1234 abc" starts with a number
```

Το $ δηλώνει το τέλος του string:

In [37]:
```
a = '1234 abc'
b = 'abc 123'

# Ποια από αυτά τα string ξεκινάει με αριθμό;
for x in [a,b]:
    if re.search(r'[\d]$', x):
        print ('String "{}" ends with a number'.format(x))
```

```
String "abc 123" ends with a number
```

Η τελεία δηλώνει οποιοδίποτε χαρακτήρα!

In [38]:
```
a = 'my two letter code is: 54'
b = 'my two letter code is: %!'
c = 'my two letter code is: G44'
for x in [a,b,c]:
    if re.search(r'my two letter code is: ..$', x):
        print ('This is correct: {}'.format(x))
```

```
This is correct: my two letter code is: 54
This is correct: my two letter code is: %!
```

Μπορούμε να χρησιμοποιήσουμε παρενθέσεις για να πάρουμε ένα substring από αυτό που έχουμε κάνει search. Στο αποτέλεσμα του search πρέπει να χρησιμοποιήσουμε το `group(k)` για να πάρουμε το match της k-ης παρένθεσης:

In [39]:
```
a="my favorite variants is rs234234"
result = re.search(r'my favorite variants is (rs[\d]+)',a)
print (result.group(1))
```

```
rs234234
```

In [40]:
```
a = 'My name is John Snow'
result = re.search(r'My name is ([\w]+) ([\w]+)', a)
print (result.group(1), result.group(2))
```

```
John Snow
```

Αντί να κάνουμε `search` για να βρούμε ένα regexp σε ένα string μπορούμε να κάνουμε `match` για να δούμε αν ΟΛΟ το string ικανοποιεί μία regexp:

In [41]:
```
a = 'this is SNP: rs1234567'
result = re.search(r'rs[\d]+', a)
print (result)

a = 'this is SNP: rs1234567'
result = re.match(r'rs[\d]+', a)
print (result)
```

```
<re.Match object; span=(13, 22), match='rs1234567'>
None
```

Δηλαδή το `re.match('PATTERN', x)` είναι ισοδύναμο με το `re.search('^PATTERN$', x)`

Επίσης με τη `findall` μπορούμε να βρούμε όλα τα substrings ενώς string που ικανοποιούν ένα regexp.

In [44]:
```
a = ' rs1213  XYZ rs9897   HELLO rs432432342 '
re.findall(r'rs[\d]+', a)
```

Out[44]:
```
['rs1213', 'rs9897', 'rs432432342']
```

Προσοχή! αυτά είναι μόνο τα βασικά των regular expressions. Υπάρχουν πολύ περισσότερα για τα οποία μπορείτε να διαβάσετε στο documentation της python (https://docs.python.org/3/library/re.html).

πεδία συναρτήσεων

Οι συναρτήσεις μπορούν να έχουν ενσωματωμένο documentation. Αυτό πρέπει να είναι ένα string που δηλώνεται στην αρχή της συνάρτησης:

In [247]:
```
def f():
    '''
    Αυτή η συνάρτηση υπολογίζει τα πάντα.
    '''
    return 42
```

Μπορούμε να προσπελάσουμε το documentation αυτό:

In [248]:
```
f.__doc__
```

Out[248]:
```
'\n    Αυτή η συνάρτηση υπολογίζει τα πάντα.\n    '
```

Επίσης μπορούμε να πάρουμε το όνομα μιας συνάρτησης:

In [250]:
```
f.__name__
```

Out[250]:
```
'f'
```

In [266]:
```
def f():
    return 10

def g():
    return 20

a=f
print (a.__name__)
a=g
print (a.__name__)
```

```
f
g
```

Υπάρχουν πολλά άλλα πεδία για άλλες χρήσεις:

In [267]:
```
dir(f)
```

Out[267]:
```
['__annotations__',
 '__call__',
 '__class__',
 '__closure__',
 '__code__',
 '__defaults__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__get__',
 '__getattribute__',
 '__globals__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__kwdefaults__',
 '__le__',
 '__lt__',
 '__module__',
 '__name__',
 '__ne__',
 '__new__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

pass

Σε οποιοδίποτε καινούργιο indentation (for, while, def, if, ...) πρέπει υποχρεωτικά να έχουμε τουλάχιστον μία εντολή:

In [254]:
```
if True:
```

```
  File "<ipython-input-254-2c8a33b52dfb>", line 1
    if True:
            ^
SyntaxError: unexpected EOF while parsing
```

Αν για κάποιο λόγο δεν θέλουμε να βάλουμε καμία εντολή, τότε μπορούμε να χρησιμοποιήσουμε τη `pass` :

In [255]:
```
if True:
    pass
```

In [283]:
```
def f():
    pass
```

Το `pass` μπορούμε να το βάλουμε οπουδίποτε και δεν κάνει.. [τίποτα (https://en.wikipedia.org/wiki/NOP#Python)](https://en.wikipedia.org/wiki/NOP#Python):

In [284]:
```
print ('Hello')
pass
print ('World')
```

```
Hello
World
```

is

Έχουμε δει πολλές φορές τον τελεστή `is` . Το έχουμε χρησιμοποιήσει για να δούμε τι τύπος είναι μία μεταβλητή:

In [417]:
```
a = [1,2,3]
type(a) is list
```

Out[417]:
```
True
```

Η is κοιτάει αν δύο μεταβλητές είναι ίδιες. Δηλαδή είναι οι ίδιες μεταβλητές. Ή αλλιώς κοιτάει αν αναφέρονται στην ίδια θέση μνήμης. Αυτό είναι διαφορετικό από το ίσες: Δύο μεταβλητές είναι *ίσες* (==) αν έχουν την ίδια τιμή. Δύο μεταβλητές είναι *ίδιες* αν αναφέρονται στην ίδια θέση μνήμης. Παραδείγματα:

In [418]:
```
a = [1,2,3]
b = [1,2,3]
print (a==b)
print (a is b)
```

```
True
False
```

In [419]:
```
a = [1,2,3]
b = a
print (a==b)
print (a is b)
```

```
True
True
```

Όταν δύο μηταβλητές είναι ίδιες (και όχι απλά ίσες) τότε αν αλλάξεις την μία, αλλάζει και η άλλη:

In [420]:
```
a = [1,2,3]
b=a
b[0] = 100
print (a)
```

```
[100, 2, 3]
```

Ένα άλλο παράδειγμα:

In [422]:
```
a = [1,2,3,4]
b = [a,a,a]
print (b)
a[0]=100
print (b)
```

```
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
[[100, 2, 3, 4], [100, 2, 3, 4], [100, 2, 3, 4]]
```