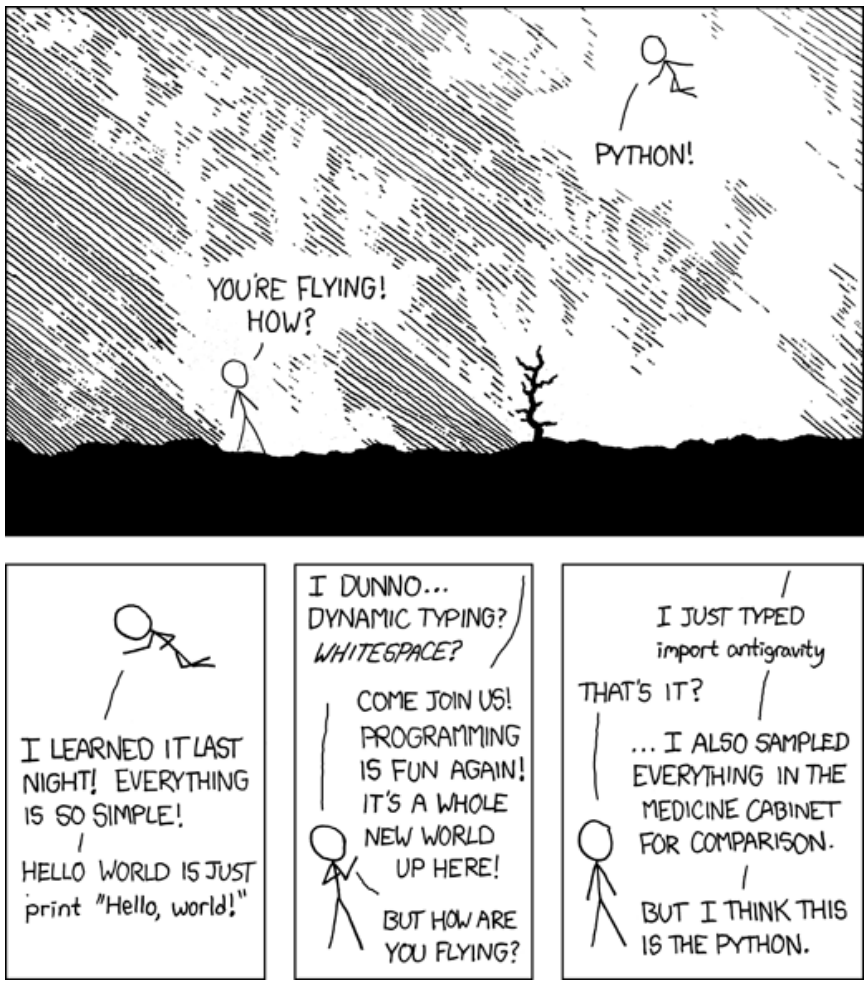
Python

Ljiljana Despalatović

Sveučilišni odjel za stručne studije
Sveučilište u Splitu

Uvod

```
import antigravity
```



Uvod

- Guido Von Rossum, 1991. g.
- Benevolent Dictator for Life
  http://www.theperlreview.com/Articles/v0i2/guido_van_rossum.pdf
- Monty Python's Flying Circus
- dinamički programski jezik
- success stories http://brochure.getpython.info/media/releases/prerelases/psf-python-brochure-vol-1-final-content-preview
- instalacija www.python.org

Svojstva

- dinamički jezik
- multiparadigmatski jezik
  - proceduralnan
  - objektno orijentiran jezik
  - funkcionalni jezik
- interaktivan
- portabilan
- open source
- strma krivulja učenja, lak za početnike

Organizacija i pokretanje programa

- Interaktivno zadavanje naredbi, naredbu po naredbu.

tty: /bin/bash

```
$ python
>>> a = 13
>>> b = "a"
>>> a*b
'aaaaaaaaaaaaa'
```

  - pogodno za testiranje dijelova kôda

Organizacija i pokretanje programa

- Pisanje skripti.

```
a = 13
b = 'a'
print (a * b)
```

tty: /bin/bash

```
$ python test01.py
aaaaaaaaaaaaa
```

Organizacija i pokretanje programa

- Pisanje skripti.

```
#!/usr/bin/python
a = 13
b = 'a'
print (a * b)
```

tty: /bin/bash

```
$ chmod 755 test01.py
$ ./test01.py
 aaaaaaaaaaaaa
```

- Korištenje IDE programa (PyDev plugin for Eclipse, Spyder, PyCharm, Komodo,...).

Leksička struktura Pythona

- Komentari počinju znakom #.
- Imena varijabli sadrže alfanumeričke znakove i znak podvlačenja (*underscore*), znamenka ne smije biti na prvom mjestu.
- Operatori: `+`, `-`, `~`, `*`, `**`, `/`, `<<`, `>>`, `&`, `|`, `^`, `and`, `or`, `not`, `in`, `not in`, `is`, `is not`, `<`, `>`, `!=`, `<>`, `==`, `<=`, `>=`
- http://www.tutorialspoint.com/python/python_basic_operators.htm
- Blokovi su određeni uvlačenjem retka. Preporuka 4 *spacea*.
- Pep8 *Style Guide for Python Code*
  https://www.python.org/dev/peps/pep-0008

Significant whitespace

C

```
for (i = 0; i < n; ++i)
{
    printf("i = %d\n", i);
    s += i;
}
```

C

```
for (i = 0; i < n; ++i){
    printf("i = %d\n", i);
    s += i;
}
```

C

```
for (i = 0; i < n; ++i)
    {
    printf("i = %d\n", i);
    s += i;
    }
```

C

```
for (i = 0; i < n; ++i)
  {
    printf("i = %d\n", i);
    s += i;
  }
```

Significant whitespace

- ne miješati <tab> i space (iako većina IDE programa konvertira <tab> u space)

python

```
for i in range(10):
    print "i = ", i
    s += i
```

python

```
import this
```

Tipovi

- numerički (`int`, `long`, `float`, `complex`)
- `None` (singleton object)
- `bool` (`True`, `False`)
- sekvence
  - stringovi (`str`)
  - liste (`list`)
  - n-torke (`tuple`)
  - virtualne sekvence (`xrange`)
- mapiranje (`dict`)
- skupovi (`set`)

I još tipova...

- files
- funkcije/metode
- moduli
- tipovi/klase
- iteratori/generatori

python

```
>>> import types
>>> dir(types)
```

Tipovi

Dvije su vrste tipova

- promjenjivi (mutabilni) - `list`, `dict`
- nepromjenjivi (nemutabilni) - `string`, `tuple`, numerički tipovi, skupovi

python

```
>>> a = 'foo'
>>> id(a)
140309840450144
>>> b = a
>>> id(b)
140309840450144
>>> a += 'dada'
>>> a
'foodada'
>>> id(a)
140309854849232
```

Tipovi

- Liste su mutabilne.

python

```
>>> l = [1,2,3,4]
>>> id(l)
140203169257664
>>> l.append("aaa")
>>> id(l)
140203169257664
```

- Unutar nepromjenjivog objekta možemo mijenjati promjenjive objekte

python

```
>>> a = (1,2,[])
>>> a[2].append(4)
>>> a
(1,2,[4])
```

Sekvence

Slice notacija

- `seq` je lista, string ili tuple
- `seq[low:high:step]`
- `seq[:], seq[::]`
- `seq[low:]`
- `seq[:high]`
- `seq[low:high]`
- `seq[::step]`
- `seq[low::step]`
- `seq[:high:step]`

Slice assignment

Samo za mutabilne sekvence.

python

```
>>> lst = [1,2,3,4,5,6]
>>> lst[2:3] = "hahaha"
>>> lst
[1, 2, 'h', 'a', 'h', 'a', 'h', 'a', 4, 5, 6]
>>> lst[2:3] = ["hahaha"]
>>> lst
[1, 2, 'hahaha', 'a', 'h', 'a', 'h', 'a', 4, 5, 6]
```

List (set, tuple) comprehension

```
new_list = [expression(i)for i in old_list if filter(i)]
```

python

```
>>> word_list = ['ovo', 'je', 'lista', 'rijeci']
>>> new_list = [word[0] for word in word_list]
>>> new_list
['o', 'j', 'l', 'r']

>>> new_list = [word[len(word)-1] for word in word_list]
>>> new_list
['o', 'e', 'a', 'i']

>>> word = 'lista'
>>> new_list = [i for i in range(len(word_list)) if word==word_list[i]]
>>> new_list
[2]
```

List (set, tuple) comprehension

comprehension.py

```
def power(n, k):
    return n**k

def myprint(i, x):
    print i , "->", x

k = 3
lst = [[power(n, k) for n in range(1, 11)] for k in range(10)]
[myprint(i, lst[i]) for i in range(len(lst))]
```

List (set, tuple) comprehension

primes.py

```
import math
m = 100000
n=100100

#list
primes = [x for x in xrange(m, n) if all(x%y for y in xrange(2, min(x, int(math.
    sqrt(n)))))]
print primes

#set
primes = {x for x in xrange(m, n) if all(x%y for y in xrange(2, min(x, int(math.
    sqrt(n)))))}
print primes

#generator
primes = (x for x in xrange(m, n) if all(x%y for y in xrange(2, min(x, int(math.
    sqrt(n))))))
print [p for p in primes]

#tuple
primes = tuple((x for x in xrange(m, n) if all(x%y for y in xrange(2, min(x, int(
    math.sqrt(n)))))))
print(primes)
```

Neke naredbe

- `pass`
- `print`
- `if`
- `while`
- `for`
- `break`
- `continue`
- `with`

with.py

```
with open("03_naredbe.tex") as f:
    data = f.read()

with open('output.txt', 'w') as f:
    f.write('Hi there!')
```

Naredbe

naredbe.py

```
number = 100

guess = int(input('Enter an integer : '))

while guess != number:
    if guess < number:
        print('broj je veci, pokusaj ponovo')
        guess = int(input('Enter an integer : '))

    elif guess > number:
        print('broj je manji, pokusaj ponovo')
        guess = int(input('Enter an integer : '))
print('bravo!')
```

Naredbe i izrazi

- Python strogo odvaja naredbe i izraze.
- Dodjeljivanje (*assigment*) je naredba, nije izraz.

Sve je objekt!

http://effbot.org/zone/python-objects.htm

- Može se pridružiti varijabli ili proslijediti funkciji kao argument.
  - Stringovi su objekti. Funkcije su objekti. Moduli su objekti. Tipovi su objekti.
- Svaki objekt ima svojstva:
  - identifikator `id(obj)`,
  - tip `type(obj)`,
  - vrijednost.
- Identifikator i tip su nepromjenjivi.
- Vrijednost objekta može biti promjenjiva (*mutable*) ili nepromjenjiva (*immutable*)
- Svaki objekt može imati:
  - nula ili više metoda,
  - nula ili više imena.

  Imena su dio prostora imena (*namespace*).

Imena

- Ime se može pridružiti bilo kojem objektu.

python

```
a = 10
b = "tralala"
c = {2:'a', 3:4}
d = [a,b,c]
```

python

```
def fibonaci(k):
    a, b = 1, 1
    i = 2
    while i <= k:
        a, b = b, a + b
        i += 1
    return a

f = fibonaci
print f(5)
```

And now for something completely different

- Umjesto varijabli Python ima imena i veze (*bindings*).

python

```
class First():
    a = "prva"

f = First()
print id(f), f.a
s = f
print id(s), s.a
```

python

```
>>> a = 3            # a je integer
>>> id(a)
33160248
>>> a = "neki string" # a je string
>>> id(a)
140203169416896
```

Namespace

- Svaki *package*, modul, klasa, funkcija i metoda imaju svoj *namespace*.
- Prilikom izvršavanja funkcije, metode, modula, package ili kreiranja klase stvara se *namespace*.
- Prilikom završetka izvršavanja funkcije ili metode, *namespace* nestaje, kao i lokalne varijable.

namespace.py

```
print "modul name ", __name__

n = 20

def fcja1():
    n = 15
    print n, locals()

def fcja2():
    print n, locals()
    n = 15

fcja1()
fcja2()
#UnboundLocalError: local variable 'n' referenced before assignment
```

namespace.py

```
def fcja2():
    global n
    print n, locals()
    n = 15
    print n, locals()
```

Funkcije

test-funkcije.py

```
def f(n, s, l):
    n = 2
    s += "novi"
    l.extend([5,6])
    print 'f:', n, s, l

def main():
    n = 1
    s = "test";
    l = [0,1,2,3]
    print 'main:', n, s, l
    f(n, s, l)
    print 'main: ', n, s, l

main()
```

Funkcije

call-by-value ili call-by-reference?

- nijedno od toga!

call-by-sharing (call-by-object, call-by-object-reference)

- argumenti su objekti i mogu se mijenjati samo ako su mutabilni
- prilikom izvršavanja funkcije objekt dobije drugo ime
- `http://eev.ee/blog/2012/05/23/python-faq-passing/`
- `http://www.python-course.eu/passing_arguments.php`

Funkcije

python

```
def funkcija(n):
    print id(n), n
    n+=2
    print id(n), n

k = 3
print id(k), k
funkcija(k)
print id(k), k
-----------------
21782824 3
21782824 3
21782776 5
21782824 3
```

python

```
def funkcija(n):
    print id(n), n
    n+=[2]
    print id(n), n

k = [3]
print id(k), k
funkcija(k)
print id(k), k
-----------------
140416810024328 [3]
140416810024328 [3]
140416810024328 [3, 2]
140416810024328 [3, 2]
```

Argumenti funkcije

- *required arguments*

python

```
>>> def fun(n, k):
...   return n**
...
>>> fun(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: fun() takes exactly 2 arguments (1 given)
```

- *keyword arguments*

python

```
>>> fun(k = 5, n = 2)
32
```

- *default arguments*

python

```
>>> def fun(n, k = 1):
...   return n**k
...
>>> fun(4)
4
```

Argumenti funkcije

- *variable-length argument*
  pakiranje argumenata

fun-var.py

```
def fun(n, *args):
    print "args are ", type(args)
    ret = []
    for i in args:
        ret.append(n**i)
    return ret

fun(2, 7, 8, 9]
-------------------
[128, 256, 512]
```

ali ne može ovako

python

```
lst = [7, 8, 9]
fun(2, lst)
```

Tome možemo doskočiti tako da raspakiramo argumente pri pozivu:

python

```
>>> fun(2, *lst)
```

Argumenti funkcije

- *variable-length argument*
  pakiranje *keyword* argumenata

fun-var.py

```
def fun2(n, **kwargs):
    print "kwargs are ", type(kwargs)
    print kwargs
    ret = []
    for i in kwargs.values():
        ret.append(n**i)
    return ret

print fun2(2, a = 2, b = 3)
```

opet ne može ovako

fun-var.py

```
dct = {'a':2, 'b':3}
print fun2(2, dct)
```

ali može ovako (raspakiravanje dictionaryja)

fun-var.py

```
dct = {'a':2, 'b':3}
print fun2(2, **dct)
```

Ugnježđene funkcije

ugnjezdjena.py

```
def ratio_below_avg(data):
    num_of_data = len(data)
    def avg(data):
        return sum(data)/float(num_of_data)
    avg_value = avg(data)
    return [el for el in data if el < avg_value]

lst = [5, 7, 4, 6, 3, 2]
print ratio_below_avg(lst)
```

Lambda funkcije

- Lambda izrazi mogu imati bilo koji broj argumenata, vraćaju jedan izraz.
- Ne mogu sadržavati naredbe niti više izraza.
- Lambda izraz proizvodi (neimenovani) funkcijski objekt.
- Mogu se pisati na mjestu izraza.

```
lambda [arg1 [,arg2,...,argn]] izraz
```

ugnjezdjena.py

```
def ratio_above_avg(data):
    avg_value = lambda data: sum(data)/float(len(data))
    return [el for el in data if el > avg_value(data)]

lst = [5, 7, 4, 6, 3, 2]
print ratio_above_avg(lst)
```

Lambda funkcije

Lambda funkcije mogu biti argumenti funkcije.

lambda.py

```
def suma(f, lst):
    return sum([f(n) for n in lst ])

square = lambda x : x*x
cube = lambda x : x**3

l = xrange(100000)
print(suma(square, l))
print(suma(cube, l))
```

Lambda izrazi mogu imati više argumenata.

lambda.py

```
def suma_pot(f, k, lst):
    return sum([f(n, k) for n in lst ])

l = xrange(100000)
pot = lambda x, i : x**i
print(suma_pot(pot, 3, l))
```

Lambda funkcija i funkcija map

- Najčešće se koriste u kombinaciji sa `map`, `filter` i `reduce` funckijama.
- `ret_seq = map(func, seq)` primjenjuje funkciju na sekvencu.

map.py

```
seq = [12, 13, 15, 18, 22, 26, 30, 30, 25, 22, 16, 13]

fahr_seq = map(lambda x : x * 9./5 + 32, seq)
print("fahrenheit", fahr_seq)

cels_seq = map(lambda x : (x - 32) * 5./9, fahr_seq)
print("celsius", cels_seq)
```

- `map` se može primjeniti na dvije sekvence jednake duljine

map.py

```
newlst = map(lambda x, y : (x, y), fahr_seq, cels_seq)
print(newlst)
```

Lambda funkcija i funkcija `filter`

- `ret_lst = filter(func, lst)` u novu listu stavlja elemente iz liste za koje `func` vraća `True`

filter.py

```
lst = [12, 13, 15, 18, 22, 26, 30, 30, 25, 22, 16, 13]
avg = sum(lst)/float(len(lst))

above_avg = filter(lambda x : x >= avg, lst)
print above_avg
```

Lambda funkcija i funkcija `reduce`

- `ret = reduce(func, seq)` primjenjuje funkciju `func` redom na po dva prva elementa. U svakom ponavljanju sekvenca se smanji za jedan element. Ponavljanje se ponavlja dok u sekvenci ne ostane samo jedan element. On je rezultat.

reduce.py

```
seq = (4, 6, 8, 2, 5)
max_el = reduce (lambda a, b : a if a > b else b, seq)
```

Generatori

- Generator je specijalni tip *iterable* objekta.
  - iteracija - dohvaćanje jednog po jednog elementa iz niza elemenata
  - *iterable* - objekt po kojem se može iterirati (ima `__iter__()` metodu)
  - iterator - objekt koji implementira iteraciju (mora imati `next()`) metodu
  - lista je primjer *iterablea* (sekvence, generatori i *mapping* su *iterabli*)

python

```
>>> lst = [1,2,3]
>>> it = iter(lst)
>>> it.next()
1
>>> it.next()
2
>>> it.next()
3
```

- Generatori nemaju *random access* i *slicing*. Elementima se pristupa redom.
- Elementi se kreiraju za vrijeme iteracije.

Generatori

- Generatori se kreiraju koristeći *generator expressions* ili pozivajući generator funkcije
- *Generator expressions* pišu se kao *list comprehensions* ali u okruglim zagradama

python

```
>>> lst = [3, 6, 7]
>>> gen = (x for x in lst)
>>> gen
<generator object <genexpr> at 0x7fbbd40339b0>
>>> gen.next()
1
>>> gen.next()
2
>>> gen.next()
3
```

Generatori

- Funkcije generatori vraćaju iteratore koristeći ključnu riječ `yield`.
- Iterator se koristi koristeći `next` metodu ili iteriranjem po generatoru.

generator.py

```
def power(n):
    """ yield power of n, create generator """
    k  = 0
    while(True):
        yield n**k
        k += 1

for i in power(2):
    print i
    if i > 10**50:break
```

- `yield` vraća vrijednost izraza, sačuva se vrijednost lokalnih varijabli i pozicija `yield` naredbe.
- Pri sljedećem pozivu, izvršavanje se nastavlja od linije nakon `yield` naredbe i sa lokalnim vrijednostima iz prethodnog poziva.

Moduli

- Svaka `*.py` datoteka je modul. Modul je *namespace*. Popis svih modula: `help('modules')`
- Veći programi su organizirani kao skupina modula.
- Više modula zajedno čini paket.
- Učitavanje modula naredbom `import`. Izvođenje naredbi modula obavlja se samo jednom.

python

```
>>> import test01
aaaaaaaaaaaaa
>>> import test01
>>>
```

- Nastaje *bytecode* tj. `*.pyc` datoteka.
- Eventualno ponovno učitavanje i izvođenje naredbi nekog (možda promijenjenog) modula izvodi se funkcijom `reload` iz modula `imp` `imp.reload(test01)`

Moduli

- Modul uvezen naredbom `import` postaje objekt.
- Popis atributa koje taj objekt ima mogu se dobiti sa `dir` ili `help('imemodula')`.

```
>>> dir(datetime)
['MAXYEAR', 'MINYEAR', '__doc__', '__file__', '__name__', '__package__', 'date
    ', 'datetime', 'datetime_CAPI', 'time', 'timedelta', 'tzinfo']
```

- Atributi koji počinju i završavaju sa dvostukim znakom podvlačenja su ugrađeni atributi.
- Popis atributa i njihovih tipova može se dobiti koristeći *read-only* rječnik `__dict__` (`datetime.__dict__`) sadrži kompletan prostor imena (*namespace*) modula.

Moduli

Lokacije gdje se traže moduli:

- u tekućem direktoriju,
- u PYTHONPATH shell varijabli,
- u *defaultnom* python direktoriju (na Linuxu obično `/usr/lib/python` ili `/usr/local/lib/python`).

Sve ih možemo vidjeti u varijabli `sys.path`.

Moduli

- Import dijelova prostora *namespacea*:
  - `from math import *` puni sve definicije u prostor imena
  - `from math import fabs`

Moduli

Varijabla `__name__` je postavljena na `__main__` kada se izvodi skripta ili pokreće interaktivni prompt.

```
if __name__ == '__main__':
    print "samostalno"
else:
    print "importano"
```