

Python, II deo

© Predrag Pejović, 

jedna knjiga . . .

How to Think Like a Computer Scientist Learning with Python

Allen Downey, Jeffrey Elkner, Chris Meyers

Green Tea Press, Wellesley, Massachusetts

...i jedan citat:

strana 90:

... One of the characteristics of algorithms is that they do not require any intelligence to carry out. They are mechanical processes in which each step follows from the last according to a simple set of rules.

In our opinion, it is embarrassing that humans spend so much time in school learning to execute algorithms that, quite literally, require no intelligence.

On the other hand, the process of designing algorithms is interesting, intellectually challenging, and a central part of what we call programming.

Python, dosta kalkulatora: programiranje!

- ▶ pokrenete IDLE
- ▶ File, New File (Ctrl + N)
- ▶ kucate program
- ▶ sintaksna provera: Run, Check Module (Alt + X)
- ▶ start programa: Run Module (F5)

Python, for petlja, continue

```
a = []
for i in range(100):
    if i % 4 != 0:
        a.append(i)
        continue
    print 'izbacen', i
print
print a
```

Python, for petlja, da dodamo i break

```
a = []
for i in range(100):
    if i % 4 != 0:
        a.append(i)
        continue
    print 'izbacen', i
    if (i+1) % 77 == 0:
        break

print
print a
```

Python, input

```
x = input('unesi neki podatak: ')
print type(x)
print x
```

probati: 1, 3.0, 'podatak'

Python, raw_input

```
x = raw_input('unesi neki podatak: ')
print type(x)
print x
```

probati: 1, 3.0, 'podatak'

Python, formatted print

calculator mode, again

```
print '{'  
print '{}'  
print '{}'.format('Pera')  
print '{}, {}'.format('Mika', 'Laza')  
print '{0}, {1}'.format('Mika', 'Laza')  
print '{1}, {0}'.format('Mika', 'Laza')  
print '{1}, {1}'.format('Mika', 'Laza')
```

Python, formatted print, brojevi

```
print 'p = {}, q = {}'.format(3, 4)
print 'p = {}, q = {}'.format(3.0, 4)
print 'p = {1}, q = {0}'.format(3, 4)
import math
print 'pi = {:.2f}'.format(math.pi)
print 'pi = {:.5f}'.format(math.pi)
print 'pi = {:10.5f}'.format(math.pi)
print 'n = {:10d}'.format(100)
print 'n = {:4d}'.format(100)
print 'n = {:3d}'.format(100)
print 'n = {:1d}'.format(100)
```

ima još mogućnosti, pogledajte Tutorial
postoji i formatiranje sa %, staro (next slide)
koristiti str.format() metod za nove programe!

Python, formatted print, stari sistem

```
print 'pi = %5.3f' % math.pi
print 'pi = %7.4f' % math.pi
n = 9
print 'polagalo je %d studenata' % n
print 'polagalo je %10d studenata' % n
```

vuče se po starim programima
izbegavati!

Python, stringovi, još ponešto

```
a = 'Petar'  
b = 'Marko'  
len(a); len(b)  
a + b  
a * 3  
3 * a  
(a + 2 * b) * 3  
a - b  
a == a  
a == b  
a < b  
b < a  
'1' < '2'  
'1' < 2
```

Python, nije sve u matematici (?)

```
import string as s
dir(s)
help(s)
s.ascii_letters
s.printable
s.uppercase
s.digits
s.octdigits
```

Python, string methods

```
del s

'Pera'.upper()
'Pera'.lower()
'Pera'.center(20)
'Pera'.isalpha()
'Pera6'.isalpha()

'kako da ne'.capitalize()
'kako da ne'.title()
'kako da ne'.split()
```

i tako dalje, manual for string methods, pogledajte pre nego što reinvent

Python, OŠ, 1

Vreme je za zadatak:

- ▶ ulazni podatak je broj, $n < 20$ (npr.)
- ▶ treba odštampati LEPU (formatiranu) tablicu množenja

Polako, sada imamo preča posla ...

Python, files, pisanje

```
f = open('proba.txt', 'w')
print f
type(f)
f
f.write('prvi red\n')
f.write('drugi red\n')
f.write('treći red\n')
f.close # metod, obavezne()
f.close()
type(f)
f
```

pogledajte fajl proba.txt

Python, files, čitanje

```
f = open('proba.txt', 'r')
print f
f
type(f)
a = f.read()
a
print a
f.close()
f
```

Python, files, čitanje po 10 bajtova

```
f = open('proba.txt', 'r')  
f.read(10)  
f.read(10)  
f.read(10)  
f.read(10)  
f.close()
```

Python, files, čitanje po redovima

```
f = open('proba.txt', 'r')  
f.readline()  
f.readline()  
f.readline()  
f.readline()  
f.close()
```

Python, OŠ, 2

Vreme je za zadatak:

- ▶ ulazni podatak je broj, $n < 20$ (npr.)
- ▶ treba formirati LEPU (formatiranu) tablicu množenja
- ▶ i zapisati je u .txt file

Polako, još uvek imamo preča posla ...

Python, OOP

OOP, Python's middle name

- ▶ korisnik definiše svoje tipove podataka, klase
- ▶ nad definisanim podacima definiše funkcije, tzv. metode, već ih videli na delu
- ▶ čisto i pregledno
- ▶ može, a ne mora
- ▶ šteta propustiti

Python, OOP, konvencije za dalje ...

- ▶ pokrenete IDLE, tu je **interpreter**
- ▶ otvorite **editor**: File, New Window, ...
- ▶ ono što kucate u interpreteru je **crveno**
- ▶ ono što kucate u editoru je **plavo**
- ▶ kucate “as is”, kreativnost za sada nije na ceni
- ▶ file neka se zove `vector.py`

Python, class

kucate u editoru, kada završite proverite sintaksu i izvršite,
to su Alt+X i F5

```
class Vector:  
    pass
```

Python, class

```
type(Vector)
a = Vector()
type(a)
print a
a.x = 1
a.y = 2.0
a.z = 3j
type(a.x)
type(a.y)
type(a.z)
print a.x, a.y, a.z
del a
type(a.x)
```


Python, class help

```
help(Vector)
```

```
class Vector:  
    """Vector(x, y, z)  
  
    klasa koja definise vektore  
    i operacije nad njima"""
```

note: nema više `pass`!

```
help(Vector)
```

Python, inicijalizacija, dodajemo na class, help string ostaje

```
class Vector:
    . . .
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

i = Vector(1, 0, 0)
```

Python, inicijalizacija, dodajemo na class, help string ostaje

inicijalizacija i se dodaje u class file, vector.py pazite na indent, inicijalizacija i class isto!
4 argumenta?

```
type(i)
print i
print i.x, i.y, i.z
```

```
x = Vector()
```

Python, inicijalizacija, default

```
class Vector:  
    . . .  
    def __init__(self, x = 0, y = 0, z = 0):  
        self.x = x  
        self.y = y  
        self.z = z
```

```
i = Vector(1, 0, 0)  
j = Vector(0, 1, 0)  
k = Vector(0, 0, 1)
```

```
x = Vector()  
print x.x, x.y, x.z  
print i.x, i.y, i.z  
help(Vector)
```

Python, string method, lepo prikazivanje instance

dodajete samo funkciju:

```
class Vector:
    . . .
    def __str__(self):
        return 'Vector(' + str(self.x) + ', ' + str(self.y) + ', ' + str(self.z) + ')'
```

```
help(Vector)
print i
print j
a = Vector(2, 2, 0)
print a
```

Python, operator overloading, sabiranje vektora

```
class Vector:  
    . . .  
    def __add__(self, other):  
        return Vector(self.x + other.x, self.y + \  
            other.y, self.z + other.z)
```

```
help(Vector)  
print i + j  
print i + j + k  
  
print Vector.__add__(i, j)  
  
print i.__add__(j)
```

Python, operator overloading, oduzimanje vektora

```
class Vector:  
    . . .  
    def __sub__(self, other):  
        return Vector(self.x - other.x, \  
            self.y - other.y, self.z - other.z)
```

```
help(Vector)  
print i - j  
print i - j - k
```

```
print Vector.__sub__(k, j)  
print k.__sub__(j)
```

Python, da zakomplikujemo malo ...

u interpreteru:

```
def sv(a, b):  
    return a + b
```

```
print sv(i, j)  
print sv(3, 4)  
sv  
type(sv)
```

```
del sv
```

```
sv = Vector.__add__  
sv  
type(sv)
```

```
print sv(i, j)  
print sv(3, 4)
```


Python, ope^t operator overloading, negativni vektor

```
class Vector:  
    . . .  
    def __neg__(self):  
        return Vector(-self.x, -self.y, -self.z)
```

```
help(Vector)  
print -i  
print -(-i)
```

```
print Vector.__neg__(k)  
print k.__neg__()
```

Python, štampanje kao metod

```
class Vector:  
    . . .  
    def pr(self):  
        print self
```

```
help(Vector)  
help(Vector.pr)  
i.pr  
i.pr()  
Vector.pr(i)  
k.__neg__().pr()
```

Python, absolutna vrednost

```
class Vector:
    . . .
    def absval(self):
        return math.sqrt(self.x ** 2 + \
            self.y ** 2 + self.z ** 2)

help(Vector)
help(Vector.absval)
i.absval
i.absval()
import math
i.absval()
Vector.absval(i)
a = Vector(2, 2, 0)
a.absval(); Vector.absval(a); a.absval() ** 2
a.x
```

Python, vektorski proizvod

```
class Vector:  
    . . .  
    def vecprod(self, other):  
        x = self.y * other.z - self.z * other.y  
        y = self.x * other.z - self.z * other.x  
        z = self.x * other.y - self.y * other.x  
        return Vector(x, y, z)
```

```
print Vector.vecprod(i, j)  
print Vector.vecprod(j, i)  
print i.vecprod(j)  
print j.vecprod(i)  
i.vecprod(j).pr()  
j.vecprod(i).pr()
```

Python, skalarni (dot) proizvod

```
class Vector:  
    . . .  
    def dotprod(a, b):  
        dp = a.x * b.x + a.y * b.y + a.z * b.z  
        return dp
```

```
print Vector.dotprod(i, j)  
print Vector.dotprod(j, i)  
a = Vector(2, 2, 0)  
print a.dotprod(j)  
print a.dotprod(k)  
a.dotprod(k).pr()  
k.dotprod(i).pr() # zašto ne radi?
```

Python, proizvod množenja skalarom

```
class Vector:  
    . . .  
    def scalprod(self, other):  
        return Vector(self.x * other, \  
            self.y * other, self.z * other)
```

```
print Vector.scalprod(i, 3)  
print Vector.scalprod(3, i)  
print k.scalprod(3)  
print 3.scalprod(k)  
j.scalprod(3).pr()  
3.scalprod(i).pr()  
a = 3  
a.scalprod(i).pr()
```

Python, overloaded *

```
class Vector:  
    . . .  
    def __mul__(self, other):  
        return Vector.dotprod(self, other)
```

```
print i * 3  
print i * i  
print i * j  
print i * k  
print i.__mul__(i)  
(i * i).pr()  
i * i  
i * j
```

Python, overloaded *

```
class Vector:  
    . . .  
    def __rmul__(self, other):  
        return Vector.scalprod(self, other)
```

```
print 3 * i  
print i * 3  
print i * j  
print i * i  
print i.__rmul__(3)  
i.__rmul__(3).pr()
```


Python, a sada poređenje, 1

```
class Vector:
    . . .
    def __cmp__(self, other):
        a = self.absval()
        b = other.absval()
        if a > b:
            return 1
        elif a < b:
            return -1
        else:
            return 0
```

Python, a sada poređenje, 2

```
a = Vector(2, 2, 2)
a > i
a == i
a < i
i > i
i == i
i != i
i < i
i == j
i == k
```

I ovo je kraj priče o vektorima, nema više red i blue!

Python, class and object variables

```
class Brojac:  
    broj = 0  
    def __init__(self):  
        Brojac.broj += 1  
        print 'sada ih ima', Brojac.broj  
    def __del__(self):  
        Brojac.broj -= 1  
        print 'sada ih ima', Brojac.broj
```

Python, class and object variables, primena

```
a = Brojac()
b = Brojac()
c = Brojac()
d = Brojac()
del c
a = Brojac()      # pise, pa brise
del a
del d
del b
```

Python, a sada nešto sasvim različito, map

```
x = range(11)
print x
def sq(a):
    return a ** 2

y = map(sq, x)
print y
```

Python, a sada nešto neočekivano

```
help([])  
help(())  
help({})  
a = (1, 2, 3, 4, 5)  
a[0]  
a[3]  
a[2] = 5
```

Python, dictionary

```
t = {'peja': 313, 'laza': 312, 'dule': 311}
print t
print t['peja']
t['peja'] = 314
print t
del t['peja']
print t
'dule' in t
'peja' in t
t.keys()
t['peja'] = 313
print t
```

Python, sets

```
a = 'abracadabra'  
b = 'alacazam'  
x = set(a)  
print x  
y = set(b)  
print y  
x - y  
y - x  
x | y  
x & y  
x ^ y
```


Python, exceptions, 0

```
def reciproc(x):  
    try:  
        return 1 / float(x)  
    except:  
        print 'Houston, we have a problem!'
```

Python, exceptions, 1

```
def reciproc(x):  
    try:  
        return 1 / x  
    except ZeroDivisionError:  
        print 'delis nulom!'
```

```
reciproc(3)  
reciproc(3.)  
reciproc(0)  
reciproc(0.)  
reciproc('a ovo?')
```

Python, exceptions, 2

```
def reciproc(x):  
    try:  
        return 1 / float(x)  
    except ZeroDivisionError:  
        print 'delis nulom!'  
    except ValueError:  
        print 'nije dobar broj'
```

```
reciproc(3)  
reciproc(3.)  
reciproc(0)  
reciproc(0.)  
reciproc('a ovo?')  
reciproc(1+0j)
```

Python, exceptions, 3

```
def reciproc(x):  
    try:  
        return 1 / float(x)  
    except ZeroDivisionError:  
        print 'delis nulom!'  
    except (ValueError, TypeError):  
        print 'nije dobar broj'
```

```
reciproc(3)  
reciproc(3.)  
reciproc(0)  
reciproc(0.)  
reciproc('a ovo?')  
reciproc(1+0j)
```

Python, exceptions, 4

- ▶ “built-in” exceptions?
- ▶ <http://docs.python.org/2/library/exceptions.html>
- ▶ ovo je samo početak ...
- ▶ možete sami da definišete exception ...
- ▶ ili da raise exception ...
- ▶ ako vas zanima
<http://docs.python.org/2/tutorial/errors.html>

Python, start programa

```
python ime_programa.py
```

ili

```
./ime_programa.py
```

akko je PRVA linija programa:

```
#! /usr/bin/python
```

i ako je urađeno

```
chmod +x ime_programa.py
```

čega se sećate, naravno; da izbegnete `./` dopunite `PATH`

Python, encoding, 1

```
#!/usr/bin/python
```

```
print 'ja se ne zovem Čedomir Š. Petrović'
```

```
print 'ja se zovem Предраг Пејовић'
```

Python, encoding, 2, mora drugi red!

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

print 'ja se ne zovem Čedomir Š. Petrović'
print 'ja se zovem Предраг Пејовић'
```


Python, rendering

```
print '\n\n\n'
```

```
print r'\n\n\n'
```

```
print '\\n\\n\\n'
```

Python, budoucnost

```
from __future__ import division
```

```
i
```

```
from __future__ import unicode_literals
```

Python, nije kraj, ovo je početak!

- ▶ ima još mnogo toga, ovo je samo osnova
- ▶ pre svega biblioteke, moduli ...
- ▶ mnogo toga je gotovo, vaša je primena!
- ▶ razlika 2 i 3
- ▶ od sledeće godine 3?
- ▶ ostaju numpy, matplotlib, ipython, pylab, sympy
- ▶ a sada: tablica množenja