

Programski jezik Python

Prof. dr Igor Dejanović (igord at uns.ac.rs)

Kreirano 2022-10-17 Mon 12:17, pritisni ESC za mapu, m za meni, Ctrl+Shift+F za pretragu

Sadržaj

1. Kratak pregled Python-a
2. Leksičke konvencije i sintaksa
3. Tipovi i objekti
4. Operatori i izrazi (TODO)
5. Struktura programa i kontrola toka
6. Funkcije i funkcionalno programiranje
7. Klase i objektno-orientisano programiranje
8. Moduli i paketi
9. Alati i okruženja
10. Pakovanje i distribucija aplikacija
11. Reference

Kratak pre-gled Python-a

Python

- Razvoj započet 1989 u Holandiji kao hobij projekat Gvida Van Rosuma.
Danas jedan od najpopularnijih jezika.
- Interpretiran dinamički jezik visokog nivoa.
- Više paradigmi: imperativno, proceduralno, objektno, funkcionalno...
- Akcenat na efikasnosti programera i čitkosti koda.
- Cross-platform
- Sveobuhvatna i veoma razvijena standardna biblioteka.
- Jezik ima više implementacija.
- Koristi se za desktop i web aplikacije, mobilne aplikacije, administrativne skripte, upravljačke skripte, u ugrađenim sistemima...
- Upotrebljava se u firmama širom sveta: Google, Disney, Dropbox, Industrial Light & Magic...

Zen of Python

```
>>> import this
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.

Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.

Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Implementacije Python-a

- CPython
- PyPy
- Jython
- Iron Python
- pyjs
- ...

Python konzola

- Pokreće se pozivom Python interpretera bez parametara.

```
$ python
Python 3.4.1 (default, May 19 2014, 17:23:49)
[GCC 4.9.0 20140507 (prerelease) ] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>> 23423432 ** 34
36992864259838982449973046100677855738848171498810334668814347544
72272789093305975943853887303038138168893642829742372685558166974
99603306904629343154070096117901999442230973428786292674630445031
96766248000024671594323356975802355740978014370737946624
>>>
```

Jednostavan program na Python-u

```
principal = 1000      # Početni iznos
rate = 0.05            # Kamatna stopa
numyears = 5
year = 1
while year <= numyears:
    principal = principal * (1 + rate)
    print(year, principal)
    year += 1
```

- Varijable - imena/reference za objekte.
- Objekti imaju nepromenljiv tip. Varijable mogu da menjaju objekat koji referenciraju.

print i formiranje izlaza

- Slično **printf** funkciji u C-u.
- Upotrebom string interpolacije (operator **%**)

```
print("%3d %0.2f" % (year, principal))
```

ili upotrebom **format** funkcije

```
print(format(year, "3d"), format(principal, "0.2f"))

# ili format funkcijom nad stringom
print("{0:3d} {1:0.2f}".format(year, principal))
```

ili upotrebom f-stringova

```
print(f"{year:3d} {principal:0.2f}")
```

Uslovi

```
if a < b:  
    print("Computer says Yes")  
else:  
    print("Computer says No")
```

```
if product == "game" and type == "pirate memory" \  
    and not (age < 4 or age > 8):  
    print("I'll take it!")
```

```
if suffix == ".htm":  
    content = "text/html"  
elif suffix == ".jpg":  
    content = "image/jpeg"  
elif suffix == ".png":  
    content = "image/png"  
else:  
    raise RuntimeError(  
        "Unknown content type")
```

Fajl u|az/iz|az

```
f = open("foo.txt")
line = f.readline()
while line:
    print(line, end=' ')
    line = f.readline()
f.close()
```

Isti program u kraćoj formi:

```
with open("foo.txt") as f:
    for line in f:
        print(line, end='')
```

Pisanje u fajl:

```
f = open("out", "w") # Otvaranje za pisanje - "w"
while year <= numyears:
    principal = principal * (1 + rate)
    print("%3d %0.2f" % (year, principal), file=f)
    # Alternativno f.write("%3d %0.2f\n" % (year, principal))
    year += 1
f.close()
```

Stringovi

```
a = "Hello World"
b = 'Python is groovy'
c = """Computer says 'No'"""
print('Content-type: text/html
<h1>Hello World</h1>
Click <a href="http://www.python.org">here</a>.
')
b = a[4] # b = 'o'

c = a[:5] # c = "Hello"
d = a[6:] # d = "World"
e = a[3:8] # e = "lo Wo"

g = a + " This is a test"

x = "37"
y = "42"
z = x + y # z = "3742" (konkatanacija stringova)
z = int(x) + int(y) # z = 79 (Integer +)
```

Liste

- Liste su sekvence proizvoljnih objekata (referenci)

```
names = [ "Dave", "Mark", "Ann", "Phil" ]
a = names[2]                                # Vraća treći objekat iz liste - "Ann"
names[0] = "Jeff"                            # Menja prvi objekat-referencu na "Jeff"
names.append("Paula")                         # Dodaje "Paula" na kraj liste
names.insert(2, "Thomas")                      # Ubacuje "Thomas" na lokaciju 2

b = names[0:2]                               # Vraća [ "Jeff", "Mark" ]
c = names[2:]                                 # Vraća [ "Thomas", "Ann", "Phil", "Paula" ]
names[1] = 'Jeff'                             # Menja drugi element sa 'Jeff',
names[0:2] = ['Dave', 'Mark', 'Jeff']          # Menja prva dva elementa sa liste
                                                # sa listom na desnoj strani
a = [1, 2, 3] + [4, 5]                        # Rezultat je [1, 2, 3, 4, 5]

names = []                                     # Prazna lista
names = list()                                 # Prazna lista

a = [1, "Dave", 3.14, ["Mark", 7, 9, [100, 101]], 10]
a[1]                                         # "Dave"
a[3][2]                                       # 9
a[3][3][1]                                     # 101
```

List comprehensions

```
import sys # Učitavanje sys modula
if len(sys.argv) != 2 # Proveri broj argumenta
    print("Please supply a filename")
    raise SystemExit(1)

f = open(sys.argv[1]) # Ime fajla je dato kao parametar
lines = f.readlines() # Pročitaj sve linije u listu
f.close()

# Konvertuje sve vrednosti u linijama teksta u float
fvalues = [float(line) for line in lines]

# Pronađi min i max vrednosti
print("The minimum value is ", min(fvalues))
print("The maximum value is ", max(fvalues))
```

N-torke (Tuples)

- Nepromenjiva struktura - *immutable*

```
stock = ('GOOG', 100, 490.10)
address = ('www.python.org', 80)
person = (first_name, last_name, phone)

# ići samo
stock = 'GOOG', 100, 490.10
address = 'www.python.org', 80
person = first_name, last_name, phone

# Načini navođenja
a = ()           # 0-tuple (prazan tuple)
b = (item,)      # 1-tuple (obratiti pažnju na zarez)
c = item,        # 1-tuple (obratiti pažnju na zarez)

# "raspakivanje"
name, shares, price = stock
host, port = address
first_name, last_name, phone = person
```

Primer upotrebe n-torki i lista

```
# File containing lines of the form "name,shares,price"
filename = "portfolio.csv"
portfolio = []
with open(filename) as f:
    for line in f:
        fields = line.split(",")
        name = fields[0]
        shares = int(fields[1])
        price = float(fields[2])
        stock = (name, shares, price)
        portfolio.append(stock)

>>> portfolio[0]
('GOOG', 100, 490.10)
>>> portfolio[1]
('MSFT', 50, 54.23)
>>> portfolio[1][1]
50
>>> portfolio[1][2]
54.23
>>>
```

```
total = 0.0
for name, shares, price in portfolio:
    total += shares * price
```

Skupovi (*sets*)

Neuređena kolekcija objekata.

```
s = set([3, 5, 9, 10])          # Kreira skup brojeva
s = {3, 5, 9, 10}               # Alternativno
t = set("Hello")                # Kreira skup jedinstvenih karaktera
>>> t
set(['H', 'e', 'l', 'l', 'o'])
```

Operacije nad skupovima:

```
a = t | s      # Unija skupova t i s
b = t & s      # Presek skupova t i s
c = t - s      # Razlika skupova t i s
d = t ^ s      # Simetrična razlika skupova t i s
# (elementi koji pripadaju ili skupu t ili
# skupu s ali ne i preseku)
```

Dodavanje i uklanjanje elemenata.

```
t.add('x')           # Dodavanje jednog elementa u t
s.update([10, 37, 42]) # Dodavanje više elemenata u s
t.remove('H')         # Uklanjanje elementa
```

Rečnici (*dictionaries*)

- Asocijativni niz objekata indeksiranih ključevima.
- Ključ može biti bilo koji nepromenjivi objekat (*immutable*).

```
# Dva načina kreiranja praznog rečnika
stock = {}
stock = dict()

# Kreiranje rečnika sa podacima
stock = {
    "name" : "GOOG",
    "shares" : 100,
    "price" : 490.10
}

# Upotreba
name = stock["name"]
value = stock["shares"] * stock["price"]

# Upis vrednosti
stock["shares"] = 75
stock["date"] = "June 7, 2007"
```

```

# Mogu se koristiti za brzo pronađenje podataka
prices = {
    "GOOG": 490.10,
    "AAPL": 123.50,
    "IBM": 91.50,
    "MSFT": 52.13
}

# Default vrednosti
if "SCOX" in prices:
    p = prices["SCOX"]
else:
    p = 0.0

# ili kada
p = prices.get("SCOX", 0.0)

# Lista ključeva
symbs = list(prices) # symbs = [ "AAPL", "MSFT", "IBM", "GOOG" ]

# ili
symbs = prices.keys()

```

Iteracija i petlje

```
for n in [1,2,3,4,5,6,7,8,9] :  
    print("2 na stepen %d je %d" % (n, 2**n))  
  
for n in range(1,10) :  
    print("2 na stepen %d je %d" % (n, 2**n))  
  
a = range(5)          # a = [0, 1, 2, 3, 4]  
b = range(1, 8)       # b = [1, 2, 3, 4, 5, 6, 7]  
c = range(0, 14, 3)   # c = [0, 3, 6, 9, 12]  
d = range(8, 1, -1)   # d = [8, 7, 6, 5, 4, 3, 2]  
  
for i in range(100_000_000) :      # i = 0, 1, 2, ..., 999999999 ali "lenje"  
    statements
```

- Iterator protokol
- Stringovi su sekvence - podržavaju iterator protokol

```
a = "Hello World"  
# Štampa pojedinačna slova stringa a  
for c in a:  
    print(c)
```

- Liste takođe.

```
b = ["Dave", "Mark", "Ann", "Phil"]  
# Štampa članove liste b  
for name in b:  
    print(name)
```

- I mape

```
c = { 'GOOG' : 490.10, 'IBM' : 91.50, 'AAPL' : 123.15 }
# Štampa sve članove rečnika c
for key in c:
    print(key, c[key])
```

- I fajlovi

```
# Štampa sve linije fajla foo.txt
f = open("foo.txt")
for line in f:
    print(line)
```

- Proizvoljni objekti mogu da podrže iterator protokol.
- Kreiranje iteratora upotrebom funkcija - generatori (videti u nastavku)

Funkcije

- Kreiraju se ključnom rečju **def**
- Ukoliko funkcija nema povratnu vrednost implicitno vraća **None**

```
def fibonacci(n):
    """Returns element of fibonacci array at given position."""
    if n < 0:
        raise ValueError("Index must be >=0.")
    if n < 2:
        return 1
    return fibonacci(n-2) + fibonacci(n-1)
```

```
def f(a, b=5):
    return a + 2 * * b
>>> f(3)
35
>>> f(3, 5)
35
>>> f(3, 6)
67
```

Parametri i vrednosti parametara se mogu upariti po poziciji ili po nazivu:

```
def f(a, b=5):
    return a + 2 ** b

>>> f(b=3, a=1)
9
```

Promenjivi broj parametara, po poziciji i po imenu:

```
def f(*args, **kwargs):
    for a in args:
        print(a)
    for k, v in kwargs.items():
        print(k, v)

>>> f(34, 56, 67, b=12, c=89, foo="bar")
34
56
67
b 12
foo bar
c 89
```

- Sve reference unutar funkcije su unutar opsega funkcije (scope).
- Ako treba da referenciramo globalnu varijablu deklarišemo je sa ključnom rečju **global**.

```
count = 0
...
def foo1():
    count = 1      # Kreiranje lokalne varijable count
...
def foo2():
    global count
    count += 1   # Izmene globalne varijable count
```

Generatori

Umesto jedne vrednosti funkcija može generisati sekvencu vrednosti.

```
def countdown(n):
    print("Counting down!")
    while n > 0:
        yield n # Generisanje vrednosti (n)
        n -= 1
```

- Poziv funkcije vraća tzv. *generator objekat*.
- Ključna reč **yield** označava povratak jedne vrednosti sekvence.

Generatori - upotreba

```
>>> c = countdown(5)
>>> c.__next__()
Counting down!
5
>>> c.__next__()
4
>>> c.__next__()
3
```

```
>>> for i in countdown(5) :
...     print(i, end=' ')
Counting down!
5 4 3 2 1
```

Svaki objekat (klasa) koji implementira generator protokol može da se koristi kao generator.

Korutine (*coroutines*)

- Koncept obrnut generatorima.
- Funkcije koje mogu spolja da prime sekvencu vrednosti u toku izvršavanja.

```
def print_matches(matchtext):
    print("Tražim", matchtext)
    while True:
        line = (yield) # Preuzmi liniju teksta spolja
        if matchtext in line:
            print(line)
        >>> matcher = print_matches("python")
        >>> matcher.__next__() # Postavlja se na prvi (yield)
Tražim python
>>> matcher.send("Hello World")
>>> matcher.send("Python is cool")
python is cool
>>> matcher.send("Wow!")
>>> matcher.close() # Na kraju je potrebno zatvoriti korutinu
```

Omogućavaju implementaciju *producer-consumer* ili *pipe* obrasca bez upotrebe niti i višenitnog programiranja.

Objekti i klase

- Sve vrijednosti su objekti.
- Objekat se sastoji od internih podataka i metoda koje operišu nad njima.
- Metode i atributi objekta se mogu izlistati ugrađenom funkcijom `dir`.

Stringovi su objekti.

```
>>> dir("foo")
['__add__', '__class__', '__contains__', '__delattr__',
... '__len__', '__lt__', '__mod__', '__mul__',
... 'capitalize', 'center', 'count', 'decode', 'encode',
... 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
```

Brojevi su objekti.

```
>>> dir(2)
['__abs__', '__add__', '__and__', '__class__', '__cmp__',
... '__pow__', '__radd__', '__rand__', '__rdiv__', '__coerce__',
... 'bit_length', 'conjugate', 'denominator', 'imag', 'numerator', 'real']
```

Funkcije su objekti

```
>>> dir(fibonacci)
['__call__', '__class__', '__closure__', '__code__',
'__dict__', 'func_doc', 'func_globals', 'func_name']
```

Liste su objekti.

```
>>> items = [37, 42]
>>> dir(items)
['__add__', '__class__', '__contains__', '__delattr__',
'__append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
```

Objekti i klase - specijalne metode

- Specijalne metode su oblika xxx
- Ove metode se koriste za implementaciju npr. operatora (npr. `+`, `-`, `*`, `in`...)
- Ove metode se koriste za implementaciju npr. `len`).

```
>>> items = [37, 42]
>>> items.__add__([73, 101])
[37, 42, 73, 101]
# je ekvivalentno sa
>>> items + [73, 101]
```

Klasa može da proizvoljno redefiniše specijalne metode.

Objekti i klase - konstruktor

Specijalna metoda `__init__` predstavlja konstruktor.

Primer: implementacija steka

```
class Stack(object):
    def __init__(self):
        self.stack = []
    def push(self, object):
        self.stack.append(object)
    def pop(self):
        return self.stack.pop()
    def length(self):
        return len(self.stack)

s = Stack()
s.push("Dave")
s.push(42)
s.push([3, 4, 5])
x = s.pop()
y = s.pop()
del s
```

- Svaka metoda prima objekat kao eksplicitan prvi parametar.
- Po konvenciji parametar nazivamo `self`.

Pošto je stek vrlo sličan Python listi možemo direktno naslediti ugrađenu listu.

```
class Stack(list): # Nasleđujemo listu
    # Dodajemo push metodu da bi implementirali
    # stack interfejs
    # Napomena: liste već imaju pop() metodu.
    def push(self, object):
        self.append(object)
```

Ovako kreirana klasa ima sve osobine liste.

```
>>> s = Stack()
>>> s.push(2)
>>> s.push(3)
>>> s.push(4)
>>> s
[2, 3, 4]
>>> s[:2]
[2,
```

Objekti i klase - vrste metoda

Klasa može da definiše različite vrste metoda.

```
class EventHandler(object):
    @staticmethod
    def dispatcherThread():
        while (1):
            # Wait for requests
            ...
            EventHandler.dispatcherThread() # Poziv static metode kao da je funkcija
```

@staticmethod je dekorator. Više u nastavku.

Izuzeci

Ukoliko dođe do graške u programu javlja se izuzetak

Ispis može biti poput ovoga:

```
Traceback (most recent call last):
  File "foo.py", line 12, in <module>
    IOError: [Errno 2] No such file or directory: 'file.txt'
```

Izuzeci se mogu uhvatiti i obraditi.

```
f = open("file.txt", "r")
try:
    ...
    obrada fajla
except Exception as e:
    ...
    obrada izuzetka
finally:
    # Ovaj blok se uvek izvršava na kraju
    # bez obzira šta da se desi
    f.close()
```

Izuzeci se programski izazivaju na mestu detektovanja nevalidnog stanja sa:

```
raise RuntimeError("Computer says no")
```

Izuzeći - konteksti

- Upotreba nekog resursa uvek zahteva njegovo oslobođanje (zatvaranje) po završetku upotrebe.
- Ovo oslobođanje može biti teže izvodljivo u kontekstu izuzetaka.
- Zato je uvedena ključna reč `with`.

```
import threading  
message_lock = threading.Lock()  
...  
with message_lock:  
    messages.add(newmessage)
```

- Izlaskom iz `with` bloka, bilo regularno ili zbog izuzetka biće automatski obavljeno oslobođanje resursa.
- Objekti koji mogu da se navedu u iskazu `with` implementiraju određeni kontekst protokol (dve specijalne metode: `__enter__` i `__exit__`)

Moduli

- Veće programe je pozeljno razbiti u više fajlova/modula.
- Python omogućava import-ovanje definicija iz drugih fajlova/modula.
- Python moduli su fajlovi sa ekstenzijom `.py`

```
# file : div.py
def divide(a,b):
    q = a/b
    r = a - q*b
    return (q,r)
```

```
import div
a, b = div.divide(2305, 29)
```

```
import div as foo
a, b = foo.divide(2305, 29)
```

```
from div import divide
a, b = divide(2305, 29)
```

```
from div import *
```

I moduli su oggetti

```
>>> import string
>>> dir(string)
['__builtins__', '__doc__', '__file__', '__name__', '__idmap__',
 '__imapL', '__lower__', '__swapcase__', '__upper__', 'atof', 'atof_error',
 'atoi', 'atoi_error', 'atol', 'atol_error', 'capitalize',
 'capwords', 'center', 'count', 'digits', 'expandtabs', 'find',
 ...]
```

Leksičke konvencije i sintaksa

Struktura linije i indentacija

```
a = math.cos(3 * (x - n)) + \n    math.sin(3 * (y - n))
```

```
if a:  
    statement1      # Ispravna indentacija  
    statement2  
else:  
    statement3  
    statement4      # Neispravna indentacija
```

```
if a: statement1  
else: statement2  
  
if a:  
    pass  
else:  
    statements
```

- Nije propisana širina uvlačenja ali mora biti konzistentna.
- Preporučeno je 4 **space** karaktera za uvlačenje.
- Preporuka je da se koriste **space** karakteri umesto **tab**.

String literali

- Navode se unutar jednostruktih ili dvostruktih znakova navoda ili trostruktih za višelinjske stringove.
- Unutar stringova karakter \ (backslash) služi da definiše tzv. *escape* sekvencu odnosno da omogući navođenje specijalnih znakova.
- Moguće je navesti i unicode kod sa prefiksom \u
- Za detalje videti reference.

Kontejneri

- Vrednosti koje se navode unutar zagrada `[...], (...), {...}` predstavljaju kolekciju objekata koja se nalazi unutar liste, n-torke ili rečnika.

```
a = [ 1, 3.4, 'hello' ]
b = ( 10, 20, 30 )
c = { 'a': 3, 'b': 42 }
```

- Elementi kontejnera se mogu navoditi u više linija bez upotrebe znaka za nastavak linije `(\)`
- Takođe, na kraju liste može da postoji `,`, i to je sintaksno dozvoljeno.

```
a = [ 1,
      3.4,
      'hello',
    ]
```

Tipovi i objekti

Terminologija

- Svi podaci Python programa su objekti.
- Objekti imaju identitet, tip i vrednost.

```
>>> a = 42
>>> id(a)
140649856584416
>>> id(42)
140649856584416
>>> type(a)
<class 'int'>
>>> b = a
>>> id(b)
140649856584416
>>> type(b)
<class 'int'>
>>>
```

- Jednom kreiran, identitet i tip objekta su nepromenjivi.
- Ukoliko je vrednost objekta nepromenjiva kažemo da je objekat nepromenjiv (*immutable*).

- Objekti koji sadrže reference na druge objekte se nazivaju kontejneri ili kolekcije.
- Objekte karakterišu atributi i metode.
- Atributi su podaci pridruženi objektima.
- Metode su funkcije koje vrše određene operacije nad objektom.

```
a = 3 + 4j      # Kreiranje kompleksnog broja
x = a.real      # Realni deo (atribut)
b = [1, 2, 3]    # Kreiranje liste
b.append(7)     # Dodavanje novog elementa upotrebom append metode
```

Identitet i tip objekta

```
# Poređenje dva objekta
def compare(a,b):
    if a is b:
        # a i b su isti objekat
    ...
    if a == b:
        # a i b imaju istu vrednost
    ...
    if type(a) is type(b):
        # a i b su istog tipa
    ...

```

```
if type(s) is list:
    s.append(item)
if type(d) is dict:
    d.update(t)
```

```
if isinstance(s,list):
    s.append(item)
if isinstance(d,dict):
    d.update(t)
```

Duck Typing

- Metode objekta definišu njegovo ponašanje. Polimorfizam se realizuje različitim ponašanjem pri pozivu metoda istog imena. Nije bazirano na tipu i nasleđivanju.

If it walks like a duck and quacks like a duck, it must be a duck.

```
def sum(a, b):  
    return a + b      # a i b podržavaju + operaciju  
  
>>> sum(2, 5)  
7  
>>> sum(2.5, 6.7)  
9.2  
>>> sum(True, False)  
1  
>>> sum("Hello ", "World!")  
Hello world!
```

Protokoli

- Koncept tesno povezan sa *duck typing*.
- Predstavlja određeno ponašanje objekta (skup metoda, atributa i njihove semantike).
- Ako kažemo da objekat podržava neki protokol znamo šta možemo da očekujemo od njega i u kom kontekstu možemo da ga koristimo bez obzira kog je tipa.
- Na primer, ako je objekat sekvenca (podržava protokol sekvence) tada znamo da možemo da koristimo isecanje (*slice*), iteraciju itd.
- U drugim jezicima se realizuje sa interfejsima (*interfaces*) ili osobinama (*traits*).

Reference i kopije

```
>>> a = [1, 2, 3, 4]          # b je referencia na listu a
>>> b = a
>>> b is a
True
>>> b[2] = -100              # Promena elementa u b
>>> a
>>> [1, 2, -100, 4]          # Element je promenjen u a jer je to
>>>                                # isti objekat
```

Plitko i duboko kopiranje

```
>>> a = [ 1, 2, [3, 4] ]
>>> b = list(a)
>>> b is a
False
>>> b.append(100)
>>> b
[1, 2, [3, 4], 100]
>>> a
[1, 2, [3, 4]]
>>> b[2][0] = -100
>>> b
[1, 2, [-100, 4], 100]
>>> a
[1, 2, [-100, 4]]
```

```
>>> import copy
>>> a = [1, 2, [3, 4]]
>>> b = copy.deepcopy(a)
>>> b[2][0] = -100
>>> b
[1, 2, [-100, 4]]
>>> a
[1, 2, [3, 4]]
```

First-Class Objects

- Svi objekti u Python-u su "prvog reda" (*first-class*)
- Ovo znači da svi objekti koji se mogu imenovati (referencirati) imaju isti status.

```
items = {  
    'number' : 42  
    'text' : "Hello World"  
}  
  
>>> items["func"] (-45)      # Poziva abs (-45)  
45  
>>> items["mod"].sqrt(4)    # Poziva math.sqrt (4)  
2.0  
>>> try:  
    ...  
    x = int("a lot")  
    # Isto kao except ValueError as e:  
except items["error"] as e:  
    ...  
    print("Couldn't convert")  
    ...  
Couldn't convert  
>>> items["append"] (100)    # nums.append (100)  
>>> nums  
[1, 2, 3, 4, 100]
```

Ugrađeni tipovi za podatke

- None: `NoneType`
- Brojevi: `int`, `long`, `float`, `complex`, `bool`
- Sekvence: `str`, `unicode`, `list`, `tuple`
- Mape: `dict`
- Skupovi: `set` (*mutable*), `frozenset` (*immutable*)

Sekvence

- Uređena kolekcija objekata indeksirana nenegativnim rednim brojem.
- Stringovi - nepromenjiva sekvenca karaktera.
- n-torka(**tuple**) - nepromenjiva sekvenca proizvoljnih objekata.
- Sve sekvence podržavaju **slicing** i iteracije.

Operacije nad sekvencama

```
s[i]                      # Indeksni pristup
s[i:j]                    # Isjecanje (slicing)
s[i:j:korak]              # Prošireno isjecanje
len(s)                     # broj elemenata sekvene
min(s), max(s)            # minimalna/maksimalna vrednost u sekvenci
sum(s, [initial])         # sumiranje sekvene
all(s)                     # da li su svi elementi sekvenice True
any(s)                     # da li je bilo koji element u listi True
for a in s:                # iteracija
    ...
    
```

Operacije nad promenjivim sekvencama

```
s[i] = v  
s[i:j] = t  
s[i:j:korak] = t  
s[i]  
s[i:j]  
s[i:j:korak]
```

Liste

- Liste su sekvence.
- Svaka sekvencia se može konvertovati u listu sa `list(s)`.
- Definišu sledeće metode:
 - `s.append(x)` - dodavanje na kraj
 - `s.extend(t)` - proširenje sa listom `t`
 - `s.count(x)` - broj pojava vrednosti `x`
 - `s.index(x)` - pozicija prve pojave vrednosti `x`
 - `s.insert(i, x)` - umetanje vrednosti `x` na poziciju `i`
 - `s.pop()` - izbacivanje elementa sa kraja liste
 - `s.remove(x)` - izbacivanje elementa `x` iz liste
 - `s.reverse()` - obrtanje liste u mestu
 - `s.sort([key, [, reverse]])` - sortiranje liste u mestu

Stringovi

- Stringovi u Python-u 2 mogu biti **byte** i **unicode** stringovi.
- U python-u 2 unicode stringovi imaju prefix **u**.

```
ustr = u'Ово је ћирилични unicode стринг!'
bstr = 'Ово је byte string!'
```

Literali se navode unutar znakova navoda. Koriste se trostrukki za višelinjske stringove.

```
a = 'Ovo je string'
b = "I ovo je string"
c = """ Ovo je viselinijski
string
Evo jos jedne linije
"""
d = '''
I ovo je viselinijski
string
'''
```

- Stringovi su nepromenjivi objekti (*immutable*). Sve metode stringa koje vraćaju string kreiraju novi string.
- Neke od metoda:

```
s.capitalize()                                # Prvi karakter postaje veliko slovo.  
s.center(width [, pad])                      # Centririra string unutar zadate širine.  
s.find(sub [, start [, end]])                # Pronalazi podstring  
s.isalnum()                                   # True ukoliko je alfanumerik  
s.isdigit()                                   # True ukoliko su svi karakteri cifre  
s.lower()                                     # Sva slova postaju mala  
s.split([sep [,maxsplit]])                   # Deli string na mestu separator i vraca  
                                              # listu podstringova
```

Stringovi - formatiranje

- Tri načina:
 - Konkatanacija - operator **+** - izbegavati.
 - Interpolacija - **%** operator.
 - **format** metoda
 - f-stringovi - noviji i preferirani način.

```
>>> "Odgovor je %d" % 42
'Odgovor je 42'
>>> "Prvi=%s, drugi=%d, treći=%s" % ("prvi", 23, "third")
'Prvi=prvi, drugi=23, treći=third'
>>> "Prvi=%(prvi)s, drugi=%(drugi)d, pa opet %(prvi)s"
% {'drugi':11, 'prvi':'34'}
'Prvi=34, drugi=11, pa opet 34'
>>> "{} ribi {} rep.".format("Riba", "grize")
'Riba ribi grize rep.'
>>> "{1} ribi {0} rep.".format("Riba", "grize")
'grize ribi Riba rep.'
>>> "{ko} ribi {sta} rep.".format(ko="Riba", sta="grize")
'Riba ribi grize rep.'
>>> "{ko} ribi {sta} rep.".format(sta="soli", ko="riba")
'riba ribi soli rep.'
```

Mape

- Promenjive, neuređene kolekcije proizvoljnih objekata indeksirane proizvoljnim objektom (uz određena ograničenja).
- Rečnici (**dict**) su ugrađeni tip i predstavljaju implementaciju **hash** tabela ili asocijativnih nizova.
- **dict** kao ograničenje za ključeve zahteva neepromenjivost (*immutability*) jer **hash** vrednost mora biti konstantna.
- Operacije:

```
m = {}                      # Kreiranje praznog rečnika
m = {                        # Kreiranje rečnika sa elementima
    'BG': 11000,
    'NS': 21000,              # Zarez na kraju je dozvoljen
}
m['KG'] =                  # Upis u rečnik
m['NS'] =                  # Čitanje vrednosti
21000                         # Brisanje vrednosti
del m['BG']                  # Ključevi i vrednosti mogu biti
m[42] = 'Odgovor'            # različitog tipa
```

```

len(m)                                # Broj elemenata rečnika
'NS' in m                            # Provera pripadnosti
m.clear()                             # Uklanja sve elemente
m.copy()                               # Vraća kopiju od m
m.get(k[, default])                  # Vraća objekat pod ključem k a ako
                                      # ne postoji vraća v
m.items()                            # Vraća sekvencu (ključ, vrednost) parova
m.keys()                              # Vraća kolekciju ključeva
m.values()                            # Vraća kolekciju vrednosti
m.setdefault(k[, v])                 # Vraća m[k] ako postoji a ako ne vraća
                                      # v i postavlja m[k]=v
m.update(b)                           # Proširuje m sa elementima mape b
m.pop(k[, default])                  # Uklanja i vraća m[k] ukoliko postoji ili
                                      # default ukoliko ne postoji

```

Dict comprehensions

Treba da kreiramo mapu od sekvence ključeva i vrednosti:

```
mapa = {}  
for idx, kljuc in enumerate(kljucevi) :  
    mapa[kljuc] = vrednosti[idx]
```

ili upotrebom *dict comprehensions*:

```
mapa = { kljuc:vrednost for kljuc, vrednost in zip(kljucevi, vrednosti) }
```

Skupovi

- Neuređene kolekcije jedinstvenih elemenata.
- Elementi nisu indeksirani. Ne postoji *slice* operator.
- Elementi moraju biti nepromenjivi (*immutable*).
- Dve vrste:
 - `set` - promenljivi skup
 - `frozensest` - nepromenjivi skup
- Instanciraju se pozivom sa parametrom koji implementira iterator protokol ili alternativno upotrebom `{...}` sintakse.

```
s = set([1, 5, 10, 15])           # ili s = {1, 5, 10, 15}
f = frozensest(['a', 37, 'hello'])
```

- Set

Skupovi - operacije

```
len(s)                                # Broj elemenata
s.copy()                               # Kopija
s.intersection(t)                      # s & t - presek
s.union(t)                             # s | t - unija
s.difference(t)                        # s - t - razlika
s.symmetric_difference(t)              # s ^ t - simetrična razlika
s.isdisjoint(t)                        # True ako nemaju zajedničkih elem.
s.issubset(t)                           # True ako je s podskup od t
s.issuperset(t)                         # True ako je s nadskup od t

# Promenivi skupovi još imaju i
s.add(element)
s.remove(element)
s.clear()
s.update(iterable)                     # Dodaje sve elemente iterabilne kolekcije na s
...
```

Set comprehensions

```
{c for c in 'abracadabra' if c not in 'abc'}
```

Callables

- Objekti koji podržavaju semantiku poziva.
- Funkcije, Klase, metode.
- Tretiraju se kao i svi drugi objekti - mogu biti elementi kolekcija, mogu se prosledjivati kao parametri, biti povratne vrednosti drugih callables itd.

```
def foo(x,y):  
    return x + y  
  
bar = lambda x,y: x + y  
  
funkcije = [foo, bar]  
for f in funkcije:  
    print(f(2, 3))  
  
def div_by_maker(x):  
    def div_by(b):  
        return b/x  
    return div_by  
  
a = div_by_maker(5)      # = 4  
a(20)                   # = 4  
b = div_by_maker(2)      # = 10  
b(20)                   # = 5  
b(10)
```

Postepeno tipiziranje (*Gradual Typing*)

Statičko vs. dinamičko tipiziranje

- Statičko - tipovi poznati u vreme kompajliranja. Provera tipova (*type checking*) može da se uradi od strane kompajlera.
- Dinamičko - tipovi poznati u vreme izvršavanja (*run-time*). Provera tipova nije moguća a priori.

Prednosti statičkog tipiziranja

- Greške u tipovima pronalazi kompjuter.
- Bolji opis očekivanih ulaza funkcija. Automatska dokumentacija.
- Lakše i bezbednije refaktorisanje.

Postepeno tipiziranje

- Pristup kod koga delovi koda mogu biti staticki tipizirani dok drugi delovi mogu biti dinamički tipizirani.
- Moguća je delimična provera tipova.
- Postepeno se deo koji je staticki tipoziran može povećavati.
- Sve netipizirane varijable i parametri funkcija se implicitno tretiraju kao tip **Any** koji je kompatibilan sa svim tipovima i svi tipovi su kompatibilni sa njim.

Nagoveštaji tipova (*Type Hints*)

- Pristup kod koga se sprovodi postepeno tipiziranje navođenjem nagoveštaja koji su opcioni.
- Nagoveštaje koristi staticki proveravač tipova (*Static Type Checker*) - poseban alat koji se izvršava nad izvornim kodom kao deo CI/CD procedura.
- Uvedeno u Python verziji 3.5
- Statički proveravač tipova - mypy

Primeri

```
def greeting(name: str) -> str:  
    return 'Hello ' + name
```

```
def countTruthy(elements: List[Any]) -> int:  
    return sum(1 for elem in elements if elem)
```

```
from typing import TypeVar, Text
AnyStr = TypeVar('AnyStr', Text, bytes)

def concat(x: AnyStr, y: AnyStr) -> AnyStr:
    return x + y
```

```
from typing import Dict, List, Optional

class Node:
    ...

    class SymbolTable(Dict[str, List[Node]]):
        def push(self, name: str, node: Node) -> None:
            self.setdefault(name, []).append(node)

        def pop(self, name: str) -> Node:
            return self[name].pop()

        def lookup(self, name: str) -> Optional[Node]:
            nodes = self.get(name)
            if nodes:
                return nodes[-1]
            return None
```

Operatori i izrazi (TODO)

Struktura programa i kontrola tok'a

Struktura programa

- Svaki iskaz se tretira na isti način. Nema specijalnih iskaza.
- Svaki iskaz se može pojaviti bilo gde u programu.

```
if debug:  
    def square(x):  
        if not isinstance(x, float):  
            raise TypeError("Expected a float")  
        return x * x  
  
else:  
    def square(x):  
        return x * x
```

Uсловно izvršavanje

```
if expression:  
    statements  
elif expression:  
    statements  
elif expression:  
    statements  
...  
else:  
    statements
```

Petlje i iteracije

while expression:
statements

For petlja (s podržava iterator protokol):

for i in s:
statements

Ili razloženo:

```
it = s.__iter__() # Iterator za kolekciju s
while True:
    try:
        i = it.__next__()
        # Obradi element i
    except StopIteration:
        # Nema više elemenata
        break
    ...

```

Raspakivanje elemenata

Ukoliko su svim elementi kolekcije sekvence iste dužine moguće je uraditi sledeće:

```
# s je oblika [ (x1,y1,z1), (x2,y2,z2), ... ]  
for x,y,z in s:  
... Obroda x, y, z elemenata
```

Indeks u **for** petlji

- Kada se iterira kroz elemente kolekcije nekad je potrebno znati indeks.

```
i = 0
for x in s:
    statements
    i += 1

# Ili jednostavnije
for i, x in enumerate(s):
    statements
```

Parallelna iteracija

Često je potrebno iterirati parallelno kroz više kolekcija.

```
# s i t su dve sekvence
i = 0
while i < len(s) and i < len(t):
    x = s[i] # Uzmi i-ti element iz s
    y = t[i] # Uzmi i-ti element iz t
    statements
    i += 1
```

ili jednostavnije:

```
for x,y in zip(s,t):
    statements
```

For - else, break, continue

- Iz `for` petlje može prevremeno da se izđe upotrebom `break` iskaza.
- `For` petlja može imati opcioni `else` blok koji se izvršava ukoliko se petlja nije završila prevremeno (`break` iskaz).

```
for line in open("foo.txt"):  
    stripped = line.strip()  
    if not stripped:  
        break  
        # process the stripped line  
    ...  
    else:  
        raise RuntimeError("Missing section separator")
```

- Sa **continue** se može direktno preći na sledeći ciklus iteracije.

For - else, break, continue

```
for line in open("foo.txt"):  
    stripped = line.strip()  
    if not stripped: # Ako je linija prazna  
        continue # Preskoči je  
    # Obradi sve linije koje nisu prazne
```

Prepoznavanje obrazaca (*Pattern Matching*)

- Python 3.10
- PEP 636
- Iskaz prepoznavanja strukture i povezivanja varijabli (*binding*)

```
command = input("What are you doing next?\n")  
# analyze the result of command.split()  
[action, obj] = command.split()  
... # interpret action, obj
```

- Šta ako komanda ima više od dve reči?

```
match command.split() :  
    case [action, obj]:  
        ... # interpret action, obj
```

- povezuje `action = subject[0]` i `obj = subject[1]`.

- Može imati više grana:

```
match command.split() :  
    case [action] :  
        ... # interpret single-verb action  
    case [action, obj] :  
        ... # interpret action, obj
```

- Struktura obrazca može imati i konkretnie vrednosti:

```
match command.split() :
    case ["quit"]:
        print("Goodbye!")
        quit_game()
    case ["look"]:
        current_room.describe()
    case ["get", obj]:
        character.get(obj, current_room)
    case ["go", direction]:
        current_room = current_room.neighbor(direction)
    # The rest of your commands go here
```

- Podrazumevani obrazac i prepoznavanje više vrednosti:

```
match command.split() :  
    case ["quit"] : ... # Code omitted for brevity  
    case ["go", direction] : ...  
    case ["drop", *objects] : ...  
    ... # Other cases  
    case _ :  
        print(f"Sorry, I couldn't understand {command!r}")
```

- Obrasci se mogu komponovati:

```
match command.split():
    ...
    ... # Other cases
    case ["north"] | ["go", "north"]:
        current_room = current_room.neighbor("north")
    case ["get", obj] | ["pick", "up", obj] | ["pick", obj, "up"]:
        ...
        ... # Code for picking up the given object
```

Funkcije i funkcionalno programiranje

Osnove

Definisanje funkcije:

```
def add(x, y):  
    return x + y
```

Osnove

Definisanje funkcije:

```
def add(x, y):  
    return x + y
```

Lambda funkcija:

```
l = lambda x, y: x + y
```

Podrazumevana vrednost parametara

```
def split(line, delimiter=','):  
    statements
```

Povezivanje je u trenutku kreiranja funkcije.

```
a = 10  
def foo(x=a):  
    return x  
  
a = 5      # Redefinisanje varijable 'a'  
foo()      # Vraća 10 (podrazumevana vrednost nije promenjena)
```

Napomena kod *mutable* tipova

Problem:

```
def foo(x, items=[]):
    items.append(x)
    return items

foo(1)    # Vraća [1]
foo(2)    # Vraća [1, 2]
foo(3)    # Vraća [1, 2, 3]
```

Rešenje:

```
def foo(x, items=None):  
    items = [] if items is None else items  
    items.append(x)  
    return items
```

Promenjiv broj argumenta - po poziciji

```
def fprintf(file, fmt, *args):  
    file.write(fmt % args)
```

Poziv `fprintf` - `args` postaje n-torka (42,"hello world", 3.45)

```
fprintf(out, "%d %s %f", 42, "hello world", 3.45)
```

n-torke možemo i "raspakovati" pri pozivu upotrebom `*` operatora

```
def printf(fmt, *args):  
    # Poziv druge funkcije i prosledjivanje argumenta  
    fprintf(sys.stdout, fmt, *args)
```

Ili na primer

```
a = (2, 3)  
f = lambda x, y: x + y  
print(f(*a)) # n-torka a se "razlaže" i prosledjuje poziciono
```

Prosleđivanje parametara po nazivu

```
def foo(x, y, z, w):  
    statements
```

Prosleđivanje vrednosti parametara po nazivu:

```
foo(x=3, y=22, w='hello', z=[1,2])
```

Prosleđivanje parametara po nazivu

```
def foo(x, y, z, w):  
    statements
```

Prosleđivanje vrednosti parametara po nazivu:

```
foo(x=3, y=22, w='hello', z=[1, 2])
```

Može i kombinovano

```
foo(3, 22, w='hello', z=[1, 2])
```

Prosleđivanje parametara po nazivu

```
def foo(x, y, z, w):  
    statements
```

Prosleđivanje vrednosti parametara po nazivu:

```
foo(x=3, y=22, w='hello', z=[1,2])
```

Može i kombinovano

```
foo(3, 22, w='hello', z=[1,2])
```

Ali ne i ovako - višestrukе vrednosti za y

```
foo('hello', 3, z=[1,2], y=22)
```

Promenjiv broj argumentenata - imenovani parametri

```
def make_table(data, **parms):
    # Preuzimanje konfiguracionih parametara
    fgcolor = parms.pop("fgcolor", "black")
    bgcolor = parms.pop("bgcolor", "white")
    width = parms.pop("width", None)
    ...
    # Nema više opcija
    if parms:
        raise TypeError("Konfiguracione opcije '%s' nisu podržane" % list(parms) )

make_table(items, fgcolor="black", bgcolor="white", border=1,
           borderstyle="grooved", cellpadding=10,
           width=400)
```

Promenjiv broj argumenta - kombinovan prenos

Mogu se kombinovati pozicioni i imenovani parametri dok god se imenovani (`**`) nalaze na kraju

Različit broj pozicionih i imenovanih parametara

```
def spam(*args, **kwargs):
    # args je n-torka sa pozicionim parametrima
    # kwargs je rečnik sa imenovanim parametrima
    ...
```

Možemo i posleđivati parametre drugim funkcijama. To se često koristi kod tzv. *wrapper* ili *proxy* funkcija

```
def callfunc(*args, **kwargs):
    func(*args, **kwargs)
```

Prenos parametara i povratne vrednosti

Prenos se obavlja po referenci.

```
a = [1, 2, 3, 4, 5]
def square(items):
    for i, x in enumerate(items):
        items[i] = x * x # Menja elemente u mestu

square(a) # Promena u [1, 4, 9, 16, 25]
```

Ako funkcija vraća više vrednosti to se može učiniti n-torkom (**tuple**).

```
def factor(a):
    d = 2
    while (d &lt;= (a / 2)):
        if ((a / d) * d == a):
            return ((a / d), d)
        d = d + 1

(x, y) = factor(1234)
# ili jednostavno
x, y = factor(1234)
```

Opseg važenja (*scoping rules*) - lokalne i globalne varijable

```
a = 42
def foo():
    a = 13
foo()
# ovde je a 42

a = 42
b = 37
def foo():
    global a      # Deklariramo 'a' kao globalnu
    a = 13
    b = 0
foo()
# a je 13. b je još uvek 37.
```

Opseg važenja (*scoping rules*) - ne-lokalne varijable

```
def countdown(start):
    n = start
    def display():
        print('T-minus %d' % n)
    def decrement():
        n -= 1 # <- Greška! Local variable 'n' referenced before assignment
    while n > 0:
        display()
        decrement()
```

```
def countdown(start):
    n = start
    def display():
        print('T-minus %d' % n)
    def decrement():
        nonlocal n
        n -= 1
    while n > 0:
        display()
        decrement()
```

Dekoratori

- Dekorator obrazac.
- Funkcije koje prihvataju kao parametar funkciju (ili uopšte **callable**) i vraćaju izmenjenu verziju.

```
@trace
def square(x):
    return x*x

# Ovo je ekvivalentno sa
def square(x):
    return x*x
square = trace(square)
```

```
enable_tracing = True
if enable_tracing:
    debug_log = open("debug.log", "w")

def trace(func):
    if enable_tracing:
        def callf(*args, **kwargs):
            debug_log.write("Calling %s: %s\n" %
                            (func.__name__, args, kwargs))
            r = func(*args, **kwargs)
            debug_log.write("%s returned %s\n" %
                            (func.__name__, r))

            return r
        return callf
    else:
        return func
```

Dekoratori (2)

Mogu da se stekuju:

```
@foo  
@bar  
@spam  
def grok(x):  
    pass
```

je isto što i

```
def grok(x):  
    pass  
grok = foo(bar(spam(grok)))
```

Dekoratori (3)

Mogu da imaju parametre:

```
# Event handler decorator
event_handlers = { }

def eventhandler(event):
    def register_function(f):
        event_handlers[event] = f
        return f
    return register_function

# Sto je ekvivalentno sa
def handle_button(msg):
    ...

temp = eventhandler('BUTTON')
handle_button = temp(handle_button)
```

List comprehensions opet

```
nums = [1, 2, 3, 4, 5]
squares = []
for n in nums:
    squares.append(n * n)

# Ekvivalentno
nums = [1, 2, 3, 4, 5]
squares = [n * n for n in nums]
```

```
# Opští oblik sintakse
[expression for item1 in iterable1 if condition1
    for item2 in iterable2 if condition2
    ...
    for itemN in iterableN if conditionN]

# Što je ekvivalentno sa
s = []
for item1 in iterable1:
    if condition1:
        for item2 in iterable2:
            if condition2:
                ...
                for itemN in iterableN:
                    if conditionN: s.append(expression)
```

List comprehensions primeri

```
a = [-3, 5, 2, -10, 7, 8]
b = 'abc'

c = [2*s for s in a]
d = [s for s in a if s >= 0]
e = [(x, y) for x in a
      for y in b
      if x > 0]
f = [(1, 2), (3, 4), (5, 6)]
g = [math.sqrt(x * x + y * y)
      for x, y in f] # g = [2.23606, 5.0, 7.81024]
```

Generator izrazi

Slično kao *list comprehensions* ali ne kreiraju listu već generator objekat koji izračunava vrednosti na zahtev (lenja evaluacija).

```
# Opšti oblik sintakse
(expression for item1 in iterable1 if condition1
     for item2 in iterable2 if condition2
     ...
     for itemN in iterableN if conditionN )
```

```
>>> a = [1, 2, 3, 4]
>>> b = (10*i for i in a)
>>> b
<generator object at 0x590a8>
>>> b.next()
10
>>> b.next()
20
...
...
```

Generator izrazi - primer

```
f = open("data.txt")
lines = (t.strip() for t in f)

comments = (t for t in lines if t[0] == '#')

for c in comments:
    print(c)

# Uvek se može konvertovati u listu
clist = list(comments)
```

lambda iskaz

Kreiranje anonimne funkcije.

Sintaksa:

```
lambda args : expression
```

Primeri:

```
a = lambda x, y : x+y  
r = a(2, 3)
```

Osnovna namena - kratke callback funkcije.

Primer - *case-insensitive* sortiranje:

```
names.sort(key=lambda n: n.lower())
```

Klase i objektno-orientisano programiranje

Klase

- Kolekcija funkcija (metoda), varijabli (atributa) i dinamičkih atributa(*properties*).

```
class Account(object):  
    num_accounts = 0  
  
    def __init__(self, name, balance):      # Konstruktor  
        self.name = name  
        self.balance = balance  
        Account.num_accounts += 1           # Přístup deljenom class atributu  
  
    def __del__(self):  
        Account.num_accounts -= 1  
  
    def deposit(self, amt):  
        self.balance = self.balance + amt  
  
    def withdraw(self, amt):  
        self.balance = self.balance - amt  
  
    def inquiry(self):  
        return self.balance
```

class iskaz

- Klasa je python objekat koji se kreira kada interpreter nađe na `class` iskaz i uspešno ga obradi.
- Referenca na ovaj objekat je ime klase.
- `class` iskaz nije ni počemu poseban i može da se koristi bilo gde se mogu koristiti i drugi iskazi.

```
def napravi_klasu():
    class Mojaklasa(object):
        def __init__(self, a):
            self.a = a
    return Mojaklasa

k = napravi_klasu()                      # k je klasa
m = napravi_klasu()                      # m je klasa
id(k) != id(m)                          # ali nova
```

Instanciranje

- Instanciranje objekta se obavlja pozivom klase.
- Klasa je *callable*.

```
a = Account("Guido", 1000.00) # Poziva Account.__init__(a, "Guido", 1000.00)
b = Account("Bill", 10.00)
```

Referenciranje atributa i metoda

```
a.deposit(100.00)      # Poziva Account.deposit(a, 100.00)
b.withdraw(50.00)       # Poziva Account.withdraw(b, 50.00)
name = a.name            # Pristup 'name' atributu
```

Opseg važenja (*scoping*)

- Klase definišu prostor imena (*namespace*) ali metode nemaju prostor imena.
- Pristup atributima iz metoda mora biti potpuno kvalifikovan
 - U tu svrhu koristi se eksplicitna `self` referenca.

```
class Foo(object):  
  
    def bar(self):  
        print("bar!")  
  
    def spam(self):  
        bar(self)      # Neispravno! 'bar' baca NameError izuzetak  
        self.bar()    # Ispravno  
        Foo.bar(self) # Takođe ispravno
```

Nasledjivanje

```
import random

class EvilAccount(Account):

    def inquiry(self):
        if random.randint(0, 4) == 1:
            return self.balance * 1.10
        else:
            return self.balance

c = EvilAccount("George", 1000.00)
c.deposit(10.0)
available = c.inquiry()
```

- Naslednica može da doda nove atribute.

```
class EvilAccount(Account):

    def __init__(self, name, balance, evilfactor):
        Account.__init__(self, name, balance)
        self.evilfactor = evilfactor

    def inquiry(self):
        if random.randint(0, 4) == 1:
            return self.balance * 1.10
        else:
            return self.balance
```

super funkcia

```
class MoreEvilAccount(EvilAccount) :  
  
    def deposit(self, amount) :  
        self.withdraw(5.00)  
        EvilAccount.deposit(self, amount)  
  
class MoreEvilAccount(EvilAccount) :  
  
    def deposit(self, amount) :  
        self.withdraw(5.00)  
        super().deposit(amount)
```

Višestruko nasleđivanje

```
class DepositCharge(object):
    fee = 5.00
    def deposit_fee(self):
        self.withdraw(self.fee)

class WithdrawCharge(object):
    fee = 2.50
    def withdraw_fee(self):
        self.withdraw(self.fee)

class MostEvilAccount(EvilAccount,
                      DepositCharge,
                      WithdrawCharge):
    def deposit(self, amt):
        self.deposit_fee()
        super().deposit(amt)

    def withdraw(self, amt):
        self.withdraw_fee()
        super().withdraw(amt)

d = MostEvilAccount("Dave", 500.00, 1.10)
d.deposit_fee()      # DepositCharge.deposit_fee(). Fee je 5.00
d.withdraw_fee()    # WithdrawCharge.withdraw_fee(). Fee je 5.00 ??
```

Višestruko nasleđivanje - MRO

- *Method Resolution Order* - MRO

```
>>> MostEvilAccount.mro
(<class '__main___.MostEvilAccount'>,
 <class '__main___.EvilAccount'>,
 <class '__main___.Account'>,
 <class '__main___.DepositCharge'>,
 <class '__main___.WithdrawCharge'>,
 <type 'object'>)
>>>
```

Polimorfizam, dinamičko povezivanje, *duck typing*

- Korišćenje objekta bez obzira na njegov konkretni tip.
- Dovoljno je samo da ima određene atribute i metode tj. određeno ponašanje.
- U Python-u nije bitna ni hijerarhija nasleđivanja.
- Primer: *file-like* objekti iz standardne biblioteke.

static metode

```
class Foo(object):
    @staticmethod
    def add(x, y):
        return x + y

x = Foo.add(3, 4)
```

```
class Date(object):
    def __init__(self,year,month,day):
        self.year = year
        self.month = month
        self.day = day

    @staticmethod
    def now():
        t = time.localtime()
        return Date(t.tm_year, t.tm_mon, t.tm_day)

    @staticmethod
    def tomorrow():
        t = time.localtime(time.time() + 86400)
        return Date(t.tm_year, t.tm_mon, t.tm_day)

a = Date(1967, 4, 9)
b = Date.now() # Calls static method now()
c = Date.tomorrow() # Poziva static metodu tomorrow()
```

class metode

- Metode klase koje primaju `class` objekat klase nad kojom su pozvane.

```
class Times(object):
    factor = 1

@classmethod
def mul(cls,x):
    return cls.factor * x

class TwoTimes(Times):
    factor = 2

x = TwoTimes.mul(4)      # Poziva Times.mul(TwoTimes, 4) -> 8
```

Primer upotrebe **class** metode - problem

class EuroDate(Date):

```
# Izmena string konverzije da koristi evropske datume
def __str__(self):
    return "%02d/%02d/%4d" % (self.day, self.month, self.year)
```

- Problem je ukoliko se pozove `EuroDate.now()` biće vraćena instanca **Date** klase.

Primer upotrebe `class` metode - rešenje

```
class Date(object):
    ...
    @classmethod
    def now(cls):
        t = time.localtime()
        # Kreiranje objekat odgovarajućeg tipa
        return cls(t.tm_year, t.tm_month, t.tm_day)

class EuroDate(Date):
    ...
    a = Date.now()      # Poziva Date.now(Date) i vraća Date
    b = EuroDate.now()  # Poziva Date.now(EuroDate) i vraća EuroDate

    # Jedna napomena. Metode su dostupne i na instancama
    a = Date(1967, 4, 9)
    b = d.now()         # Poziva Date.now(Date)
```

Properties

- Specijalna vrsta atributa koja dinamički izračunava svoju vrednost.

```
class Circle(object):  
    def __init__(self, radius):  
        self.radius = radius  
  
    @property  
    def area(self):  
        return math.pi * self.radius ** 2  
  
    @property  
    def perimeter(self):  
        return 2 * math.pi * self.radius
```

```
>>> c = Circle(4.0)  
>>> c.radius  
4.0  
>>> c.area  
50.265482445743669  
>>> c.perimeter  
25.132741228718345  
>>> c.area = 2  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: can't set attribute  
>>>
```

Properties - setters

```
class Foo(object):
    def __init__(self, name):
        self.__name = name
    @property
    def name(self):
        return self.__name
    @name.setter
    def name(self, value):
        if not isinstance(value, str):
            raise TypeError("Must be a string!")
        self.__name = value
    @name.deleter
    def name(self):
        raise TypeError("Can't delete name")

f = Foo("Guido")
n = f.name # Pozivā f.name() – getter
f.name = "Monty" # Pozivā setter name(f, "Monty")
f.name = 45 # Pozivā setter name(f, 45) –> TypeError
del f.name # Pozivā deleter name(f) –> TypeError
```

Enakpsulacija i privatni atributi

```
class A(object):
    def __init__(self):
        self.__X = 3
    def __spam(self):
        pass
    def bar(self):
        self.__spam() # Poziva A.__spam()

class B(A):
    def __init__(self):
        A.__init__(self)
        self.__X = 37
    def __spam(self):
        pass
```

Redefinisnje operatora (*operator overloading*)

- Svi operatori korišćeni u Python-u (npr. `+`, `-`, `*`, `/`, `in`, `[]...`) su opisani specijalnim metodama i mogu se redefinisati.

```
class Complex(object):  
    def __init__(self, real, imag=0):  
        self.real = float(real)  
        self.imag = float(imag)  
    def __repr__(self):  
        return "Complex(%s, %s)" % (self.real, self.imag)  
    def __str__(self):  
        return "(%g+%gi)" % (self.real, self.imag)  
    # self + other  
    def __add__(self, other):  
        return Complex(self.real + other.real, self.imag + other.imag)  
    # self - other  
    def __sub__(self, other):  
        return Complex(self.real - other.real, self.imag - other.imag)
```

- Napomena: Ovo je samo ilustrativan primer - Python već ima ugrađen tip kompleksnih brojeva.

Pričadnost klasi ili tipu

```
class A(object): pass
class B(A): pass
class C(object): pass
a = A() # Instance of 'A'
b = B() # Instance of 'B'
c = C() # Instance of 'C'

type(a) # Vraća klasu A (class objekat)
isinstance(a, A) # True
isinstance(b, A) # True, B nasleđuje A
isinstance(b, C) # False, B ne nasleđuje C

issubclass(B,A) # True
issubclass(C,A) # False
```

dataclass

- Python 3.7
- PEP 557

- Definisanje tipa/klase namenjene čuvanju podataka
- *Promenjive imenovane torke sa podrazumevanim vrednostima*

```
@dataclass
class InventoryItem:
    '''Class for keeping track of an item in inventory.'''
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

- **@dataclass** anotacija dodaje podrazumevane metode `__init__`, `__repr__`,
`__eq__`, ...

Moduli i paketi

Moduli i `import` iskaz

- Svaki pajton fajl može da se koristi kao modul.
- `import` iskaz *uvodi* definicije iz drugog modula u tekući prostor imena (*namespace*)

```
# spam.py
a = 37
def foo():
    print("I'm foo and a is %s" % a)
def bar():
    print("I'm bar and I'm calling foo")
foo()

class Spam(object):
    def grok(self):
        print("I'm Spam.grok")
```

```
import spam
x = spam.a
spam.foo()
s = spam.Spam()
s.grok()
...
```

import - razni oblici

```
import socket, os, re  
  
import spam as sp  
import socket as net  
sp.foo()  
sp.bar()  
net.gethostname()
```

```
if format == 'xml':  
    import xmlreader as reader  
elif format == 'csv':  
    import csvreader as reader  
data = reader.read_data(filename)
```

Import samo određenog objekta.

```
from spam import foo  
foo()  
spam.foo()
```

Import na više linija

```
from spam import foo,  
bar,  
Spam
```

Promena imena pri importu.

```
from spam import Spam as Sp  
s = Sp()
```

Import svih definicija u tekući prostor imena.

```
from spam import *
```

Definisanje šta se uvozi kod import `*`.

```
# module: spam.py
__all__ = [ 'bar', 'Spam' ]
```

Opseg važenja se ne menja.

```
from spam import foo
a = 42
foo() # Ispisuje "I'm foo and a is 37"
```

```
from spam import bar
def foo():
    print("I'm a different foo")
bar() # Kada bar pozove foo(), poziva se spam.foo(), a ne
      # definicija foo() iz ovog fajla
```

Izvršavanje glavnog programa

- `import` iskaz izvršava kod u prostoru imena pozivaoca.
- Svaki modul definiše implicitno varijablu `__name__` koja predstavlja ime modula.
 - Ukoliko se modul startuje kao nezavisan program i tada dolazi do izvršavanja koda ali će `__name__` varijabla imati vrednost `__main__`.
 - Pajton program se startuje sa:
- Možemo u modulu imati ovakav kod da bi obezbedili drugačije tretiranje modula pri importu i pri startovanju kao nezavisan program.

```
$ python moj_program.py
```

```
if __name__ == '__main__':
    # Startovan kao program
else:
    # Importovan kao modul
```

- Za više informacija videti relativni import i ime paketa na SO

Alati i okruženja

IPython

- Interaktivni *shell* sličan standardnom
- *Read-Eval-Print-Loop*
- Razvoj kroz eksperimentisanje

IPython mogućnosti

- Dopuna sa TAB tasterom
- Istraživanje objekata sa `?`
- `Autoreload` modula
- `Magic` funkcije

Primer sesije

```
[igor@sizif] $ ipython2
Python 2.7.8 (default, Jul 1 2014, 17:30:21)
Type "copyright", "credits" or "license" for more information.

IPython 2.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help   -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: print("Hello world!")
Hello world!

In [2]:
```

Dopuna koda

Pritisak na taster TAB

```
In [4]: import os  
In [5]: os.pa          os.path      os.pathconf  
       os.pardir        os.pathsep  
       os.pathconf_names  
  
In [5]: os.pa
```

Informacije o objektima

Iza naziva referenčne stavitvi znak .

```
In [7]: map?  
Type:          builtin_function_or_method  
String form:  <built-in function map>  
Namespace:    Python builtin  
  
Docstring:  
map(function, sequence[, sequence, ...]) -> list
```

Return a list of the results of applying the function to the items of the argument sequence(s). If more than one sequence is given, the function is called with an argument list consisting of the corresponding item of each sequence, substituting None for missing values when not all sequences have the same length. If the function is None, return a list of the items of the sequence (or a list of tuples if more than one sequence).

In [8]:

Proširene informacije o objektima

Iza naziva reference staviti znak ??

```
In [2]: import os
```

```
In [3]: os.path.abspath??  
Type:    function  
String form: <function abspath at 0x7f723641b848>  
File:    /usr/lib/python2.7 posixpath.py  
Definition: os.path.abspath(path)  
Source:  
def abspath(path):  
    """Return an absolute path. """  
    if not isabs(path):  
        if isinstance(path, _unicode):  
            cwd = os.getcwdu()  
        else:  
            cwd = os.getcwd()  
        path = join(cwd, path)  
    return normpath(path)
```

paste više linija koda

- Ponekad je zgodno u cilju testiranja *paste*-ovati blok koda na konzolu uz očuvanje uvlačenja.
- Za ovu namenu koristi se magična funkcija `%paste`

```
In [5]: %paste
def napravi_klasu():
    class MojaKlasa(object):
        def __init__(self, a):
            self.a = a
    return MojaKlasa
## -- End pasted text --
```

Reload modula

- Problem kod izmene koda posle import-a.
- Dva načina:
 1. **reload** funkcija:

```
reload(moj_modul)
```

1. **autoreload** ekstenzija:

```
%load_ext autoreload  
%autoreload 2
```

Zadatak

Upotrebom IPython konzole:

- kreirati niz od 100 celih brojeva iz intervala **[1, 1000]** tako da se brojevi ne ponavljaju.
- Pronaći minimalni i maksimalni element niza.
- Sortirati niz u opadajućem redosledu.
- Napraviti modul sa funkcijom koja vraća sortirani niz sa brojem elemenata zadatim kao parametar.
- Uraditi import ove funkcije u IPython sesiju.
- Obrnuti smjer sortiranja.
- Uraditi **reload** modula u IPython-u bez restarta i verifikovati da je funkcija izmenjena.

Pomoć: **random** modul

PyCharm

The screenshot shows the PyCharm IDE interface with the following details:

- Project Bar:** Shows the project name "Arpeggio [Arpeggio]" and the file path ".../examples/calc.py - PyCharm Community Edit...".
- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Toolbars:** Standard toolbar with icons for Open, Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others.
- Structure View:** Shows the project structure with files like calc.py, __init__.py, and various Arpeggio-related files.
- Code Editor:** The main editor window displays the Python code for "calc.py". The code defines a grammar for a simple expression evaluator using Arpeggio's PEG grammar definition.

```
#####
# Name: calc.py
# Purpose: Simple expression evaluator example
# Author: Igor R. Dejanovic <igor DOT dejanovic AT gmail DOT com>
# Copyright: (c) 2009-2014 Igor R. Dejanovic <igor DOT dejanovic AT gmail DOT com>
# License: MIT License

# This example demonstrates grammar definition using python constructs as
# well as using semantic actions to evaluate simple expression in infix
# notation.

from arpeggio import Optional, ZeroOrMore, OneOrMore, EOF, SemanticAction, \
    ParserPython
from arpeggio.export import PMDOTExporter, PTDOTExporter
from arpeggio import RegExMatch as _


def number():
    return r'\d*\.\d+'
def factor():
    return Optional(['+', '-']), number()
    ('*', expression, '+'))
def term():
    return factor(), ZeroOrMore(['*', '/', '-'], factor)
def expression():
    return term(), ZeroOrMore(['+', '-'], term)
def calc():
    return OneOrMore(expression), EOF

# Semantic actions
class ToFloat(SemanticAction):
    Converts node value to float.

    def first_pass(self, parser, node_children):
        if self._first_pass:- Tool Windows: Favorites, Structure, Event Log.
- Status Bar: Shows file paths like ".../examples/calc.py", "Structure", and "Event Log".

```

PyCharm - osobine

- Osnovne IDE operacije: navigacija, bojenje i dopuna koda, prikaz strukture koda...
- Podrška za analizu koda i refaktorisanje
- Integrisani debager. Podrška za testiranje
- Podrška za *Django* web framework, editovanje `HTML` i `JavaScript` fajlova
- Komercijalni (firma *JetBrains*) - dostupan u *Community verziji*
- Pisan u Javi, radi na svim vodećim OS

Eclipse PyDev

PyDev - arpeggio/examples/bibtex.py - Eclipse

File Edit Source Refactoring Navigate Search Project Pydev Run Window Help

Quick Access

PyDev Pack

PyDev Pack

Arpeggio [Arpeggio]

- > Arpeggio
- > Arpeggio.egg-info
- > examples
- > init_.py
- > bib_parse_tree_
- bibtex.py
- calc_parse_tree.
- calc_parse_parser
- calc_peg.py
- calc_profile.py
- calc.profile
- calc.py
- calc2.py
- csv_parse_tree_
- examples

first

41 # Semantic actions

42 # Semantic actions

43 class BibFileSem(SemanticAction):

44 Just returns list of child nodes (bibentry)

45 def first_pass(self, parser, node, children)

46 if parser.debug:

47 print("Processing Bibfile")

48 # Return only dict nodes

49 return [x for x in children if type(x)

50 # Return only dict nodes

51 # comment_entry

52 # bibentry

53 # field

54 # fieldvalue

55 class BibEntrySem(SemanticAction):

56 Constructs a map where key is bibentry file

57 Key is returned under 'bibkey' key. Type i

58 def first_pass(self, parser, node, children)

59 if parser.debug:

60 print("Processing bibentry %s" %

61 csv_parse_tree_

62 examples

Outline

type filter text

print_function(_future)

pprint

sys

* (arpeggio)

PMDOTExporter.PTDOI

= RegExMatch (arpeggio)

bibfile

comment_entry

bibentry

field

fieldvalue

fieldvalue_braces

fieldvalue_quotes

fieldname

comment

bibtype

bibkey

fieldvalue_quoted_cont

PyDev - Osobine

- Slobodan softver otvorenog koda.
- Dostupan kao skup plugin-a za *Eclipse*
- Osnovne operacije: navigacija, strukturni prikaz, bojenje i dopuna koda...
- Podrška za refaktorisanje ali trenutno na nižem nivou od *PyCharm*.
- Integrirani debager, interaktivna konzola, podrška za testiranje
- Podrška za *Django* i Django template
- Pisan u *Javi*, radi na svim vodećim OS

Editori

- VS Code
- vim, neovim
- emacs
- Sublime
- Atom
- ...

Pakovanje i distribucija aplikacija

Pakovanje i distribucija aplikacija u Python-u

- Distutils
- setuptools
- pip
- PyPi
- virtualenv
- wheels

Distutils

- Standardna biblioteka za upravljanje paketima.
- Dolazi uz instalaciju Pythona.

SetupTools

- Naprednija verzija biblioteke za upravljanje paketima.
- Dobrim delom kompatiblina sa *Distutils*.
- Podrazumenvano se instalira u virtuelno okruženje (`virtualenv`)

setup.py fajl

- Metapodaci python paketa + informacije za *build*.

- Primer:

```
*!/usr/bin/env python

from setuptools import setup
*from distutils.core import setup

setup(name='ImePaketa',
      version='1.0',
      description='Opis paketa',
      author='Ime i prezime autora',
      author_email='mailautora@negde.com',
      url='http://ulrprojekta.com/',
      packages=['prvipaket', 'drugipaket',
                'drugipaket.podpaket'],
      )
```

Instalacija iz setup.py

Instalacija iz izvornog koda sa `setup.py` fajlom se obavlja komandom:

```
$ python setup.py install
```

Instalacija za razvoj

Ukoliko kôd koji želimo da instaliramo još uvek razvijamo a želimo da izbegnemo ponovnu instalaciju posle svake izmene potrebno je da instaliramo paket na sledeći način:

```
$ python setup.py develop
```

Za deinstalaciju razvojnog paketa koristi se:

```
$ python setup.py develop --uninstall
```

Zadatak

- Kreirati fajl `gsearch.py` sa sadržajem sa sledećeg slajda.
- Napraviti `setup.py` za njega i instalirati ga u `develop` modu.
- Pokrenuti `IPython` sa proizvoljne lokacije i importovati `gsearch` modul.
 - Pozvati `search` funkciju.
 - Izmeniti `search` funkciju.
- Importovati `gsearch` modul i verifikovati da je izmenjen.

Primer

```
#!/usr/bin/python3
from urllib import parse, request
import json

def search(query):

    url = "http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q="

    query = parse.urlencode( { 'q' : query } )
    response = request.urlopen( url + query ).read()
    data = json.loads( response.decode("utf-8") )
    results = data[ 'responseData' ][ 'results' ]

    ret_results = []
    for result in results:
        title = result[ 'title' ]
        url = result[ 'url' ]
        ret_results.append( (title, url))

    return ret_results

if __name__=="__main__":
    for title, link in search("fakultet tehnickih nauka"):
        print("{} [{}]\n".format(title, link))
```

- Primer preuzet sa ove adrese i prilagođen kursu

Kreiranje distribucija iz `setup.py`

- Kreiranje *source* distribucije:

```
$ python setup.py sdist
```

- Kreiranje *binarne* distribucije:

```
$ python setup.py bdist
```

- Npr. kreiranje binarnog installera za windows se obavlja sledećom komandom:

```
$ python setup.py bdist_winst
```

Python wheels

- Novi način distribucije paketa.
- Cekstenzije su prekompljirane za ciljnu platformu.
- Brža instalacija.

Kreiraju se sa:

```
$ python setup.py bdist_wheel
```

Python Package Index - PyPI

- PyPI (Python **P**ackage **I**ndex) predstavlja repozitorijum python paketa.
- Dostupan je na adresi <https://pypi.python.org/>
- Paketi se mogu pretraživati i prezimati putem web interfejsa ali i putem **pip** alata.

pip alat

Osnovne komande:

- Pretraga paketa po nazivu:

```
pip search deo_imena
```

- Instalacija paketa:

```
pip install ime_paketa
```

- Prikaz instaliranih paketa:

```
pip list
```

- *Upgrade* paketa:

```
pip install --upgrade ime_pake
```

- Deinstalacija paketa:

```
pip uninstall ime_paketa
```

virtualenv

- Problem sa zavisnošću i kolizijom između verzija.
- **virtualenv** omogućava kreiranje izolovanih Python okruženja sa svojim skupom paketa.
- Kada se aktivira određeno okruženje, sistemski paketi kao i paketi iz drugih okruženja se ne vide.
- Kreiranje novog okruženja:

```
$ virtualenv JSD
Using base prefix '/usr'
New python executable in /home/igor/VirtualEnv/JSD/bin/python3
Also creating executable in /home/igor/VirtualEnv/JSD/bin/python
Installing setuptools, pip, wheel...done.
$
```

virtualenvwrapper

- Skup proširenja **virtualenv** alata.
- Upravljanje virtualnim okruženjima.

```
mkvirtualenv typhoon
```

Aktivacija:

```
workon typhoon  
pip install ...  
pip list ...
```

Brisanje:

```
rmvirtualenv typhoon
```

Aktivacija virtuelnog okruženja

```
$ source JSD/bin/activate  
(JSD) $
```

Listanje paketa u okruženju:

```
(JSD) $ pip list  
pip (8.1.2)  
setuptools (28.6.1)  
wheel (0.30.0a0)  
(JSD) $
```

- U okruženju se podrazumjevano ne vide sistemski paketi.
- Okruženje je izolovano od drugih okruženja.

Instalacija paketa u virtuelno okruženje

```
(JSD) $ pip install textX
Collecting textX
  Downloading textX-1.4.tar.gz
    Collecting Arpeggio (from textX)
      Downloading Arpeggio-1.5.tar.gz
        Building wheels for collected packages: textX, Arpeggio
          Running setup.py bdist_wheel for textX done
          Stored in directory: /home/igor/.cache/pip/wheels/...
          Running setup.py bdist_wheel for Arpeggio done
          Stored in directory: /home/igor/.cache/pip/wheels/...
        Successfully built textX Arpeggio
      Installing collected packages: Arpeggio, textX
      Successfully installed Arpeggio-1.5 textX-1.4
```

Sada se Django biblioteka nalazi u aktivnom okruženju:

```
(JSD) $ pip list
Arpeggio (1.5)
pip (8.1.2)
setuptools (28.6.1)
textX (1.4)
wheel (0.30.0a0)
(JSD) $
```

Zadatak

- Kreirati novo virtuelno okruženje *JSD*.
- Aktivirati ga
- Instalirati sledeće pakete: *IPython, Arpeggio, textX, Django*
- Izlistati sve instalirane pakete
- Deinstalirati paket *Django*
- Konfigurisati PyDev da prepoznaje pakete iz novog okruženja. Ovo se može verifikovati sa:

```
import arpeggio
```
- Ukoliko *arpeggio* nije na putanji PyDev će označiti grešku da modul ne postoji.

Reference

- Beazley, David M. *Python essential reference*. Addison-Wesley Professional, 2009.
- Python dokumentacija
 - [virtualenv dokumentacija]
(<http://virtualenv.readthedocs.org/en/latest/virtualenv.html>)
 - [pip dokumentacija] (<https://pip.readthedocs.org/en/latest/>)