Python za Java programere

izvor: http://python4java.necaiseweb.org

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2019.

Python za Java programere

- izvršavanje programa
- osnovne konstrukcije
- funkcije i metode
- interakcija sa korisnikom
- stringovi
- grananje i petlje
- funkcije
- moduli
- liste (nizovi)
- tekstualni fajlovi
- rečnici
- klase, veze među klasama
- preklapanje operatora
- izuzeci
- nasleđivanje

Izvršavanje Python programa

Python program je tekstualni fajl koji sadrži naredbe. Primer:

```
# Classic hello world program in Python.

print("Hello World")
print("How are you today?")
```

Nema `main()`

Prva naredba koja se izvršava je prva naredba u fajlu na globalnom nivou (izvan svih funkcija i metoda).
U prethodnom primeru, to je prva `print` naredba.

```
python myprogram.py
```

Shebang za Linux i macOS

Dodati kao prvi red u fajlu

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

Zatim uključiti execute bit:

```
chmod +x myprogram.py
```

Nadalje se program može pokrenuti i ovako:

```
./myprogram.py
```

Interaktivni mod

Poziv Python interpretera:

```
$ python
```

```
Python 3.7.4 (default, Jul  9 2019, 18:13:23)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1
>>> y = x + 1
>>> y
2
```

Osnovne konstrukcije

Hello world u Javi:

```java
// Hello World in Java.
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Hello world u Pajtonu:

```python
# Hello World in Python.
print("Hello World")
```

Format reda

- naredba se ne završava sa tačka-zarez
- kraj linije je i kraj naredbe

```
print("Hello World!")
print("How are you today?")
```

- u Javi naredba se završava sa tačka-zarez i može se protezati kroz više linija:

```
// Java statement
System.out.print
  (
    "Hello World!"
    )
;
```

Format reda

- naredba se može protezati kroz više redova
- backslash za nastavak u sledećem redu

```
result = (someValue * 5 + anotherValue * 12) \
         - (originalValue * 2)
name = "John "\
       "Smith"
```

- poziv funkcija ima zagrade, pa se naredba ne mora nastavljati sa backslash

```
myFunction(a, b,
           "name",
           avg)
```

Komentari

- komentar počinje znakom hash i pruža se do kraja reda
- backslash za nastavak u sledećem redu

```
# This is a comment.
result = 0    # so is this
```

Blok naredbi

- umesto vitičastih zagrada blok se definiše uvlačenjem redova

```
while i <= 20:
    total = total + i
    i = i + 1
print("The total =", total)
```

- ovakav primer u Javi glasio bi

```
while (i <= 20) {
  total = total + i;
  i = i + 1;
}
System.out.println("The total = " + total);
```

Uvlačenje naredbi

- naredbe iza kojih sledi blok imaju dvotačku na kraju
- samo naredbe u bloku mogu biti dodatno uvučene
- top-level naredbe ne smeju biti uvučene

- nije važan broj razmaka prilikom uvlačenja
- važno je da su sve naredbe uvučene za isti broj razmaka
- prva naredba sa drugačijim uvlačenjem označava kraj bloka
- u interaktivnom modu, prazan red isto završava blok

- preporuka iz PEP8: **uvlačenje za 4 razmaka**
- uvek razmak, nikad tab

Identifikatori

- case sensitive
- mogu da sadrže slova, cifre, underscore
- ne smeju početi cifrom

- rezervisane reči

```
and       assert    break     class     continue
def       del       elif      else      except
exec      finally   for       from      global
if        import    in        is        lambda
not       or        pass      print     raise
return    try       while
```

Identifikatori

- nazivi tipova i ugrađenih funkcija
- mogu se redefinisati
- ali to je loša ideja

```
float
int
str
sum
max
```

Tipovi podataka

- svi tipovi podataka su objekti
- nema razlike između primitivnih tipova i objekata kao u Javi

```
x = 1
y = 2L   # samo u Python 2.x
z = "tralala"
p = 0.1
q = 1e-2   # 0.01
r = 3 + 4j
```

Tipovi podataka

Python 3	**Java**	**napomena**
`int`	`long`	`int` se uglavnom ponaša kao Java `long`, ali nema ograničenje na veličinu broja!
`float`	`double`	64-bitni IEEE 754
`complex`	—	kompleksni brojevi
`bool`	`boolean`	`True` ili `False`
`str`	`String`	nepromenljivi nizovi karaktera, jednostruki ili dvostruki navodnici
`list`	`T[]`	nizovi: auto-resize, heterogeni
`tuple`	—	niz koji se ne može menjati nakon kreiranja
`dict`	—	asocijativna mapa (rečnik)
`set`	—	neuređena kolekcija objekata, bez duplikata

Promenljive

- svi tipovi podataka su objekti
- čuvaju se reference na objekte
- nema deklaracije promenljivih; one se kreiraju prilikom prve dodele

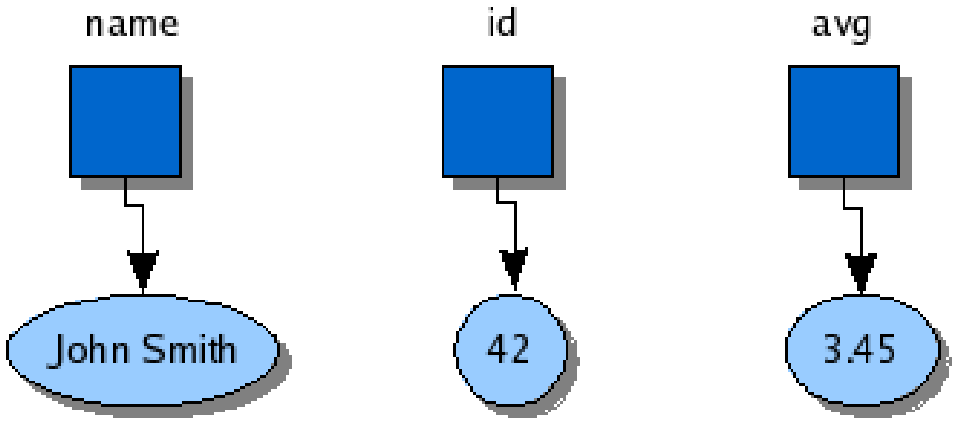
```
name = "John Smith"
id = 42
avg = 3.45
```

- Java ekvivalent

```
String name = "John Smith";
int id = 42;
double avg = 3.45;
```

Promenljive

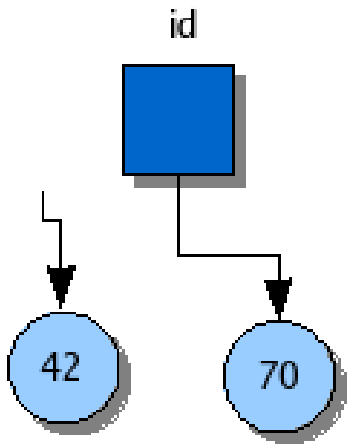
- promenljiva nema tip
- može čuvati referencu na bilo koji objekat



Dodela vrednosti

- kada se nova referenca dodeli promenljivoj, stara referenca se gubi

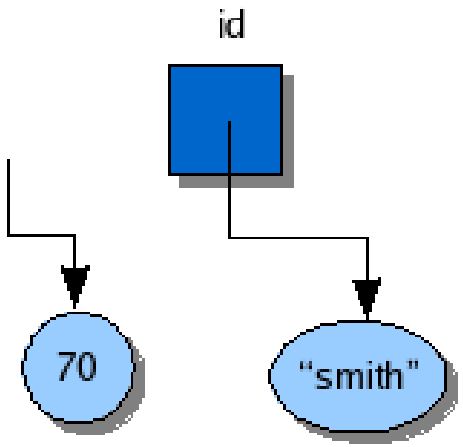
```
id = 70
```



Dodela vrednosti

- nova referenca može biti na objekat drugog tipa

```
id = "smith"
```



Konstante: ne postoje

- dve konstante u Javi

```
final double TAX_RATE = 0.06;
final int MAX_SIZE = 100;
```

- konvencija za promenljive kojima nećemo menjati vrednost: uppercase

```
TAX_RATE = 0.06
MAX_SIZE = 100
```

Aritmetički operatori

operator	napomena
+	sabiranje
-	oduzimanje
*	množenje
/	deljenje
**	stepenovanje
//	celobrojno deljenje
%	ostatak

- ima +=, -=, ...
- nema ++ i --

Mešanje numeričkih tipova: `1 + 2.0`

- operand manjeg ranga se konvertuje u veći rang
- rang tipova:
  `complex` > `float` > `int`

Konverzija tipova

- automatska konverzija samo za numeričke tipove unutar numeričkih izraza
- sve ostale konverzije moraju biti eksplicitne

```
x = 100
y = "abc"
z = y + str(x)
```

Glavni program i Java

```
// Sumation.java
// Compute the sum of the first 100 integer values and print
// the results.
public class Summation {
  public static void main(String[] args) {
    final int NUM_VALUES = 100;
    int summation = 0;
    int i = 0;

    while (i <= NUM_VALUES) {
      summation = summation + 1;
      i = i + 1;
    }

    System.out.println("The sum of the first " + NUM_VALUES
                       + " integers is " +  summation);
  }
}
```

Glavni program i Python

```
# summation.py
# Compute the sum of the first 100 integer values and print
# the results.

# Initialize a constant variable.
NUM_VALUES = 100

# Compute the sum.
summation = 0
i = 1
while i <= NUM_VALUES:
    summation = summation + i
    i = i + 1

# Print the results.
print("The sum of the first", NUM_VALUES,
      "integers is", summation)
```

Funkcija

- slična `static` metodi u Javi
- koriste se nezavisno od objekata
- u Pajtonu se ne definišu unutar klase
- (ali i mogu)

```
x = 100
y = "abc"
z = y + str(x)
```

Ugrađene funkcije

- neke funkcije su deo jezika kao ugrađene (built-in)
- uvek su dostupne

```
# Compute the absolute value of the integer x
y = abs(x)
```

Prenos parametara

- parametri funkcije se uvek prenose **po referenci**
- više parametara se razdvaja zarezom
- nema navođenja tipova parametara

- mora se paziti da se prilikom poziva prosledi odgovarajući tip
- funkcije se mogu napisati i tako da primaju parametre različitog tipa

Rezultat funkcije

- rezultat je uvek referenca na objekat
- ili `None`
- ako se ne navede rezultat funkcije, podrazumevano se vraća `None`

Pojam modula

- standardna biblioteka sadrži funkcije i klase organizovane u **module**
- modul je fajl sa Pajton kodom
- definicije iz drugog Pajton fajla (modula) možemo koristiti pomoću `import` naredbe

```
from math import *
y = sqrt(x)
```

- kod Jave konvencija je: jedna klasa – jedan fajl
- kod Pajtona to ne mora biti tako

Import iz modula

- možemo importovati pojedinačne komponente iz modula

```
from math import sqrt
y = sqrt(x)
```

- možemo importovati ceo modul
- njegove komponente navodimo uz ime modula

```
import random
z = random.randrange(0, 10)
```

Modul `math`

funkcija	**opis**
`ceil(x)`	najmanji ceo broj veći od x
`floor(x)`	najveći ceo broj manji od x
`sqrt(x)`	kvadratni koren od x
`sin(x)`, `cos(x)`, `tan(x)`	trigonometrijske funkcije
`degrees(x)`	pretvara radijane u stepene
`radians(x)`	pretvara stepene u radijane
...	...

Import iz modula

- klasa predstavlja klasu objekata
- objekti su instance klase
- objekti se mogu upotrebiti nakon kreiranja
- svi literali u Pajtonu rezultuju kreiranim objektima

- za kreiranje objekta direktno od klase, poziva se konstruktor kao da je nezavisna funkcija

```
from datetime import date
today = date(2019, 10, 1)
```

- kada se objekat kreira možemo pozivati njegove metode

```
whichDay = today.weekday()
```

Konstruktori za numeričke tipove

- numerički tipovi imaju konstruktore za kreiranje objekata
- mogu poslužiti i za konverziju tipova

konstruktor	**opis**
`complex(x, y)`	kreira `complex` objekat, `x` i `y` moraju biti numerički tipovi
`float(x)`	konvertuje string ili numerički tip u `float`
`int(x)`	konvertuje string ili numerički tip u `int`

Standardni ulaz

```java
// Java example
Scanner keyboard = new Scanner(System.in);
System.out.println("What is your name? ");
String name = keyboard.next();
```

```python
# Python example
name = input("What is your name? ")
```

Unos numeričkih podataka

```java
// Java sample
System.out.print("What is your GPA?");
double gpa = keyboard.nextDouble();
```

```python
# Python example
gpa = float(input("What is your GPA? "))
```

Standardni izlaz

```
avg = grade / 3.0
print(avg)
```

Funkcija `print` može primiti više parametara:

```
print("Your average grade =", avg)
```

Funkcija `print` ne mora završiti ispis prelaskom u novi red:

```
print("Your average grade = ", end='')
print(avg)
```

Primer programa (Java)$_1$

```
/* Wages.java
 * Computes the taxes and wages for an employee given the
 * number of hours worked and their pay rate.
*/
import java.util.*;
public class Wages {
   public static void main(String[] args) {
      final double STATE_TAX_RATE = 0.035;
      final double FED_TAX_RATE = 0.15;
      double hours, payRate;
      double wages, stateTaxes, fedTaxes, takeHome;
      String employee;
      Scanner keyboard = new Scanner( System.in );
      System.out.print( "Employee name: " );
      employee = keyboard.next();
      System.out.print( "Hours worked: " );
      hours = keyboard.nextDouble();
      System.out.print( "Pay rate: " );
      payRate = keyboard.nextDouble();
```

Primer programa (Java)$_2$

```
        wages = hours * payRate;
        stateTaxes = wages * STATE_TAX_RATE;
        fedTaxes = wages * FED_TAX_RATE;
        takeHome = wages - stateTaxes - fedTaxes;
        System.out.println( "PAY REPORT" );
        System.out.println( "Employee: " + employee );
        System.out.println( "--------------------------------" );
        System.out.println( "Wages:       " + wages );
        System.out.println( "State Taxes: " + stateTaxes );
        System.out.println( "Fed Taxes:   " + fedTaxes );
        System.out.println( "Pay:         " + takeHome );
    }
}
```

Primer programa (Python)

```
# wages.py
# Computes the taxes and wages for an employee given the
# number of hours worked and their pay rate.
STATE_TAX_RATE = 0.035
FED_TAX_RATE = 0.15
employee = input("Employee name: ")
hours = float(input("Hours worked: "))
payRate = float(input("Pay rate: "))
wages = hours * payRate
stateTaxes = wages * STATE_TAX_RATE
fedTaxes = wages * FED_TAX_RATE
takeHome = wages - stateTaxes - fedTaxes
print "PAY REPORT"
print "Employee: ", employee
print "---------------------------------"
print "Wages:        ", wages
print "State Taxes: ", stateTaxes
print "Fed Taxes:   ", fedTaxes
print "Pay:          ", takeHome
```

Stringovi

- stringovi su instance klase `str`
- literal predstavlja referencu na objekat u memoriji

```
'string'
"string's"
```

Pojavljivanje literala je kao poziv konstruktora. Sledeća dva reda daju isti rezultat.

```
name = "John Smith"
name = str('John Smith')
```

Konverzija u string

- string konstruktor se može upotrebiti i za konverziju tipova

```
x = 45
intStr = str(x)           # '45'
floatStr = str(56.89)     # '56.89'
boolStr = str(False)      # 'False'
```

Escape sekvence

- vrlo slično kao u Javi

```
msg = 'Start a newline here.\nusing the \\n character.'
```

sekvenca	rezultat
\\	backslash (\)
\'	jednostruki navodnik (')
\"	dvostruki navodnik (")
\n	line feed karakter (LF, 10)
\t	tab karakter (HT, 9)

Višelinijski stringovi

- čuva se sav whitespace

```
"""This is a string which
can continue onto a new line. When printed, it will appear
   exactly as written between the trip quotes.
"""

'''Here is another
        multiline string example
     using triple single quotes.'''
```

Konkatenacija

```
strvar = 'This is '
fullstr = strvar + "a string"
```

- nema automatske konverzije tipova u string prilikom konkatenacije

```
result = "The value of x is " + str(x)
```

- konkatenacija može i bez operatora ako su dva stringa jedan do drugog

```
print("These two string literals " "will be concatenated.")
```

Dužina stringa

```java
// Java string length
System.out.println("Length of the string = " + name.length());
```

```python
# Python string length
print("Length of the string =", len(name))
```

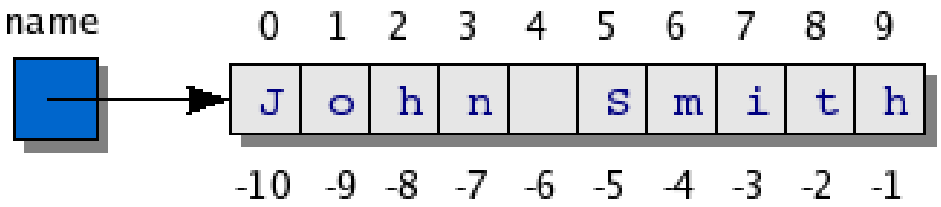
Pristup znakovima u stringu po indeksu

```java
// Java character access
String msg = "This is a string";
System.out.println( "The first character is " +
                    msg.charAt( 0 ) );
System.out.println( "The last characater is " +
                    msg.charAt( msg.length() - 1 ) );
```

```python
# Python character access
msg = "This is a string!"
print("The first character is", msg[0])
print("The last character is", msg[len(msg) - 1])
```

Negativni indeksi u nizu

```
# poslednji znak u stringu
print("The last character is", msg[-1])
```



Podstringovi

```java
// Java substring extraction
String name = "John Smith";
String first = name.substring(0, 4);  // "John"
String last = name.substring(5);      // "Smith"
```

Operator isecanja (*slicing*) u Pajtonu postiže isti rezultat:

```python
# Python substring extraction using slicing
name = "John Smith"
first = name[0:4]
last = name[5:]
```

Umnožavanje stringova

```
print("-----------------------------------------")
```

je isto što i

```
print("-" * 45)
```

Formatiranje stringa

- operator ostatka (%) je definisan za stringove
- kreira novi string nalik `printf` metodi u Javi

```
output = "The average grade = %5.2f" % avgGrade
print(output)
```

- može i sa više vrednosti

```
print("Origin: (%d, %d)\n" % (pointX, pointY))
```

Formatiranje stringa

opšta struktura formata je

`%[flags][width][.precision]code`

segment	**opis**
`flags`	vodeće nule (0) i poravnanje po levoj (-) ili desnoj (+) ivici
`width`	broj znakova za prikaz
`precision`	broj decimala
`code`	specifikacija formata

Format specifier

kod	**opis**
`%s`	string ili drugi objekat
`%c`	karakter (iz ASCII vrednosti)
`%d`	celobrojna vrednost
`%i`	celobrojna vrednost (isto kao `%d`)
`%u`	neoznačeni ceo broj
`%o`	oktalni ceo broj
`%x`	heksadecimalni ceo broj
`%X`	kao `%x` samo uppercase
`%e`	float sa eksponentom
`%E`	kao `%e` samo uppercase
`%f`	float bez eksponenta
`%g`	isto kao `%e` ili `%f`
`%G`	kao `%g` samo uppercase
`%%`	literal %

Primer sa formatiranjem stringova

```
# Computes the taxes and wages for an employee given the
# number of hours worked and their pay rate. The results
# are printed using formatted strings.
STATE_TAX_RATE = 0.035
FED_TAX_RATE = 0.15
employee = input("Employee name: ")
hours = float(input("Hours worked: "))
payRate = float(input("Pay rate: "))
wages = hours * payRate
stateTaxes = wages * STATE_TAX_RATE
fedTaxes = wages * FED_TAX_RATE
takeHome = wages - stateTaxes - fedTaxes
print("PAY REPORT")
print("Employee: %s" % employee)
print("---------------------------------")
print("Wages:       %8.2f" % wages)
print("State Taxes: %8.2f" % stateTaxes)
print("Fed Taxes:   %8.2f" % fedTaxes)
print("Pay:         %8.2f" % takeHome)
```

Tip bool

- dve vrednosti, `True` i `False`
- numeričke vrednosti jednake nuli tumače se kao `False`, inače kao `True`
- za druge objekte, ne-prazni objekti tumače se kao `True`, prazni kao `False`
- funkcija `bool(x)` poštuje ova pravila za konverziju

```
x = bool("")      # False
y = bool(0)       # False
z = bool(17.2)    # True
w = bool("abc")   # True
```

Logički izrazi

- rezultat je `bool` vrednost
- relacioni operatori: $<$, $>$, $<=$, $>=$, $==$, $!=$, $<>$
- logički operatori: `and`, `or`, `not`

Poređenje i reference

- Java: operator == poredi reference, a ne vrednosti

```
String name1 = "Smith";
String name2 = "Jones";
if (name1 == name2)
  System.out.println("Name1 and Name2 are aliases.");
```

- Python: operator == poredi vrednosti
- poređenje objekata podrazumeva poređenje svih atributa
- operator `is` poredi reference

```
name1 = "Smith"
name2 = "Jones"
name3 = name1
result1 = name1 is name2  # False
result2 = name1 is name3  # True
```

Null referenca

- null referenca je predstavljena posebnim izrazom `None`

```
# The is operator along with None can tests for null references.
result1 = name1 is None
result2 = name1 == None
```

Naredba if

- nema zagrade oko uslova
- dvotačka pre bloka naredbi
- blok može imati jednu naredbu, ali je i dalje blok

```
import math

value = int(input("Enter a value: "))

if value < 0:
    print("The value is negative. Converting to positive.")
    value = abs(value)

sqroot = math.sqrt(value)

print("The square root of", value, "is", sqroot)
```

if-else

- `else` takođe ima svoj blok naredbi
- dakle, i dvotačku

```
if value < 0:
    print("The value is negative.")
    value = abs(value)
else:
    print("The value is positive.")
```

Višestruko grananje

- ne postoji `switch`
- koristi se `if - elif - else` konstrukcija
- uslovi se testiraju sekvencijalno, do prvog ispunjenog ili kraja

```
if avgGrade >= 90.0 :
    letterGrade = "A"
elif avgGrade >= 80.0 :
    letterGrade = "B"
elif avgGrade >= 70.0 :
    letterGrade = "C"
elif avgGrade >= 60.0 :
    letterGrade = "D"
else:
    letterGrade = "F"
```

Poređenje stringova

- Java: pomoću metoda klase String

```
String str1 = "Abc Def";
String str2 = "Abc def";
if (str1.equals(str2))
  System.out.println("Equal!!");
else
  System.out.println("Not Equal!!");
```

- Python: pomoću običnih operatora

```
str1 = "Abc Def"
str2 = "Abc def"
if str1 == str2:
    print "Equal!!"
else:
    print "Not Equal!!"
```

Podstring u stringu

- Java: metoda contains

```
if (name1.contains("Smith"))
  System.out.println(name1);
```

- Python: operator in

```
if "Smith" in name1:
    print name1
if "Jones" not in name1:
    print "Name not found"
```

While petlja

- i za *event-controlled* i za *count-controlled* petlje

```
theSum = 0
i = 1
while i <= 100:
    theSum = theSum + i
    i = i + 1
print("The sum =", theSum)
```

- while petlja očekuje blok naredbi

```
value = 0
while value < 0:
    value = int(input("Enter a postive integer: "))
```

For petlja

- iterira kroz sekvencu bilo čega - tuple, list, str, ...

```
for <loop-var> in <object>:
    <statement-block>
```

- indeks promenljiva uzima vrednost iz sekvence

```
for i in range(1, 11):
    print(i)
```

Primer: iteracija

```
# avgvalue.py
# Reads a group of positive integer values from the user,
# one at a time, until a negative value is entered. The average
# value is then computed and displayed.

total = 0
count = 0

value = int(input("Enter an integer value (< 0 to quit): "))
while value >= 0:
    total = total + value
    count = count + 1
    value = int(input( "Enter an integer value (< 0 to quit): "))

avg = float(total) / count
print("The average of the", count, "values you entered is", avg)
```

Primer: stringovi

```
# countvowels.py
# This program iterates through a string and counts the number of
# vowels it contains.

text = input("Enter a string to be evaluated: ")

numVowels = 0
for ch in text:
    if ch in "aeiou":
        numVowels = numVowels + 1

print("There are " + str(numVowels) + " vowels in the string.")
```

Definicija funkcije

- obuhvata zaglavlje i telo
- zaglavlje počinje sa def
- telo je blok naredbi

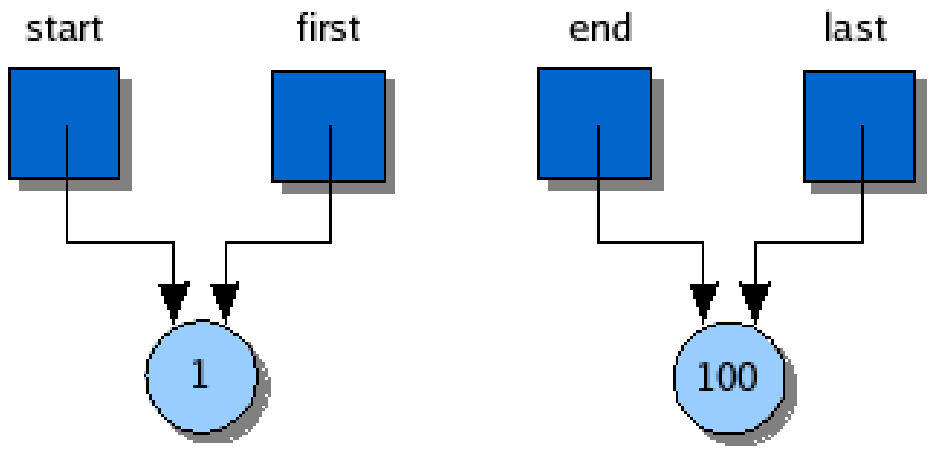
```
def sumRange(first, last):
    total = 0
    i = first
    while i <= last:
        total = total + i
        i = i + 1
    return total
```

- sopstvena funkcija poziva se kao i svaka druga

```
start = 1
end = 100
theSum = sumRange(start, end)
print("The sum of the integers between", start,
      "and", end, "is", theSum)
```

Parametri funkcije

- parametri su uvek reference na objekte
- formalni parametri pokazuju na iste objekte na koje i stvarni



- vrednost parametra može se promeniti tako da promena bude vidljiva po povratku iz funkcije
- ako je parametar promenljivi objekat

Tipizacija parametara i polimorfizam

- tipovi parametara se ne navode
- funkcija se može pozvati za parametre različitih tipova
- na primer, `float` umesto `int`

```
theSum = sumRange(1.37, 15.78)
```

Podrazumevane vrednosti parametara

```
def sumRange2(first, last, step=1) :
    total = 0
    i = first
    while i <= last :
        total = total + i
        i = i + step
    return total
```

- može se pozvati na više načina:

```
theSum = sumRange2(1, 100)
theSum = sumRange2(1, 100, 2)
```

Podrazumevane vrednosti parametara

- svi parametri nakon prvog koji ima default moraju imati default
- ovo je nemoguće:

```
def foo(argA, argB=0, argC):
    print(argA, argB, argC)
```

Imenovani parametri u pozivu

- redosled parametara ne mora se poštovati ako se parametri imenuju

```
theSum = sumRange2(last = 100, step = 3, first = 1)
```

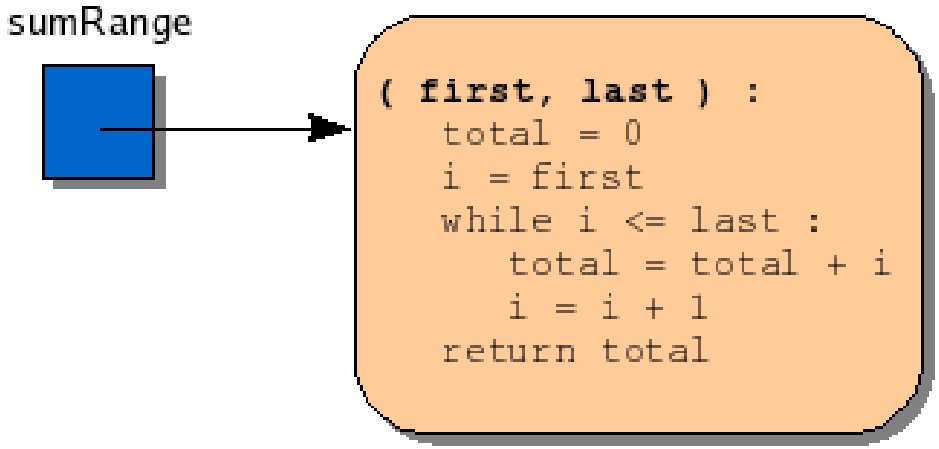
Rezultat funkcije

- svaka funkcija vraća vrednost
- ako ne definišemo povratnu vrednost, vraća se `None`
- moguće je vratiti više vrednosti odjednom smeštajući ih u tuple

```
def powr(x):
    return x, x**2, x**3, x**4
```

Funkcije su objekti

- `def` je naredba koju izvršava interpreter
- kreira funkcijski objekat
- naziv funkcije je zapravo promenljiva koja pokazuje na objekat



Funkcije su objekti

- naredbe u telu funkcije se ne izvršavaju dok se funkcija ne pozove
- bez obzira što se naredbe interpretiraju i konvertuju u bajt-kod
- može se navesti poziv funkcije pre njene definicije

```
def run():
    value = int(input("Enter a value: "))
    print("The double of your value is",
          doubleIt(value))

def doubleIt(num):
    return num * 2

run()
```

Poziv funkcije pre definicije — izuzetak

- poziv funkcije pre njene definicije nije moguć na globalnom nivou
- jer se tada još nije izvršila `def` naredba

```
# does not work
callIt(5)

def callIt(num):
    return pow(num, 2)
```

Redosled funkcija

- redosled funkcija u fajlu nije bitan
- redosled naredbi na globalnom nivou jeste
- zgodan recept
  1. naredbe na globalnom nivou stavi u posebnu funkciju
  2. pozovi je na kraju fajla

Redosled funkcija: primer

```
from random import *
MIN_SIDES = 4

def main():
    print("Dice roll simulation.")
    numSides = int(input( "How many sides should the die have? "))

    if numSides < MIN_SIDES :
        numSides = MIN_SIDES
    value = rollDice(numSides)
    print("You rolled a", value)

def rollDice(nSides):
    die1 = randint(1, nSides + 1)
    die2 = randint(1, nSides + 1)
    return die1 + die2

main()
```

Opseg vidljivosti promenljivih

- built-in: promenljive i literali definisani na nivou jezika; uvek dostupni
- global ili module: promenljive kreirane na najvišem nivou u fajlu (izvan funkcija i klasa); dostupne samo u okviru svog modula
- local: promenljive kreirane u okviru funkcije su vidljive samo unutar funkcije; parametri su lokalni
- instance: promenljive definisane kao atributi objekta

Upotreba globalnih promenljivih

- mogu se koristiti unutar funkcija
- primer:

```
varA = 40

def one(varB):
    varC = varA + varB
    return varC

print(varA, one(20))
```

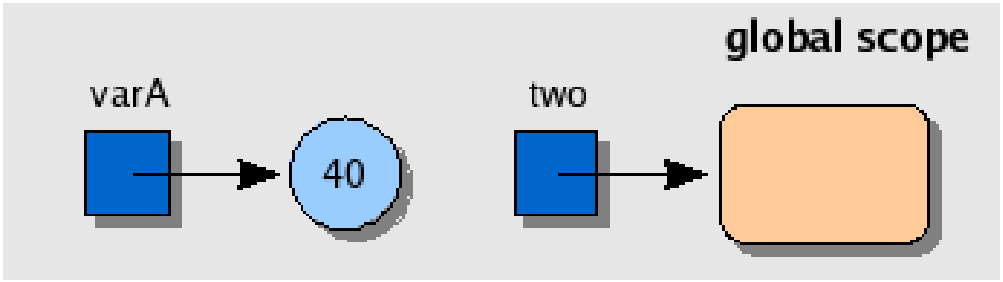
Upotreba globalnih promenljivih

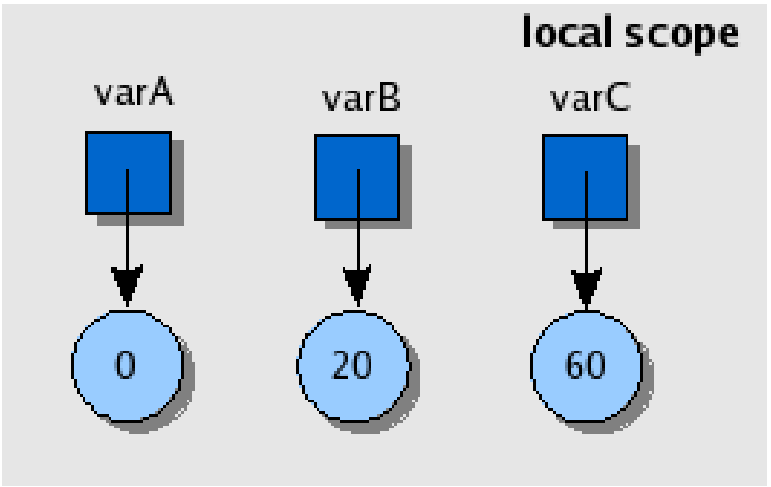
- šta ako pokušamo da menjamo vrednost globalne promenljive unutar funkcije?
- primer:

```
varA = 40

def one(varB):
    varC = varA + varB
    varA = 0
    return varC

print(varA, one(20))
```

Upotreba globalnih promenljivih

- dodela `varA = 0` unutar funkcije će kreirati novu lokalnu promenljivu





Izmena globalnih promenljivih unutar funkcije

- moramo da naglasimo da koristimo globalnu promenljivu
- `global` naredba

```
varA = 40

def one(varB):
    global varA
    varC = varA + varB
    varA = 0
    return varC

print(varA, one(20))
```

Moduli

- module možemo koristiti da grupišemo srodne klase i funkcije
- veći program se sastoji od početnog fajla i dodatnih modula (fajlova)

Primer upotrebe modula 1

```
# driver.py
# The driver file which contains the statements creating
# the main flow of execution.

import modA
import modB

value1 = int(input("Enter value one: "))
value2 = int(input( "Enter value two: "))
resultA = modA.funcA(value1, value2)
resultB = modB.funcB(resultA)
print("Results = ", resultA, resultB)
```

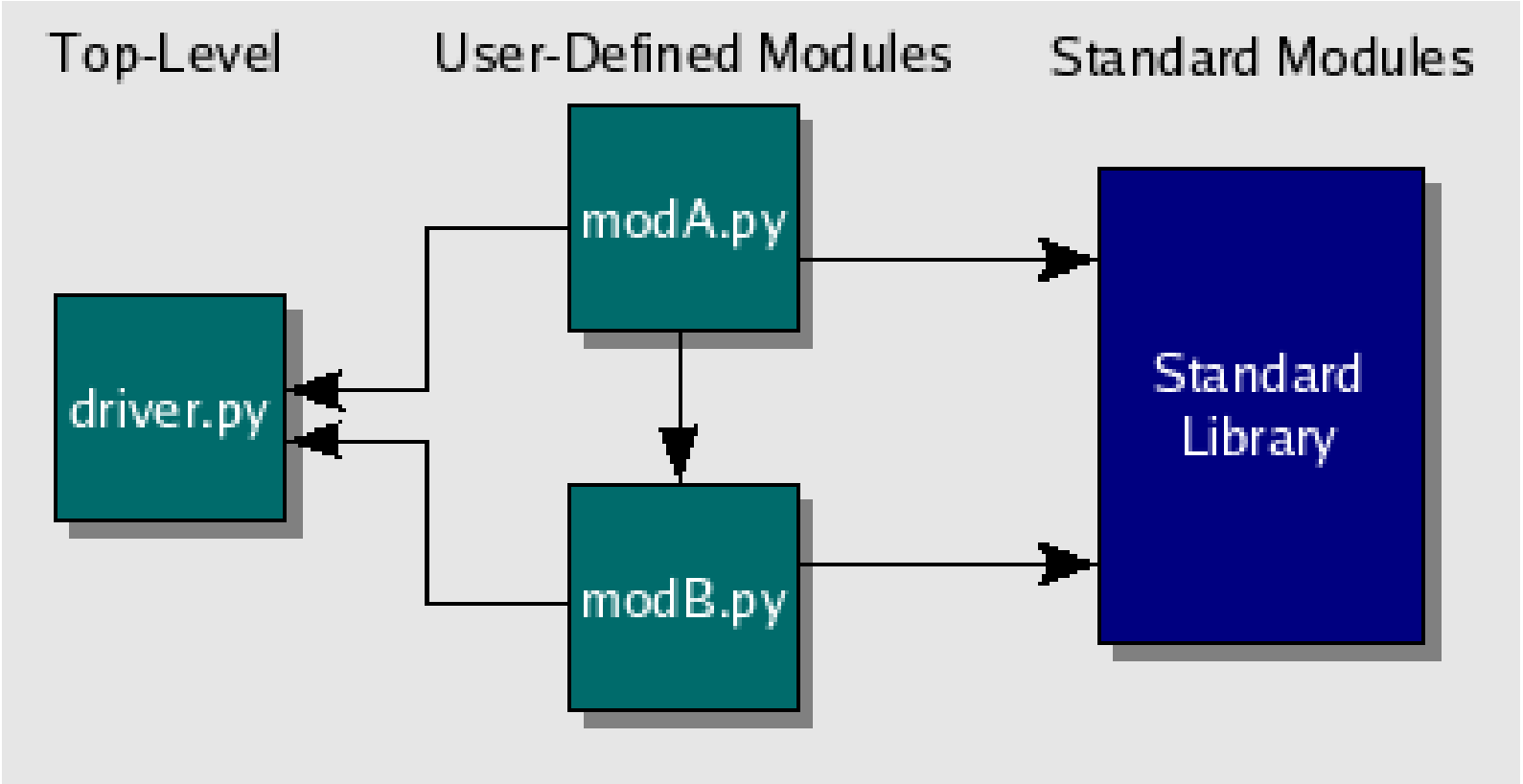
Primer upotrebe modula 2

```
# modA.py
# A user-defined module which defines a function and
# imports a standard module.

from math import *

def funcA(x, y):
    d = sqrt(x * x + y * y)
    return d
```

Primer upotrebe modula 3

```
# modB.py
# A second user-defined module which defines a function and
# imports the other user-defined module as well as a
# standard module.

from modA import *

def funcB(a) :
    b = funcA(a, a * 4)
    return b / 3 + funcC()

def funcC(x = 0):
    return x * 12
```

Primer upotrebe modula 4



Primer upotrebe modula 5

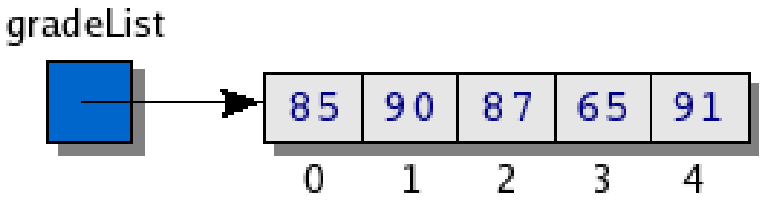
- `modA` je importovan dva puta: od strane `driver` i `modB`
- modul se interpretira jednom, bez obzira što se može importovati više puta!

Python lista

- terminologija: Python lista je zapravo niz
- čuva reference na objekte (bilo kog tipa)
- nema fiksnu veličinu
- automatsko proširivanje

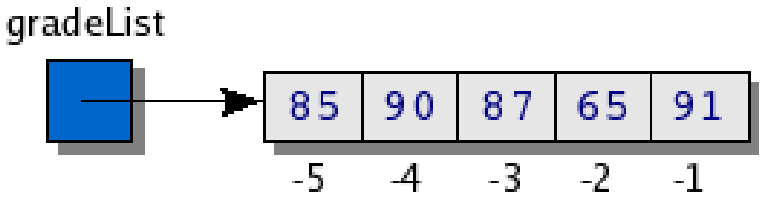
```
// Java niz
int[] gradeList = {85, 90, 87, 65, 91};
```

```
# Python lista
gradeList = [85, 90, 87, 65, 91]
```



Negativni indeksi

- elementima liste može se pristupiti i preko negativnih indeksa
- indeks -1 predstavlja poslednji element liste



Indeks izvan opsega

- Java: `ArrayIndexOutOfBoundsException`
- Python: `IndexError`

Inicijalizacija liste

```java
// inicijalizacija Java niza
int[] gradeHist = new int[10];
for(int i = 0; i < gradeHist.length; i++)
  gradeHist[i] = 0;
```

```python
# inicijalizacija Python liste
gradeHist = []
for i in range(10):
    gradeHist.append(0)

# može i mnogo kraće
gradeHist = [0] * 10
```

Iteracija kroz listu

```
total = 0
for value in valueList:
    total = total + value
avg = float(total) / len(valueList)
print("The average value = %6.2f" % avg)
```

Izmena elementa liste

```
# initialize list
x = [0] * 10

# change element at index 3
x[3] = 1

# element does not have to be an int
x[6] = "test"
x[8] = [1, 2, 3]

# slice out elements 3-5 in a new list
y = x[3:6]
```

Tuple: nepromenljiva lista

```
t = (0, 2, 4)             # 3 element tuple
a = (2,)                  # 1 element tuple
b = ('abc', 1, 4.5, 5)    # 4 element mixed tuple
c = (0, ('a', 'b'))       # nested tuple
```

Operacije nad listom

operacija	opis
`a = []`	Kreira praznu listu; kao kreiranje novog `ArrayList` objekta u Javi
`b = [0, 1, 2]`	Kreira i inicijalizuje listu pomoću literala
`b[i]`	Pristupa `i`-tom elementu liste `b`
`b * 4`	Kreira novu listu ponavljanjem liste `b` 4 puta
`a + b`	Konkatenira liste `a` i `b` u novu listu
`del b[i]`	Uklanja `i`-ti element iz liste `b`
`for x in list`	Iterira kroz elemente liste
`x in list`	Određuje da li se `x` nalazi u listi
`len(b)`	Vraća dužinu liste `b`
`b.append(obj)`	Dodaje `obj` na kraj liste `b`
`b.extend(a)`	Dodaje sve elemente liste `a` na kraj liste `b`
`b.index(x)`	Traži objekat `x` u listi `b`, vraća njegovu poziciju ili izuzetak
`b.reverse()`	Obrće redosled elemenata liste `b`
`b.sort()`	Sortira elemente liste `b`

Tekstualni fajlovi: otvaranje i zatvaranje

```
infile = open('records.txt', 'r')     # read from file
outfile1 = open('report1.txt', 'w')   # write to file (start from empty)
outfile2 = open('report2.txt', 'a')   # append to file

infile.close()
outfile1.close()
outfile2.close()
```

with naredba

```
infile = open('records.txt', 'r')
# some statements with infile
infile.close()
```

je ekvivalentno sa

```
with open('records.txt', 'r') as infile:
    # some statements with infile
```

Čitanje iz fajla

Čitanje fajla red po red:

```
with open('records.txt', 'r') as infile:
    while (line = infile.readline()) != "":
        print(line)

# kraće
with open('records.txt', 'r') as infile:
    for line in infile:
        print(line)
```

Čitanje celog fajla u niz redova:

```
all_lines = infile.readlines()
```

Čitanje celog fajla u jedan string:

```
text = infile.read()
```

Pisanje u fajl

```
contents = ['Text', 'Text2', 'Text3']

with open('output.txt', 'w') as outfile:
    for line in contents:
        outfile.write(line + '\n')
```

Rečnik

- rečnik (dictionary), za razliku od liste, ne poznaje redosled elemenata
- elementi rečnika nisu dostupni putem indeksa, već putem ključa
- rečnik čuva parove ključ-vrednost

```
>>> lozinke = {"guido":"superprogrammer",
        "turing":"genius", "bill":"monopoly"}
>>> lozinke["guido"]
'superprogrammer'
>>> lozinke["bill"]
'monopoly'
```

- šta je ovde ključ, a šta vrednost?

Rečnik 2

- rečnici se mogu menjati

```
>>> lozinke["bill"] = "bluescreen"
>>> lozinke
{'turing': 'genius', 'bill': 'bluescreeen', 'guido': \
 'superprogrammer'}
```

- originalni redosled nije očuvan!

Rečnik 3

- ključ može biti bilo koji immutable tip - broj, string, tuple
- vrednost može biti bilo kog tipa

- rečnici se retko javljaju u drugim programskim jezicima
- u Pythonu se često koriste
- efikasni su, mogu primiti stotine hiljada elemenata

Dodavanje elemenata u rečnik

- rečnik može biti prazan: {}
- dodavanje elementa u rečnik: upotrebi se novi ključ

```
>>> lozinke["newuser"] = "jasamnovi"
>>> lozinke
{'turing': 'genius', 'bill': 'bluescreeen', 'guido': \
 'superprogrammer', 'newuser': 'jasamnovi'}
```

Primer: učitavanje rečnika iz fajla

```
passwd = {}
with open('passwords', 'r') as infile:
    for line in infile:
        user, passw = line.split()
        passwd[user] = passw
```

Operacije nad rečnikom

operacija	**značenje**
`<dict>.has_key(<key>)`	vraća `True` ako postoji dati ključ
`<dict>.keys()`	vraća listu ključeva
`<dict>.values()`	vraća listu vrednosti
`<dict>.items()`	vraća listu tupli `(key,val)`
`del <dict>[<key>]`	uklanja element iz rečnika
`<dict>.clear()`	uklanja sve elemente iz rečnika

Primer: Java klasa

```
public class Point {
  private int xCoord;
  private int yCoord;
  public Point() { xCoord = yCoord = 0; }
  public Point(int x, int y) {
    xCoord = x;
    yCoord = y;
  }
  public String toString() {
     return "(" + xCoord + ", " + yCoord + ")";
  }
  public int getX() { return xCoord; }
  public int getY() { return yCoord; }
  public void shift(int xInc, int yInc) {
    xCoord = xCoord + xInc;
    yCoord = yCoord + yInc;
  }
}
```

Primer: Python klasa

```
class Point:
    def __init__(self, x=0, y=0):
        self.xCoord = x
        self.yCoord = y

    def __str__(self):
        return "(" + str(self.yCoord) + ", " +
                str(self.yCoord) + ")"

    def getX(self):
        return self.XCoord

    def getY(self):
        return self.yCoord

    def shift(self, xInc, yInc):
        self.xCoord += xInc
        self.yCoord += yInc
```

Klase — Java vs Python

- nema modifikatora pristupa

- prvi parametar metoda mora biti referenca na objekat
- konvencija je da se taj objekat zove `self`
- `this` (Java) $\Leftrightarrow$ `self` (Python)

- konstruktor je `__init__`
- ne može postojati više konstruktora

- Bruce Eckel vs Guido van Rossum:
  Why explicit self has to stay
  `http://neopythonic.blogspot.com/2008/10/why-explicit-self-has-to-stay.html`

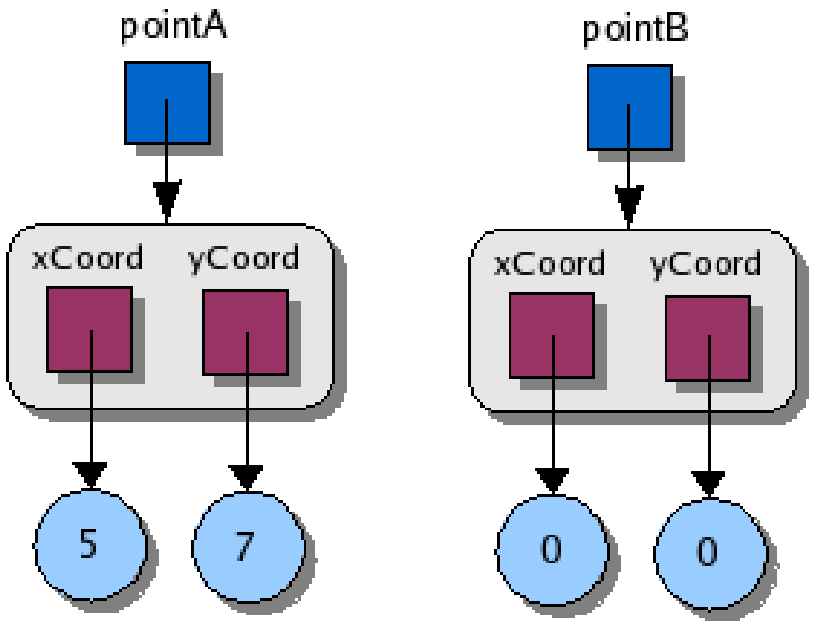
Klase — Java vs Python

- Java: jedna klasa – jedan fajl, isto ime klase i fajla

- Python: ne mora tako
- modul može imati 0, 1 ili više klasa
- fajl se ne mora zvati kao klasa

Kreiranje objekata

```
from point import *

pointA = Point(5, 7)
pointB = Point()
```



Metode

- metoda mora biti definisana unutar klase
- prvi parametar mora biti `self`

- poziv metode:

```
pointA.shift(10, 15)
```

- zapravo se tumači kao:

```
Point.shift(pointA, 10, 15)
```

Konstruktor

- može biti samo jedan konstruktor u klasi
- zove se `__init__`
- parametri konstruktora mogu imati default vrednosti
- konstruktor nije obavezan

Atributi

- nema eksplicitnog definisanja atributa!
- atribut objekta se može dodati u konstruktoru ili drugim metodama – u toku izvršavanja programa
- $\Rightarrow$ ne moraju sve instance klase imati isti skup atributa!

- tipična (ne i obavezna) inicijalizacija atributa je u konstruktoru:

```
def __init__(self, x=0, y=0):
    self.xCoord = x
    self.yCoord = y
```

Atributi

- nema eksplicitnog definisanja atributa!
- atribut objekta se može dodati u konstruktoru ili drugim metodama – u toku izvršavanja programa
- $\Rightarrow$ ne moraju sve instance klase imati isti skup atributa!

- tipična (ne i obavezna) inicijalizacija atributa je u konstruktoru:

```
def __init__(self, x=0, y=0):
    self.xCoord = x
    self.yCoord = y
```

Magične metode

metoda	**opis**
`__eq__(self, other)`	implementacija operatora ==
`__lt__(self, other)`	implementacija operatora <
`__le__(self, other)`	implementacija operatora <=
`__ne__(self, other)`	implementacija operatora !=
`__add__(self, other)`	implementacija operatora +
`__iadd__(self, other)`	implementacija operatora +=
`__int__(self)`	konverzija u `int`
`__str__(self)`	konverzija u `str`
`__repr__(self)`	konverzija u machine-readable tekst
`__hash__(self)`	izračunavanje heša za objekat
`__len__(self)`	implementacija funkcije len(obj)
`__getitem__(self, key)`	pristup po ključu `obj[key]`
`__contains__(self, key)`	implementacija operatora `in`
`__iter__(self)`	implementacija iteratora kroz elemente
...	...

Statički atributi

- definisani unutar klase, ali ne unutar metode

```
class MyClass:
    i = 3

print(MyClass.i)  # 3

obj1 = MyClass()
print(obj1.i)     # 3

obj2 = MyClass()
obj2.i = 4
print(obj1.i)     # 4
```

Statičke metode

- dekorator `@staticmethod` ili `@classmethod`

```
class MyClass:

    @staticmethod
    def f1(arg1, arg2):
        ...

    @classmethod
    def f2(cls, arg1, arg2):
        ...

MyClass.f1(0, 0)
MyClass.f2(0, 0)

obj = MyClass()
obj.f1(0, 0)
obj.f2(0, 0)
```

Izuzeci

- opšta forma:

```
try:
    # possible errors here
except IndexError as ex:
    # handle this type of exception here
except RuntimeError as ex:
    # handle another type of exception here
except:
    # handle all other types of exceptions
else:
    # executed when there are no exceptions
finally:
    # cleanup actions that must always be executed
```

Izuzeci: primer

```
try:
    x = 0
    myList = [12, 50, 5, 17]
    print(myList[3] / x)

except IndexError as ex:
    print("Error: Index out of range.")
except ZeroDivisionError as ex:
    print("Error: Attempt to divide by zero.")
except (RuntimeError, TypeError, NameError):
    pass  # do nothing
except:
    print("Error: An exception was raised.")
```

Kreiranje izuzetaka

```
def min(value1, value2):
    if value1 == None or value2 == None:
        raise TypeError  # raise exception here
    if value1 < value2:
        return value1
    else:
        return value2
```

Naredba assert

```
def average(valueList):
    total = 0
    # if fails, raises AssertionError:
    assert len(valueList) > 0, "Empty list in average()."
    for x in valueList:
        total = total + x

    avg = total / len(valueList)
    return avg
```

Nasleđivanje: roditelj

```
class Fish:
    def __init__(self, first_name, last_name="Fish",
                 skeleton="bone", eyelids=False):
        self.first_name = first_name
        self.last_name = last_name
        self.skeleton = skeleton
        self.eyelids = eyelids

    def swim(self):
        print("The fish is swimming.")

    def swim_backwards(self):
        print("The fish can swim backwards.")
```

Nasleđivanje: dete

```
class Trout(Fish):
    pass

terry = Trout("Terry")
print(terry.first_name + " " + terry.last_name)
print(terry.skeleton)
print(terry.eyelids)
terry.swim()
terry.swim_backwards()
```

Nasleđivanje: dete sa dodatnim metodama

```
class Clownfish(Fish):
    def live_with_anemone(self):
        print("The clownfish is coexisting with sea anemone.")


casey = Clownfish("Casey")
print(casey.first_name + " " + casey.last_name)
casey.swim()
casey.live_with_anemone()
```

Nasleđivanje: redefinisanje metoda

```
class Shark(Fish):
    def __init__(self, first_name, last_name="Shark",
                 skeleton="cartilage", eyelids=True):
        self.first_name = first_name
        self.last_name = last_name
        self.skeleton = skeleton
        self.eyelids = eyelids

    def swim_backwards(self):
        print("The shark cannot swim backwards, "
              "but can sink backwards.")
```

Poziv super()

```
class Trout(Fish):
    def __init__(self, first_name, water = "freshwater"):
        self.water = water
        super().__init__(self, first_name)
```

Višestruko nasleđivanje

```
class Coral:
    def community(self):
        print("Coral lives in a community.")

class Anemone:
    def protect_clownfish(self):
        print("The anemone is protecting the clownfish.")

class CoralReef(Coral, Anemone):
    pass
```