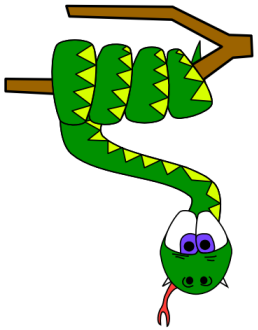


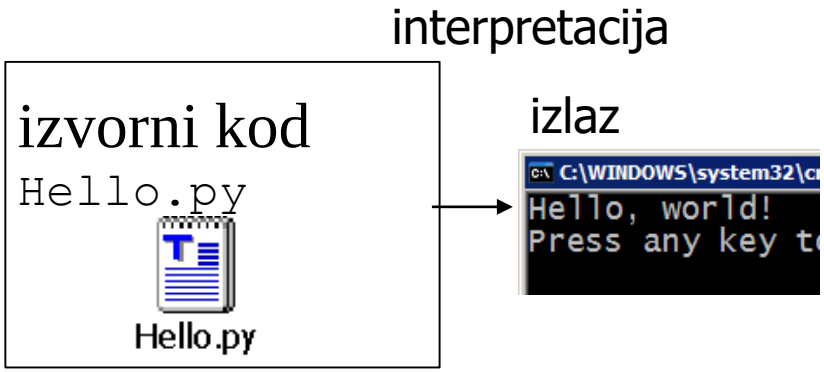
Uvod u Python (I dio)

Slavica Tomović (slavicat@ucg.ac.me)
Elektrotehnički fakultet, Podgorica
Univerzitet Crne Gore

Kompajliranje koda

- Mnogi jezici zahtijevaju kompajliranje programa u formu koju mašina razumije.

- Python se direktno interpretira u mašinske instrukcije.



Sintaksa

- **izrazi**: Set operacija za izračunavanje neke vrijednosti.
  Primjer: `1 + 4 * 3`

- **Aritmetički operatori**:

`+ - * /`	sabiranje, oduzimanje/negacija, množenje, dijeljenje
`%`	ostatak pri dijeljenju
`**`	eksponent

- **prioritet**: Redosled izvršavanja operacija
  `* / % **` imaju veći prioritet u odnosu na +–

  `1 + 3 * 4` je `13`

- **Kada dijelimo cijele brojeve rezultat je takođe cijeli broj**
  - Primjer:
    - `35 / 5` je `7`
    - `84 / 10` je `8`
    - `156 / 100` je `1`

Korisne matematičke komande

- Python nudi bogat set komandi za obavljanje raznih operacija.

Naziv komande	Opis
`abs(`***vrijednost***`)`	apsolutna vrijednost
`ceil(`***vrijednost***`)`	zaokruživanje na veću cjelobrojnu vr.
`cos(`***vrijednost***`)`	kosinus, u radijanima
`floor(`***vrijednost***`)`	zaokruživanje na manju cjelobrojnu vr.
`log(`***vrijednost***`)`	logaritam, osnova *e*
`log10(`***vrijednost***`)`	logaritam, osnova 10
`max(`***vr1***`, `***vr2***`)`	veća od dvije vrijednosti
`min(`***vr1***`, `***vr2***`)`	manja od dvije vrijednosti
`round(`***vrijednost***`)`	zaokruživanje na najbližu vrijednost
`sin(`***vrijednost***`)`	sinus, u radijanima
`sqrt(`***vrijednost***`)`	kvadratni korijen

Konstanta	Opis
`e`	2.7182818...
`pi`	3.1415926...

- Da bi bili u mogućnosti da koristimo većinu ovih komandi potrebno je importovati ***math*** biblioteku:

```
from math import *
```

Varijable

- **varijabla**: Imenovani prostor u memoriji koji skladišti neku vrijednost.
  - Upotreba:
    - računanje izraza,
    - skladištenje rezultata u varijablu,
    - korišćenje varijable u ostatku programa.

- **dodjela vrijednosti**:
  - Sintaksa:

    naziv varijable = ***vrijednost***

  - Primjer: `x = 5`

    `gpa = 3.14`

  - **Nije potrebna deklaracija varijable!**

print

- `print` : Štampanje teksta u konzoli.
- **Sintaksa:**

```
print("Poruka")
print(Izraz)
```

  - Moguće je štampati više izraza/poruka na istoj liniji

```
print(Vr1, Vr2, ..., VrN)
```

- **Primjeri:**

```
print("Zdravo!")
god = 45
print("Imate", 65 - god, "godina do penzije")
```

Izlaz:

```
Zdravo!
Imate 20 godina do penzije
```

input

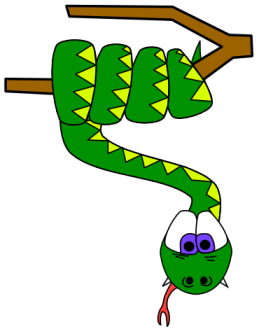
- `input` : Čita ulaz iz konzole
  - Kod python verzije 3, ovaj metod konvertuje unos u string
  - Moguće je skladištiti rezultat **input** funkcije u varijablu.
  - Primjer:

```
god = input("Koliko imate godina?")
print("Imate", 65 - int(god), "godina do penzije")
```

Izlaz:

```
Koliko imate godina? 53
Imate 12 godina do penzije
```

- **ZA VJEŽBU:** Napisati program koji za zadati input, koji predstavlja broj indeksa studenta, štampa osnovne informacije o studentu (ime, prezime, datum i mjesto rođenja)



Petlje i uslovi

`for` petlja

- **`for` petlja**:
  - Sintaksa:

```
for nazivVarijable in grupaVariabli:
    izraz
```

    - Dio koda koji se izvršava u okviru petlje mora imati istu indentaciju (tab ili space).
    - ***`grupaVarijabli`*** može biti opseg cijelih brojeva, specificiran `range` funkcijom.

  - Primjer:

```
for x in range(1, 6):
    print(x, "kvadrirano je", x * x)
```

Izlaz:

```
1 kvadrirano je 1
2 kvadrirano je 4
3 kvadrirano je 9
4 kvadrirano je 16
5 kvadrirano je 25
```

range

- Funkcija `range` definiše opseg cijelih brojeva:

`range(`***start***`, ***stop***`)` - cijeli brojevi izmedju ***start*** (uključujući) i ***stop*** (bez)

- Moguće je dodati i treći argument koji definiše promjene između vrijednosti.

`range(`***start***`, ***stop***`, ***step***`)` - cijeli brojevi izmedju ***start*** i ***stop,*** pri čemu se susjedni brojevi razlikuju za ***step.***

- **Primjer:**

```
for x in range(5, 0, -1):
    print(x)
print("Kraj!")
```

Izlaz:

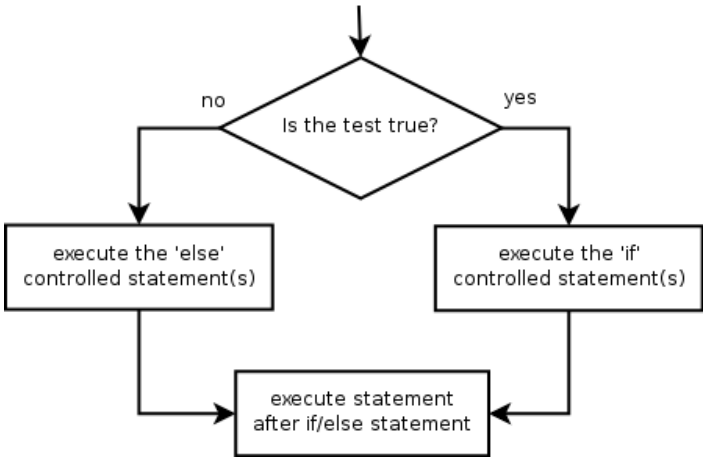
```
5
4
3
2
1
Kraj!
```

ZA VJEŽBU: Napisati Python program koji računa faktorijal broja.

if/else

- **`if/else`**: Izvršava jedan blok instrukcija ukoliko je zadati uslov ispunjen (**True**), i drug blok instrukcija ako uslov nije ispunjen (**False)**.
  - Sintaksa:

```
if uslov:
    instrukcije
else:
    instrukcije
```

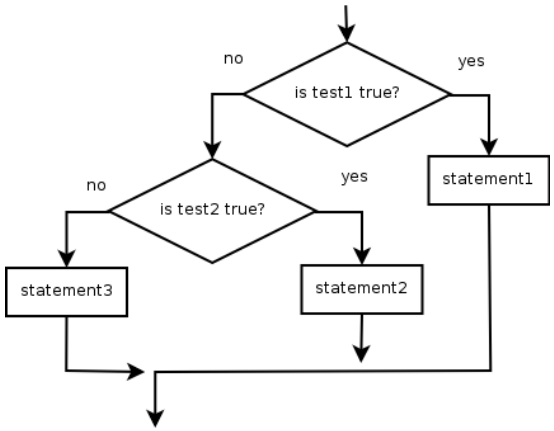


- Primjer:

```
gpa = 1.4
if gpa > 2.0:
    print("Dobrodosli na ETF!")
else:
    print("Vasa prijava je odbijena.")
```

- Dodatni uslovi se kreiraju sa `elif` ("else if"):

```
if uslov:
    instrukcije
elif uslov:
    instrukcije
else:
    instrukcije
```



while

- **`while` petlja**: Izvršava set instrukcija sve dok je uslov zadovoljen (**True**).
- Sintaksa:

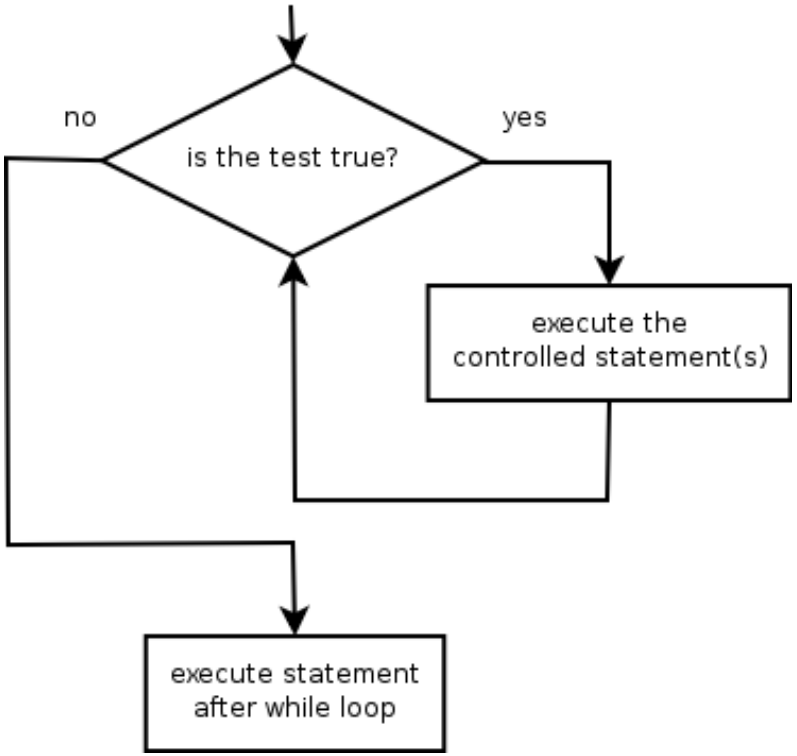
```
while uslov:
    instrukcije
```

- Primjer:

```
br = 1
while br < 200:
    print(br)
    br = br * 2
```

  - **Izlaz**:

```
1 2 4 8 16 32 64 128
```



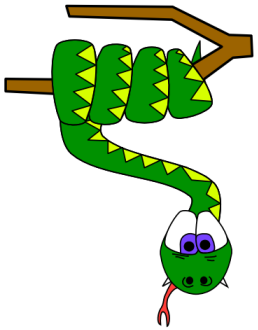
Logičke operacije

- Relacioni operatori:

Operator	Meaning	Example	Result
==	identički jednako	`1 + 1 == 2`	`True`
!=	različito	`3.2 != 2.5`	`True`
<	manje	`10 < 5`	`False`
>	veće	`10 > 5`	`True`
<=	manje ili jednako	`126 <= 100`	`False`
>=	veće ili jednako	`5.0 >= 5.0`	`True`

- Logički izrazi mogu se kombinovati sa logičkim operatorima:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`



Obrada teksta i fajlova

Stringovi

- **string**: Sekvenca teksta u programu.
  - Stringovi počinju i završavaju se apostrofima.
  - Primjeri:

```
"Zdravo"
“Ovo je string"
“Ovo je takodje string. Moze biti veoma dug!"
```

- Specijalni karakteri uvode se pomoću obrnute kose crte:
  - `\t` tab
  - `\n` nova linija
  - `\"` apostrof
  - `\\` obrnuta kosa crta
  - Primjer: `“Dobar\tdan\nKako si?"`

Indeksiranje

- Karakteri u stringu su indeksirani brojevima od 0 pa naviše:
  - Primjer:

```
ime = "P. Diddy"
```

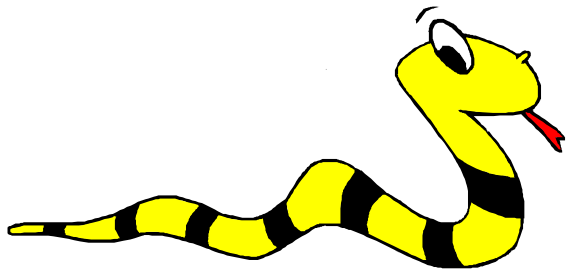
indeks	0	1	2	3	4	5	6	7
karakter	P	.		D	i	d	d	y

- Pristupanje individualnom karakteru stringa:
  imeVarijable `[` ***indeks*** `]`
  - Primjer:

```
print(ime, "pocinje sa", ime[0])
```

Izlaz:

```
P. Diddy pocinje sa P
```



String - karakteristike

- `len(`***string***`)` - broj karaktera stringa
- ***string***`.lower()` - verzija stringa sa malim slovima
- ***string***`.upper()` - verzija stringa sa velikim slovima

- Primjer:

```
ime = "Martin Douglas Stepp"
duzina = len(ime)
veliko_ime = ime.upper()
print(veliko_ime, "ima", duzina, "karaktera“)
```

Izlaz:

```
MARTIN DOUGLAS STEPP ima 20 karaktera
```

Obrada teksta

- Ispitivanje, mijenjanje i formatiranje teksta.
  - često se koriste petlje koje ispituju karaktere stringa jedan po jedan

- `for` petlja može ispitati svaki karakter u stringu redom.
  - Primjer:

```
for c in "booyah":
    print c
```

Izlaz:

```
b
o
o
y
a
h
```

Obrada fajla

- Mnogi programi obrađuju podatke iz fajlova.

- Čitanje sadržaja fajla:

```
nazivVarijable = open("imeFajla").read()
```

Primjer:

```
fajl_tekst = open("racun.txt").read()
```

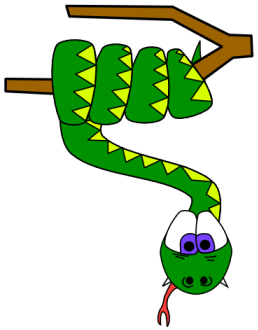
- Čitanje fajla liniju-po-liniju:

```
for linija in open("imeFajla").readlines():
    instrukcije
```

Primjer:

```
brojac = 0
for linija in open("racun.txt").readlines():
    brojac = brojac + 1
print("Fajl sadrzi", brojac, "linija.“)
```

- **Za vježbu:** Napravite program koji u zadatom fajlu broji samoglasnike



Složeni tipovi podataka

Lista i tuple

- Tuple – nepromjenjivi skup elemenata koji predstavlja jednu logičku cjelinu
- Lista – promjenjivi niz podataka bilo kojeg tipa

- Tuple se definiše pomoću običnih zagrada i zareza:

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Liste se definišu pomoću uglastih zagrada i zareza:

```
>>> li = ["abc", 34, 4.34, 23]
```

- Elementima jednog tupla ili liste pristupamo isto kao karakterima u stringu – korišćenjem uglastih zagrada:

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]      # Drugi element tupla.
 'abc'

>>> li = ["abc", 34, 4.34, 23]
>>> li[1]       # Drugi element liste.
  34
```

Negativno indeksiranje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

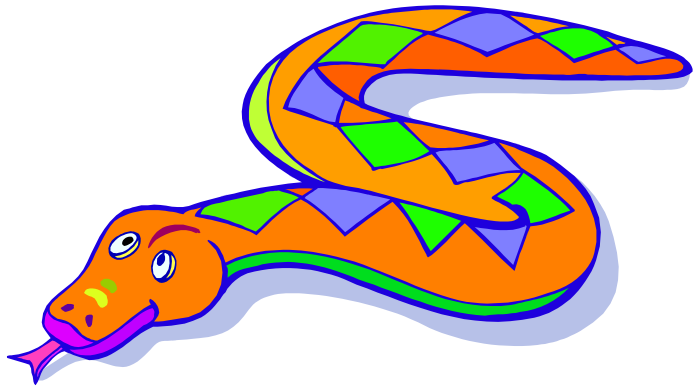
Pozitivno indeksiranje: gledano sa lijeva, brojimo od 0.

```
>>> t[1]
'abc'
```

Negativno indeksiranje: gledano sa desna, počinjemo od -1.

```
>>> t[-3]
4.56
```

- **Na isti način indeksiraju se stringovi, liste i tuples.**



Lista i Tuple - slajsovanje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Kako izdvojiti prva tri elementa tupla/liste?

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

Možemo koristiti i negativo indeksiranje za izdvanje poslednja tri elementa:

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Opcioni argument dozvoljava selekciju svakog n-tog elementa. Na primjer, izdvojimo svaki drugi element:

```
>>> t[1:-1:2]
('abc', (2,3))
```

Lista i Tuple - slajsovanje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Ukoliko ne navedemo prvi indeks, podrazumijeva se da počinjemo od prvog elementa:

```
>>> t[:2]
(23, 'abc')
```

Ukoliko ne navedemo drugi indeks, kopiranje se vrši od prvog indeksa pa do kraja:

```
>>> t[2:]
(4.56, (2,3), 'def')
```

‘in’ operator

- Možemo provjeriti prisustvo elementa u listi korišćenjem „in“ operatora

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- Za stringove i podstringove:

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Operator „in“ takođe se koristi u okviru petlji:

```
t = [1, 2]
for element in t:
    print(element)
```

Rezultat: 1
2

‘+’ operator

- Operator + koristimo za povezivanje više stringova/lista/tuple-ova:

```
>>> (1, 2, 3) + (4, 5, 6)
 (1, 2, 3, 4, 5, 6)

>>> [1, 2, 3] + [4, 5, 6]
 [1, 2, 3, 4, 5, 6]

>>> “Hello” + “ ” + “World”
 ‘Hello World’
```

Operacije nad listama

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append('a')   # Dodavanje elementa
>>> li
[1, 11, 3, 4, 5, 'a']
```

Dodavanje elementa na tačno određenoj poziciji:

```
>>> li.insert(2, 'i')
>>> li
[1, 11, 'i', 3, 4, 5, 'a']
```

Operacije nad listama

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')      # Indeks prvog pojavljivanja argumenta
    1

>>> li.count('b')      # Broj pojavljivanja argumenta
    2

>>> li.remove('b')     # Uklanja zadati argument na najmanjem indeksu
>>> li
  ['a', 'c', 'b']
```

- Obrtanje redosleda i sortiranje elemenata:

```
>>> li = [5, 2, 6, 8]
>>> li.reverse()     # Obrnut redosled elemenata
>>> li
  [8, 6, 2, 5]

>>> li.sort()        # Sortiranje liste
>>> li
  [2, 5, 6, 8]
```

Rečnici

- Rečnici vrše mapiranje podataka koje nazivamo „ključevima“ sa „vrijednostima“.
  - **Ključevi** mogu biti bilo koji nepromjenjivi tip podataka
  - **Vrijednosti** mogu biti bilo koji tip podataka
  - Jedan rečnik može da skladišti **vrijednosti** različitog tipa

- Moguće je dodati, modifikovati, izbrisati bilo koji *ključ-vrijednost* par
- Python rečnici su poznati i kao *hash* tabele i asocijativni nizovi

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
123
>>> d['bozo']
Traceback (innermost last):
  File '<interactive input>' line 1, in ?
KeyError: bozo
```

Ažuriranje rečnika

- Ključevi moraju biti jedinstveni
- Dodjelom vrijednosti postojećem ključu briše se njegova stara vrijednost

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user'] = 'clown'
>>> d
{'user':'clown', 'pswd':1234}
>>> d['id'] = 45
>>> d
{'user':'clown', 'id':45, 'pswd':1234}
```

- Redosled *key-value* parova u rečniku je neuređen. Par koji smo dodali prvi, možda se ne nalazi na prvom mjestu u rečniku.
- Brisanje elemenata rečnika:

```
>>> del d['user']  # Ukloni jedan par
>>> d
{'id':45, 'pswd':1234}
>>> d.clear()      # Ukloni sve
```

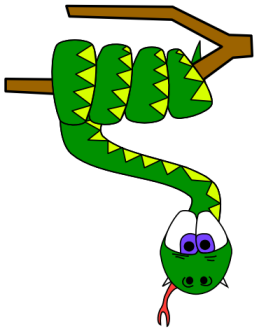
Korisne funckije za pristup podacima u rečniku

```
>>> d = {'user':'bozo', 'pswd':1234, 'id':34}

>>> d.keys()  # Lista ključeva!
['user', 'pswd', 'id']

>>> d.values() # Lista vrijednosti!
['bozo', 1234, 34]

>>> d.items()  # Lista tuple parova (ključ,vrijednost)
  [('user','bozo'), ('pswd',1234), ('id',34)]
```



Funkcije

Funkcije

Definisanje funckije počinje sa `def`

Ime funkcije i njeni argumenti

```
def vrati_odgovor(imeFajla):
    """ Objašnjenje """
    linija1
    linija2
    return brojac
...
```

Prva linija sa manjim „razmakom“ (indentation) izvršava se van funkcije

‘return’ indicira vrijednost koja se vraća kao rezultat.

Nije potrebna deklaracija tipa ni za argumente ni za rezultat

Funkcije

- Pozivanje funkcije:

```
>>> def myfun(x, y):
                return x * y
>>> myfun(3, 4)
           12
```

- **Sve funkcije u Python-u vraćaju rezultat**
  - čak iako nema *return* izraza u kodu.
- **Funkcije bez *return* izraza vraćaju specijalnu vrijednost *None***
  - *None* je specijalna konstatna u programskom jeziku.
  - *None* je ekvivalent *null* konstanti u Javi.
  - Interpreter ne može da štampa *None*
- Dvije funkcije ne mogu imati isto ime

Funkcije

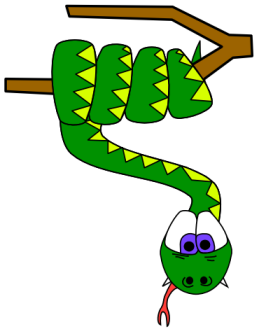
- Funkciji je moguće proslijediti default argumente

```
>>> def myfun(b, c=3, d="hello"):
            return b + c
>>> myfun(5,3,"hello")
>>> myfun(5,3)
>>> myfun(5)
```

Svi navedeni pozivi funkcije vraćaju rezultat 8.

- Redosled argumenata može biti izmijenjen:

```
>>> def myfun (a, b, c):
        return a-b
>>> myfun(2, 1, 43)
  1
>>> myfun(c=43, b=1, a=2)
  1
```



Klase i objekti

Definisanje klase

- Definisanje klase:

```
class imeKlase:
        izrazi
```

- Primjer:

```
class Tacka:
        x = 0
        y = 0

p1 = Tacka()
p1.x = 2
p1.y = -5
```

tacka.py

```
1  class Tacka:
2      x = 0
3      y = 0
```

- Deklarisanje atributa može se vršiti unutar klase ili u konstruktoru
- Svaki atribut ili metod imenovan sa dvije „_“ crte na početku i jednom je privatnog tipa
- Nasleđivanje:

```
class etf_student(student):
```

Korišćenje klase

`import` **`class`**

Klijentski programi moraju importovati klasu koju koriste.

point_main.py

```
1  from Tacka import *
2
3
4  p1 = Tacka()
5  p1.x = 7
6  p1.y = -3
7  ...
8
9  # Python objekti su dinamički (možemo dodavati nove
10 atribute bilo kada!)
   p1.ime = "Petar"
```

Metodi objekta

```
    ...
def imeMetoda(self, parametar, ..., parametar):
```

- `self` mora biti prvi parametar bilo kojeg metoda objekta
  - predstavlja "implicitni parametar"
- Atributima objekta se uvijek pristupa referencom na `self`

```
class Tacka:
    def pomjeri(self, dx, dy):
        self.x += dx
        self.y += dy
    ...
```

ZA VJEŽBU: Dodati metode koji računaju rastojanje između dvije tačke, postavljaju koordinate i računaju rastojanje od koordinatnog početka!

Pozivanje metoda

- Klijent može pozivati metode objekta na dva načina:
  1. `objekat.metod(parametri)`
  2. `Klasa.metod(objekat, parametri)`

- Primjeri:
  - `p = Tacka(3, -4)`
  - `p.pomjeri(1, 5)`
  - `Tacka.pomjeri(p, 1, 5)`

Konstruktori

```
def __init__(self, parametar, ..., parametar):
        izrazi
```

- **Konstruktor** je specijalni metod naziva `__init__`
- Primjer:

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

Kako kreirati objekat klase Tacka bez prosleđivanja koordinata?

Dodatna dokumentacija

- http://python.org/
  - dokumentacija, tutorijali ...
- Knjige:
  - *Learning Python*, Mark Lutz
  - *Python Essential Reference*, David Beazley
  - *Python Cookbook*, Martelli, Ravenscroft and Ascher
  - (online http://code.activestate.com/recipes/langs/python/)
  - http://wiki.python.org/moin/PythonBooks