

# Python tutorijal

# Sadržaj:

## 1. Upoznavanje sa razvojnim okruženjem

- Pycharm

## 2. Osnovi Python jezika

- Sintaksa i konvencije
- Kontrola toka programa
- Osnovni tipovi podataka
- Prosleđivanje parametara (funkcija i komandna linija)
- Osnovne funkcije za rad sa datotekama

## 3. Vežbanje

- Kontrola toka programa, petlje i strukture podataka
- Funkcije i rad sa datotekama
- Prosleđivanje ulaznih argumenata programa

# Upoznavanje sa Python razvojnim okruženjem

# Python razvojna okruženja

- Visual Studio 2015 Community/Enterprise + PTVS (Win)
- Bilo koji tekst editor: Notepad++, Vi, Gedit
- Python IDE (Win/Linux/MacOS)
- **PyCharm** (Win/Linux/MacOS)
- Enthought Canopy (Win/Linux/MacOS)

# Python razvojna okruženja

Na vežbama ćemo koristiti:

- Python 3.x
- Pycharm - <https://www.jetbrains.com/pycharm/>

# Uvod u Pycharm

- Kreiranje novog projekta
- Konfiguracija projekta
- *Hello world* program
- *main* funkcija
- Moduli u Python-u

# Sintaksa i konvencije (1/6)

## Struktura programskog koda:

- Jedan iskaz po liniji

```
>> a = x + var
```

- Razlomljeni iskazi

```
>> math.sqrt(math.sin(b) + \  
    math.cos(c))
```

- Komentari počinju sa #

# Sintaksa i konvencije (2/6)

Indentacija definiše programski blok

```
# Function foo
```

```
def foo(n):
```

```
···· result = []
```

```
···· a, b = 0, 1
```

```
···· while b < n:
```

```
······ result.append(b)
```

```
······ a, b = b, a+b
```

```
······ return result
```

```
# Function foo
```

```
def foo(n, data):
```

```
···· ret = []
```

```
···· if n==False:
```

```
······ ret = process1(data)
```

```
···· else:
```

```
······ ret = process2(data)
```

```
···· return ret
```



# Sintaksa i konvencije (2/6)

Indentacija definiše programski blok

```
# Function foo
```

```
def foo(n):
```

```
·····result = []
```

```
·····a, b = 0, 1
```

```
·····while b < n:
```

```
········result.append(b)
```

```
········a, b = b, a+b !!!
```

```
········return result
```

```
# Function foo
```

```
def foo(n, data):
```

```
·····ret = []
```

```
·····if n==False:
```

```
········ret = process1(data)
```

```
·····else: !!!
```

```
······ret = process2(data) !!!
```

```
·····return ret
```

# Sintaksa i konvencije (3/6)

## Imena promenljivih, funkcija i klasa:

- Mogu početi slovom ili donjom crtom
- Mogu sadržati brojeve
- Nisu dozvoljeni
  - Specijalni znaci: @, %, \$ itd.
  - Rezervisanje reči
- Izbegavati korišćenja donje crte na početku
- Rezervisane reči:

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

# Sintaksa i konvencije (4/6)

## Literali:

- Numerički literali:
  - Boolean (True, False)
  - Celobrojna vrednost (1, 65536, 0xDEADBEEF, itd.)
  - Vrednost sa pokretnim zarezom (1.44, 1.2345e+02, itd.)
  - Kompleksna vrednost (4 + 5j)
- Znakovni literali:
  - Niz znakova (“ovo je string”, ‘i ovo je string’, “’Ovo’ je string unutar stringa”, itd.)
  - Nizovi znakova predstavljeni su u ASCII formatu (1 znak - 1 byte)
  - Specijalni karakteri (\t \n itd.)

# Sintaksa i konvencije (5/6)

## Aritmetički operatori (C\C++ like):

- Operatori se ponašaju očekivano: +, -, /, \*, %
- Specijalni operatori:
  - Operator // (deljenje sa zaokruživanje na prvi manji broj)
  - Operator \*\* (a \*\* b - broj *a* stepenovan *b* puta)

## Operatori poređenja (C\C++ like):

- Operatori se ponašaju očekivano: >, <, ==, !=, >=, <=
  - Povratna vrednost je True ili False

## Logički operatori:

- Operatori se ponašaju očekivano: ***and***, ***or***, ***not***
  - Povratna vrednost je True ili False

# Sintaksa i konvencije (6/6)

## Operatori dodele (C\C++ like):

- Operatori se ponašaju očekivano: =, +=, -=, /=, itd.

## Operatori za rukovanje bitima (C\C++ like):

- Operatori se ponašaju očekivano: <<, >>, &, |, ^, ~

## Operatori identiteta (*Identity operators*):

- Proverava da li su dve vrednosti (ili promenljive) identične, tj. da li se nalaze na istoj memorijskoj lokaciji
  - Ako su dve vrednosti jednake ne implicira da su identične
- Operatori identiteta: *is*, *is not*

# Kontrola toka programa (1/4)

## Iskazi **if/else/elif**:

- Iskazi *if* i *else* imaju istu semantiku kao u C\C++ jeziku
- Iskaz *elif* je skraćeno od ‘else-if’

### # if/else/elif example

```
if x < 0:  
    .... print("x < 0")  
elif x == 0:  
    .... print("x = 0")  
else:  
    .... print("x > 0")
```

### # if/else example

```
if x < 0:  
    .... print("x < 0")  
else if x == 0:  
    .... print("x = 0")  
else:  
    .... print("x > 0")
```

# Kontrola toka programa (2/4)

## **while** petlja:

- *while* petlja služi za uzastopno izvršenje niza iskaza dok je određeni uslov zadovoljen

`while_stmt ::= "while" expression ":" suite ["else" ":" suite]`

**# while loop example**

```
x = 100
```

```
print("x < 0")
```

```
while x != 0:
```

```
···· print("x = ", x)
```

```
···· x -= 1
```

# Kontrola toka programa (3/4)

## for petlja:

- *for* petlja služi za prolazak (itreriranje) kroz skupove podataka kao što su znakovni nizovi, torke i liste (sekvence)

`for_stmt ::= "for" target_list "in" expression_list ":" suite ["else" ":" suite]`

*# for loop example*

`x = 1, 2, 3, 4, 5`

`for it in x:`

`···· print("it = ", it)`

- Za potrebe inicijalizacije koristiti `range(start, stop, [step])` klasu

*# for loop example*

`for it in range(5):`

`···· x = it`



# Kontrola toka programa (4/4)

Iskaze **return**, **break** i **continue** koristiti u skladu sa C\C++ semantikom

- **break** iskaz prekida izvršavanje petlje unutar koje se nalazi
- **continue** iskaz započinje novi ciklus petlje unutar koje se nalazi
- **return** iskaz napušta izvršavanje funkcije unutar koje se nalazi

# Osnovni tipovi podataka (1/9)

## Konverzije:

- `int(x, (b))` - Tip *string* u tip *int* baze *b*
- `float(x)` - Tip *string* u tip *float*
- `complex(r,i)` - Kompleksni broj od *r* i *j*
- `str(x)` - Bilo koji tip *string*
- `chr(i)` - Tip *int* u tip *char* (do 255)
- `ord(c)` - Tip *char* (Unicode) u tip *int*
- `hex(i)` - Tip *int* u tip *string*, heksadecimalni zapis
- `bin(i)` - Tip *int* u tip *string*, binarni zapis
- `oct(i)` - Tip *int* u tip *string*, oktalni zapis

# Osnovni tipovi podataka (2/9)

Logički izrazi:

- **True**, logički tačna trvdnja
- **False**, logički netačna trvdnja
- **0**, None i/ili prazna kolekcija
- **!=0**, kolekcija koja nije prazna

# Osnovni tipovi podataka (3/9)

Osnovne kolekcije su:

- **Lista** (list)
  - **Rečnik** (dict)
  - **Torke** (tuples)
  - **Skupovi** (sets)
  - **Znakovni nizovi** (strings)
- 
- Za više detalja o strukturama podataka, funkcijama i metodama pogledati zvaničnu dokumentaciju (<https://docs.python.org/3/>)

# Osnovni tipovi podataka (4/9)

## Lista

- Kolekcija koja sadrži podatke istog ili različitog tipa odvojene zarezima

```
>> l = []
```

```
>> l = [1, 2, 'a', "b"]
```

- Liste se mogu menjati - *mutable* su

```
>> l[1] = 5
```

- Funkcije za rad sa listom:

- **len**(listname) # vraća dužinu prosleđene liste
- **append**(value) # dodaje novi element na kraj liste
- **insert**(index, value) # dodaje novi element
- **remove**(value) # briše element *value* iz liste
- **del** l[index] # briše element sa indeksom *index*
- **clear**() # briše sve elemente iz liste

# Osnovni tipovi podataka (5/9)

## Rečnik

- Kolekcija koja sadrži parove tipa *ključ:vrednost*, gde se *ključ* koristi za indeksiranje *vrednosti*
- Ključevi mogu biti svi tipovi koji se ne mogu menjati (immutable)

```
>> d = {}
```

```
>> d = {"key1" : 1, "key2" : 2}
```

- Rečnik se može menjati - *mutable je*

```
>> d["key1"] = 5
```

- Rad sa rečnikom:

```
- d["key3"] = 3 # dodavanje novog elementa
```

```
- del d["key1"] # brisanje elementa
```

```
- list(d) # vraća listu svih ključeva u rečniku
```

# Osnovni tipovi podataka (6/9)

## Torka

- Kolekcija koja sadrži podatke tipa n-torke

```
>> t = ((),)
```

```
>> t = ('a', 2, 3)
```

```
>> t = 'a', 2, 3
```

- Torka se **ne** može menjati - *immutable* je

# Osnovni tipovi podataka (7/9)

## Skup

- Kolekcija koja **ne** sadrži duplikate

```
>> s = set()
```

```
>> s = {1, 2, 2, 3, 4, 5}
```

- Skup se može menjati - *mutable* je
- Podržava matematičke operacije za rad sa skupovima:

```
- a = {1, 2, 3}
```

```
- b = {1, 4, 5}
```

```
- a - b    # sve što je u a, a nije u b
```

```
- a | b     # sve što je u a ili b ili u oba
```

```
- a & b     # sve što je u a i u b
```

```
- a ^ b     # sve što je u a i b, ali nije u oba
```



# Osnovni tipovi podataka (8/9)

## Znakovni niz

```
>> "" ili None  
>> s = "some string"  
>> s = "C:\some\name"  
>> s = r"C:\some\name"
```

- Znakovni niz se ne može menjati - *immutable je*

# Osnovni tipovi podataka (9/9)

## Operacije nad kolekcijama (sekvencama):

- $s1 + s2$  - spajanje
- $s1 * n$  - uzastopno se  $s$  ponavljanja  $n$  puta
- $v1, v2, v3, v4 = s$  - raspakivanje (npr:  $s$  je torka od 4 elementa)
- $s[i]$  - indeksiranje
- $s[i:j]$  - isecanje
- $s[i:j:k]$  - isecanje sa korakom  $k$
- $x \text{ in } s$  - da li  $s$  sadrži  $x$
- $\text{len}(s)$  - dužina  $s$
- $\text{any}(), \text{sum}(), \text{min}(), \text{max}()$

# Prosleđivanje ulaznih argumenata funkcijama

Prosleđivanje ulaznih argumenata funkcijama obavlja se tako što se u pozivu funkcije redom navode promenljive koje se prosleđuju

*#This prints a passed info into this function*

```
def print_info(name, surname, age = 35):
```

```
···· print("Name: ", name, \
······ "Surname: ", surname, \
······ "Age ", age)
···· return
```

(i) `print_info("Petar", "Petrovic", 32)`

(ii) `print_info("Petar", "Petrovic")`

(iii) `print_info("Petar")` !!! Greška

# Duboka i plitka kopija objekta

- Duboka (deep) kopija objekta ne menja original

```
>> l[:], foo(l[:])
```

- Plitka (shallow) kopija objekta menja original

```
>> l, foo(l)
```

- Napomene:

1. Za sve sekvence koji se mogu menjati (**mutable**) prosleđuje se **referenca**
2. Za sve sekvence koji se ne mogu menjati (**immutable**) pravi se nova **kopija**

# Prosleđivanje argumenata komandne linije

Protrebno je koristiti Python **sys** modul

- `sys.argv` predstavlja listu ulaznih argumenata
- `len(sys.argv)` daje broj ulaznih argumenata
- `sys.argv[0]` je ime programa

*# Input arguments example (input.py)*

**import sys**

`print("argv len: ", len(sys.argv))`

`print("argv[0]: ", sys.argv[0])`

`print("argv[1] : ", sys.argv[1])`

`print("argv[2] : ", sys.argv[2])`

Poziv: `$python input.py jedan 2`

Izlaz:

`argv len: 3`

`argv[0]: C:\path\input.py`

`argv[1]: jedan`

`argv[2]: 2`

# Rad sa datotekama

Za rukovanje sa datotekama koriste se funkcije:

- `open(file_name, mode)`, `close()`
- `write(str)`, `read(size)`, `readline()`
- Kao u C\C++ jeziku režimi rada sa datotekama su: `read ('r')`, `write ('w')`, `append ('a')`, `read-write('r+')`

**# Read from input and write to output file**

```
import sys
fin = open("input_file_name", 'r')
fout = open("output_file_name", 'w')
for line in fin:
    ... fout.write(line)
fin.close()
fout.close()
```

# Dodatni materijali

Više informacija o svemu do sada pomenutom kao i Python API dostupno je u zvaničnoj Python web dokumentaciji:

<https://docs.python.org/3/>

# Primer

Napisati program koji listu parova prvih 10 prirodnih brojeva čuva u rečniku pod ključem koji definiše korisnik

- Ime ključa treba preuzeti sa komandne linije
- Parovi brojeva (1,2), (3,4) ... (9,10) predstavljaju torke
- Lista se čuva u rečniku koji mapira string na listu



# Zadaci

Napomena: Sve funkcije implementirati u posebnoj python datoteci, a pozivati ih iz main funkcije datoteke main.py.

1. Napisati funkciju koja računa zbir prvih  $N$  prirodnih brojeva;
2. Napisati funkciju koja računa zbir kvadrata prvih  $N$  prirodnih brojeva; parametar  $N$  se unosi kao ulazni argument programa;
3. Napiši program koji na ulazu prima dva stringa i na osnovu njih formira i ispisuje novi string koji se sastoji od dva puta ponovljena prva tri karaktera iz prvog stringa i poslednja tri karaktera drugog stringa;
4. Inicijalizovati listu sa prvih 100 brojeva i ispisati je u obrnutom redosledu;
5. Koristeći strukturu **rečnink** izračunati broj ponavljanja reči koje se nalaze u datoteci dict\_test.txt;
6. Napisati program koji u listu smešta četiri torke formata (*integer*, *float*, *string*) i ispisuje ih, nakon čega briše prvu torku koja je ubačena u listu;
7. Ponoviti zadadak 6 ali umesto liste koristiti skup;