

# **OSNOVE PROGRAMIRANJA PYTHON**

Profesor  
Milan Paroški

Novi Sad, 2024/2025



**RASPORED DO KRAJA SEMESTRA:**

**Za K2:**

**02.12.2024. – 9. Organizacija programskog koda**

**K2: 13.12.2024.**

**Za ZI:**

**16.12.2023. 10. Rad sa fajlovima**

**23.12.2023. 11. Analiza algoritama**

**30.12.2023. 12. Osnove objektno orijent. programiranja**



**IT**

**K2 i Popravak K1**

PETAK 13.12.2024 15:00-16:00 E001

**SII**

**K2 i Popravak K1**

PETAK 13.12.2024 16:00-18:00 E001



## **K2 i Popravak K1**

**13.12.2024. PETAK, lokacija: E001**

### **SMER IT:**

<b>K2:</b>	<b>15:00-15:30</b>	<b>30 minuta</b>
<b>Popravak K1:</b>	<b>15:30-16:00</b>	<b>30minuta</b>



**SMER SII:**

**K2 praktično:                    16:00-16:40            40minuta**

**K2 teorija:                        16:40-17:00            15minuta**

**K1 praktično:                    17:00-17:40            40minuta**

**K1 teorija:                        17:40-18:00            10minuta**



## **IT**

**K2 I K1**

**Teorija : 15 zadataka**

**Praktično : 15 zadataka**

**Mtutor: 30 zadataka/30 bodova/30 minuta**

## **SII**

**K2 i K1**

**Teorija: 15 zadataka**

**mTutor: 15 zadataka/15bodova/15 minuta**

**Praktično : 2 zadatka**

**Assigment : 2 zadatka/15 bodova/40 minuta**

## Ispitni rok

**K1:30**

**K2:30**

**Završni ispit teorija:9 bodova/16 zadataka/15 minuta**

**Završni ispit praktično:24 poena**

**IT smer: 3 zadataka/60 minuta**

**SII smer: projekat/Nevena**

**Sumarno:**

**Prisutnost (10) + K1(30) + K2(30)+ ZITeor(9) + ZIPrak(24) +**

**Projekat(10) + Aktivnost(5+5) =123 poena**

2024270398	10/09/24, 10:15:01 AM	10/09/24, 3:45:32 PM	5h 30m 31s
2024270823	10/16/24, 10:06:29 AM	10/16/24, 2:25:55 PM	4h 19m 26s
2024271055	10/23/24, 9:50:46 AM	10/23/24, 3:07:45 PM	5h 16m 59s
2023271533	11/04/24, 2:47:42 PM	11/04/24, 7:03:35 PM	4h 15m 52s

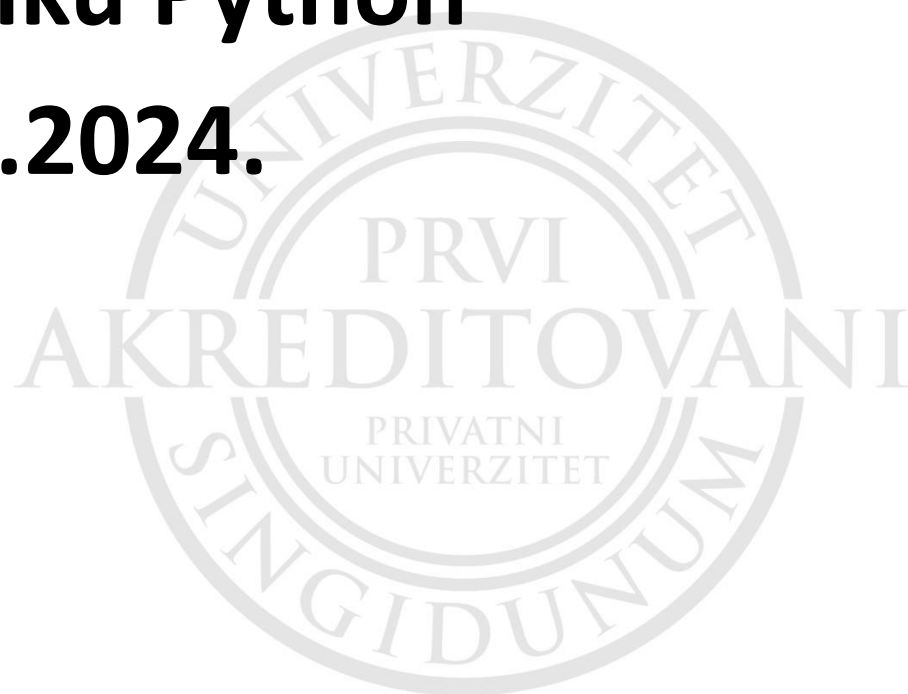




# **Poglavlje 9**

## **Organizacija programskog koda u jeziku Python**

### **02.12.2024.**



# Sadržaj

1. Uvod
2. Upotreba modula u jeziku Python
3. Specifikacija modula
4. Projektovanje softvera s vrha (top-down)
5. Moduli u jeziku Python
6. Primeri programa



## 9.1 Uvod

Softverski sistemi - predstavljaju izuzetno složene entitete.

Neki od softverskih sistema spadaju u najsloženije sisteme koje su ljudi stvorili, npr.:

- operativni sistemi i web čitači sastoje se od 50–100 miliona ? LOC (lines of code - linija koda),
- poslovni informacioni sistem SAP sastoji se od oko 240 miliona linija koda u programskom jeziku visokog nivoa,
- pretraživač Google je softverski sistem čiji obim se procenjuje na 2 milijarde linija koda.

Obim nekih poznatijih softverskih sistema iz kategorije ugrađenih, operativnih i informacionih sistema:

*Ugrađeni softver (embeeded)*

Space Shuttle



400.000

Rover Curiosity



2.500.000

Mercedes klase S 2014



65.000.000

Moderni luksuzni automobili

100.000.000

*Operativni sistemi*

Red Hat Linux 7



30.000.000

Windows XP



45.000.000

Windows 7, 8, 10



50 .. 80.000.000

MAC OS X 10



86.000.000

*ERP poslovni informacioni sistemi*

SAP NetWeaver (ABAP)



238.000.000

Google pretraživač (2015)



2.000.000.000

**YouTube** – Koristi Python za backend servise kako bi omogućio brz razvoj i skaliranje.

**Instagram** – Python je ključni deo backend infrastrukture za rukovanje milijardama korisnika.

**Spotify** – Python se koristi za preporučivanje pesama i obradu velikih količina podataka.

**Google** – Python je osnovni jezik za razvoj mnogih AI rešenja.

**Dropbox** – Ceo klijentski softver je napisan u Pythonu.

**Netflix** – Koristi Python za preporučivanje sadržaja i analizu podataka o korisnicima

**Ubuntu Installer (Ubiquity)** – Python je korišćen za razvoj instalacionog softvera.

**Blender** – Program za 3D modeliranje i animaciju koristi Python za skriptovanje.

- Koliko je štampanih stranica potrebno za prikaz jednog miliona linija koda ?
- 18.000 (1.000.000/55)
- **Programski kod složenijih softverskih sistema nije dovoljno pregledan** ako se predstavi kao niz funkcija istog nivoa.
- **Složeni programi se razvijaju timski, pa se program deli na takve celine** koje mogu da razvijaju manje grupe ili pojedinci.
- **Zbog toga se programski kod u jeziku Python na najvišem nivou organizuje kao skup modula** (modules). Šta su moduli?
- Manje programske celine sastavljene od funkcija i drugih objekata, koji se mogu hijerarhijski organizivati u pakete (packages).

## **Modularno projektovanje (modular design) omogućava:**

- 1. podelu izuzetno velikih programa u manje delove**, koji imaju jasne funkcije i kojima je lakše upravljati;
- 2. raspodelu programskih zadataka** na veći broj programera ili razvojnih timova;
- 3. nezavisni razvoj i testiranje pojedinih celina** (modula), koji se mogu kasnije uključiti u različite složenije sisteme;
- 4. lakše izmene programskog koda**, koje se mogu vršiti samo u određenim modulima, te ih nije potrebno sprovoditi i u ostalim delovima složenog softverskog sistema.

## Prednosti upotrebe modula su:

- **U projektovanju softvera** (software design)?

upotreba modula je **način razvoja dobro projektovanog softvera**.

- **U razvoju softvera** (software development) ?

upotreba modula predstavlja **način podele zadataka programiranja** i višestruke upotrebe razvijenog programskog koda.

- **U testiranju softvera** (software testing) ?

upotreba modula omogućava **zasebno testiranje delova programa** i njihovu integraciju u toku testiranja.

- **U održavanju softvera**, koji čini najveći deo životnog veka softvera?  
**olakšava unošenje izmena u pojedine funkcije programa.**



## 9.2 Upotreba modula u jeziku Python

Dobro projektovan softver u jeziku Python sastoji se od skupa modula.

Modul označava projektovane i/ili implementirane funkcionalnosti koje će se uključiti u neki program.

Modul se sastoji od skupa funkcija i drugih programskih objekata.

Primeri gotovih programskih modula su ranije korišćeni moduli ?

**turtle i random.**

**Svaki fajl na kojem je program u jeziku Python predstavlja modul i može se uključiti u druge programe pomoću naredbe import.**

### **pangram.py:**

*recenica = "Фијуче ветар у шибљу, леди пасаже и куће иза њих и  
гунђа у оџацима"*

*slova = set("абвгдђежзијклљмнњопрстћуфхцџш")*

*slova\_u\_recenici = set(recenica.lower()) & slova*

*a=len(slova\_u\_recenici)*

*print("Broj različitih slova u rečenici je:", a)*

### **Izvršiti program pangram.py:**

Broj različitih slova u rečenici je: 30

### **Poziv.py:**

*import pangram*

*Pangram*

### **Izvršiti program poziv.py:**

Broj različitih slova u rečenici je: 30

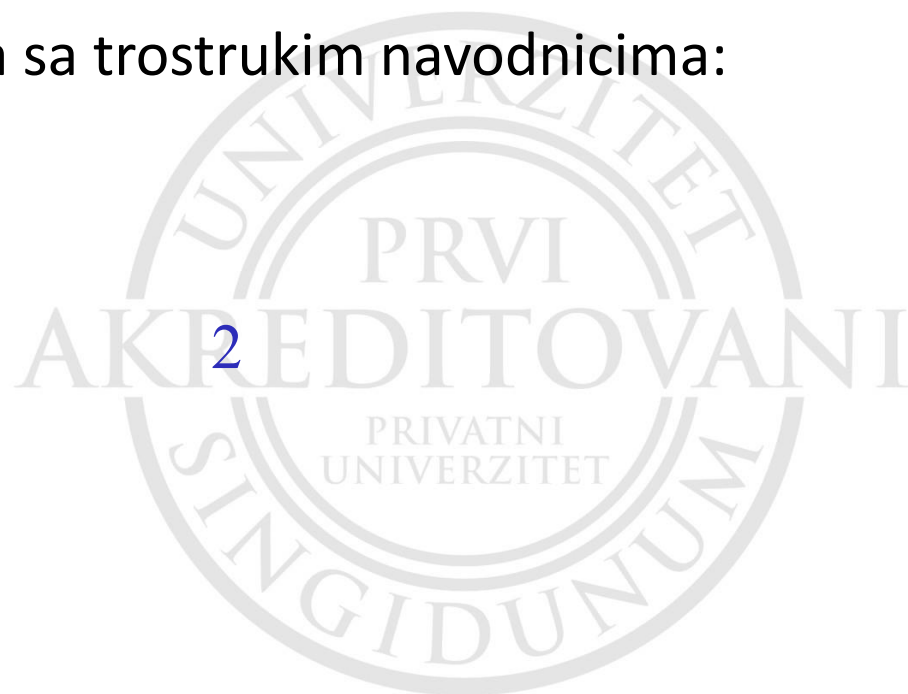


# Dokumentacijski string

**Dokumentacijski string** daje mogućnost programeru da unese određene komentare i objašnjenja za funkcije koje sam piše. Python dozvoljava pisanje dokumentacijskih stringova, poznatijih kao **docstring**.

Dokumentacijski string se često proteže u nekoliko fizičkih linija, te se stoga obično označava sa trostrukim navodnicima:

```
def razlika(x,y):  
    """Preuzmi dva broja i  
    izračunaj razliku"""  
    return x-y  
print(razlika(6,4))  
#Da li će nešto ispisati?
```



Uloga **docstringa** je slična komentarima u bilo kojem programskom jeziku, ali im je korisnost još i to, jer su dostupni pri procesu pokretanja programa.

**U prethodnom primeru to nismo videli...**

Programerska okruženja i drugi alati mogu koristiti dokumentacijske stringove da podsete programera kako koristiti određene objekte - deluju kao pomoć programa kada se zatraže.

U prethodnom primeru je napisan dokumentacijski string za funkciju **razlika** tj. kratko objašnjenje šta ta funkcija radi.

Da bi to objašnjenje videli tj. da bi se dokumentacijski string ispisao na komandnoj liniji koristimo **help(razlika)**.

# Pre toga da pojasnimo help:

## help(print) #Uneti ovo

Help on built-in function print in module builtins:

```
print(*args, sep=' ', end='\n', file=None, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

sep

string inserted between values, default a space.

end

string appended after the last value, default a newline.

file

a file-like object (stream); defaults to the current sys.stdout.

flush

whether to forcibly flush the stream.

- \*args: ?

Možeš proslediti jedan/više argumenata koje želiš da odštampaš.

- sep: ?

Separator između objekata (podrazumevano je razmak ' ')

- end: ?

String koji se dodaje na kraj ispisa (default je novi red '\n').

- file: ?

Odredište ispisa (podrazumevano je terminal/konzola).

- flush: ?

Ako je True, isprazniće izlazni tok odmah.

Primer:

```
print("Hello", "World", sep=",,, ", end="!\n", flush=True)
```

Rezultat?

Hello,,, World!

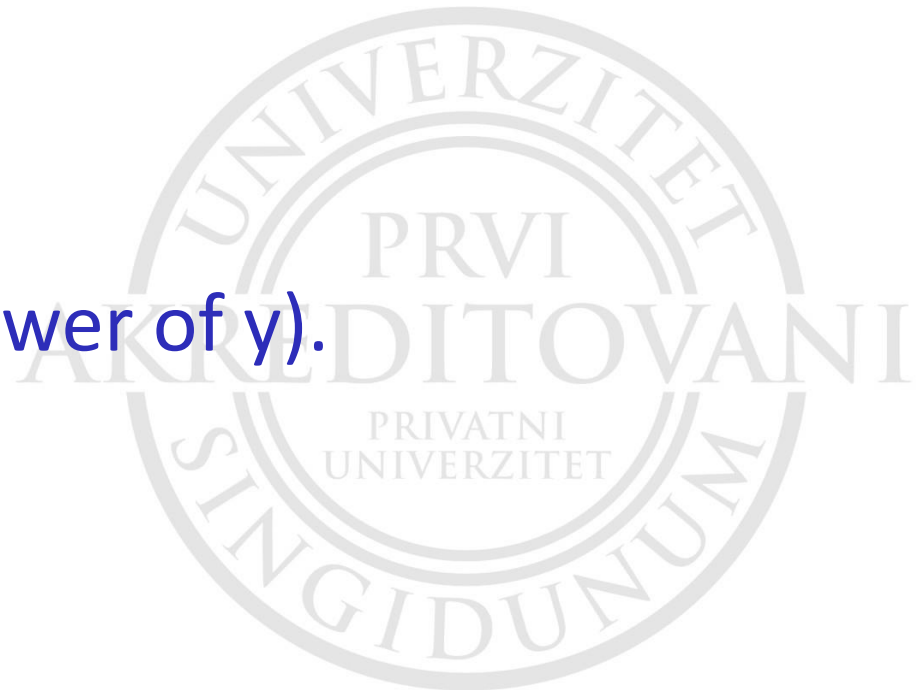


```
from math import *  
help('math.pow')
```

Help on built-in function pow in math:

```
math.pow = pow(x, y, /)
```

Return  $x^{**}y$  (x to the power of y).



```
def razlika(x,y):
```

```
    """Preuzmi dva broja i izračunaj razliku"""
```

```
    return x-y
```

```
help(razlika) #pri pokretanju programa će da se
```

```
    #ispiše docstring za funkciju razlika
```

```
print(razlika(6,4))
```

Help on function razlika in module \_\_main\_\_:

razlika(x, y)

Preuzmi dva broja i izračunaj razliku



# Primer:

```
def odbrojavanje(n):  
    print(n)  
    if n > 1:  
        odbrojavanje(n-1)  
    print("n=",n)  
print(odbrojavanje(5))
```

5  
4  
3  
2  
1  
n= 1  
n= 2  
n= 3  
n= 4  
n= 5  
None

```
def odbrojavanje(n):  
    """odbrojanje brojeva do n"""  
    print(n)  
    if n > 1:  
        odbrojavanje(n-1)  
    print("n=",n)  
print(odbrojavanje(5))  
print(odbrojavanje.__doc__)
```

5  
4  
3  
2  
1  
n= 1  
n= 2  
n= 3  
n= 4  
n= 5  
None  
odbrojanje

Zašto se štampa:NONE?

problem je što funkcija odbrojavanje() nema eksplicitnu povratnu vrednost.

Kada pozovete `print(odbrojavanje(5))`, funkcija vraća `None`, a `print()` ispisuje tu vrednost.

Kako to rešiti:

Umesto

`print(odbrojavanje(5))`

Treba: `odbrojavanje(5)`



```
def odbrojavanje(n):  
    """odbrojanje brojeva do n"""  
    print(n)  
    if n > 1:  
        odbrojavanje(n-1)  
    print("n=",n)  
  
odbrojavanje(5)  
print(odbrojavanje.__doc__)
```

5

4

3

2

1

n= 1

n= 2

n= 3

n= 4

n= 5

odbrojanje brojeva do n

```
def odbrojavanje(n):
```

```
    """odbrojanje brojeva do n"""
```

```
    print(n)
```

```
    if n > 1:
```

```
        odbrojavanje(n-1)
```

```
    print("n=",n)
```

odbrojanje brojeva do n  
Skloniti #

```
#odbrojavanje(5)
```

```
print(odbrojavanje.__doc__)
```

```
#Šta će sada odštampati?
```



```
def odbrojavanje(n):  
    """odbrojanje brojeva do n  
    stampa n  
    petlja  
    rekurzija"""  
    print(n)  
    """stampa n1"""  
    if n > 1:  
        """petlja1"""  
        odbrojavanje(n-1)  
        """rekurzija1"""  
    print("n=",n)  
    """sklonjenja stampa1"""  
#odbrojavanje(5)  
print(odbrojavanje.__doc__)
```

odbrojanje brojeva do n  
stampa n  
petlja  
rekurzija

Štampa se tekst iza  
definicije funkcije

# Kada se skloni #?

5

4

3

2

1

n= 1

n= 2

n= 3

n= 4

n= 5

odbrojanje brojeva do n

stampa n

petlja

rekurzija



## 9.4 Projektovanje softvera s vrha (top-down)

Projektovanje softvera "s vrha" ili "odozgo" (top-down) ?

podrazumeva postepenu dekompoziciju problema u manje module, dok se ne dobiju dovoljno male i jasne celine, koje je lako programski realizovati.

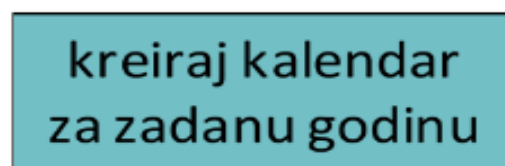
Primer projektovanja s vrha programa za štampanje kalendara za zadanu godinu:



Primer:1



sledeća dekompozicija



Primer projektovanja s vrha (top-down)



## Nedostaci:

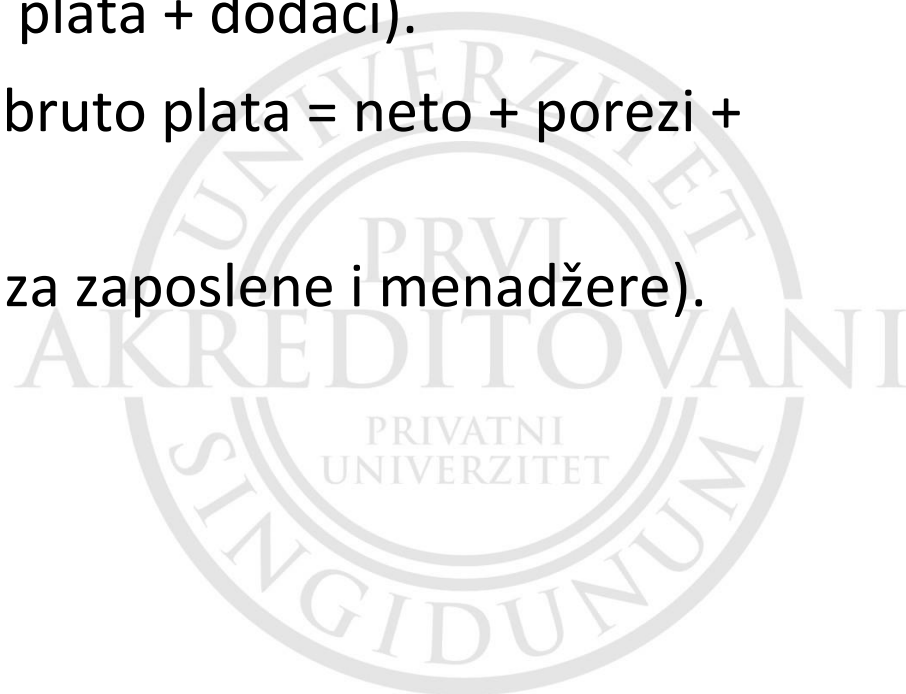
- Zahteva detaljno planiranje unapred.
- Može biti teško predvideti sve detalje na početku, što može dovesti do ponovnog planiranja.
- Ovaj pristup je suprotan **bottom-up** pristupu, gde se razvoj započinje implementacijom osnovnih komponenti, a zatim se one integrišu u celokupan sistem.

# Primer : LD

## **Visoki nivo - Definisanje glavnih funkcionalnosti:**

Na početku razmatramo osnovnu funkcionalnost programa, koja će biti izračunavanje plata. Glavne komponente sistema mogu biti:

- 1. Prikupljanje podataka o zaposlenima** (ime, radno vreme itd.).
- 2. Računanje neto plate** (osnovna plata + dodaci).
- 3. Računanje poreza i doprinosa** (bruto plata = neto + porezi + doprinosi).
- 4. Generisanje izveštaja** (izveštaji za zaposlene i menadžere).



## **Razlaganje funkcionalnosti na niže nivoe:**

Zatim se ova funkcionalnost razlaže na niže nivoe. Na primer:

### **1. Prikupljanje podataka o zaposlenima:**

#### ***1.1. Unos podataka o zaposlenima :***

- IME,
- Radno vreme
- Koeficijenat
- Staž
- Broj sati u mesecu
- Topli obrok i prevoz
- GO,PO,BO,NO,Praznik
- dodaci

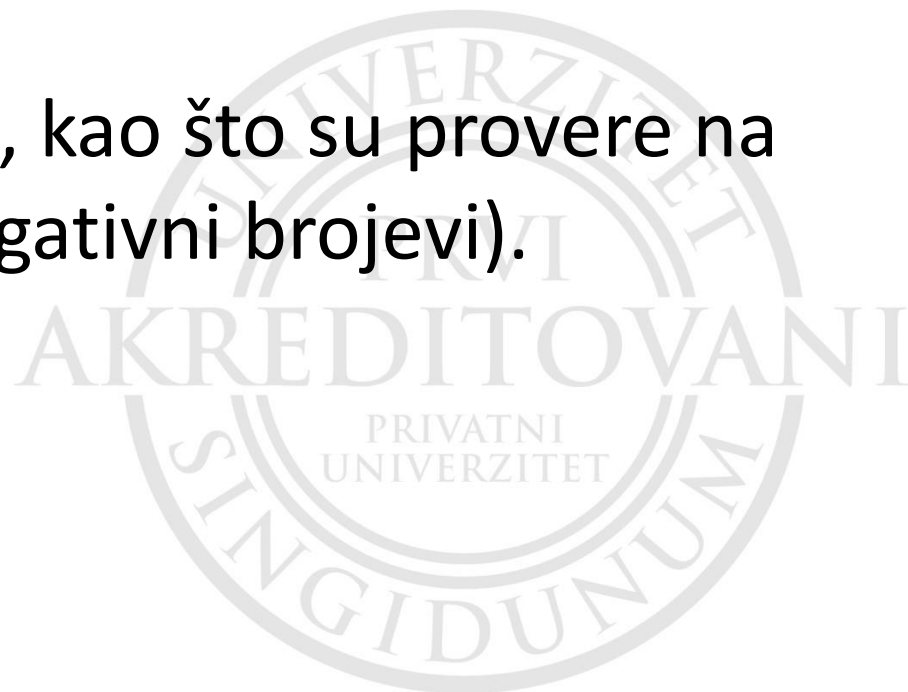
#### **1.2. Validacija podataka (da li su svi podaci tačni i potpuni).**



## **1.1. Unos podataka o zaposlenima:**

1.1.1. Program bi mogao imati formular za unos osnovnih podataka (ime, pozicija, osnovna plata, itd.).

1.1.2. Validacija podataka, kao što su provere na greške u unosu (npr. negativni brojevi).



## Testiranje i optimizacija

**Top-Down** pristup omogućava da se testira svaki deo sistema pojedinačno, počevši od najviših funkcionalnosti prema nižim nivoima.

Optimizacija sistema može uključivati dodavanje novih funkcionalnosti (kao što je slanje plata direktno na bankovne račune ili generisanje poreza na osnovu najnovijih zakonskih izmena).

## 2.čas



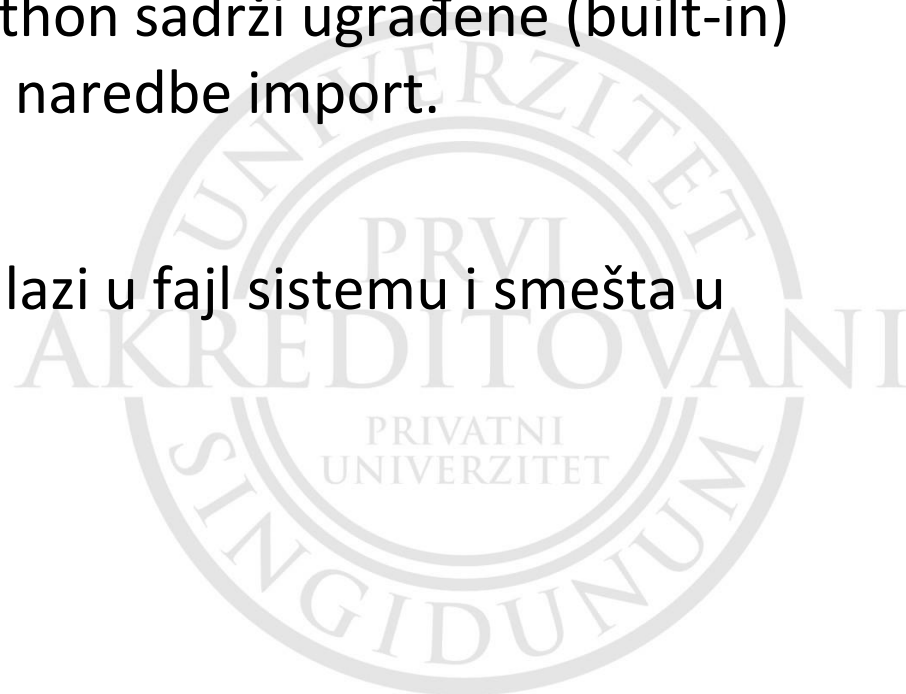
## 9.5 Moduli u jeziku Python

Moduli u jeziku Python sadrže definicije, ali mogu da sadrže i naredbe, koje se izvršavaju samo jednom, obično radi inicijalizacije.

Standardna biblioteka jezika Python sadrži ugrađene (built-in) module, koji se koriste pomoću naredbe `import`.

Prilikom uvoza, modul se pronalazi u fajl sistemu i smešta u memoriju.

Redosled pretraživanja je:



# 1. Prvo se traži u tekućem folderu to je direktorijum iz koga je pokrenut program

Kako možemo videti u kom smo folderu?:

```
import os
```

```
print(os.getcwd())
```

Rezultat: C:\Users\Milan\Documents

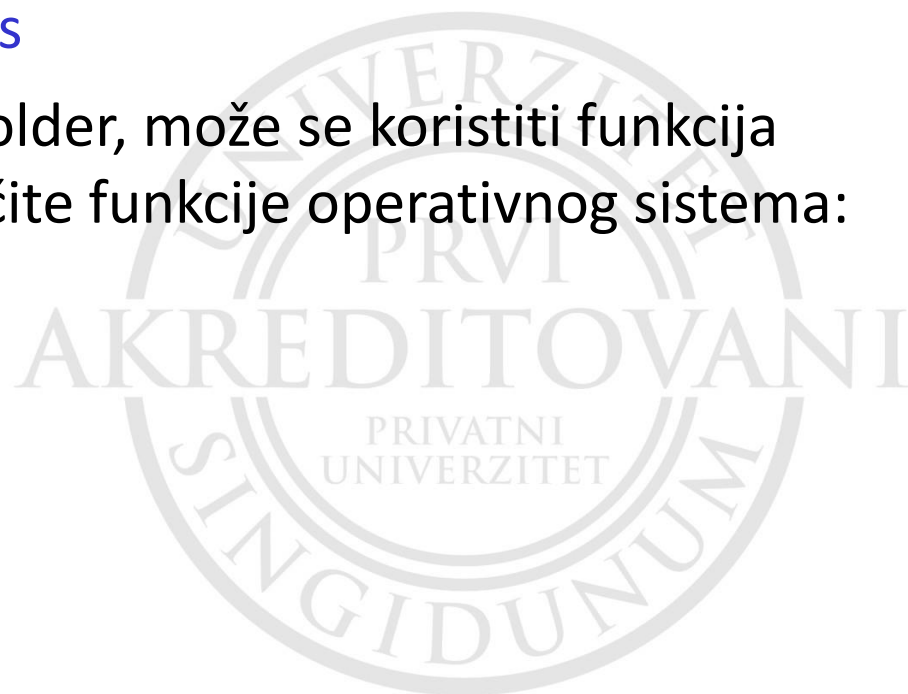
Ukoliko je potrebno promeniti radni folder, može se koristiti funkcija **chdir iz modula os**, koji sadrži različite funkcije operativnog sistema:

```
import os
```

```
os.chdir(„C:\Milan“)
```

```
print(os.getcwd())
```

Rezultat: C:\Milan





Šta se dešava kada uradimo turn off pa turn on IDLE?

Opet je onaj stari folder, tekući folder

**2. Ako se fajl ne pronade, traži se u folderu navedenom u promenljivoj PYTHONPATH** (može se postaviti komandom set PYTHONPATH);

**3. Ako se ne pronade ili ova promenljiva nije definisana, traži se u folderu koji je definisan prilikom instalacije sistema, npr. C:\Python\Lib;**

**4. Ako se modul ne pronade, dojavljuje se greška (ImportError).**

# Šta je PYTHONPATH?

## Primer: putanja.py

```
import sys  
print('Argumenti komandne linije su: ')  
for i in sys.argv:  
    print(i)  
print('PYTHONPATH je', sys.path, '\n')
```



## Rezultat:

Argumenti komandne linije su:

C:/Users/Milan/Documents/putanja.py

PYTHONPATH je ['C:/Users/Milan/Documents', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64\\Lib\\idlelib', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64\\python37.zip', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64\\DLLs', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64\\lib', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64', 'C:\\Program Files (x86)\\Microsoft Visual Studio\\Shared\\Python37\_64\\lib\\site-packages']

1. uvezli smo sys modul pomoću import naredbe.
2. sys modul sadrži funkcionalnosti vezane za Python-ov interpreter i njegovu okolinu, odnosno sistem.
3. Kada Python izvrši import sys komandu, on u stvari traži sys modul
4. Ako je modul pronađen, komande koje se nalaze u telu tog modula se pokreću i modul postaje dostupan za korišćenje.
5. argv promenljivoj u sys modulu se pristupa tako što koristimo oznaku tačka odnosno sys.argv
6. sys.argv promenljiva sadrži listu argumenata komandne linije.
7. Python čuva argumente komandne linije u sys.argv promenljivoj za nas da ih po potrebi koristimo
8. **sys.path promenljiva sadrži spisak imena direktorijuma iz kojih se uvoze moduli**

- **sys.argv** predstavlja listu ulaznih argumenata
- `len(sys.argv)` daje broj ulaznih argumenata –
- `sys.argv[0]` je ime programa kojim smo pozvali ovo

Proba.py:

```
import sys
```

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

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

argv len: 1

argv[0]:

C:/Users/Milan/AppData/Local/Programs/Python/Python311/poziv.py



Promenljiva sa nazivom `sys.path` u sebi sadrži spisak imena direktorijuma iz kojih Python uvozi module.

```
import sys
```

```
for i in sys.path:
```

```
    print('Tražim module u:', i)
```



## Rezultat:

Tražim module u: C:/Users/Milan/AppData/Local/Programs/Python/Python311

Tražim module u:

C:\Users\Milan\AppData\Local\Programs\Python\Python311\Lib\idlelib

Tražim module u:

C:\Users\Milan\AppData\Local\Programs\Python\Python311\python311.zip

Tražim module u: C:\Users\Milan\AppData\Local\Programs\Python\Python311\DLLs

Tražim module u: C:\Users\Milan\AppData\Local\Programs\Python\Python311\Lib

Tražim module u: C:\Users\Milan\AppData\Local\Programs\Python\Python311

Tražim module u:

C:\Users\Milan\AppData\Local\Programs\Python\Python311\Lib\site-packages



- Python prvo uvozi module koji se nalaze u direktorijumu našeg programa.
- Znači, ukoliko želimo da NAŠ modul bude dostupan nekom našem programu, dovoljno je da taj modul prebacimo u isti folder u kojem je i naš program.





# Kako se pišu moduli?

Primer koda jednostavnog modula, koji može da se sačuva na folderu u fajlu **mojmodul1.py**:

```
# Moj prvi modul
```

```
def primermodula():
```

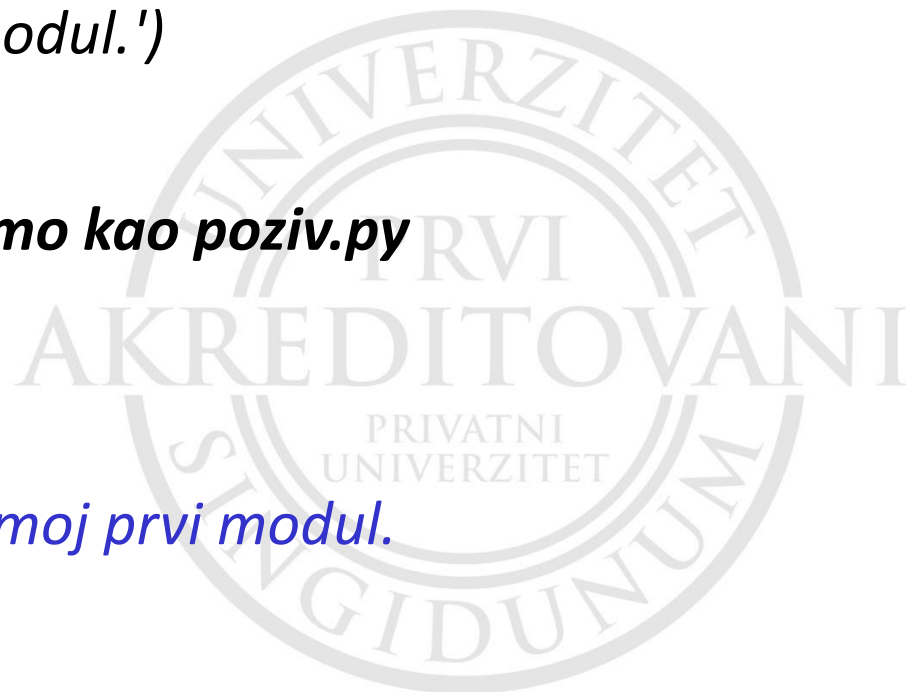
```
    print('Zdravo! Ovo je moj prvi modul.')
```

**POZIV MODULA: ovo dole sačuvamo kao poziv.py**

```
import mojmodul1
```

```
mojmodul1.primermodula()
```

**Rezultat:** *Zdravo! Ovo je moj prvi modul.*



- Prebacimo modul mojmodul.py:
- Iz direktorijuma Documents u dir Downloads
- Probajmo sad izvršenje:
  - `from mojmodul`
  - `import primermodula`
- Javi se greška
  - **Traceback (most recent call last):**
  - **File "C:/Users/Milan/Documents/test.py", line 1, in <module>**
  - **`from mojmodul import primermodula`**
  - **ModuleNotFoundError: No module named 'mojmodul'**
- Prebacimo sad u :
- C:\Users\Milan\AppData\Local\Programs\Python\Python311\DLLs ‘,
- **Sad radi**

Modul treba biti smešten u istom direktorijumu kao i program u koji ga uvozimo, ili u jednom od direktorijuma navedenih u sys.path.



# Drugi način poziva modula

Druga verzija koristi

**From import**

```
#import mojmodul
```

```
#mojmodul.primermodula()
```

```
from mojmodul import primermodula
```

```
primermodula()
```

*Rezultat:*

*Zdravo! Ovo je moj prvi modul.*



Vratimo mojmodul.py u Documents

Ako stavimo \*, onda uvodimo sve promenljive a ne samo primermodula()

```
#import mojmodul
```

```
#mojmodul.primermodula()
```

```
#from mojmodul import primermodula
```

```
#primermodula()
```

```
from mojmodul import *
```

```
primermodula()
```

*Rezultat:*

*Zdravo! Ovo je moj prvi modul.*



Modul može da sadrži više funkcija:

**mojmodul.py**

```
def primermodula():
```

```
    print('Zdravo! Ovo je moj prvi modul.')
```

```
def primermodula2():
```

```
    print('Ovo je druga funkcija')
```

```
def primermodula3():
```

```
    print('Ovo je treća funkcija')
```



Pozivanje modula:

**Pozivmojmodul.py**

```
import mojmodul
```

```
mojmodul.primermodula()
```

```
mojmodul.primermodula2()
```

```
mojmodul.primermodula3()
```

*Rezultat:*

*Zdravo! Ovo je moj prvi modul.*

*Ovo je druga funkcija*

*Ovo je treća funkcija*



Зен Python-a

***Import this***

Rezultat:

The Zen of Python, by Tim Peters

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!



# Funkcija dir

```
import sys  
print(dir(sys))
```

Ugradjena dir  
funkcija **izlistava**  
identifikatore  
(funkcije i  
promenljive)  
**definisane u tom**  
**modulu**

```
['__breakpointhook__', '__displayhook__', '__doc__', '__excepthook__',  
 '__interactivehook__', '__loader__', '__name__', '__package__', '__spec__',  
 '__stderr__', '__stdin__', '__stdout__', '_base_executable',  
 '_clear_type_cache', '_current_frames', '_debugmallocstats',  
 '_enablelegacywindowsfsencoding', '_framework', '_getframe', '_git', '_home',  
 '_xoptions', 'api_version', 'argv', 'base_exec_prefix', 'base_prefix',  
 'breakpointhook', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats',  
 'copyright', 'displayhook', 'dllhandle', 'dont_write_bytecode', 'exc_info',  
 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info',  
 'float_repr_style', 'get_asyncgen_hooks',  
 'get_coroutine_origin_tracking_depth', 'get_coroutine_wrapper',  
 'getallocatedblocks', 'getcheckinterval', 'getdefaultencoding',  
 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getprofile',  
 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval', 'gettrace',  
 'getwindowsversion', 'hash_info', 'hexversion', 'implementation', 'int_info',  
 'intern', 'is_finalizing', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path',  
 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'set_asyncgen_hooks',  
 'set_coroutine_origin_tracking_depth', 'set_coroutine_wrapper',  
 'setcheckinterval', 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace',  
 'stderr', 'stdin', 'stdout', 'thread_info', 'version', 'version_info', 'warnoptions',  
 'winver']
```

```
import math
```

```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',  
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',  
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',  
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',  
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2',  
'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',  
'tanh', 'tau', 'trunc']
```

```
import math
```

```
print(dir(math))
```

```
from math import sin, pi
```

```
print('Pi je približno', pi)
```

```
print('sin(0) =', sin(0))
```

```
print(math.sqrt(9))
```

Pi je približno 3.141592653589793  
sin(0) = 0.0  
3.0

A može i ovako bez linije 3 (koja je razlika?):

```
import math
print(dir(math))
#from math import sin, pi
print('Pi je približno', math.pi)
print('sin(0) =', math.sin(0))
print(math.sqrt(9))
```



```
import mojmodul  
print(dir(mojmodul))
```

Vraća listu imena iz ovog modula (mojmodul)

Rezultat:

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',  
  '__name__', '__package__', '__spec__', 'primermodula']
```



Rad van IDLE (u konzoli):

```
>>> dir()
```

Rezultat:

```
['__annotations__', '__builtins__', '__doc__', '__loader__',  
  '__name__', '__package__', '__spec__']
```

Ako definišemo dve promenljive: a,b

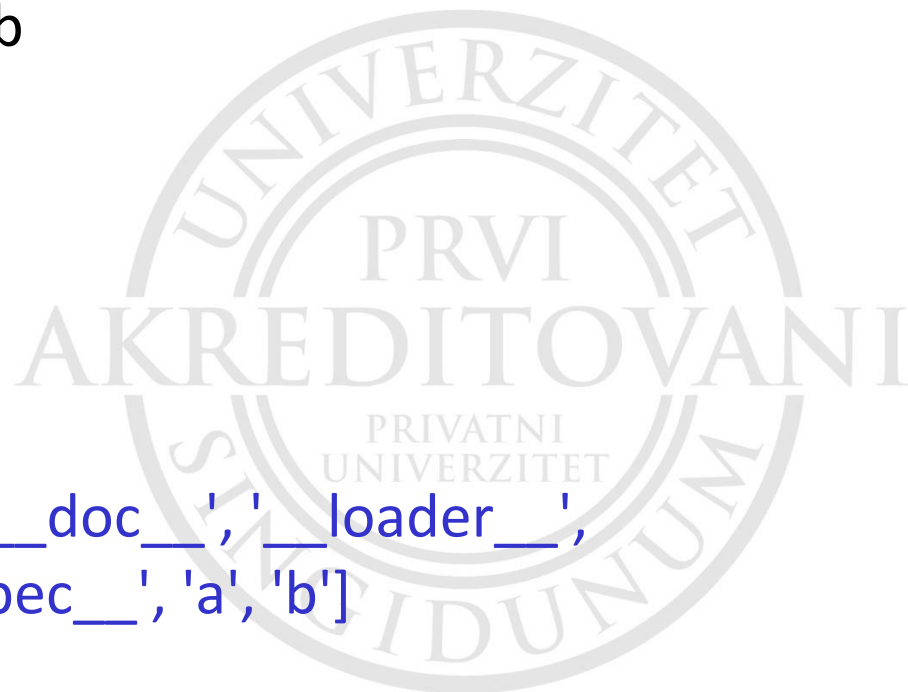
```
>>> a=5
```

```
>>> b="Milan"
```

```
>>> dir()
```

Rezultat:

```
['__annotations__', '__builtins__', '__doc__', '__loader__',  
  '__name__', '__package__', '__spec__', 'a', 'b']
```



Kada izbrišemo te dve promenljive neće se prikazivati sa dir()

Kako se brišu promenljive?

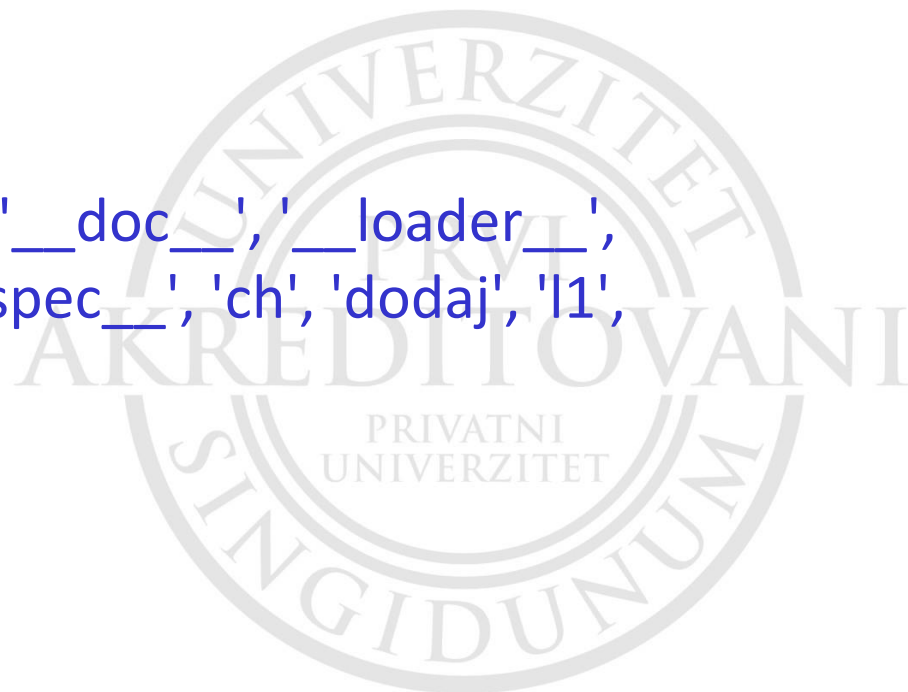
```
>>> del a
```

```
>>> del b
```

```
>>> dir()
```

Rezultat:

```
['__annotations__', '__builtins__', '__doc__', '__loader__',  
  '__name__', '__package__', '__spec__', 'ch', 'dodaj', 'l1',  
  'matrica']
```



dir() funkcija se može primeniti na bilo koji objekat.

## **PROBATI u konzoli:**

dir(print) - atributi prikaza,

dir(str) - atributi klase stringova.

Ili probati u IDLE:

```
print(dir(str))
```

Da li može :print(dir(print)) ?

da

Pakovanje -hijerarhija i način organizovanja svojih programa:

- Promenljive obično idu unutar funkcija.
- Funkcije i globalne promenljive obično idu unutar modula.

## 9.5.2 Moduli i paketi

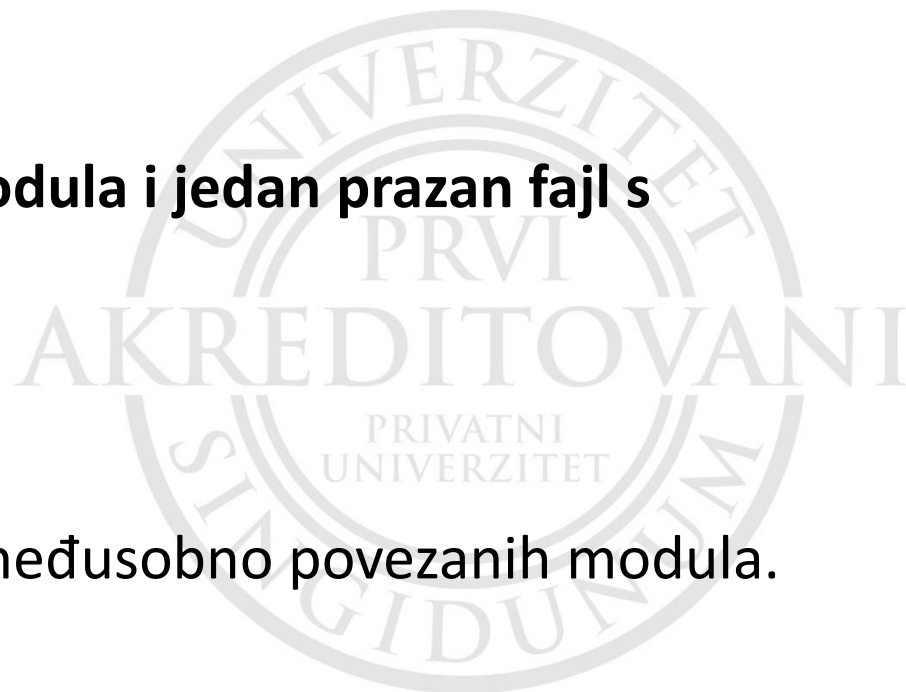
**Modul u jeziku Python je programski fajl, koji sadrži kod koji će se koristiti u drugim programima.**

Veliki broj modula na istom nivou otežava njihovu upotrebu, pa je uveden **mehanizam paketa, koji omogućava hijerarhijsku organizaciju modula.**

**Paket je folder koji sarži skup modula i jedan prazan fajl s rezervisanim nazivom :**

`__init__.py`

Na folderu paketa čuva se skup međusobno povezanih modula.





Npr. sadržaj foldera paketa sa modulima za rad sa slikama može biti:

U folderu slike napraviti 5 modula imena (mojmodul1.py) sa sadržajem:

```
def primer1():
```

```
    print('Ovo je primer 1')
```

```
....
```

```
def primer2():
```

```
    print('Ovo je primer 2')
```

```
def primer5():
```

```
    print('Ovo je primer 5')
```

```
    print(__name__ )
```

*Sadržaj foldera slike:*

*slike/*

*\_\_init\_\_.py*

*mojmodul1.py*

*mojmodul2.py*

*mojmodul3.py*

*mojmodul4.py*

*mojmodul5.py*

memoriserati Poziv.py u direktorijumu iznad slike i probati:

Poziv.py:

```
import mojmodul5
```

```
mojmodul5.primer5()
```

**Ne Radi**

Prebaciti Poziv.py u direktorijum SLIKE i probati:

**Radi**



Prilikom uvoza ovih modula, **nazivu fajla se kao prefiks dodaje naziv foldera** na kome se nalaze, npr. za uvoz modula `mojmodul1.py` treba navesti:

```
import slike.mojmodul1  
slike.mojmodul1.primer1()
```

```
import slike.mojmodul2  
slike.mojmodul2.primer2()
```

***Sad radi!***



Ponekad je pogodno da se kompletan paket modula učitava jednom naredbom.

Zato je potrebno u fajlu `__init__.py` pripremiti spisak modula u posebnoj promenljivoj rezervisanog naziva `__all__`, npr.

```
__all__=["mojmodul1","mojmodul2","mojmodul3","mojmodul4","mojmodul5"]
```

Za uvoz svih modula jednog paketa tada je dovoljna jedna naredba, npr.

```
from slike import *
```

**Documents\pozivmojmodulPrimer1.py**

```
from slike import *  
mojmodul1.primer1()
```

Rezultat: Ovo je primer 1

**Probati promenuti `__all__` da bude bez  
mojmodul1:**

```
__all__=["mojmodul2","mojmodul3","mojmodul4","mojmodul5"]
```

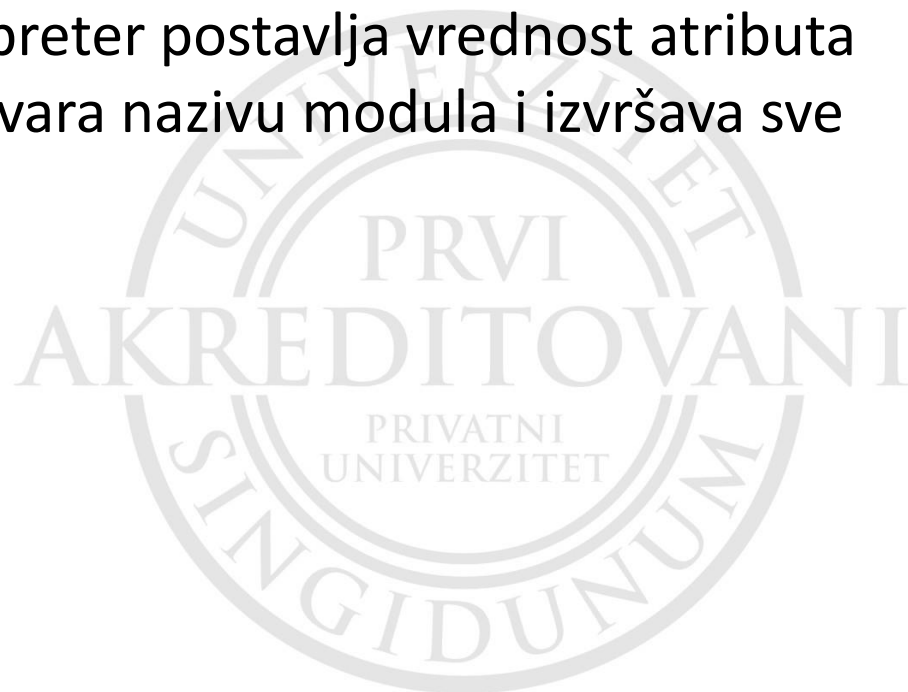
***Sada neće raditi pozivmojmodulPrimer1***

## 9.5.3 Upotreba modula

U kodu modula se na nivou glavnog programa može izvršiti provera načina njegove upotrebe:

da li se koristi kao zaseban program ili kao modul.

Prilikom učitavanja modula, interpreter postavlja vrednost atributa **\_\_name\_\_** na vrednost koja odgovara nazivu modula i izvršava sve naredbe modula.



U slučaju da pokrenemo neki program1, u samom tom programu

`__name__` će biti jednako "`__main__`",  
Odnosno jednako imenu glavne metode

Dok u slučaju da pokrenemo program2 koji poziva program1 naredbom import

`__name__` će biti jednako "program1".



## Program1.py:

```
def primer2():  
    print('Ovo je primer 2')  
    print(__name__ )  
primer2()
```

Rezultat:

Ovo je primer 5

\_\_main\_\_



Sadržaj: \_\_name\_\_





# 3.čas



## Program2.py:

```
import program1  
program1.primer2()
```

Sadržaj: \_\_name\_\_

Rezultat:

Ovo je primer 2

Sadržaj promenljive je: program1



Za glavni program koji koristi modul, vrednost atributa

`__name__` je:

`"__main__"`,

tako da je pogodno na najvišem nivou odvojiti kod modula i aplikativnog programa jednostavim ispitivanjem:

```
if __name__ == '__main__':
```

```
<glavni program>
```

Na taj način se programski kod modula na osnovnom nivou neće izvršiti, jer nije namenjen inicijalizaciji modula.



```
def primer5():
```

```
    print('Ovo je primer 5')
```

```
    print(__name__)
```

```
    if __name__ == '__main__':
```

```
        print("ovo je glavni program")
```

```
    else:
```

```
        print("ovo nije glavni program")
```

```
primer5()
```

Ako se pokrene primer5.py, rezultat je:

Ovo je primer 5

\_\_main\_\_

ovo je glavni program

Ako se pokrene iz programa:

```
from test import *
```

```
mojmodul5.primer5()
```

Rezultat je:

Ovo je primer 5

test.mojmodul5

ovo nije glavni program

Ovo je primer 5

test.mojmodul5

ovo nije glavni program

## 9.6 Primer programa

Primer upotrebe modula u razvoju softvera je mala biblioteka grafičkih funkcija razvijenih korišćenjem postojećih funkcija modula turtle, koje se mogu višestruko upotrebljavati i pojednostaviti programiranje.

Modul **POZIVCRTANJA.PY** sadrži nekoliko jednostavnih funkcija potrebnih u vektorskoj grafici za :

- crtanje linije između dve zadane tačke,
- crtanje tačke ili ispis teksta na zadanim koordinatama,
- crtanje kružnice zadanog poluprečnika s centrom u zadanoj tački,
- crtanje pravougaonika zadane širine i visine s centrom u zadanoj tački.

1. Kreirati folder recimo: turtle
2. Napraviti 5 modula
3. Svaki testirati
4. Napraviti glavni program
5. testirati



## 2.1. kakav je ovo modul?

```
def modul1(x1, y1, x2, y2):
```

```
    turtle.penup()          # podizanje pera
```

```
    turtle.goto(x1, y1)
```

```
    turtle.pendown()
```

```
    turtle.goto(x2, y2)
```

```
import turtle
```

```
modul1(100,100,-100,-100)
```

*Ovo je modul za?*

*Crtanje linije od (x1, y1) do (x2, y2)*

*Memorisati modul: crtanjelinije.py*

*Kako pokrenuti ovaj modul?*

Kakav je ovo modul?

*def modul2(s, x, y):*

*turtle.penup()*                      *# podizanje pera*

*turtle.goto(x, y)*

*turtle.pendown()*                  *# spuštanje pera*

*turtle.write(s)*

*Šta radi ovaj modul?*

*Ispis teksta od koordinata (x, y). Kako se poziva?*

*import turtle*

*modul2("Zdravo",50,50)*

*Memorisati modul: ispisteksta.py*





## Šta radi ovaj modul?

*def modul3(x, y):*

*turtle.penup()* *# podizanje pera*

*turtle.goto(x, y)*

*turtle.pendown()* *# spuštanje pera*

*turtle.begin\_fill()*

*turtle.circle(3)*

*turtle.end\_fill()*

*import turtle*

*modul3(200,200)*

*Šta radi modul?*

*# Crtanje tačke na koordinatama (x, y)*

*Memorisiati modul: crtanjetačke.py*



Kako dodati da je i poluprečnik promenljiva?

# Crtanje tačke na koordinatama (x, y) sa poluprecnikom z

def modul3(x, y, z):

turtle.penup() # podizanje pera

turtle.goto(x, y)

turtle.pendown() # spuštanje pera

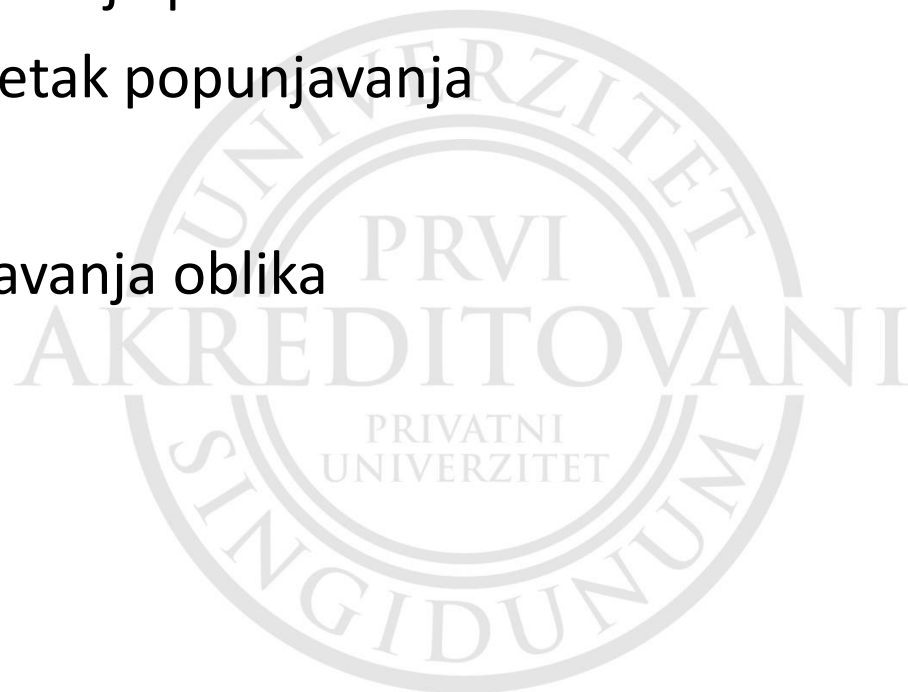
turtle.begin\_fill() # početak popunjavanja

turtle.circle(z)

turtle.end\_fill() # popunjavanja oblika

import turtle

modul3(200,200,25)



## Šta radi ovaj modul?

*def modul4(x=0, y=0, radius=100):*

*turtle.penup()* *# podizanje pera*

*turtle.goto(x, y)*

*turtle.pendown()* *# spuštanje pera*

*turtle.circle(radius)*

*import turtle*

*modul4()*

## Šta radi modul?

*# Crtanje kruga zadanog poluprečnika s centrom u tački (x,y)*

*Memorisati modul: crtanjekruga.py*



Kako nacrtati da krug bude na sredina ekrana?

# Crtanje kruga zadanog poluprečnika s centrom u(x,y)

def modul4(x=0, y=0, radius=100):

turtle.penup() # podizanje pera

turtle.goto(x, y-radius)

turtle.pendown() # spuštanje pera

turtle.circle(radius)

import turtle

modul4()



## Šta radi ovaj modul?

```
def modul5(x=0, y=0, width=100, height=200):
```

```
    turtle.penup()                # podizanje pera
```

```
    turtle.goto(x, y)
```

```
    turtle.pendown()            # spuštanje pera
```

```
    turtle.right(90)
```

```
    turtle.forward(height)
```

```
    turtle.right(90)
```

```
    turtle.forward(width)
```

```
    turtle.right(90)
```

```
    turtle.forward(height)
```

```
    turtle.right(90)
```

```
    turtle.forward(width)
```

```
import turtle
```

```
modul5()
```

*Šta radi?*

*# Crtanje pravougaonika zadanih dimenzija od (x, y)*

*Memorisati modul: crtanjepravougaonika.py*



## Kako napraviti da bude u centru ekrana?

```
# Crtanje pravougaonika zadanih dimenzija od (x, y)
def drawRectangle(x=0, y=0, width=100, height=200):
    turtle.penup()                # podizanje pera
    turtle.goto(x + width/2, y + height/2)
    turtle.pendown()              # spuštanje pera
    turtle.right(90)
    turtle.forward(height)
    turtle.right(90)
    turtle.forward(width)
    turtle.right(90)
    turtle.forward(height)
    turtle.right(90)
    turtle.forward(width)
import turtle
drawRectangle()
```



## 4. GLAVNI PROGRAM

Na osnovu modula, s ovako definisanim funkcijama za crtanje, mogu se pisati kraći programi, kao što je npr. kratki testni program koji po jednom poziva svaku od funkcija modula korisneTurtleFunkcije:

### **POZIVCRTANJA.PY**

```
import turtle
```

```
from crtanjelinije import *
```

```
# Crtanje linije između(-80,-80) i (80,80)
```

```
modul1(-80, -80, 80, 80)
```

ZAŠTO CRTA 2 LINIJE?



## KOMPLETAN GLAVNI PROGRAM

```
import turtle

from crtanjelinije import modul1

# Crtanje linije između(-80,-80) i (80,80)
modul1(-80, -80, 80, 80)

from ispisteksta import modul2
modul2("Crtanje u Pythonu",0,0)

from crtanjetačke import modul3
modul3(100,100)


from crtanjekruga import modul4
modul4(100,100,200)

from crtanjepravougaonika import modul5
modul5(0,0,100,150)
```

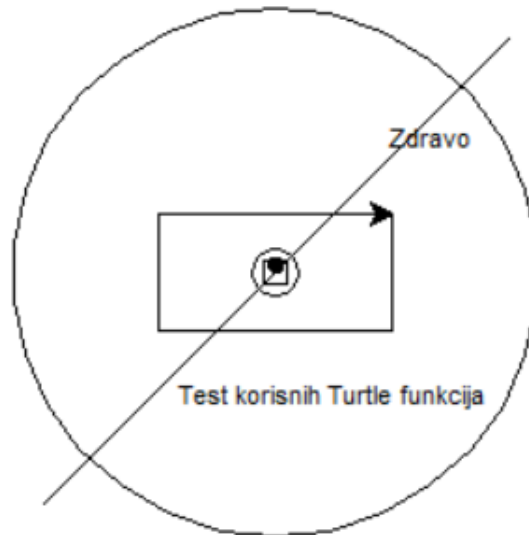




Rezultat izvršavanja programa prikazuje se u grafičkom prozoru:

 Python Turtle Graphics

— □ ×



# Znači imamo 3 načina poziva modula

1. Nakon što smo uključili matematički modul koji nam je zatrebao za računanje korena, isti taj modul povezujemo s funkcijom tako što **ispred imena funkcije napišemo ime modula i povežemo ih tačkom.**

```
import math #poziv modula  
print (math.sqrt(25))
```

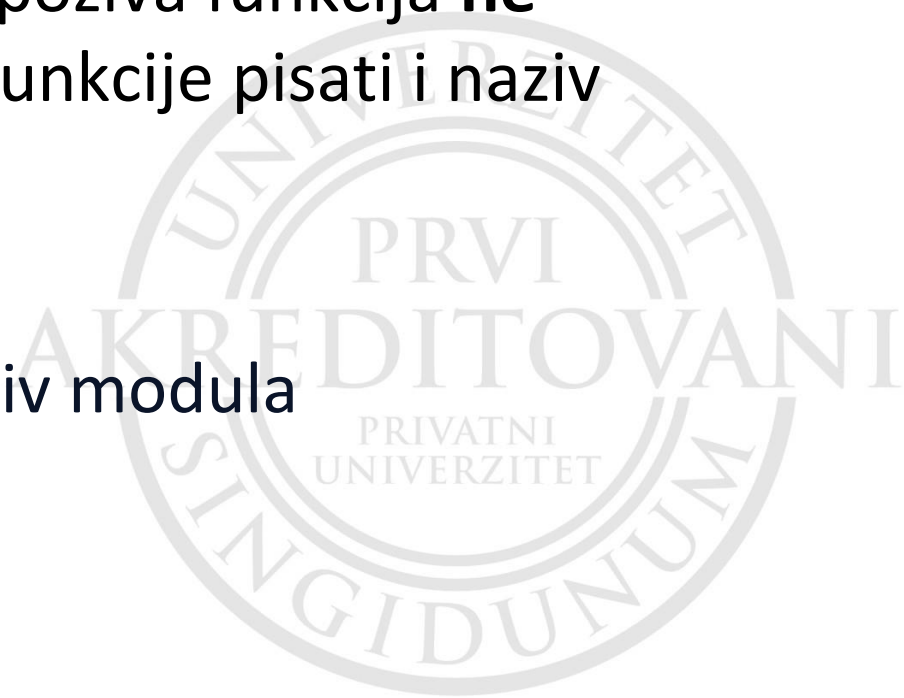


2. Ako znamo tačno koja nam funkcija treba, uključivanje radimo tako što

- **napišemo ime modula iz kojeg uključujemo**
- **funkciju zatim ime funkcije ili više njih odvojenih zarezom.**

Posle ovog uključivanja kod poziva funkcija **ne trebamo** više ispred imena funkcije pisati i naziv modula

```
from math import sqrt,sin #poziv modula  
print (sqrt(81))  
print (sin(0))
```



### 3. Umesto imena funkcije stavlja se znak \*

**Znak \*** simbolizuje sve funkcije koje se nalaze u modulu, tako da se nakon takvog uključivanja mogu upotrebljavati sve funkcije iz modula i to bez pisanja imena modula kao prefiksa.

```
from math import * #poziv modula  
print (sqrt(25))  
print (sin(0))
```



1 način:

```
import math #poziv modula
```

```
print (math.factorial(5))
```

```
print (math.log(1))
```

2. Način

```
from math import factorial, log
```

```
print (factorial(5))
```

```
print (log(1))
```

3. Način

```
from math import *
```

```
print (factorial(5))
```

```
print (log(1))
```



# Ugradjeni moduli

```
import platform
```

```
x = platform.system()
```

```
print(x)
```

#Šta je rezultat?

Windows

```
import platform
```

```
x = platform.processor()
```

```
print(x)
```

#Šta je rezultat?

Intel64 Family 6 Model 126

Stepping 5, GenuineIntel

```
import platform  
x = platform.machine()  
print(x)  
#Šta je rezultat?
```

AMD64

```
import platform  
  
x = dir(platform)  
print(x)  
#Šta je rezultat
```

lista svih atributa i metoda koje modul platform sadrži. Funkcija dir() vraća alfabetski sortiranu listu imena dostupnih atributa (funkcija, klasa i varijabli

# Rezultat :

```
['_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',  
 '__builtins__', '__cached__', '__copyright__', '__doc__', '__file__', '__loader__',  
 '__name__', '__package__', '__spec__', '__version__', '_comparable_version',  
 '_component_re', '_default_architecture', '_follow_symlinks',  
 '_get_machine_win32', '_ironpython26_sys_version_parser',  
 '_ironpython_sys_version_parser', '_java_getprop', '_libc_search',  
 '_mac_ver_xml', '_node', '_norm_version', '_os_release_cache',  
 '_os_release_candidates', '_os_release_line', '_os_release_unescape',  
 '_parse_os_release', '_platform', '_platform_cache', '_pypy_sys_version_parser',  
 '_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file',  
 '_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output',  
 '_ver_stages', 'architecture', 'collections', 'freedesktop_os_release', 'functools',  
 'itertools', 'java_ver', 'libc_ver', 'mac_ver', 'machine', 'node', 'os', 'platform',  
 'processor', 'python_branch', 'python_build', 'python_compiler',  
 'python_implementation', 'python_revision', 'python_version',  
 'python_version_tuple', 're', 'release', 'sys', 'system', 'system_alias', 'uname',  
 'uname_result', 'version', 'win32_edition', 'win32_is_iot', 'win32_ver']
```



## **Mymodule.py:**

```
person1 ={  
    "name":"John",  
    "age": 36,  
    "country":  
    "Norway"  
}  
print(person1["age"])
```

## **Variable u Modulu**



```
person1 ={  
    "name":"John",  
    "age": 36,  
    "country":  
    "Norway"  
}  
#print(person1["age"])
```

```
pozivmodule.py:  
import mymodule  
print(mymodule.person1["age"])
```

## **Mymodule1.py**

```
person1 ={  
    "name":"John",  
    "age": 36,  
    "country":  
    "Norway"  
}
```

```
person2 ={  
    "name":"Petar",  
    "age": 63,  
    "country":  
    "Serbia"  
}  
print(person1["age"])  
print(person2["name"])
```

**36**  
**Petar**  
**>>>**



## Uvezi iz modula

Možete izabrati da uvezete samo delove iz modula, koristeći ključnu reč **from**.

### **Pozivmodule1.py**

```
from mymodule1 import person2  
print (person2["name"])
```

36

Petar

Petar

>>>



16.12.2024.



```
from mymodule1 import person2, person1  
print (person2["name"])  
print(person1["age"])
```

Petar  
36



a može i ovako:

```
from mymodule1 import *  
print (person2["name"])  
print(person1["age"])
```

