

# Programiranje 1

## Funkcije u programskom jeziku Python

Univerzitet u Beogradu  
Elektrotehnički fakultet  
2019/2020.

# Sadržaj

---

- Uvod u funkcije
- Definisanje funkcije
- Pozivanje funkcije
- Prenos parametara
- Rekurzija
- Prostor imena

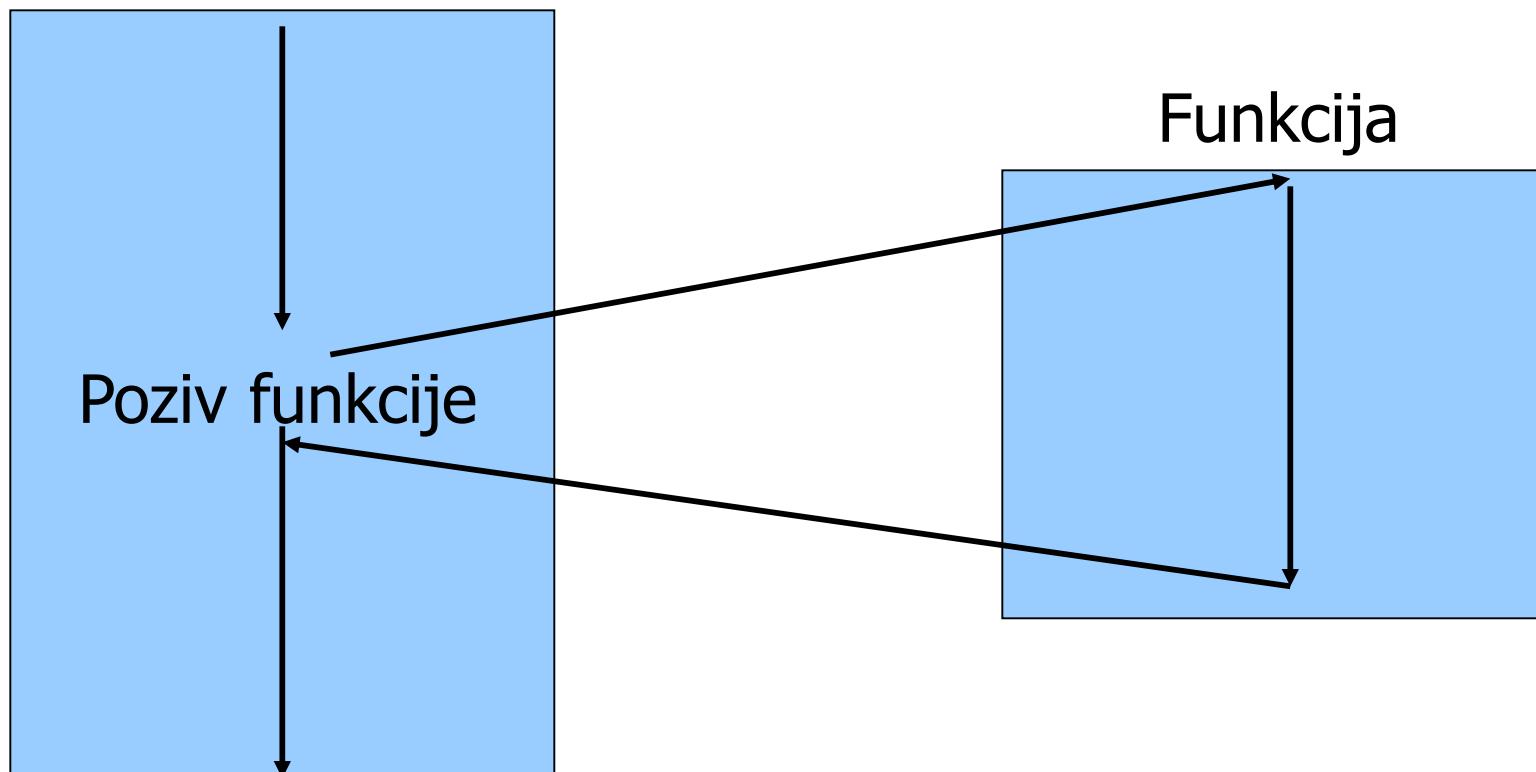
# Uvod u funkcije (1)

---

- Glavni cilj: modularizacija programskog koda
- Odgovara razvoju programa u koracima preciziranja
- Problem se razbija na manje,  
dobro definisane celine koje su slabo spregnute
  - Korišćenjem argumenata
- Omogućava se razvoj i testiranje nezavisnih celina
  - Potencijalno od strane više programera
- Rešenje postaje jednostavnije za sagledavanje
- Sistem se lakše održava
- Podržani principi: *apstrakcija* i *dekompozicija*

# Uvod u funkcije(2)

Glavni program ili  
druga funkcija



# Definisanje funkcije (1)

---

- Funkcija je blok čiji početak označava ključna reč **def** iza koje sledi ime funkcije i zagrade ( )
- Ukoliko postoje parametri (argumenti) funkcije, oni se navode između tih zagrada
- Zatim se opcionalo navodi dokumentacioni string funkcije *docstring*
- Blok koji sadrži naredbe funkcije počinje znakom : i nazubljen je
- Funkcija sadrži naredbu povratka iz funkcije **return**
  - Uz **return** se opcionalo navodi izraz, čiji rezultat se vraća pozivajućem (pot)programu
  - Naredba **return** uz koju nije naveden izraz ekvivalentna je sa **return None**

# Definisanje funkcije (2)

---

Sintaksa:

```
def functionname( parameters ):  
    """function_docstring"""  
    function_suite  
    return [expression]
```

# Definisanje funkcije (3)

---

- Primer funkcije: ispis stringa

```
def ispis ( str ):  
    """ Ispis stringa str  
    zadatog kao parametar """  
    print(str)  
    return
```

- Poziv funkcije

```
ispis ("Dobar dan!")  
ispis ("Python je popularan jezik")
```

# Argumenti (parametri) funkcije

---

- Argumenti (parametri) koji se navode između zagrada u definiciji funkcije nazivaju se **formalni argumenti**
- Argumenti koji se navode između zagrada prilikom pozivanja funkcije nazivaju se **stvarni argumenti**
- Promenom stvarnih argumenata kod različitih poziva postiže se ponovno korišćenje koda (*reusability*)
- Postiže se i ušteda memorijskog prostora koji bi inače sadržao replicirani kod i time je sam program kraći

# Prenos argumenata

```
def parnost( i ):  
    """  
        Vraca True ako je ceo broj i paran,  
        a False ako je neparan  
    """
```

Formalni argument

```
    print("unutar funkcije parnost")  
    return i%2 == 0
```

Poziv funkcije:

```
res = parnost (3)  
print(res)
```

Stvarni argument

# Prenos argumenata - teorija

---

- U različitim programskim jezicima:
  - **Prenos po vrednosti (C)**
    - Stvarni argument je izraz koji se izračunava i njegova vrednost dodeljuje formalnom argumentu
    - Promena vrednosti formalnog argumenta unutar funkcije nema uticaja na stvarni argument
  - **Prenos po referenci (adresi) (Pascal VAR, Perl)**
    - U funkciju se prenosi referenca (adresa) stvarnog argumenta
    - Promenom vrednosti formalnog argumenta unutar funkcije menja se stvarni argument
  - **Prenos po imenu (Algol 60)**
    - Efekat kao da se u telu funkcije formalni argument zamenjuje stvarnim argumentom
    - Argument se evaluira pri korišćenju, a ne pri pozivu

# Prenos argumenata u Python-u

---

- Zbunjujuće!
  - Neki izvori kažu da je prenos argumenata u Python-u po vrednosti, a neki da je prenos po referenci!
- Koristi se zapravo **prenos po referenci objekta**
- U Python-u je svaka promenljiva objekat
- Objekti mogu biti:
  - Nepromenljivog tipa (*immutable*)  
*bool, int, float, str, tuple, frozenset*
  - Promenljivog tipa (*mutable*)  
*list, set, dict*

# Prenos argumenata u Python-u (1)

---

- Kada se objekat nepromenljivog tipa prenosi kao argument, to je slično kao prenos po vrednosti:
  - Promena formalnog argumenta unutar funkcije ne utiče na stvarni argument
  - Prilikom te promene zapravo se formira novi objekat, koji ima novi identitet

```
def ref_demo(x):  
    print("x=", x, " id=", id(x))  
    x=42  
    print("x=", x, " id=", id(x))
```

# Prenos argumenata u Python-u (2)

```
>>> x = 9
```

```
>>> id(x)
```

```
9251936
```

```
>>> ref_demo(x)
```

```
x= 9 id= 9251936
```

```
x= 42 id= 9252992
```

```
>>> id(x)
```

```
9251936
```

```
>>>
```

Unutar funkcije x je formirano kao novi (lokalni) objekat sa vrednošću 42

Nakon izvršenja funkcije imamo prvobitni objekat x sa vrednošću 9

# Prenos argumenata u Python-u (3)

```
>>> def nema_promene(gradovi):
...     print(gradovi)
...     gradovi = gradovi + ["Rim", "London"]
...     print(gradovi)
...
>>> lokacije = ["Beograd", "Novi Sad", "Niš"]
>>> nema_promene(lokacije)
['Beograd', 'Novi Sad', 'Niš']
['Beograd', 'Novi Sad', 'Niš', 'Rim', 'London']
>>> print(lokacije)
['Beograd', 'Novi Sad', 'Niš']
```

Unutar funkcije gradovi je formirano  
kao novi (lokalni) objekat,  
jer mu se dodeljuje vrednost

# Prenos argumenata u Python-u (4)

```
>>> def ima_promene(gradovi):  
...     print(gradovi)  
...     gradovi += ["Rim", "London"]  
...     print(gradovi)  
...  
>>> lokacije = ["Beograd", "Novi Sad", "Niš"]  
>>> ima_promene(lokacije)  
['Beograd', 'Novi Sad', 'Niš']  
['Beograd', 'Novi Sad', 'Niš', 'Rim', 'London']  
>>> print(lokacije)  
['Beograd', 'Novi Sad', 'Niš', 'Rim', 'London']
```

Unutar funkcije operatorom += gradovi se modifikuju po "in place" principu

# Prenos argumenata u Python-u (5)

```
>>> def bez_promene(gradovi):  
...     print(gradovi)  
...     gradovi += ["Rim", "London"]  
...     print(gradovi)  
...  
>>> lokacije = ["Beograd", "Novi Sad", "Niš"]  
>>> bez_promene(lokacije[:])  
['Beograd', 'Novi Sad', 'Niš']  
['Beograd', 'Novi Sad', 'Niš', 'Rim', 'London']  
>>> print(lokacije)  
['Beograd', 'Novi Sad', 'Niš']
```

U funkciju se prenosi kopija cele liste lokacije[:] pa se modifikuje po "in place" principu samo ta kopija!

# Prenos argumenata u Python-u (6)

---

- Kada postoji više argumenata, stvarni argumenti treba da se slažu sa formalnim argumentima po korektnom pozicionom redosledu
- U Python-u je dozvoljeno i da redosled bude drugačiji, kao i da se neki stvarni argument izostavi, ukoliko se u pozivu funkcije eksplisitno navede ime formalnog argumenta (*keyword parameter*)
- U slučaju da se pri definisanju funkcije za neki formalni parametar navede vrednost (*default parameter*) ta vrednost će se podrazumevati ukoliko se odgovarajući stvarni argument izostavi
- Moguće je i da funkcija ima promenljivi broj argumenata

# Navođenje imena formalnog argumenta

```
>>> def osoba_ispis(ime, starost):  
...     print("Ime: ",ime)  
...     print("Starost: ",starost)  
...     return  
  
>>> osoba_ispis(starost=21, ime="Petar")
```

Ime: Petar

Starost: 21

Redosled stvarnih argumenata je obrnut, ali se navode imena stvarnih argumenata sa dodelom!

```
>>> osoba_ispis("Lana", 19)
```

Ime: Lana

Starost: 19

# Izostavljanje formalnog argumenta

---

```
>>> def osoba_ispis(ime, starost=35):  
...     print("Ime: ",ime)  
...     print("Starost: ",starost)  
...     return  
>>> osoba_ispis(starost=28, ime="Marko")  
Ime: Marko  
Starost: 28  
>>> osoba_ispis(ime="Milica")  
Ime: Milica  
Starost: 35
```

Izostavljanje stvarnog argumenata za formalni argument **starost** uzrokuje dodelu podrazumevane vrednosti 35

# Promenljivi broj argumenata (1)

---

```
>>> def varparfu(*x):  
...     print(x)  
...     return  
>>> varparfu()  
()  
>>> varparfu(71, "ETF", "sala 309")  
(71, 'ETF', 'sala 309')
```

Funkcija može imati proizvoljan broj parametara, nije obavezan ni jedan i to je označeno znakom \*

# Promenljivi broj argumenata (2)

---

```
>>> def aritmeticka_sredina(x, *l):  
...     """ Racuna aritmeticku sredinu  
...     ne-nultog broja brojeva """  
...     sum = x  
...     for i in l:  
...         sum += i  
...     return sum / (1.0 + len(l))
```

Prvi argument koji odgovara x je obavezan, dok je proizvoljan broj argumenata u nastavku neobavezan i organizovan u tuple; \* ispred parametra l to označava!

# Promenljivi broj argumenata – poziv

---

```
>>> aritmeticka_sredina(1,2,3) ← Pakovanje  
2.0 argumenata  
  
>>> l=[5,6,7,8]  
  
>>> aritmeticka_sredina(l)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
    File "<stdin>", line 7, in aritmeticka_sredina  
TypeError: unsupported operand type(s) for /:  
'list' and 'float'  
>>> aritmeticka_sredina(*l) ← Raspakivanje,  
6.5 pa pakovanje argumenata
```

# Funkcija kao parametar funkcije (1)

---

```
>>> def funkcija(fnc,arg):  
...     """ Argument funkcije je druga funkcija  
...     fnc, dok je arg numericki argument """  
...     x=fnc(arg)  
...     return x  
...  
>>> def f1(arg):  
...     return arg+1  
...  
>>> def f2(arg):  
...     return arg-1  
...
```

# Funkcija kao parametar funkcije (2)

---

```
>>> funkcija(f1,5)
6
>>> funkcija(f2,5)
4
>>> import math
>>> funkcija(math.factorial,4)
24
>>> funkcija(math.sqrt,100)
10.0
```

# Lambda funkcije (1)

- Lambda funkcija je tipično mala anonimna (bezimena) funkcija
- Može imati proizvoljan broj argumenata
- Sadrži samo jedan izraz čiji rezultat vraća
  - Nema naredbe `return`
- Sintaksa:

lambda *argumenti* : *izraz*

- Primer

```
>>> (lambda x, y: x + y)(2, 3) # bezimena
5
>>> x = lambda a, b: a * b      # dodeljeno ime x
>>> print(x(5, 6))
30
```

# Lambda funkcije (2)

---

- I ovde moguće izostavljanje i podrazumevane vrednosti parametara kao kod običnih funkcija:

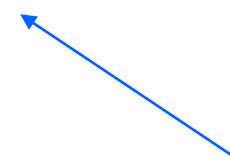
```
>>> (lambda x, y, z: x + y + z)(1, 2, 3)
6
>>> (lambda x, y, z=3: x + y + z)(1, 2)
6
>>> (lambda x, y, z=3: x + y + z)(1, y=2)
6
>>> (lambda *args: sum(args))(1,2,3)
6
```

# Lambda funkcije (3)

---

- Kombinovanje sa "pravom" funkcijom:

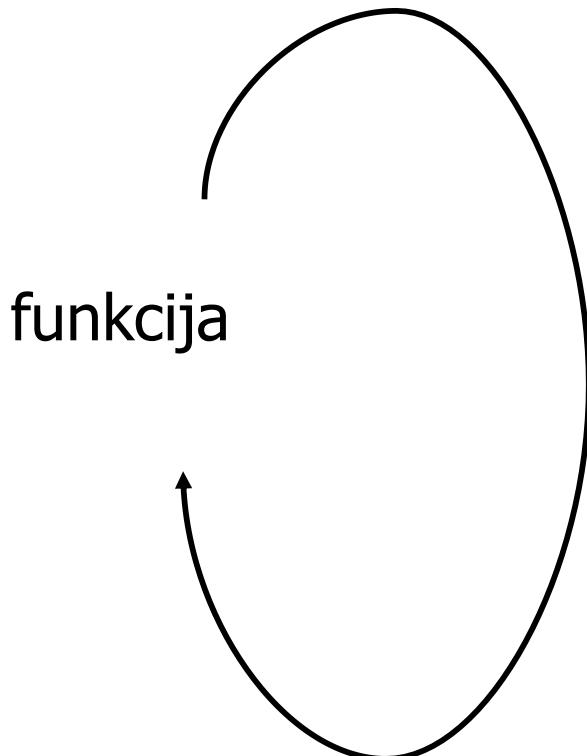
```
>>> def mnozac(n):  
...     return lambda a : a * n  
>>> dublira = mnozac(2)  
>>> triplira = mnozac(3)  
>>> print(dublira(11))  
22  
>>> print(triplira(11))  
33
```



Vraća funkciju kao rezultat  
sa fiksiranim parametrom n!

# Rekurzija

---



- **Direktna rekurzija:**
  - Unutar tela funkcije, ona poziva samu sebe, sa izmenjenim parametrim
  - Mora postojati „nerekurzivni slučaj“ gde se ta funkcija dalje ne poziva, kako bi se proces okončao
    - Terminalni uslov funkcije

# Računanje faktorijela broja (1)

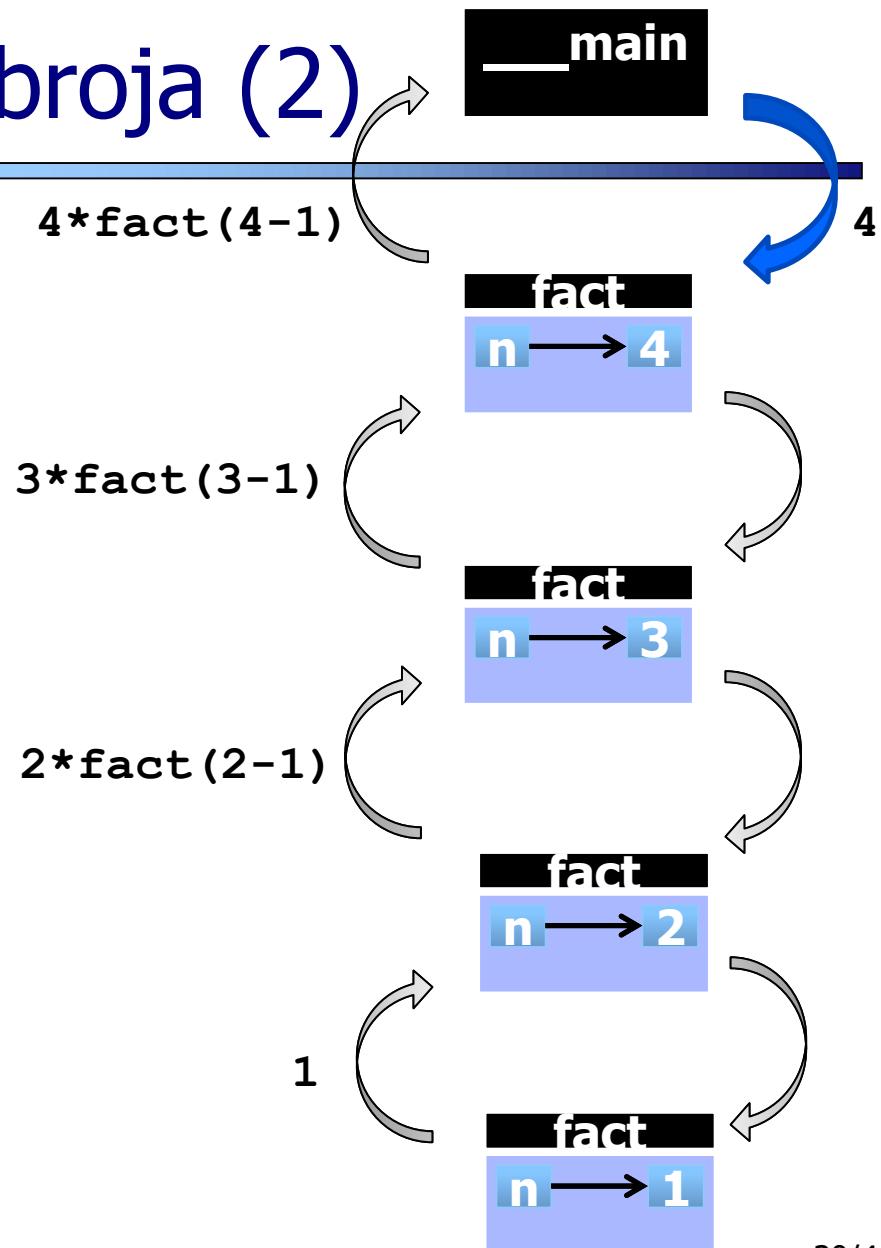
---

- Po definiciji:
  - $n! = n(n-1)!$
  - $F(n) = nF(n-1)$
- Rešavanje:
  - Po definiciji, terminalni uslovi su:
    - $F(0) = 1$
    - $F(1) = 1$
  - Ako je  $n \neq 1$ , rezultat je  $n * F(n-1)$
  - U suprotnom, rezultat je 1
    - Zbog terminalnih uslova

# Računanje faktorijela broja (2)

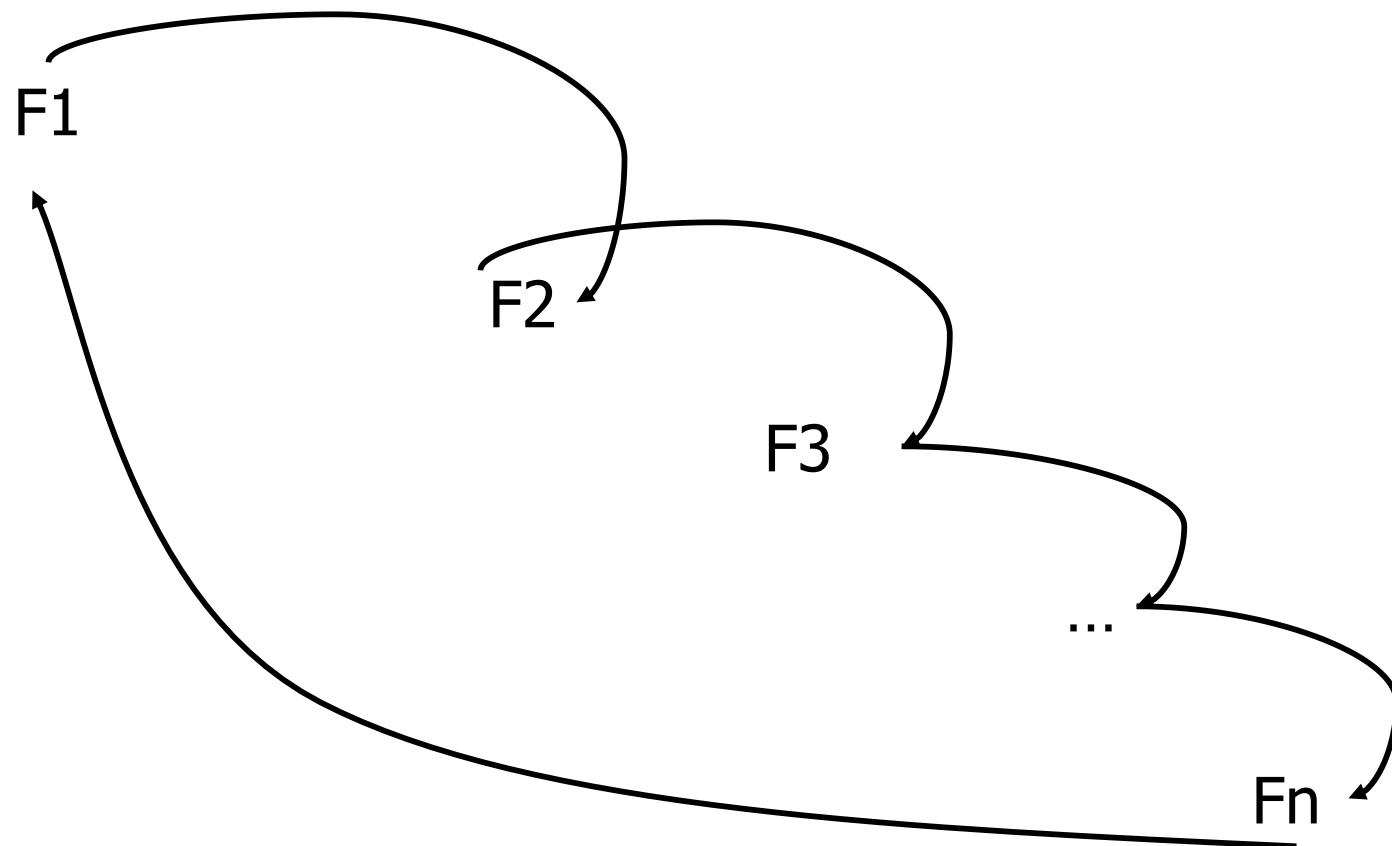
## ○ Rešenje

```
def fact(n):  
    if n==1 or n==0:  
        return 1  
    return fact(n-1)*n
```



# Indirektna rekurzija

---



# Primer indirektne rekurzije

---

```
>>> def ping(i):
...     if i>0:
...         return pong(i-1)
...
...     else:
...         return "0"
>>> def pong(i):
...     if i>0:
...         return ping(i-1)
...
...     else:
...         return "1"
```

# Primer indirektne rekurzije

---

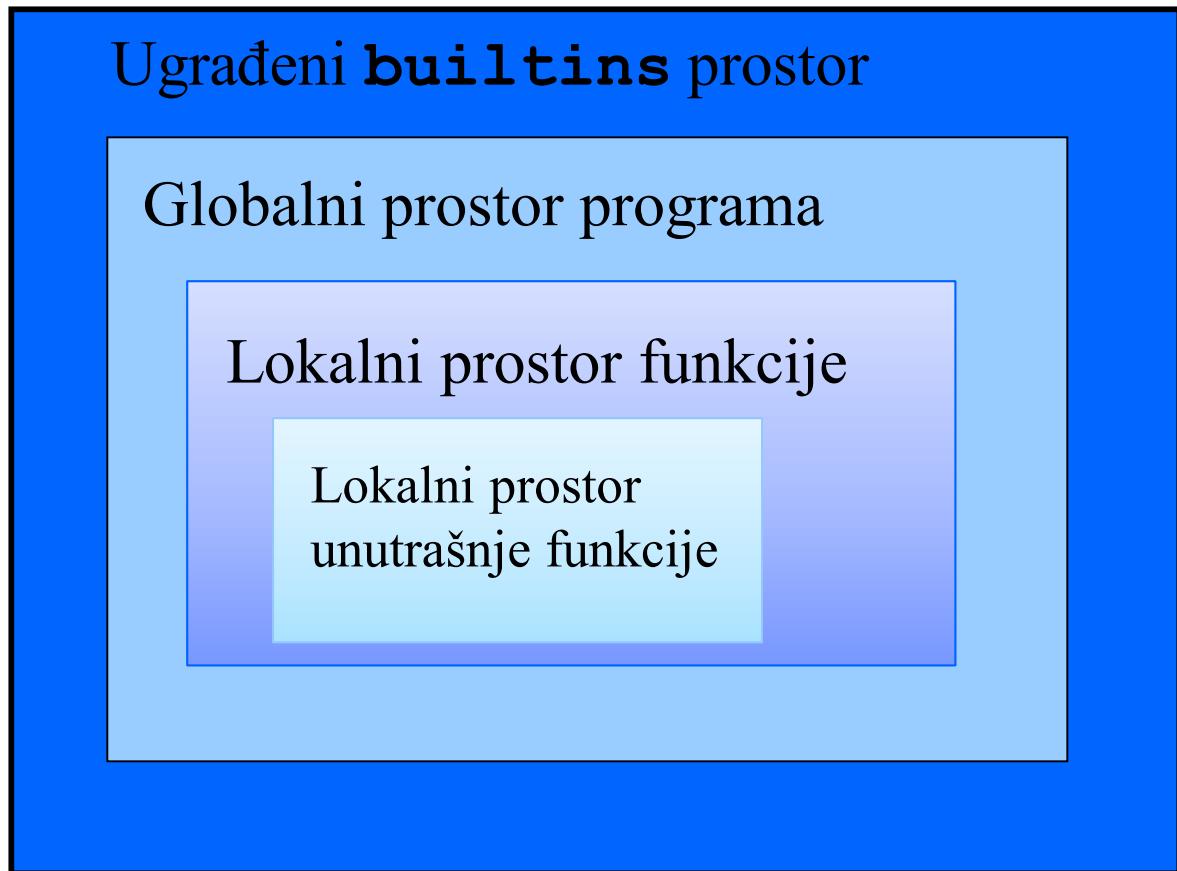
```
>>> print(ping(3))  
1  
          ping(3) -> pong(2) -> ping(1) -> pong(0) -> 1  
  
>>> print(ping(4))  
0  
          ping(4) -> pong(3) -> ping(2) -> pong(1) -> ping(0) -> 0  
  
>>> print(pong(3))  
0  
          pong(3) -> ping(2) -> pong(1) -> ping(0) -> 0  
  
>>> print(pong(4))  
1  
          pong(4) -> ping(3) -> pong(2) -> ping(1) -> pong(0) -> 1
```

# Prostor imena i opseg važenja promenljivih (1)

---

- Promenljive predstavljaju imena za objekte
- Najčešće se veza imena i objekta uspostavlja dodelom vrednosti
- Prostor imena realizuje se internu tabelom imena
- Tokom izvršavanja programa postoje bar dva (osnovna) prostora imena
- Isto ime može postojati u više prostora imena
- Isto ime odnosi se tada na različite objekte

# Prostor imena i opseg važenja promenljivih (2)



- Python dozvoljava da se funkcija definiše unutar druge funkcije
- Ime se najpre traži u lokalnom prostoru imena
- Ako se ne nađe, ide se nivo više u hijerarhiji

# Prostor imena i opseg važenja promenljivih (3)

---

- Po pokretanju interpretera, formira se prostor imena ugrađenog **builtins** prostora
- On sadrži predefinisana imena
  - Npr. `print`
- Kada se program pokrene,  
kreira se globalni prostor programa
- Pri pozivu svake funkcije kreira se  
lokalni prostor imena te funkcije
- Svaki od ovih prostora ima svoj životni vek,  
koji traje tokom izvršavanja odgovarajućeg koda

# Lokalne i globalne promenljive (1)

---

```
>>> def spoljna_funkcija():
...     a = 20
...     def unutrasnja_funkcija():
...         a = 30
...         print('a =',a)
...     unutrasnja_funkcija()
...     print('a =',a)
a = 10
>>> spoljna_funkcija()
>>> print('a =',a)
a = 30
a = 20
a = 10
```

# Lokalne i globalne promenljive (2)

---

```
>>> def spoljna_funkcija2():
...     global b
...     b = 20
...     def unutrasnja_funkcija():
...         global b
...         b = 30
...         print('b =' ,b)
...     unutrasnja_funkcija()
...     print('b =' ,b)
...     b = 10
```

# Lokalne i globalne promenljive (3)

---

```
>>> spoljna_funkcija2()
b = 30
b = 30
>>> print('b =' ,b)
b = 10
```

- Globalnoj promenljivoj može se pristupiti iz svih funkcija
- Neophodno deklarisanje naredbom `global`
- Oprezno upotrebljavati globalne promenljive
  - Kvare enkapsulaciju programskog koda

# Organizacija izvornog koda

---

- Kompleksan programski sistem dekomponuje se na logičke celine
- Modularne sisteme je lakše testirati, nadograđivati i održavati
- Dekompozicija:
  - Na sistemskom nivou (klase, odnosno objekti)
  - Na organizacionom nivou (biblioteke i moduli)
  - Na funkcionalnom nivou (funkcije)

# Koncept modula

---

- Potrebno je omogućiti da se funkcija definisana u jednoj .py datoteci poziva iz druge .py datoteke
- Modul je datoteka sa ekstenzijom .py koja sadrži proizvoljan izvorni kod
  - Najčešće definicije funkcija i definicije globalnih promenljivih
  - Nije namenjena samostalnom izvršavanju, već kao biblioteka funkcija za složenije programe
- Skripta je program smešten u jednoj datoteci koji se pokreće u komandnom režimu operativnog sistema

# Koncept modula - naredba `import`

---

- Uvođenje modula u program – naredba `import`
- Učitavaju se naredbe modula i kreira novi imenski prostor za taj modul
- Funkcije i globalne promenljive postaju vidljive po notaciji `<ime_modula>.<ime_u_modulu>`
- Primer: nakon `import math` koristi `math.sqrt(16)`
- Moguće da se neko ime ponavlja
  - “Jača” su imena u programu nego u uvedenom modulu

# Koncept modula - naredba `import`

---

- Kod naredbe `import` može se koristiti alternativno ime za modul:

```
>>> import math as m
```

```
>>> m.sin(m.pi)
```

```
>>> dir(m) # daje listu imena iz modula
```

- Python tretira sve funkcije i module kao objekte
- Može se uraditi dodela i kasnije izbeći notacija sa tačkom

```
>>> sin = math.sin
```

```
>>> sin(0)
```

# Koncept paketa

---

- Organizuje više modula u jedinstvenu imensku hijerarhiju
  - Moduli se nalaze u zajedničkom direktorijumu kao `.py` fajlovi
  - Direktorijum sadrži obavezno i datoteku `__init__.py`
- Datoteka `__init__.py` može da bude čak i prazna, inače se automatski izvrši pri uvođenju paketa
- Omogućena upotreba različitih biblioteka koje sadrže module sa istim imenom
  - Na primer `paket.modul.fja(arg)`

# Literatura – knjige

---

- M. Kovačević, Osnove programiranja u Pajtonu, Akademска misao, Beograd, 2017.
- M. Lutz, Learning python: Powerful object-oriented programming, 5th edition, O'Reilly Media, Inc., 2013.
- J. Zelle, Python Programming: An Introduction to Computer Science, 3rd Ed., Franklin, Beedle & Associates, 2016.
- D. Beazley, B. K. Jones, Python Cookbook, 3rd edition, O'Reilly Media, 2013.
- A. Downey, J. Elkner, C. Meyers, How To Think Like A Computer Scientist: Learning With Python, free e-book

# Literatura – *online* izvori

---

- Python 3.8.0 documentation,  
<https://docs.python.org/3/index.html>
- Colin Morris, 7-day Python course,  
<https://www.kaggle.com/learn/python>
- Learn Python, Basic tutorial,  
<https://www.learnpython.org/>
- TutorialsPoint, Python tutorial  
<https://www.tutorialspoint.com/python/index.htm>