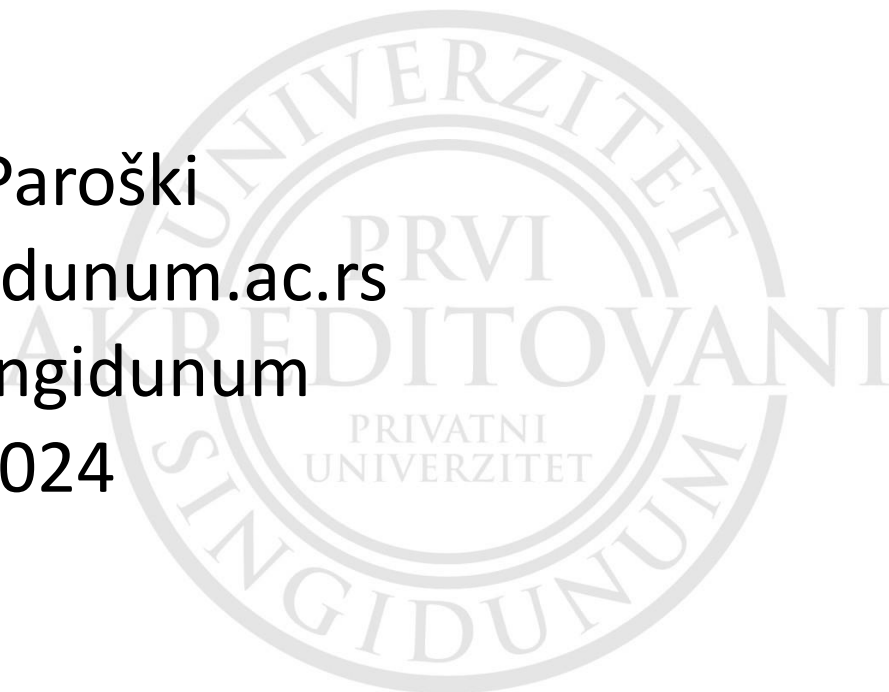


# 5. Funkcije u jeziku Python

Dr Milan Paroški  
[mparoski@singidunum.ac.rs](mailto:mparoski@singidunum.ac.rs)  
Univerzitet Singidunum  
2023/2024



# Sadržaj

1. Uvod
2. Definicija korisničke funkcije
3. Upotreba funkcije
4. Argumenti funkcije
5. Oblast definisanosti promenljivih
6. Privremene funkcije
7. Primeri programa



# K1

**29.10.2024. UTORAK**

**SII: 16:17h**

**IT:17-18h**

**E001**

## SOFTVERSKO I INFORMACIONO INŽENJERSTVO

PRVI KOLOKVIJUM-KLASIČNA NASTAVA 28.10-02.11.2024.

profesor	asistent	DAN	datum	satnica	sala
Milan Paroški	Nevena Radešić	UTORAK	29.10.2024.	16:00-17:00	E001

## INFORMACIONE TEHNOLOGIJE

PRVI KOLOKVIJUM-KLASIČNA NASTAVA 28.10-02.11.2024.

profesor	asistent	DAN	datum	satnica	sala
Milan Paroški	Nevena Radešić	UTORAK	29.10.2024.	17:00-18:00	E001a

№	Po	U	St	Če	Pe	So	№	Aktivnosti studenata
IX	23	24	25	26	27	28	29	I nedelja nastave
X	30	1	2	3	4	5	6	II nedelja nastave
	7	8	9	10	11	12	13	III nedelja nastave
	14	15	16	17	18	19	20	IV nedelja nastave
	21	22	23	24	25	26	27	V nedelja nastave / Oktobarski ispitni rok
	28	29	30	31	1	2	3	VI nedelja nastave - kolokvijumska nedelja
XI	4	5	6	7	8	9	10	VII nedelja nastave
	11	12	13	14	15	16	17	VIII nedelja nastave
	18	19	20	21	22	23	24	IX nedelja nastave
	25	26	27	28	29	30	1	X nedelja nastave/ Novembarski ispitni rok
XII	2	3	4	5	6	7	8	XI nedelja nastave
	9	10	11	12	13	14	15	XII nedelja nastave - kolokvijumska nedelja
	16	17	18	19	20	21	22	XIII nedelja nastave
	23	24	25	26	27	28	29	XIV nedelja nastave
	30	31	1	2	3	4	5	XV nedelja nastave

# BLOK 2 - raspored

Blok 2 ( 28.1				
Dan	Predmet	Grupa	Od	Do
Ponedeljak	Menadžment	IGMNG	10:00	13:00
			13:00	15:00
	Osnove programiranja	svi	15:00	18:00

Predavanja



Utorak	Organizacija i arhitektura računara	svi	10:00	13:00
	Engleski jezik 1	svi	13:00	16:00
	Osnove programiranja	OSP-G2	16:00	18:00
Sreda	Organizacija i arhitektura računara	AR-G1	08:00	10:00
	Menadžment	IGMNG	10:00	13:00
			13:00	15:00
	Osnove programiranja	OSP-G1	16:00	18:00

Vežbe SII



Četvrtak	Osnove programiranja	svi	08:00	10:00
	Matematika	svi	10:00	11:00
			11:00	13:00
			13:00	16:00

Vežbe IT



# Kalendar – predavanja sledeći čas: 04.11.

Activity **Calendar**


Today < > November 2024


	04 Monday	05 Tuesday
15	<div>Osnove programiranja - Predavanja - dr Milan Paroški A201 Milan Paroški</div>	
16		
17		
18		




<https://testns.singidunum.ac.rs/Student?>

Univerzitet Singidunum

 UNIS<sup>®</sup>  
MultiTutor

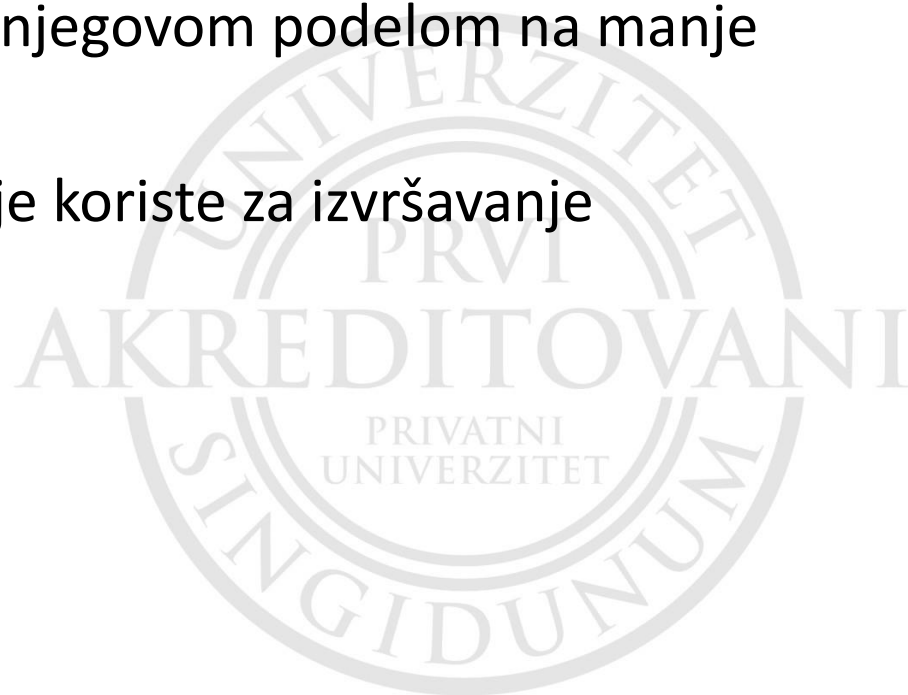
 MTutor Novi Sad

Пријава / Login	
Факултет / Faculty:	ПФБ / МТ 
Локација / Location:	Интернет (109.93.191.63)
Индекс / Student No:	<input type="text" value="Унесите свој број индекса / Enter your index number"/>
ЈМБГ / UMCN:	<input type="text"/>
<b>Note:</b> Foreign students should enter their passport number in the UMCN field	
<input type="button" value="Пријава / Login"/>	



# Pojam funkcije

- Korisničke **funkcije** su *imenovane grupe naredbi* koje izvršavaju određeni zadatak
- Funkcije omogućavaju:
  - višestruku upotrebu dela programskog koda i time skraćenje dužine programa
  - **bolju organizaciju koda programa** njegovom podelom na manje celine, koje je lakše razumeti
- Funkcije imaju listu parametara koje koriste za izvršavanje određenog zadatka, npr. ?  
abs(n)





Da li funkcije vraćaju kao rezultat određenu vrednost?

**Da:** `abs(-2)`      kako će glasiti komanda?

```
print(abs(-2))
```

rezultat je :2

**Ne:**

```
result = print("Hello, World!")
```

```
print(result)
```

Rezultat:

Hello, World!

None

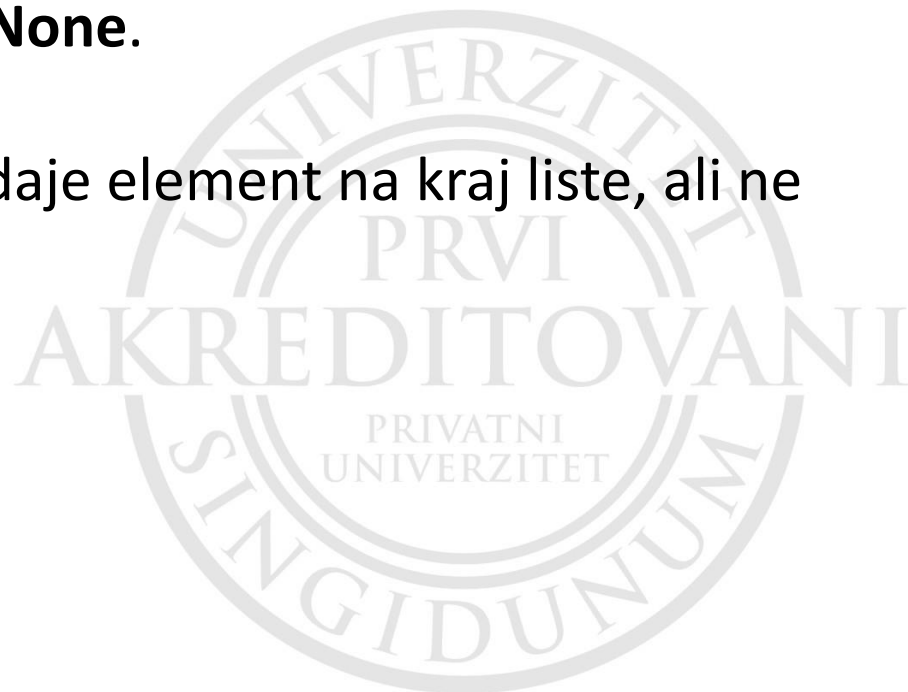
Pythonu, postoje ugrađene funkcije koje ne vraćaju eksplicitnu vrednost, već podrazumevano vraćaju **None**. Primer takvih funkcija su one koje obavljaju neku operaciju bez potrebe da vraćaju rezultat.

Pojašnjenje:

funkcija **print()** u Pythonu ne vraća vrednost, već samo ispisuje tekst na ekran.

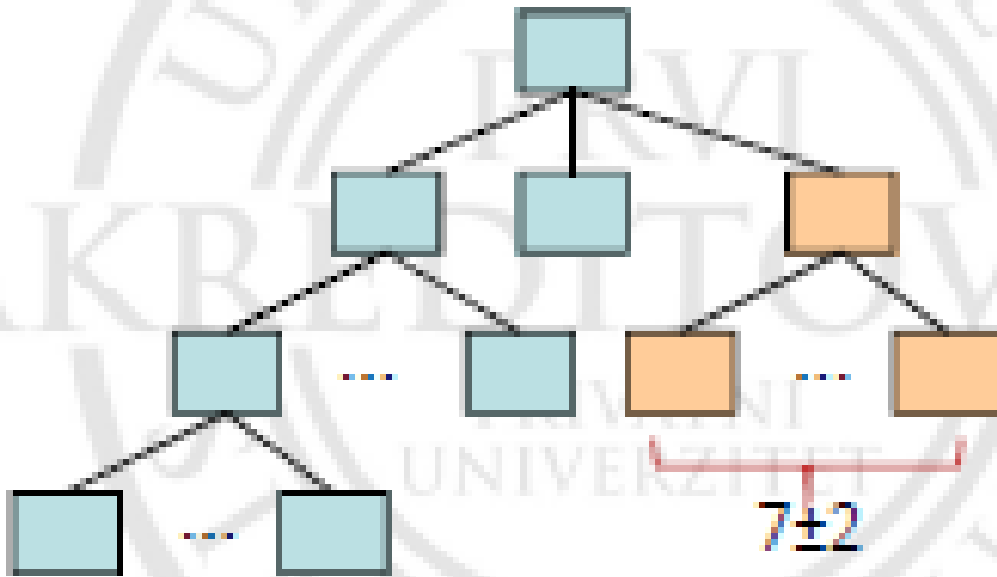
Svaka funkcija koja eksplicitno ne vraća vrednost pomoću **return** vraća podrazumevanu vrednost **None**.

**append()** – metoda liste koja dodaje element na kraj liste, ali ne vraća vrednost.



# Dekompozicija programskog koda

- Dekompozicija programskog koda u manje celine neophodna je prilikom razvoja kvalitetnog softvera, da se **obiman programski kod podeli na male, razumljive celine**
- Višestruka dekompozicija proizvodi hijerarhiju manjih programskih celina
  - poželjno je **na svakom nivou hijerarhije izvršiti podelu na ograničen broj manjih celina.**
  - celina koja se sastoji od velikog broja manjih delova sama postaje nerazumljiva
  - preporuka za *maksimalni* broj grananja u hijerarhiji dekompozicije programa može biti tzv. "Milerov magični broj  $7 \pm 2$ "



Tipična hijerarhijska dekompozicija.

Koliko je minimalno a koliko maksimalno elemenata u jednom hij.nivou?

Na svakom novou hijerarhije se pojavljuje najviše 5-9 elemenata

Dubina hijerarhije je nekoliko nivoa

Broj elemenata eksponencijalno raste s dubinom hijerarhije

## 2. Definicija korisničke funkcije

- Sintaksa definicije funkcije
- Primer



# Sintaksa definicije funkcije

- Nova funkcija definiše se naredbom:

```
def <nazivFunkcije> (<lista parametara>):  
<blok naredbi>
```



Primer:

```
def maksimum(n1, n2):  
    if n1 > n2:  
        rezultat = n1  
    else:  
        rezultat = n2  
    return rezultat
```

Upotreba (poziv) funkcije:

```
veci = maksimum(x, y)
```

*naziv funkcije*

*lista formalnih parametara, koji mogu imati podrazumevajuće (default) vrednosti*

*zaglavlje funkcije*

*telo funkcije (blok naredbi)*

*vrednost koju funkcija vraća kao rezultat; funkcija može istovremeno da vrati i više vrednosti: **return** n1, n2, ...*

*lista aktuelnih parametara (agumenata funkcije)*

```
def maksimum(n1,n2):  
    if n1>n2:  
        rezultat=n1  
    else:  
        rezultat=n2  
    return rezultat
```

Propustiti ovo sa RUN. Šta se dobije?

```
veci=maksimum(15,16)  
print(veci)  
16
```

Probati?

Dodati na kraj ovo:





# Ubacivanje DEF u fajl

## **I kolokvijum 05 korišćenje funkcije DEF.py**

```
def maksimum(n1,n2):  
    if n1>n2:  
        rezultat=n1  
    else:  
        rezultat=n2  
    return rezultat  
veci=maksimum(15,16)  
print(veci)
```



## I kolokvijum 05 Upotreba funkcije IF umesto ponavljanja.py

```
zbir = 0
```

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

```
    zbir = zbir + i
```

```
    print("i=",i," Zbir brojeva od 1 do 10 je", zbir)
```

```
zbir = 0
```

```
for i in range(21,31):
```

```
    zbir = zbir + i
```

```
    print("1=",i," Zbir brojeva od 21 do 30 je", zbir)
```

```
zbir = 0
```

```
for i in range(180,191):
```

```
    zbir = zbir + i
```

```
    print("i=",i," Zbir brojeva 180 do 190 je", zbir)
```

ŠTA ĆE ISPISATI OVAJ PROGRAM?



## Rezultat izvršavanja:

i= 1 Zbir brojeva od 1 do 10 je 1  
i= 2 Zbir brojeva od 1 do 10 je 3  
i= 3 Zbir brojeva od 1 do 10 je 6  
i= 4 Zbir brojeva od 1 do 10 je 10  
i= 5 Zbir brojeva od 1 do 10 je 15  
i= 6 Zbir brojeva od 1 do 10 je 21  
i= 7 Zbir brojeva od 1 do 10 je 28  
i= 8 Zbir brojeva od 1 do 10 je 36  
i= 9 Zbir brojeva od 1 do 10 je 45  
i= 10 Zbir brojeva od 1 do 10 je 55

1= 21 Zbir brojeva od 21 do 30 je 21  
1= 22 Zbir brojeva od 21 do 30 je 43  
1= 23 Zbir brojeva od 21 do 30 je 66  
1= 24 Zbir brojeva od 21 do 30 je 90  
1= 25 Zbir brojeva od 21 do 30 je 115  
1= 26 Zbir brojeva od 21 do 30 je 141  
1= 27 Zbir brojeva od 21 do 30 je 168  
1= 28 Zbir brojeva od 21 do 30 je 196  
1= 29 Zbir brojeva od 21 do 30 je 225  
1= 30 Zbir brojeva od 21 do 30 je 255

i= 180 Zbir brojeva 180 do 190 je 180  
i= 181 Zbir brojeva 180 do 190 je 361  
i= 182 Zbir brojeva 180 do 190 je 543  
i= 183 Zbir brojeva 180 do 190 je 726  
i= 184 Zbir brojeva 180 do 190 je 910  
i= 185 Zbir brojeva 180 do 190 je 1095  
i= 186 Zbir brojeva 180 do 190 je 1281  
i= 187 Zbir brojeva 180 do 190 je 1468  
i= 188 Zbir brojeva 180 do 190 je 1656  
i= 189 Zbir brojeva 180 do 190 je 1845  
i= 190 Zbir brojeva 180 do 190 je 2035

Primer uprošćavanja:

```
def zbir(n1,n2):
```

```
    n=0
```

```
    for i in range(n1,n2+1):
```

```
        n=n+i
```

```
    return n
```

**Zašto piše:  $n2+1$ ?**

```
print("zbir brojeva od 1 do 10 je",zbir(1,10))
```

A može da se doda i:

```
print("zbir brojeva od 21 do 30 je",zbir(21,30))
```

```
print("zbir brojeva od 81 do 90 je",zbir(81,90))
```

## Rezultat:

zbir brojeva od 1 do 10 je 55

zbir brojeva od 21 do 30 je 255

zbir brojeva od 81 do 90 je 855



# 3. Upotreba funkcije

1. Pozivanje funkcije

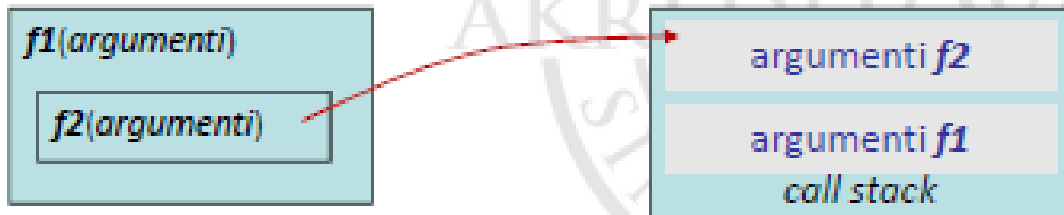
2. Primer



# Pozivanje funkcije

- Funkcija se poziva po svom *nazivu*, uz navođenje liste aktuelnih argumenata
  - Funkcije se mogu pozivati iz samih funkcija. Šta je to? Ungnježdeni pozivi. Dozvoljeni su i rekurzivni pozivi. Prilikom svakog aktiviranja funkcije, različiti aktuelni parametri funkcije iz svakog poziva čuvaju se u posebnoj sistemskoj strukturi, tzv. **steku izvršavanja** ili steku poziva (*call stack*) i koriste se obrnutim redom

# Definicija steka



Stek je takva struktura podataka iz koje se prvi uzima element koji je poslednji ubačen.

## 1. LIFO?

– Funkcija se izvršava za aktuelne vrednosti parametara koje se nalaze na vrhu steka (*top*), na slici prvo *f2* pa *f1*

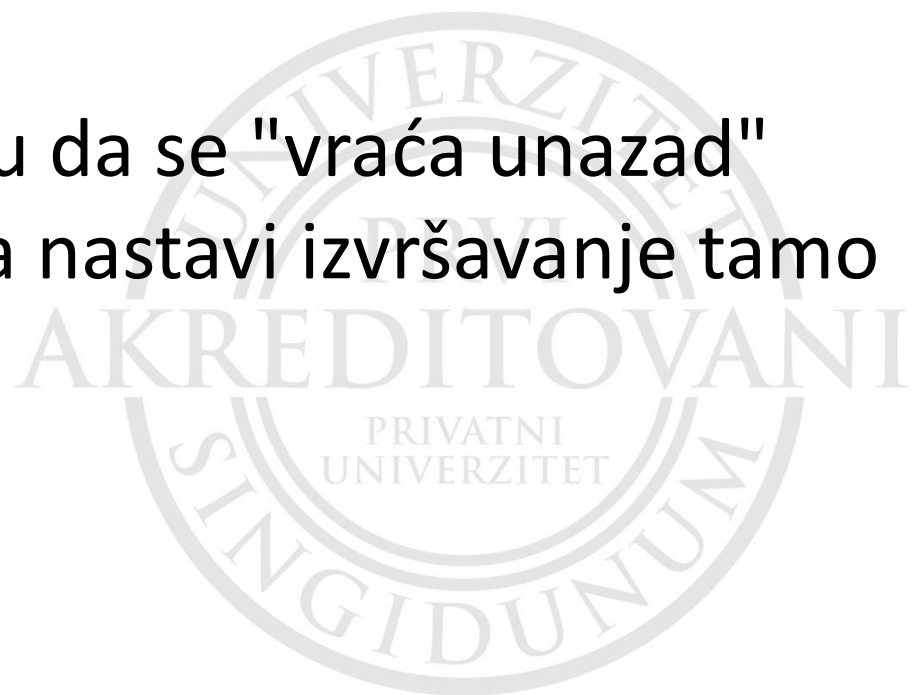
Stek, vizuelno, mozemo zamisliti ovako:





Svaki put kada funkcija u programu bude pozvana, sistemski stek će zabeležiti trenutno stanje programa, uključujući vrednosti lokalnih promenljivih i poziciju u kodu.

Ovo omogućava programu da se "vraća unazad" kada se funkcija završi i da nastavi izvršavanje tamo gde je prekinuto.



Pored LIFO koji smo malopre naveli, postoje i FIFO i FILO metode:

## 2. FIFO ?

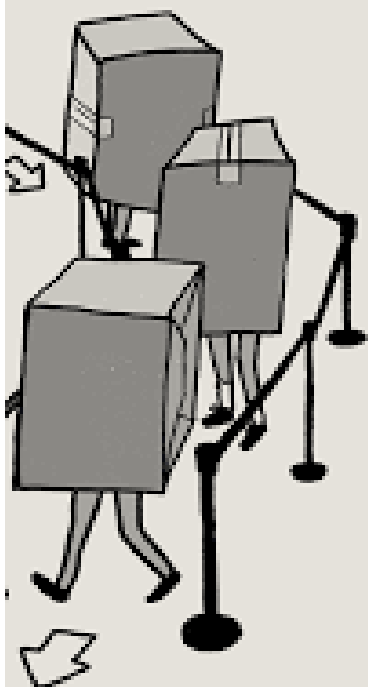
First in First Out

FIFO metoda se najčešće koristi u redovima, odnosno *queues*

FILO:

Prvi tanjir koji si stavio je **poslednji** koji ćeš uzeti (First In, Last Out - FILO).

Poslednji tanjir koji si stavio je **prvi** koji ćeš uzeti (Last In, First Out - LIFO).



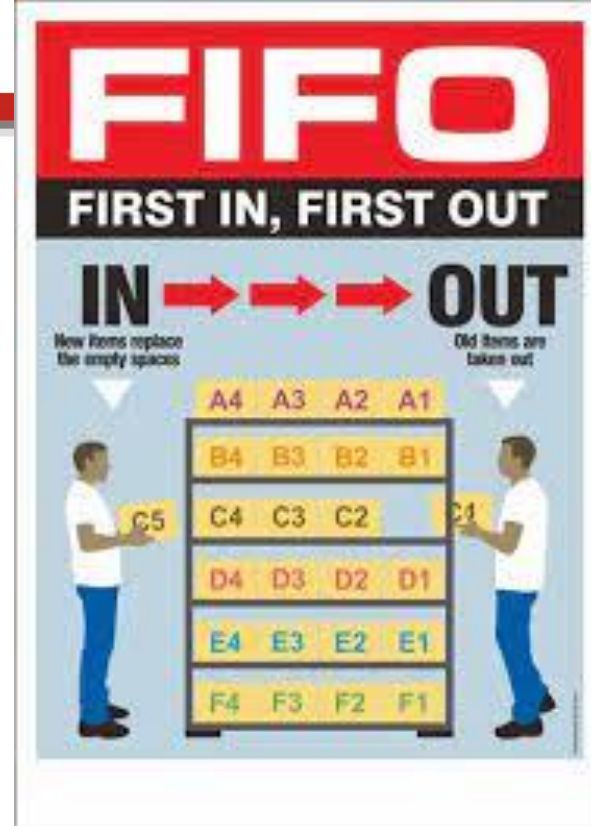
## First In, First Out (FIFO)

*['fərst 'in 'fərst 'aʊt]*

An asset and inventory management approach in which assets produced or acquired first are sold, used, or disposed of first.



## 2. FIFO

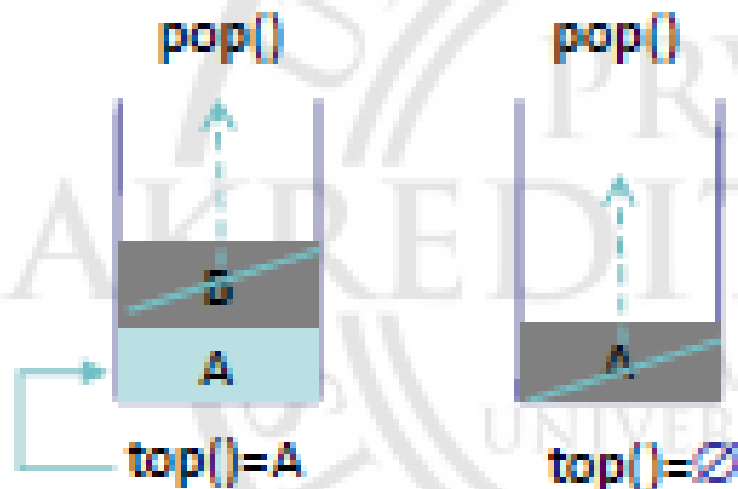
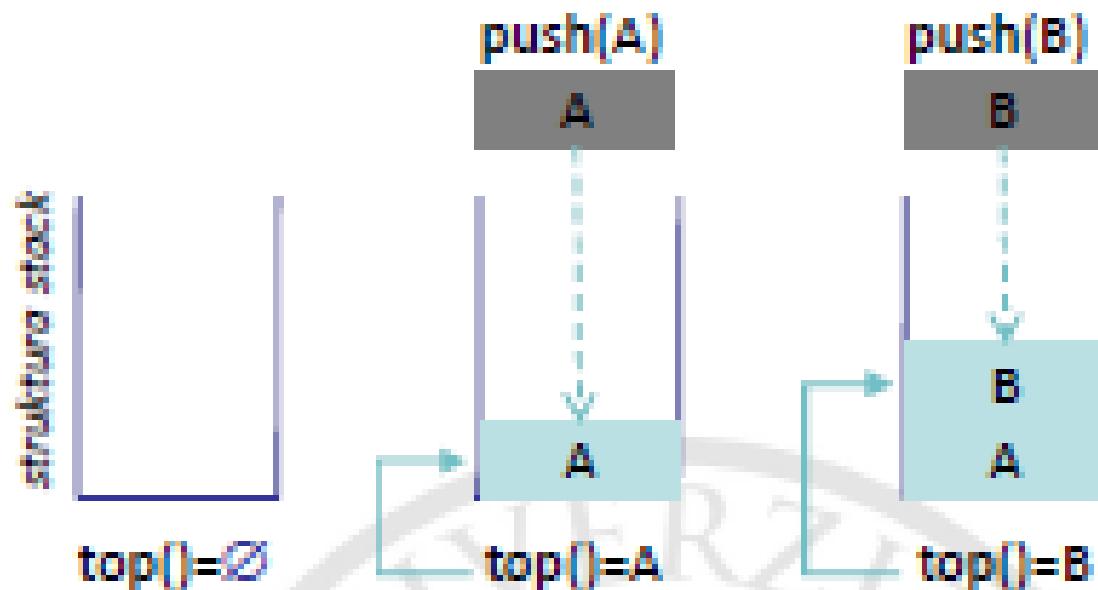


# Struktura stek (*stack*)

- Struktura podataka koja omogućava pamćenje redosleda izvršavanja operacija
  - dodavanje i brisanje vrši se s jedne strane, *vrha* steka (*top*)
  - operacije nad listom su `push(x)` i `pop()`, a pristup je moguć samo elementu na vrhu steak funkcijom `top()`

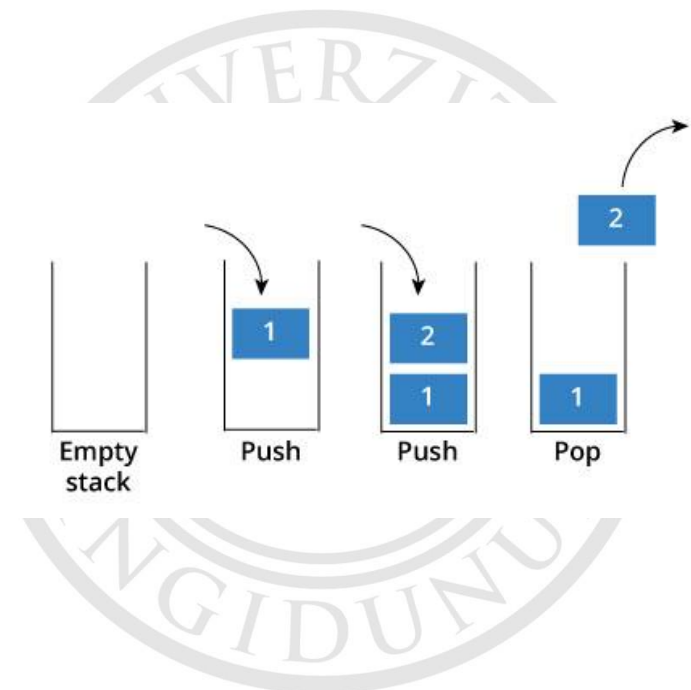
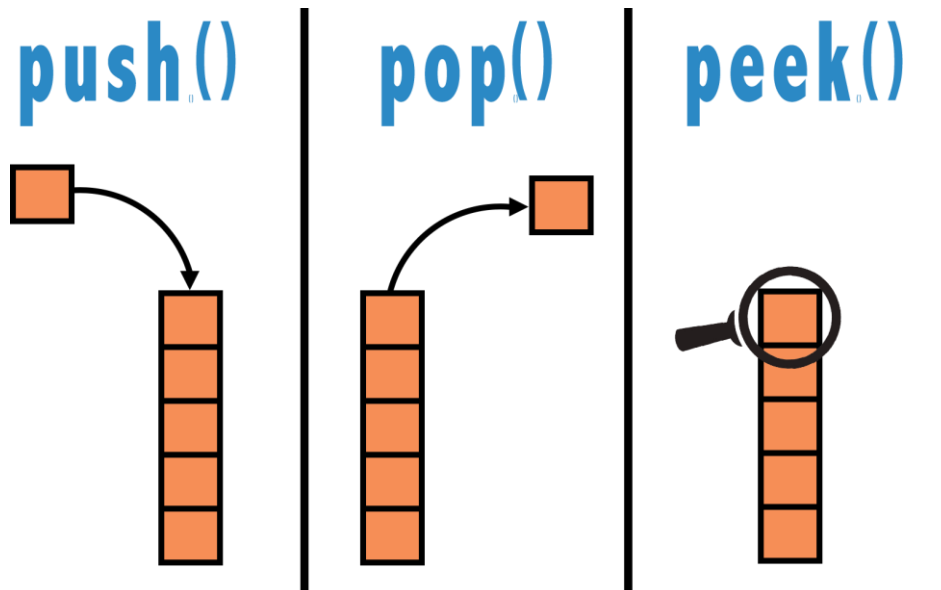
*Analogija s operacijama slaganja tanjira*

# Osnovne operacije



# Operacije steka

- **Push** (guranje): Dodavanje novog čvora na vrh steka. Ovo se radi kada želite skladištiti novu vrednost na steku.
- **Pop** (skidanje): Uklanjanje čvora sa vrha steka. Ovo se radi kada želite dohvatiti i ukloniti poslednju vrednost sa steka.
- **Peek** (pogled): Ovo vam omogućava da vidite vrednost koja se nalazi na vrhu steka bez da je uklonite.



**Praćenje redosleda akcija:** Stek je koristan za praćenje redosleda akcija i omogućava povratak na prethodna stanja, što je od suštinskog značaja u mnogim aplikacijama, kao što je "Undo" funkcionalnost.



**Ograničena veličina:** Veličina steka je obično fiksirana pri inicijalizaciji, što može dovesti do problema ako stek postane pretrpan ili ako je potrebno dinamičko proširenje.

Kako se to zove?

Stack overflow

**Nedovoljna fleksibilnost:** Stak se ne može koristiti za situacije gde je potreban pristup elementima na proizvoljnim pozicijama (kao što je slučaj sa listom) ili za situacije gde je potreban brz pristup sredini strukture.



# Kreiranje steka

```
stack = []
```

# Dodavanje elemenata na stek (push operacija)

```
stack.append(10)
```

```
stack.append(20)
```

```
stack.append(30)
```

```
print("Stek nakon push operacija:", stack)
```

Šta će ispisati?

Stek nakon push operacija: [10, 20, 30]



Šta će ovo izvršiti?

# Uklanjanje elemenata sa steka (pop operacija)

```
top_element = stack.pop()
```

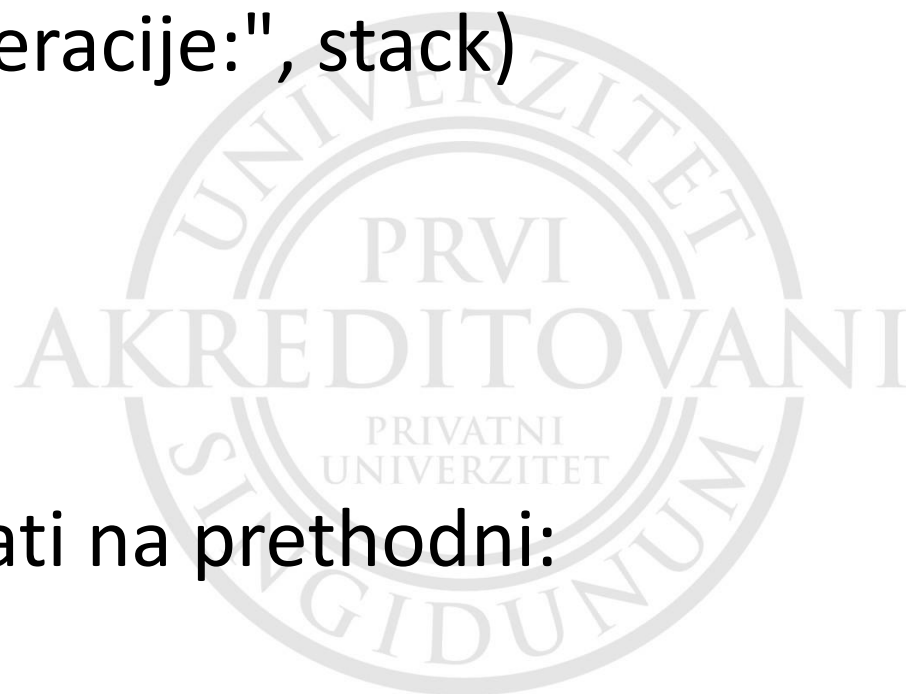
```
print("Uklonjeni element:", top_element)
```

```
print("Stek nakon pop operacije:", stack)
```

NIŠTA. Zašto

Nije definisan stack

Treballi smo ovaj kod dodati na prethodni:



# Kreiranje steka

```
stack = []
```

# Dodavanje elemenata na stek (push operacija)

```
stack.append(10)
```

```
stack.append(20)
```

```
stack.append(30)
```

```
print("Stek nakon push operacija:", stack) # Output: [10, 20, 30]
```

# Uklanjanje elemenata sa steka (pop operacija)

```
top_element = stack.pop()
```

```
print("Uklonjeni element:", top_element)
```

```
print("Stek nakon pop operacije:", stack)
```

Rezultati:

Stek nakon push operacija: [10, 20, 30]

Uklonjeni element: 30

Stek nakon pop operacije: [10, 20]




- **peek()**: Vraća stavku sa vrha steka bez uklanjanja
- (u Pythonu možete to uraditi koristeći `stack[-1]`).

Šta će se desiti kad na program sa prethodnog slajda dodamo komandu:  
`print(len(stack))`

Rezultat?

2

Šta će se desiti kad na program sa prethodnog slajda dodamo komande:  
`print(stack[-1])`  
`print(stack[-2])`  
`print(stack[-3])`



20

10

Traceback (most recent call last):

File

"C:/Users/Milan/AppData/Local/Programs/Python/Python311/test.py", line 14, in <module>

`print(stack[-3])`

IndexError: list index out of range

**isEmpty():** Proverava da li je stek prazan  
(u Pythonu možete koristiti `len(stack) == 0`).

Dodati na kraj prethodnog programa:

```
if len(stack) == 0:
```

```
    print("Prazan")
```

```
else:
```

```
    print("Nije prazan")
```

Rezultat?

Nije prazan



```
stack = []  
  
# Dodavanje elemenata na stek (push operacija)  
stack.append(10)  
stack.append(20)  
stack.append(30)  
print("Stek nakon push operacija:", stack) # Output: [10, 20, 30]  
  
# Uklanjanje elemenata sa steka (pop operacija)  
top_element = stack.pop()  
print("Uklonjeni element:", top_element)  
print("Stek nakon pop operacije:", stack)  
print(len(stack))  
print(stack[-1])  
print(stack[-2])  
if len(stack) == 0:  
    print("Prazan")  
else:  
    print("Nije prazan")
```



## 4. Argumenti funkcije

- Aktuelni argumenti mogu se funkciji prenositi **po poziciji** (redosledu) ili **po njihovom nazivu**, kao *naziv=vrednost*

- Npr. funkcija definisana kao

def f(p1,p2,p3) naš primer: **zbir(n1,n2)**

**1. može se pozvati s vrednostima argumenata po poziciji** kao:

f(1, 12, 400)

def zbir(n1,n2):

    n=0

    for i in range(n1,n2+1):

        n=n+i

    return n

print(zbir(1,4))

Rezultat je:10



2. ili **po *nazivima*** argumenata, proizvoljnim redom:

$f(p_3=400, p_1=1, p_2=12)$

Za naš slučaj?

```
def zbir(n1,n2):
```

```
    n=0
```

```
    for i in range(n1,n2+1):
```

```
        n=n+i
```

```
    return n
```

```
print(zbir(n2=4,n1=1))
```

Rezultat je:10



**3. Ili *mešovito*, ali prvo vrednosti argumenata po poziciji:**

**$f(1, p_3=12, p_2=400)$**

**def zbir(n1,n2):**

**n=0**

**for i in range(n1,n2+1):**

**n=n+i**

**return n**

**print(zbir(1,n2=4))**

Rezultat je:10

4. ali *ne* po poziciji *nakon* navođenja argumenta po nazivu:

f(1, p2=12, 400)

Za naš slučaj?

```
def zbir(n1,n2):
```

```
    n=0
```

```
    for i in range(n1,n2+1):
```

```
        n=n+i
```

```
    return n
```

```
print(zbir(n1=1,4))
```

[SyntaxError:](#)



### 3. $f(1, p_3=12, p_2=400)$

```
def zbir(n1,n2,n3):
```

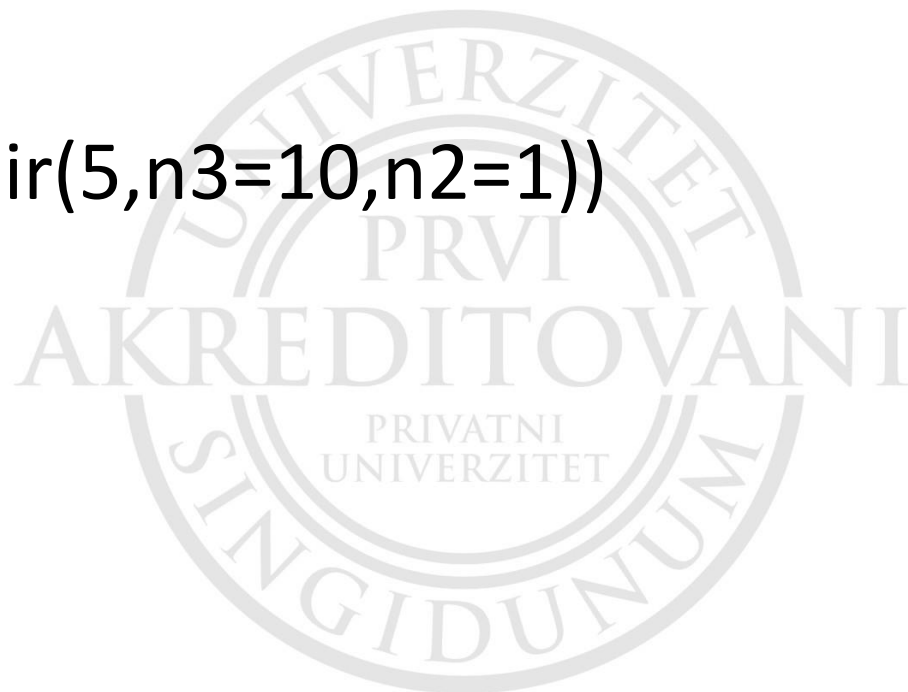
```
    n=n1+n2+n3
```

```
    return n
```

```
print("zbir tri broja je",zbir(5,n3=10,n2=1))
```

Da li je ovo moguće?

DA: zbir tri broja je 16




```
def zbir(n1,n2,n3):
```

```
    n=n1+n2+n3
```

```
    return n
```

```
print("zbir tri broja je",zbir(5,n2=10,1))
```

Da li je ovo moguće?

 SyntaxError



positional argument follows keyword argument

OK

ali *ne* po poziciji *nakon* navođenja argumenta po nazivu:  
f(1, p2=12, 400) ovo je slučaj 4

# Prenos argumenata po vrednosti

- Prenos argumenata funkciji u jeziku Python vrši se po vrednosti

(*pass-by-value*)

- Navođenje naziva promenljive kao aktuelnog parametra samo kopira *vrednost* promenljive, koja se smešta u stek poziva i zatim koristi u telu funkcije

- Npr. funkcija ***povecaj*** neće promeniti vrednost promenljive *brojac* koja se prenosi kao argument:

```
def povecaj(x):
```

```
    x = x + 1
```

```
    print("x=",x)
```

```
    print("brojac=",brojac)
```

```
brojac = 1
```

```
povecaj(brojac)
```

```
print("na kraju brojac je",brojac)
```

x= 2

brojac= 1

na kraju brojac je 1

## 5. Oblast definisanosti promenljivih

- Promenljive/objekti u jeziku Python mogu biti:
  - ugrađeni (*built-in*)
  - lokalni i
  - globalni objekti





# Lokalni i globalni objekti u funkcijama

- Lokalni objekti/promenljive nastaju i dostupne su samo u funkciji u kojoj se koriste.
  - nakon završetka rada funkcije, lokalne promenljive više *nisu dostupne*.  
Prilikom svakog narednog izvršavanja funkcije *ponovo se kreiraju*
- Globalni objekti/promenljive definišu se u glavnom programu, globalno su dostupni i mogu se koristiti u svakoj funkciji
- Kakvu dostupnost imaju Ugrađeni objekti/promenljive ?  
takođe imaju globalnu dostupnost
- Zavisnost programskog koda od *globalnih* objekata smatra se lošom programerskom praksom, jer ograničava višestruku upotrebljivost koda i može biti uzrok programskih greški

```
def f1():
```

```
    a=2
```

```
    def f2():
```

```
        a=3
```

```
    print(a)
```

f2()

Šta je rezultat?

Traceback (most recent call last):

File "C:/Users/Milan/Documents/test.py", line 7, in <module>

Zašto?



```
def f1():
```

```
    a=2
```

```
    def f2():
```

```
        a=3
```

```
    print(a)
```

```
f1()
```

Šta je rezultat:

2



```
def f1():
```

```
    a=2
```

2

```
    def f2():
```

1

```
        a=3
```

```
    print(a)
```

```
a=1
```

```
f1()
```

```
print(a)
```



```
def f1():
```

```
    a=2
```

```
    def f2():
```

```
        a=3
```

```
        print(a)
```

```
3
```

```
2
```

```
1
```

```
>>>
```

```
f2()
```

```
print(a)
```

```
a=1
```

```
f1()
```

```
print(a)
```

