Prezentacija održana na Filozofskom fakultetu u Beogradu 28.12.2019. godine,
u okviru kursa Odabrane teme iz psiholingvistike i kvantitativne lingvistike

Python za neprogramere

veoma **veoma** kratak kurs

dr Đorđe Đurđević

Uvod – ciljevi kursa

- Ciljevi
  - Upotreba jezika Python
  - Automatizacija obrade podataka
  - Automatizacija pripreme podataka
- Zašto Python?
  - Mnogo biblioteka, efikasno i produktivno programiranje
  - Gde se koristi Python?
    - Google, Yahoo, CERN, NASA, naučne institucije

Uvod – Verzije jezika i alati

- Dve bitne grane: 2.x i 3.x
  - Postoje značajne razlike između 2.7 i 3.0 u samom jeziku!!!
    - Način upotrebe nekih operatora
    - …
  - Nisu sve biblioteke “prebačene” na verziju 3.0
  - Mi koristimo verziju 2.7
- Alati
  - PyScripter
  - IntelliJ Idea
  - …

Definicije

- Podatak
  - entitet kome se programskim putem pristupa putem imena (identifikatora)
  - ima lokaciju u memoriji računara (ne mora uvek)
- Tip
  - način tumačenja memorijskog sadržaja koji zauzima entitet
  - precizira skup **vrednosti** i **operacija** nad podacima
  - osnovni tipovi: `int`, `bool`, `float`, `str`
- Promenljiva
  - podatak kome je moguće menjati vrednost

Komentari u izvornom kodu

- U jednom redu, iza simbola **#**
  - Ostatak reda iza tog simbola je komentar
- Unutar tzv. *heredocs*
  - tekst koji se nalazi između 2 para od po 3 uzastopna znaka navoda: """ ... """

```
# primer linijskog komentara.

""" Primer višelinijskog komentara. Može da sadrži prelaz
u novi red. Služi da se podrobnije opiše šta i kako
deo koda radi, i zbog čega.

Komentarišite kod, i zbog sebe i zbog drugih!!!
"""
```

Celi brojevi – tip **int**

- Vrednosti: celi brojevi
  … -4, -3, -2, -1, 0, 1, 2, 3, 4, …
- Operacije: *****, **+**, ******, **/**, **–**
- Rezultat operacije deljenja 2 cela broja: ceo broj
  - 10/3 = 3 ?
- Literali
  - Ako počinju sa 0, onda su oktalne vrednosti: 010
  - Ako počinju sa 0x, onda su heksadecimalne: 0x1ac

Realni brojevi – tip **float**

- Vrednosti: podskup skupa realnih brojeva
  1.4, 2.333, -.04, -35e-3 ($-35 \cdot 10^{-3}$), ...
- Operacije: *****, **+**, ******, **/**, **–**
- Rezultat operacije deljenja 2 realna broja:
  realan broj
  10.0/3.0 = 3.333 (potrebno da najmanje 1 operand bude realan broj)
- Ograničena tačnost, zaokruživanje

- `round(3.1415 * 4, 3)`

  `12.566`

Pregled operatora nad numeričkim tipovima

Operator	Značenje	Primenljiv na	Primer
-	Negacija, promena znaka	1 operand	`-2` `-3.5`
-	Oduzimanje	2 operanda	`1 - 3 → -2` `2 - 5.4 → -3.6`

/	Deljenje	2 operanda. Celobrojno ili realno deljenje, zavisi od operanada	`1/3 → 0`
			`1/3.0 → 0.33333331`
			`5/2 → 2`
			`5.0/2 → 2.5`

*	Množenje	2 operanda	`2*2`

%	Ostatak pri deljenju	2 operanda	`5%2 → 1`
			`10 % 3 → 1`
			`11 % 3 → 2`

**	Stepenovanje	2 operanda	`2**8 → 256` `2.0**8 → 256.0`
//	Deljenje sa zaokruživanjem "na niže"	2 operanda	`9//2 → 4` `9.5//2 → 4.0`

Eksplicitna konverzija

- Potrebna da bi programer ***eksplicitno*** naznačio sa kojim tipom podataka želi da radi
- Sintaksa: `tip(izraz)`
  - Konvertuje tip izraza u navedeni tip
- Primer: dati su celobrojne promenljive **`a`**, **`b`**; želimo realno deljenje **`a/b`** (ne celobrojno)?
  - `float(a)` daje realnu vrednost koja odgovara "realnom" **`a`**
  - `float(a)/b`
- A ako imamo realan broj, a želimo ceo?
  - `int(a)` vraća celi deo

Znakovni nizovi (niske) – tip **string**

- **`"Zdravo"`** ili **`'Zdravo'`**
- Moguće nadovezivanje, operator **+**
  - **`"Zdravo" + 'svete'`**
- Indeksiranje: moguće dohvatiti karakter na proizvoljnoj poziciji

```
>>> a="abrakadabra"
>>> a[1]
'b'
>>> a[0]
'a'
```

- ... ali ne **može tako da se promeni** (string-ovi su nepromenljivi)

Logički podaci – tip **bool**

- Moguće 2 vrednosti
  - **`True`** (logička istina)
  - **`False`** (logička neistina)
- Logički operatori:
  - **`and, or, not`**
- Relacioni operatori:
  - **`  >  <=  =  >=  >  !=`**
- Rezultat logičkih i relacionih operatora – **`bool`**

```
>>> a.startswith("ab")
True
>>> 5 > 2
True
```

```
>>> 5 > 2 < 2 > 1
False
>>> True < False
False
>>> True > False
True
```

Promenljive (1)

- Promenljiva
  - podatak čija vrednost može da se promeni
  - zauzima prostor u memoriji
  - sadržaj memorije se tumači prema tipu primenljive
  - pristupa se putem imena (identifikatora)
- Ime promenljive (identifikator)
  - Može biti proizvoljne dužine, razlikuju se mala i VELIKA slova
  - Ne sme početi cifrom, preporučuje se da počne malim slovom
  - Znak "_" se koristi za odvajanje reči u nazivu sačinjenom od više reči, npr. `cena_kafe`. Može da počne ovim znakom, ali se ne **preporučuje**.
  - Ne sme da bude rezervisana reč
  - Lokalna ili globalna
- Može se **definisati**. Prvi put kada se upotrebi ime promenljive (u dodeli vrednosti), tada se ta promenljiva definiše, i ostatak programa je može koristiti
- Može se **promeniti** vrednost promenljivoj

Promenljive (2)

```
cena_kafe = 99.99
broj_kafa = 3
cena_porudzbine = cena_kafe * broj_kafa
#... neki kod, nebitno …
cena_porudzbine = 1.2*cena_porudzbine
```

Definicija novog realnog podatka

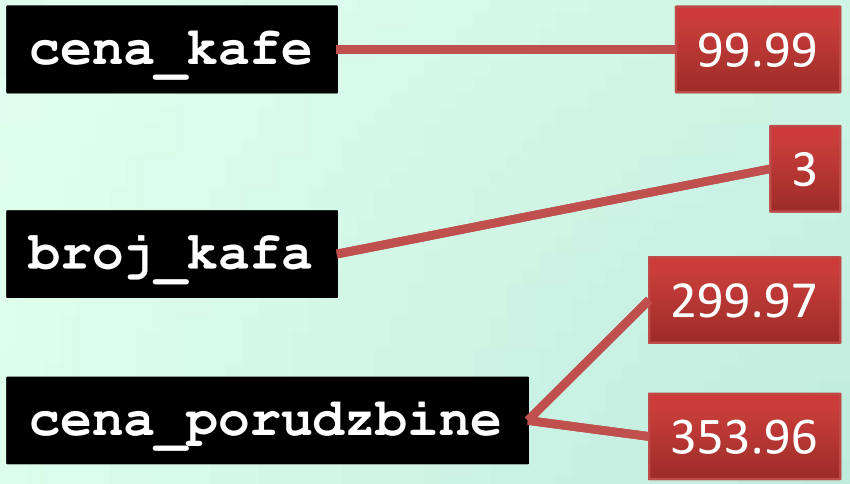
Definicija novog celobrojnog podatka

Dodela vrednosti

- Operator = pri definiciji i dodeli vrednosti

- Jednom upotrebljeno ime može da promeni podatak na koji se odnosi

```
a = 5          # type(a) → int
b = 5.5        # type(b) → float
a = b          # type(a) → float
```



Osnovni izlaz – funkcija **print**

- Funkcija **`print`** "štampa" vrednosti koje joj se zadaju
  - Sve vrednosti razdvojene zarezma:

```
print 2.0, x, 7+2*3,"Nova Godina!"
```

- Promenljive u tekstu ispisa

```
print "Racun za {} kafa = {}".format(broj_soljica, cena_kafe)
```

- Vrednosti predate funkciji **`format`**
  se uzimaju redom
  i pojavljuju *umesto* vitičastih zagrada

Osnovni ulaz – funkcija **input**

- Funkcija **`raw_input`**
  - "sirovi" unos
  - pročita tekst i taj rezultat dopremi do programa - string
- Funkcija **`input`**
  - pročita tekst, a zatim pokuša da formira podatak nekog od tipova
    - **`float`**
    - **`int`**
    - **`bool`**
- Ako je nemoguće, desi se greška!
- Najbolje: **`float(raw_input())`** ili **`int(raw_input())`**
  - Time se sigurno pročitano konvertuje u željeni tip

Kontrolne strukture

- Čemu služe kontrolne strukture?
  - Kontrolišu tok programa
  - Obezbeđuju da se određeni delovi programa budu izvršeni ako su ispunjeni određeni uslovi
  - Selekcija (izbor jednog od većeg broja ishoda)
  - Ciklusi
    - ponavljanje nekih instrukcija
      - određen broj puta
      - dok je određen uslov ispunjen

Uslovna grananja (if-else)

- Izvršavanje dela koda pod uslovom da je neki uslov zadovoljen:

```
if uslov:
        telo_then_grane # blok koda
else:
        telo_else_grane # blok koda
```

- Objašnjenje:
  - *uslov* je izraz **bool** tipa
  - *telo_then_grane*, *telo_else_grane* su sekvence naredbi (tzv. *blokovi*)
    - Proizvoljan broj, najmanje 1 naredba
    - Ako je samo 1 naredba, ona se može navesti u produžetku (iza "**:**")
    - Ako ima više naredbi, svaka mora biti u novom redu, uvučena za (dodatne) 4 razmaknice!!!
  - Dvotačke su obavezne!
  - **else** se može izostaviti – nije obavezno

if – elif – else

- **`If-else`** omogućuje međusobno isključivanje blokova koda
  - then blok u slučaju da je uslov ispunjen
  - else blok u slučaju da nije
- Ako uslov **`if`**-a nije ispunjen, kako postaviti novi uslov?
- Ostvaruje se pomoću ključne reči **`elif`**:
  - omogućuje da se proveri istinitost više izraza
    i da se izvrši blok koda za **prvi izraz** (u nizu) koji je istinit
  - **Samo jedan blok se izvrši**
    čak iako se desi da bude više istinitih

Ciklusi

- Neke obrade je potrebno učiniti veći broj puta
  - Za svaki karatker (slovo) u reči (rečenici)
  - Svaki element niza
  - Određeni broj puta: na primer (5, n)
  - Do zadovoljenja nekog uslova
- Ciklusi (petlje)
  - for
  - while

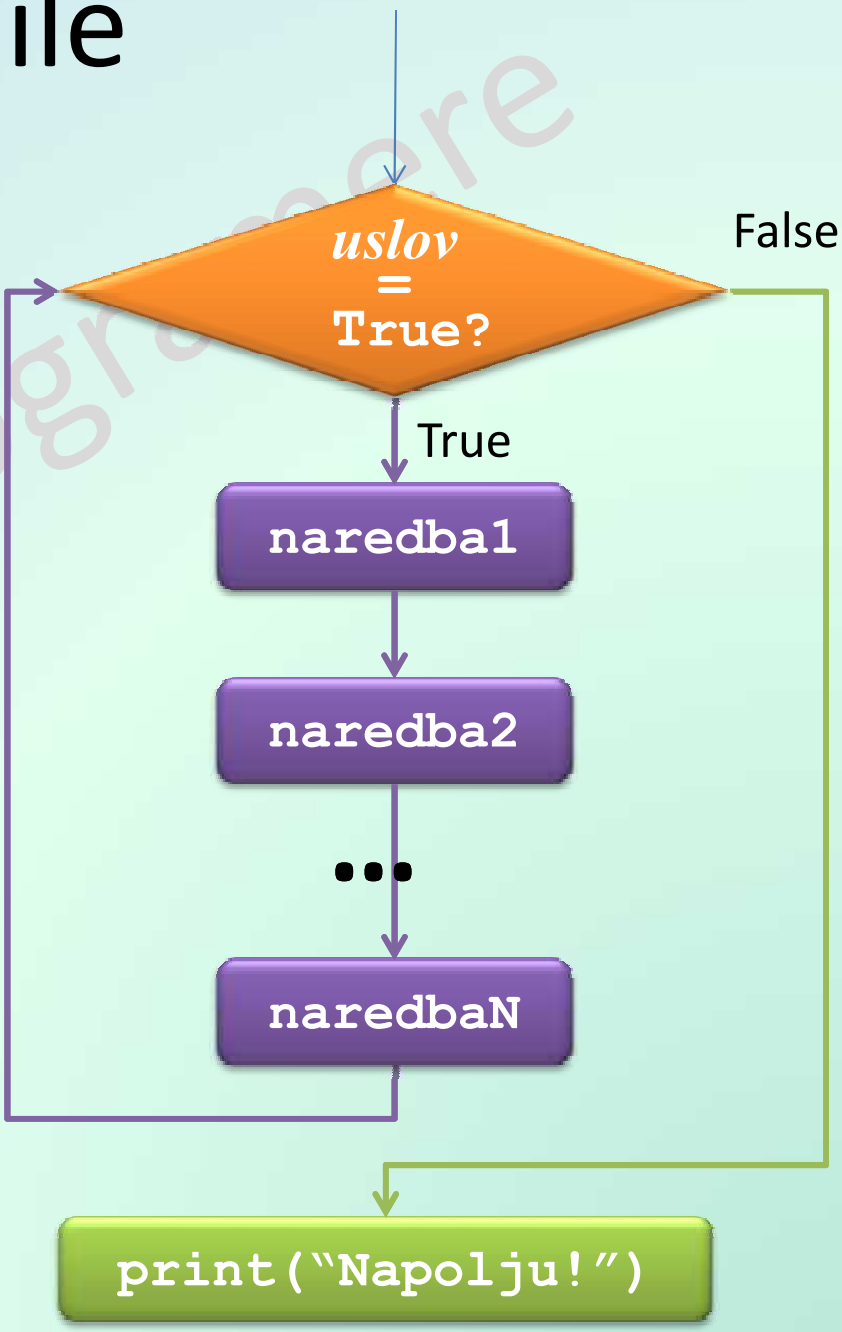
Ciklus while

- Izvršavaj sve dok je (logički) uslov zadovoljen:

```
while uslov:
    naredba_1
    naredba_2
    #...
    naredba_N
# van ciklusa
```

Ciklus while

```
while uslov:
      naredba_1
      naredba_2
      #...
      naredba_N
# van petlje
print("Napolju!")
```



Ciklus for

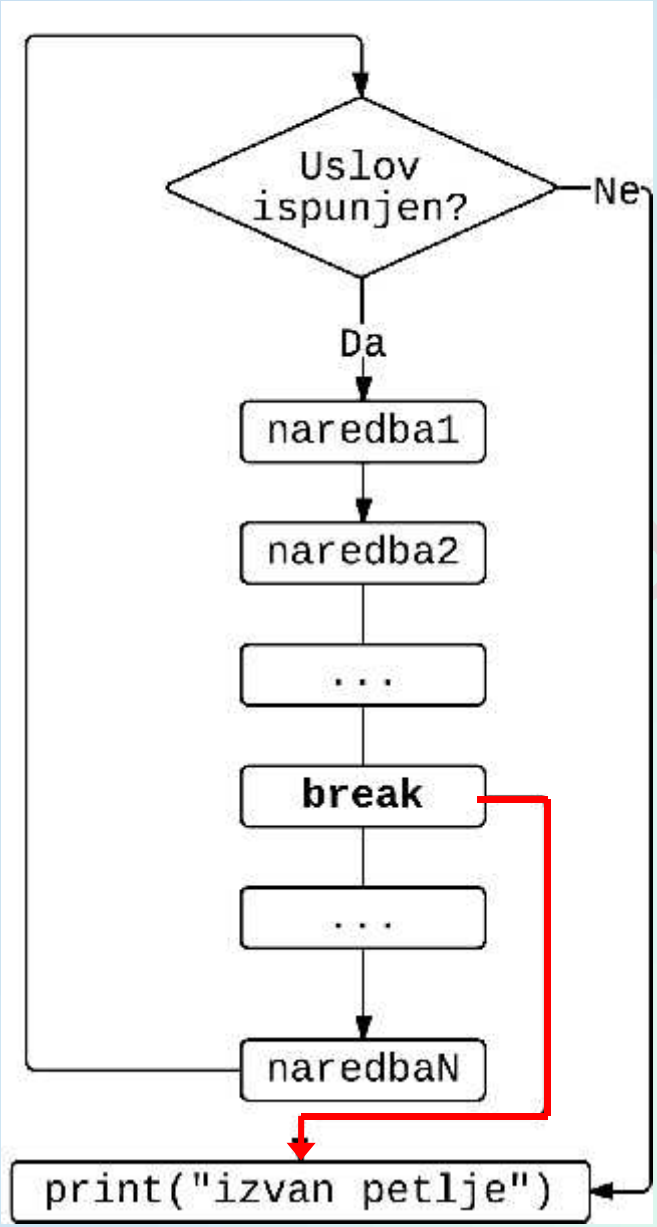
- Služi da se izvrši
  - unapred određeni broj puta (u određenom opsegu)
  - za svaki element neke kolekcije
- Ne zavisi od logičkog uslova
- Sintaksno: mora joj se zadati lista (ili tuple, ili string)
- Promenljiva **`for`** petlje
  - Redom uzima vrednosti niza, jednu po jednu
  - Za savki element niza – po jedna iteracija

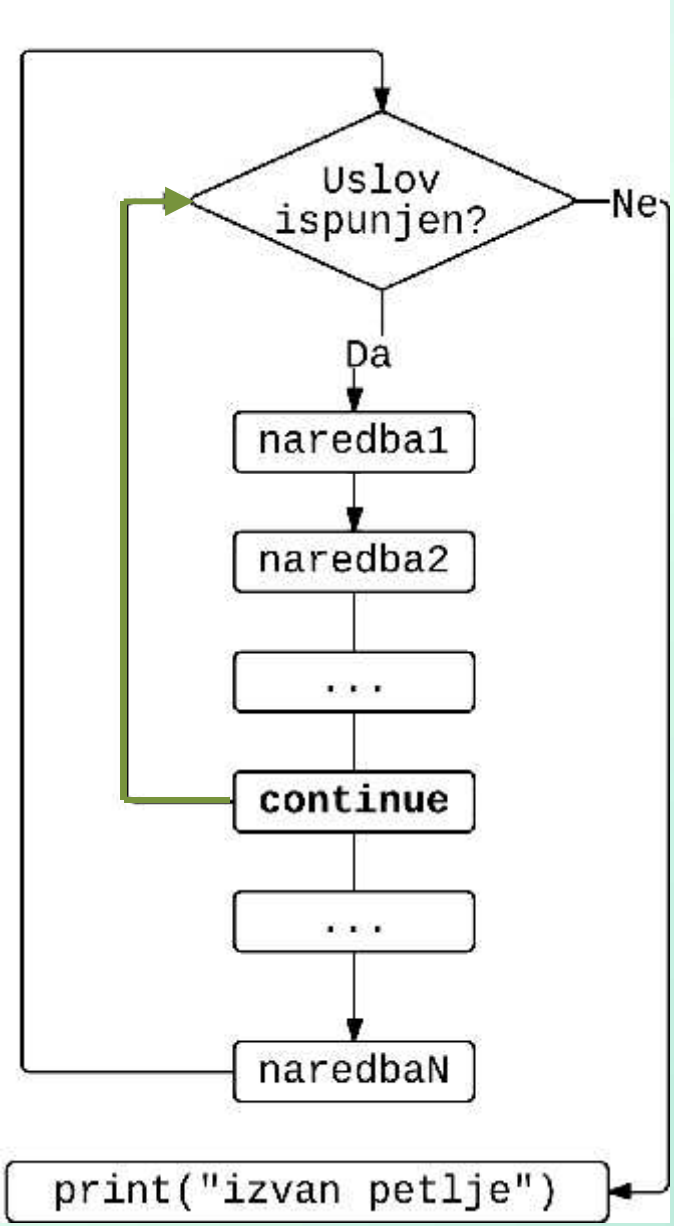
for – funkcija range

```
for broj in range(2, 7):
    print ("Broj: {}".format(broj))
```

- Range vraća listu (niz) brojeva između 2 i 7
  - Donja granica je uključena
  - Gornja nije
- Iterator – **`broj`** – uzima redom vrednosti iz te liste, od 2 do 6, zaključno sa 6.

Naredbe break i continue





Liste - uvod

- Lista = linearna kolekcija (raznorodnih) podataka
- Prazna lista se definiše kao `[]`
  - `prazna_lista = []`
- Liste su promenljiv tip objekta (mutable object type)
  - Sadržaj liste se može menjati nakon stvaranja liste

Liste – funkcije (1)

- Definisanje liste sa inicijalnim elementima:
  - `jezici = [“python”, “c++”, 42, -1.0]`
  - Dozvoljeno je mešati “babe i žabe” (type mixing)
- Dodavanje elementa na kraj liste:

```
jezici.append([“java”, “javascript”])
```

- Veličina liste:

```
len(jezici)
```
→ Da li je veličina liste 5 ili 6?

- Brisanje elementa iz liste:

```
del jezici[4]# po indeksu, 0-based indexing

jezici.remove(-1.0)
  # po vrednosti, samo prva instanca
```

Liste – funkcije (2)

- Dohvatanje i-tog elementa liste:
  - `print jezici[2]` → 42 # 0-based indexing, a[i - 1] → i-ti element
  - `print jezici[-1]` → 42 # a[-i] predstavlja i-ti element sa kraja liste
- Minimalni/maksimalni element liste:
  - `print min(jezici)` → 42
  - `print max(jezici)` → python
  - https://docs.python.org/2/library/stdtypes.html#comparisons
- Sortiranje liste:
  - ```
    print sorted(jezici)
    # rezultat je ispisana nova sortirana lista
    ```
  - ```
    jezici.sort() # lista jezici se sortira
    print jezici
    ```
- Ispitivanje postojanja elementa:
  - `if "python" in jezici: ...`

Liste – funkcije (3)

- Suma elemenata liste:
  - `brojevi = [1, 2, 3, 4, 5]`
  - `print sum(brojevi)` → 15
- Dodavanje liste na kraj liste:
  - `brojevi.extend([7, 8, 9])`
  - `brojevi += [7, 8, 9]`
  - `brojevi = brojevi + [7, 8, 9]`
  - `brojevi.append([7, 8, 9]) # NIKAKO!!!`
- Umetanje elementa na zadatu poziciju:
  - `brojevi.insert(0, 42)`
    `# dodajemo 42 na početak liste, tj. na poz. 0`

Liste – funkcije (4)

- Izbacivanje elementa sa zadate pozicije:
  - `treci_element = brojevi.pop(2)`
    `# povratna vrednost je dati element`
  - `poslednji_element = brojevi.pop()`
    `# podrazumevana vrednost argumenta je len - 1`
- Odsecanje podliste (slicing):
  - `podlista_brojeva = brojevi[low:high]`
    `# elementi iz opsega [low, high)`
- Prebrojavanje instanci određene vrednosti:
  - `broj_petica = brojevi.count(5)`

Liste – funkcije (5)

- Iteriranje po članovima liste:
  - `for element in list: # element je nepromenljiv`
  - da bi smo menjali članove liste moramo ih dohvatiti po indeksu
  - možemo koristiti funkciju `range(i, j, [k])` koja vraća listu koja sadrži elemente u intervalu `[i, j]` sa korakom k između dva susedna elementa, k je podrazumevano 1
  - ```
    for i in range(len(brojevi)):
        brojevi[i] = 42
    ```
- Množenje liste brojem:
  - veoma korisno za incijalizaciju kada znamo broj elemenata unapred
  - `sve_jed = [1] * 100 # [1, 1, 1, …]` – 100 elemenata
    ```

Tuple – uvod i funkcije

- Ponekad želimo da naše liste budu nepromenljive (immutable)
- U Pythonu upravo postoji takav ugrađen tip koji se naziva *torka* (eng. **tuple**)
- Kreiranje torke:
  - `moja_torka = (1, 2, 3, 4, 5)`
- Metode i funkcije iste kao kod listi (ako su dozvoljene!)
- Torke od jednog elementa se inicijalizuju sa zarezom na kraju, kako bi se razrešila sledeća dvosmislenost:

```
(4 + 2) # aritmeticki izraz
(4 + 2, ) # torka
```

Matrice – uvod i funkcije

- Matrice, kao i nizovi, nisu deo jezika, pa ih moramo sami napraviti
- Matricu pravimo kao listu listi:
  - matrix = [[value] * numberOfColumns] * numberOfRows # NIKAKO!!!
  - dobićemo jednu istu listu ponovljenu onoliko puta koliko ima redova
  - svaki red će biti jedna ista lista, kada menjamo element u jednom redu, promeniće se svi elementi u svim redovima koji su na istoj koloni
  - matrix = [[value] * numberOfColumns for i in range(numberOfRows)]
  - ovako već može ☺
- Ne postoje posebne metode i operacije nad matricama
  - možemo koristiti metode i operacije za liste nad redovima koji su liste

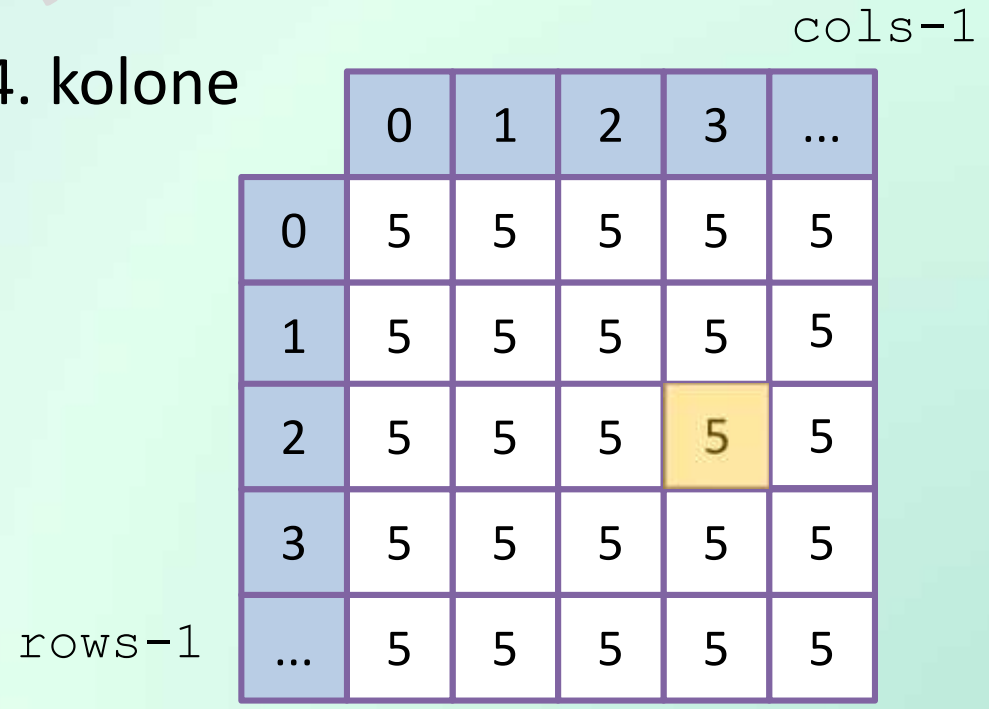
Matrice – primer

```
matrix = [[5] * cols for i in range(rows)]
```

Pristup elementu matrice:

matrica[2][3]

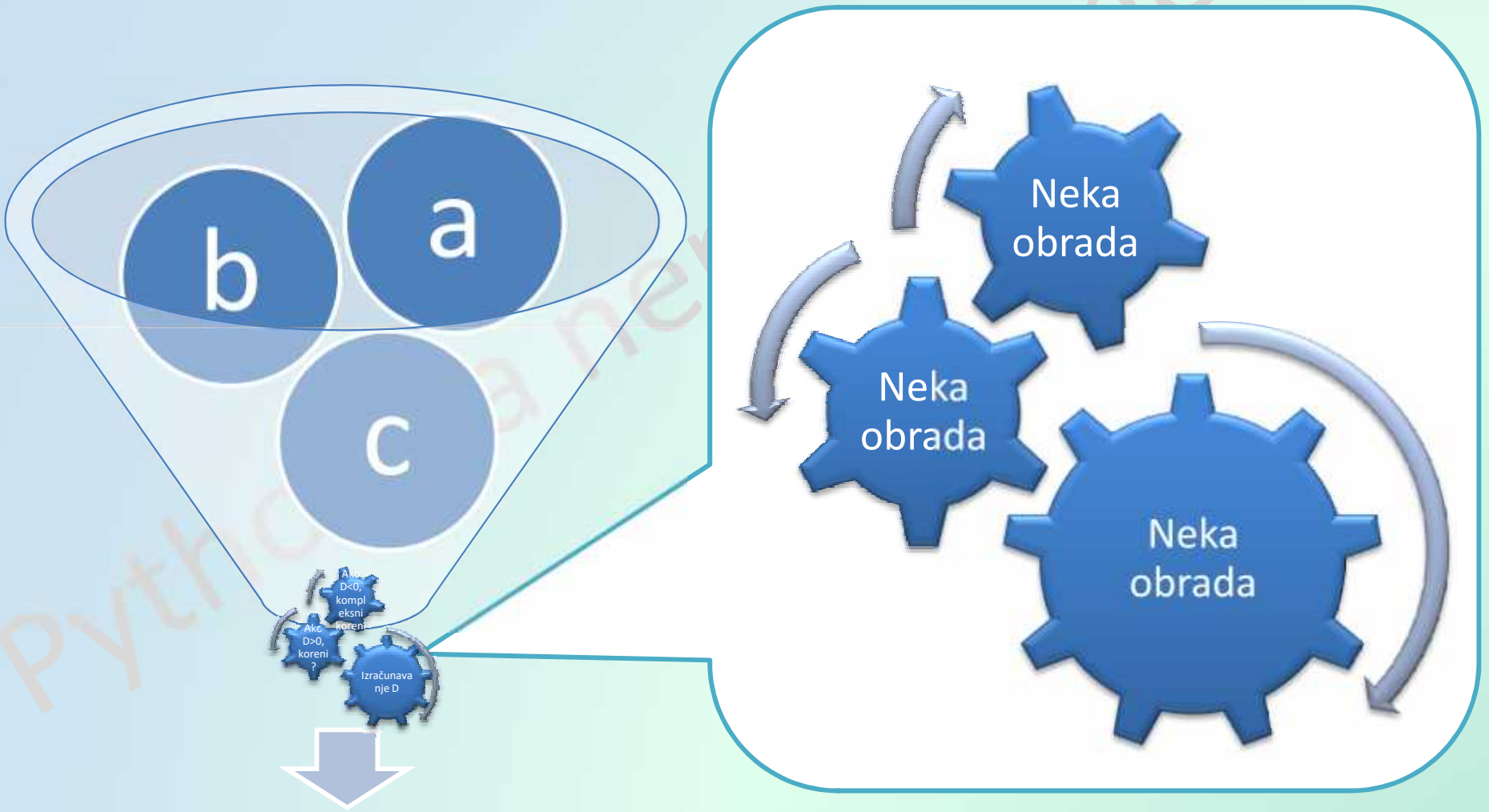
pristupa elementu 2. vrste, 4. kolone



Rečnici

- Rečnik = asocijativna kolekcija
  - podatku pridružujemo ključ
  - brzo pronalaženje po ključu
- Sintaksa:
  - `>>> a = { kljuc1 : vrednost1, kljuc2 : vrednost2 }`
- Primer:
  - `gradovi = { "Francuska" : "Pariz", "Japan" : "Tokio" }`
  - `gradovi["Francuska"] → "Pariz"`
  - `gradovi.has_key("Nemacka") → False`

Uvod u funkcije



Definicija funkcije

```
def farenheit_to_celsius(temp_f):
    temperatura_c = (temp_f- 3.2) / 1.8
    # proizvoljne naredbe...
    return temperatura_c
```

`temp_f` dobija konkretnu vrednost prilikom poziva funkcije.

Funkcija treba da obavi nekakav zadatak ili izvrši nekakva izračunavanja i pruži svom pozivaocu rezultat svog rada

Napomene:

- Mora biti definisana pre “glavnog” programa
- Redosled definisanja proizvoljan
- Glavni program: kôd koji ne pripada ni jednoj funkciji

Sintaksa funkcija

```
def farenheit_to_celsius(temp_f):
    temperatura_c = (temp_f- 3.2) / 1.8
    # proizvoljne naredbe...
    return temperatura_c
```

- Unutar zagrada: spisak imena argumenata koji se predaju funkciji
- Iza dve tačke (:), u narednom redu započinje blok koda
  - to je ***telo*** funkcije
- Mogu biti proizvoljne naredbe
  - Telo funkcije sadrži instrukcije kojima se definiše način na koji se obavljaju izračunavanja
- **Indentacija** je od suštinskog značaja !
  - svaka naredba mora biti uvučena za 1 TAB
- Ne može da se definiše funkcija u funkciji (lokalna funkcija)

Poziv i povratna vrednost

- Funkcija obično ima rezultat
- Ključna reč **`return`** za vraćanje rezultata – to je vrednost funkcije

```
temperatura_c = farenheit_to_celsius(100)
```

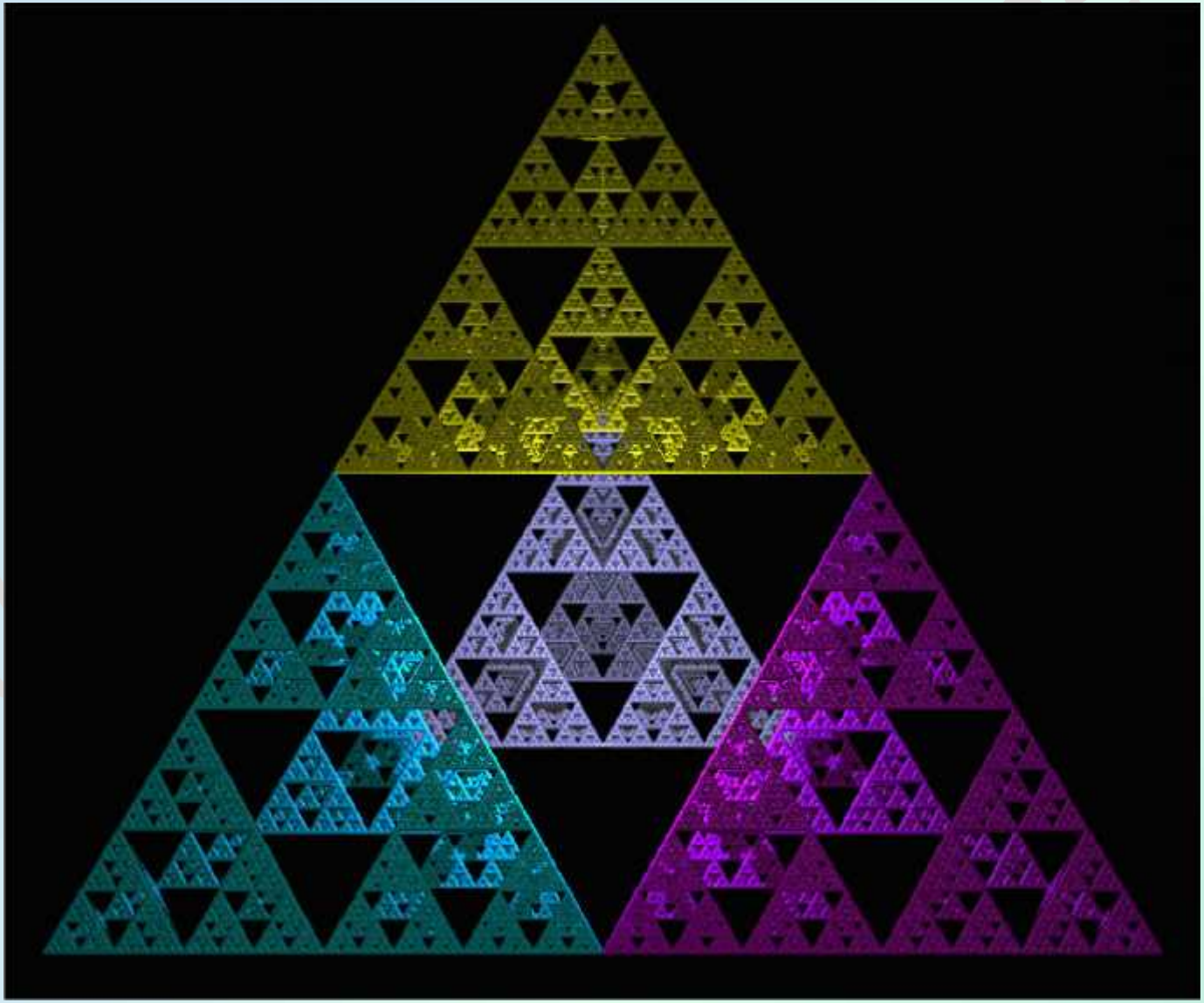
- Funkcija ponekad ne mora imati rezultat – samo nešto uradi (npr. odštampa nešto, sortira niz…)
- Moguće navesti **`return`** bez ikakvog izraza iza
  - Funkcija se prekida
  - Vraća se specijalna vrednost **`None`**

Rekurzivna funkcija

- Funkcija koja poziva samu sebe
  (sa drugim vrednostima parametara) je **rekurzivna**
- Rekurzija je postupak rešavanja problema
  koji među svojim koracima sadrži referisanje samog tog postupka
  - razlaganje složenog problema na jednostavnije, ali iste prirode
  - razlaganje se završava kada je problem dovoljno jednostavan
    da se trivijalno rešava

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

Rekurzija - Primeri



Rekurzija - Primeri