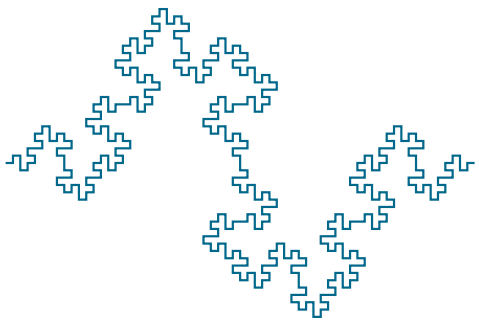
Uvod u programski jezik `Python`

Optimizacione i upravljačke tehnologije u arhitektonskom projektovanju

Milan R. Rapaić

Fakultet tehničkih nauka, Novi Sad



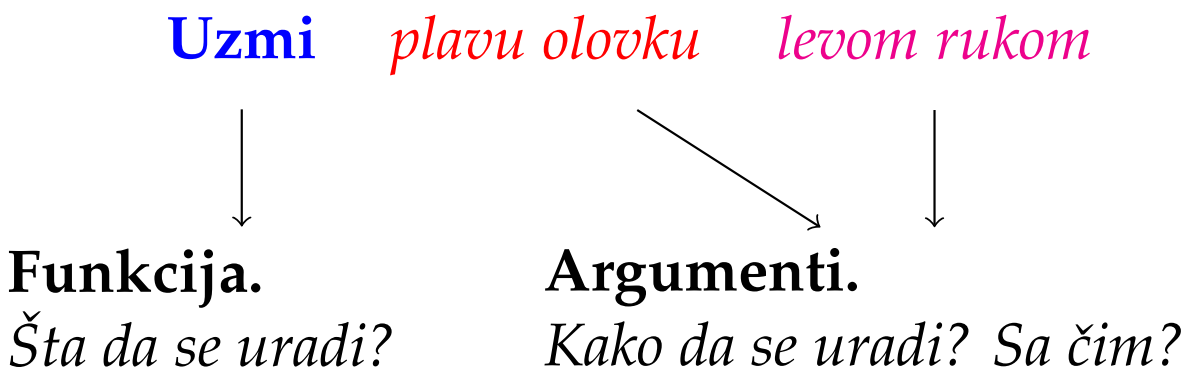
24. septembar 2013

Osnovni pojmovi

PROMENLJIVE I FUNKCIJE, ARITMETIČKI I LOGIČKI OPERATORI, USLOVNO IZVRŠAVANJE KODA

Pojam funkcije i argumenata

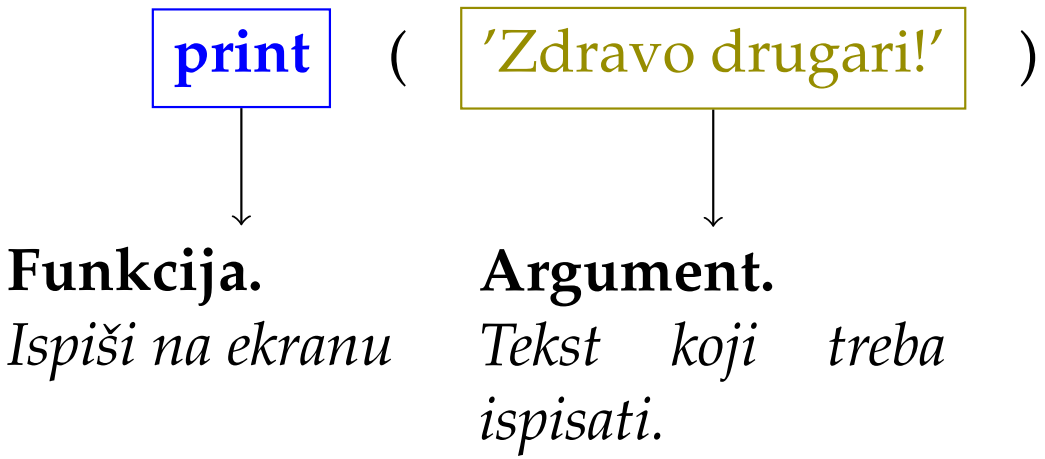
- Kao i govorni jezici, **svaki programski jezik je sredstvo komunikacije**. Govorni jezici su sredstvo komunikacije među ljudima. **Programski jezik služi za komunikaciju između čoveka i mašine (računara).**



PRIMER

Primer 1: Ispisivanje poruke na ekranu

```
print('Zdravo drugari!')
```



FUNKCIJE U PYTHON-U

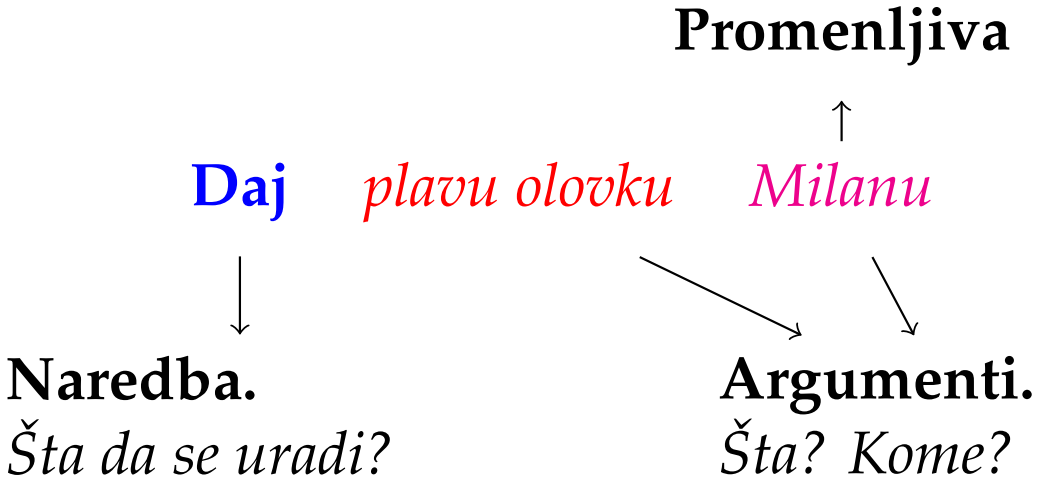
OSNOVNI POJMOVI

- Svaka funkcija u `Python`-u je jednoznačno određena svojim **imenom**.
- **Pozivanjem** funkcije od računara se zahteva da se funkcija **izvrši**.
- Svaka funkcija može imati *ni jedan*, *jedan ili više* **argumenata**. Argumenti bliže određuju način na koji se funkcija izvršava. Argumentima se, takođe, mogu prosleđivati informacije neophodne za izvršenje naredbe.
- Funkcije mogu biti ugrađene ili mogu biti definisane od strane korisnika. Zapravo, možemo razlikovati tri vrste funkcija
  1. Funkcije koje su **ugrađene** (*built-in*), definisane u samom `Python`-u
  2. Funkcije koje moramo uvoziti iz spoljašnjih **paketa** (*package*) i **modula** (*module*) (objasnićemo kasnije ...). Neke od ovih funkcija su definisali stvaraoci `Python`-a. Na raspolaganju su, međutim, i mnogobrojni paketi posebne namene koje su projektovali i implementirali ljudi koji nemaju neposrednog kontakta sa razvojnim timom `Python`-a.
  3. Korisničke funkcije – Funkcije koje definišemo mi sami.

Poziv funkcije

```
ime_funkcije (argument1, argument2, ...)
```

Promenljive



- Promenljive su imena koja opisuju vrednosti, podatke, međurezultate u proračunima, te uopšte *bilo koje objekte u programu kojima ćemo pristupati kasnije*.
- Poput naredbi (funkcija), i promnljive imaju svoja **imena**.
- Promenljive se veoma često koriste kao argumenti naredbi (funkcija).

Promenljiva kao argument u pozivu naredbe (funkcije)

```
poruka = 'Zdravo drugari!'
print(poruka)
```

Osnovni tipovi promenljivih

- Svaka promenljiva se, sem imenom, karakteriše i svojim **tipom**. Tip promenljive određuje prirod informacije koju promenljiva nosi.
- Poput naredbi, tipovi promenljivih mogu biti ugrađeni, definisani u odgovarajućem spoljnjem paketu ili modulu, a možemo ih definisati i mi sami.

Tip promenljive	**Primeri**
ceo broj (`int`)	1, -6, ...
realan broj (`float`)	1.0, -3.14, 6.22e-8, ...
logička vrednost (`bool`)	**True**, **False**
tekst (`str`)	'Proizvoljan niz karaktera', ...

Dodela vrednosti

Operacija =

Jednostavne dodele vrednosti

```
x = 2
y = 6.0
z = 'Zdravo'
w = True
```

Dvostruke dodele vrednosti

```
x, y = 2, 6.0
z, w = 'Zdravo', True
```

Višestruke dodele vrednosti

```
x, y, z, w = 2, 6.0, 'Zdravo', True
```

Aritmetičke operacije

Operacije nad numeričkim vrednostima

Operacija	Primeri
sabiranje (+)	`1+2`
oduzimanje (-)	`1-2`
množenje (*)	`1*2`
deljenje (/)	`1/2`
celobrojno deljenje (//)	`1//2`
ostatak pri deljenju (%)	`1%2`
stepenovanje (**)	`1**2`

Primeri

```
x = 1 + 2    # x <- 3
x = 1 // 2    # x <- 0
x = 1 / 2    # x <- 0 ili x <- 0.5 u zavisnosti od verzije.
x = 1.0 / 2    # x <- 0.5!  Prvi argument je realan!
x = 1 / 2.0    # x <- 0.5!  Drugi argument je realan!
```

Logičke operacije

Operacije nad logičkim vrednostima

Operacija	Primeri
logičko **i** (**and**)	`x` **and** `y`
logičko **ili** (**or**)	`x` **and** `y`
logičko **ne** (**not**)	**not** `x`

and

	T	**F**
T	**T**	**F**
F	**F**	**F**

or

	T	**F**
T	**T**	**T**
F	**T**	**F**

not

	T	**F**
	F	**T**

Operacije poređenja

Logičke operacije nad numeričkim vrednostima

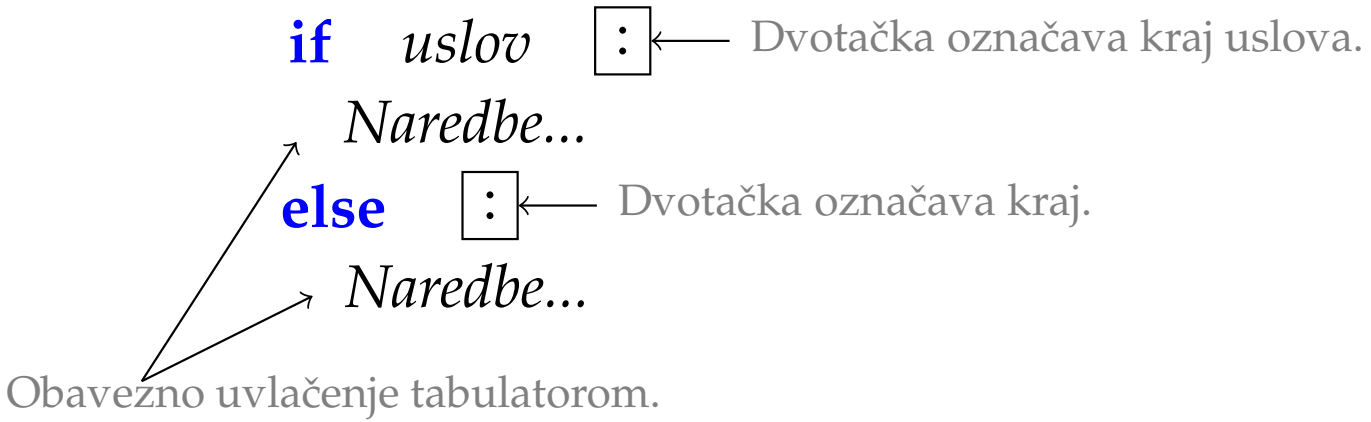
Operacija	Primeri
jednako (==)	`x == y`
veće od (>)	`x > y`
manje od (<)	`x < y`
veće ili jednako od (>=)	`x >= y`
manje ili jednako od (<=)	`x <= y`
nejednako (različito) (<> ili !=)	`x != y`

- Operacije poređenja su definisane i za mnoge nenumeričke tipove podataka.
- Svaka od ovih operacija se može definisati i za proizvoljne tipove podataka koje definiše korisnik.
- Složena poređenja se mogu načiniti pomoću logičkih operacija, recimo:
  `x>3 and x<=18 and (x % 2 != 0)`
  je tačno za sve parne brojeve između 3 i 18.

USLOVNO IZVRŠAVANJE

NAREDBE **if** I **if-else**

- Uslovno izvršavanje koda omogućava da se dati deo koda izvrši samo pod uslovom da je određeni uslov zadovoljen.



Primer

```
if y>6:
    x = 3
```

Primer

```
if y>6:
    x = 3
else:
    x = 2
```

Povratne vrednosti funkcija

- `Python` funkcije nejčešće obrade argumente i proizvedu odgovarajući rezultat.
- *Svaka* funkcija u `Python`-u nakon poziva "vraća" **povratnu vrednost**.
  - > Primera radi, poziv `type(p)` vraća tip promenljive `p`.
- Čak i funkcije koje ne vrše proračune, formalno vraćaju vrednost. Takva povratna vrednost ima i posebno ime: **None**.
  - > Primer funkcije koja vraća **None** vrednost je **print** funkcija.
- Formalno posmatrano, funkcije u `Python`-u mogu tačno jednu povratnu vrednost. Primenom nekih složenijih jezičkih konstrukata (koje ćemo definisati kasnije), može se postići da funkcije *prividno* vraćaju veći broj vrednosti.
  - > Primer funkcije koja vraća složenu povratnu vrednost (prividno dve vrednosti) je ugrađena funkcija **divmod**. Ova funkcija jednovremeno vraća i količnik i ostatak pri celobrojnom deljenju brojeva.

Primer

```
x = abs(-2)      # x <- 2.
print(x)     # Ispisuje 2 na ekranu.
a, b = divmod(4, 3)     # a <- 1, b <- 1
```

DEFINISANJE FUNKCIJA

- Kada se neki deo koda u istom ili sličnom obliku ponavlja iznova i iznova, pogodno ga je definisati u obliku funkcije.
- Modularizacija koda (podela na manje segmente jasno definisane namene) poželjna je (i neophodna) iz više razloga
  - > Funkcija se definiše na jednom mesto, a poziva proizvoljan broj puta.
  - > Funkcija se može testirati nezavisno od koda u kome će se pozivati.
  - > Sve izmene treba vršiti na jednom i samo jednom mestu.

```
def ime_funkcije (arg1, arg2, ..., argn) :
    Naredbe...
```

Definisanje funkcija

Primer

Definisati funkciju koja za dati uneti broj (nazovimo ga x) vraća broj $x + 6$, ako je x paran broj, a $x/2$ ako je x neparan broj.

Definicija funkcije

```
def primer (x):
  if x % 2 == 0:    # Ispitujemo da li je x paran broj.
    return x+6   # Ako jeste, vracamo x+6.
  else:
    return x/2   # Ako nije, vracamo x/2.
```

Poziv funkcije

```
y = primer(6)
print(y)
```

Definisanje funkcija

Alternativno, čitkije rešenje

- Jedan od razloga zbog koga je zgodno i potrebno definisati funkcije jeste radi povećanja *čitljivosti* koda. `Python` je jezik koji je čuvan po čitljivosti svog programskog koda (sa tom je namenom i projektovan).
- Konstrukt **if** `x % 2 == 0` se nakon dovoljno dugo vremena može učiniti nečitkim (naročito ukoliko ne proveravate parnost brojeva na ovaj način tako često).
- Umesto toga, korisnije je definisani posebnu funkciju koja samo proverava parnost, a potom tu funkciju koristiti u uslovu.

Definicija funkcije – Drugi način

```
def is_even(x):
  return x % 2 == 0   # Ako je x parno, ostatak je nula.

def primer (x):
  if is_even(x):    # Ispitujemo da li je x paran broj.
    return x+6   # Ako jeste, vracamo x+6.
  else:
    return x/2   # Ako nije, vracamo x/2.
```

USLOVNA DODELA VREDNOSTI

ILI: JOŠ JEDAN NAČIN DA SE URADI PRETHODNI ZADATAK...

- Uslovna dodela vrednosti omogućava da se na čitkiji, jednostavniji način promenljivoj dodeli vrednost u zavisnosti od ispunjenosti nekog uslova.

Sintaksa

```
promenljiva = vrednost 1 if uslov else vrednost 2
```

Definicija funkcije – Treći način

```
def is_even(x):
  return x % 2 == 0   # Ako je x parno, ostatak je nula.

def primer (x):
  return x+6 if is_even(x) else x/2
```

Kolekcije i petlje

LISTE, N-TORKE, SKUPOVI, REČNICI
for PETLJA I **while** PETLJA

KOLEKCIJE

- Kao što im i samo ime kaže, kolekcije su objekti koji služe za grupisanje i organizaciju drugih objekata.
- Kolekcije se među sobom razlikuju prema strukturi (odnosno prema uređenju objekata koje sadrže), te prema performansama.
- Pozivom **len** `(k)` dobija se broj elemenata kolekcije `k`. Pripadnost elementa `e` kolekciji `k` proverava se konstrukcijama `e` **in** `k`, odnosno `e` **not in** `k`.

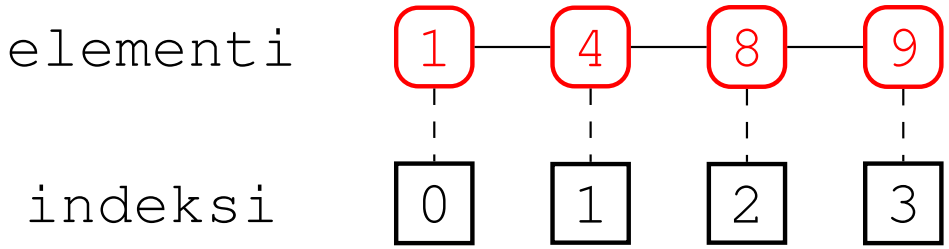
list Lista predstavlja niz objekata koji slede jedan iza drugog. Svaki objekat se na jedinstven način može identifikovati pomoću **indeksa**. Indeks predstavlja rednu broj objekta u listi, pri čemu se početni objekat indeksira nulom, a potonji objekti jedan za drugim brojevima 1, 2, ... Poslednji objekat ima indeks L-1, gde je L dužina liste (broj objekata koji su u njoj sadržani).

tuple N-torka je strukturno identična listi, razlika je u performansama i načinu upotrebe.

dict Rečnik predstavlja skup parova **ključ/vrednost**, pri čemu je **ključ** identifikator koji na jedinstven način identifikuje **vrednost**. Rečnik je, u suštini, preslikavanje koje svakom ključu pridružuje vrednost na jedinstven način.

set Neuređen skup objekata koje se ne mogu ponavljati.

LISTE



Primeri

```
x = [1, 4, 8, 9]   # Lista elemenata
y = []   # Prazna lista
z = [2.0, 'Zdravo']   # Lista koja sadrzi raznorodne elemente
q = ['Zdravo', [1.3, 2], []]   # Lista koja sadrzi druge liste
```

- Lista predstavlja niz objekata (proizvoljnog tipa, objekti u istoj listi mogu biti raznorodni, a neki ili svi i sami mogu biti liste).
- Svakom elementu liste na jednoznačan način se pridružuje nenegativan ceo broj koji nazivamo **indeksom**. Kažemo da se lista indeksira od nule.

INDEKSIRANJE LISTI

PRISTUPANJE ELEMENTIMA LISTE...

- Indeksiranje jeste pristupanje elementima liste po rednom broju, počev od 0.
- Indeksiranje se vrši pomoću uglastih zagrada: `ime_liste[indeks]`.

Primer

```
lst = ['a', 'b', 'c', 'd']   # Definisanje liste
print(lst[0])   # Na ekranu se ispisuje 'a'.
lst[1] = '2'   # Element 'b' se zamenjuje sa '2'.
```

- Ukoliko se zahteva element koji ne postoji u listi (recimo da lista ima ukupno 5 elemenata, sa zahteva se 10-ti), javlja se greška.
- Dužina liste, odnosno broj elemenata u listi, dobija se pozivom funkcije **len**.
- Lista se može indeksirati i negativnim brojevima. Tada je -1 poslednji, -2 predposlednji, itd. broj u listi.

Primer

```
N = len(lst)   # Promenljiva N sadrzi broj elemenata liste.
lst[N]   Grska!  Psolednji element liste indeksira se sa N-2
```

SEČENJE LISTI (*slicing*)

PRISTUPANJE PODLISTAMA...

- Sečenje jeste postupak formiranja podlisti.
- Sečenje se vrši pomoću sintekse slične indeksiranju. Indeksiranje se može vršiti u osnovnoj varijanti (specificiranjem početnog i krajnjeg indeksa) i u proširenoj varijanti (specificiranjem početnog i krajnjeg indeksa i koraka)

```
> ime_liste[start_indeks:stop_indeks]
> ime_liste[start_indeks:stop_indeks:korak]
```

- Pri indeksiranju, element čiji je indeks jednak `stop_indeks`-u se *ne uzima* u obzir. (Sem ukoliko `start_indeks` i `stop_indeks` nisu jednaki).
- Ukoliko se prvi indeks izostavi, smatra se seče od prvog elementa.
- Ukoliko se poslednji indeks izostavi, smatra se da se seče do zadnjeg elementa.

Primeri sečenja liste

```
lst = ['a', 'b', 'c', 'd']   # Definisanje liste
x = lst[0:1]   # x <- ['a']
y = lst[1:3]   # x <- ['b', 'c']
z = lst[1:]    # z <- ['b', 'c', 'd']
```

SEČENJE LISTI (*slicing*)

SLOŽENIJI PRIMERI

Primeri sa negativnim indeksima

```
lst = ['a', 'b', 'c', 'd']   # Definisanje liste
x = lst[1:-1]   # x <- ['b', 'c']
y = lst[1:-2]   # x <- ['b']
```

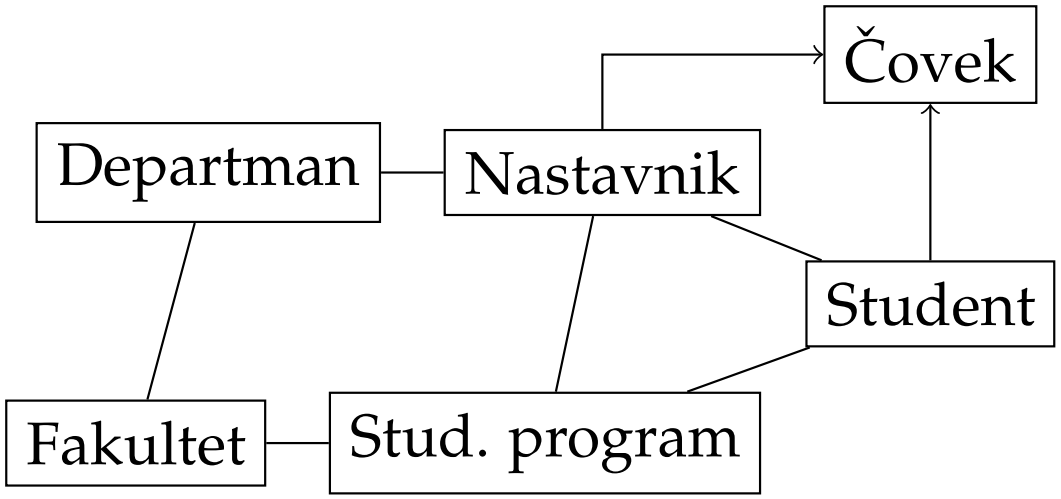
Primeri sa korakom

```
lst = ['a', 'b', 'c', 'd']   # Definisanje liste
x = lst[1:4:2]   # x <- ['b', 'd']
y = lst[::2]   # x <- ['a', 'c'] (svaki drugi element)
z = lst[::-1]   # x <- ['d', 'c', 'b', 'a'] (obrnuti poredak)
```

OBJEKTI

OSNOVNI ELEMENTI OBJEKTNO-ORIJENTISANOG PROGRAMIRANJA

- Svaki element programskog koda u `Python`-u je **objekat** (u OO smislu). Brojevne i logičke konstante su objekti, tekst je objekat, liste su objekti. Čak su i same funkcije, moduli i paketi zapravo objekti u `Python`-u.
- Objekat se može shvatiti kao *podatak* (ili skup podataka) sa pridruženim *ponašanjem*. Uopštenje pojma tipa podataka je **klasa** – *klasa je tip objekta*.
- Objekti mogu biti međusobno povezani raznolikim vezama.
- Za naše primene, upotreba objekata se može smatrati naprednom tehnikom programiranja. Ipak, razumevanje elementarne sintakse OO programiranja u `Python`-u je neophodno.



Objekti

Funkcije članice

- Funkcije članice su funkcije koje "pripadaju" nekom objektu. *Funkcije članice opisuju ponašanje klase.*
- Funkcije članice se pozivaju na specifičan način: `ime_objekta.ime_funkcije(arg1, arg2, ...)`
- Ime objekta ima dvojaku ulogu:
  - Sa jedne strane, ime objekta je deo imena funkcije. Funkcija pripada objektu, pa je ime objekta deo imena same funkcije. Takav šablon imenovanja se javlja na više mesta u `Python`-u.
  - Sa druge strane, sam objekat je prvi argument funkcije članice – svaka funkcija članica može da pristupa podacima koji pripadaju objektu.

OSNOVNE OPERACIJE NAD LISTAMA

NEKE FUNKCIJE ČLANICE KLASE **list**

append `lst.append(e)` dodaje element `e` na kraj liste `lst`.

extend `lst.extend(l)` dodaje sve elemente liste `l` na kraj liste `lst`.

insert `lst.insert(index, object)` umeće `objekat` pre indeksa `index`. Nakon umetanja `lst[index]` će biti jednako `object`.

index `lst.index(value)` vraća indeks na kome se vrednost `value` pojavljuje prvi put. Dodatnim argumentima mogu se specificirati početni i krajnji indeks pretrage. U slučaju da traženi element ne postoji, javlja se greška.

count `lst.count(value)` vraća broj ponavljanje vrednosti `value` u listi.

remove `lst.remove(value)` uklanja prvo pojavljivanje vrednosti `value` u listi. Javlja grešku ukoliko data vrednost ne postoji.

Bitna operacija nad listama koja nije implementirana kao funkcija članica jeste uklanjanje elementa po indeksu. Ova operacija se vrši pomoću ugrađene funkcije **del**. Tako recimo, **del** `(lst[i])` uklanja element sa indeksom `i` iz liste.

N-TORKE (*tuples*)

- Koncepcijski, N-torke se ne razlikuju od listi – predstavljaju uređene skupove vrednosti koje se mogu ponavljati. Svaka vrednost je na jednoznačan način određena svojim rednim brojem (indeksom).
- N-torke se definišu na sličan način kao liste, jedino se uglaste zagrade zamenjuju običnim. Recimo `(1, 2)`.
- N-torke se indeksiraju na isti način kao i liste.
- Za razliku od listi koje su osmišljene kao fleksibilni, promenljivi objekti (*mutable*) N-torke su osmišljene kao kruti, nepromenljivi objekti (*immutable*). Posledično, neke od operacija koje možemo vršiti na listama (kao što je umetanje vrednosti, uklanjanje elementa i sl.) ne mogu se vršiti nad N-torkama. Sa druge strane, operacije koje se mogu vršiti nad N-torkama obično su znatno efikasnije od odgovarajućih operacija nad listama.
- N-torke se široko koriste pri implementaciji različitih funkcionalnosti u samom `Python`-u.

N-torke (*tuples*)

Operacije nad n-torkama

Primer – Dozvoljene operacije

```
tpl = ('a', 'b', 'c', 'd')   # Definisanje N-torke
x = tpl[1]   # x <- 'b'.  Pristup elementu sa indeksom 1
y = tpl[1:3]   # y <- ('b', 'c').  Secenje N-torke.
z = tpl[::-1]   # z <- ('d', 'c', 'b', 'a').  Obrce redosled.
z.count(value)   # Isto kao kod listi.
z.index(value)   # Isto kao kod listi.
```

Primer – Nedozvoljene operacije

```
tpl = ('a', 'b', 'c', 'd')   # Definisanje N-torke
tpl[1] = 'f'   # Nedozvoljeno!
tpl.append('f')   # Nedozvoljeno!
tpl.extend(k)   # Nedozvoljeno!
tpl.remove(val)   # Nedozvoljeno!
del(tpl[1])   # Nedozvoljeno!
```

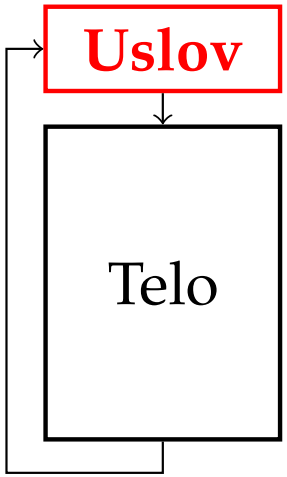
PETLJE

OSNOVNI POJMOVI

Petlje su programske konstrukcije koje omogućavaju da se deo koda izvršava veći broj puta. Razlikujemo dve vrste petlji:

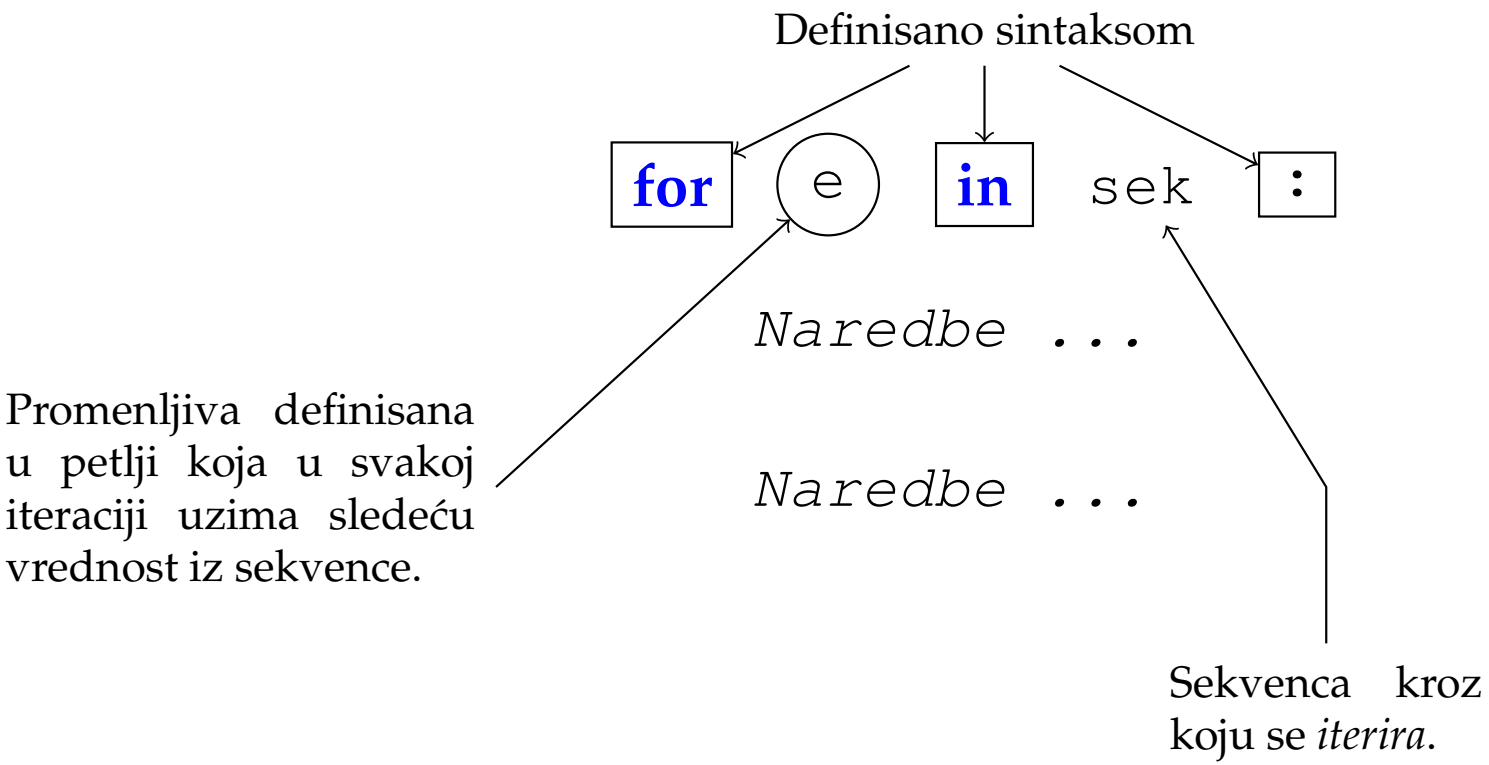
for-petlja Nabrojiva ili **for** petlja jeste petlja u kojoj se kod izvršava po jednom za svaki element neke sekvence (recimo liste ili N-torke).

while-petlja Nenabrojiva ili **while** petlja jeste petlja u kojoj se kod izvršava sve dok je neki *logički uslov* ispunjen.



- **Iteracija** predstavlja jedan ciklus petlje (odnosno skup operacija koje se izvrše u jednom ciklusu).
- **for** petlja se izvršava unapred poznat broj iteracija.
- **while** petlja se izvršava broj puta koji je unapred (u principu) nepoznat - iteracije slede jedna za drugom sve dok je uslov ispunjen.

for PETLJA


Definisano sintaksom
for
e
in
sek
:
Naredbe ...
Naredbe ...
Promenljiva definisana u petlji koja u svakoj iteraciji uzima sledeću vrednost iz sekvence.
Sekvenca kroz koju se *iterira*.

JEDNOSTAVNI PRIMERI UPOTREBE PETLJI

Primer

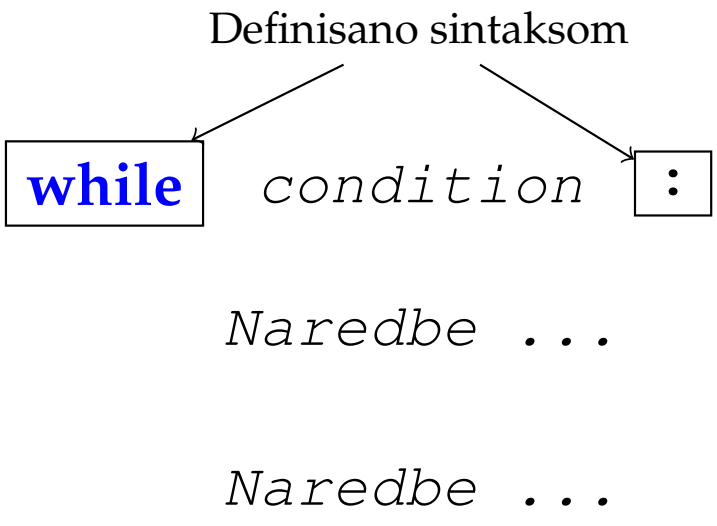
```
# Kod koji ispisuje svaki element liste na ekranu
for e in lst :
  print(e)
```

Veoma često (*gotovo uvek*) elementima liste je zgodno pristupati po indeksu. To je naročito slučaj ukoliko jednovremeno treba pristupati većem broju listi, ili ukoliko je potrebno prostupati prethodnom i sledećem elementu datog elementa. Tada se koristi ugrađena funkcija **range**. **range** (N) vraća sekvencu čiji su elementi 0, 1, ..., N-1. Postoji i varijanta naredbe kojom se mogu zadati i početni indeks, kao i korak (slično kao kod sečenja liste).

Primer

```
# Kod koji u listu a upisuje kvadrate elemenata u listi b
for i in range(len(a)) :
  b[i] = a[i]**2
```

while PETLJA



Numpy i Matplotlib

OSNOVI PAKETA NUMPY I MATPLOTLIB