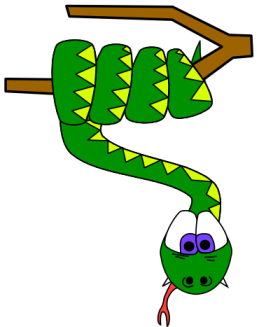


Uvod u Python (II dio)

Slavica Tomović (slavicat@ucg.ac.me)

Elektrotehnički fakultet, Podgorica

Univerzitet Crne Gore



Složeni tipovi podataka

Lista i tuple

- Tuple – nepromjenjivi skup elemenata koji predstavlja jednu logičku cjelinu
- Lista – promjenjivi niz podataka bilo kojeg tipa

- Tuple se definiše pomoću običnih zagrada i zareza:

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Liste se definišu pomoću uglastih zagrada i zareza:

```
>>> li = ["abc", 34, 4.34, 23]
```

- Elementima jednog tupla ili liste pristupamo isto kao karakterima u stringu – korišćenjem uglastih zagrada:

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

```
>>> tu[1]          # Drugi element tupla.
```

```
'abc'
```

```
>>> li = ["abc", 34, 4.34, 23]
```

```
>>> li[1]         # Drugi element liste.
```

```
34
```

Negativno indeksiranje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Pozitivno indeksiranje: gledano sa lijeva, brojimo od 0.

```
>>> t[1]
```

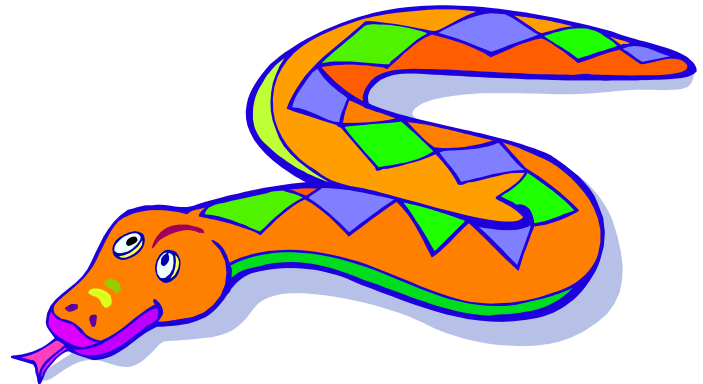
```
'abc'
```

Negativno indeksiranje: gledano sa desna, počinjemo od -1.

```
>>> t[-3]
```

```
4.56
```

•Na isti način indeksiraju se stringovi, liste i tuples.



Lista i Tuple - slajsovanje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Kako izdvojiti prva tri elementa tupla/liste?

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

Možemo koristiti i negativno indeksiranje za izdvanje poslednja tri elementa:

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Opcioni argument dozvoljava selekciju svakog n-tog elementa. Na primjer, izdvojimo svaki drugi element:

```
>>> t[1:-1:2]
('abc', (2,3))
```

Lista i Tuple - slajsovanje

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Ukoliko ne navedemo prvi indeks, podrazumijeva se da počinjemo od prvog elementa:

```
>>> t[:2]  
(23, 'abc')
```

Ukoliko ne navedemo drugi indeks, kopiranje se vrši od prvog indeksa pa do kraja:

```
>>> t[2:]  
(4.56, (2,3), 'def')
```

'in' operator

- Možemo provjeriti prisustvo elementa u listi korišćenjem „in“ operatora

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- Za stringove i podstringove:

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Operator „in“ takođe se koristi u okviru petlji:

```
t = [1, 2]
for element in t:
    print element
```

Rezultat: 1
2

'+' operator

- Operator + koristimo za povezivanje više stringova/lista/tuple-ova:

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
```

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

```
>>> "Hello" + " " + "World"
'Hello World'
```


Operacije nad listama

```
>>> li = [1, 11, 3, 4, 5]
```

```
>>> li.append('a') # Dodavanje elementa
```

```
>>> li
```

```
[1, 11, 3, 4, 5, 'a']
```

Dodavanje elementa na tačno određenoj poziciji:

```
>>> li.insert(2, 'i')
```

```
>>> li
```

```
[1, 11, 'i', 3, 4, 5, 'a']
```

Operacije nad listama

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')      # Indeks prvog pojavljivanja argumenta
1
>>> li.count('b')     # Broj pojavljivanja argumenta
2
>>> li.remove('b')    # Uklanja zadati argument na najmanjem indeksu
>>> li
['a', 'c', 'b']
```

❑ Obrtanje redosleda i sortiranje elemenata:

```
>>> li = [5, 2, 6, 8]
>>> li.reverse()     # Obrnut redosled elemenata
>>> li
[8, 6, 2, 5]
>>> li.sort()        # Sortiranje liste
>>> li
[2, 5, 6, 8]
```

Rečnici

Rečnici vrše mapiranje podataka koje nazivamo „ključevima“ sa „vrijednostima“.

- **Ključevi** mogu biti bilo koji nepromjenjivi tip podataka
- **Vrijednosti** mogu biti bilo koji tip podataka
- Jedan rečnik može da skladišti **vrijednosti** različitog tipa

Moguće je dodati, modifikovati, izbrisati bilo koji *ključ-vrijednost* par
Python rečnici su poznati i kao *hash* tabele i asocijativni nizovi

```
>>> d = { 'user' : 'bozo' , 'pswd' : 1234 }
```

```
>>> d[ 'user' ]
```

```
'bozo'
```

```
>>> d[ 'pswd' ]
```

```
123
```

```
>>> d[ 'bozo' ]
```

```
Traceback (innermost last):
```

```
File '<interactive input>' line 1, in ?
```

```
KeyError: bozo
```

Ažuriranje rečnika

- Ključevi moraju biti jedinstveni
- Dodjelom vrijednosti postojećem ključu briše se njegova stara vrijednost

```
>>> d = {'user': 'bozo', 'pswd': 1234}
>>> d['user'] = 'clown'
>>> d
{'user': 'clown', 'pswd': 1234}
>>> d['id'] = 45
>>> d
{'user': 'clown', 'id': 45, 'pswd': 1234}
```
- Redosled *key-value* parova u rečniku je neuređen. Par koji smo dodali prvi, možda se ne nalazi na prvom mjestu u rečniku.
- Brisanje elemenata rečnika:

```
>>> del d['user'] # Ukloni jedan par
>>> d
{'id': 45, 'pswd': 1234}
>>> d.clear() # Ukloni sve
```

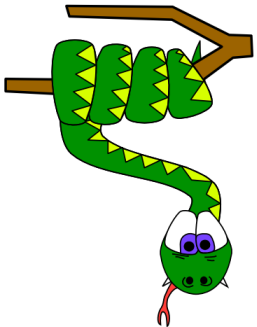
Korisne funkcije za pristup podacima u rečniku

```
>>> d = {'user': 'bozo', 'pswd': 1234, 'id': 34}
```

```
>>> d.keys() # Lista ključeva!  
['user', 'pswd', 'id']
```

```
>>> d.values() # Lista vrijednosti!  
['bozo', 1234, 34]
```

```
>>> d.items() # Lista tuple parova (ključ, vrijednost)  
[('user', 'bozo'), ('pswd', 1234), ('id', 34)]
```



Funkcije

Funkcije

Definisanje funkcije počinje sa **def**

Ime funkcije i njeni argumenti

```
def vrati_odgovor(imeFajla):  
    """ Objašnjenje """  
    linija1  
    linija2  
    return brojac
```

...

Prva linija sa manjim „razmakom“ (indentation) izvršava se van funkcije

‘return’ indicira vrijednost koja se vraća kao rezultat.

Nije potrebna deklaracija tipa ni za argumente ni za rezultat

Funkcije

- Pozivanje funkcije:

```
>>> def myfun(x, y):  
            return x * y  
>>> myfun(3, 4)  
12
```

- **Sve funkcije u Python-u vraćaju rezultat**
 - čak iako nema *return* izraza u kodu.
- **Funkcije bez *return* izraza vraćaju specijalnu vrijednost *None***
 - *None* je specijalna konstanta u programskom jeziku.
 - *None* je ekvivalent *null* konstanti u Javi.
 - Interpreter ne može da štampa *None*
- Dvije funkcije ne mogu imati isto ime

Funkcije

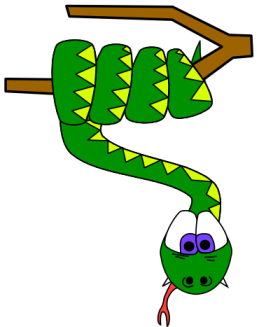
- Funkciji je moguće proslijediti default argumente

```
>>> def myfun(b, c=3, d="hello"):  
        return b + c  
  
>>> myfun(5, 3, "hello")  
>>> myfun(5, 3)  
>>> myfun(5)
```

Svi navedeni pozivi funkcije vraćaju rezultat 8.

- Redosled argumenata može biti izmijenjen:

```
>>> def myfun(a, b, c):  
        return a-b  
  
>>> myfun(2, 1, 43)  
1  
  
>>> myfun(c=43, b=1, a=2)  
1
```



Klase i objekti

Definisanje klase

- Definisanje klase:

```
class imeKlase:  
    izrazi
```

- Primjer:

```
class Tacka:  
    x = 0  
    y = 0  
  
p1 = Tacka()  
p1.x = 2  
p1.y = -5
```

tacka.py

```
1 class Tacka:  
2     x = 0  
3     y = 0
```

- Deklarisanje atributa može se vršiti unutar klase ili u konstruktoru

- Svaki atribut ili metod imenovan sa dvije „_“ crte na početku i jednom je privatnog tipa

- Nasleđivanje:

```
class etf_student(student):
```

Korišćenje klase

```
import class
```

Klijentski programi moraju importovati klasu koju koriste.

point_main.py

```
1  from Tacka import *
2
3
4  p1 = Tacka()
5  p1.x = 7
6  p1.y = -3
7  ...
8
9  # Python objekti su dinamički (možemo dodavati nove
10 # attribute bilo kada!)
    p1.ime = "Petar"
```

Metodi objekta

...

```
def imeMetoda(self, parametar, ..., parametar):
```

- `self` mora biti prvi parametar bilo kojeg metoda objekta
 - predstavlja "implicitni parametar"
- Atributima objekta se uvijek pristupa referencom na `self`

```
class Tacka:  
    def pomjeri(self, dx, dy):  
        self.x += dx  
        self.y += dy  
    ...
```

ZA VJEŽBU: Dodati metode koji računaju rastojanje između dvije tačke, postavljaju koordinate i računaju rastojanje od koordinatnog početka!

Pozivanje metoda

- Klijent može pozivati metode objekta na dva načina:
 1. `objekat.metod(parametri)`
 2. `Klasa.metod(objekat, parametri)`
- Primjeri:
 - `p = Tacka(3, -4)`
 - `p.pomjeri(1, 5)`
 - `Tacka.pomjeri(p, 1, 5)`

Konstruktori

```
def __init__(self, parametar, ..., parametar):  
    izrazi
```

- **Konstruktor** je specijalni metod naziva `__init__`
- Primjer:

```
class Tacka:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
    ...
```

Kako kreirati objekat klase Tacka bez prosleđivanja koordinata?

Dodatna dokumentacija

- <http://python.org/>
 - dokumentacija, tutorijali ...
- Knjige:
 - *Learning Python*, Mark Lutz
 - *Python Essential Reference*, David Beazley
 - *Python Cookbook*, Martelli, Ravenscroft and Ascher
 - (online <http://code.activestate.com/recipes/langs/python/>)
 - <http://wiki.python.org/moin/PythonBooks>

