Uvod u programski jezik Pajton - 4. deo

Uvod u merno-informacione sisteme

Izuzeci

- Prilikom generisanja greške u kodu se generiše (ispaljuje) izuzetak (*exception*)
- Kako bi se rukovalo izuzecima neophodno je koristiti blok *try - except*
- Nakon ključne reči *except* navodi se izuzetak koji se želi uhvatiti

```
1  a=0
2  b=5
3
4  try:
5      c=b/a
6  except ZeroDivisionError:
7      print('Nemoguce je deliti nulom')
```

- U slučaju da postoji više izuzetaka moguće je napraviti više *except* blokova
- Samo će se ispisati poruka za prvi generisani izuzetak
- Na kraju se može navesti *except* blok bez oznake izuzetka i on se izvršava samo u slučaju da nabrojani izuzetak ne postoji

```
1  try:
2      a = (1,2,3,4)
3      b = 0
4      print(a[6] / x)
5  except IndexError:
6      print('Nepostojeci indeks')
7  except ZeroDivisionError:
8      print('Nije moguce deljenje nulom')
9  except:
10     print('Nepoznata greska')
```

- U slučaju da se želi izvršiti kod bez obzira da li je došlo do greške ili ne, moguće ga je dodati u *finally* blok

```
1 f = open("file.txt")
2 try:
3     f.write("tekst")
4 except:
5     print("Greska")
6 finally:
7     f.close()
```

- U slučaju da se želi izvršiti kod samo u slučaju da nije došlo do greške, moguće ga je dodati u *else* blok

```
1 a = 2
2 b = 4
3 try:
4     print(f'Rezultat deljenja je {a/b}')
5 except:
6     print('Doslo je do greske prilikom deljenja')
7 else:
8     print('Uspesno deljenje')
```

- Blokovi *try*, *except*, *else* i *finally* se mogu kombinovati (važan je poredak naredbi)

```
1  a = 2
2  b = 0
3  try:
4      print(f'Rezultat deljenja je {a/b}')
5  except:
6      print('Doslo je do greske prilikom deljenja')
7  else:
8      print('Uspesno deljenje')
9  finally:
10     print('Uvek se izvrsava')
```

- Ako se želi ispisati poruka o izuzetku moguće je uhvatiti izvor izuzetka i korišćenjem funkcije print prikazati ga u konzoli

```
1 def prosledi_broj(br):
2     if type(br) is int:
3         return br*42
4     else:
5         raise TypeError('Mora biti broj')
6 try:
7     print(prosledi_broj('a'))
8 except Exception as e:
9     print(e)
```

Funkcija *map*

- Funkcija *map* vraća iterator pošto primeni funkciju na svaki od elemenata prosleđenog iteratora (liste, torke itd.)

```
1  def provera_parnosti(broj):
2      if broj % 2 == 0:
3          return 'Paran'
4      else:
5          return 'Neparan'
6
7  lista = [17, 2, 23, 43, 57]
8
9  rezultat = list(map(provera_parnosti, lista))
10
11 print(rezultat)
12 #['Neparan', 'Paran', 'Neparan', 'Neparan', 'Neparan']
```

Funkcije *any* i *all*

- Funkcije *any* i *all* se koriste za proveru da li su u nekom iteratoru (torci ili listi) barem jedan ili svi *True* elementi

```
1  a = [True, True, False, True]
2  b = [True, True, True, True]
3  c = [False, False, False, False]
4
5  print(f'Any: {any(a)}, all: {all(a)}')
6  #Any: True, all: False
7
8  print(f'Any: {any(b)}, all: {all(b)}')
9  #Any: True, all: True
10
11 print(f'Any: {any(c)}, all: {all(c)}')
12 #Any: False, all: False
```

Funkcija *filter*

- Funkciji *filter* neophodno je proslediti funkciju koja specificira pravilo po kom se vrši filtriranje, kao i iterator koji predstavlja podatke koji će se filtrirati
- Povratna vrednost je novi niz koji sadrži filtrirane podatke

```
1  godine = [4, 22, 13, 45]
2
3  def provera(x):
4    if x < 18:
5      return False
6    else:
7      return True
8
9  punoletni = filter(provera, godine)
10
11 for i in punoletni:
12   print(i)
```

Lambda

- Lambda predstavlja funkciju koja se definiše u jednoj liniji
- Može primiti bilo koji broj parametara, ali može imati samo jedan izraz
- Nakon računanja izraza dobija se povratna vrednost kao izlaz lambda funkcije

```
1 sum = lambda a, b : a + b
2 print(sum(11, 12))
```

List comprehension

- *List comprehension* predstavlja mehanizam za realizaciju liste koja se pravi po jasno definisanom pravilu

```
1 lista = [i*2 for i in range(10)]
2 print(lista)
3 #[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```
1 lista = [i for i in 'Zdravo']
2 print(lista)
3 #['Z', 'd', 'r', 'a', 'v', 'o']
```

Moduli

- Moguće je realizovati i sopstveni modul, neophodno je da ima ekstenziju .py
- Neka je realizovan modul zbir.py sa funkcijom suma koja računa zbir dva prosleđena broja

```
1 #modul zbir.py
2 def suma(a,b):
3     return a+b
```

- U glavnom programu (modul koji se pokreće) uvozi se funkcija suma iz modula zbir

```
1 #modul glavnog programa
2 from zbir import suma
3
4 print(suma(3,5))
```

Bitski operatori

- Bitski (*bitwise*) operatori predstavljaju operatore za manipulaciju na nivou bita
- Ako se želi dobiti heksadecimalna reprezentacija nekog broja moguće je korsititi funkciju *hex*

```
1 print(hex(3643))#0xe3b
```

- Ako se želi dobiti binarna reprezentacija broja moguće je koristiti funkciju *bin*

```
1 print(bin(3643))#0b111000111011
```

Operator	Opis
A&B	Bitska I operacija
A|B	Bitska ILI operacija
A^B	Bitska ekskluzivno ili operacija
~A	Bitska negacija
A<<n	Levo pomeranje za n mesta
A>>n	Desno pomeranje za n mesta

```
1  a=10#0b01010
2  b=22#0b10110
3
4  print(a&b)#2=0b10
5  print(a|b)#30=0b11110
6  print(a^b)#28=0b11100
7  print(~a)#-11=0b10101 (2c=~num+1)
8  print(b<<1)#44=0b101100
9  print(a>>2)#2=0b10
```

- Negativni brojevi se čuvaju u komplementu dvojke
  - Počinje se sa istim pozitivnim brojem
  - Negacija svih bitova broja (prvi komplement)
  - Na rezultat se dodaje jedan

Matrice

- Pajton nema ugrađeni tip za rukovanje matricama
- Matrica se realizuje pomoću ugnježdavanjem (lista u listi)

```
1 m = [[1, 2, 3],
2      [-1, 2, 4],
3      [5, 7, 11]]
```

- Pošto se matrica sastoji iz više listi može se desiti da broj kolona u svakom redu nije isti
- Dodatni problem je što se svaka manipulacija mora realizovati od početka jer matrica nije ugrađeni tip

```
1 m = [[1, 2, 3],
2      [-1, 2, 4],
3      [5, 7, 11]]
4
5 for i in m:
6     for j in i:
7         print(j)
```

- Rad sa ovako realizovanim matricama nije jednostavan
- Kao rešenje korsiti se biblioteka *numpy*

numpy biblioteka

- Instalacija pozivom *pip install numpy*
- Biblioteka za rad sa numeričkim tipovima podataka
- Numpy se može koristiti za rad sa višedimenzionalnim nizovima (*ndarray*)
- Svakom elementu se može pristupiti navođenjem indeksa u uglastim zagradama (M[2][0])

```
1 import numpy as np
2
3 #definisanje niza
4 M = np.array([[3, 1.2, 11.2], [3.6, 6.6, 1.1]])
5 M = np.zeros((2, 3))# upisuje sve 0
6 M = np.ones((2, 3))# upisuje sve 1
7 M = np.arange(6)# generise redom brojeve
8 M = np.arange(6).reshape(2, 3)# generise matricu
```

numpy biblioteka - Aritmetika

```
1 import numpy as np
2 m1=np.zeros((3,2)) #popunjava matricu 3x3 nulama
3 m1[1,1]=3
4 print(m1)
5 m2=np.matrix([[1,2,3],[4,5,6]])
6 print(m2)
7 print(m2.transpose())#transponuje matricu
8 print(np.dot(m1,m2))#mnozi matrice
```

- Za više informacija: *numpy.org*

Hvala na pažnji!