VEŽBA 6

Pregled

Kao i mnogi drugi programski jezici i Python podržava razbijanje programa u sitnije delove. Podela najnižeg nivoa je podela na funkcije. Posebna vrsta funkcije koja ima sposobnost da generiše sekvencu (npr. listu) naziva se *generatorom*.

Definicija funkcije

Funkcija se u Pythonu definiše na sledeći način:

```
def ime_funkcije(lista argumenata):
     blok_tela_funkcije
```

Vraćanje vrednosti iz funkcije

Svaka funkcija može da vrati rezultat u vidu povratne vrednosti. Povratak iz funkcije i vraćanje vrednosti postiže se naredbom *return*. U Pythonu povratna vrednost može da bude i n-torka. U tom slučaju naredba *return* ima više parametara, a zagrade za obeležavanje n-torke nisu neophodne.

```
return x            # vraca jednu vrednost
return a, b, c      # vraca 3 vrednosti (uredjenu trojku)
```

Generatorska funkcija

Generatorska funkcija je po svemu identična običnoj funkciji s tom razlikom da u sebi sadrži naredbu *yield*. Ova funkcija privremeno obustavlja izvršenje funkcije da bi ona proizvela element sekvence. Za vremen davanja elementa funkcija zadržava svoje kompletno stanje. Rad generatorske funkcije se završava izvršenjem naredbe *return* ili dostizanjem kraja funkcije.

```
def nas_range(n):
  '''Ova funkcija generise sekvencu od 0 do (n-1)'''
  i=0
  while i<n:
    yield i
    i+=1
```

Presretanje grešaka u Python-u

U programiranju se javljaju različiti tipovi grešaka. Sintaksne greške predstavljaju situaciju u kojoj prevodilac ili interpreter za određeni jezik ne mogu da protumače određene delove teksta programa jer nisu u skladu sa određenim pravilima tog jezika. Ovakve greške je neophodno otkloniti pre nego što će program moći da bude preveden ili izvršen i taj posao je u izvesnoj meri olakšan porukama prevodioca ili interpretera (daje se razlog greške i linija u kojoj se nalazi).

Drugi tip greške posledica je fizičke nemogućnosti da se neka operacija obavi. Na primer: promenljiva kojoj se pristupa nije definisana, pristupa se elementu liste koji je van opsega liste, pokušava se deljenje sa nulom ili se zahteva otvaranje fajla koji ne postoji. Ovakve greške se ne mogu utvrditi ili predvideti pre pokretanja programa jer one nastaju u toku rada programa – engl.

Runtime error. Onog trenutka kada se dese, interpreter ili operativni sistem daju poruku o grešci koja više ili manje tačno ukazuje na njen uzrok. Da bi se izbeglo njihovo pojavljivanje u toku rada programa postoje dve mogućnosti:

i. Ova mogućnost je nezavisna od programskog jezika i podrazumeva pisanje delova programa koji proveravaju postoji li mogućnost da se kritični deo programa uspešno završi. Na primer, u skladu sa prethodnim primerom: pre pristupa promenljivoj proverava se njena definisanost, pre pristupa elementu liste proverava se dužina liste, pre deljenja proveri se da li je delitelj različit od nule i pre otvaranja falja proverava se njegovo postojanje. Ova rešenja su korektna, međutim ona čine program teže čitljivim jer se u njega unose elementi koji se ne tiču algoritma rada, nego služe isključivo za sprečavanje pojave grešaka u toku rada.

ii. U nekim programskim jezicima (Python spada u takve jezike) postoji mogućnost „*hvatanja*“ grešaka. Nakon što se greška desi, ona može biti uhvaćena, nakon čega će se izvršiti deo programa koji će adekvatno reagovati na tu grešku. Taj deo takođe piše autor programa. U nastavku sledi opis hvatanja grešaka u Python-u.

```
try:
  blok programa u kom se greske hvataju
except klasa_greske_1:
  blok za reagovanje na greške tipa klasa_greske_1
except klasa_greske_2:
  blok za reagovanje na greške tipa klasa_greske_2
except:
  blok za reagovanje na sve ostale greške
```

Blok programa u kojem će greške, ukoliko se dese, biti uhvaćene ograničen je na blok iza ključne reči `try`. Svaka greška koja se može desiti spada u određenu klasu. Na primer, greška deljenja sa nulom pripada klasi `ZeroDivisionError`. Obrada, odnosno reakcija na grešku sledi u jednom od blokova navedenih iza ključne reči `except`. Ako se desi greška određene klase, izvršavanje programa će se nastaviti od odgovarajućeg except bloka. Poslednji (ili jedini) blok može biti naveden bez klase greške i on će biti izvršen ako greška ne spada ni u jednu klasu definisanu prethodnim except blokovima.

Ovakvi blokovi se mogu naći u svim delovima programa, što uključuje i funkcije svih nivoa, kao i module i metode objekata. Blokovi mogu biti i ugnežđeni, tj. jedan blok može sadržati druge blokove. Ako neki blok ne hvata grešku određenog tipa, ona se prosleđuje bloku koji taj blok okružuje itd. Najširi (podrazumevani) blok je ceo program, a greške koje do tog nivoa stignu zaustavljaju program i objavljuju se kao greške.

Zadaci

1. Napraviti funkciju koja obavlja sortiranje liste bez korišćenja dodatne liste, promenama na listi koja je prvi argument funkcije. Algoritam treba da bude *selection sort*. Kriterujum sortiranja treba zadati funkcijom koja se prenosi kao drugi argument.

   Ovaj algoritam je implementiran u prethodnoj vežbi sa tom razlikom što je tamo stvarana nova lista, a stara je svođena na nultu dužinu stalnim vađenjem najmanjeg elementa iz nje. Kao pomoć, funkcionalna verzija ovakve funkcije data je u dodatku na kraju ove vežbe zajedno sa kriterijumskom funkcijom.

Razlika u implementaciji treba da bude to što umesto izbacivanja najmanjeg elementa i njegovog dodavanja na početak nove liste u ovom slučaju sve potrebne izmene obavljamo zamenom mesta pojedinačnim elementima. Algoritam u pseudokodu dat je na kraju ovog teksta.

2. Na isti način kao u prethodnoj funkciji implementirati *bubble sort*. Kao i u prethodnom zadatku u rešenju ne treba koristiti dodatnu listu nego sve izmene uraditi na listi koja se prosleđuje kao parametar funkcije.
3. Napisati generatorsku funkciju koja vraća prvih n elemanata Fibonačijevog niza. Testirati pomoću `for` petlje.
4. Napisati generatorsku funkciju koja vraća prvih n elemanata Fibonačijevog niza u uređenim parovima. Uređeni parovi treba da predstavljaju *i*-ti i *(i+1)*-ti element niza. Nakon testiranja pomoću for petlje dopuniti program tako da izračunava količnik *(i+1)*-tog i *i*-tog elementa niza za svaki par od 0 do *n*. *n* zadaje korisnik. Presretanjem greške klase ZeroDivisionError ispisivati znak beskonačno – `'oo'` u slučaju deljenja sa nulom. Koristiti funkciju `float()` za konverziju celih brojeva u pokretni zarez za forsiranje deljenja u pokretnom zarezu.

Dodaci

Funkcija koja implementira selction sort korišćenjem nove liste

Sledeća funkcija implementirana je u *Python*-u:

```
def sort(ls, kt):
  """ Sortira listu zadatu kao prvi argument,
  rezultat vraca kao novu sortiranu listu,
  kriterijum sortiranja zadaje se kao funkcija"""
  ls = ls[:]   # pravi tzv. duboku kopiju stare liste,
               # inace bi ona bila ispraznjena
  dl = len(ls)
  xdl = dl
  i = 0
  nls = []
  while i<dl:
    me = ls[0]
    xx = 0
    j = 1
    while j<xdl:
      if kt(ls[j], me):
        me = ls[j]
        xx = j
      j += 1
    xdl -= 1
    nls.append(ls.pop(xx))
    i += 1
  return nls
```

Kriterijumska funkcija

Kao i prethodna, i ova funkcija je implementirara u *Python*-u.

```
def krit(a, b):
  """ Funkcija koja zadaje kriterijum neopadajuce liste"""
  return a<b
```

Selection sort algoritam implementiran zamenom mesta elementima

Sledeća funkcija data je u *pseudo-kodu*. On svojom formom najviše podseća na jezik C, za uslove i petlje čak su i korišćene ključne reči iz tog jezika.

```
// SELECTION-SORT ALGORITAM BEZ KREIRANJA NOVE LISTE
dl = dužina_list(lista)
i = 0
while ( i < (dl-1) )
{
  me = ls[i]
  xx = i
  k = i + 1
  while ( k < dl )
  {
    if ( ls[k] |*|[1] me)
    {
      me = ls[k]
      xx = k
    }
    k = k + 1
  }
  if (i != xx)
  {
    ls[xx] = ls[i]
    ls[i] = me
  }
  i = i + 1
}
```

Bubble sort algoritam – objašnjenje

Kod ovog algoritma sortiranja, kriterijum sortiranja se primenjuje na parove susednih elemenata u listi. Kroz listu se u petlji prolazi od početka do kraja liste, a ako se utvrdi da je jedan par susednih elemenata u pogrešnom redosledu, zamene im se mesta. Kroz listu se prolazi u pomenutom redosledu sve dok se ne konstatuje da je broj zamena mesta bio jednak nuli – to znači da je lista ispravno sortirana u skladu sa kriterijumom. Sledi primer za sortiranje niza u neopadajući redosled. Kriterijum sortiranja je provera *da li je element na višoj poziciji manji od onog na nižoj*.

1 Znak |*| označava proizvoljnu operaciju poređenja na osnovu koje će se lista sortirati. Za sortiranje u neopadajućem redosledu ovu oznaku treba zameniti znakom <

2	5	8	6	3	7	polazni niz, 1. prolazak, nađen par
2	5	6	8	3	7	1. prolazak, nađen par
2	5	6	3	8	7	1. prolazak, nađen par, kraj prolaza
2	5	6	3	7	8	2. prolazak, nađen par, kraj prolaza
2	5	3	6	7	8	3. prolazak, nađen par, kraj prolaza
2	3	5	6	7	8	sortiran niz

Kraj algoritma je dostignut kada u poslednjem prolasku kroz listu nije izvršena nijedna zamena mesta. Kada se poslednja vrsta u tabeli pogleda, može se konstatovati da nijedan od susednih parova nije u obrnutom redosledu što znači da je lista sortirana.

Fibonačijev niz

Za definiciju Fibonačijevog niza uzeti:

$$F_i = F_{i-1} + F_{i-2}$$

$$F_0 = 0;\ F_1 = 1$$

Primer: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Količnik $\frac{F_i}{F_{i-1}}$ konvergira vrednosti tzv. „zlatnog preseka“.