VEŽBA 5

Pregled

Python je savremeni programski jezik koji danas ima široku primenu u mnogim oblastima. Velika prednost je što programi napisani u ovom programskom jeziku ne zahtevaju prevođenje, nego se mogu izvršavati neposredno iz oblika programskog teksta – izvornog koda (engl. *source code*). Programi pisani na ovaj način se često nazivaju **skriptovima**, pa ni Python nije izuzetak.

Pokretanje programa pisanih u jeziku Python

U okruženju operativnog sistema Linux, Python programi se najčešće pokreću iz shell-a, slično shell skriptovima. Mogu biti pokretani i kao desktop aplikacije ili kao ekstenzije drugih softverskih sistema, o čemu će biti više reči u narednim vežbama.

Pokretanje eksplicitnim navođenjem interpretera

Za izvršavanje programa napisanog u jeziku Python neophodno je odgovarajuće okruženje (interpreter), a ono se može pozvati komandom `python3`. Nakon komande, prvi parametar treba da bude ime fajla koji sadrži program – najčešće ima ekstenziju `.py`. Sledeći primer pokreće program `test.py`.

```
$ python3 test.py
```

Pokretanje Python programa kao komande

Shell je u stanju da zaključi koji je interpreter potreban za izvršenje nekog fajla kao komande ako se na sam početak teksta (u prvi red) stavi komanda oblika

```
#!/usr/bin/python3
```

Znak # u većini skript jezika – pa i u Python-u – označava **komentar**, zbog čega će ovaj red biti zanemaren od strane Python okruženja. Kao i u slučaju shell skriptova, fajlu sa programom se mora dodeliti pravo izvršavanja (komandom `chmod a+x test.py`). Ako su oba navedena uslova ispunjena, program se može pokrenuti kao

```
$ ./test.py
```

Promenljive

Kao i u mnogim drugim programskim jezicima, promenljive su ključne jer omogućuju pristup podacima. Operacija dodele je ključna u ovom procesu i obavlja se korišćenjem operacije =. Npr.

```
x = 10
y = 200.24
z = 'tekstualni podatak'
```

Osobenost programskog jezika Python u odnosu na tradicionalne programske jezike (poput jezika C) je da se tip promenljivih određuje **dinamički**. To znači da se ne određuje unapred (i trajno) koja

je promenljiva kog tipa, nego to zavisi od konkretne situacije, operacija koje su prethodno izvršene ili podataka koji su u datom trenutku na raspolaganju.

Može se zamisliti da je veza između imena promenljive i konkretnog podatka (konkretnog tipa, videti sledeći odeljak) u memoriji računara privremena. Kao da u nekom intervalu određeno **ime promenljive** ukazuje na **određeni podatak**, a ta veza se **ostvaruje** operacijom dodele. Sledećom operacijom dodele ista ta promenljiva može da pokazuje na sasvim drugi podatak, čak i drugačijeg tipa.

Na imena promenljivih nameću se uobičajena ograničenja, pa mogu biti proizvoljne dužine, ne smeju sadržati specijalne simbole (osim donje crte) i razmake i ne smeju počinjati brojem.

Osnovni tipovi podataka

U jeziku Python postoje sledeći bazični tipovi podataka:

- `int` – celobrojni tip podatka, brojevi takođe mogu da budu i negativni
- `float` – broj u pokretnom zarezu
- `bool` – logički tip podatka, vrednost ograničena na tačno (`True`) i netačno (`False`)
- `str` – string, proizvoljni niz karaktera

Konverzija između tipova podataka

Ako u nekom aritmetičkom izrazu učestvuje više različitih osnovnih tipova podataka, doći će do automatske konverzije između tipova `int` i `float` tako da se operacije izvrše zadržavajući tačnost koliko god je to moguće.

Eksplicitna konverzija je takođe moguća (a često je i neophodna) primenom imena navedenih tipova kao funkcija. Npr. ovakva konverzija je obavezna ukoliko postoji potreba da se numerički podatak unet preko terminala pretvori u broj sa kojim se dalje obavljaju aritmetičke operacije:

```
x = '865'    # broj 865 je zadak kao string
r = int(x) * 10  # pomnozice se 865 sa 10
s = str(r)   # ovo ce rezultat konvertovati u string
```

Aritmetičke operacije

Na numeričke vrednost se mogu primenjivati aritmetičke operacije sabiranja (+), oduzimanja (-), množenja (*) i deljenja (/).

Sve operacije imaju svoje odgovarajuće prioritete, a navođenjem zagrada u izrazima prioritet se može nametnuti eksplicitno na uobičajeni način (prvo se izračunava izraz unutar zagrade).

```
r = 10 * r
p = 8.3 * (r + 9)  # zagrade odredjuju prioritet
```

Ovo se odnosi na numeričke tipove (int i float). Primena na druge vrste podataka može rezultovati greškom ili rezultatom koji nije očekivan. Npr. primena operacije + na stringove obavlja njihovo spajanje.

Kumulativne operacije

Sa desne strane operacije dodele se može pojaviti i promenljiva kojoj se dodeljuje vrednost. Ovakvi izrazi se elegantnije (i jasnije) mogu zapisati primenom kumulativnih operacija, koje se mogu primeniti na sve aritmetičke operacije:

```
k = k + 1
k += 1     # cistiji i jasniji oblik prethodne operacije
# -------
m = m * (k * 2 - 100)
m *= (k * 2 - 100)  # kumulativni oblik prethodne operacije
```

Korišćenje sistema ulaza/izlaza prema terminalu

izlaz prema terminalu – ispis

Za ispis na terminalu koristi se funkcija iz standardne biblioteke `print`. Optimalno formatirani ispis se obavlja automatski za sve tipove podataka bez njihovog posebnog obeležavanja. Jednom naredbom se može ispisati i veći broj podataka i oni se tada odvajaju *zarezom*. Prelazak u novi red se obavlja nakon što je i poslednji podatak (parametar funkcije) ispisan.

```
print('Broj je:', x)   # prelazak u novi red se podrazumeva
print('Unesite broj: ', end='')  # bez prelaska u novi red
```

ulaz sa terminala – unos

Ulaz sa terminala obavlja se ugrađenom funkcijom `input`. Funkcija uvek vraća string čiji je sadržaj celokupan red (do prelaska u novi red, uključujući prazna mesta, zareze, itd.) koje je korisnik uneo.

```
r = input()    # unos celokupnog reda teksta sa std. ulaza
pr = input('Unesi podatak: ')  # moguc je i ispis poruke
```

Vraćeni podatak je string, što ga čini nepogodnim kao numerički podatak.

Rad sa stringovima

String je jedan od osnovnih tipova podataka u jeziku Python i služi za skladištenje nizova karaktera, odnosno teksta. Jednom kreiran string se više ne može menjati i zbog toga se svrstava u *nepromenljive* tipove podataka (*immutable*).

Kreiranje stringova

Navođenjem teksta između jednostrukih ili dvostrukih navodnika definiše se string. Ova dva tipa navodnika potpuno su ravnopravna.

```
s='Oba tipa navodnika su "ravnopravna"'
p="Drugi tip moze se koristiti u stringu: pyton's challenge"
```

Određivanje dužine stringa

Funkcija `len` čiji je parametar string vratiće celi broj čija vrednost odgovara dužini stringa.

```
duz = len(s)
```

Pristup elementima i delovima stringa

Operator `[start:stop:step]` primenjen na string daje novi string sastavljen od elemenata stringa na koji je operator primenjen. Svi elementi stringa su indeksirani rednim brojem, a prvi element ima indeks `0`. Negativni indeksi broje karaktere od kraja stringa i indeks `-1` se odnosi na poslednji karakter sitringa. U gorenavedenom formatu `start` se odnosi na prvu poziciju koja će biti uključena u rezultat, `stop` na prvu poziciju koja neće biti uključena u rezultat, a `step` na korak kojim će biti uzimani elementi iz navedenog opsega. Ako se neki od parametara ne navede koristiće se podrazumevane vrednosti a to su prvi karakter za početak, poslednji za kraj, a `1` za korak.

```
s='0123456789'
s[3:6]    # '345'
s[:6:2]   # '024'
s[::3]    # '0369'
```

Nadovezivanje stringova

Rezultat ovog „sabiranja“ stringova je novi string dobijen nadovezivanjem stringova koji u ovoj operaciji učestvuju.

```
a='prvi'+' drugi' # novi string: 'prvi drugi'
m='a'+'b'+'c' # novi string: 'abc'
m+=a # novi string: 'abcprvi drugi'
```

Operacije poređenja

U programima se redovno javlja potreba da se upoređuju podaci. Rezultat ovih operacija je uvek logičkog tipa (tip `bool`). Operacije poređenja se primenjuju na dva operanda između kojih se nalaze (npr. `a < b`) i u jeziku Python one su sledeće:

- < leva strana je manja od desne
- > leva strana je veća od desne
- <= leva strana je manja od desne ili joj je jednaka
- >= leva strana je veća od desne ili joj je jednaka
- == leva strana je jednaka desnoj
- != leva strana je različita od desne

U slučaju operacija poređenja, važi ista napomena kao i za aritmetičke operacije. Ako se porede numerički podaci, rezultati poređenja će biti tačni sa stanovišta matematike. Primena na druge tipove podataka prouzrokovaće ili grešku ili neočekivani rezultat.

Upravljačke strukture `if` i `while` jezika Python

Provera uslova i petlje suštinski su deo svakog programskog jezika, pa i Python-a. U odnosu na druge jezike Python poseduje neke osobenosti.

Provera uslova sa if

Struktura `if-else` kao i kod drugih imperativnih jezika služi za odlučivanja u toku programa. U Python-u ima sledeću formu:

```
if uslov_1:
  blok_1
elif uslov_2:
  blok_2
else:
  blok_3
```

Uslovi navedeni uz `if` i `elif` su bilo koji izraz koji se izračunava u neku vrednost. Smatra se tačnim u svim slučajevima osim kada je ta vrednost: `False`, `0`, `0.0`, `''`. Poslednji element je prazan string. Isto se ponašaju i druge prazne strukture podataka, ali one ovde neće biti razmatrane.

Blok 1 se izvršava ako je uslov 1 zadovoljen, blok 2 ako uslov 1 nije, ali uslov 2 jeste, a blok 3 ako nijedan od prethodnih uslova nije zadovoljen. `elif` i `else` segmenti su opcioni, dok `elif` segmenata može biti i više od jednog ako za tim postoji potreba.

Python je osoben po tome što se blokovi raspoznaju i istovremeno i definišu nivoom uvučenosti od desne ivice – **indentacijom**. Posebna oznaka za početak i kraj bloka ne postoji. Na ovaj način programer je nateran da ispravno formatira svoj program.

Petlje tipa while

I `while` petlja je uobičajena konstrukcija u programskim jezicima. U Python-u ima sledeći oblik:

```
while uslov:
  blok_1
else:
  blok_2
```

Uslov se ponaša identično uslovu u `if` konstrukciji. Dok je ispunjet telo petlje označeno sa `blok_1` se izvršava. Prvi put kada uslov nije tačan izvršava se `blok_2`, i to samo jednom.

Prekidanje i vraćanje na početak petlji

Svaka petlja u Python-u može da se prekine naredbon `break`, ili da se vrati na mesto provere uslova naredbom `continue`. Ovim blok `else` iz prethodnog odeljka dobija nešto više smisla jer izlazak iz petlje pomoću `break` otvara mogućnost prekidanja `while` petlje, a da je sam uslov petlje pri tome ispunjen. `break` i `continue` se skoro uvek koriste zajedno sa uslovom tipa `if`.

Zadaci

1. Napisati program i snimiti ga pod imenom `hello.py`. On treba da ispiše string *„Hello World!“*. Nakon što uspešno proradi, izmeniti program tako da u petlji 10 puta redom ispiše navedeni string.
2. Napisati program koji omogućava unos proizvoljnog stringa. Nakon unosa, program ispisuje uneti string `K` puta. `K` zavisi od dužine stringa: jednako je dužini stringa ako je ona manja od

10, a odgovara rezultatu celobrojnog deljenja dužine stringa sa 10 ako je ona veća ili jednaka 10.

3. Napisati program koji omogućava sledeće: korisnik preko tastature unosi broj `x`, a odmah potom računar izračunava vrednost `3 * x + 5` i ispisuje je u sledećem redu na ekranu. Nakon toga očekuje se unos sledećeg broja. Postupak se ponavlja sve dok korisnik ne unese vrednost 0.

   Primer:

   ```
   Unesite broj X : 5
   (3 * X) + 5  je : 20
   Unesite broj X : 7
   (3 * X) + 5  je : 26
   Unesite broj X : 0
   Kraj!
   ```

   Napomena: Korisnik je u mogućnosti da unese čak i string koji ne može da se protumači kao broj, što može izazvati neispravan rad programa. U rešenju nije neophodno obezbediti rešenje za taj problem.

4. Napisati program u Python-u koji omogućava korisniku da unese proizvoljan string preko terminala, a onda će taj string biti ispisan na terminalu:

   a) u izvornom obliku

   b) karakter po karakter

   c) karakter po karakter u obrnutom redosledu

   d) od njega će biti kreiran string u obrnutom redosledu karaktera i ispisan u celini.