Rad sa petljama

Generalno govoreći, naredbe se u Pythonu izvršavaju sekvencijalno, što znači da se prva linija koda izvršava prva, nakon nje druga i tako dalje. Postoje situacije kada želimo da blok koda izvršimo n puta. U tim slučajevima možemo korisiti neku od petlji koje Python podržava:

- while petlja;
- for petlja.

Funkcija range()

U Pythonu, funkcija range() se često koristi u radu sa petljama. Podržava tri vrste prosleđivanja argumenata i vraća nazad u program objekat tipa generator (više o tome u narednim lekcijama). Naredbom list(range(n)) prebacujemo generator u nama već poznat tip podatka – listu.

Ako joj se prosledi samo jedan broj, vratiće listu sa toliko elemenata u sebi. Ako joj se proslede dva argumenta, vratiće u program sekvencu koja počinje prvim argumentom i završava se drugim argumentom ali drugi element ne ulazi u izbor. Tri broja prosleđena kao argumenti funkciji range imaju sličnu funkcionalnost kao kada je pozivamo sa dva argumenta, s tim što je treći argument u ovom slučaju zapravo interval. Funkcija range() prima samo cele brojeve kao parametre. Sledeći primer to pojašnjava:

Primer funkcije Range()

```
print(list(range(5)))
print(list(range(5,10)))
print(list(range(5,10,2)))
```

Objašnjenje:

Prvi ispis će biti [0, 1, 2, 3, 4], drugi [5, 6, 7, 8, 9] a treći [5, 7, 9].

Takođe, u trećem slučaju kao vrednost intervala se može proslediti negativan broj.

While petlja

While petlja se u Pythonu koristi za ponavljanje bloka naredbi dokle god je prosleđeni uslov True.

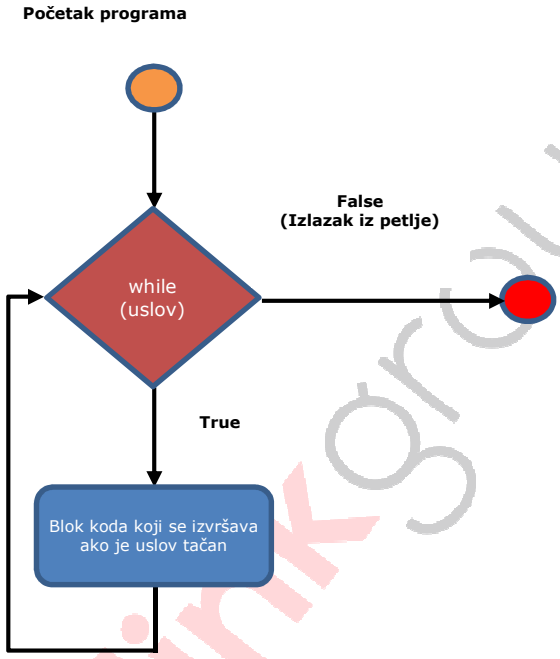
Primer sintakse while petlje

```
while (uslov ili ekspresija):
    blok koda koji treba da se izvrši.

naredba koja će se izvršiti nakon završetka while petlje
```

Glavna programska petlja će prvo proveriti zadati uslov u while petlji. Ako je uslov True, petlja će nastaviti sa izvršavanjem naredbi unutar while petlje. Kada taj zadati uslov postane False, glavna programska petlja će izaći iz petlje i nastaviti sa izvršavanjem ostalih naredbi u programu/skripti.

Grafički prikaz toka programa kada glavna programska petlja naiđe na while petlju je prikazan na sledećoj ilustraciji:



Slika 10.1. Prikaz toka programa u while petlji

Primer upotrebe while petlje:

Radno okruženje

```
limit = 5
counter = 1
while (counter < limit):
    print(counter)
    counter += 1

print(counter)
```

Objašnjenje:

Program će ispisivati brojeve od 1 do 5 odn. sve dok brojač ne bude bio veći od limita. Kada se desi da ovaj uslov nije tačan petlja staje sa radom. Na kraju ispisujemo vrednost brojača posle petlje, a to je 5.

Analiza

Definisali smo dve promenljive, limit i counter, sa inicijalnim vrednostima 5 i 1. Izraz `counter < limit` će biti tačan za prvo izvršavanje ovog bloka koda. U tom prvom izvršavanju, ispisaće se vrednost promenljive counter i vrednost te promenljive će se povećati za 1. Na početku četvrtog izvršavanja, promenljiva broj će imati vrednost 4. Pošto je za nju uslov tačan, ispisaće se njena vrednost i odmah nakon toga, ta vrednost će se povećati za 1. Kako je u sledećem izvršavanju vrednost 5, uslov `counter < limit` će postati netačan i program će nastaviti dalje sa izvršavanjem naredbi izvan ove petlje.

Beskonačna petlja

Treba biti pažljiv pri korišćenju while petlje. Da smo u prethodnom primeru nakon svakog narednog prolaza kroz petlju zaboravili da povećamo vrednost za promenljive 'counter' za jedan, petlja se nikad ne bi završila, jer bi uslov `counter < limit` uvek bio tačan (1 će uvek biti manje od 5) i glavna programska petlja nikad ne bi nastavila sa izvršavanjem sledećih naredbi.

Nekada su beskonačne petlje i korisne. Takvi primeri su rad sa semaforima ili klijent/server arhitekturama.

Primer beskonačne petlje:

```
i=0
while(i<10):
   print(i)
```

Objašnjenje:

Brojač i je na početku 0 i uslov petlje je prošao jer je 0<10, tada se ispisuje brojač i. Problem nastaje jer se brojač ne povećava i uslov će opet biti ispitan sa istom vrednošću 0<10 čime dobijama beskonačnu petlju odn. petlju bez kraja.

While-else petlja

Sintaksa Pythona nam dozvoljava i korišćenje else naredbi, i to po sledećem principu:

- dok su zadati uslov ili ekspresija tačni, while petlja će se izvršavati;
- u slučaju kada zadati uslov ili ekspresija postani netačni, izvršiće se blok koda unutar else naredbe.

Primer sintakse while-else petlje

```
while (uslov ili ekspresija):
    blok koda koji treba da se izvrši.
else:
    blok koda koji treba da se izvrši.
```

Ove petlje se ređe viđaju u praksi.

Napomena

Tokom ovog kursa često ćemo se sretati sa terminom *iteracija*, odnosno *iterirati*. On se najviše odnosi na petlje i sekvence. Naime, iterirati znači kretati se pomoću petlje, više puta, kroz elemente prosleđene kolekcije.

Vežba

Data je promenljiva a postavljena na vrednost 15 i promenljiva b sa početnom vrednošću 1. U while petlji realizovati množenje ova dva broja, ali nakon svake iteracije, promenljivu b povećati za jednu vrednost. U svakoj iteraciji ispisati rezultat kao i operande naredbom print(). While petlju zaustaviti kada promenljiva b dostigne vrednost 5.

Rešenje

```
a = 15
b = 1
while b < 5:
    print(str(a)+ "*"+str(b)+"="+str(a*b))
    b += 1
```

Ispis

```
15*1=15
15*2=30
15*3=45
15*4=60
```

Objašnjenje:

Vežba se može rešiti na više načina. U našem rešenju smo prvo postavili uslov while petlje < 5 (manji od pet). Potom smo postavili da se u svakoj iteraciji ove petlje ispisuje rezultat množenja ova dva broja i odmah nakon toga zadali naredbu da se vrednost promenljive b uveća za 1 i ta novostvorena vrednost ponovo smesti u promenljivu b (operator +=).

For petlja

Ova petlja se dosta razlikuje od for petlji u drugim programskim jezicima, koji proveravaju da li je petlja dostigla određenu zadatu vrednost i, ako nije, povećavaju je u svakoj iteraciji. U Pythonu, for petlja iterira kroz elemente date sekvence ne menjajući joj redosled, što znači da se for petlja može koristiti nad listama, n-torkama, stringovima i generatorima. Sintaksa for petlje izgleda ovako:

```
for element in sekvenca:
    blok koda koji treba da se izvrši.
```

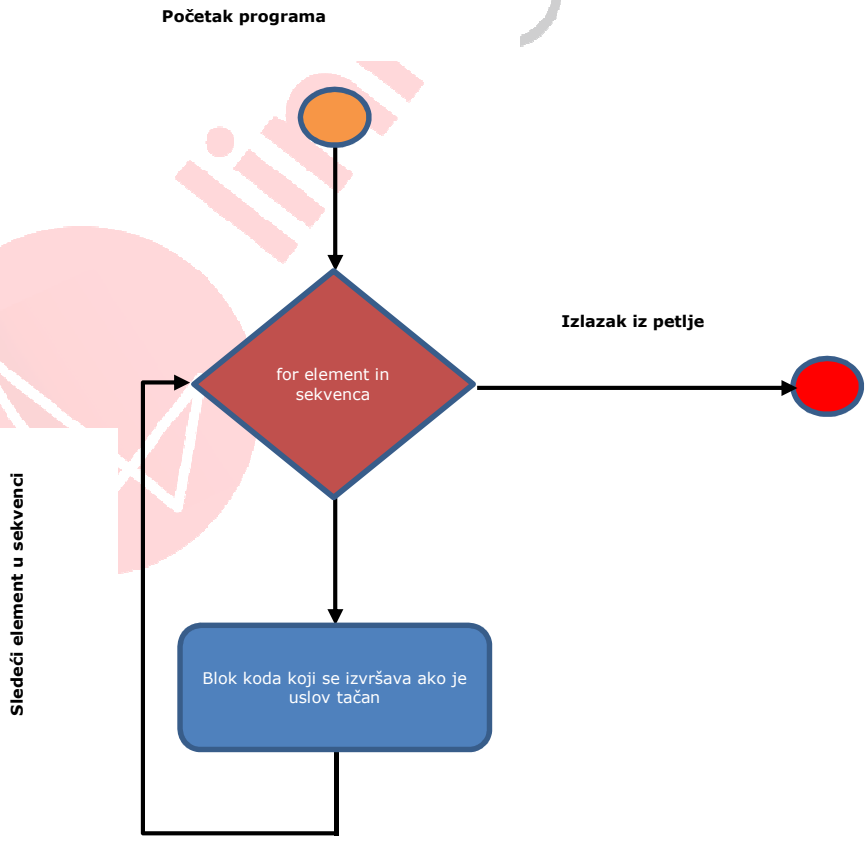
Kao i kod If naredbi, i ovde se mora voditi računa o uvlačenju bloka koda koji treba izvršiti u okviru ove petlje.

Ova petlja funkcioniše na sledeći način:

- Kompajler polazi od sekvenca objekta, što znači da će iterirati kroz njega i prvu vrednost iz date sekvence (prvi element) dodati promenljivoj element. Ako je promenljiva tipa string i to s = 'Python', u prvoj iteraciji će promenljiva element imati vrednost P, u sledećoj iteraciji vrednost y i tako do poslednjeg elementa, koji će biti n.
- Dalje, nakon dodele vrednosti iz sekvence sekvenca promenljivoj element, glavna programska petlja će stupiti u „telo" petlje i izvršavati naredbe koje su deo tog bloka koda.
- Nakon što je blok koda izvršen, glavna programska petlja će se vratiti na početak for naredbe i dodeliti sledeću vrednost iz sekvence promenljivoj element.
- Ovaj proces će se ponavljati dokle god sve vrednosti iz date sekvence nisu prošle, osim ako se petlja ne prekine na drugačiji način.

Dakle, ovde uviđamo slično ponašanje kao i kod If naredbe. Opet je reč o uslovu. Dok god postoje elementi u sekvenci kroz koje je potrebno iterirati, taj uslov će biti True; kada ponestane elemenata za iteraciju u datoj sekvenci – uslov postaje False i program izlazi iz petlje i prelazi na izvršavanje sledećih naredbi.

Grafički prikaz toka programa kada glavna programska petlja naiđe na for petlju je prikazan na sledećoj slici:



Slika 10.2. Prikaz toka programa u for petlji

Upotreba for petlje pomoću range() funkcije kojoj su prosleđeni start i stop parametri (uključuje start parameter, ali generiše sekvencu brojeva do stop parametra):

Primer for petlje uz range() funkciju	
Kod	**Rezultat**

`for nbr in range(0,5):`
`  print("Current nbr is: ", nbr)`
	`Current nbr is:  0`
	`Current nbr is:  1`
	`Current nbr is:  2`
	`Current nbr is:  3`
	`Current nbr is:  4`

Tabela 10.1. For petlja uz range() funkciju

Analiza

- Prva iteracija – for nbr in range(0,5); for 0 in range(0,5) – dakle, ovaj uslov je True, jer je vrednost 0 u opsegu 0–5; programska petlja ulazi u blok koda ispod petlje i ispisuje vrednost promenljive nbr, koja u ovom trenutku iznosi 0.
- Druga iteracija – for 1 in range(0,5) – ovaj uslov je True, jer je i broj 1 u opsegu 0–5; programska petlja ulazi u blok koda ispod petlje i ispisuje vrednost promenljive nbr, koja u ovom trenutku iznosi 1.
- Ista procedura se ponavlja za ostale brojeve dok se ne dođe do slučaja gde je vrednost promenljive nbr = 4. Taj slučaj predstavlja ujedno i poslednji tačan uslov u ovoj iteraciji i nakon nje programska petlja će izaći iz ove petlje i preći na dalje izvršavanje programa.

Primer korišćenja for petlje na listi stringova pomoću indeksa

```
languages = ['German', 'Spanish', 'French']
for index in range(len(languages)):
    print("Language on index {},is ={}".format(index,
languages[index]))
```

Analiza

Ovaj primer se razlikuje od prethodnog u jednoj bitnoj stavci, a to je: umesto da iteriramo kroz samu sekvencu elemenata, ovde iteriramo kroz listu indeksa liste languages. Listu postojećih indeksa ove liste smo stvorili komandom range(len(lista)). Funkcija len() u ovom slučaju vraća veličinu (broj elemenata) prosleđene sekvence, pa se ta vrednost prosleđuje funkciji range(). Time dobijamo listu indeksa i sada te elemente koristimo kako bismo pristupili određenom elementu liste. Pošto naša lista languages ima tri elementa, funkcija len() je vratila vrednost 3, dok je funkcija range () generisala vrednosti 0, 1, 2. U svakoj iteraciji smo koristili jednu od ove tri vrednosti da pristupimo elementu liste languages na tom indeksu.

For-else petlja

Sintaksa Pythona nam dozvoljava i korišćenje else naredbi, i to po sledećem principu:

- dok postoje elementi u sekvenci, for petlja će se izvršavati;
- u slučaju kada ponestane elemenata, izvršiće se blok koda unutar else naredbe.

Primer sintakse For-else petlje

```
for element in sekvenca:
    blok koda koji treba da se izvrši.
else:
    blok koda koji treba da se izvrši.
```

Ove petlje se ređe viđaju u praksi.

Pitanje

Petlja definisana na sledeći način: `while None:`

- izvršiće se jednom
- **neće se izvršiti nijednom**
- zauvek će se izvršavati

Objašnjenje:

Petlja se neće izvršiti nijednom, jer ključna reč None ima vrednost False.

Naredbe za kontrolu petlji

Nekada želimo da izađemo iz petlje bez čekanja da se ona završi. Za ove operacije imamo na raspolaganju tri naredbe: break, continue i pass.

Naredba break

Kada glavna programska petlja dođe do ove naredbe, petlja prestaje sa izvršavanjem i program nastavlja dalje do sledeće naredbe koja je izvan petlje. Može se koristiti i u for i u while petlji. Korišćenje ove naredbe se vidi na sledećem primeru:

Primer naredbe break

```
for i in range(0,5):
    print(i)
    if i == 2:
        break
```

Nakon što je prevodilac uočio da je uslov `i == 2` tačan, stupio je u izvršavanje if bloka unutar ove for petlje – a to je izvršavanje naredbe break, čime je izvršavanje petlje prekinuto iako sekvenca nije do kraja iterirana.

Ako iskoristimo naredbu break, a koristimo for-else / while-else petlje, onda se else blok koda neće izvršiti.

Naredba continue

Ova naredba se koristi za preskakanje dela koda, vraćanje na početak petlje i nastavljanje sa sledećim elementom u sekvenci. Može se koristiti jedino u petljama (for i while petlje). Slučaj korišćenja ove naredbe se vidi na sledećem primeru:

Primer naredbe continue

```
for i in range(0,5):
    if i == 2:
        continue
    print(i)
```

Nakon što se uslov naredbe `if i == 2` ispunio, program je izvršio naredbu continue i, umesto da nastavi sa izvršavanjem naredbe print(i), preskočio ju je i nastavio sa sledećim elementom sekvence. Zato u ispisu ovog programa nedostaje vrednost 2.

Naredba pass

Ova naredba se koristi kada nam je deo koda potreban, ali trenutno nismo sigurni u njegovu implementaciju. Obično se koristi kada želimo samo da definišemo funkciju, a logiku ispišemo kasnije.

Primer naredbe pass

```
for i in range(0,5):
    pass

print(i)
```

S obzirom na to da u for petlji nije moguće izostaviti blok sa kodom, a trenutno ne znamo šta tačno želimo da uradimo sa samom petljom, dodali smo naredbu `pass`, što ne znači da se petlja neće izvršiti. Za petlju znamo da se izvršila po promenljivoj i koja je nakon izvršavanja ove petlje zadržala njenu poslednju vrednost – poslednja vrednost u zadatom opsegu range funkcije, dakle 4.

Vežba

Koristeći for petlju ispišite prvih deset negativnih brojeva. Nakon završetka petlje ispišite string Done!.

Radno okruženje

Rešenje

```
for i in range(-10, 0):
   print(i,end=',')
print("\nDone!")
```

Objašnjenje:

Jedno od rešenja za ovu vežbu je korišćenje range() funkcije gde je početak intervala -10, a kraj intervala 0. Podsetimo se da range() funkcija ne uzima u obzir krajnji interval, pa zato, ako želimo i vrednost -1 u opsegu, moramo zadati 0 kao krajnji interval. Prilikom ispisa na ekran, u print() funkciji smo koristili end parametar, kojim nagoveštavamo kako želimo da nam se linija koju ispisujemo završi – podrazumevana vrednost ovog argumenta je prelazak u novu liniju (izlazna sekvenca \n). Kao što vidite, rezultat koda je:
-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,
Done!

Pogledajte šta se dešava kada poslednja linija (print("\nDone!")) počinjete jedan tab dalje, zajedno sa (print(i,end=',')) linijom. Šta ste primetili?

Ono što vidite se dešava zato što… uvek obratite pažnju na…

Rezime

- Petlje nam omogućavaju da jedan kod izvršimo više puta.
- Petlje u Pythonu su: for petlja i while petlja.
- Ugrađena funkcija range() nam vraća generator sa traženim intervalnim vrednostima. Ako prosledimo jedan argument, dobićemo vrednosti do tog prosleđenog broja (počevši od nule). Ako prosledimo dva argumenta, prosledili smo početak i kraj željenog intervala. U slučaju prosleđivanja tri argumenta, sa prva dva argumenta prosleđujemo početak i kraj intervala, a trećim prosleđujemo korak intervala.
- While petlja se definiše pomoću ključne reči while koju prati uslov ili ekspresija.
- While petlja će se završiti kada prosleđeni uslov ili ekspresija postanu netačni.
- Beskonačna while petlja nastaje kada zadati uslov nikada ne postane netačan.
- While-else petlja nam omogućava da iskontrolišemo šta će se desiti odmah nakon što uslov u while petlji postane netačan.s
- For petlja se definiše pomoću ključne reči for i koristi se za iteraciju prosleđene sekvence.
- For-else petlja nam omogućava da iskontrolišemo šta će se desiti odmah nakon što for petlja prođe kroz poslednjih element sekvence.
- Naredbe za kontrolu petlji su break, continue i pass.
- Naredbu break koristimo kada želimo da izađemo iz trenutnog nivoa petlje.
- Naredbu continue koristimo kada želimo da preskočimo deo koda trenutne petlje i da pređemo na sledeći element sekvence.
- Naredbu pass koristimo kada nam je deo koda potreban, ali trenutno njegova logika nije ispisana, pa umesto nje pišemo pass.