1 Prisjetimo se...

U ovom poglavlju naučit ćeš:

- jednostavne tipove podataka
- osnovne naredbe u *Pythonu*
- kako napisati vlastitu funkciju.

1.1. Jednostavni tipovi podataka i naredba pridruživanja

Kada u programu želimo nešto izračunati, koristit ćemo se naredbom pridruživanja kojom možemo pridružiti ili promijeniti vrijednosti varijabli s lijeve strane. Podatci pohranjeni u varijable mogu biti različitih tipova. Prisjetimo se prvo jednostavnih tipova podataka u *Pythonu*. To su cjelobrojni tip (**int**), decimalni broj prikazan s pomoću zapisa s pomičnom točkom (**float**) i logički tip (**Bool**).

U tablici 1.1 prikazane su osnovne operacije s cjelobrojnim i decimalnim brojevima.

Tablica 1.1. Osnovne operacije s brojevima

prioritet	operacija	naredba pridruživanja	vrijednost pohranjena u varijabli c	
			za a = 6 i b = 4	za a = – 6.5 i b = 4
prvi	**	c = a ** b	1296	1785.0625
drugi	*	c = a * b	24	-26.0
	/	c = a / b	1.5	-1.625
	//	c = a // b	1	-2.0
	%	c = a % b	2	1.5
treći	+	c = a + b	10	-2.5
	-	c = a - b	2	-10.5

Možemo uočiti da je rezultat cjelobrojnog dijeljenja `6` sa `4` jednak `1` te se mnogi iznenade kad vide naizgled sličan primjer `-6.5 // 4` kod kojeg rezultat nije `-1.0` već `-2.0`. Kod cjelobrojnog dijeljenja broj se uvijek zaokružuje na manji pa kako je `-6.5 / 4 = -1.625`, pri zaokruživanju na manji broj rezultat će biti `-2`.

Ako je više operacija samo drugog prioriteta (`*`, `/`, `//`, `%`) u nizu, one se računaju slijeva nadesno, dok se u izrazu kod kojeg imamo više operacija potenciranja zaredom, vrijednost izračunava zdesna nalijevo.

U izrazima se naravno mogu koristiti okrugle zagrade te onda izrazi unutar njih imaju viši prioritet.

Osim tipova podataka za pohranu brojeva imamo i logički tip podataka (`Bool`) koji ima dvije vrijednosti: `False` i `True`. U logičkim izrazima koriste se logički operatori `not`, `and` i `or`. Kod usporedbi koristimo se relacijskim operatorima, a rezultat tih usporedbi je logička vrijednost.

U izrazima će se **relacijske** operacije primjenjivati nakon **aritmetičkih**, a **logičke** operacije nakon relacijskih.

```
>>> 2 + 3 > 5 and 3 > 2
False
```

Prvo će se izvesti zbrajanje, zatim usporedba te na kraju logička operacija `and`.

Kod pridruživanja vrijednosti varijabli, vrijednost će se pohraniti u neku od slobodnih memorijskih lokacija. U osnovi, varijabla će sadržavati adresu te memorijske lokacije.

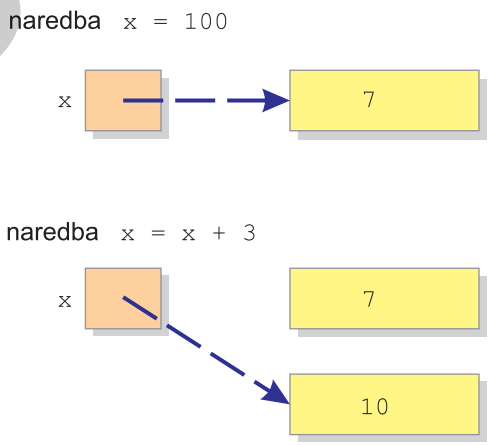
Prisjetimo se što se događa kada napišemo:

```
>>> x = 7
>>> x = x + 3
>>> x
10
```

Kada napišemo naredbu `x = 7`, tada će varijabla `x` pokazivati na neku, nama nepoznatu, memorijsku lokaciju gdje će biti zapisana vrijednost `7`.

Kada se izrazom s desne strane naredbe pridruživanja izračuna nova vrijednost, ona će se smjestiti na potpuno novu lokaciju, a varijable `x` će se usmjeriti na tu novu vrijednost, slika 1.1. Do stare vrijednosti više ne možemo doći. U velikoj većini slučajeva nama je sasvim nevažno je li nova vrijednost na istom mjestu ili je ona smještena na neku novu lokaciju. Zbog toga možemo jednostavno govoriti da se promijenila vrijednost varijable.

Programski jezik *Python* autonomno omogućuje da se memorijske lokacije koje više nisu aktivne, tj. koje se više ne koriste, mogu koristiti za pohranu novih vrijednosti.



Slika 1.1. Smještaj vrijednosti u varijablu

U *Pythonu* možemo koristiti i operatore proširenog pridruživanja.

```
>>> a = 17
>>> b = 9
>>> c = 4
>>> a *= b // c
>>> a
```

```
34
>>> a, b, c = 17, 9, 4
>>> a = a * b // c
>>> a
38
```

Naredba `a *= b // c` ekvivalentna je naredbi `a = a * (b // c)` gdje se prvo izračunava vrijednost cjelobrojnog dijeljenja, a onda rezultat množenja. Možemo zaključiti da se prvo izračunava vrijednost s desne strane naredbe proširenog pridruživanja, a zatim se izvršava operacija uz naredbu pridruživanja.

1.2. Upis i ispis vrijednosti

Kada želimo da naš program učita neku vrijednost, upotrebljavamo funkciju `input()`. Ta funkcija vraća podatak koji smo utipkali u obliku stringa tj. kao niz znakova kodiranih UNICOD-om te se on pridružuje zadanoj varijabli. Prisjetimo se da je string tip podataka koji je omeđen jednostrukim ili dvostrukim navodnicima.

Ako želimo da upisana vrijednost bude prikazana kao cijeli broj ili kao decimalni broj, primijenit ćemo funkcije koje će string pretvoriti u željeni oblik podatka.

```
>>> n = input()
567
>>> n
'567'
>>> n = int(n)
>>> n
567
```

Python nam omogućuje korištenje jedne funkcije unutar druge, kao što je to u sljedećem primjeru.

```
>>> x = float(input())
567
>>> x
567.0
```

Podatke ispisujemo s pomoću funkcije `print()`.

```
>>> print(x)
567.0
>>> print(n, x)
567 567.0
```

Funkcije `input()` i `print()` te metoda `format()` služe za interakciju programa s korisnikom primjenom tekstualnog sučelja. U osnovi funkcije `input()` i `print()` ispisuju string koji je važno funkcionalno oblikovati. Za to se možemo poslužiti i funkcijom `str()` koja broj pretvara u string te operatorima + i * koji spajaju dva stringa ili uvišestručuju neki string.

Kada napišemo ime neke funkcije i otvorimo okruglu zagradu, na zaslonu će se pojaviti okvir s osnovnim uputama vezanim za tu funkciju. Za funkciju `print()` to će izgledati kao na slici 1.2.

```
>>> print(
        print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False
```

Slika 1.2. Prikaz funkcije `print()`

Tekst u okviru je tekst koji je autor funkcije napisao unutar dokumentacijskog stringa. To je string koji je omeđen trima uzastopnim navodnicima.

Danas je uobičajeno da se i u naredbi za ispis koristi samo jedan string pri čemu možemo koristiti opciju f-string (za inačice 3.6. i više) ili metodu `.format()`.

Ako koristimo metodu `format()`, sadržaj stringa će osim teksta, koji se ispisuje, imati i izraze oblika {0}, {1}... Vrijednosti u vitičastim zagradama bit će nenegativni cijeli brojevi. Iza takva stringa dolazi točka, a potom riječ `format()`. Nakon riječi `format` u zagradama se navode vrijednosti koje će zamijeniti odgovarajuće izraze {0}, {1}... . Prva od tako navedenih vrijednosti će kod ispisa zamijeniti izraz {0}, druga izraz {1} itd. Vrijednosti mogu biti neke konstantne vrijednosti, sadržaji varijabli, izrazi s varijablama i slično.

Pogledajmo primjer ispisa korištenjem metode `format()`.

```
>>> naslov = 'Primjena zakona komutacije na zbroj dvaju brojeva'
>>> a = 2
>>> b = 7
>>> print('{0}\n{1} + {2} = {2} + {1} = {3}'.format(naslov, a, b, a + b))
Primjena zakona komutacije na zbroj dvaju brojeva
2 + 7 = 7 + 2 = 9
```

Ako se pak koristimo oblikovanjem ispisnog stringa preporučenom opcijom `f` ili `F`, gornji primjer će izgledati:

```
>>> naslov = 'Primjena zakona komutacije na zbroj dvaju brojeva'
>>> a = 2
>>> b = 7
>>> print(f'{naslov}\n{a} + {b} = {b} + {a} = {a+b}')
Primjena zakona komutacije na zbroj dvaju brojeva
2 + 7 = 7 + 2 = 9
```

Kod oblikovanja stringa unutar vitičastih zagrada možemo naznačiti tip podataka koji ispisujemo i to tako da dodamo dvotočku i oznaku tipa:

- `d` – ako ispisujemo cijeli broj
- `f` – ako ispisujemo decimalni broj
- `s` – ako ispisujemo string.

Pogledajmo nekoliko načina ispisa decimalnog broja.

```
>>> a = 815
>>> b = 3
>>> print(f'{''Količnik''} brojeva {a} i {b} je {a/b}')
Količnik brojeva 815 i 3 je 271.6666666666667
>>> print(f'{''Količnik''} brojeva {a} i {b} je {a/b:f}')
Količnik brojeva 815 i 3 je 271.666667
>>> print(f'{''Količnik''} brojeva {a} i {b} je {a/b:2f}')
Količnik brojeva 815 i 3 je 271.67
```

Vidimo da ako kod decimalnog broja stavimo oznaku formata `f`, dobivamo ispis na šest decimalnih mjesta. Ako želimo ispis s nekim drugim brojem decimalnih mjesta, onda ćemo prije oznake tipa naznačiti na koliko decimalnih mjesta želimo ispis. U trećem slučaju su to dva decimalna mjesta.

Formatirani ispis omogućava dodatno formatiranje kod ispisa brojeva. Ispis cijelih brojeva moguće je formatirati naredbom `{nd}`, pri čemu je `n` neki prirodan broj koji označava na koliko će se mjesta ispisati određeni prirodan broj. Ako je `n = 5`, a broj koji ispisujemo je 365, to znači da će se broj 365 ispisati na ukupno 5 mjesta. Ispis broja započinje od krajnjeg desnog mjesta pa će ispis imati sljedeći oblik:

		3	6	5

što znači da će ispred broja biti ispisana dva prazna mjesta.

Kada prazna mjesta kod ispisa broja želimo zamijeniti nulama, napisat ćemo:

```
>>> print(f'{365:05d}')
00365
```

Slično se može formatirati i ispis decimalnog broja, pri čemu u tom slučaju formatiranje ima dva parametra `{n.mf}`. Kao i kod ispisa cijelog broja `n` je ukupni broj mjesta na koji će se decimalni broj ispisati (uključujući i decimalnu točku), dok je `m` broj znamenaka iza decimalne točke koje će biti ispisane. Pogledajmo primjer:

```
>>> print(f'{13/4 :7.2f}')
```

Za ispis broja koristit će se ukupno 7 mjesta od čega će jedno biti rezervirano za decimalnu točku, dva mjesta će biti za decimalni dio broja (ispisat će se samo dvije decimalne znamenke) dok će preostala 4 mjesta biti za cjelobrojni dio broja. Grafički će taj ispis imati sljedeći oblik:

			3	.	2	5

S obzirom na to da će se ukupni broj mjesta uvijek po potrebi povećati, vrlo često nećemo ga niti unaprijed odrediti, odnosno taj ćemo dio jednostavno izostaviti i pisat ćemo samo točku i broj znamenaka iza decimalne točke.

```
>>> print('f{13 / 4 :.3f}')
3.250
```

Za oblikovanje stringa za ispis postoji i cijeli niz dodatnih mogućnosti, no njih ćemo pojedinačno objasniti u trenutku njihove primjene.

Na sličan način možemo formatirati i ispis prilikom unosa podataka kao u primjeru 1.1.

Primjer 1.1.

Anja ima `n` kuna i za taj novac želi kupiti olovke. Olovka košta `c` kuna. Napišimo program koji će učitati koliko novaca ima Anja i cijenu olovke, a ispisati najveći mogući broj olovaka koje Anja može kupiti te koliko će joj novaca ostati.

Napomena: Cijena olovke bit će prirodan broj.

Priprema zadatka:

ulaz: količina novaca `n` i cijena olovke `c`, cjelobrojne vrijednosti

izlaz: najveći broj olovaka `broj`, ostatak novaca `ostatak`, cjelobrojne vrijednosti.

Algoritam:

```
broj = n // c
ostatak = n % c
```

Testni primjer:

ulaz	izlaz	obrazloženje
45 7	6 3	Anja može kupiti najviše 6 olovaka. Ostat će joj 3 kune.

Programski kôd:

```
n = int(input('Koliko novaca ima Anja: '))
c = int(input('Cijena olovke: '))
broj = n // c
ostatak = n % c
print('Anja može kupiti najviše', broj, 'olovaka i ostat će joj', ostatak, 'kn.')
```

1.3. Naredbe grananja i ponavljanja

U tablici 1.2. dani su opći oblici naredbi grananja i ponavljanja.

Tablica 1.2. Naredbe grananja i ponavljanja

naredba grananja	naredba ponavljanja s uvjetom	naredba s unaprijed zadanim brojem ponavljanja uz korištenje `range()`

`if uvjet:`
`  blok naredbi1`
`elif uvjet:`
`  blok naredbi2`
`else:`
`  blok naredbi3`
	`while uvjet:`
	`  blok naredbi`
		`for i in range([a], b, [k]):`
		`  blok naredbi`

Kod `if` naredbe moguće je koristiti oblik bez opcije `elif`, odnosno bez opcije `elif` i/ili **else**. Kod naredbe `while`, naredbe unutar petlje će se ponavljati dok je uvjet zadovoljen.

Naredba `for` može biti napisana i bez funkcije `range()`, no onda moramo imati neki iterabilni objekt čije će vrijednosti poprimati varijabla `i`.

Primjer 1.2.a Napišimo program koji ispisuje djelitelje učitanog prirodnog broja.

Priprema zadatka:

ulaz: `b` – prirodni broj, cjelobrojna vrijednost

izlaz: `djelitelji` – djelitelji učitanog broja, cjelobrojne vrijednosti.

Algoritam:

Ako je neki broj djelitelj nekog broja, onda je ostatak pri dijeljenju ta dva broja jednak 0. Dakle, za neki učitani broj treba provjeriti s kojim je sve brojevima djeljiv, počevši od 1 do njega samog.

Testni primjer:

ulaz	izlaz
48	1 2 3 4 6 8 12 16 24 48
7	1 7
1	1

Programski kôd:

```
b = int(input('b = '))
for i in range(1, b + 1):
    if b % i == 0:
        print(f'{i},', end=' ')
```

Primjer 1.2.b Napišimo program koji ispisuje djelitelje učitanih prirodnih brojeva. Program treba upisivati brojeve dok se ne upiše 0. Kod ispisa djelitelji trebaju biti odvojeni zarezom.

Priprema zadatka:

ulaz: `b` – pojedini učitani prirodni broj, cjelobrojna vrijednost

izlaz: `djelitelji` – djelitelji učitanog broja, cjelobrojne vrijednosti.

Algoritam:

S obzirom na to da je svaki broj sigurno djeljiv s 1 i sa samim sobom, algoritam možemo ubrzati tako da provjeravamo je li broj djeljiv s brojevima od 2 do polovine tog broja. Na taj će način se naša `for` naredba za svaki učitani broj izvesti dvostruko brže.

Za provjeru s kojim je brojevima broj djeljiv, koristit ćemo `for` petlju, a za upis brojeva koristit ćemo `while` petlju.

Željeni izgled zaslona:

```
b = 2
Djelitelji broja 2 su: 1, 2.
b = 49
Djelitelji broja 49 su: 1, 7, 49.
b = 48
Djelitelji broja 48 su: 1, 2, 3, 4, 6, 8, 12, 16, 24, 48.
b = 1
Djelitelj broja 1 je 1.
b = 0
```

Programski kôd:

```
b = int(input('b = '))
while b > 0:   #možemo pisati samo b, što znači da je b različit od 0
    if b > 1:
        print(f'Djelitelji broja {b} su: 1,', end=' ')
        for i in range(2, b // 2 + 1):
            if b % i == 0:
                print(f'{i},', end=' ')
        print(f'{b}.')
    else:
        print('Djelitelj broja 1 je 1.')
    b = int(input('b = '))
```

Na prvi pogled programski kod je nepregledniji, no dobili smo oblik ispisa kakav je bio zadan u zadatku. Poboljšanje algoritma ima smisla ako ćemo ga koristiti i za velike brojeve.

1.4. Funkcije i moduli

Razumno je i praktično imalo složeniji problem podijeliti na manje jednostavnije potprobleme koje možemo riješiti funkcijama. Funkcije se pak mogu grupirati u module.

1.4.1. Kako izgledaju funkcije?

Funkcije u *Pythonu* (a slično je i u drugim programskim jezicima) zasebne su cjeline koje obavljaju neku zadaću. Dakle, to su manji zasebni programski kodovi. Opći oblik definicije funkcije je:

```
def ime_funkcije(popis_parametara):
    blok naredbi
    return vrijednost
```

Prisjetimo se da funkcije u *Pythonu* ne moraju uvijek imati popis parametara i ne moraju vraćati neku vrijednost.

Kod nekih ćemo funkcija prilikom korištenja moći naznačiti ulazne vrijednosti ili prihvatiti već unaprijed definirane vrijednosti, npr. kod funkcije `input()`, dok kod nekih koje nam služe samo za preglednije oblikovanje programa nećemo morati imati izlazne parametre, npr. `print()`.

Kada se u funkciji dođe do naredbe `return`, napušta se izvođenje funkcije koja u tom trenutku vraća vrijednosti navedene iza ključne riječi `return`. Naredbi `return` može biti više unutar neke funkcije, tj. funkciju možemo napustiti na više mjesta.

Neke funkcije mogu vraćati i više od jedne vrijednosti: dvojku, trojku ili, u općem slučaju, ***n*-torku** vrijednosti. Zbog toga je u *Pythonu* uveden i posebni tip podataka koji se zove ***n*-torka** (engl. ***tuple***). Pojedina vrijednost *n*-torke može se posebno dohvatiti i upotrijebiti (ali se ne može mijenjati).

1.4.2. Ugrađene funkcije

U *Pythonu* postoji cijeli niz funkcija koje nam pomažu pri rješavanju problema. Neke od tih funkcija su standardne ugrađene funkcije (engl. *built-in functions*): `int()`, `float()`, `bin()`, `hex()`, `ord()`, `input()`, `print()`, `abs()`, `round()` itd.

Funkcije mogu vraćati i više vrijednosti, tj. *n*-torku. Primjer ugrađene funkcije koja vraća više vrijednosti je `divmod(a, b)`. Ona vraća dvojku: količnik i ostatak cjelobrojnog dijeljenja:

```
(količnik, ostatak) = divmod(a, b)
```

Prilikom upisa imena ugrađene funkcije i upisa otvorene zagrade pojavljuju se informacije o samoj funkciji i o obliku upisa ulaznih parametara.

1.4.3. Funkcije koje se nalaze u modulima

Osim što imamo standardne ugrađene funkcije, tzv. jezgrene funkcije, samom instalacijom *Pythona* dostupne su nam i funkcije koje su pohranjene u dodatnim datotekama koje se nazivaju **modulima**. Primjer takvog modula je `math` koji sadrži funkcije koje su na neki način srodne. Radi se o datoteci koja sadrži matematičke funkcije i neke konstante koje se koriste u matematici.

Uz instalaciju *Pythona* dostupni su nam mnogi unaprijed gotovi moduli.

Kako se koristiti tim modulima kao i modulima koje ćemo sami izraditi?

Funkcije modula bit će nam dostupne ako to u programu najavimo, a uobičajeno je to napraviti na samom početku programa. Najavu korištenja modula možemo napraviti na više načina, tablica 1.3.

Tablica 1.3. Načini najave uporabe modula

opis najave	primjer poziva funkcije	objašnjenje
`import math`	`y = math.cos(x)`	najava korištenja modula, potrebno je navesti i ime modula
`from math import cos, sin`	`y = cos(x)`	najava korištenja navedenih funkcija modula
`from math import *`	`y = cos(radians(stupnjevi))`	najava korištenja svih funkcija modula

1.4.4. Pisanje vlastitih funkcija

Primjer 1.3. Napišimo program koji će za *n* upisanih prirodnih brojeva ispisivati koliko ih je prostih.

Priprema zadatka:

ulaz: `n` – broj koliko ćemo brojeva upisati, cjelobrojna vrijednosti

`b` – pojedini učitani prirodni broj veći od 1, cjelobrojna vrijednost

izlaz: `broj_prostih` – broj prostih brojeva, cjelobrojna vrijednost.

Matematička osnova algoritma:

Prosti brojevi su svi prirodni brojevi veći od 1 koji su djeljivi samo s brojem 1 i sami sa sobom. Broj 1 po definiciji nije ni prost broj ni složen broj.

Razrada algoritma:

Algoritam za provjeru je li broj *b* prost: ponavlja se postupak provjere djeljivosti broja od vrijednosti 2 do $\sqrt{b}$.

Vidimo da ćemo za svaki od brojeva trebati provjeravati je li prost, što je u osnovi zasebna funkcionalnost i dobro ju je napisati u obliku funkcije.