Podloge za stručno usavršavanje učitelja osnovnih škola
za domenu
Računalno razmišljanje i programiranje

01

Interaktivno sučelje Pythona, prirodni brojevi i nizovi znakova (stringovi), funkcije `int()`, `str()`, `input()` i `print()`

Uz dozvolu izdavača korišteni su sadržaji iz priručnika:

Leo Budin Predrag Brođanac Zlatka Markučič
Smiljana Perić Dejan Škvorc Magdalena Babić

Računalno razmišljanje i programiranje u Pythonu
Element, Zagreb, 2017

Uključivanje računala

Prilikom uključivanja računala pokrenut će se operacijski sustav – glavni program za nadzor i obavljanje svih poslova u računalu. Na zaslonu računala pojavit će se slika koju zovemo **grafičkim korisničkim sučeljem** operacijskog sustava (engl. *Graphical User Interface – GUI*). Posredstvom tog sučelja možemo pokrenuti ostale programe koji su pohranjeni u datotekama na čvrstom disku računala. To uobičajeno činimo dvostrukim klikom miša na imena tih datoteka ili na sličice (ikone) koje ih **simboliziraju**. Pokretanjem nekog programa najčešće se otvara posebni **prozor** s grafičkim sučeljem koji korisniku omogućuje rad s programom (pisanje teksta, crtanje crteža, pokretanje filmova, igranje igara i sl.).

Prozori radnog okruženja Pythona

Prilikom pripreme programa u *Pythonu* i prilikom njihova izvođenja susrest ćemo se s tri vrste takvih prozora:

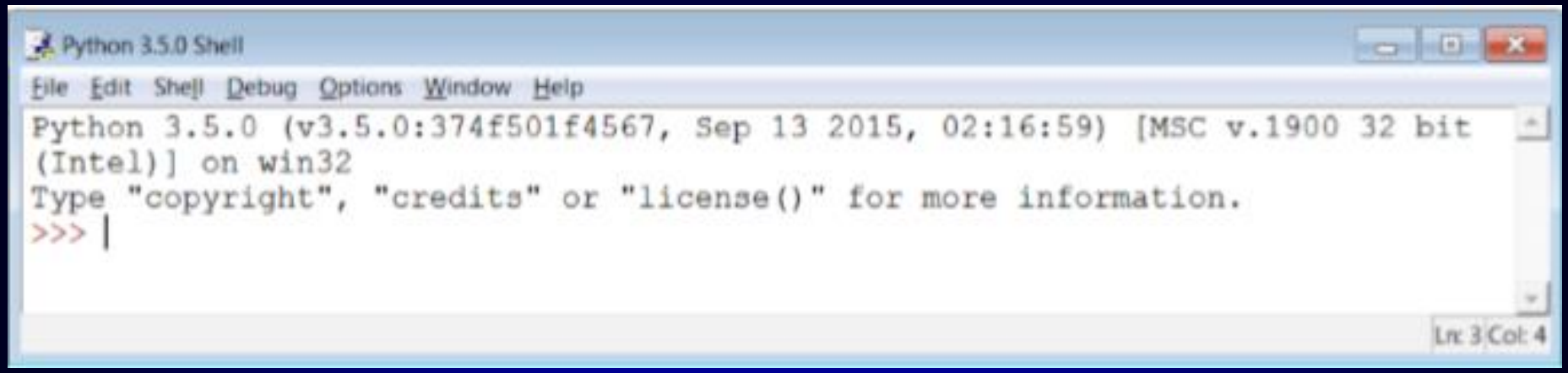
- Prozor **interaktivnog sučelja** poslužit će nam za prikaz podataka koje tipkovnicom unosimo u računalo i prikaz izlaznih podataka iz računala – rezultata. Interaktivno sučelje služit će nam i za izvođenje pojedinačnih naredbi.
- Prozor za pisanje programa po svojoj ulozi podsjeća na prozor bilo kojeg programa za pisanje i uređivanje (editiranje) tekstova tako da se naziva i **editorom**. Sadržaj tog prozora moći ćemo pohraniti kao datoteku i trajno sačuvati na tvrdom disku računala.
- Prozor za slikovni (grafički) prikaz kraće nazivamo **grafički prozor**.

Interaktivno sučelje Pythona

Prilikom instaliranja programskog paketa Python sve njegove programske datoteke bit će smještene u mapu **`Python 3.x.y`**. (brojevi **`x.y.`** označavaju inačicu programa).
Klikom miša na tu mapu pokazat će se podmapa a u nutar nje ime datoteke **`IDLE`**. Klik miša na tu datoteku otvara prozor interaktivnog sučelja Pythona.

Ikonu za otvaranje te datoteke možemo preseliti na radnu vrpcu i time pojednostavniti otvaranje interaktivnog sučelja.

Ta ikona izgleda ovako: 

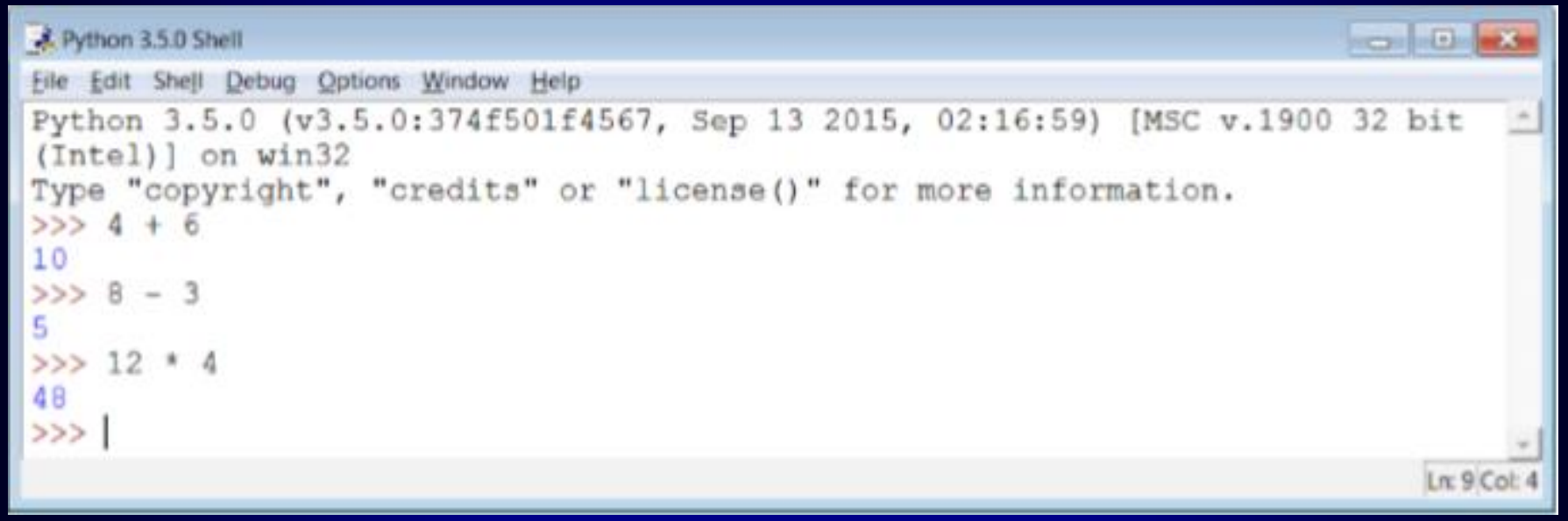


U prvim redovima tog prozora vidimo osnovne podatke o inačici programa koja je instalirana na našem računalu. Nakon toga vidimo red koji na početku ima tri znaka >>> i treptajuću oznaku "|". To je oznaka koja pokazuje da je program spreman prihvatiti znakove koje ćemo utipkavati s tipkovnice. Uobičajeni naziv za oznaku >>> je *prompt*. Naziv *prompt* preuzet je iz engleskoga jezika.

Taj će nam prozor biti osnovna veza s računalom prilikom pripremanja i izvođenja programa. Posredstvom tog prozora pokretat ćemo prozor za pisanje programa, preko njega ćemo unositi ulazne podatke u program i u njemu će programi ispisivati rezultate svojeg izvođenja.

Dakle, posredstvom tog prozora mi prenosimo sadržaje računalu i računalo ispisuje povratne informacije. Kažemo da smo preko tog prozora u stalnoj interakciji s računalom pa ga zbog toga i zovemo **interaktivnim sučeljem**.

Interaktivno sučelje kao kalkulator



U daljnim primjerima nećemo više rabiti slike cijelog prozora već ćemo samo prikazivati sadržaj prozora

Iz sljedećih se primjera vide pravila izračunavanja aritmetičkih izraza:

```
>>> 3 + 8
11
>>> 8 + 3
11
>>> 3 + (8 + 5)
16
>>> (3 + 8) + 5
16
>>> 3 * (8 + 5)
39
>>> 3 * 8 + 3 * 5
39
```

U *Pythonu* se aritmetički izrazi pišu po pravilima koja su vrlo slična onima koje poznajemo iz matematike. **Operatori** (simboli matematičkih operacija) za zbrajanje (+) i oduzimanje (–) jednaki su onima u matematici. Operator za množenje je u *Pythonu* zvjezdica (*) dok u matematici rabimo točku na sredini.

Za dijeljenje prirodnih brojeva u *Pythonu* postoje dva operatora. To su:

- operator za računanje količnika `//` (uzastopno napišemo znak /)
- operator za računanje ostatka pri dijeljenju `%`.

Imena pretinaca – programske varijable, naredba pridruživanja

U većini slučajeva korisno bi bilo brojeve s kojima računamo, a i same rezultate računanja pohraniti u spremnik računala što bi nam omogućilo njihovo ponovno korištenje. Pritom svaka vrijednost koju trebamo pohraniti mora dobiti svoj pretinac u memoriji računala. Pretince ćemo imenovati tako da ih možemo razlikovati, primjerice: `pretinac_a`, `pretinac_b`, `pretinac_c`.

U pretinac ćemo pohraniti sadržaj (vrijednost) **naredbom pridruživanja**. U toj naredbi s lijeve strane znaka pridruživanja (=) treba napisati naziv pretinca, a s desne strane znaka pridruživanja vrijednost koju se u pretinac želi pohraniti. Pogledajmo primjer:

```
>>> pretinac_a = 3
>>> pretinac_b = 8
>>> pretinac_c = pretinac_a + pretinac_b
>>> pretinac_a
3
>>> pretinac_b
8
>>> pretinac_c
11
```

Sadržaj pojedinih pretinaca možemo pogledati tako da napišemo njegovo ime i pritisnemo tipku <unos>.

Imena pretinaca moraju se pisati u skladu s određenim pravilima. Imena se mogu sastojati od proizvoljnog broja znakova koji mogu biti:

- velika i mala slova (možemo se koristiti i slovima hrvatske abecede s dijakritičkim znakovima: č, ć, đ, š i ž)
- znamenke dekadskog brojevnog sustava (0, 1, 2, 3, 4, 5, 6, 7, 8 i 9)
- znak donje crtice (_)

s time da ime ne smije početi znamenkom. Kod korištenja slova hrvatske abecede s dijakritičkim znakovima mogu se pojaviti neki problemi ako program izvodimo na različitim računalima.

Pogledajmo neke primjere:

```
jedan2tri          #ispravno ime
jedan_2_tri        #ispravno ime
jedan-2-tri        #neispravno ime (crtica nije dozvoljeni znak)
1dvatri            #neispravno ime (prvi znak mora biti slovo)
a_0                #ispravno ime
a.1                #neispravno ime (točka nije dozvoljeni znak)
a 3                #neispravno ime (praznina nije dozvoljeni znak)
a3                 #ispravno ime
```

Za pisanje imena u pravilu se nećemo koristiti velikim slovima. Velikim ćemo se slovima koristiti u posebnim slučajevima. O tim će slučajevima biti riječi u narednim poglavljima.

U *Pythonu* postoje tzv. rezervirane riječi koje imaju posebna značenja i ne smiju se rabiti kao imena.

Sve što je napisano iza znaka # do kraja reda Python zanemaruje. Tako pišemo komentare!

O naredbi pridruživanja

U programskom jeziku *Python* se znakom "=" označava pridruživanje vrijednosti s desne strane znaka "=" varijabli koja se nalazi s lijeve strane znaka "=". Ovdje treba napomenuti da znak "="nema isto značenje kao znak "=" u matematici. Uočimo da u tom smislu naredbu `c = a + b` tumačimo: prvo se određuje vrijednost izraza koji stoji desno od znaka pridruživanja "=" te se ta vrijednost pridružuje varijabli `c` čije ime stoji lijevo od znaka pridruživanja.

U programiranju ćemo vrlo često trebati prebrojiti koliko se puta ponovio neki događaj. Kako bismo riješili taj problem, potrebno je napisati niz naredbi. Pogledajmo primjer:

```
>>> brojilo = 0
>>> brojilo = brojilo + 1
>>> brojilo
1
>>> brojilo = brojilo + 1
>>> brojilo
2
>>> brojilo += 1
>>> brojilo += 1
>>> brojilo
4
```

Naredba pridruživanja može se primijeniti tako da se ista vrijednost istovremeno pohrani u više varijabli pa se može pisati:

```
>>> a = b = c = 1
>>> a, b, c
(1, 1, 1)
>>> a += 1
>>> b += 2
>>> c += 3
>>> a
2
>>> b
3
>>> c
4
>>> a, b, c
(2, 3, 4)
```

Nizovi znakova - stringovi

Nizovi znakova mogu se pohranjivati u varijable na isti način kao što smo to činili s brojevima. Lijevo od znaka pridruživanja napisat ćemo ime varijable, a desno od znaka pridruživanja ćemo umjesto brojeva napisati niz znakova omeđenih znakovima navodnika. U engleskom se jeziku za takav niz znakova rabi riječ *string*. U hrvatski jezik preuzeli smo tu englesku riječ pa ćemo naizmjence rabiti nazive **niz znakova** ili **string**.

Pritom možemo rabiti dvije vrste navodnika:

- uobičajeni navodni znak " (koji se na hrvatskoj tipkovnici nalazi na tipki s brojem 2) ili
- jednostruki navodni znak ' (koji se na hrvatskoj tipkovnici nalazi na tipki sa znakom ?).

```
>>> s0 = 'Ovo je tekst omeđen jednostrukim navodnicima'
>>> s0
'Ovo je tekst omeđen jednostrukim navodnicima'
>>> s1 = "Ovo je tekst omeđen dvostrukim navodnicima"
>>> s1
"Ovo je tekst omeđen dvostrukim navodnicima"
```

Mi ćemo rabiti jednostruke navodne znakove !
Jedino ako se u tekstu pojavljuje upravni govor rabit ćemo i dvostruke:

```
>>> s2 = 'Ana je rekla: "Pero mora dati 17 kuna!"'
>>> s2
'Ana je rekla: "Pero mora dati 17 kuna!"'
```

Operatori nadovezivanja (+) i uvišestručenja (*) stringa

Vidjeli smo kako se s brojevima mogu obavljati osnovne aritmetičke operacije. No, operatori `+` i `*` koji s brojevima obavljaju aritmetičke operacije zbrajanja i množenja služe za spajanje i uvišestručenje stringova. Pogledajmo:

```
>>> s3 = 'a'
>>> s4 = '12'
>>> s3, s4
('a', '12')
>>> s3 + s4
'a12'
>>> s5 = 3 * s3 + 4 * s4
>>> s5
'aaa12121212'
```

Stringovi imaju važnu ulogu u komunikaciji čovjeka i računala pa ćemo o njima još mnogi naučiti!

Veza između stringova i brojeva, Funkcije `int()` *i* `str()`

Dosad smo naučili kako se u interaktivnom sučelju varijablama pridružuju prirodni brojevi i stringovi. Tako možemo pisati:

```
>>> a, b = 12, 15                                    #1
>>> a
12
>>> b
15
>>> a + b                                            #2
27
>>> c, d = '12', '15'                                #3
>>> c + d                                            #4
'1215'
```

Naredbom (#1) pohranili smo u varijable `a` i `b` brojčane vrijednosti koje smo naredbom (#2) zbrojili i dobili točan rezultat. Jednako tako, naredbom (#3) pripisali smo varijablama `c` i `d` stringove koji su nakon toga naredbom (#4) spojeni u novi string.

Uočimo da smo naredbom (#1) u istoj naredbi dvjema varijablama pridružili vrijednosti. Taj postupak zovemo višestruko pridruživanje.

Uočimo da smo naredbom (#1) u istoj naredbi dvjema varijablama pridružili vrijednosti. Taj postupak zovemo višestruko pridruživanje.

Međutim, pokušamo li znakom + povezati varijablu koja sadrži broj s varijablom koja sadrži string, dobit ćemo sljedeći ispis:

```
>>> a + c
Traceback (most recent call last):
 File "<pyshell#16>", line 1, in <module>
a + c
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

U zadnjem redu tog pomalo nerazumljivog ispisa može se prepoznati da operator + ne dopušta da operandi budu različitog tipa.

Naime, podatci u programskom jeziku *Pythonu* (slično je i u drugim programskim jezicima) svrstavaju se u nekoliko tipova. Tako prirodni brojevi spadaju u klasu cijelih brojeva. Engleski naziv za cijele brojeve je *integer* odakle potječe naziv za tip `int`.

S druge strane, svi stringovi pripadaju tipu `str`.

U *Pythonu* postoje funkcije koje omogućuju pretvaranje jednog tipa podataka u drugi. Za sada su nam zanimljive dvije takve funkcije:

- funkcija `int()` koja prevodi tip `str` u tip `int`
- funkcija `str()` koja prevodi tip `int` u tip `str`.

Primijetimo da `int()` i `str()` pisani sa zagradom označavaju funkcije dok `int` i `str` bez zagrada označavaju tip podatka.

```
>>> s0, s1 = '13', '14'                                    #5
>>> s0, s1                                                 #6
('13', '14')
>>> s0 + s1                                                #7
'1314'
>>> int(s0) + int(s1)                                      #8
27
>>> int(s0 + s1)                                           #9
1314
```

Naredba (#5) pridružuje varijablama `s0` i `s1` stringove, a naredbom (#6) vidimo kako su te dvije vrijednosti pohranjene u varijablama. Naredba (#7) spaja ta dva stringa u jedan string. U naredbama (#8) i (#9) vidljivo je djelovanje funkcije `int()`. Možemo zaključiti da se djelovanjem funkcije `int()` string prikazuje kao cjelobrojna vrijednost.

Funkcija `int()` prihvaća samo stringove s dekadskim znamenkama. Ako se u stringu nalaze neki drugi znakovi, javit će pogrešku. Evo primjera:

```
>>> a = int('123c')
Traceback (most recent call last):
 File "<pyshell#0>", line 1, in <module>
     a = int('123c')
ValueError: invalid literal for int() with base 10: '123c'
```

U stringu koji nastojimo pretvoriti u broj nalazi se znak jednog slova. Funkcija `int()` ne može takav string pretvoriti u dekadski broj i zbog toga javlja pogrešku. U zadnjem redu ovog ispisa može se pročitati da se radi o pogrešci u vrijednosti (`ValueError`) s naglaskom da se u stringu nalazi nedozvoljeni znak.

Funkcija `input()`

Dosad smo pojedinim varijablama pridruživali vrijednosti tako da smo desno od znaka pridruživanja napisali neku vrijednost tipa `int` ili tipa `str` ili smo s desne strane znaka pridruživanja napisali izraz kojim se izračunava vrijednost koja se potom pridružila varijabli.

Na taj način možemo mijenjati sadržaj bilo kojoj varijabli tako dugo dok naredbe pišemo u interaktivnom sučelju. No kada ćemo izvoditi programe pohranjene u programskim datotekama, trebat ćemo pojedinim varijablama pridružiti vrijednosti prilikom izvođenja programa. Zbog toga postoji i drugi način za pridruživanja vrijednosti pojedinim varijablama i to uporabom ugrađene programske funkcije `input()`. Ova se funkcija kao i ostale funkcije može rabiti i u interaktivnom sučelju.

Utipkamo li ime `input` i otvorenu zagradu u interaktivnom sučelju pojavit će se sljedeći prikaz:

```
>>> input(
        Read a string from standard input. The trailing newline is stripped.
```

Pogledajmo sljedeći primjer:

```
>>> ime = input('Upiši svoje ime: ')                    #1
Upiši svoje ime: Marko
>>> ime                                                 #2
'Marko'
>>> broj = input('Upiši broj: ')                        #3
Upiši broj: 23
>>> broj                                                #4
'23'
```

Opišimo naredbe:

#1	Na zaslonu će se umjesto uobičajenog *prompta* `>>>` ispisati tekst koji je naveden kao parametar u funkciji `input()` tj. tekst `Upiši svoje ime:`. Taj nam je tekst podsjetnik da moramo utipkati svoje ime.
#2	Provjeravamo sadržaj varijable ime i vidimo da je utipkani string, u našem primjeru to je `Marko`, pohranjen u varijabli `ime`.
#3	Dajemo uputu korisniku da upiše broj te će se utipkani niz znakova `(23)`, pohraniti u varijablu `broj`.
#4	Provjeravamo sadržaj varijable `broj`.

Uočimo da smo nakon ispisa *prompta* utipkali niz znakova bez navodnika. *Python* će tom nizu znakova dodati navodnike i pohraniti ih u obliku stringa. I kada upišemo neki broj (niz znamenaka) *Python* će mu dodati navodnike i pohraniti ga u obliku stringa.

Kada string u kojem je pohranjen niz znakova želimo u računalu prikazati kao broj, trebamo se koristiti standardnom funkcijom `int()`. Funkciju `int()` možemo zapisati:

```
>>> broj = int(broj)                                    #5
>>> broj                                                #6
23
```

Naredbom (#5) pretvorili smo string u broj, pa se provjerom (#6) može ustanoviti da je u varijabli broj sada uistinu zapisana vrijednost broja, a ne više string. To je vidljivo iz ispisa jer je 23 ispisan bez navodnika.

Naredbe (#3) i (#5) možemo povezati te upisivanje broja možemo obaviti na sljedeći način:

```
>>> broj = int(input('Upiši broj: '))                   #7
Upiši broj: 17                                          #8
>>> broj
17
```

Primjer:

U interaktivnom sučelju učitajmo dva broja i ispišimo njihov zbroj.

Rješenje:

```
>>> a = int(input('a = '))
a = 7
>>> b = int(input('b = '))
b = 4
>>> c = a + b
>>> c
11
```

Funkcija `print()`

Kod pisanja i izvođenja programa koji će biti pohranjen u datoteci, za ispis vrijednosti varijabli bit će nam potrebna ugrađena funkcija `print()`. Funkcija `print()` služi za ispisivanje sadržaja na zaslon računala. Ta funkcija ne vraća nikakve vrijednosti. Ona obavlja posao ispisivanja. Za razliku od funkcije `input()`, koja se bavi samo stringovima, funkcija `print()` ispisuje i brojeve i stringove. Pogledajmo:

```
>>> a = 13                                    #1
>>> b = '13'                                  #2
>>> print(a)                                  #3
13
>>> print(b)                                  #4
13
```

Opišimo naredbe:

#1	U varijablu `a` pohranili smo vrijednost broja 13.
#2	U varijablu `b` pohranili smo string sa sadržajem 13.
#3	Korištenjem funkcije `print()` na zaslonu računala ispisali smo sadržaj varijable `a`.
#4	Korištenjem funkcije `print()` na zaslonu računala ispisali smo sadržaj varijable `b`.

Uočimo da će funkcija `print()` obje vrijednosti ispisati jednako pa na to svojstvo funkcije `print()` treba obratiti posebnu pažnju.

Da se radi o različitim sadržajima vidimo na sljedeći način:

```
>>> print(3 * a)          #izrazom 3 * a se množe brojevi 3 i 13
39
>>> print(3 * b)          #izrazom 3 * b se uvišestručuje string b
131313
```

Utipkamo li ime `print` i otvorenu zagradu u interakrivnim sučelju pojavit će se sljedeći prikaz:

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

Evo objašnjenja opisa djelovanja funkcije `print()`:

- `value, ...,` – označava da funkcija prihvaća više vrijednosti za ispis te da ih treba napisati odvojene zarezom
- `sep=' '` – označava da će se pri ispisu između tih vrijednosti umetnuti razmak
- `end='\n'` – označava da će se ispis zaključiti prelaskom u novi red.

Preostale dvije oznake možemo ovdje zanemariti i nećemo ih objašnjavati.

Posebni dvoznak `\n` (lijevo nagnuta kosa crta koja se nalazi na tipki sa slovom *Q* i malo slovo *n*) u nekom stringu označava prelazak u novi red. Varijablama `sep` i `end` unaprijed su pridijeljene navedene vrijednosti i ne moramo ih posebno isticati pri uporabi funkcije `print()` ako smo zadovoljni time da se između vrijednosti koje se ispisuju umeće jedna praznina i da se na kraju ispisa prelazi u novi red.

Postavljene vrijednosti možemo mijenjati i tako utjecati na oblik ispisa. Pogledajmo primjere:

```
>>> print('a', 'b', 'c'); print('d', 'e', 'f')                    #5
a b c
d e f
>>> print('a', 'b', 'c', end=' '); print('d', 'e', 'f')           #6
a b c d e f
>>> print('a', 'b', 'c', sep='-', end='-'); print('d', 'e', 'f', sep='-')   #7
a-b-c-d-e-f
```

U primjerima s oznakom:

#5	Dvije funkcije `print()` ispisat će svaka po tri vrijednosti i nakon ispisa prijeći u novi red. Uočimo da su funkcije napisane u istom retku te su radi toga odvojene znakom točke sa zarezom (;).
#6	Ako u prvoj funkciji `print()` promijenimo varijablu `end` tako da ona umjesto znaka za prelazak u novi redak sadrži znak razmaka (`' '`), dobit ćemo ispis svih šest vrijednosti u jednom retku.
#7	U varijable `sep` i `end` možemo upisati proizvoljne vrijednosti i dobiti različite ispise. U primjeru s oznakom (#5) smo tako u njih upisali znak za crticu.

Znamo da funkcija `print()` prihvaća za ispis i brojeva (tip `int`) i znakovne nizove (tip `str`) pa možemo pisati:

```
>>> a = int(input('a = '))
a = 7
>>> b = int(input('b = '))
b = 6
>>> c = a + b
>>> print(a, '+', b, '=', c)
7 + 6 = 13
```

Funkcija `print()` ispisuje pet vrijednosti: tri broja iz varijavli `a`, `b` i `c` i dva string '+' i '=' . Između svake od njih stavit će jedan razmak (jer je `sep=' '`).

Ispis nekog sadržaja možemo obaviti i tako da po želji oblikujemo jedan string i nakon toga taj string navedemo kao vrijednost koju će ispisati funkcija `print()`.

Međutim, pri oblikovanju složenijih stringova ne mogu se miješati tipovi `int` i `str`. Zbog toga ćemo brojeve morati pretvoriti u tip `str`. Pogledajmo kako se to može načiniti:

```
>>> s = str(a) + ' + ' + str(b) + ' = ' + str(c)
>>> s
'7 + 6 = 13'
>>> print(s)
7 + 6 = 13
```

Oblikovanje ispisnih stringova metodom `format()`

Dosad smo naučili da uz **operatore** u *Pythonu* postoje **programske funkcije** kojima se obavljaju različite operacije. Uz to postoje još i *metode*. Metode djeluju slično kao i funkcije no pišemo ih na drugi način. Ovdje ćemo upoznati jednu od metoda koja će nam pomoći u oblikovanju ispisa.

U prethodnoj smo točki vidjeli da oblikovanje složenijeg ispisa nije baš jednostavno. Želimo li ispisati malo složeniji tekst, moramo dobro paziti kako ćemo oblikovati ispisni string. Zbog toga je u inačicama *Pythona 3.x.x* uveden praktičniji način oblikovanja ispisnog stringa uporabom metode naziva `format()`.

```
>>> print('7 + 6 = 13')                                    #1
7 + 6 = 13
>>> print('{0} + {1} = {2}'.format(7, 6, 13))             #2
7 + 6 = 13
```

#1	Ispisni string napisali smo u obliku koji želimo vidjeti u ispisu. No, takvim oblikovanjem ispisa ispisujemo točno taj string.
#2	Ako želimo na mjestima gdje stoje brojevi 7, 6 i 13 napisati neke druge brojeve, onda u stringu ta mjesta možemo označiti redom brojevima (počevši brojem 0) unutar vitičastih zagrada. Iza takvog, novog stringa stavljamo točku i pozivamo metodu `format()`. Unutar zagrada metode `format()` redom ćemo napisati one vrijednosti koje želimo da se ispišu na mjestima gdje se unutar stringa nalaze vitičaste zagrade. U ovom primjeru napisali smo iste brojeve kao u naredbi (#1) i dobili jednaki ispis.

```
>>> print('{0} + {1} = {2}'.format(7, 6, 7 + 6))                    #3
```

#3 U metodi `format()` mogu se umjesto pojedinih vrijednosti napisati i aritmetički izrazi slično kao i u funkciji `print()`.

```
>>> a = 7; b = 6                                                    #4
>>> print('{0} + {1} = {2}'.format(a, b, a + b))                   #5
7 + 6 = 13
```

#4 Varijablama `a` i `b` pridružene su vrijednosti.

#5 U metodi `format()` se umjesto pojedinih vrijednosti i aritmetičkih izraza mogu napisati i imena varijabli.

```
>>> print('{0} + {1} = {1} + {0}'.format(a, b))                    #6
7 + 6 = 6 + 7
```

#6 U stringu koji prethodi metodi `format()` mogu se unutar vitičastih zagrada neki brojevi napisati više puta, odnosno na različitim pozicijama. Ti brojevi imaju vezu s redoslijedom vrijednosti unutar metode `format()`. U našem slučaju smo na taj način oblikovali ispis u kojem je vidljiv zakon komutativnosti.

```
>>> a = 15; b = 17                                                  #7
>>> print('{} + {} = {}'.format(a, b, a + b))                      #8
15 + 17 = 32
```

#7 Varijablama `a` i `b` pridružene su nove vrijednosti.

#8 U stringu koji prethodi metodi `format()` mogu se izostaviti brojevi unutar vitičastih zagrada. Neki se brojevi mogu napisati više puta. U tom slučaju *Python* će automatski vrijednosti iz metode `format()` redom ispisati na pozicijama vitičastih zagrada.