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

04
Prirodni i cijeli brojevi u programima, funkcije koje vraćaju vrijednosti

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

Cijeli brojevi u Pythonu, tip `int`

Naučili smo da će pri unosu brojeva posredstvom funkcije `input()` oni biti pohranjeni u radnom spremniku kao string te da ih funkcijom `int()` moramo prevesti u tip `int`. Tako možemo pisati:

```
>>> a = input('Upiši broj: ')
Upisati broj: -17
>>> a                                                    #1
'-17'
>>> b = input('Upiši broj: ')
Upisati broj: +18
>>> b                                                    #2
'+18'
>>> a + b                                                #3
'-17+18'
>>> a = int(a)
>>> a                                                    #4
-17
>>> b = int(b)
>>> b                                                    #5
18
>>> a + b                                                #6
1
>>>
```

Nakon utipkavanja `-17` u varijabli `a` nalazi se string `'-17'`(#1), a nakon utipkavanja `+18` u varijabli `b` string `'+18'`(#2). Operator nadovezivanja `+` u izrazu (#3) koji vrijedi za stringove nadovezat će ta dva stringa. Tek kada funkcijom `int()` pretvorimo stringove u brojeve, moći ćemo obaviti i aritmetičku operaciju zbrajanja i dobiti očekivani rezultat zbrajanja.

Pravila računanja s cijelim brojevima

U operacijama s cijelim brojevima korisna nam je **apsolutna vrijednost** broja. Apsolutna vrijednost negativnog broja dobiva se tako da se negativni predznak promijeni u pozitivni. Apsolutna vrijednost pozitivnog broja je taj isti broj. U *Pythonu* postoji funkcija `abs()` koja određuje apsolutnu vrijednost broja. Tako je:

```
>>> a = -11
>>> a
-11
>>> abs(a)
11
>>>
```

Iz matematike znamo osnovna pravila računanja s pozitivnim i negativnim brojevima, odnosno s cijelim brojevima:

```
>>> (+4) + (+7)
11
>>> (-4) + (-7)
-11
>>>
```

Kada pišemo cijele brojeve na papiru, često broj zajedno s predznakom stavljamo u zagrade, kao na primjer: (−7), (+3), (−10), kako bi se znak predznaka (+ ili −) razlikovao od operacije zbrajanja ili oduzimanja koje se označavaju tim istim znakovima. U *Pythonu* brojeve s predznakom također možemo pisati u zagradama, ali i ne moramo jer *Python* zna razlikovati znak predznaka od znaka kojim se označavaju operacije zbrajanja i oduzimanja.

Iz tog razloga kod pisanja naredbi u *Pythonu* možemo izostaviti zagrade i znak + ispred pozitivnih brojeva:

```
>>> 4 + 7
11
>>> -4 + -7
-11
>>>
```

Pri zbrajanju brojeva suprotnih predznaka možemo postupiti tako da nađemo razliku između apsolutnih vrijednosti brojeva (tako da od veće apsolutne vrijednosti oduzmemo manju) i ispred nje stavimo predznak broja koji ima veću apsolutnu vrijednost. *Python* taj postupak zna provoditi korektno:

```
>>> -4 + 7
3
>>> 4 + -7
-3
>>> 17 + -18
-1
```

Množenje i dijeljenje obavlja se tako da se operacije dijeljenja i množenja obave s apsolutnim vrijednostima brojeva i zatim se ispred rezultata stavi predznak + ako oba broja imaju isti predznak, odnosno predznak – ako brojevi koji sudjeluju u operaciji imaju različite predznake:

```
>>> 3 * -7
-21
>>> -3 * -7
21
>>> 15 // 3
5
>>> -15 // 3
-5
>>> 15 // -3
-5
>>> -15 // -3
5
```

Kao i u matematici, tako i operacije u *Pythonu* imaju svoje prioritete izračunavanja. Prvo se računaju operacije množenja i dijeljenja, a zatim zbrajanja i oduzimanja. Ako u nizu imamo više operacija tzv. istog prioriteta, operacije će se izvoditi slijeva udesno. Imamo li u izrazu zagrade, prvo će se izračunati izraz unutar zagrada.

```
>>> 3 * 4 + 7
19
>>> 3 + 4 * 7
31
>>> (3 + 4) * 7
49
>>> 3 - 6 + 7
4
>>> 3 * 4 // 5 * 7                                    #5
14
>>> (3 * 4) // (5 * 7)
0
>>> 29 // 4 % 2                                       #6
1
```

U primjeru (#5) prvo će se pomnožiti 3 i 4, zatim će se rezultat 12 cjelobrojno podijeliti sa 5 te će rezultat u tom trenutku biti 2, što će množenjem sa 7 dati rezultat 14. U primjeru (#6) prvo će se 29 cjelobrojno podijeliti sa 4 i tako dobiven rezultat, 7, primjenom operacije za ostatak cjelobrojnog dijeljenja dat će rezultat 1.

Funkcije koje vraćaju vrijednost, naredba `return`

Vlastite funkcije koje smo do sada pripremali nisu izračunavale neke vrijednosti koje bi se mogle rabiti izvan funkcije.

Međutim, upoznali smo neke ugrađene funkcije kao što su:

- funkcija `int()` koja ulazni string pretvara u broj tj. vraća vrijednost tipa `int`
- funkcija `str()` koja ulazni broj pretvara u string tj. vraća vrijednost tipa `str`
- funkcija `divmod()` koje za dva ulazna broja izračunava količnik i ostatak tj. vraća dva broja tipa `int`.

```
>>> a = '-17'                          #1
>>> b = int(a)                         #2
>>> b                                  #3
-17
>>> c = str(b)                         #4
'-17'                                  #5
>>> kol, ost = divmod(17, 5)           #6
>>> kol, ost                           #7
(3, 2)
```

Iz naredbi (#1), (#2) i (#3) vidimo da će funkcija `int()` vratiti vrijednost tipa `int` koja iznosi −17 i tu vrijednost će spremiti u varijablu `b`, dok iz naredbi (#4) i (#5) zaključujemo da funkcija `str()` vraća vrijednost tipa `str` i sprema je u varijablu `c`.

Funkcija `divmod()` u naredbi (#6) vraća dvije vrijednosti, količnik i ostatak, od kojih će se količnik pripisati varijabli `kol`, a ostatak varijabli `ost` u što se možemo uvjeriti pregledom sadržaja tih varijabli (#7).

I mi možemo definirati funkcije koje vraćaju neku vrijednost. Vraćanje vrijednosti obavlja se naredbom `return` (engl. vratiti). Naredba `return` okončava izvođenje funkcije i program se nastavlja izvoditi naredbom koja je napisana iza naredbe u kojoj je funkcija pozvana, ali će funkcija vratiti vrijednost izraza koji je napisan iza riječi `return`.

```
>>> def udvostruči(a):                                    #8
        b = 2 * a                                         #9
        return b                                          #10
>>>
>>> udvostruči(7)                                         #11
14
>>> udvostruči('abc')                                     #12
'abcabc'
>>>
```

Funkcija `udvostruči()` definirana je naredbama (#8), (#9) i (#10) te će vratiti dvostruku vrijednost ulaznog parametra ako je on tipa `int` ili dvaput ponovljeni string ako je ulazni parametar tipa `str`. To pokazuju rezultati izvođenja naredbi (#11) i (#12). U tijelu funkcije prvo se u pomoćnu vrijablu naredbom (#9) pohranila vrijednost koju funkcija treba vratiti, a zatim je u naredbi (#10) ime te varijable napisano iza riječi `return`.

```
>>> def utrostruči(a):                                    #13
        return 3 * a                                      #14
>>>
>>> utrostruči(7)                                         #15
21
>>> utrostruči('abc')                                     #16
'abcabcabc'
```

Funkcija `utrostruči()` (#13) definirana je bez pomoćne varijable te je u naredbi (#14) iza riječi `return` napisan izraz kojim se određuje povratna vrijednost. Uporabu te funkcije ilustriraju naredbe (#15) i (#16).

Promotrimo što će se dogoditi kada bismo u definiciji funkcije `udvostruči()` (#17) izostavili naredbu `return`.

```
>>> def udvostruči(a):                                    #17
        b = 2 * a                                         #18
>>>
>>> udvostruči(7)                                         #19
>>>                                                       #20
```

Naredbe (#17) i (#18) identične su kao naredbe (#8) i (#9) u prethodnoj definiciji funkcije `udvostruči()`, ali je sada izostavljena naredba (#10). Ovako napisana funkcija `udvostruči()` izračunat će dvostruku vrijednost ulaznog parametra, ali tu vrijednost neće vratiti na mjesto u programu s kojeg je pozvana (#19). Izračunana vrijednost će zbog toga nakon završetka izvođenja funkcije biti izgubljena. To se vidi po tome što *Python* nije ništa ispisao kao rezultat izvođenja funkcije `udvostruči()` (#20).

Poziv funkcije može se pojaviti u izrazu s desne strane naredbe pridruživanja i u tom slučaju se vrijednost koju funkcija vraća rabi za određivanje vrijednosti te desne strane. U najjednostavnijem slučaju kada s desne strane piše samo poziv funkcije, vrijednost koju funkcija vraća bit će pridružena varijabli čije ime piše s lijeve strane. Pogledajmo:

```
>>> c = udvostruči(7)                                    #21
>>> c
14
>>> d = utrostruči('abc')                                #22
>>> d
'abcabcabc'
```

Naredbama (#21) i (#22) pohranili smo u varijable s imenima `c` i `d` vrijednosti što su ih vratile funkcije `udvostruči()`, odnosno `utrostruči()`. Pohranjene vrijednosti možemo rabiti na razne načine kao primjerice na ovaj:

```
>>> a = 10                                               #23
>>> for i in range(10):                                  #24
        a = udvostruči(a)                                #25
        print(a, end=' ')                                #26

20 40 80 160 320 640 1280 2560 5120 10240
>>>
```

Vrijednost varijable `a` (#23) udvostručuje se u svakom prolazu kroz petlju (#24). Utvrdimo još jednom da naredba pridruživanja (#25) prvo određuje vrijednost s desne strane znaka pridruživanja s trenutnom vrijednošću varijable `a` i tek nakon toga novu izračunanu vrijednost pohranjuje u varijablu s imenom `a`. Već smo naučili kako niz od tih deset vrijednosti ispisujemo u istom redu primjenom opcijskog parametra `end` u funkciji `print()` (#26).

Moguće je definirati i funkcije koje vraćaju više od jedne vrijednosti. Tada ih sve treba napisati iza riječi `return` i odvojiti zarezima. Tako bismo umjesto ugrađene funkcije `divmod()` mogli napisati svoju vlastitu funkciju koja bi izračunala količnik i ostatak cjelobrojnog dijeljenja. Ona bi mogla izgledati ovako:

```
>>> def kol_ost(a, b):                                    #27
        return a // b, a % b                              #28

>>> kol_ost(17, 5)                                        #29
(3, 2)                                                    #30
>>> kol, ost = kol_ost(23, 7)                             #31
>>> kol
3
>>> ost
2
>>>
```

Naša se funkcija zove `kol_ost()` i ima dva ulazna parametra (#27). To su: djeljenik *a* i djelitelj *b*. Funkcija vraća dvije vrijednosti: količnik i ostatak dijeljenja (#28). Pozivom funkcije s nekim vrijednostima (#29) funkcija će vratiti par vrijednosti (#30). Kada se funkcija nalazi s desne strane naredbe pridruživanja, vraćene vrijednosti bit će pridružene varijablama koje stoje s lijeve strane znaka pridruživanja (#31).

Pisanje programa s funkcijama

Iz dosadašnjih jednostavnih primjera uvjerili smo se da se određeni programski zadatak može riješiti na različite načine. Kod složenijih će problema to još više doći do izražaja. Nekada je i teško izabrati najpovoljniji način rješavanja zadanog problema.

No, vrlo je važno da program koji napišemo bude što razumljiviji nama samima, ali i drugima. Programi koji se ne koriste samo za učenje programiranja, već rješavaju realne probleme, često se moraju mijenjati. Izmjene su potrebne zbog nadogradnje programa novim mogućnostima ili zbog ispravljanja pogrešaka. Taj se proces naziva održavanjem programa, a u njemu često sudjeluje veći broj osoba. Zbog toga ćemo sve programe koje budemo pisali nastojati oblikovati tako da budu pregledni i razumljivi.

Ustanovili smo već da je vrlo korisno programe osmisliti na takav način da se oni sastoje od funkcija koje rješavaju pojedine podzadatke.

Nadalje, ustanovili smo da je djelovanje pojedinih funkcija poželjno opisati dokumentacijskim tekstom kako bi se funkcije mogle rabiti s razumijevanjem.

Napišimo program za uvježbavanje zbrajanja. U prvi trenutak takav program se čini nepotrebnim jer u Pythonu zbrajanje možemo obaviti jednostavno uporabom operatora +. No, program će nam poslužiti za bolje razumijevanje djelovanja funkcija te dodatno ilustrirati formatiranje stringova te uporabu dokumentacijskih tekstova.

```
#Jednostavni program za uvježbanje zbrajanja - program_4_1.py

def tko_zadaje():
    '''Vraća se utipkano ime'''
    return input('Tko će zadavati zadatke? ')                    #1

def tko_rješava():
    '''Vraća se utipkano ime'''
    return input('Tko će rješavati zadatke? ')                   #2

def koliko_zadataka():
    '''Vraća se utipkani broj zadataka koji će se zadati'''
    return int(input('Koliko zadataka će trebati riješiti? n = ')) #3
def upis_pribrojnika(ime):
    '''Vraćaju se dvije utipkane vrijednosti prevedene u tip int'''
    print('{}, upiši vrijednost dvaju pribrojnika!'.format(ime))
    a = int(input('Upiši a = '))                                 #4
    b = int(input('Upiši b = '))                                 #5
    return a, b                                                  #6
```

```
def upis_pribrojnika(ime):
    '''Vraćaju se dvije utipkane vrijednosti prevedene u tip int'''
    print('{}, upiši vrijednosti dvaju pribrojnika!'.format(ime))
    a = int(input('Upiši a = '))                                    #4
    b = int(input('Upiši b = '))                                    #5
    return a, b                                                     #6

def upis_zbroja(ime):
    '''Traži utipkavanje zbroja prethodno utipkanih brojeva'''
    input('{}, upiši zbroj zadanih pribrojnika! '.format(ime))      #7

def ispisati_zbroj(a, b):
    '''Ispisuje se pribrojnike a, b i njihov zbroj a + b'''
    print('Ispravno rješenje je:\n{0} + {1} = {2}'.format(a, b, a + b)) #8

def main():
    ime_z = tko_zadaje()
    ime_r = tko_rješava()
    n = koliko_zadataka()
    for i in range(n):
        print(42 * '-')                                             #9
        a, b = upis_pribrojnika(ime_z)
        upis_zbroja(ime_r)
        ispisati_zbroj(a, b)

main()
```

Iz dokumentacijskih tekstova jednostavno je razabrati što program tj. pojedina funkcija radi, tako da nam dodatni komentari uglavnom nisu potrebni. Vidimo da funkcije `tko_zadaje()` *i* `tko_rješava()` *vraćaju utipkane stringove (#1) i (#2), dok funkcija* `koliko_zadataka()` *vraća broj tipa* `int` *(#3). Funkcija* `upis_pribrojnika()` *prihvaća dva broja (#4) i (#5) i vraća ih kao par brojeva (#6) dok funkcije* `upis_zbroja()` *i* `ispisati_zbroj()` *(#7) i (#8) ne vraćaju nikakve vrijednosti. U glavnoj funkciji uočimo naredbu (#9) koja crta crtu sastavljenu od četrdeset dva znaka* `'-'`.

Izvođenjem tog programa mogli bismo dobiti sljedeći ispis u interaktivnom sučelju:

```
Tko će zadavati zadatke? Ana
Tko će rješavati zadatke? Marija
Koliko zadataka će trebati riješiti? n = 3
----------------------------------------
Ana, upiši vrijednost dvaju pribrojnika!
Upiši a = 4
Upiši b = 7
Marija, upiši zbroj zadanih pribrojnika! 11
Ispravno rješenje je:
4 + 7 = 11
----------------------------------------
Ana, upiši vrijednost dvaju pribrojnika!
Upiši a = 37
Upiši b = 14
Marija, upiši zbroj zadanih pribrojnika! 53
Ispravno rješenje je:
37 + 14 = 51
```

Lako je uočiti da ovaj program ima mnoge nedostatke. U prvom je redu nepraktično zadavati zadatke utipkavanjem pribrojnika. Bilo bi praktično kada bi ih računalo moglo samostalno generirati, a od korisnika bi se onda zahtijevalo da samo upiše rješenje.

Nadalje, bilo bi korisno kada bismo ispravnost odgovora mogli odrediti programom, a ne usporedbom upisanog rezultata i ispisane izračunane vrijednosti kako to činimo pri izvođenju ovog programa. Ovdje smo sami morali ustanoviti da utipkana vrijednost 53 nije točna jer je 37 + 14 = 51.

Vidjet ćemo da oba ta nedostatka možemo odstraniti.

Generiranje nasumičnih brojeva

U 3. poglavlju upoznali smo se s nekim funkcijama iz modula `turtle` i naučili smo da je prije njihove uporabe modul trebalo aktivirati naredbom `import`. Iz modula `turtle` trebali smo više funkcija pa smo ih sve uvezli na sljedeći način:

```
>>> from turtle import *
>>>
```

U *Pythonu* postoji i modul s imenom `random` koji sadrži niz funkcija za obradu nasumičnih ili slučajnih podataka. Nas u ovom trenutku zanima samo jedna od tih funkcija. To je funkcija `randint()` koja pri svakom pozivu vraća jedan nasumično odabrani cijeli broj iz zadanog intervala brojeva. Zbog toga ćemo tu funkciju uvesti ovako:

```
>>> from random import randint
>>>
```

Pri svakom pozivu te funkcije ona će vratiti jednu nasumično odabranu vrijednost iz zadanog intervala vrijednosti. Želimo li generirati niz nasumičnih brojeva iz intervala od 1 do 9, napisat ćemo to na sljedeći način:

```
>>> randint(1, 9)
1
>>> randint(1, 9)
5
>>> randint(1, 9)
7
>>> randint(1, 9)
2
>>> randint(1, 9)
6
```

U svakom pozivu, funkcija će generirati jedan broj između granica napisanih u zagradi, uključujući i vrijednosti granica. Vidimo da je u svakom od pet poziva s jednakim argumentima, funkcija vratila različite vrijednost. U trenutku poziva funkcije `randint()` ne znamo koju će vrijednost funkcija vratiti, ali znamo da će vratiti vrijednost koja je unutar zadanih granica.

Interval u koji su uključene i granice naziva se zatvorenim intervalom i obilježava se uglatim zagradama. Tako možemo, primjerice, generirati 20 nasumičnih dvoznamenkastih brojeva iz intervala [10, 90]. To će biti brojevi u granicama od 10 do 99, uključujući i granice:

```
>>> for i in range(20):
        print(randint(10, 99), end=' ')

82 95 28 90 72 41 75 10 49 91 61 40 94 94 23 14 63 96 25 15
```

Svaki puta kada ponovimo gornju `for` petlju, dobit ćemo drukčiji niz nasumičnih brojeva.

Promijenit ćemo program za uvježbavanje zbrajanja iz primjera 4. 1. tako da se nasumično određuje broj zadataka koji će trebati riješiti dok se pribrojnici zadaju funkcijom `randint()`. *Taj bi program mogao izgledati ovako:*

```
#Program za uvježbavanje zbrajanja s nasumičnim brojevima - program_4_2.py

from random import randint

def koliko_zadataka():
    '''Vraća se nasumično određen broj zadataka iz intervala [1,8]'''
    return randint(1, 8)

def upis_granica():
    '''Funkcija vraća:
          dg -  utipkanu donju granicu nasumičnih brojeva
          gg -  utipkanu gornju granicu nasumičnih brojeva
     '''
    dg = int(input('Upiši dg = '))
    gg = int(input('Upiši gg = '))
    return dg, gg
def ispisati_zbroj(a, b):
    '''Ispisuje pribrojnike a, b i njihov zbroj a + b'''
    print('Ispravno rješenje je:\n{} + {} = {}'.format(a, b, a + b))
```

```
def main():
    n = koliko_zadataka()
    print('Bit će zadan sljedeći broj zadataka: {}'.format(n))
    dg, gg = upis_granica()
    print('Pribrojnici će biti iz intervala [{},{}]'.format(dg, gg))
    for i in range(n):
        print(42 * '-')
        a, b = randint(dg, gg), randint(dg, gg)
        int(input('Koliko je\n{} + {} = '.format(a, b)))
        ispisati_zbroj(a, b)

main()
```

Iz dokumentacijskih tekstova pojedinih funkcija jasno je što pojedine funkcije rade pa nije potrebno dodatno objašnjavanje.

Jednim od izvođenja ovog programa dobiven je sljedeći ispis:

```
Bit će zadan sljedeći broj zadataka: 3
Upisati dg = 10
Upisati gg = 99
Pribrojnici će biti iz intervala [10,99]
----------------------------------------
Koliko je
58 + 67 = 115
Ispravno rješenje je:
58 + 67 = 125
----------------------------------------
Koliko je
30 + 73 = 103
Ispravno rješenje je:
30 + 73 = 103
----------------------------------------
```

Ispis trećeg zadatka je izostavljen.

Vidimo da je u prvom zbroju upisano pogrešno rješenje. Naš je program ispisao ispravno rješenje tako da korisnik može vidjeti da je pogriješio.

Međutim, bilo bi jako korisno upozoriti korisnika na pogrešku i eventualno zatražiti ponovni upis rješenja. Zbog toga su nam u programskom jeziku potrebne naredbe kojima će se programski uspoređivati brojevi i na osnovi tih usporedbi odlučivati što treba poduzimati. Time ćemo se baviti u sjedećem poglavlju.