Osnove programiranja (Python) : priručnik za polaznike

Hruška, Marko; Domšić, Josip; Kurtović, Gorana; Kožul, Mia

Osnove programiranja (*Python*)

D450

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu inačicu priručnika izradio je autorski tim Srca u sastavu:

Autor: Marko Hruška

Recenzent: Josip Domšić

Urednica: Gorana Kurtović

Lektorica: Mia Kožul

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10000 Zagreb

edu@srce.hr

ISBN 978-953-8172-30-4 (meki uvez)
ISBN 978-953-8172-31-1 (PDF)

Verzija priručnika D450-20190517

Sadržaj

Uvod

Svrha ovoga tečaja jest upoznavanje s osnovama programskoga jezika *Python*. Prema mnogim studijama učenje programskoga jezika *Python* je izrazito jednostavno, ponajviše zahvaljujući tomu što ima jednostavnu sintaksu. Jedan od razloga zašto se danas sve više programira u *Pythonu* je jednostavna sintaksa.

Preduvjet za pohađanje i razumijevanje ovoga tečaja jest poznavanje osnova rada na računalu. Priručnik se sastoji od 9 poglavlja koja se odrađuju u četiri dana, po četiri sata dnevno, koliko traje tečaj. Na kraju svakog poglavlja nalaze se vježbe koje će polaznici prolaziti zajedno s predavačem. Mogući savjeti i zanimljivosti istaknuti su u okvirima sa strane.

Na ovom tečaju polaznici će naučiti osnovne koncepte programiranja u programskom jeziku *Python*. Tako naučeni osnovni koncepti na ovom tečaju primjenjivi su i na većinu ostalih programskih jezika, čime se polaznicima omogućava da nakon završenog tečaja s lakoćom svladavaju i ostale programske jezike. Naučeni osnovni koncepti omogućavaju lakše snalaženje prilikom nadograđivanja vlastitoga znanja nakon tečaja uz pomoć *web*-pretraživača i ostale literature. Nakon tečaja polaznici će posjedovati znanje za rješavanje lakših programskih zadataka u programskom jeziku *Python*.

Python je interpreterski programski jezik. Interpreterski programski jezici su jezici kod kojih se izvorni kôd izvršava direktno uz pomoć interpretera, tj. kod ovakvih tipova programskih jezika nema potrebe za kompajliranjem prije izvršavanja, tj. prevođenjem u izvršni oblik.

Programi pisani u programskom jeziku *Python* su kraći, a i za njihovo pisanje utrošak vremena je puno manji. *Python* programerima dopušta nekoliko stilova pisanja programa: strukturno, objektno orijentirano i aspektno orijentirano programiranje.

U ovom tečaju obrađeno je strukturno programiranje. Kod strukturnog programiranja glavni je naglasak za kontrolu programa na korištenju struktura poput funkcija, metoda, odluka, petlji. Ovakav način programiranja omogućava programerima da se program razlomi na manje logičke dijelove, tj. u funkcije te se tako povećava modularnost programskoga kôda koji je moguće višekratno iskoristiti. Ovakav pristup idealan je za kraće logičke programe, no kod velikih projekata nije dovoljno učinkovit i nakon nekog vremena programerima je sve teže razvijati nove komponente, tj. funkcionalnosti. Objektno orijentirano programiranje i aspektno orijentirano programiranje neće se obrađivati u ovom tečaju.

Napomena - info

Python je dobio ime po seriji: "Monty Python's Flying Circus".

Kako je *Python* interpreterski tip jezika, programi pisani u njemu se za posljedicu sporije izvršavaju za razliku od programa koji su pisani u programskim jezicima *C*, *C++*, *Java* itd. Zadnje stabilne verzije *Pythona* su: *Python 3.7.** i *Python 2.7.14*. U ovom tečaju koristit ćemo verziju *Python 3*.

Izradu programskoga kôda moguće je podijeliti na nekoliko koraka:

1. shvatiti problem koji je potrebno riješiti
2. rastaviti problem na manje dijelove (module)
3. prepoznati koji su programski elementi potrebni za rješavanje problema
4. povezati programske elemente u smislen algoritam
5. napisati program u odabranom programskom jeziku
6. testirati program, pronaći rubne slučajeve te popraviti eventualne greške.

U priručniku važni pojmovi pisani su **podebljano**. Ključne riječi, imena varijabli, funkcija, metoda i ostale programske konstrukcije pisane su drugačijim fontom od uobičajenog, na primjer: `print("Hello World!")`. Nazivi na engleskom jeziku pisani su *kurzivom* i u zagradi, na primjer: "varijabla (engl. *variable*)"**.**

Programski kôd pisan je na sljedeći način:

U prvom dijelu bit će napisan programski kôd.

```
for i in range(2):
    print("Hello World!")
```

```
Izlaz:
    Hello World!
    Hello World!
```

U drugom dijelu bit će prikazan dobiven izlaz programskoga kôda.

1. Uvod u programiranje

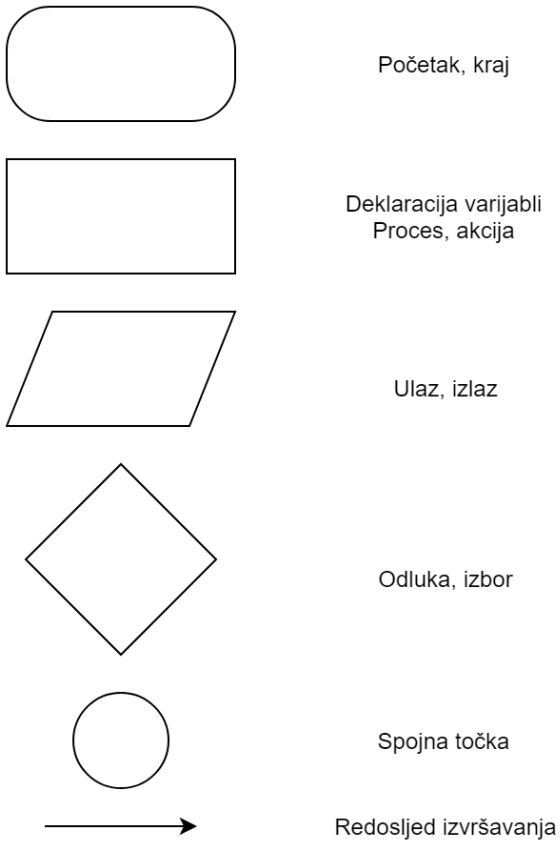
Po završetku ovog poglavlja polaznik će moći:

- *objasniti kako dijagram toka prikazuje algoritam*
- *nacrtati dijagram toka za jednostavan algoritam*
- *tumačiti pseudo-kôd*
- *napisati pseudo-kôd za jednostavan algoritam*
- *pripremiti radno okruženje za rad s programskim jezikom Python.*

1.1. Dijagrami toka

Dijagram toka je grafički prikaz algoritma. Dijagram toka sastoji se od niza elemenata koji su međusobno povezani strelicama čiji smjer diktira smjer realizacije programa.

Najčešće korišteni elementi:



Dijagrami toka pomažu programerima u vizualizaciji algoritma, ali i za bolje shvaćanje problema kojeg rješavaju.

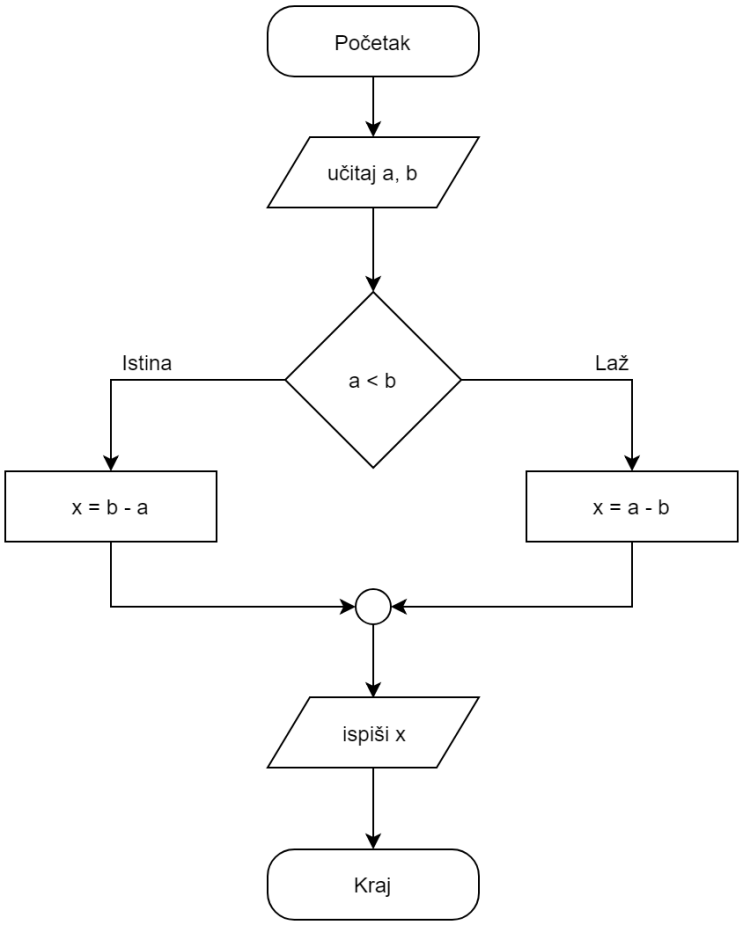
***Python* – verzije**

Verzija *Pythona* je označena s A.B.C. Svako od ta tri slova označava promjene koje su se dogodile prema važnosti. Slovo C označava sitne promjene, slovo B označava veće promjene, dok slovo A označava velike promjene.

Primjer: program učitava dvije vrijednosti (učitane vrijednosti spremaju se u varijable). Od veće vrijednosti oduzima se manja učitana vrijednost. Tako dobiveni rezultat koji se ispisuje uvijek je veći ili jednak nuli (>= 0).

Varijable – ovaj pojam detaljnije je objašnjen u poglavlju *2. Osnove programskog jezika Python*. Varijable su dijelovi programa koji se koriste kako bi se spremile učitane vrijednosti. Vrijednosti se nekoj varijabli pridružuju pomoću matematičkog operatora jednako tj. "=". Ime varijable u koju želimo spremiti neku vrijednost uvijek se nalazi s lijeve strane znaka "=" dok se s desne strane nalazi vrijednost koju želimo spremiti u varijablu.

Obrazloženje rješenja*:* priloženi dijagram toka prikazuje algoritam koji učitava dva broja, učitani brojevi spremaju se u varijable `a` i `b`. Nakon učitavanja vrijednost slijedi odluka o tome je li vrijednost varijable `a` manja od vrijednosti varijable `b`. Ako je uvjet provjere istinit, tada se u varijablu `x` sprema rezultat oduzimanja `b - a`, dok se u suprotnom primjeru u varijablu `x` sprema rezultat oduzimanja `a - b`. Nakon izvršenog oduzimanja rezultat spremljen u varijablu `x` ispisuje se na ekran.



Napomena

Opisivanje algoritma dijagramom toka je prikladno za kraće / lakše algoritme te za početnike kako bi lakše vizualizirali problem kojeg rješavaju.

Zaključak: priloženi algoritam radi matematičku operaciju oduzimanja čiji rezultat nikada neće biti negativan, već će uvijek biti `x >= 0`.

1.2. Pseudo-kôd

Kako su dijagrami toka prilikom opisivanja složenih algoritama izrazito nepregledni, bolja zamjena za njih je pseudo-kôd. Pseudo-kôd koristi termine govornoga jezika i on je puno prikladniji za opis složenijih algoritama. Prilikom opisa nekog algoritma pseudo-kôd u svojem sadržaju koristi izraze kojima se ljudi svakodnevno koriste. Na autoru pseudo-kôda je da odluči koliko će detaljno ići riječima opisivati neki algoritam (treba pronaći zlatnu sredinu jer se s previše detalja gubi smisao opisivanja algoritma pseudo-kôdom, a opet ako imamo premalo detalja, algoritam može biti nejasan, osobito početnicima). Pseudo-kôd mora biti napisan općenito tako da se ne vide detalji implementacije koji su ovisni o nekom konkretnom programskom jeziku, na primjer *Pythonu*.

Napomena

U knjigama se složeniji algoritmi opisuju pomoću pseudo- kôda.

Kod pseudo-kôda za pridruživanje vrijednosti varijablama koristi se oznaka "`:=`" i ostale standardne matematičke operacije.

U nastavku je prikazan pseudo-kôd čija je logika identična logici algoritma prikazanog u prethodnom poglavlju.

```
početak

učitaj(a, b)

ako je a < b tada
      x := b - a
inače
      x := a - b

ispiši(x)

kraj
```

1.3. Priprema radnog okruženja

Na stranici https://www.python.org/ skinuti i instalirati zadnju verziju *Python 3.7.x*. Ako instalacija *Python 3.7.x*. nije dostupna, skinuti zadnju dostupnu instalaciju *Python 3.x*.

Prilikom instalacije koristiti unaprijed postavljene opcije, tj. nije potrebno ništa mijenjati.

Pokrenuti *IDLE* koji dolazi sa samom instalacijom *Pythona*.

Start → Python 3.7 → IDLE (Python 3.7 32-bit)

Gore otvoreni program možemo smatrati kao naredbenim retkom gdje se naredbe izvršavaju jedna po jedna. Ovakav način pokretanja programskoga kôda napisanog u *Pythonu* obično se koristi kada na brzinu želimo izvršiti nekoliko naredbi. Kod ovakvog pristupa programiranju, program koji napišemo je "jednokratan" što znači da ako ponovo želimo pokrenuti isti skup programskih linija, sve te linije moramo ponovo napisati ili koristiti metodu kopiraj-zalijepi (engl. *Copy-Paste*). Ako želimo napisati program koji se sastoji od više linija, koji

lagano možemo uređivati kao i svaki drugi dokument na računalu, za to je najprikladnije da preko *Python IDLE-a* otvorimo novi dokument u kojem ćemo pisati programski kôd:

U programu koji smo prethodno otvorili: *IDLE (Python 3.7 32-bit)* odaberemo:

File → New file

Primijetimo da nam se sada otvorio prozor za unos teksta. U ovom prozoru pišemo programski kôd. Nakon što želimo pokrenuti napisani program to možemo napraviti pritiskom na tipku: F5 ili pak kroz izbornik:

Run → Run module

Nakon što pokrenemo napisani programski kôd, rezultat izvođenja programa pojavit će nam se u prozoru koji smo prvo otvorili, tj. *IDLE (Python 3.7 32-bit)*.

Primjer: Hello World

U prozoru za unos teksta napišemo naredbu: `print("Hello World!")` te pokrenemo program. Nakon pokretanja programa rečenica `"Hello World!"` mora se pojaviti u *IDLE-u*. Naredba `print()` nam služi za ispis teksta i ona je detaljnije objašnjenja u poglavlju *5. Ulazno izlazne funkcije*. `"Hello World!"` je niz znakova (engl. *String*). Nizovi znakova pojašnjeni su u poglavlju *2. Osnove programskog jezika Python*.

Za sada uzmimo naredbu `print()` kao naredbu koju pozivamo kada želimo ispisati neku proizvoljnu vrijednost, a vrijednost koju želimo ispisati stavljamo u oble zagrade.

1.4. Vježba: Dijagram toka, pseudo-kôd, prvi program

1. Kreirajte dijagram toka koji učitava tri broja. Ispišite sumu (a + b + c) ili umnožak unesenih brojeva (a * b * c) ovisno o tome koja je od tih dviju vrijednosti veća.

   Primjer: učitani brojevi su 1, 2, 2, dijagram toka mora ispisati brojku 5 zato što suma unesenih brojeva (1 + 2 + 2) iznosi 5, a umnožak unesenih brojeva (1 * 2 * 2) iznosi 4, tj. suma unesenih brojeva je veća.
2. Napravite prethodni zadatak pomoću pseudo-kôda.
3. Kreirajte dijagram toka koji učitava dva broja. Ispišite učitane vrijednosti u uzlaznom poretku.
   Primjer: za učitane brojeve 50 i 10, ispis mora izgledati: "10, 50".
4. Napravite prethodni zadatak pomoću pseudo-kôda.
5. Kreirajte dijagram toka koji učitava dva broja. Ispišite količnik učitanih brojeva (rezultat dijeljenja učitanih brojeva). U slučaju dijeljenja s nulom, ispišite da to nije ispravno.
6. Napravite prethodni zadatak pomoću pseudo-kôda.
7. Pokrenuti *Python IDLE* te ispisati proizvoljni niz znakova.

1.5. Pitanja za ponavljanje: Dijagram toka, pseudo-kôd

1. Kod dijagrama toka koji oblik služi za upis ulaznih / ispis izlaznih vrijednost?
2. Koja je prednost pseudo-kôda pred dijagramom toka?

2. Osnove programskoga jezika *Python*

Po završetku ovog poglavlja polaznik će moći:

- *opisati kôd komentarima*
- *definirati i koristiti varijable*
- *odrediti vrstu varijable*
- *koristiti aritmetičke operatore, operatore usporedbe te ugrađene funkcije i metode.*

2.1. Komentari

Kao i u svakom programskom jeziku tako i u *Pythonu* postoje komentari koji programerima olakšavaju pisanje programskog kôda. Korisno je opisati što se kojim od dijelova programskoga kôda želi postići. Takvi opisi zovu se komentari.

Komentari u *Pythonu* počinju znakom "hash" - `#`. Nakon znaka za početak komentara, on se proteže do kraja linije.

Kako su komentari tu da pojasne dio kôda, oni se ne interpretiraju, tj. prilikom izvođenja programa se ignoriraju.

Komentari

Preporuča se kompliciranije dijelove kôda komentirati. Ako je programski kôd kvalitetno iskomentiran, programer koji čita takav kôd puno će brže shvatiti logiku programa.

```
# Ovo je prvi komentar
var1 = 1  # ovo je drugi komentar
          # ... a ovo je treći komentar!
text = "# Ovo nije komentar jer se nalazi unutar
navodnika."

print(var1)
print(text)
```

```
Izlaz:
     1
     # Ovo nije komentar jer se nalazi unutar
     navodnika.
```

Uz komentare koji se protežu od znaka "hash" - `#` pa do kraja linije, postoje komentari pomoću kojih možemo napisati neki opisni tekst kroz nekoliko linija ili pak iskomentirati cijeli odsječak programskog kôda. Komentari koji se protežu kroz nekoliko programskih linija počinju i završavaju s ''' ili """ (tri jednostruka ili tri dvostruka navodnika). Bitno je naglasiti da znak za početak komentara koji se proteže kroz nekoliko programskih linija mora biti napisan na početku retka, tj. ispred znaka za početak komentara u nekoj liniji ne smije pisati nikakav programski kôd.

```
var1 = 1
print(var1)
''' Ovo je komentar
koji se proteže
kroz nekoliko
programskih linija '''
```

```
var1 = 1
print(var1)
""" Ovo je komentar
koji se proteže
kroz nekoliko
programskih linija """
```

Nazivi varijabli

Nazivi varijabli morali bi logički predstavljati vrijednost koja je u varijabli spremljena. Na primjer, ako se u varijabli nalazi suma brojeva, prikladnije je takvu varijablu nazvati `suma`, a ne `umnozak`.

2.2. Varijable

Varijable su dijelovi programa koji se koriste kako bi se spremile vrijednosti. Tim vrijednostima (informacijama) pristupa se iz raznih dijelova računalnih programa. Svaka varijabla određena je svojim imenom i memorijskom lokacijom na kojoj je zapisana određena vrijednost.

Najjednostavniji opis pojma varijabla bio bi da je varijabla "kutija" u koju se spremaju stvari (u našem slučaju vrijednosti) te se prema potrebi te stvari mogu uzeti (koristiti, pridruživati drugim varijablama), mijenjati s drugim stvarima (s drugim vrijednostima) itd. U varijable je moguće spremiti svakakve vrijednosti, no u ovom će se dijelu tečaja u varijable spremati samo brojevi radi lakšeg razumijevanja gradiva.

Vrijednosti se nekoj varijabli pridružuju pomoću matematičkog operatora "=". Ime varijable u koju želimo spremiti neku vrijednost uvijek se nalazi s lijeve strane znaka "=" dok se s desne strane nalazi vrijednost koju želimo spremiti u varijablu.

U donjem primjeru možemo vidjeti da se varijabli čije je ime `mojaVarijabla` pridružuje vrijednost 5. U drugoj liniji pozivom funkcije `print(mojaVarijabla)` *Python* ispisuje vrijednost koja je spremljena u varijabli imena `mojaVarijabla`, iz tog razloga se na zaslon ispisuje vrijednost 5.

```
mojaVarijabla = 5
print(mojaVarijabla)
```

```
Izlaz:
     5
```

Napomena

Prilikom kreiranja imena varijabli, potrebno je dobro razmisliti o najboljem imenu za neku varijablu, jer ako programer pametno odabere ime, to će mu omogućiti lakše pisanje programskoga kôda, a posljedično i lakše snalaženje u programskom kôdu. Posebice, ako će taj isti kôd ići čitati, na primjer, par tjedana nakon što je programski kôd napisan.

Primjer: ako je potrebno u varijablu spremiti trenutnu godinu, tada je tu varijablu pametnije nazvati `trenutnaGodina`, a ne `trenutniMjesec`.

2.3. Aritmetički operatori

Python podržava nekoliko unaprijed definiranih aritmetičkih operatora za rad s brojevima. U nastavku je prikazana tablica u kojoj se nalazi popis aritmetičkih operatora koji se mogu koristiti.

Operator	Namjena	Primjer	Rezultat a = 10;b = 5
+	Zbrajanje (engl. *addition*)	x = a + b	x = 15
−	Oduzimanje (engl. *subtraction*)	x = a – b	x = 5
*	Množenje (engl. *multiplication*)	x = a * b	x = 50
/	Dijeljenje (engl. *division*)	x = a / b	x = 2
%	Ostatak dijeljenja (engl. *modulus*)	x = a % b	x = 0
**	Potenciranje (engl. *power*)	x = a ** b	x = 100000
//	Cjelobrojno dijeljenje (engl. *flor division*)	x = a // b	x = 2

Također, uz gore navedene aritmetičke operatore u programskim jezicima, pa tako i u *Pythonu*, postoji skraćena verzija operatora pridruživanja. U nastavku je prikazana tablica tih operatora.

Skraćeni operatori

Korištenjem skraćenih operatora povećava se čitljivost programskoga kôda.

Operator	Primjer	Osnovni oblik	Rezultat a = 10; b = 5
+=	a += b	a = a + b	a = 15
−=	a -= b	a = a – b	a = 5
*=	a *= b	a = a * b	a = 50
/=	a /= b	a = a / b	a = 2
%=	a %= b	a = a % b	a = 0
**=	a **=b	a = a ** b	a = 100000
//=	a //= b	a = a // b	a = 2
=	*a* = *b*	a = b	a = 5

Kao što se može vidjeti iz gornje tablice, skraćeni operatori omogućavaju kraći zapis jedne te iste stvari koja se u svim slučajevima može odraditi s osnovnim aritmetičkim operatorima. Uzmimo za primjer operator "+=", ako je varijablu `a`, potrebno uvećati za vrijednost varijable `b`, sa skraćenim aritmetičkim operatorom to bi se zapisalo kao `a += b`. To je identičan izraz kao `a = a + b`, zapisan na drugačiji način. Pomoću ovih operatora programer može pojednostaviti izgled programskoga kôda.

2.4. Operatori usporedbe

Operatori usporedbe uspoređuju vrijednosti s lijeve i s desne strane operatora te odlučuju o vrijednosti koju će vratiti. Operatori usporedbe vraćaju vrijednosti 0 (netočno, engl. *False*) ili 1 (točno, engl. *True*).

Operator	Opis	Primjer	Rezultat a = 5; b = 10	Rezultat a = 5; b = 5
==	Usporedba jednakosti	a == b	False	True
!=	Usporedba nejednakosti	a != b	True	False
>	Strogo veće od	a > b	False	False
<	Strogo manje od	a < b	True	False
>=	Veće ili jednako od	a >= b	False	True
<=	Manje ili jednako od	a <= b	True	True

Operatori usporedbe najčešće se koriste za kontrolu toka programa, tj. u uvjetima provjere točnosti rezultata logičkih operatora.

2.5. Logički operatori i logički izrazi

Logički operatori koriste se za kreiranje složenijih logičkih izraza. Za kreiranje složenijih izraza koriste se tri logička operatora, a to su:

- Operator `and` – operacija I (konjunkcija)
- Operator `or` – operacija ILI (disjunkcija)
- Operator `not` – operacija NE (negacija).

Napomena

Sintaksa označavanja logičkih operatora u programskom jeziku *C*:

Operator I

```
A && B
```

Operator ILI

```
A || B
```

Operator NOT

```
!A
```

Operator	Opis
`and`	Operacija I, konjunkcija
`or`	Operacija ILI, disjunkcija
`not`	Operacija NE, negacija

Tablica stanja operatora I (`and`):

A	B	A and B
False	False	False
True	False	False
False	True	False
True	True	True

Operator `I` daje rezultat `True` samo u jednom slučaju, a to je kada oba operanda imaju vrijednost `True`.

Tablica stanja operatora ILI (`or`):

A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

Operator `ILI` daje rezultat `True` u slučaju da barem jedan od operanada ima vrijednost `True`, tj. operator `ILI` daje vrijednost `False` samo u slučaju kada svi operandi imaju vrijednost `False`.

Tablica stanja operatora NE (`not`):

A	not A
False	True
True	False

Operator `NE` daje invertiranu vrijednost.

```
>>> False and False
False
>>> True and False
False
>>> False and True
False
>>> True and True
True
>>> False or False
False
>>> True or False
True
>>> False or True
True
>>> True or True
True
>>> not False
True
>>> not True
False
>>> not 0
True
>>> False And False
SyntaxError: invalid syntax
>>> False AND False
SyntaxError: invalid syntax
>>> FALSE and False
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    FALSE and False
NameError: name 'FALSE' is not defined
```

Kod složenijih logičkih izraza moguće je istovremeno kombinirati više vrsta operacija: aritmetička operacija, operatori usporedbe, logička operacija.

Redoslijed izvođenja operacija dan je sljedećom tablicom:

Redoslijed	Operacija
1.	Aritmetička operacija
2.	Operatori usporedbe
3.	Logička operacija

Napomena

Ako ne želimo razmišljati kakav je redoslijed izvođenja operacija (aritmetičkih operacija, operatora usporedbe, logičkih operacija) tada možemo koristiti zagrade.

```
>>> 5 + 8 < 20
True
>>> 5+ 8 > 20 / 2
True
>>> (5 + 8) > (20 / 2)
True
>>> (5 + 8) > (20 / 2) and (100 /2) > 10
True
>>> ( (5 + 8) > (20 / 2) ) and ( (100 /2) > 10 )
True
```

2.6. Tipovi brojeva – `int`, `float`, `complex`

Python ima tri osnovna tipa brojeva:

- cijeli brojevi
- decimalni brojevi
- kompleksni brojevi.

Kao dodatak tu se može uvrstiti i **boolean** tip podataka koji je podskup cijelih brojeva (boolean tip podataka može poprimiti vrijednosti 0 ili 1, tj. istina ili laž).

Preciznost decimalnih brojeva izravno je vezana za računalo na kojem se program izvršava, tj. ako isti program izvršavamo na 2 različita tipa računala postoji mogućnost da će preciznost decimalnoga dijela broja biti različita.

Kompleksni brojevi sastoje se od realnog i imaginarnog dijela, svaki od njih prikazan je jednim decimalnim brojem. Kako bi se dohvatio realni i/ili imaginarni dio iz kompleksnog broja spremljenog u varijabli `z`, potrebno je koristiti `z.real` i `z.imag`.

```
z = complex(1.11, 2.22)
print(z)
print(z.real)
print(z.imag)
```

```
Izlaz:
      (1.11+2.22j)
      1.11
      2.22
```

2.7. Nizovi znakova

Kako bi se u varijable mogao spremati tekst, tj. znakovni niz, koristi se tip podataka **str**, tj. **string**.

Niz znakova (engl. *String*) može se zapisati na nekoliko načina.

Uporabom:

- jednostrukih navodnika

  `'Dozvoljavaju unutar niza "dvostruke" navodnike'`

```
nizZnakova = 'Ovo je neki "niz" znakova!'
print(nizZnakova)
```

```
Izlaz:
     Ovo je neki "niz" znakova!
```

Navodnici u *C-u*

U programskom jeziku *C*, postoji velika razlika između korištenja jednostrukih i dvostrukih navodnika. U jednostruke navodnike se stavlja jedno slovo, dok se u dvostruke navodnike stavlja niz znakova.

- dvostrukih navodnika

  `"Dozvoljavaju unutar niza 'jednostruke' navodnike"`

```
nizZnakova = "Ovo je neki 'niz' znakova!"
print(nizZnakova)
```

```
Izlaz:
     Ovo je neki 'niz' znakova!
```

Python tretira jednostruke navodnike identično kao i dvostruke navodnike.

- trostrukih navodnika

  `'''Tri jednostruka''',"""Tri dvostruka"""`

Trostruki navodnici omogućavaju protezanje niza znakova kroz nekoliko linija. Svi uneseni razmaci uključeni su u niz znakova.

```
nizZnakova = '''Prva linija
Druga linija
    Treca linija s razmakom ispred'''
print(nizZnakova)
```

```
Izlaz:
     Prva linija
     Druga linija
         Treca linija s razmakom ispred
```

Dohvaćanje vrijednosti unutar niza znakova

Kod nizova znakova svaki znak ima svoju poziciju, pozicija na kojoj se neko slovo nalazi zove se **indeks**. Ako se želi dohvatiti znak koji je prvi u nizu, na primjer u riječi "Hello!" želi se dohvatiti znak "H", to slovo u logici programiranja nema poziciju 1, već ima poziciju, tj. indeks 0.

Ako se želi dohvatiti element na nekom točno određenom indeksu, to se radi tako da nakon imena varijable u kojoj se nalazi niz znakova napišu uglate zagrade i u njima indeks pozicije s koje se želi dohvatiti slovo. Također, postoji konvencija preko koje se dohvaćaju elementi od nekog

do nekog indeksa. Ako se želi dohvatiti znakove od, na primjer, 6. do 9. (slova "Worl" u frazi "Hello World!") indeksa, to će se napraviti tako da se napiše `nizZnakova[6:10]`.

```
nizZnakova = 'Hello World!'
print(nizZnakova[0])
print(nizZnakova[:5])
print(nizZnakova[6:10])
```

```
Izlaz:
      H
      Hello
      Worl
```

U nastavku su popisani operatori za rad s nizovima znakova.

Operator	**Opis**	**Rezultat** a = "abc", b = "def"
`+`	Spajanje	a + b = abcdef
`*`	Ponavljanje	a * 2 = abcabc
`[]`	Dohvaćanje vrijednosti na indeksu	a[0] = a
`[:]`	Vraća dio niza omeđenog indeksima	a[0:2] = ab
`in`	Provjerava nalazi li se dani znak u nizu	'c' in a => TRUE
`not in`	Provjerava da li se dani znak NE nalazi u nizu	'H' not in a => TRUE

2.8. Ugrađene funkcije i metode

Budući da postoji jako puno ugrađenih funkcija i metoda, gotovo ih je nemoguće sve popisati. U ovom poglavlju bit će opisane samo najčešće korištene funkcije i metode za rad s brojevima i nizovima znakova.

U nastavku slijedi popis najčešće korištenih funkcija za rad s brojevima.

Naziv	**Opis**	**Rezultat** x = 10	**Rezultat** x = 5.5	**Tip**
`int(x)`	Pretvara vrijednost `x` u cijeli broj	10	5	Funkcija
`float(x)`	Pretvara vrijednost `x` u decimalni broj	10.0	5.5	Funkcija
`complex(x)`	Pretvara vrijednost `x` u kompleksni broj	(10 + 0j)	(5.5+0j)	Funkcija
`hex(x)`	Pretvara vrijednost `x` u heksadecimalni zapis	0xa	-	Funkcija
`oct(x)`	Pretvara vrijednost `x` u oktalni zapis	0o12	-	Funkcija

Napomena – razmaci u kôdu

Razmaci napisani u kôdu ne utječu na izvršavanje kôda, tj. ne prikazuju se u rezultatima. Preporuča se pisanje razmaka kako bi se što više povećala čitljivost i preglednost programskoga kôda.

```
a = 10
b = 5.5

print(int(b))
print(float(a))
print(complex(a))
print(hex(a))
print(oct(a))
```

```
Izlaz:
      5
      10.0
      (10+0j)
      0xa
      0o12
```

U nastavku slijedi popis najčešće korištenih funkcija i metoda za rad s nizovima znakova. U donjoj tablici u stupcu rezultat bit će prikazan rezultat nakon što se pripadajuća funkcija ili metoda izvrši nad nizom znakova: `nizZnakova = "hello WORLD"`.

Naziv	Opis	Rezultat	Tip
`capitalize()`	U danom nizu prvo slovo stavlja u veliko slovo, a sva ostala slova u mala.	Hello world	Metoda
`len(nizZnakova)`	Vraća duljinu niza znakova.	11	Funkcija
`lower()`	Pretvara cijeli niz znakova u mala slova.	hello world	Metoda
`upper()`	Pretvara cijeli niz znakova u velika slova.	HELLO WORLD	Metoda
`title()`	Sve riječi počinju velikim slovom, a ostatak riječi napisan je malim slovima.	Hello World	Metoda
`lstrip()`	Miče sve praznine s lijeve strane niza.	-	Metoda
`rstrip()`	Miče sve praznine s desne strane niza.	-	Metoda
`strip()`	Miče sve praznine s desne i lijeve strane niza.	-	Metoda
`str()`	Prima objekt i vraća niz znakova.	-	Funkcija

```
nizZnakova = "hello WORLD"

print(nizZnakova.capitalize())
print(len(nizZnakova))
print(nizZnakova.lower())
print(nizZnakova.upper())
print(nizZnakova.title())
```

```
a = 123
b = str(a)
print(type(a))
print(type(b))
```

```
Izlaz:
      Hello world
      11
      hello world
      HELLO WORLD
      Hello World
      <class 'int'>
      <class 'str'>
```

U niže navedenom primjeru programskoga kôda prikazano je korištenje metoda: `lstrip()`, `rstrip()` i `strip()`. Kao što je navedeno u gornjoj tablici te metode miču praznine s lijeve i/ili desne strane niza znakova. Sivom bojom su označene praznine (razmaci).

```
nizZnakova = "    hello WORLD    "

print(nizZnakova)
print(nizZnakova.lstrip())
print(nizZnakova.rstrip())
print(nizZnakova.strip())
```

```
Izlaz:
      ----hello WORLD----
      hello WORLD----
      ----hello WORLD
      hello WORLD
```

Razlika i način pozivanja između funkcija i metoda:

- Funkcija je dio kôda koji se poziva preko imena funkcije i podaci (vrijednosti) se u nju prenose eksplicitno. Na temelju primljenih podataka funkcija će napraviti obradu i prema potrebi, tj. ovisno o implementaciji vratiti povratnu vrijednost.

  Funkcija se poziva na način:

  ```
  imeFunkcije(<argumenti>)
  ```

- Metoda je dio kôda koji se također poziva preko svojeg imena, no ona je povezana s objektom (varijablom) na temelju kojega se metoda poziva, prijenos vrijednosti se vrši implicitno. Za potrebe ovoga tečaja možemo smatrati da su metode identične funkcijama. Ovaj tečaj neće se baviti detaljnijim opisom razlika između funkcija i metoda.

  Metoda se poziva na način (objekt za sada možemo smatrati varijablom):

  ```
  objekt.imeMetode()
  ```

2.9. Vježba: Rad s varijablama

1. Kreirajte dvije varijable i pridijelite im neke vrijednosti te uz pomoć aritmetičkih operatora (*zbrajanje, oduzimanje, množenje, dijeljenje, ostatak dijeljenja, potenciranje, cjelobrojno dijeljenje*) ispišite rezultate.
2. Kreirajte dvije brojčane varijable te uz pomoć skraćenih aritmetičkih operatora (*zbrajanje, oduzimanje, množenje, dijeljenje, ostatak dijeljenja, potenciranje, cjelobrojno dijeljenje*) ispišite rezultate. *Napomena*: nakon svakog korištenja skraćenog aritmetičkog operatora potrebno je varijablu čija se vrijednost mijenja postaviti na početnu vrijednost.
3. Napišite program koji će u varijable `a` i `b` spremiti dva dvoznamenkasta broja. U varijablu `a` pohranite zadnju znamenku broja koji se nalazi u varijabli `b`, a u varijablu `b` pohranite zadnju znamenku broja koja se nalazi u varijabli `a`. Ispišite sadržaj varijabli `a` i `b`.
   Napomena: 159%10 = 9, 159%100 = 59.
4. Kreirajte dvije brojčane varijable te ispišite rezultat operatora usporedbe (*usporedba jednakosti, nejednakosti, strogo veće, strogo manje, veće ili jednako, manje ili jednako*).
5. Kreirajte dvije varijable i pridijelite im vrijednosti `True` ili `False`. Ispišite rezultate logičkih operatora (*`and`*, *`or`*, *`not`*) za dvije prethodno kreirane varijable.
6. Prethodni zadatak uredite tako da isprobate korištenje komentara.
7. Napišite program u kojem su navedene varijable *`a=5, b=10, c=15, d=21`*. Program mora vratiti aritmetičku sredinu danih brojeva. Rezultat spremite u varijablu jer će se vrijednost koristiti i u sljedećem zadatku. Aritmetička sredina računa se tako da se sve dane vrijednosti zbroje te se rezultat podijeli s brojem varijabli.
8. Iskoristite prethodni zadatak te dobiveni rezultat pretvorite u cijeli broj i tu vrijednost spremite u varijablu. Nad cjelobrojnom aritmetičkom sredinom koju ste prethodno spremili u varijablu napravite kvadriranje i ispišite dobiveni rezultat.
9. Korištenjem skraćenih aritmetičkih operatora rezultat dobiven iz prethodnog zadatka pomnožite sa 100 i ispišite dobivenu vrijednost.
10. Pomoću operatora usporedbe provjerite je li broj koji ste dobili u prethodnom zadatku manji od 500. Ispišite rezultat usporedbe (ispis će biti vrijednosti "`True`" ili "`False`").
11. Spremite sljedeći niz znakova u varijablu i ispišite taj niz: "*I'm from Croatia!*".
12. Iz niza znakova u prethodnom zadatku dohvatite i ispišite riječ "*from*".
13. Kreirajte dvije varijable i pridružite im neki niz znakova. Isprobajte kako funkcioniraju operatori za rad s nizovima znakova (*spajanje, ponavljanje, dohvaćanje vrijednosti na indeksu, vraćanje dijela niza omeđenog indeksima, provjera nalazi li se dani znak u nizu, provjera NE nalazi li se dani znak u nizu*).

Napomena

Zadatak se rješava uz pomoć aritmetičkog operatora %.

14. Kreirajte tri brojčane varijable te pomoću njih isprobajte funkcije za rad s brojevima (`int()`, `float()`, `complex()`, `hex()`, `oct()`).
15. Kreirajte varijablu te u nju spremite neki niz znakova. Tu varijablu upotrijebite kako biste isprobali korištenje funkcija i metoda za rad s nizovima znakova (`capitalize()`, `len()`, `lower()`, `upper()`, `title()`, `lstrip()`, `rstrip()`, `strip()`).
16. * U varijablu upišite neki proizvoljni niz znakova. Nad varijablom pozovite odgovarajuću funkciju koja će vratiti duljinu upisanoga niza znakova te rezultat spremite u varijablu. Na temelju duljine niza ispišite sve znakove do polovice niza. Primjer: ako imamo niz od 14 znakova (abcdefghijklmn), potrebno je ispisati 1., 2., 3., 4., 5., 6. i 7. znak (abcdefg).
17. * Napišite program koji će u varijablu spremiti neku vrijednost temperature (izražene u stupnjevima Celzijevim). Na ekran ispisati vrijednost temperature u farenhajtima. Formula: $(x \times \frac{9}{5}) + 32$, `x` je vrijednost izražena u stupnjevima Celzijevim.

2.10. Pitanja za ponavljanje: Rad s varijablama

1. Je li na svim računalima preciznost zapisa decimalnih brojeva jednaka?
2. Koji je skraćeni oblik aritmetičkog operatora ako neku varijablu želimo uvećati za neki broj?
3. Je li u *Pythonu* kod navođenja varijable potrebno eksplicitno navesti kojeg je tipa neka varijabla – *int, float, string*?

3. Upravljanje tokom programa

Po završetku ovog poglavlja polaznik će moći:

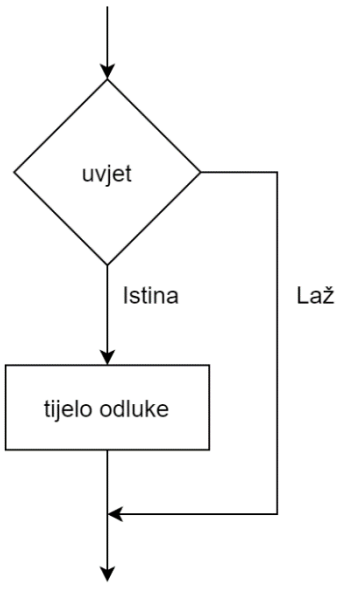
- *kontrolirati tok programa petljama* `while` *i* `for`*, naredbama* `if`*,* `break` *i* `continue`*.*

Najvažniji dio programiranja jest kontrola toka programa. Kontrola toka programa omogućava da odlučujemo na koji način se program ponaša u zadanim okolnostima, tj. zadacima koji se moraju odraditi ovisno o ulaznim vrijednostima.

U *Pythonu* postoje dva osnovna elementa za kontrolu toka, a to su odluke i petlje. Oduke nam pomažu odlučivati koji dio programskoga kôda je potrebno izvršiti, a koji dio programskoga kôda je za ulazne vrijednosti nebitan. Petlje olakšaju ponavljanje stvari, npr. ako se žele ispisati svi brojevi od 1 do 1000, to je moguće napraviti na teži i lakši način. Teži način je da se ručno poziva funkcija `print()` te joj se ručno predaju vrijednosti od 1 do 1000, no taj način programerima oduzima puno vremena. Bolji način je da se ispis odradi pomoću petlje čija će početna vrijednost biti 1, završna vrijednost 1000, a faktor uvećanja 1. Na ovaj način se s nekoliko linija programskoga kôda može ispisati 1000 brojeva, a prema potrebi količina ispisanih brojeva može se jednostavno povećati ili smanjiti.

3.1. Naredba `if`

Naredba `if` pripada kontroli toka za odluke. U najjednostavnijem obliku ova struktura sastoji se od zaglavlja odluke i tijela odluke. Zaglavlje odluke sastoji se od ključne riječi `if` i uvjeta, dok se tijelo odluke sastoji od jedne ili više naredbi koje rade neku korisnu akciju. U nastavku slijedi dijagram toka.



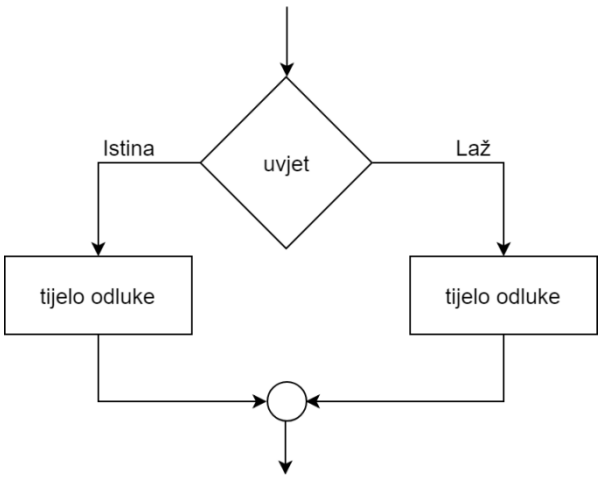
Opis dijagrama: ako je uvjet zadovoljen, izvršava se tijelo odluke, no ako izraz nije zadovoljen, tijelo odluke se neće izvršiti. Sintaksa `if` odluke može se vidjeti u nastavku. Sintaksa zahtjeva da se napiše ključna riječ `if`, nakon toga logički izraz (logički izraz uvijek na kraju završi u `True` ili `False` vrijednosti). Ključne riječi su riječi koje su unaprijed rezervirane od strane korištenoga programskog jezika, u ovom slučaju od strane *Pythona*. Iza ključnih riječi običnu se kriju naredbe. Svaki programski jezik ima skup ključnih riječi i te riječi ne mogu se koristiti kao imena varijabli. Nakon logičkog izraza potrebno je staviti dvotočku koja interpreteru označava da slijedi tijelo odluke.

Primjer:

```
if 5==5:
    print(5)
```

```
Izlaz:
     5
```

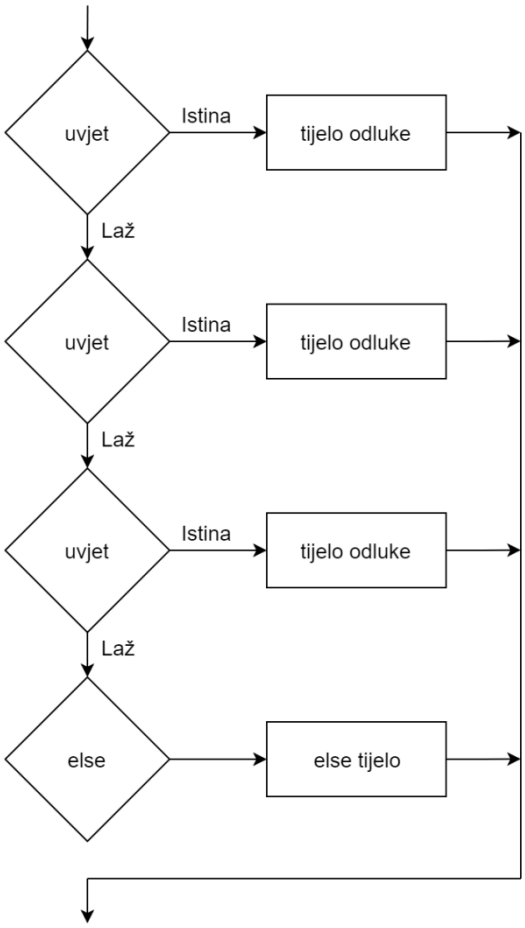
Naredbom `if` moguće je napraviti puno stvari, no kako bi se programerima što više olakšao posao, uvedena je i `else` naredba, koja zajedno s `if` naredbom čini `if-else` odluku. Kod ovakvog tipa odluke, uvijek će biti izvršen jedan dio. Ako je uvjet zadovoljen, izvršit će se tijelo `if` odluke, u suprotnom će se izvršiti tijelo `else` odluke.




```
x = 5

if x>10:
    print("Uvjet je zadovoljen!")
else:
    print("Uvjet nije zadovoljen!")
```

```
Izlaz:
     Uvjet nije zadovoljen!
```

Naredbama `if` i `if-else` pridodaje se još i `if-elif-else` uvjet. Naredba `elif` služi za "neograničen" broj uvjeta, tj. provjera.



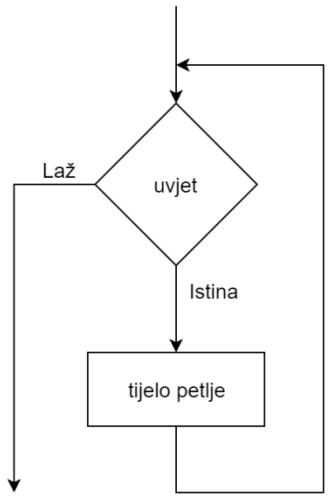
Kada je ovakav skup naredbi ulančan, tada će se izvršiti samo tijelo onog uvjeta koji će prvi biti zadovoljen. Nakon što neki od uvjeta bude zadovoljen i njegovo tijelo odluke se izvrši, ostali uvjeti se više ne provjeravaju. U slučaju da nijedan od uvjeta nije zadovoljen, izvršava se skup naredbi koje pripadaju `else` dijelu `if-elif-else` sintakse.

```
x=5
if x==1:
    print("X = 1")
elif x==2:
    print("X = 2")
elif x==5:
    print("X = 5")
else:
    print("Nijedan od uvjeta nije ispravan!")
```

```
Izlaz:
     X = 5
```

3.2. Petlja `while`

Ova petlja prije izvršavanja programskog kôda koji joj pripada, ispituje da li je uvjet istinit ili ne. Ako je uvjet istinit tada će se izvršiti tijelo petlje, dok se u suprotnom tijelo petlje preskače i izvršava se programski kôd napisan ispod petlje. Sintaksa `while` petlje sastoji se od ključne riječi `while`, nakon toga slijedi logički izraz te dvotočka koja interpreteru označava da nakon nje slijedi tijelo petlje.



U nastavku se nalazi primjer ispisivanja fraze "Hello World!". Ako se ta fraza želi ispisati 10 puta, to se može napraviti na 2 načina. Nizanjem `print()` funkcije 10 puta ili pak pomoću `while` petlje.

```
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Hello World!")
```

Kod ovog zadatka poprilično je jednostavno ispisati 10 puta neku frazu, no zamislimo da je potrebno napraviti program koji 1000 puta ispisuje zadanu frazu. Zbog toga je ovaj problem najelegantnije riješiti upotrebom petlje.

```
x = 0
while x < 10:
    print("Hello World!")
    x = x + 1
```

```
Izlaz:
     Hello World!
     Hello World!
     Hello World!
     Hello World!
     Hello World!
```

```
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
```

Napomena: potrebno je obratiti pozornost da se ne izazove beskonačna petlja. Beskonačna petlja se izazove tako da je uvjet provjere, tj. logički izraz zauvijek istinit. U nastavku slijedi primjer kod kojega se fraza "Hello World!" ispisuje beskonačno puta.

```
while True:
      print("Hello World!")
```

```
Izlaz:
      Hello World!
      Hello World!
      Hello World!
      .....
```

3.3. Petlja `for`

Za razliku od nekih drugih programskih jezika, kao što su *C*, *C++*, *Java* i tako dalje, u programskom jeziku *Python* petlja `for` koristi se na drugačiji način. Kod *Pythona* petlja `for` nema početnu ni završnu vrijednost, a ni aritmetički faktor uvećanja. Petlja `for` u *Pythonu* iterira kroz elemente zadane sekvence. Zadana sekvenca može biti lista ili niz znakova. U nastavku slijedi primjer iteracije kroz elemente liste.

Primjer:

```
for i in range(10):
      print("Hello World!")
```

```
Izlaz:
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
      Hello World!
```

U gornjem su primjeru riječi `for` i `in` ključne riječi. Ova će se petlja okrenuti 10 puta, što je određeno funkcijom `range()`. U nastavku slijedi opis funkcionalnosti funkcije `range()`. Varijabla `i` pri svakoj iteraciji poprima drugu vrijednost, tj. sljedeću vrijednost koja se nalazi u listi koju je funkcija `range()` kreirala.

Funkcija `range()` kreira listu elemenata. Funkcija `range()` može primiti jednu ili dvije vrijednosti. Ako primi jednu vrijednost, tj. ako je poziv funkcije `range()` sljedećeg oblika:

```
range(10)
```

tada se kreira lista od 10 elemenata, prvi element poprima vrijednost 0, dok zadnji element poprima vrijednost 9. Ta lista izgleda ovako:

0	1	2	3	4	5	6	7	8	9

Također, ova funkcija može poprimiti i dvije vrijednosti:

```
range(10, 15)
```

Tada se kreira lista od 5 elemenata, prvi element poprima vrijednost 10, dok zadnji element poprima vrijednost 14. Ta lista izgleda ovako:

10	11	12	13	14

U slučaju da je druga vrijednost koju prima funkcija `range()` manja ili jednaka naspram prvoj vrijednosti, rezultat će biti prazna lista.

```
range(10, 10)
```

U nastavku slijedi primjer koji određuje je li neki proizvoljni broj spremljen u varijablu `n` prost ili nije. Prosti brojevi ili prim-brojevi su svi prirodni brojevi strogo veći od broja 1, a koji su djeljivi bez ostatka samo s brojem 1 i sami sa sobom (na primjer: 2, 3, 5, 7, 11, 13, 17, ...).

```
n = 55

jePrim = True
for x in range(2, n):
    if n % x == 0:
        jePrim = False

if jePrim == True:
     print('Broj', n, 'je prost!')
else:
     print('Broj', n, 'nije prost!')
```

```
Izlaz:
     Broj 55 nije prost!
```

Ako se želi ispisati lista prostih brojeva iz intervala [2, 10], najprije se pomoću petlje izgenerira popis svih brojeva, neovisno o tome je li generirani broj prost ili ne. U nastavku slijedi primjer koji ispisuje sve brojeve iz intervala [2, 7].

```
for n in range(2, 8):
    print(n)
```

```
Izlaz:
     2
     3
     4
     5
     6
     7
```

Ova petlja ispisuje sve brojeve iz intervala [2, 7]. No ako se žele ispisati samo prosti brojevi, tada je potrebno funkciju `print(n)` zamijeniti implementacijom koja otkriva ako neki broj jest ili nije prost. U nastavku slijedi primjer kôda koji ispisuje sve proste brojeve iz intervala [2, 10].

Osim pojedinačnog korištenja petlji (`while` i `for`) moguće ih je i grupirati, tj. unutar jedne petlje staviti drugu petlju (neovisno o tome je li vanjska petlja `for` ili `while`). Nema ograničenja razine grupiranja petlji unutar petlje.

```
for n in range(2, 11):
    jePrim = True
    for x in range(2, n):
        if n % x == 0:
            jePrim = False
    if jePrim == True:
        print('Broj', n, 'je prost!')
    else:
        print('Broj', n, 'nije prost!')
```

```
Izlaz:
     Broj 2 je prost!
     Broj 3 je prost!
     Broj 4 nije prost!
     Broj 5 je prost!
     Broj 6 nije prost!
     Broj 7 je prost!
     Broj 8 nije prost!
     Broj 9 nije prost!
     Broj 10 nije prost!
```

3.4. Naredba `break` i `continue`

Naredba `break`

Naredba `break` nam služi da izađemo iz `for` ili `while` petlje. Kako petlje mogu biti jedna unutar druge, treba obratiti pozornost na to da naredba `break` izlazi samo iz petlje unutar koje je pozvana. Ova naredba se dosta često koristi kod sprječavanja beskonačnih petlji. Na primjer, uvjet `while` petlje postavi se tako da zauvijek bude točan, a onda unutar tijela `while` petlje naredbom `if` provjeravamo ako je neki uvjet zadovoljen. U slučaju da je uvjet zadovoljen, naredbom `break` izlazimo iz petlje.

U nastavku slijedi poboljšani primjer programskog kôda koji određuje ako je broj prost ili ne. Za razliku od prethodnog primjera ovog istog zadatka, u ovom primjeru jednom kada se pronađe da je neki broj `n` djeljiv s nekim drugim brojem, daljnja analiza djeljivosti broja `n` s ostalim brojevima prestaje. Unutarnja `for` petlja u `if` tijelu sadržava ključnu riječ `break`, `break` nam u ovom slučaju omogućava da izađemo iz unutarnje `for` petlje.

```
for n in range(2, 11):
    jePrim = True
    for x in range(2, n):
        if n % x == 0:
            jePrim = False
            break
    if jePrim == True:
        print('Broj', n, 'je prost!')
    else:
        print('Broj', n, 'nije prost!')
```

```
Izlaz:
     Broj 2 je prost!
     Broj 3 je prost!
     Broj 4 nije prost!
     Broj 5 je prost!
     Broj 6 nije prost!
     Broj 7 je prost!
     Broj 8 nije prost!
     Broj 9 nije prost!
     Broj 10 nije prost!
```

Naredba `continue`

Naredba `continue` skače na novu iteraciju petlje. Za razliku od naredbe `break`, naredba `continue` ne prekida izvođenje programa. Sljedeći kôd koji ispituje je li broj u danoj listi paran ili neparan.

```
for num in range(2, 6):
    if num % 2 == 0:
        print("Parni broj:", num)
        continue
    print("Neparni broj:", num)
```

```
Izlaz:
     Parni broj: 2
     Neparni broj: 3
     Parni broj: 4
     Neparni broj: 5
```

Identičnu funkcionalnost iz ovog primjera moguće je napraviti korištenjem `if-else` naredbe. U nastavku slijedi primjer:

```
for num in range(2, 6):
    if num % 2 == 0:
        print("Parni broj:", num)
    else:
        print("Neparni broj:", num)
```

```
Izlaz:
     Parni broj: 2
     Neparni broj: 3
     Parni broj: 4
     Neparni broj: 5
```

3.5. Vježba: Rad s kontrolom toka i petljama

1. U varijablu `r` spremite neki broj koji predstavlja radijus kugle. Ako je radijus ispravno upisan (radijus ne može biti negativan), ispišite radijus i volumen kugle, $V = \frac{4}{3} \times r^3 \times \pi$. Inače, ispišite poruku da je vrijednost u varijabli `r` neispravna.
2. U varijable `a` i `b` spremite dva broja. Ako je jedan od brojeva veći od 100, a onaj drugi manji od 100, tada ispišite poruku "Jedna je veća, a druga je manja od 100.". Također, ispišite prikladne poruke i za slučaj ako su obje vrijednosti veće od 100 ili pak za slučaj ako su obje vrijednosti manje od 100 te ako su obje vrijednosti jednake 100.
3. U varijable `a` i `b` spremite dva broja. Ako je vrijednost varijable `a` bar za 50 veća od vrijednosti varijable `b`, a uz to je vrijednost varijable `b` parna, tada ispišite poruku "Uvjeti su zadovoljeni.", u suprotnom ispišite poruku "Uvjeti nisu zadovoljeni.".
4. U varijable `a` i `b` spremite dva broja. Ispišite poruku "Zadovoljava." ako se barem jedan od brojeva nalazi u intervalu [5, 20], u suprotnom ispišite poruku "Ne zadovoljava.".
5. U varijable `a`, `b`, `c`, `d`, `e` spremite pet različitih brojeva. Ako su bar tri od pet brojeva veći od 100, tada ispišite poruku "Zadovoljava.", u suprotnom ispišite poruku "Ne zadovoljava.".
6. U varijable `a1`, `a2`, `b1`, `b2` spremite četiri cijela broja. Neka vrijednosti koje spremite u varijable zadovoljavaju sljedeće uvjete: `a1` < `a2` i `b1` < `b2`. Te vrijednosti predstavljaju granice dvaju intervala [`a1`, `a2`] – prvi interval i [`b1`, `b2`] – drugi interval. Provjerite je li drugi interval smješten unutar prvog intervala.
   Napomena: provjerite je li početna granica prvog intervala manja ili jednaka početnoj granici drugog intervala (`a1` ≤ `b1`) te je li završna granica prvog intervala veća ili jednaka završnoj granici drugog intervala (`b2` ≤ `a2`).
   Ako su ovi uvjeti zadovoljeni, ispišite poruku "Zadovoljava.", u suprotnom ispišite poruku "Ne zadovoljava.".
7. Ispišite sve parne brojeve između 1 i 1000 koju su istovremeno djeljivi i s 5 i s 13.
8. Napišite program koji sadrži varijablu u kojoj je upisan proizvoljni niz znakova. Ispišite koliko velikih slova se nalazi u nizu. Ako je neko od unesenih slova u nizu "A", brojanje velikih slova je potrebno prekinuti i ispisati informaciju da je veliko slovo "A" pronađeno.
9. * Napišite program koji sadrži varijablu u kojoj je upisan proizvoljni niz znakova i brojčanu varijablu `n`. Provjerite je li vrijednost varijable `n` manja od broja znakova u nizu. Ako je vrijednost varijable `n` veća ispišite informaciju o grešci. Ispišite iz niza znakova svako `n`-to slovo. Na primjer, ulazni niz je "ABCDEFGH", `n` je 2, tada je izlaz "ACEG".
10. * Napišite program koji ispisuje koliko ima prostih brojeva između dva proizvoljna broja.
11. ** Napišite program koji će inicijalizirati varijablu `n` na proizvoljnu cjelobrojnu vrijednost. Vrijednost varijable `n` neka predstavlja red

Napomena

$\pi - 3.1415$

Napomena

Uvjet provjere ako je u varijablu `x = 'e'`, upisano malo slovo:

```
x >= 'a' and x <= 'z'
```

Napomena

11. i 12. zadatak: ako se ne želi nakon svakog poziva funkcije `print()` ispisati i novi redak, to je moguće postići sa sljedećom sintaksom:

```
print(vrijednost, end='')
```

Ako se želi ispisati samo novi redak bez ikakve vrijednosti to je moguće postići sa sljedećom sintaksom:

```
print()
```

tablice. Ispisati tablicu veličine `n` redaka i `n` stupaca. Vrijednost 1 neka se nalazi na glavnoj dijagonali, a vrijednost 0 na svim ostalim mjestima. U nastavku slijedi primjer za `n=5`:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

Napomena: za ispis vrijednost, tako da se nakon ispisa vrijednosti dane `print()` funkciji ne ispiše novi redak koristiti sljedeću sintaksu: `print(vrijednost, end='')`

12. ** Odaberite proizvoljno koordinatu T=(x,y), vrijednosti varijabli `x` (stupac) i `y` (redak) neka budu manje od 10. Program neka ispiše polje 10x10 čiji su svi elementi vrijednosti "`-`" osim koordinate T čija je vrijednost "`X`".

    Primjer: `T=(1,1)`

```
- - - - - - - - - -
- X - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

13. ** Napišite program koji ispisuje sumu znamenaka nekog višeznamenkastog broja. Na primjer, suma broja 159 iznosi 15.

    Napomena: 159%10 = 9, 159 // 10 = 15.

3.6. Pitanja za ponavljanje: Rad s kontrolom toka i petljama

1. Koja je razlika između `while` i `for`?
2. Koja je razlika između `break` i `continue`?

4. Funkcije

Po završetku ovog poglavlja polaznik će moći:

- *definirati i pozivati funkcije*
- *određivati parametre funkcija*
- *koristiti naredbu* `return`
- *razlikovati lokalne i globalne varijable.*

4.1. Definicija funkcije

Funkcija je dio programskoga kôda koji je organiziran prema određenoj funkcionalnosti (logičkoj cjelini). Cilj korištenja funkcija jest da se dijelovi funkcionalnosti koji se ponavljaju u programskom kôdu napišu samo jednom i to unutar funkcije. Korištenjem funkcija jedan te isti kôd se može iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u nekom trenutku potrebna. Ovakav pristup nam omogućava veću preglednost i organizaciju programskoga kôda. Funkcije se koriste kao osnovni elementi te se na temelju njih grade složenije strukture.

Glavna misao vodilja prilikom kreiranja funkcija morala bi biti da funkcionalnosti funkcija budu što jednostavnije, tj. podijeljene na što manje logičke cjeline. Takva organizacija nam omogućuje veću ponovnu iskoristivost programskoga kôda.

Funkcije mogu biti unaprijed napisane. Ako su funkcije unaprijed napisane, to su funkcije koje su ugrađene u sâm programski jezik ili su dio standardnog ili uključenog paketa. Pojam paket detaljnije je obrađen u poglavlju *6. Moduli i paketi*.

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije. U nastavku slijedi primjer definicije funkcije:

```
def sumaBrojeva():
    # tijelo funkcije
```

Opis funkcije iz gornjeg primjera:

Svaka funkcija koju želimo implementirati započinje ključnom riječju ***def*** nakon čega slijedi ime funkcije, u gornjem slučaju ime funkcije je `sumaBrojeva()`. Ime funkcije može biti proizvoljnoga naziva, no treba pripaziti da se funkcija ne zove poput neke od ključnih riječi u *Pythonu* ili pak imenom neke druge funkcije. Nakon imena funkcije obavezne su zagrade i nakon zagrada dvotočka.

Ono što je bitno je to da tijelo funkcije mora biti uvučeno. Za razliku od drugih programskih jezika koji koriste vitičaste zagrade ili ključne riječi za razlikovanje programskih blokova, u *Pythonu* se koristi uvlačenje. U gornjem slučaju tijelo funkcije je programski blok koji je potrebno na neki

način omeđiti kako bi se prilikom izvršavanja znalo koje sve linije programskoga kôda potpadaju u funkciju.

***Python* koristi uvlačenje kao način razlikovanja programskih blokova.** Povećanje uvlačenja znači da dolazi novi, ugniježđeni blok, smanjenje označava kraj trenutačnoga bloka programskoga kôda.

4.2. Poziv funkcije

Nakon što je funkcija definirana i implementirana, možemo ju pozvati tako da napišemo njeno ime te u oblim zagradama navedemo odgovarajuće argumente. Argumenti funkcije će biti obrađeni u nastavku tečaja.

Sintaksa poziva funkcije je:

```
imeFunkcije()
```

```
def prviPoziv():
    print("Hello World!")

prviPoziv()
```

```
Izlaz:
     Hello World!
```

Kao što se može vidjeti iz gornjeg primjera, najprije je napisana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izveden sâm poziv funkcije `prviPoziv()`. Ono što je bitno za primijetiti jest da definicija funkcije mora biti napisana ispred samog poziva funkcije. Ako se poziv funkcije u programskom kôdu nalazi ispred definicije funkcije, kod pokretanja programa dogodit će se greška. U primjeru koji se nalazi niže može se vidjeti pozivanje funkcije prije njene definicije.

```
prviPoziv()

def prviPoziv():
    print("Hello World")
```

```
Izlaz:
     NameError: name 'prviPoziv' is not defined
```

4.3. Parametri funkcije

Da bi funkcijama mogli slati različite vrijednost na temelju kojih će funkcije raditi neku obradu, u zaglavlju funkcije u oblim zagradama upisuju se parametri koje funkcija prima.

4.3.1. Formalni parametri

Formalni parametri funkcije mogu se mijenjati ovisno o potrebi. U istom programu možemo nekoliko puta pozvati jednu te istu funkciju, recimo funkciju `kvadrat()`. Funkcija `kvadrat()` prima samo jedan parametar i ovisno o vrijednosti toga parametra ona izračunava kvadrat unesenoga broja. Prvi put pozovemo funkciju `kvadrat(3)`, a drugi put

`kvadrat(5)`. Primijetimo da rezultat funkcije prilikom prvog poziva mora biti 9, a rezultat funkcije prilikom drugog poziva mora biti 25, tj. da rezultat ovisi o vrijednostima koje je funkcija primila.

Formalni parametri su parametri, tj. varijable koje se nalaze u definiciji funkcije.

```
def kvadrat(x):
```

Varijabla `x` je formalni parametar. Niže je prikazan primjer definicije funkcije `kvadrat()`. Iz priloženoga se vidi da funkcija prima jedan parametar imena `x`. U tijelu funkcije na temelju vrijednosti u varijabli `x` izračunava se i ispisuje kvadrat prenesene vrijednosti.

```
def kvadrat(x):
    print(x*x)
```

Funkcije mogu primati i više parametara. Na primjer, ako funkcija mora vratiti sumu tri broja, a sve tri vrijednosti su funkciji predane kao parametri, definicija funkcije izgleda ovako:

```
def suma(var1, var2, var3):
    print(var1 + var2 + var3)
```

Poziv funkcije s formalnim parametrima

U donjem primjeru vidimo definiciju funkcije `suma()` s dva formalna parametra `var1` i `var2`. Nakon same funkcije (zaglavlja i tijela funkcije) nalazi se poziv funkcije `suma(10, 20)`. U pozivu prenosimo u funkciju vrijednosti 10 i 20. Vrijednost 10 sprema se u varijablu `var1`, dok se vrijednost 20 sprema u varijablu `var2`. Prilikom poziva funkcije formalni parametri zamjenjuju se sa stvarnim argumentima, tj. s konkretnom vrijednošću.

```
def suma(var1, var2):
    print(var1 + var2)

suma(10, 20)
```

```
Izlaz:
     30
```

Napomena: u gornjem dijelu pojašnjenja uveli smo novu riječ – argument. Riječi parametar i argument često se miješaju iako se u suštini radi o jednoj te istoj stvari koja se gleda iz različitih perspektiva.

Parametri funkcije – to su varijable koje su napisane i koje se koriste u samoj funkciji (zaglavlje funkcije i tijelo funkcije). Parametre funkcije možemo smatrati kao obične varijable.

Argumenti funkcije – to su vrijednosti koje se koriste prilikom pozivanja funkcije.

4.3.2. Unaprijed zadane vrijednosti parametara

Python nam omogućuje da unaprijed zadamo predefinirane vrijednosti parametara funkcije. To je omogućeno jer broj parametara funkcije ne mora uvijek odgovarati broju argumenata koji se šalju prilikom poziva funkcije.

U nastavku je prikazan način pomoću kojega se postavlja parametru funkcije unaprijed zadana vrijednost. U donjem slučaju ta unaprijed zadana vrijednost je pridružena varijabli (formalnom parametru funkcije) `var3` na vrijednost 0. Varijabla `var3` će vrijednost 0 poprimiti samo ako se prilikom poziva funkcije prenesu 2 argumenta, a ne sva 3 argumenta koliko ih maksimalno može biti. Ako se prenesu 3 argumenta, tada varijabla `var3` poprima vrijednost trećega prenesenog argumenta.

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20)
```

```
Izlaz:
      30
```

U gornjem primjeru prilikom poziva funkcije `suma()` prenesene su vrijednosti 10 i 20, tj. prenesena su samo 2 argumenta. U ovom slučaju varijabla `var3` poprima predefiniranu vrijednost, a to je vrijednost 0. Varijable `var1` i `var2` poprimaju vrijednost iz poziva funkcije – 10 i 20.

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20, 500)
```

```
Izlaz:
      530
```

U ovom primjeru prenose se 3 argumenta, tj. vrijednosti. Kako funkcija `suma()` može poprimiti maksimalno 3 parametra, prilikom poziva funkcije `suma()` prenijeli smo 3 argumenta u funkciju. Varijabla `var3` neće poprimiti predefiniranu vrijednost 0, već će poprimiti vrijednost trećega prenesenog argumenta iz poziva funkcije `suma()`, tj. vrijednost 500.

4.4. Naredba `return`

Do sada sve gore napisane funkcije u pozivajući dio kôda nisu vraćale nikakvu vrijednost, već su samo ispisivale rezultat bilo kvadriranja, bilo sumiranja. Ako se u pozivajući dio kôda želi vratiti vrijednost koju je funkcija, na primjer, izračunala, to se radi pomoću naredbe `return`.

```
def kvadriranje(x):
    return x*x

var1 = kvadriranje(3)
var2 = kvadriranje(4)
print(var1+var2)
```

```
Izlaz:
      25
```

U gornjem primjeru vidimo funkciju koja implementira matematičku operaciju kvadriranje. Za razliku od prethodnih funkcija, ova funkcija ne ispisuje rezultat kvadriranja. U ovoj funkciji rezultat kvadriranja zahvaljujući naredbi `return` postaje povratna vrijednost funkcije te se ta povratna vrijednost funkcije u gornjem primjeru kod prvog poziva funkcije `kvadriranje()` sprema u varijablu `var1`, a kod drugog poziva funkcije povratna vrijednost se sprema u varijablu `var2`.

4.5. Vidljivost i životni vijek varijable

Vidljivost varijable je pojam koji određuje iz kojih dijelova programa je neka varijabla dohvatljiva, tj. dostupna. Varijable mogu biti lokalne ili globalne.

Životni vijek varijable je pojam koji je određen trenutkom u kojem je varijabla nastala u memoriji i trenutkom u kojem se ta ista varijabla briše iz memorije. Životni vijek varijable ograničen je završetkom bloka programskoga kôda u kojem je ta varijabla nastala, na primjer, funkcija.

4.5.1. Lokalna varijabla

Lokalne varijable su varijable koje su korištene unutar tijela funkcija, u ovu kategoriju možemo svrstati i parametre funkcija. Načelno možemo smatrati da se parametri funkcija ponašaju identično kao i obične varijable unutar tijela funkcije.

Vidljivost lokalnih varijabli – lokalne varijable vidljive su samo unutar funkcije u kojoj su prvi put upotrijebljene. Prvom upotrebom može se smatrati izraz pridruživanja vrijednosti varijabli, na primjer, `var=10`.

Životni vijek varijable – ako je varijabla definirana unutar funkcije, nakon što se izvođenje funkcije završi (bilo pomoću naredbe `return` ili pak završetkom tijela funkcije) varijabla se trajno uništava. Što znači da sljedeći poziv iste funkcije više ne može vidjeti vrijednost varijable iz prethodnog poziva (jer je varijabla prilikom izlaska iz funkcije trajno uništena).

```
def test():
    var=5

test()
print(var)
```

```
Izlaz:
     NameError: name 'var' is not defined
```

U gornjem programu u glavnom dijelu programa pozvana je funkcija `test()`, unutar funkcije `test()` definirana je varijabla `var` s vrijednošću 5. Nakon što izađemo iz funkcije (funkcija ništa ne ispisuje već samo postavlja vrijednost varijable `var`) u glavnom programu pokušavamo ispisati vrijednost varijable `var`, no varijabla `var` nakon izlaska iz funkcije više ne postoji. Nakon izlaska iz funkcije životni vijek

varijable je završio tako da te varijable više nema ni u memoriji, a k tome varijabla `var` je lokalna varijabla te joj nije moguće pristupiti iz glavnog dijela programa jer je vidljiva samo unutar funkcije.

4.5.2. Globalna varijabla

Uz lokalne postoje i globalne varijable. Globalne varijable su varijable koje su kreirane i korištene u glavnom dijelu programa i njihov životni vijek je tako dug dok se program izvršava. Globalne varijable se zovu globalne jer osim što se njima može pristupati iz glavnog dijela programa, tim varijablama se može pristupati i iz funkcija.

```
def test():
    print(var)

var=5
test()
```

```
Izlaz:
      5
```

U gornjem programu kreirana je varijabla `var` i pridružena joj je vrijednost 5. Nakon toga u sljedećoj liniji je pozvana funkcija `test()`. Funkcija `test()` ispisuje sadržaj varijable `var`, no primijetimo da funkcija nema ni formalnih parametara ni varijabli koje bi bile definirane u njoj, no da svejedno uspijeva ispisati vrijednost varijable `var`. To znači da je varijabla `var` globalna varijabla koja je vidljiva iz svih dijelova programa (što uključuje i funkcije).

4.6. Vježba: Rad s funkcijama

1. Napišite funkciju koja ispisuje proizvoljan niz znakova, na primjer "Hello World!". U glavnom dijelu programa pozovite napisanu funkciju.
2. Napišite funkciju koja prima 4 parametara. Funkcija mora ispisati rezultat matematičke formule:

   ((a*a) + (b*c) – d) / 2

   U glavnom dijelu programa pozovite napisanu funkciju.
3. Napišite funkciju `mnozenje()` koja može primiti 2 vrijednosti, od kojih je druga vrijednost unaprijed zadana, sami odredite broj koji ćete pridružiti unaprijed zadanoj vrijednosti. Funkcija mora izračunati i ispisati rezultat množenja.

   U glavnom dijelu programa pozovite funkciju dva puta. Kod prvog poziva funkcije, kao argumente funkcije navedite dvije vrijednosti (vrijednost drugog argumenta neka bude drugačija od unaprijed zadane vrijednosti). Drugi put funkciju pozovite samo jednim argumentom.
4. Napišite funkciju koja prima 2 parametara. Rezultat izračuna funkcije ovog puta se ne ispisuje izravno u funkciji nego u glavnom dijelu programa. Funkcija mora izračunati rezultat formule:

   (a*a) + (b*b)

   Rezultat spremite u varijablu koja se nalazi u dijelu programskoga kôda u kojem se funkcija poziva te ispišite tu varijablu.
5. Pozovite funkciju koja ne prima nijedan parametar, ali mora izračunati i ispisati zbroj dvaju brojeva. Brojevi neka se dohvate iz glavnog dijela programa preko globalnih varijabli.
6. Prethodni zadatak napišite, a da ne koristite globalne varijable (korištenjem parametara funkcije).
7. ** Napišite program koji će inicijalizirati u varijable n i m dva cijela broja proizvoljnih vrijednosti. Provjerite zadovoljavaju li inicijalizirane vrijednosti uvjet: `0 ≤ n ≤ m`. Ako uvjet nije zadovoljen, tada ispišite poruku: "Nedozvoljene vrijednosti!". Ako je uvjet zadovoljen izračunajte i ispišite binomni koeficijent "m povrh n", pri čemu koristite sljedeći izraz:

   $$\binom{m}{n} = \frac{m!}{n! * (m-n)!}$$

   Primjer: 5! se izračunava na način 5*4*3*2

   (Napomena: Implementirajte funkciju `fakt()`, tako implementirana funkcija izračunava faktorijele, na primjer, za poziv funkcije `fakt(5)`, povratna vrijednost je: 120. Funkcija se mora pozivati iz glavnog programa za sve 3 vrijednosti: `m!, n!, (m-n)!`)

4.7. Pitanja za ponavljanje: Rad s funkcijama

1. Funkcije možemo podijeliti na 2 dijela. Koja su to dva dijela?
2. Zaglavlje funkcije sastoji se od 3 elementa. Koji su to elementi?
3. Ako funkcija prima 5 parametara, je li prilikom poziva funkcije potrebno navesti svih 5 argumenata ili postoji način da se neki argumenti izostave i ako da, na koji način?
4. Kako se zove naredba pomoću koje funkcija vraća u pozivajući dio programa izračunatu vrijednost?
5. Kako se zove tip varijable koji je vidljiv samo u funkciji te se nakon završetka funkcije takva varijabla "uništava"?
6. Jesu li globalne varijable iz glavnog dijela programa vidljive u funkcijama?

5. Ulazno izlazne funkcije

Po završetku ovog poglavlja polaznik će moći:

- *koristiti funkcije* `print()` *i* `input()`.

Svaki računalni program treba komunicirati s okolinom (bilo da su to drugi računalni programi ili interakcija s čovjekom). Za ostvarenje komunikacije s okolinom koriste se ulazno izlazne funkcije. Izlazne funkcije (na primjer, funkcija `print()`) omogućavaju ispis podataka na zaslon, dok ulazne funkcije (na primjer, funkcija `input()`) omogućavaju unos podataka preko tipkovnice u sâm program.

U ovom poglavlju obrađuju se gore spomenute funkcije: `print()` i `input()`. Uz njih postoje i druge ulazno izlazne funkcije, poput funkcija za čitanje i pisanje u datoteke. Takve funkcije bit će obrađene u poglavlju *7. Datoteke*.

5.1. Funkcija `print()`

Kao što je već spomenuto, funkcija `print()` omogućava ispis informacija odnosno podataka na ekran. Funkciju `print()` moguće je koristiti na nekoliko različitih načina. U ovom poglavlju bit će opisani najčešće korišteni načini.

Funkcija `print()` može primiti nekoliko argumenata. U nastavku slijedi opis triju najčešće korištenih argumenata, tj. parametara funkcije `print()`.

- `*objects` – prvi argument, tj. argumenti (jer ih može biti više) u pozivu funkcije `print()` su vrijednosti (varijable) koje se žele ispisati. Broj varijabli koje se predaju funkciji `print()` na ispis je proizvoljan. Predani argumenti konvertiraju se u tip podataka *string* na način kao što to radi funkcija `str()` te se ispisuju. Konvertiranje u tip podataka *string* radi se automatski. Ako je predano više argumenata (vrijednosti) kod poziva funkcije `print()`, oni se međusobno odvajaju vrijednošću parametra `sep` i završavaju vrijednošću parametra `end`.
- `sep=' '` – ovaj parametar označava znak kojim su vrijednosti koje se ispisuju međusobno odvojene. Ako ovaj argument nije zadan u funkciji `print()`, tada on poprima predefiniranu vrijednost, tj. razmak `' '`.
- `end='\n'` – ovaj parametar označava znak koji se ispisuje na kraju izvršavanja pojedine `print()` funkcije. Ako ovaj argument nije zadan u funkciji `print()`, tada on poprima predefiniranu vrijednost, tj. znak `'\n'`. Znak `'\n'` predstavlja novi redak. Što bi značilo da nakon što se sve vrijednosti ispišu, ispisuje se i znak za novi redak.

Argumenti `sep` i `end` mogu biti izostavljeni iz poziva funkcije `print()`. U tom slučaju oni poprimaju predefinirane vrijednosti. Predefinirane vrijednosti funkcije `print()` za parametre `sep` i `end` su razmak, tj. `' '` i novi redak, tj. `'\n'`.

Ako se funkciji `print()` ne preda nijedna vrijednost, funkcija `print()` će ispisati samo vrijednost argumenta `end`, a kako nema predanih argumenata to je predefinirana vrijednost `'\n'`, tj. novi redak.

Primjeri korištenja funkcije `print()`:

```
var1 = "Hello World!"
var2 = 44
var3 = 'a'

print(var1, var2, var3)
print("Novi redak.")
```

```
Izlaz:
      Hello World! 44 a
      Novi redak.
```

U gornjem primjeru funkciji `print()` predani su samo argumenti koji se žele ispisati, bez dodatnih argumenata – `sep`, `end`. Prilikom ispisa, funkcija je između varijabli ispisala predefiniranu vrijednost, tj. razmak, a nakon ispisa svih varijabli ispisala je i znak `'\n'`, tj. sljedeći poziv funkcije `print()` nalazi se u novom retku.

```
var1 = "Hello World!"
var2 = 44
var3 = 'a'
print(var1, var2, var3, sep=' --- ')
print("Novi redak.")
```

```
Izlaz:
      Hello World! --- 44 --- a
      Novi redak.
```

U gornjem primjeru funkciji `print()`, uz varijable čije se vrijednosti žele ispisati, prenesen je i argument `sep` s vrijednošću `" --- "`. U ovom slučaju funkcija je između svih argumenata ispisala prenesenu vrijednost parametra `sep` (razmak, tri crtice, razmak).

```
var1 = "Hello World!"
var2 = 44
var3 = 'a'

print(var1, var2, var3, end=' ...\n\n')
print("Novi redak.")
print("HR ima <", 2**10 + 220, "> otoka!", sep="")
```

```
Izlaz:
      Hello World! 44 a ...

      Novi redak.
      HR ima <1244> otoka!
```

U ovom primjeru funkciji `print()` uz varijable čije se vrijednosti žele ispisati, prenesen je i argument `end` čija vrijednost je zamijenila predefiniranu vrijednost parametra `end`, tj. vrijednost `'\n'`. Te se nakon ispisa svih prenesenih argumenata u funkciju `print()` ispisao niz znakova `"...\n\n"`.

5.2. Funkcija `input()`

Gotovo je nemoguće zamisliti program koji obrađuje neki posao, a da je za tu obradu nepotrebna interakcija u smislu dohvaćanja dodatnih podataka. Podaci koje program dohvaća mogu dolaziti iz raznih izvora, na primjer, iz baze podataka, dohvaćanje podataka s Interneta, s tipkovnice itd. Ovo poglavlje obrađuje dohvaćanje podataka s tipkovnice.

Funkcija koja omogućava dohvaćanje podataka s tipkovnice zove se `input()`.

Funkcija `input()` radi na način tako da čita s tipkovnice znakove tako dugo dok se ne pritisne tipka "Enter". Nakon pritiska tipke "Enter" čitanje se završava te funkcija konvertira pročitane podatke u tip podataka *string* te tako učitane znakove vraća preko povratne vrijednosti funkcije.

```
tekst = input()
print("Unesen je tekst: ", tekst)
```

```
Izlaz:
      Ogledna recenica! # Uneseno preko tipkovnice.
      Unesen je tekst: Ogledna recenica!
```

U gornjem primjeru pozvana je funkcija `input()` koja je učitala niz znakova koji je unesen preko tipkovnice. Uneseni niz znakova je: "Ogledna recenica!". Nakon što se učitavanje završilo funkcija `input()` vraća učitani niz znakova te se taj niz znakova sprema u varijablu `tekst`. U drugoj liniji funkcija `print()` ispisuje predane argumente.

Također, funkciji `input()` možemo poslati jedan argument. Sadržaj tog argumenta se ispisuje prije nego što funkcija krene čitati unesene znakove s tipkovnice.

Napomena: ovaj argument nije obavezan i on se može zamijeniti jednostavnim ispisom željenoga teksta pomoću `print()` funkcije.

```
text = input("Unesite tekst: ")
print("Unesen je tekst:", text)
```

```
Izlaz:
      Unesite tekst: Ogledna recenica! #
      Tipkovnica.
      Unesen je tekst: Ogledna recenica!
```

U gornjem primjeru pozvana je funkcija `input("Unesite tekst: ")`, prenesen argument vrijednosti `"Unesite tekst: "` ispisuje se prije nego što se omogućava upis niza znakova preko tipkovnice. Nakon ispisanog niza funkcija `input()` kreće u čitanje znakova koji se unose preko tipkovnice. U drugoj liniji programskoga jezika pomoću funkcije

`print()` ispisuje se niz znakova koji u gornjem slučaju predstavlja opisni tekst "Unesen je tekst:" te vrijednost varijable `tekst`.

5.3. Vježba: Rad s `print()` i `input()` funkcijama

1. Napišite program koji se sastoji od pet varijabli. S jednim pozivom funkcije `print()` ispišite svih pet vrijednosti.
2. Nastavno na prethodni zadatak, doradite poziv funkcije `print()` tako da vrijednost svake pojedine varijable bude u novom redu.
3. Napišite program koji ispisuje niz znakova: "Hello World" te neka se na kraj ispisanog niza ispiše i znak "!". Uskličnik funkciji `print()` prenesite kao `end` argument.
4. Napišite program koji se sastoji od dvije varijable. Vrijednosti tih varijabli učitajte preko tipkovnice te ih ispišite na zaslon ekrana. Prilikom upisivanja vrijednosti u varijable obavijestite korisnika da mora unijeti neku vrijednost, npr. "Unesite prvi niz znakova: ". To napravite bez korištenja funkcije `print()`.

Napomena

Nakon učitavanja nekog broja s tipkovnice pomoću funkcije `input()`, taj broj je tipa *string* tj. niz znakova te ga je potrebno pretvoriti u *int* ili *float*.

5. Napišite program koji s tipkovnice učitava tri cijela broja: redni broj dana u mjesecu, redni broj mjeseca u godini i redni broj godine. Prekontrolirajte jesu li učitane vrijednosti ispravne (ispravne vrijednosti su: dan - [1, 31], mjesec - [1, 12], godina - [0, 2020]). Za ulazne podatke 19, 2, 2017, ispišite datum u sljedećem formatu:

   19. veljače, 2017.

6. Učitavajte brojeve koji predstavljaju dobiveni broj bodova na ispitu. Za broj bodova [0,50>, ispišite "Nedovoljan!", za broj bodova [50, 62.5> ispišite "Dovoljan!", za broj bodova [62.5, 75> ispišite "Dobar!", za broj bodova [75, 87.5> ispišite "Vrlo dobar!", a za broj bodova [87.5, 100] ispišite "Odličan!". U slučaju da je uneseni broj izvan raspona brojeva <0, 100>, ispišite prikladnu poruku i prekinite daljnje učitavanje broja bodova.
7. ** Učitajte s tipkovnice varijablu `n` iz intervala [3, 10]. Učitavanje ponavljati tako dugo dok se ne učita ispravna vrijednost. U slučaju neispravne vrijednosti ispisati obavijest da je vrijednost neispravna. Ispisati tablicu od `n` redaka i `n` stupaca. Format ispisa zaključiti na temelju primjera koji se nalazi u nastavku (`n=10`).

```
  1   2   3   4   5   6   7   8   9  10
     11  12  13  14  15  16  17  18  19
         20  21  22  23  24  25  26  27
             28  29  30  31  32  33  34
                 35  36  37  38  39  40
                     41  42  43  44  45
                         46  47  48  49
                             50  51  52
                                 53  54
                                     55
```

5.4. Pitanja za ponavljanje: Rad s `print()` i `input()` funkcijama

1. Je li potrebno prenijeti argumente poput `sep` i `end` u pozivu funkcije `print()`?
2. Ako se u pozivu funkcije `print()` ne prenese argument `sep`, koja će se vrijednost ispisati između dva objekta?
3. Ako je poziv funkcije `print()` oblika: `print("Hello World!")`, hoće li se automatikom ispisati predefinirana vrijednost parametra `sep` ili `end`?
4. Za što služi jedini mogući argument funkcije `input()`?
5. Na koji način možemo izbjeći slanje funkciji `input()` argument tipa *string*, a da se dobije identična funkcionalnost kao da smo taj argument prenijeli?

6. Moduli i paketi

Po završetku ovog poglavlja polaznik će moći:

- *koristiti module i pakete koji dolaze zajedno sa standardnom instalacijom Pythona.*

Moduli i paketi omogućavaju lakši i brži razvoj programskog kôda, a u konačnici i programa. Napisano je puno modula tj. paketa za *Python* koji ubrzavaju razvoj programskog kôda jednostavnim pozivanjem unaprijed napisanih funkcija, razreda, konstanti. Moduli tj. paketi su organizirani u smislene cjeline radi što lakšeg snalaženja, ne samo prilikom korištenja već i radi što lakšeg traženja po dokumentaciji.

Module i pakete koji dolaze sa standardnom instalacijom *Pythona* nazivamo *standardnom bibliotekom*.

Ponekad su programerima potrebne funkcionalnosti koje ne dolaze s modulima tj. paketima iz *standardne biblioteke*. Za takve funkcionalnosti potrebno je poslužiti se Internetom i potražiti odgovarajuće pakete te ih instalirati na računalo na kojem će se program izvršavati. U ovom tečaju obrađeni su samo moduli koji dolaze sa standardnom instalacijom *Pythona*.

Potrebno je razlikovati module i pakete. Moduli su elementi programskog kôda unutar kojih je implementirana neka specifična funkcionalnost. Uzmimo za primjer nekoliko modula koji dolaze sa standardnom bibliotekom: `math`, `cmath`, `random`, `datetime`. Paketi su cjeline koje uključuju druge module i oni omogućavaju njihovo hijerarhijsko grupiranje. Uzmimo za primjer paket koji dolazi sa standardnom bibliotekom, paket `urllib`, koji u sebi sadrži nekoliko modula:

- `urllib.request`
- `urllib.error`
- `urllib.parse`
- `urllib.robotparser`.

6.1. Rad s modulima i paketima

Funkcije, konstante i razrede koji se nalaze unutar nekog modula, moguće je koristiti na dva načina:

- najavljivanjem korištenja
- uključivanjem u program.

Najavljivanje korištenja

Najavljivanje korištenja modula ostvaruje se pomoću ključne riječi `import`. Sintaksa:

```
import <modul>
```

```
import math

print(math.sin(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
```

U gornjem primjeru prikazan je način korištenja naredbe `import`. Kod ovakvog načina korištenja naredbe `import` najavljuje se korištenje **svih** funkcija, razreda i konstanti (u daljnjem tekstu spominjat će se samo funkcije, no podrazumijeva se da se u nekom modulu mogu nalaziti i razredi i konstante) iz nekog modula.

Ako se korištenje funkcija samo najavi kao u gornjem primjeru, tada u svakom dijelu programa u kojem želimo iskoristiti funkciju iz najavljenog modula mora pisati ime modula i ime funkcije odvojene točkom. Sintaksa ovakvoga načina korištenja najavljenih funkcija jest:

```
modul.funkcija()
```

Uključivanje u program

Ako se želi izbjeći gornja sintaksa i koristiti samo ime funkcije kao poziv, a ne da se ispred imena funkcije piše ime modula u kojem se funkcija nalazi, tada je potrebno željene funkcije uključiti u program. Uključivanje funkcija u program radi se korištenjem ključnih riječi `from` i `import`. Sintaksa takvoga načina korištenja funkcija jest:

```
from <modul> import <funkcija>, <funkcija>, ...
```

Ključna riječ `from` govori koji modul će se koristiti, dok ključna riječ `import` govori koje sve funkcije iz modula će se uključiti u naš program. Kod ovakvog načina uključivanja funkcija u program, prilikom korištenja uključenih funkcija, za razliku od prijašnjeg primjera, više nije potrebno pisati u kojem modulu se nalazi korištena funkcija.

```
from math import sin, cos

print(sin(55))
print(cos(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
      Traceback (most recent call last):
        File "test.py", line 4, in <module>
          print(math.tan(55))
      NameError: name 'math' is not defined
```

U gornjem primjeru prikazana je sintaksa korištenja funkcija tako da se one uključuju u programski kôd. U program su uključene samo one funkcije koje će se koristiti u programskom kôdu, a to su `sin()` i `cos()`. Primijetimo da prilikom poziva funkcija više nije potrebno ispred funkcije pisati ime modula u kojem se ta funkcija nalazi.

U gornjem primjeru namjerno je izazvana pogreška. Funkcija `tan()` iz modula `math` nije uključena u programski kôd, a nije najavljeno ni njeno korištenje u programskom kôdu. Za demonstraciju ona nije pozvana tako da se napiše samo ime funkcije već je pozvana na način `math.tan()`, no svejedno se dogodila greška. Greška je nastala zbog toga što se u gore navedenom načinu uključivanja funkcija iz modula `math` u program nije najavilo i korištenje ostalih funkcija iz tog paketa. Ako se želi ispraviti ova greška, to je moguće napraviti na dva načina:

- Dodavanjem funkcije `tan()` u uključene funkcije:

```
from math import sin, cos, tan
```

- Najavljivanjem korištenja svih funkcija iz modula `math`:

```
import math
```

Svejedno je koji način će se odabrati.

Ako se prilikom korištenja (poziva) funkcija iz modula ne želi pisati iz kojeg modula dolazi funkcija, a ne želi se ni kod uključivanja modula raditi popis svih funkcija koje će se koristiti, to se može napraviti tako da u program uključimo sve funkcije iz nekog modula.

```
from math import *
```

Problem koji se može pojaviti prilikom ovakvoga načina korištenja modula jest to da se prilikom uključenja više različitih modula može dogoditi kolizija ako se u dva i više modula nalazi isto ime funkcije.

Konstante

U modulima se mogu nalaziti i razne konstante. Tako se u modulu `math` nalazi nekoliko matematičkih konstanti: *pi*, *e*, *tau*, *inf*, *nan*. U nastavku slijede primjeri programskog kôda unutar kojih se dohvaća i ispisuje vrijednost konstante *pi*.

Najavljivanje korištenja svih funkcija i konstanti modula `math`:

```
import math
print(math.pi)
```

```
Izlaz:
      3.141592653589793
```

Uključivanje u program konstante `pi` iz modula `math`:

```
from math import pi
print(pi)
```

```
Izlaz:
      3.141592653589793
```

6.2. Vježba: Korištenje *Python* modula i paketa

1. Napišite program koji najavljuje korištenje funkcija iz modula `math` te izračunajte i ispišite rezultat:

   sin(2) + cos(1)

2. Pronađite na Internetu u službenoj *Python 3* dokumentaciji kako se zove funkcija koja se nalazi u modulu `math` koja se koristi za korjenovanje. Preko tipkovnice unesite broj i ispišite njegov korijen.

   Službenu *Python 3* dokumentaciju možete pronaći na URL-u: https://docs.python.org/3/library/math.html

3. Korištenjem konstanti koje su pohranjene u modulu `math` ispišite vrijednost konstante PI.

4. Korištenjem konstante PI izračunajte i ispišite rezultat:

   sin(2PI) + cos(PI)

5. * U službenoj *Python 3* dokumentaciji pronađite ime funkcije za potenciranje (zamjena za aritmetički operator **). S tipkovnice učitajte cjelobrojne vrijednosti i spremite ih u varijable `a` i `b`. Nije potrebno provjeravati ispravnost učitanih brojeva. Na temelju tih vrijednosti izračunajte kvadrat zbroja $(a + b)^2$. Formula za kvadrat zbroja je $(a + b)^2 = a^2 + 2ab + b^2$. Funkcije koje ćete koristiti u ovom zadatku uključite u program.

6. * U službenoj *Python 3* dokumentaciji pronađite ime funkcije koja služi za zaokruživanje na prvi viši cijeli broj i na prvi niži cijeli broj. Broj nad kojim se želi napraviti zaokruživanje učitajte s tipkovnice. Tako učitani broj neka bude decimalni broj. Na primjer, za učitani broj 5.587, ispis na ekran mora biti sadržaja: "5, 6". U ovom zadatku najavite korištenje funkcija, no nemojte ih uključiti.

6.3. Pitanja za ponavljanje: Korištenje *Python* modula i paketa

1. Nalaze li se u modulima samo funkcije?

2. Ako najavimo korištenje modula pomoću naredbe `import`, je li potrebno prilikom poziva funkcije naznačiti u kojem modulu se pozivajuća funkcija nalazi?

7. Datoteke

Po završetku ovog poglavlja polaznik će moći:

- *kreirati, čitati, promijeniti i brisati tekstualne datoteke.*

Prilikom pokretanja *Python* skripti svi podaci koje skripta izračuna nalaze se u radnoj memoriji – RAM memorija (engl. *Random Access Memory*). Tim podacima nakon završetka skripte više nije moguće pristupiti. Ako se neki podatak želi trajno spremiti za kasnije korištenje, potrebno ga je spremiti u dio memorije koji je namijenjen trajnom spremanju podataka. Primjeri trajne memorije su: HDD (engl*. Hard Disk Drive*), SSD (engl*. Solid State Drive*), USB Flash Drive (engl*. Universal Serial Bus Flash Drive*) i slično. U trajnoj memoriji podaci ostaju zapisani i nakon gašenja računala.

U ovom poglavlju opisane su osnovne operacije nad datotekama, a to su:

- kreiranje (engl*. create*)
- čitanje (engl*. read*)
- promjena (engl*. update*)
- brisanje (engl*. delete*).

Ponegdje se ove četiri operacije spominju i pod imenom CRUD.

U trajnoj memoriji podaci su spremljeni u datoteke. Svaka datoteka opisana je ekstenzijom kojega je tipa, na primjer, *.txt, *.doc itd. Za datoteku koja se želi pročitati potrebno je imati prikladni program za čitanje datoteke, na primjer *Notepad* za *.txt*, Microsoft Word* za *.doc datoteke i tako dalje. Potrebno je obratiti pažnju da, na primjer, *.txt i *.doc tip datoteke nisu identične strukture, što znači da ako želimo sa skriptom spremati podatke u *.doc tip datoteke, potrebno je proučiti na koji način se takva datoteka gradi. Najjednostavniji tip datoteke za pisanje/čitanje je *.txt jer on ne zahtijeva nikakvu "dodatnu" strukturu.

Uz datoteke bitni su i direktoriji, tj. mape (engl. *folder*). Direktoriji omogućavaju lakše snalaženje/pronalaženje datoteka. Direktoriji se slažu u hijerarhije po nekoj smislenoj strukturi. Za razdjeljivanje nivoa direktorija koristi se znak: \ ili /.

Windows operativni sustav: `C:\Users\Username\Documents`
UNIX operativni sustav: `/home/username/Documents`

Hijerarhija mape koja vodi sve do neke određene datoteke zove se putanja (engl*. path*). Putanje možemo podijeliti na dva tipa:

- apsolutna putanja – potpuna putanja od particije sve do datoteke
- relativna putanja – putanja koja nije potpuna, već se datotekama pristupa tako da se kao početni direktorij uzima direktorij u kojem se nalazi skripta.

Primjer: struktura (hijerarhija) direktorija:

```
C:\Users\Username\Desktop (putanja)
|-- Program (direktorij)
|   |-- Datoteke (direktorij)
|   |   `-- Podaci.txt (datoteka)
|   `-- skripta.py (datoteka)
```

Ako se u skripti: `skripta.py` pomoću relativne putanje želi dohvatiti datoteka `Podaci.txt`, potrebno je napisati:

```
Datoteke\Podaci.txt
```

Ako pak se u skripti `skripta.py` za dohvaćanje datoteke `Podaci.txt` želi koristiti apsolutna putanja, potrebno je napisati:

```
C:\Users\Username\Desktop\Program\Datoteke\Podaci.txt
```

7.1. Tekstualne datoteke

Prije čitanja iz datoteke ili pak pisanja u datoteku istu je potrebno otvoriti. Datoteka se otvara pozivom metode `open()`. Metoda `open()` vraća objekt datoteke, a poziva se tako da joj se prosljeđuju dva argumenta:

- ime datoteke (s apsolutnom ili relativnom putanjom) koje će se kreirati / čitati / mijenjati
- način korištenja (engl. *mode*) – prava koja skripta ima za obradu datoteke (čitanje / pisanje, …), način na koji će se datoteka koristiti.

```
f = open('datoteka.txt', 'w')
```

Način korištenja može biti:

Mode	Opis
`r`	Predefinirana vrijednost. Datoteka se otvara za čitanje. Nije dopušteno pisanje u datoteku. Ako datoteka ne postoji, vraća se poruka o grešci.
`w`	Otvara se datoteka za pisanje. Nije dopušteno čitanje iz datoteke. Ako datoteka ne postoji, stvara se nova datoteka. Ako datoteka postoji, briše se sadržaj postojeće datoteke.
`x`	Kreira se nova datoteka. Ako datoteka postoji, vraća se poruka o grešci. Dopušteno je pisanje u datoteku. Nije dopušteno čitanje iz datoteke.
`a`	Ako datoteka ne postoji, stvara se nova datoteka. Ako datoteka postoji, novi podaci se upisuju na kraj datoteke. Nije dopušteno čitanje iz datoteke. Pokazivač za pisanje u datoteku nalazi se na kraju datoteke.

Napomena

Uz tekstualni format zapisivanja podataka postoji i binarni format zapisa podataka u datoteke. Binarni način zapisa podataka nije ništa drugo nego zapis podataka u datoteku u identičnom formatu kako se ti podaci nalaze u memoriji računala.

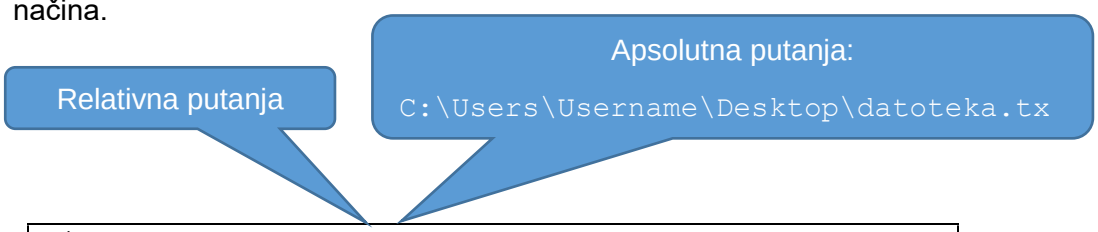
Mode	Opis
`+`	Ako se doda ovaj znak, omogućava se čitanje i pisanje iz datoteka.
`t`	Predefinirana vrijednost. Otvara se datoteka za čitanje/pisanje tekstualnih datoteka
`b`	Otvara se datoteka za čitanje/pisanje binarnih datoteka.

Prilikom poziva metode `open()` parametar `mode` je opcionalni. Ako se ovaj parametar izostavi, predefinirana vrijednost je `'r'`.

Svaki tok podataka prema datoteci koji se više ne planira koristiti potrebno je zatvoriti. Zatvaranjem se odmah oslobađaju sistemski resursi koji su korišteni do tada. Ako se tok podataka ne zatvori ručno pozivom metode `close()`, tok podataka će uništiti *Python* skupljač smeća (engl. *Garbage collector*), no njegov nedostatak je taj što se to oslobađanje resursa možda neće dogoditi odmah nakon što se tok podataka više ne koristi, već se može dogoditi da taj tok podataka još neko vrijeme egzistira u memoriji. Također, ako oslobađanje resursa prepustimo *Pythonu*, može se dogoditi da različite *Python* implementacije unište tok podataka u različitim trenucima. Sintaksa korištenja metode `close()` za oslobađanje memorije od toka podataka je:

```
f.close()
```

Dobra praksa je da se koristi ključna riječ `with` kada radimo s datotekama. Prednost ovakvoga načina programiranja jest da nakon što taj blok programskoga kôda završi, tok podataka se automatski zatvara. U nastavku je prikazana sintaksa korištenja prethodno opisanoga načina.

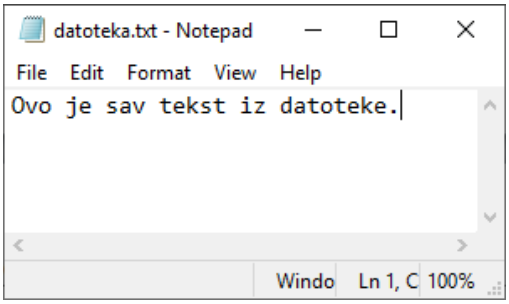



```
with open('datoteka.txt') as f:
    # Ovdje ide programska logika
# Ovdje više nije moguće koristiti tok podataka
prema datoteci datoteka.txt
```

Čitanje cijele datoteke

Kako bismo pročitali datoteku, potrebno je pozvati `f.read(size)` metodu. Ova metoda čita podatke iz datoteke i vraća ih kao niz znakova. Parametar `size` je opcionalni argument. U slučaju da je parametar `size` izostavljen ili pak negativan, tada se čita cijela datoteka. Programer mora paziti da veličina datoteke koja se čita ne bude veća od radne memorije samoga računala na kojem se skripta izvršava. U slučaju kada čitanje dođe do kraja datoteke, `f.read()` metoda vraća prazan niz znakova: `''`.

U datoteci `datoteka.txt` nalazi se niz znakova sadržaja: `"Ovo je sav tekst iz datoteke."`. U nastavku se nalazi prikaz izgleda datoteke:




```
with open('datoteka.txt') as f:
    cijelaDatoteka = f.read()
    print(cijelaDatoteka)
    cijelaDatoteka = f.read()
    print(cijelaDatoteka)
```

```
Izlaz:
     Ovo je sav tekst iz datoteke.
     ''
```

U gornjem programskom odsječku dva puta je pozvana metoda `f.read()`. Prilikom prvoga poziva ispisao se kompletan tekst koji se nalazi u datoteci `datoteka.txt`, prilikom drugog poziva čitanje datoteke nije krenulo ponovo od početka, već od mjesta gdje je prethodno čitanje datoteke stalo. Kako je u prethodnom koraku pročitana cijela datoteka, u tom koraku nije ostalo ništa više za pročitati i u varijablu `cijelaDatoteka` vraćen je prazni niz znakova, tj. `''`.

U sljedećem primjeru metodi `read()` prenosimo i parametar `size`. U ovom slučaju ovaj parametar označava koliko se znakova želi pročitati. U oba čitanja čita se po 5 znakova i kao što se može vidjeti iz ispisa kod prvog čitanja pročitan je niz znakova `"Ovo j"`, a kod drugog čitanja pročitan je niz znakova: `"e sav"`.

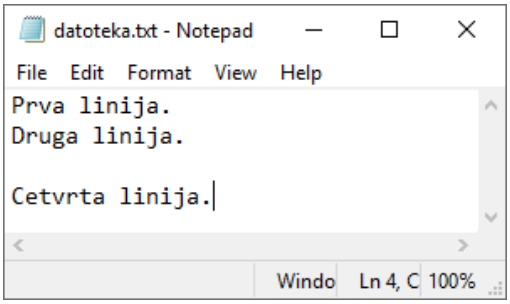
```
with open('datoteka.txt') as f:
    znakovi = f.read(5)
    print(znakovi)
    znakovi = f.read(5)
    print(znakovi)
```

```
Izlaz:
     Ovo j
     e sav
```

Čitanje iz datoteke linija po linija

Metoda `f.readline()` čita jednu liniju iz datoteke, tj. tako dugo dok se ne pročita znak za novu liniju `'\n'`. Svaka linija završava znakom `'\n'`. Ako metoda `f.readline()` vrati prazan niz znakova, to znači da je čitanje došlo do kraja datoteke. U slučaju da je negdje u datoteci pročitana prazna linija, tj. `'\n'`, tada se u povratnoj vrijednosti niza znakova nalazi znak `'\n'`.

Sadržaj datoteke `datoteka.txt` je:



Primjer čitanja linije po linije iz datoteke `datoteka.txt`

```
with open('datoteka.txt') as f:
    linija = f.readline()
    print(linija)
    linija = f.readline()
    print(linija)
    linija = f.readline()
    print(linija)
    linija = f.readline()
    print(linija)
```

```
Izlaz:
     Prva linija.

     Druga linija.



     Cetvrta linija.
```

Primijetimo da između prve i druge linije u datoteci nema praznog prostora, a prilikom ispisa postoji jedan prazan redak. To se dogodilo zato što je metoda `readline()` uz tekst pročitala i spremila u varijablu linija znak `'\n'` koji označava novi redak. Prilikom ispisa niza znakova ispisan je i novi redak, a kao što je objašnjeno u prethodnim poglavljima funkcija `print()` nakon što ispiše sve znakove koji su joj predani ispisuje još i vrijednost argumenta `end`, predefinirana vrijednost argumenta `end` je novi redak, tj. `'\n'`.

```
with open('datoteka.txt') as f:
    for linija in f.readlines():
        print(linija, end='')
```

```
Izlaz:
     Prva linija.
     Druga linija.

     Cetvrta linija.
```

Gornji programski kôd prikazuje na koji način iterirati po **cijeloj** datoteci ako nam nije poznato koliko linija ima datoteka iz koje se čita.

Ako se žele pročitati sve linije neke datoteke i tako pročitane vrijednosti (linija po linija) spremiti u listu, to se radi pozivom sljedeće funkcije ili metode: `list(f)` ili `f.readlines()`.

```
with open('datoteka.txt') as f:
    lista = list(f)
    print(lista)
```

```
Izlaz:
     ['Prva linija.\n', 'Druga linija.\n', '\n',
      'Cetvrta linija.']
```

Pisanje u datoteku

Samo čitanje iz datoteke nije dovoljno jer s čitanjem ne možemo trajno spremiti rezultate neke obrade u datoteku. Za pisanje u datoteku potrebno je koristiti metodu `f.write(string)`. Ova metoda u datoteku zapisuje niz znakova koji je spremljen u varijabli tipa *string* ili pak direktno predan. Povratna vrijednost ove metode jest broj znakova koji je uspješno zapisan u datoteku. Ako u datoteku želimo zapisivati niz znakova, prilikom otvaranja toka podataka potrebno je prenijeti i parametar `mode`. Kao što je prethodno navedeno, ako se parametar `mode` ne prenese, tada on poprima predefiniranu vrijednost, a to je dozvola samo za čitanje iz datoteke. U nastavku slijedi primjer kojim se u datoteku `datoteka.txt` upisuje niz znakova `"Test"`.

```
with open('datoteka.txt', 'w') as f:
    f.write('Test')
```

Sintaksa koja ispisuje koliko je znakova uspješno upisano u datoteku jest:

```
with open('datoteka.txt', 'w') as f:
    x = f.write('Test')
    print(x)
```

```
Izlaz:
     4
```

Ako u datoteku želimo spremiti ostale tipove objekata, npr. listu, skup, n-terac, rječnik, to je također moguće napraviti, no prije samog spremanja te je objekte potrebno pretvoriti u niz znakova. Pretvaranje nekog objekta u niz znakova tipa *string*, obrađuje se pozivom funkcije `str()`.

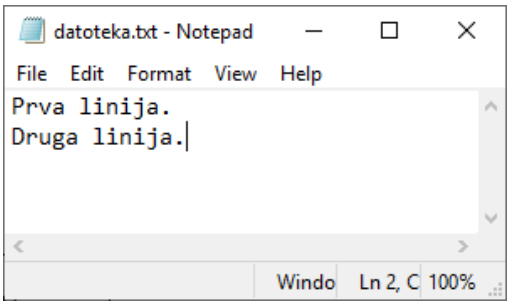
```
rjecnik = {1 : "Prvi", 2 : "Drugi"}
with open('datoteka.txt', 'w') as f:
    s = str(rjecnik)
    f.write(s)
```

Pisanje u datoteku uz postojeći sadržaj

Ako se u tekstualnu datoteku u kojoj već od prije postoji sadržaj želi upisati novi tekst (bez da se prebriše postojeći sadržaj), potrebno je otvoriti tok podataka prema toj datoteci s modom korištenja `'a'`. Ako se u tekstualnu datoteku želi dodati novi tekst, ali tako da se postojeći sadržaj tekstualne datoteke ne prebriše, to je moguće, ali tako da se novi tekst dodaje na kraj tekstualne datoteke.

Ako se tekst želi dodati na lokaciji unutar datoteke, a da to nije kraj datoteke te da se pritom ne prebriše postojeći sadržaj, to nije moguće napraviti na jednostavan način pomoću metode `write()`. To se može napraviti tako da se sadržaj tekstualne datoteke učita u radnu memoriju te da se nakon toga sadržaj iz varijable upisuje u datoteku, a na željenu lokaciju se onda ubaci novi tekst.

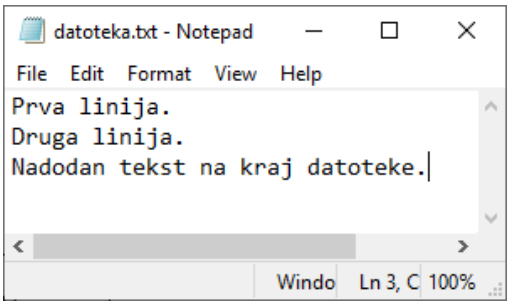
Početni sadržaj datoteke `datoteka.txt` je:



Nakon što se pokrene programski kôd oblika:

```
with open('datoteka.txt', 'a') as f:
    f.write("\nNadodan tekst na kraj datoteke.")
```

Rezultat upisa u tekstualnu datoteku je:

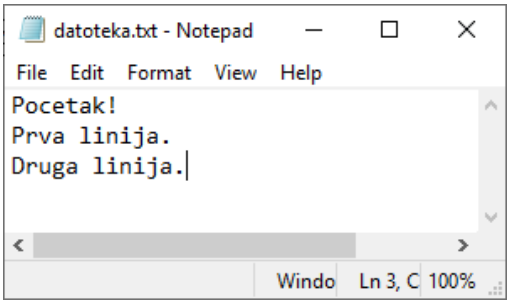


Kao što se može iz gore navedenoga programskog kôda vidjeti, ako se otvori tok podataka prema datoteci s modom tipa `'a'`, svaki put kada se pomoću metode `write()` upisuje novi tekst u datoteku taj tekst sprema se na kraj datoteke.

U slučaju da se sadržaj datoteke `datoteka.txt` želi urediti tako da se na sâm početak datoteke nadoda novi tekst, najprije je potrebno cijeli sadržaj datoteke učitati u radnu memoriju. Nakon što je sadržaj datoteke učitan u varijablu `sadrzaj`, željeni početni tekst stavlja se na početak datoteke te se nakon toga u datoteku stavlja stari sadržaj.

```
with open('datoteka.txt', 'r+') as f:
    sadrzaj = f.read()
    f.seek(0, 0)
    f.write("Pocetak!\n" + sadrzaj)
```

Rezultat gornjega programskog kôda je:



U ovom programskom kôdu tok podataka prema datoteci otvaran je načinom korištenja `'r+'`, što znači da se nad datotekom `datoteka.txt` daju prava i za čitanje i za pisanje kao što je opisano u prethodno navedenoj tablici koja opisuje načine korištenja toka podataka.

U gornjem primjeru programskoga kôda prvi put se spomenuo poziv metode `seek()`. Metoda `seek()` služi za pomicanje položaja "čitača" unutar datoteke. Sintaksa poziva te metode je:

```
f.seek(offset, from_what)
```

Parametar `offset` je pozicija koja je izračunata naspram referentne točke. Referentna točka je definirana pomoću `from_what` argumenta. Donja tablica prikazuje što znači koja vrijednost `from_what` argumenta.

Vrijednost	Značenje
0	Početak datoteke
1	Trenutna pozicija u datoteci
2	Kraj datoteke

Dodatno objašnjenje korištenja metode `seek()`: u gornjem primjeru, najprije je pomoću metode `read()` pročitana cijela datoteka. Nakon čitanja cijele datoteke pokazivač koji pokazuje gdje je čitanje datoteke stalo nalazi se na kraju datoteke. Kako je sljedeći korak zapisivanje teksta natrag u datoteku (novoga na početak i staroga koji slijedi nakon novog teksta), potrebno se pomoću metode `seek()` pozicionirati na početak tekstualne datoteke odakle će krenuti zapisivanje novoga teksta natrag u datoteku.

Također, postoji metoda `tell()` koja vraća trenutno poziciju "čitača / pisača" u datoteci. Trenutačna pozicija je broj bajtova od početka datoteke ako čitamo binarnu datoteku ili pak broj znakova ako se čita tekstualna datoteka.

7.2. Vježba: Rad s datotekama

1. Napišite program koji će kreirati praznu tekstualnu datoteku naziva `datoteka.txt` (ako datoteka identičnog imena već postoji od prije, tada ju je potrebno pregaziti s praznom datotekom) te je u tako kreiranu datoteku potrebno ispisati sve parne brojeve do 20. Parni brojevi neka međusobno budu odvojeni razmakom.
2. Izmijenite prethodni zadatak tako da se kreira datoteka naziva `datoteka.txt`. U tako kreiranu datoteku spremite sve brojeve od 1 do 10. Svaki broj neka bude u zasebnoj liniji. Nakon što je datoteka kreirana, pročitajte iz nje sve brojeve te napravite sumu pročitanih brojeva, a taj rezultat ispišite na zaslon.
3. Izmijenite prethodno napisani program tako da se ukupna suma brojeva iz datoteke `datoteka.txt` zapiše na kraj datoteke u formatu: "<55>".
4. * Pomoću programa *Notepad* kreirajte datoteku i u 10 redaka te datoteke napišite po jedan proizvoljan broj. Napišite program koji će pročitati tu datoteku te sve parne brojeve iz nje upisati u novu datoteku. Također, izračunajte sumu svih neparnih brojeva te ih zapišite u treću datoteku.

Napomena

Metoda `write()` prima jedan argument tipa niz znakova tj. *string*. Ako se pomoću te metode želi u datoteku upisati neki broj, taj broj je potrebno pomoću funkcije `str()` pretvoriti u niz znakova tj. *string*.

7.3. Pitanja za ponavljanje: Rad s datotekama

1. Koji `mode`, tj. način korištenja služi samo za čitanje iz datoteke?
2. Ako se želi prebrisati stara postojeća datoteka, koji `mode`, tj. način korištenja se mora koristiti?
3. Ako želimo i čitati i pisati u datoteku, koji je znak potrebno dodati?
4. Ako želimo obrađivati binarne datoteke, koji je znak potrebno dodati?

8. Parametri naredbenoga retka

Po završetku ovog poglavlja polaznik će moći:

- *pokretati skriptu pomoću naredbenoga retka*
- *koristiti program Task Scheduler za pokretanje skripte.*

8.1. Pokretanje skripte preko naredbenoga retka

Do sada su sve skripte pokretane tako da su se pokretale izravno iz *Python IDLE-a*. Nakon prekretanja skripte preko *Python IDLE-a*, otvara se *Python Shell* unutar kojega programer može unositi željene ulazne podatke. No, osim tog načina svaku *Python* skriptu moguće je pokrenuti i preko naredbenoga retka (engl. *Command Prompt*). To se može napraviti na 2 načina:

- Dvostrukim klikom na skriptu koja se želi pokrenuti

  Nakon dvostrukog klika na željenu skriptu, automatski se pokreće naredbeni redak unutar kojega se skripta izvršava. Ako skripta zahtjeva ulazne podatke, tražene ulazne podatke programer može unijeti kroz naredbeni redak koji se otvorio.

- Pokretanje preko naredbenoga retka

  Drugi način pokretanja *Python* programa je izravno iz naredbenog retka.

  Pokrenuti naredbeni redak: *Windows Start → All programs (*hr. *Svi programi) → Command Prompt (*hr*. Naredbeni redak)*

  Pokretanje skripte unutar naredbenoga retka moguće je napraviti tako da se napiše cijela putanja do skripte ili pak da se najprije pozicionira u direktorij u kojem se željena skripta nalazi te se nakon toga pokreće izvršavanje skripte. U nastavku će biti objašnjen način pokretanja skripte tako da se najprije izvrši pozicioniranje u direktorij u kojem se skripta nalazi.

  Pozicioniranje u određeni direktorij radi se naredbom `cd`.

  Pretpostavka je da se skripta nalazi na radnoj površini (engl*. Desktop*). Pretpostavimo da se `Desktop` na *Windows 10* operativnom sustavu nalazi na lokaciji:

  ```
  C:\Users\Username\Desktop
  ```

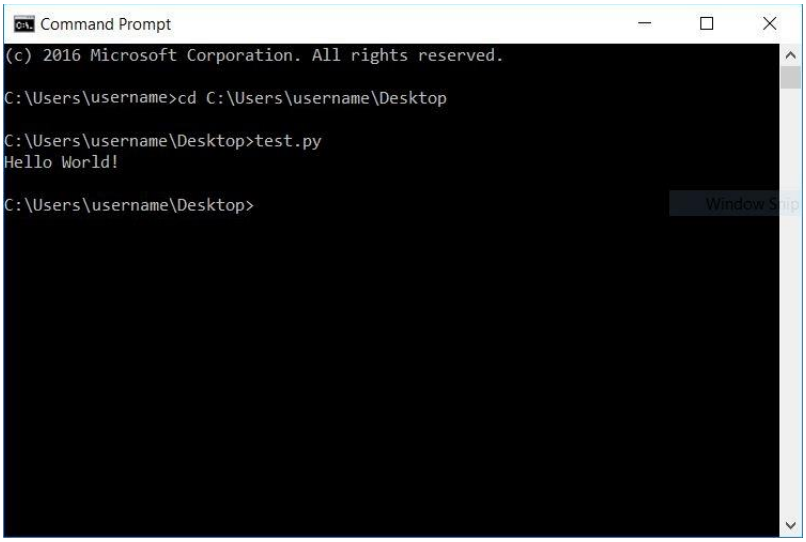
  Korisničko ime (`Username`) potrebno je zamijeniti korisničkim imenom kojim je napravljena prijava u OS.

  Naredba za pozicioniranje tada glasi:

  ```
  cd C:\Users\Username\Desktop
  ```

  Nakon što je pozicioniranje izvršeno moguće je pokrenuti *Python* skriptu tako da se u naredbeni redak ukuca njezino ime. Ako se pretpostavi da je ime skripte `test.py`, tada se ta skripta pokreće tako da se u naredbeni redak samo upiše:

```
test.py
```



Argumenti preneseni preko naredbenoga retka

Prenošenje argumenata izravno iz naredbenog retka u skriptu izuzetno je koristan način pokretanja napisanoga programskog kôda, tj. skripte.

Pokretanje skripte imena `test.py` i prenošenje argumenta `arg1`, `arg2`, `arg3` unutar naredbenoga retka radi se na način:

```
test.py arg1 arg2 arg3
```

Argumente `arg1`, `arg2`, `arg3` potrebno je zamijeniti stvarnim vrijednostima, poput: `10 20 30`. U praksi to izgleda ovako:

```
test.py 10 20 30
```

U ovom primjeru u skriptu `test.py` prenesene su tri vrijednosti. Argumenata u praksi može biti proizvoljan broj.

Ako se unutar programskog kôda žele dohvatiti parametri koji su poslani iz naredbenog retka, u programskom kôdu potrebno je uključiti paket imena `sys`.

`sys.argv` je lista koja sadržava parametre koji su predani preko naredbenoga retka skripti. Liste će detaljnije biti obrađene u poglavlju *9. Strukture podataka*.

Na poziciji indeksa 0 nalazi se ime skripte koja se pokreće, dok se na svim ostalim pozicijama nalaze predani argumenti. Programer sâm mora brinuti o tome da argumente preda ispravnim redoslijedom. Ispravan redoslijed je onaj u kojem se indeks predanog argumenta logički podudara s indeksom argumenta koji se dohvaća unutar skripte.

Sintaksa dohvaćanja argumenata unutar skripte je: `sys.argv[indeks]`, gdje indeks predstavlja redni broj

argumenta koji se želi dohvatiti. Potrebno je obratiti pažnju na to da se na indeksu 0 nalazi ime skripte koja se pokreće.

```
import sys

print("Broj danih argumenata:", len(sys.argv))

print("1. argument je vrijednosti:", sys.argv[1])
print("2. argument je vrijednosti:", sys.argv[2])
print("3. argument je vrijednosti:", sys.argv[3])

print("Ispis svih argumenata: ", str(sys.argv))
```

```
Izlaz:
     C:\Users\Username\Desktop>test.py prvi 2
     treci
     Broj danih argumenata: 4
     1. argument je vrijednosti: prviArgument
     2. argument je vrijednosti: 2
     3. argument je vrijednosti: treci
     Ispis svih argumenata:
     ['C:\\Users\\Username\\Desktop\\test.py',
     'prvi', '2', 'treci']
```

8.2. Pokretanje skripte u unaprijed zadanom trenutku (engl. *Task Scheduler*)

Ovisno o namjeni napisane skripte, ponekad je korisno imati alat koji omogućava automatsko pokretanje skripte u točno određenom vremenu ili intervalu. U ovom poglavlju bit će objašnjeno korištenje *Windows* programa *Task Scheduler*.

Task Scheduler je komponenta koja dolazi s instalacijom *Microsoft Windows* operacijskoga sustava. Njegova namjena je omogućiti automatsko izvršavanje zadataka na nekom računalu. *Task Scheduler* program može se koristiti za izvršavanje zadataka poput pokretanja napisanih skripti.

Neki od uvjeta po kojima se može dogoditi izvršavanje zadataka:

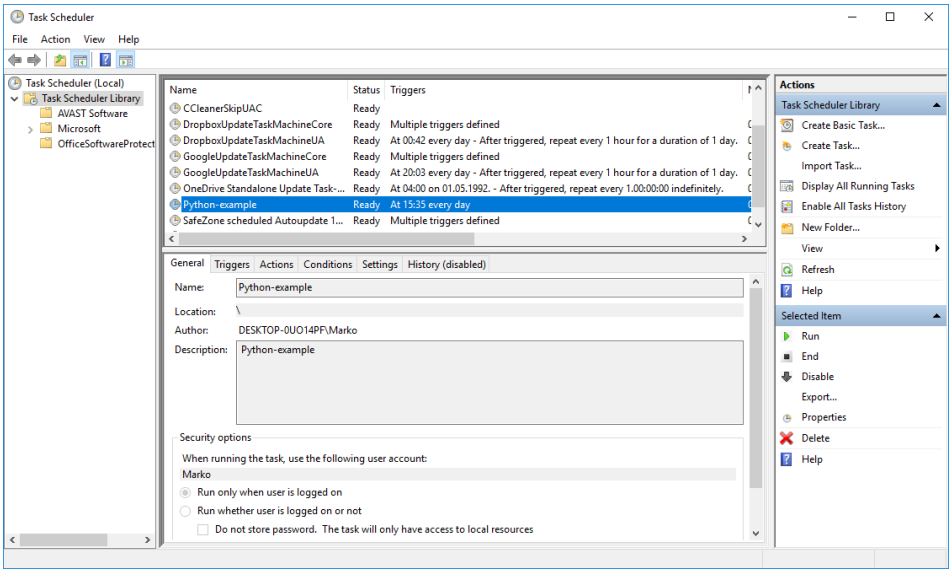
- u neko točno određeno vrijeme (jednokratno)
- u točno određeno vrijeme na dnevnoj bazi
- u točno određeno vrijeme na tjednoj bazi (moguće je odabrati dane u tjednu kada se neki zadatak izvršava),
- u točno određeno vrijeme na mjesečnoj bazi:
    - izvršavanje zadatka moguće je postaviti u točno određene dane u mjesecu
    - ili pak u točno određeni tjedan u mjesecu (1., 2., 3., 4., zadnji tjedan) + koje sve dane u tjednu

- prilikom prijave u računalo
    - za bilo kojega korisnika
    - za točno određenoga korisnika
- prilikom pokretanja računala
- nakon što računalo uđe u stanje mirovanja
- nakon detekcije nekog određenog događaja na računalu
- prilikom zaključavanja/otključavanja računala.

Program *Task Scheduler* moguće je pronaći na način na sljedećoj lokaciji:

Windows Start → Windows Administrative Tools → Task Scheduler

Nakon otvaranja programa prikazuje se *Windows* program sljedećeg izgleda:



Dizajn programa moguće je podijeliti na 3 dijela.

Uz lijevi rub nalaze se mape (direktoriji) unutar kojih je moguće smjestiti kreirane zadatke prema nekim proizvoljno smislenim cjelinama. Također, omogućeno je i kreiranje mape. Nova mapa kreira se desnim klikom na glavnu mapu ***Task Scheduler Library*** i odabirom opcije **New Folder** ili pak odabirom opcije uz desni rub **New Folder**.

Središnji dio prikazuje popis svih kreiranih zadataka, a klikom na neki željeni zadatak ispod liste zadataka prikazuju se detalji odabranoga zadatka.

S desne strane nalazi se prozor koji omogućava razne akcije, za ovaj tečaj koristit će se opcije poput **Create Task**, **Run** i **New Folder**. Opcija **Create Task** namijenjena je za kreiranje novih zadataka, dok je opcija **Run** namijenjena za ručno pokretanje kreiranoga zadatka. Ova opcija obično se koristi kada se ne želi čekati postavljeno vrijeme za

automatsko izvršavanje zadatka. Najzastupljeniji primjer za korištenje te opcije jest testiranje kreiranoga zadatka radi li on uredno.

Napomena

U ovom tečaju objašnjeno je kreiranje naprednijih zadataka (kreiranje kroz prozor koji ima više opcija). Uz takvo kreiranje zadataka postoji mogućnost kreiranja jednostavnijih zadataka pomoću "čarobnjaka".

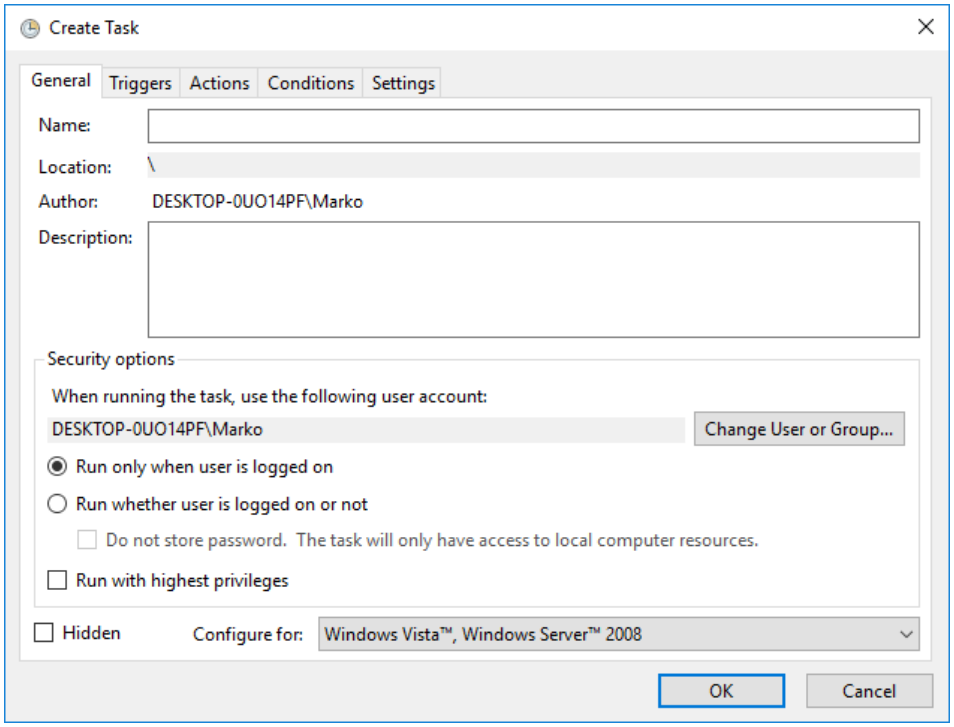
Jednostavnije zadatke moguće je kreirati klikom na opciju **Create Basic Task**.

Primjer kreiranja novoga zadatka

Radi preglednosti kreiranih zadataka neka se najprije kreira novi direktorij imena ***Python-primjeri***. To je moguće napraviti na dva načina. Desnim klikom na glavnu mapu ***Task Scheduler Library*** i odabirom opcije **New Folder** ili pak odabirom mape ***Task Scheduler Library*** i odabirom opcije uz desni rub **New Folder**.

Nakon novokreiranoga direktorija potrebno je odabrati taj direktorij i u dijelu gdje se nalazi popis akcija odabrati akciju imena **Create Task**.

Nakon odabira akcije "Create Task" otvara se prozor sljedećeg izgleda:

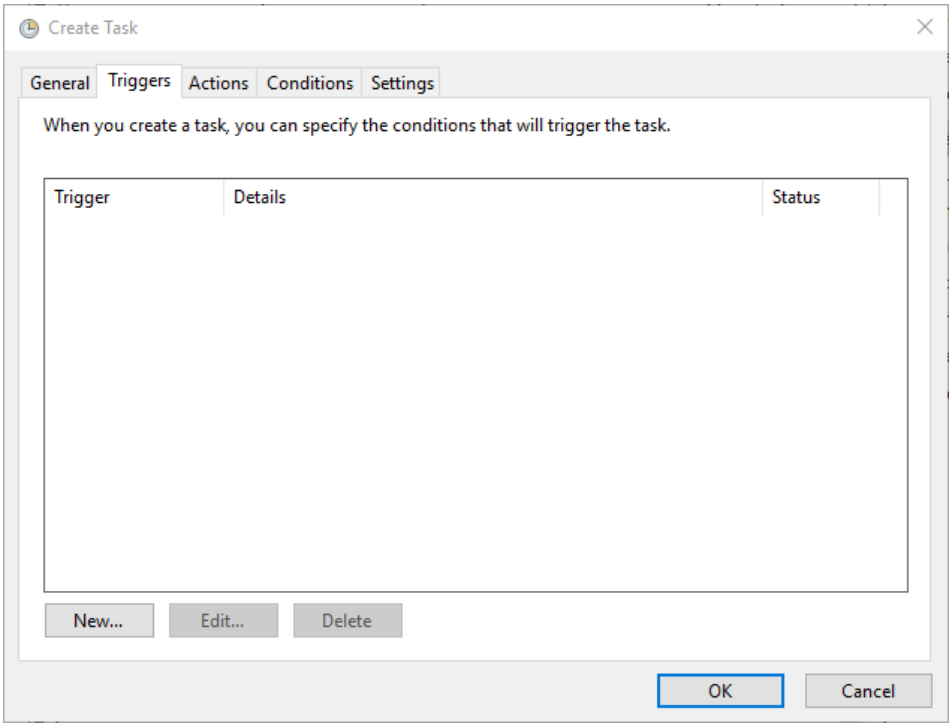


Kao što se iz priložene slike vidi, prozor se sastoji od 5 glavnih izbornika: **General**, **Triggers**, **Actions**, **Conditions** i **Settings**. U nastavku se nalazi opis što pruža svaka od prethodno navedenih stavki.

General – u ovom prozoru moguće je kreirati ime zadatka (preporuka je da ime zadatka bude deskriptivno zbog lakšeg snalaženja). Također, u ovom dijelu moguće je napisati dulji opis kreiranoga zadatka što je izrazito korisno ako postoji više zadataka koji obavljaju sličnu radnju. Zatim je moguće odabrati još nekoliko opcija, poput, je li potrebno pokretati zadatak samo ako je korisnik prijavljen ili pak to nije bitno, želi li se pokretati zadatak s najvećim privilegijama te za koji operacijski sustav se zadatak želi konfigurirati.

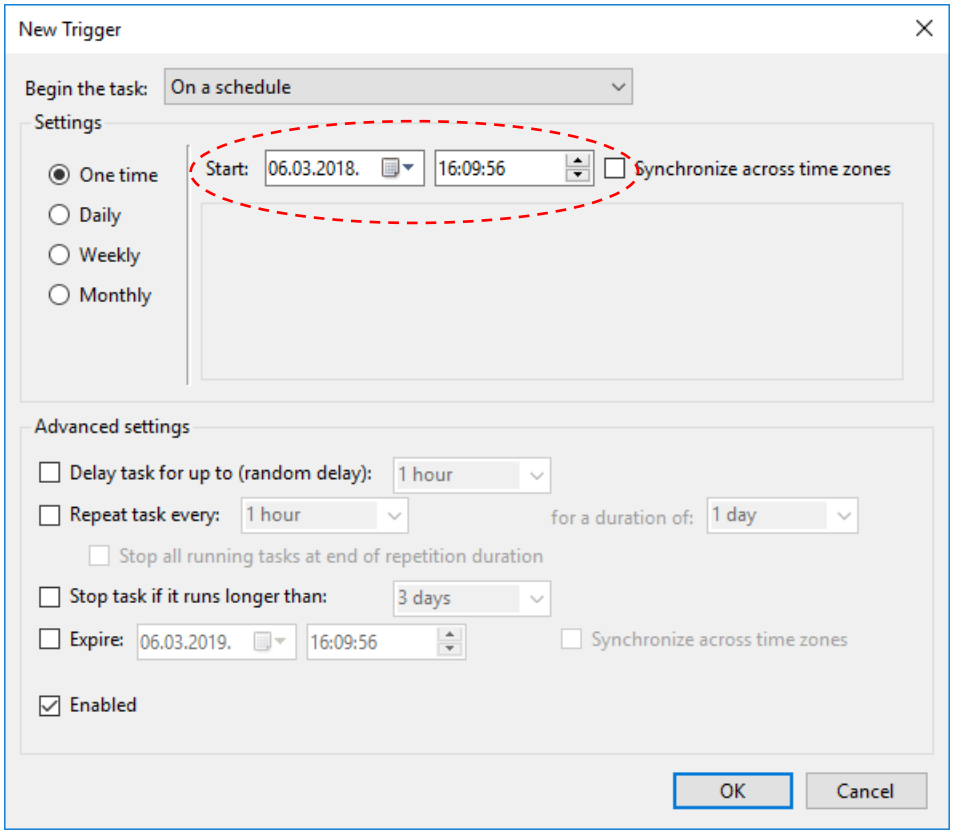
Triggers – u ovom prozoru kreiraju se okidači na temelju kojih se pokreće izvršavanje zadatka. Kreiranje okidača radi se klikom na gumb

New, odabirom okidača i klikom na gumb **Edit** moguće je urediti okidač, a odabirom i klikom na gumb **Delete** moguće je izbrisati okidač. Jedan zadatak može imati više okidača na temelju kojih se pokreće izvršavanje.



Niže se nalazi slika koja prikazuje prozor koji se otvara nakon što se klikne na gumb **New**. U ovom tečaju objasnit će se kreiranje okidača koji se okida jednokratno u točno određeno vrijeme na točno određen datum.

Kao što se na niže prikazanoj slici vidi, pri vrhu u padajućem izborniku potrebno je odabrati opciju **On a schedule** te se nakon toga u lijevom dijelu prikazuju opcije: **One time**, **Daily**, **Weekly**, **Monthly**. To su opcije koje govore na kojoj bazi se želi kreirati okidač – jednokratno, svaki dan, na tjednoj bazi ili na mjesečnoj bazi. U našem slučaju odabrano je kreiranje jednokratnog okidača – **One time**. Podaci u dijelu **Start** su podaci koji govore u točno koje vrijeme na koji datum se želi pokrenuti zadatak.



Actions – u ovom dijelu kreiraju se događaji / akcije koji će se izvršiti u onom trenutku kada se zadatak okine.

Za potrebe ovoga tečaja kreirat će se jednostavna skripta čiji je zadatak kreirati novi tekstualni dokument (*.txt) te u njega ispisati tekst "Hello World!". Kod ovakvog načina pokretanja programa bitno je obratiti pozornost na to da ne postoji mjesto poput naredbenoga retka gdje bi se rezultati obrade mogli ispisati. Zato je potrebno ispis rezultata obrade spremiti na neko trajno mjesto, a to može biti, na primjer, tekstualna datoteka.

Skripta koja kreira novu datoteku te u nju ispisuje neki tekst može biti ovakvoga sadržaja:

```
file = open("[path]/datoteka.txt", "w")
file.write("Hello World")
file.close()
```

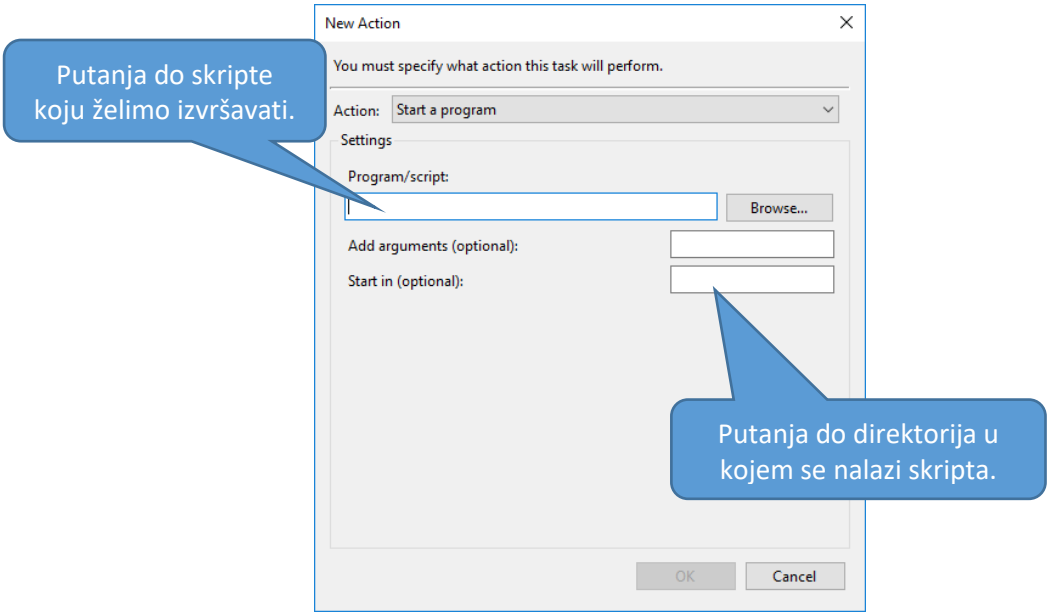
U prvoj liniji nalazi se `[path]` koji je potrebno zamijeniti putanjom do skripte. Primjer:

```
C:/Users/Username/Desktop/datoteka.txt
```

U nastavku se nalazi izgled prozora koji omogućava postavljanje skripte koja će se izvršiti prilikom izvršavanja kreiranoga zadatka.

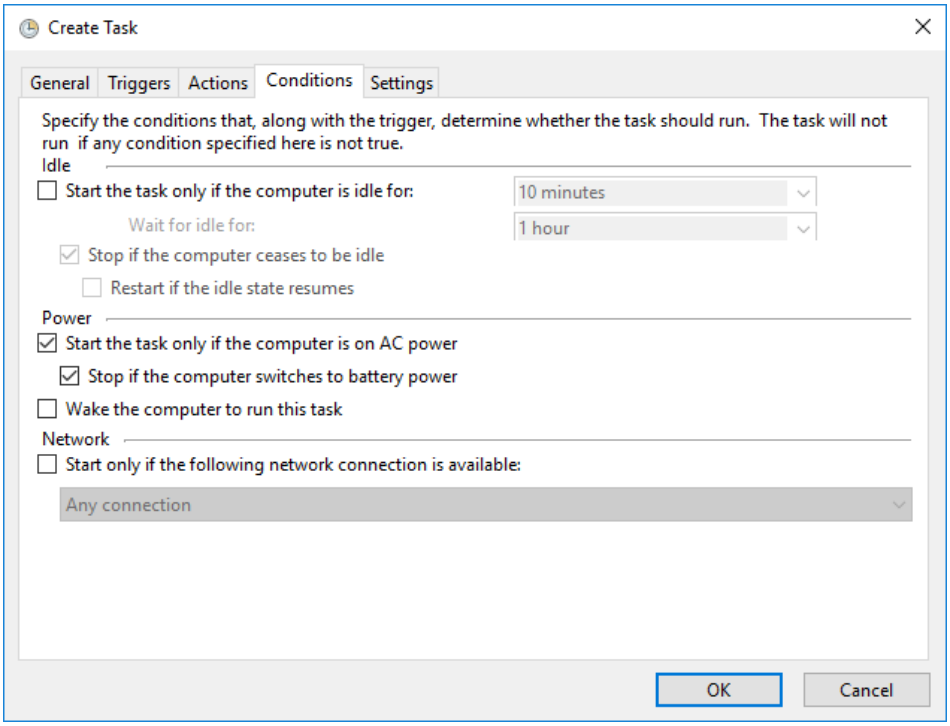
U dijelu **Action** potrebno je odabrati **Start a program** te je nakon tog odabira moguće u dijelu **Program/script** unijeti punu putanju do skripte koja se želi pokrenuti. Radi jednostavnosti i smanjenja mogućnosti greške, preporučuje se koristiti gumb **Browse**.

Kao parametar **Start in (optional)** potrebno je unijeti putanju do direktorija u kojem se skripta nalazi.

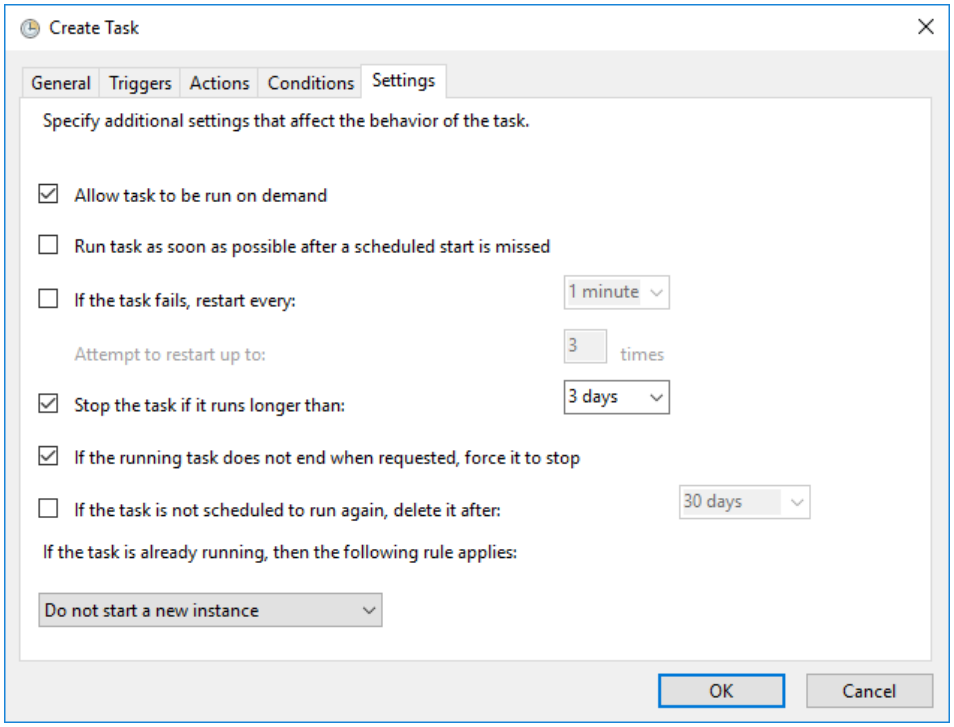


Također, unutar same skripte sve putanje do svih korištenih datoteka moraju biti apsolutne, a ne relativne, što znači da mora biti navedena puna putanja do neke željene datoteke.

Conditions – uvjeti uz kombinaciju s okidačima odlučuju mora li se neki zadatak pokrenuti. Ako je samo jedan od postavljenih uvjeta netočan, zadatak se neće pokrenuti. Za potrebe ovoga tečaja sve postavke u ovom prozoru neka ostanu nepromijenjene.



Settings – u ovom prozoru postavljaju se dodatne postavke koje će se odraziti na ponašanje kreiranoga zadatka. Najčešće korištena postavka je **Stop the task if it runs longer than**: koja nam omogućava zaustavljanje zadatka ako se on izvršava dulje od određenog vremena. Ovom postavkom sprječava se beskonačno ili neuobičajeno dugo izvršavanje zadataka. Za potrebe ovoga tečaja sve postavke u ovom prozoru neka ostanu nepromijenjene.



Nakon što su sve postavke novokreiranoga zadatka spremljene, moguće je testirati zadatak tako da se nad njim pokrene akcija **Run**. Osim ovakvog načina, moguće ga je pokrenuti namještanjem trenutka izvršavanja koji se nalazi u bliskoj budućnosti, na primjer minuta, dvije od trenutka kreiranja zadatka. Nakon što se zadatak izvrši na radnoj površini računala, mora se pojaviti tekstualna datoteka imena `datoteka.txt`.

8.3. Vježba: Naredbeni redak

1. Napišite skriptu koja od korisnika zahtjeva unos dvaju brojeva, ta skripta neka ispisuje njihov zbroj. Tako napisanu skriptu pokrenuti na oba načina pokretanja skripti opisana u ovom poglavlju. Što primjećujete kao razliku između načina pokretanja skripte dvostrukim klikom na nju i načina pokretanja skripte preko naredbenog retka?
2. Kreirajte zadatak koji će se samostalno izvršavati svaku minutu. Skripta koju će kreirani zadatak izvršiti neka ispiše prilikom svakog izvršavanja u neku proizvoljnu datoteku vrijednost "x". Obratite pozornost da se sadržaj datoteke ne smije brisati, već se mora prilikom svakog izvršavanja u datoteku dopisati jedan "x".

9. Strukture podataka

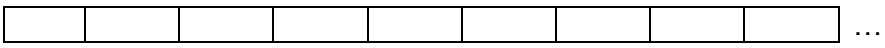
Po završetku ovog poglavlja polaznik će moći:

- *strukturirati podatke u liste, skupove, rječnike i n-terce i koristiti ih u programima.*

9.1. Lista (engl. *List*)

Liste možemo smatrati strukturama koje nam služe za organiziranje podataka u skripti. U svakodnevnom životu najprikladniji primjer jest kreiranje liste za kupovinu u trgovini. Kako bismo pojednostavili kupovinu, kreiramo dvije liste – u jednoj listi nalaze se stvari za kupnju u trgovini široke potrošnje (npr. kruh, šećer, mlijeko…), a u drugoj listi nalaze se stvari za kupnju u trgovini tehničke robe.

Za razliku od varijabli koje su obrađene u jednom od prethodnih poglavlja, ovo poglavlje bavi se listama. Liste možemo smatrati skupom (kolekcijom) varijabli koje su pohranjene u jednom objektu. Kao što je već prije opisano, u varijablu možemo spremiti samo jednu vrijednost, no to ponekad nije dovoljno. Uzmimo za primjer da imamo petlju koja učitava brojeve tako dugo dok se s tipkovnice ne unese broj 0. U tom slučaju prilikom pisanja programa nije nam unaprijed poznato koliko će brojeva korisnik unijeti te zbog toga nismo u mogućnosti unesene brojeve pohranjivati u varijable (jer ne znamo koliko varijabli različitih imena moramo kreirati). Dodatno, otegotna okolnost je to što ako koristimo klasične varijable za gore navedeni tip zadatka, tada je potrebno kontrolirati u koju od varijabli će se učitani broj upisati s obzirom na to koji se broj po redu trenutačno učitava. To je potrebno raditi kako upisana vrijednost ne bi pregazila vrijednost koju smo spremili u neku varijablu u prethodnom koraku petlje. Prednost liste je u tome da se kreira jedan objekt, tj. lista te se u tu listu jednostavno unose vrijednosti redoslijedom koji god da nam je potreban. Listu možemo zamisliti kao tablicu s jednim redom u kojem se nalaze stupci. U stupce te tablice stavljamo elemente jedan iza drugoga. Niže se nalazi grafički prikaz kako možemo zamisliti da izgleda lista u koju stavljamo elemente. Svaka od donjih ćelija može sadržavati neku vrijednost.



Liste su izrazito bitne u programiranju. U ovom poglavlju bit će opisani koncepti koji se koriste za kreiranje listi, njihovo uređivanje, traženje elemenata liste, ispis elemenata liste i ostale stvari potrebne za rad s listama.

Prilikom kreiranja liste, potrebno je posvetiti pažnju tomu što će biti spremljeno u svaku pojedinu listu prema nekoj logici, tj. potrebno je obratiti pažnju na organizaciju podataka.

Svaka stavljena vrijednost u listu dobiva svoj indeks te se dodavanjem svakoga novog elementa ta vrijednost povećava za 1. **Indeksi uvijek kreću uvijek od 0,** a ne od 1 kako bi se moglo očekivati. Recimo da se u listi nalaze imena 5 osoba:

Ivan	Luka	Nino	Jan	Tin
0	1	2	3	4

Ime osobe koja se nalazi prva u listi (Ivan) započinje indeksom 0, a zadnji element u listi (Tin) nalazi se na 4. mjestu, tj. dobit će indeks 4.

Za razliku od nekih drugih programskih jezika, na primjer programskog jezika *C*, kod *Pythona* nije potrebno definirati koji tip vrijednosti će biti pohranjen u elemente liste. *Python* dopušta da se u elemente jedne te iste liste pohranjuju vrijednosti različitih tipova.

Ivan	55	100.55
0	1	2

Iz gornjeg primjera je vidljivo da se na 0. indeksu nalazi niz znakova, na 1. indeksu nalazi se cijeli broj, a na 2. indeksu nalazi se decimalni broj. Iako je ovakvo miješanje tipova podataka unutar jedne liste dozvoljeno, dobra praksa je da se kreiraju liste čiji će svi elementi biti istoga tipa podataka. No, ako se svejedno želi koristiti lista s miješanim tipovima podataka, tada je prije svake akcije nad nekom vrijednosti preporučljivo provjeriti koji tip podatka se obrađuje.

9.1.1. Izrada liste

Sintaksa definiranja liste je:

```
imeListe = [element1, element2, …]
```

Ako se želi definirati prazna lista, to se radi na sljedeći način:

```
imeListe = []
```

Kako bi bilo moguće, na primjer, ispisati vrijednosti elemenata liste, najprije je listu potrebno kreirati. Niže se nalazi primjer kreiranja liste.

```
lista = ["Ivan", 55, 100.55]
print(lista)
```

```
Izlaz:
     ['Ivan', 55, 100.55]
```

Iz gore navedenoga primjera vidimo da se u prvoj liniji kreira lista od 3 elementa. Kreirana lista sastoji se od elemenata koji su različitoga tipa podataka (niz znakova, cijeli broj i decimalni broj). Elementi liste stavljaju se unutar uglatih zagrada. Ako se želi kreirati prazna lista, tada se može napisati samo `lista = []`. Pozivom funkcije `print()`, kojoj se kao jedini argument šalje prethodno kreirana lista, ispisuju se svi elementi liste u jednostavnom i preglednom obliku.

9.1.2. Dohvaćanje elemenata liste

Ako u listi imamo pohranjene vrijednosti, pretpostavka je da im želimo i pristupiti, jer čemu imati listu ako po listi nije moguće iterirati i dohvaćati točno određene elemente. U prethodnom dijelu prikazano je kreiranje liste i ispisivanje svih elemenata liste pomoću funkcije `print()`. U većini slučajeva samo ispis svih elemenata liste logici programa i ne donosi dodanu vrijednost. Zato pomoću indeksa elemenata liste možemo dohvaćati element po element.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]
print(lista[1])
```

```
Izlaz:
      Luka
```

Gore prikazani odsječak programskoga kôda sprema u listu 5 imena. Tih 5 imena u listu će biti spremljeno na sljedeći način:

Ivan	Luka	Nino	Jan	Tin
0	1	2	3	4

U drugoj liniji kôda vidimo da se poziva funkcija `print()`, kao argument funkcije traži se ime osobe koja ima indeks 1. Sintaksa dohvaćanja jednog elementa s neke točno određene pozicije jest: `imeListe[indeks]`. Najprije se napiše ime liste iz koje želimo dohvatiti neku vrijednost te se nakon toga u uglatim zagradama napiše indeks elementa koji se želi dohvatiti.

Gore korištena lista sastoji se od 5 elemenata, ponekad programeri žele dohvatiti više elemenata odjednom u jednom koraku. To se radi tako da se koristi sintaksa: `imeListe[pocetak:kraj]`. Vrijednost `pocetak` označava indeks od kojeg kreće ispisivanje, dok kod vrijednosti `kraj` to nije tako. Vrijednost `kraj` je uvijek za 1 broj veća od indeksa zadnjeg elementa koji se ispisuje. Također, od tih dviju vrijednosti (`pocetak` i `kraj`) jednu možemo izostaviti. Ako se izostavi vrijednost početak, to označava da će se dohvatiti elementi od početka liste pa do indeksa (`kraj-1`). Ako se pak izostavi vrijednost `kraj`, to znači da će se dohvatiti elementi od indeksa vrijednosti `pocetak`, pa sve do kraja liste.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

print(lista[1:3])
print(lista[:3])
print(lista[2:])
```

```
Izlaz:
      ['Luka', 'Nino']
      ['Ivan', 'Luka', 'Nino']
      ['Nino', 'Jan', 'Tin']
```

9.1.3. Popis akcija koje se mogu napraviti nad listom

Osim često korištenih akcija nad listom, ponekad su potrebne i akcije koje programer ne koristi često i jednostavno ih smetne s uma. Ako se na jednostavan način želi dobiti popis svih akcija koje je moguće napraviti nad listom, to je moguće dobiti tako da se pozove funkcija `dir()` koja kao argument prima listu. Funkcija `dir()` ispisuje akcije koje je moguće napraviti nad predanim objektom.

Ako se funkcija `dir()` pozove unutar samog programskog kôda (skripte). Na zaslon nam se neće ispisati ništa. Funkcija `dir()` mora se pozvati unutar naredbenoga retka. Sjetimo se poglavlja za pripremu radnog okruženja, u tom poglavlju najprije smo otvarali *IDLE*. Te smo iz *IDLE-a* otvarali program za unos teksta, tj. programskoga kôda. *IDLE* nije ništa drugo nego naredbeni redak u kojem se naredbe izvršavaju jedna po jedna. *IDLE* naredbeni redak zove se još i *Python Shell*.

Start → Python 3.7 → IDLE (Python 3.7 32-bit)

```
>>> lista = ["Ivan", 55, 100.55]
>>> dir(lista)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'indeks', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

U *Python Shellu* najprije ukucamo liniju:

```
lista = []
```

te nakon toga pozovemo funkciju `dir()`:

```
dir(lista)
```

Iz gornjeg popisa u ovom trenutku bitna su nam samo imena akcija (metoda) koja ne počinju i završavaju s "__", a to su akcije: *append*, *clear*, *copy*, *count*, *extend*, *index*, *insert*, *pop*, *remove*, *reverse*, *sort*.

- `append()` – dodaje novi element na kraj liste
- `clear()` – uklanja sve elemente iz liste
- `copy()` – kopira sve elemente liste u drugu listu
- `count()` – vraća koliko ima elemenata u listi neke određene vrijednosti
- `extend()` – proširuje postojeću listu s elementima neke druge liste

- `index()` – pronalazi i vraća indeks predanog elementa (ako ima više vrijednosti, vraća indeks najmanje vrijednosti)
- `insert()` – stavlja vrijednost u listu na točno određenu poziciju
- `pop()` – iz liste uklanja vrijednost koja se nalazi na kraju liste
- `remove()` – iz liste uklanja vrijednost koja se nalazi na točno određenom indeksu
- `reverse()` – ova metoda mijenja pozicije elemenata u suprotni raspored (prvi element postaje zadnji, drugi element postaje predzadnji itd.)
- `sort()` – metoda koja sortira elemente liste.

9.1.4. Iteriranje kroz elemente liste

Najjednostavniji način koji omogućava da se prođe kroz sve elemente liste (ili barem dio njih) je korištenjem petlje `for`. Petlji predamo cijelu listu te ona dohvaća element po element iz liste, stavlja ga u varijablu proizvoljnog imena, u donjem primjeru, ta varijabla je imena `element` te nakon toga u tijelu petlje možemo raditi obradu dobivene vrijednosti iz liste.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

for element in lista:
    print(element)
```

```
Izlaz:
      Ivan
      Luka
      Nino
      Jan
      Tin
```

9.1.5. Dodavanje novog elementa u listu

U gornjim primjerima listu smo popunjavali elementima odmah prilikom deklaracije liste. No, ako kasnije tijekom programa želimo dodavati listi nove elemente, za to moramo koristiti ugrađene metode. Za dodavanje novog elementa u listu koristi se metoda `append()`. Sintaksa dodavanja novog elementa je:

```
imeListe.append(novaVrijednost)
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

print(lista)
lista.append("NOVI")
print(lista)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Jan', 'Tin']
      ['Ivan', 'Luka', 'Nino', 'Jan', 'Tin',
      'NOVI']
```

9.1.6. Uklanjanje svih elemenata iz liste

Ako želimo isprazniti cijelu listu, to se radi pomoću metode `clear()`. Sintaksa za uklanjanje svih elemenata iz liste je:

```
imeListe.clear()
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

print(lista)
lista.clear()
print(lista)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Jan', 'Tin']
      []
```

9.1.7. Kopiranje svih elemenata liste u novu listu

Pomoću metode `copy()` možemo u novu listu kopirati sve elemente neke postojeće liste. Ovakav način kopiranja pregazit će sve postojeće elemente koji se nalaze u listi u koju kopiramo vrijednosti iz neke druge liste. Sintaksa kopiranja liste u novu listu je:

```
novaLista = lista.copy()
```

Ovaj način je izrazito jednostavan, no ako ne želimo kopirati sve elemente u novu listu, već samo određene, to se radi pomoću petlje `for` i dodavanjem odabranih elemenata u novu listu.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]
novaLista = []

print(novaLista)
novaLista = lista.copy()
print(novaLista)
```

```
Izlaz:
      []
      ['Ivan', 'Luka', 'Nino', 'Jan', 'Tin']
```

9.1.8. Broj elemenata određene vrijednosti

Ako želimo na jednostavan način prebrojiti koliko ima elemenata neke određene vrijednosti, to možemo napraviti pomoću metode `count()`. Povratna vrijednost je broj pronađenih elemenata određene vrijednosti. Sintaksa koja vraća broj elemenata određene vrijednosti je:

```
imeListe.count(vrijednost)
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Luka"]
print(lista.count("Luka"))
```

```
Izlaz:
      2
```

9.1.9. Proširivanje liste elementima druge liste

Za razliku od metode `copy()` koja briše sve postojeće elemente u listi, metoda `extend()` proširuje listu novim elementima.

Sintaksa je sljedeća:

```
novaLista.extend(lista)
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]
novaLista = [123, 456]

print(novaLista)
novaLista.extend(lista)
print(novaLista)
```

```
Izlaz:
      [123, 456]
      [123, 456, 'Ivan', 'Luka', 'Nino', 'Jan',
      'Tin']
```

9.1.10. Vježba: Liste (1)

3. Napišite listu `dani` s popisom radnih dana u tjednu. Budući da listi nedostaje vikend, dodajte ga te ispišite listu.
4. Napravite listu `radniDani` kopiranjem liste `dani` te obrišite nepotrebne elemente liste.
5. Napravite listu `vikend` iteriranjem po elementima liste `dani`.
6. Koristeći metodu extend spojite liste `radniDani` i `vikend`.

9.1.11. Traženje elementa

Metoda `index()` omogućava pronalazak indeksa na kojem se neka tražena vrijednost nalazi. Potrebno je obratiti pažnju na to da ako u listi ima više elemenata s traženom vrijednošću, ona će vratiti indeks prve pronađene vrijednosti. Povratna vrijednost metode `index()` je pozicija tražene vrijednosti, tj. indeks.

Sintaksa je sljedeća:

```
lista.index(trazenaVrijednost)
```

```
lista = ["Ivan", "Luka", "Nino", "Luka"]
print(lista.index("Luka"))
```

```
Izlaz:
      1
```

9.1.12. Dodavanje novog elementa na točno određenu poziciju

Pomoću metode `append()` u listu dodajemo novu vrijednost, no ta nova vrijednost se uvijek nalazi na zadnjoj poziciji. Ako se nova vrijednost želi

staviti na točnu određenu poziciju, koristi se metoda `insert()`. Ova metoda sve elemente od pozicije na koju se stavlja neka vrijednost pomiče za jedno mjesto udesno, tj. svi oni se sele na mjesta pomaknuta za jedan indeks više od pozicije prije umetanja novog elementa.

Sintaksa korištenja metode `append()` je:

```
lista.append(vrijednost)
```

Sintaksa korištenja metode `insert()` je:

```
lista.insert(zeljenaPozicija, vrijednost)
```

```
lista = ["Ivan", "Luka", "Nino", "Luka"]

lista.insert(2, "NOVO")
print(lista)
```

```
Izlaz:
      ['Ivan', 'Luka', 'NOVO', 'Nino', 'Luka']
```

9.1.13. Uklanjanje elementa s neke pozicije

Zadnji element u listi uklanja se pomoću metode `pop()`. Sintaksa uklanjanja zadnjeg elementa iz liste je:

```
lista.pop()
```

```
lista = ["Ivan", "Luka", "Nino", "Luka"]

print(lista)
lista.pop()
print(lista)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Luka']
      ['Ivan', 'Luka', 'Nino']
```

Ako se želi ukloniti element s neke točno određene pozicije, metodi `pop()` potrebno je prenijeti indeks pozicije. Sintaksa uklanjanja elementa iz liste s točno određene pozicije:

```
lista.pop(pozicija)
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

print(lista)
lista.pop(1)
print(lista)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Jan', 'Tin']
      ['Ivan', 'Nino', 'Jan', 'Tin']
```

9.1.14. Uklanjanje elementa određene vrijednosti

Metoda `pop()` uklanja element s određene pozicije. No, nekad se želi ukloniti element određene vrijednosti te se to radi pomoću metode `remove()`. Sintaksa je sljedeća:

```
lista.remove(vrijednost)
```

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Luka"]

print(lista)
lista.remove("Luka")
print(lista)
```

```
Izlaz:
     ['Ivan', 'Luka', 'Nino', 'Jan', 'Luka']
     ['Ivan', 'Nino', 'Jan', 'Luka']
```

Primijetimo da je u gornjem primjeru, iako se u listi nalaze dva elementa vrijednosti "Luka", izbačen samo prvi element zadane vrijednosti (element koji se nalazi na nižem indeksu). Ako se želi izbaciti i drugi element zadane vrijednosti, potrebno je ponovo pozvati metodu `remove()`.

Izbacivanje svih vrijednosti "Luka" iz liste moguće je napraviti pozivanjem metode `remove()` tako dugo dok "Luka" postoji u listi. Broj pozivanja metode `remove()` moguće je kontrolirati pomoću `while` petlje. Petlja se mora okretati tako dugo dok se "Luka" nalazi u listi.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Luka"]

print(lista)
while "Luka" in lista:
    lista.remove("Luka")
print(lista)
```

```
Izlaz:
     ['Ivan', 'Luka', 'Nino', 'Jan', 'Luka']
     ['Ivan', 'Nino', 'Jan']
```

9.1.15. Promjena redoslijeda elemenata unutar liste

Ako se želi zamijeniti redoslijed elemenata unutar liste tako da prvi element postane zadnji, drugi predzadnji itd., to je moguće napraviti pozivom metode `reverse()`. Sintaksa za promjenu redoslijeda elemenata unutar liste je:

```
lista.reverse()
```

```
lista = [1, 5, 4, 7, 8, 2, 3, 2]

print(lista)
lista.reverse()
print(lista)
```

```
Izlaz:
     [1, 5, 4, 7, 8, 2, 3, 2]
     [2, 3, 2, 8, 7, 4, 5, 1]
```

9.1.16. Sortiranje elemenata u listi

Metoda koja sortira sve elemente unutar liste po vrijednosti je `sort()`. Sintaksa za sortiranje elemenata unutar liste je:

```
lista.sort()
```

```
lista = [1, 5, 4, 7, 8, 2, 3, 2]

print(lista)
lista.sort()
print(lista)
```

```
Izlaz:
      [1, 5, 4, 7, 8, 2, 3, 2]
      [1, 2, 2, 3, 4, 5, 7, 8]
```

9.1.17. Vježba: Liste (2)

1. Napišite listu `gradovi` s barem pet gradova. Na toj listi isprobajte metode `reverse()` i `sort()`.
2. Na prvu poziciju u listi `gradovi` nadodajte grad koji se nalazi na zadnjoj poziciji te zatim pobrišite oba.
3. Ispišite sve elemente liste `gradovi` tako da se svaki nalazi u svom retku.

9.2. Skup (engl. *Set*)

Skup je kolekcija koja unutar sebe ne može sadržavati duple objekte. Drugim riječima, sve vrijednosti unutar nekog skupa su jedinstvene.

Ova struktura podataka podržava matematičke operacije poput unije, presjeka, skupovne razlike, komplementa presjeka. Ove operacije ne odnose se samo na brojeve, već i na sve ostale tipove podataka.

Sintaksa definiranja skupa je:

```
imeSkupa = {element1, element2, …}
```

Ako se želi definirati prazan skup, to se radi na način:

```
imeSkupa = set()
```

```
skup = {1, 5, 1, 5, 4, 7, 8, 2, 3, 2}
print(skup)
```

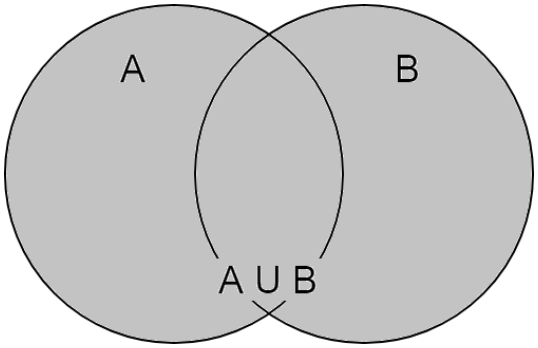
```
Izlaz:
      {1, 2, 3, 4, 5, 7, 8}
```

Iz gornjeg primjera vidljivo je da iako kod inicijalizacije postoje vrijednosti koje se ponavljaju, prilikom ispisa elemenata skupa imena `skup` više nema nijedne ponavljajuće vrijednosti već su sve vrijednosti jedinstvene.

Kao što je gore već spomenuto, nad skupom je moguće raditi matematičke operacije.

9.2.1. Unija

Unija elemenata je skup svih elemenata koji su članovi ili skupa A ili skupa B. Označava se s A | B ili matematički A U B.

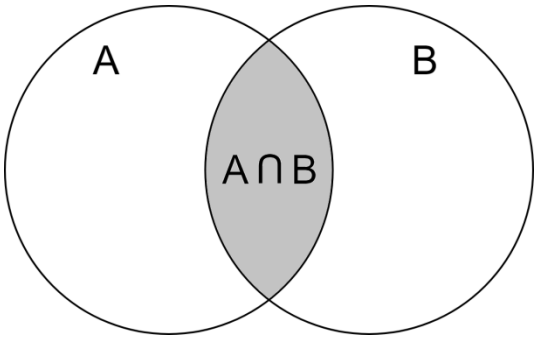



```
skup1 = {1, 5, 1, 5, 4, 7, 8, 2, 3, 2}
skup2 = {1, 2, 3, 6}

print(skup1 | skup2)
```

```
Izlaz:
      {1, 2, 3, 4, 5, 6, 7, 8}
```

9.2.2. Presjek

Presjek elemenata je skup elemenata koji su članovi i skupa A i skupa B. Označava se s A & B ili matematički A ∩ B.

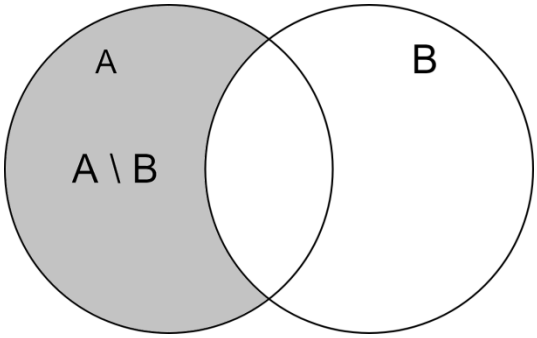



```
skup1 = {1, 5, 1, 5, 4, 7, 8, 2, 3, 2}
skup2 = {1, 2, 3, 6}

print(skup1 & skup2)
```

```
Izlaz:
      {1, 2, 3}
```

9.2.3. Skupovna razlika

Skupovna razlika skupa A i B je skup svih elemenata koji su članovi skupa A, ali nisu članovi skupa B. Označava se s A – B ili matematički A \ B.



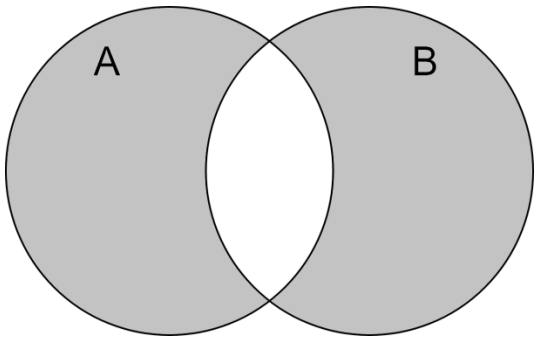

```
skup1 = {1, 5, 1, 5, 4, 7, 8, 2, 3, 2}
skup2 = {1, 2, 3, 6}

print(skup1 - skup2)
print(skup2 - skup1)
```

```
Izlaz:
      {8, 4, 5, 7}
      {6}
```

Kod prvog ispisa skupovne razlike (`skup1 - skup2`) vidi se da su rezultat brojevi `{8, 4, 5, 7}`, a to su brojevi koji su se pojavili u skupu imena `skup1`, a nisu se pojavili u skupu imena `skup2`.

9.2.4. Komplement presjeka

Komplement presjeka je skup elemenata koji se nalaze u skupu A ili skupu B, ali se istovremeno ne nalaze i u A i u B skupu. Označava se s A ^ B ili matematički (A U B)'.




```
skup1 = {1, 5, 1, 5, 4, 7, 8, 2, 3, 2}
skup2 = {1, 2, 3, 6}

print(skup1 ^ skup2)
```

```
Izlaz:
      {4, 5, 6, 7, 8}
```

9.2.5. Vježba: Skup

1. Napravite 2 skupa podataka i napunite ih proizvoljnim vrijednostima te nakon toga napravite uniju i presjek tih skupova te ispišite rezultat i za uniju i za presjek.

9.3. Rječnik (engl. *Dictionary*)

Rječnici u *Pythonu* funkcioniraju po principu ključa. Svaki zapis možemo podijeliti na dva dijela: ***ključ : vrijednost***. Preko ključa dohvaća se vrijednost koja pripada zadanom ključu. Uzmimo primjer s mjesecima, ključ je vrijednost koja je predstavljena brojem mjeseca u godini (1. mjesec, 2. mjesec…), a vrijednost je naziv mjeseca (siječanj, veljača…). Sintaksa definiranja rječnika je da unutar vitičastih zagrada navedemo parove *ključ : vrijednost*. Parovi su međusobno odvojeni zarezom.

Sintaksa definiranja rječnika je:

```
imeRjecnika = {kljuc1 : vrijednost1, kljuc2 :
vrijednos2, kljuc3 : vrijednost3, …}
```

Ako se želi definirati prazan rječnik, to se radi na način:

```
imeRjecnika = {}
```

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj", 7: "Srpanj", 8 : "Kolovoz",9 : "Rujan",
10 : "Listopad",11 : "Studeni", 12 : "Prosinac"}

print( rjecnik )
```

```
Izlaz:
     {1: 'Siječanj', 2: 'Veljača', 3: 'Ožujak', 4:
     'Travanj', 5: 'Svibanj', 6: 'Lipanj', 7:
     'Srpanj', 8: 'Kolovoz', 9: 'Rujan', 10:
     'Listopad', 11: 'Studeni', 12: 'Prosinac'}
```

Inicijalno pridruživanje vrijednosti rječniku radi se na sličan način kao i pridruživanje vrijednosti u listu, osim što su kod rječnika vrijednosti označene vitičastim zagradama `{ }`. Također, svaki element unutar rječnika sastoji se od para *ključ : vrijednost*.

Potrebno je obratiti pozornost na sljedeće stvari:

- Vrijednost ključa mora biti jedinstvena. Ako se želi staviti podatak čiji ključ već postoji unutar rječnika, postojeća vrijednost koja pripada tom ključu zamijenit će se novom vrijednosti.
- Ključ može biti bilo koji tip podatka (niz znakova, brojevi, n-torke…), no kao ključ nije moguće koristiti listu podataka. U jednom te istom rječniku kao ključ moguće je kombinirati različite tipove podataka.
- Vrijednost može biti bilo kojega tipa podatka.

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

print(rjecnik[2])
print(rjecnik[6])
```

```
Izlaz:
     Veljača
     Lipanj
```

Dohvaćanje elemenata rječnika radi se tako da se najprije napiše naziv rječnika i zatim se u uglatim zagradama stavi ključ vrijednosti koju želimo dohvatiti. Sintaksa: `nazivRjecnika[kljuc]`.

9.3.1. Dohvaćanje popisa ključeva

Pozivom metode `keys()` moguće je dohvatiti listu svih ključeva koji se nalaze unutar rječnika.

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

print(rjecnik.keys())
```

```
Izlaz:
      dict_keys([1, 2, 3, 4, 5, 6])
```

9.3.2. Dodavanje novoga para

Pozivom metode `update()` u rječnik se stavlja novi par vrijednosti.

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

rjecnik.update({7 : "Srpanj"})
print(rjecnik.keys())
```

```
Izlaz:
      dict_keys([1, 2, 3, 4, 5, 6, 7])
```

9.3.3. Dohvaćanje popisa vrijednosti

Pozivom metode `values()` dohvaća se lista vrijednosti spremljena u rječnik.

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

print(rjecnik.values())
```

```
Izlaz:
      dict_values(['Siječanj', 'Veljača', 'Ožujak',
      'Travanj', 'Svibanj', 'Lipanj'])
```

9.3.4. Brisanje vrijednosti iz rječnika

Ako se neki par (*ključ : vrijednost*) želi pobrisati iz rječnika, sintaksa je:

```
del imeRjecnika[kljuc]
```

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

del rjecnik[1]
```

```
del rjecnik[2]
print(rjecnik)
```

```
Izlaz:
      {3: 'Ožujak', 4: 'Travanj', 5: 'Svibanj', 6:
      'Lipanj'}
```

9.3.5. Dohvaćanje broja elemenata u rječniku

Dohvaćanje broja elemenata spremljenih u nekom rječniku radi se pozivom funkcije `len()`.

```
rjecnik = {1 : "Siječanj", 2 : "Veljača", 3 :
"Ožujak", 4 : "Travanj", 5 : "Svibanj", 6 :
"Lipanj"}

print(len(rjecnik))
```

```
Izlaz:
      6
```

Napomena: postoji još mnogo funkcija i metoda za rad s rječnicima, no one se u ovom tečaju neće obrađivati. Popis mogućih akcija nad rječnikom moguće je dobiti pozivom funkcije `dir()` koja je prethodno objašnjena u tečaju, a pojašnjenje o tome što koja funkcija ili metoda radi ili na koji način se koristi moguće je pronaći bilo u službenoj *Python* dokumentaciji ili pak pretraživanjem Interneta.

9.3.6. Vježba: Rječnik

1. Napravite rječnik `osoba` s ključevima `ime`, `prezime`, `godine`. Vrijednosti pridružene ključevima odredite sami. Ispišite samo vrijednosti u rječniku koristeći *for* petlju.
2. Dodajte novi par {*ključ*:*vrijednost*}. Ključ može biti adresa, oib ili nešto slično.
3. Izbrišite element *`godine`*.

9.4. N-terac (engl. *Tuple*)

N-terac je tip podatka koji nam omogućava da u njega pohranjujemo skupove vrijednosti. U načelu n-terac se ponaša na način kao i lista, ali s jednom bitnom razlikom. Razlika je ta da je n-terac nepromjenjiv (engl. *immutable*) tip podatka za razliku od liste koja je promjenjiva. Također, liste koriste uglate zagrade, dok se kod pridruživanja vrijednosti u n-terac koriste oble zagrade. Unutar n-teraca vrijednosti se odvajaju zarezom.

Sintaksa za definiranje n-terca je:

```
imeNterca = (element1, element2, element3, …)
```

Ako se želi definirati prazan n-terac, to se radi na način:

```
imeNterca = ()
```

```
nTerac = ('a', 'bcd', 123)
print(nTerac)
```

```
Izlaz:
      ('a', 'bcd', 123)
```

9.4.1. Dohvaćanje vrijednosti iz n-terca

Iz n-terca se vrijednosti dohvaćaju tako da se navede ime objekta, tj. n-terca te se u uglatim zagradama navodi indeks elementa koji se želi dohvatiti. Sintaksa za dohvaćanje vrijednosti iz n-terca:

```
imeNterca[indeks]
```

```
nTerac = (1, 2, 345, 567)
print(nTerac[2])
```

```
Izlaz:
      345
```

Također, moguće je dohvatiti i više elemenata odjednom. Na primjer:

```
imeNterca[pocetniIndeks : zavrsniIndeks]
```

Početni indeks je vrijednost od koje (uključujući) se želi dohvatiti elemente, a završni indeks je uvijek vrijednost koja je uvećana za jedan ovisno od indeksa zadnjeg elementa koji se želi dohvatiti. Na primjer, ako se žele ispisati vrijednosti na indeksima 1, 2, 3, tada se u uglatim zagradama mora napisati `[1:4]`. Također, može se izostaviti bilo vrijednost početnog indeksa, bilo vrijednost završnog indeksa. Ako se izostavi vrijednost početnog indeksa, tada se dohvaćaju svi elementi od početka n-terca. Ako se pak izostavi vrijednost završnog indeksa, tada se dohvaćaju svi elementi sve do kraja n-terca.

```
nTerac = (1, 2, 345, 567)
print(nTerac[0:3])
```

```
Izlaz:
      (1, 2, 345)
```

Ako u jednom koraku želimo u više različitih varijabli pridružiti vrijednosti koje se nalaze u n-tercu, to je moguće napraviti korištenjem sintakse:

```
var1, var2, var3 = nTerac
```

```
nTerac = (11, 22, 33)
var1, var2, var3 = nTerac

print(var1, var2, var3)
```

```
Izlaz:
      11 22 33
```

Kao što možemo vidjeti, unutar n-terca nalaze se tri vrijednosti. Te se vrijednosti spremaju u tri varijable: `var1`, `var2`, `var3`. Vrijednosti se u varijable spremaju identičnim redoslijedom kako se nalaze unutar n-terca.

9.4.2. Promjena vrijednosti unutar n-terca

Kao što je u uvodu navedeno, n-terac je nakon što je jednom kreiran nepromjenjiv, tj. nije više moguće mijenjati vrijednosti koje se nalaze unutar n-terca. Niže se nalazi prikaz kôda u kojem se pokušava promijeniti n-terac. U izlazu pokrenutoga kôda vidi se da to nije moguće napraviti.

```
nTerac = (11, 22, 33)
nTerac[0] = 0
```

```
Izlaz:
      Traceback (most recent call last):
        File "C:/test.py", line 2, in <module>
          nTerac[0] = 0
      TypeError: 'tuple' object does not support
      item assignmen
```

9.4.3. Brisanje elemenata n-terca

Kako nije moguće promijeniti vrijednost nekog elementa unutar n-terca, tako nije moguće ni izbrisati neki element unutar n-terca.

Moguće je "uništiti" cijeli n-terac. N-terac se uništava pomoću ključne riječi `del`. U niže napisanom kôdu nad varijablom `nTerac` pozvana je naredba `del`. Nakon toga taj isti `nTerac` se pokušava ispisati i vidimo da to više nije moguće jer `nTerac` više ne postoji.

```
nTerac = (11, 22, 33)

print(nTerac)
del nTerac
print("N-terac nakon brisanja:")
print(nTerac)
```

```
Izlaz:
      (11, 22, 33)
      N-terac nakon brisanja:
      Traceback (most recent call last):
        File "C:/test.py", line 5, in <module>
          print(nTerac)
      NameError: name 'nTerac' is not defined
```

9.4.4. Vježba: N-Terac

1. Napravite n-terac u kojem su pohranjeni samoglasnici. Ispišite prva tri samoglasnika.
2. Isprobajte promijeniti ili izbrisati element n-terca.

9.5. Vježba: Rad sa strukturama podataka

1. Napišite program koji se sastoji od liste. Elementi liste neka budu proizvoljnih vrijednosti. Ispišite samo one elemente koji se nalaze na parnom indeksu.
2. Napišite program koji se sastoji od liste. Program s tipkovnice čita vrijednosti i tako pročitane vrijednosti sprema u listu. Učitavanje prekinuti onoga trenutka kada korisnik unese broj 5.
3. Nadogradite prethodni zadatak tako da ako je korisnik unio manje od 6 vrijednosti u listu, na primjer korisnik je unio 3 vrijednosti (četvrta unesena vrijednost je 5), ostale 3 vrijednosti neka se postave na predefiniranu vrijednost, a to je 0.
4. Prethodni zadatak nadogradite tako da nakon završetka upisivanja petlja ispiše sve elemente liste zajedno s njihovim pripadajućim indeksima. Primjer:

```
Unesite broj: 1
Unesite broj: 2
Unesite broj: 3
Unesite broj: 4
Unesite broj: 5
[0] = 1
[1] = 2
[2] = 3
[3] = 4
[4] = 0
[5] = 0
```

5. Napravite hrvatsko-engleski rječnik. Ključ podataka neka bude hrvatska riječ, a vrijednost toga ključa neka bude engleska riječ. Napuniti rječnik s 5 elemenata.
   Napraviti beskonačnu petlju koja učitava s tipkovnice hrvatske riječi. Za svaku učitanu riječ (ako prijevod postoji) ispisati prijevod, a ako tražena riječ ne postoji, ispisati poruku da ta riječ ne postoji u rječniku.
   Učitavanje raditi tako dugo dok se ne unese znak 'x'. Potrebno je obratiti pažnju na mala/velika slova. Prijedlog je pretvarati sve u mala slova.
6. Napišite funkciju koja prima listu. Funkcija vraća boolean vrijednost istina ako su svi elementi liste parni brojevi, a boolean vrijednost laž ako postoji barem jedan broj koji nije paran.
7. Napišite program koji učitava elemente liste s tipkovnice. Učitavanje prekinuti onoga trenutka kada korisnik unese broj 5. Nakon što je lista učitana, program mora zamijeniti prvi element s posljednjim, drugi element s pretposljednjim i tako redom.
8. * Napravite rječnik proizvoljnog sadržaja sljedećeg oblika:

```
povrce = {
    'krumpir' : ['bijeli', 'crveni', 'za salatu'],
    'luk' : ['bijeli', 'crveni']
}
```

   Za svaki tip povrća ispišite broj pripadajućih vrsti. Sadržaj ispisa (temeljen na gornjem primjeru) neka bude:

Krumpir : 3
Luk : 2

Primijetite da riječi Krumpir i Luk počinju velikim početnim slovom, dok su u rječniku napisane kompletno malim slovima.

9. ** Izradite listu koja se sastoji od trojki brojeva od 1 do 40 koje zadovoljavaju izraz: $x^2 + y^2 = z$.

9.6. Pitanja za ponavljanje: Rad sa strukturama podataka

1. Koja je razlika između n-terca i liste?
2. Mogu li u skupu postojati dva podataka identične vrijednosti?
3. Od čega se sastoji svaki par elemenata u rječniku?
4. Je li moguće u rječnik naknadno dodavati parove vrijednosti?
5. Kako se zove ključna riječ za brisanje n-terca?

Dodatak: Završna vježba

1. S tipkovnice učitavajte cijele brojeve. Prvi upisani broj može biti bilo koji cijeli broj. Učitavanje ponavljati dok god je upisani broj strogo veći od prethodno upisanog broja. Ispisati sumu svih učitanih brojeva osim broja zbog kojeg je prekinuto učitavanje.
2. Učitajte s tipkovnice 2 niza znakova, svaki od tih nizova znakova spremite u zasebnu varijablu. Ispišite indekse na kojima se pojavljuju ista slova neovisno o veličini ('a' i 'A' tretirati jednako).
3. Napišite program koji s tipkovnice učitava proizvoljni cijeli troznamenkasti broj. Ako učitani broj nije troznamenkasti, ispišite poruku o greški i prekinite daljnje izvođenje programa. U slučaju da je učitani broj ispravan, ispišite prvi sljedeći troznamenkasti palindrom. Na primjer, ako je učitani broj 120, prvi sljedeći palindrom je 121.
4. * Napišite program koji učitava cijele brojeve sve dok je unesena vrijednost veća od 0. Pronađite koji od učitanih brojeva ima najmanju sumu znamenki te ispišite taj broj i sumu.
5. * U glavnom programu učitajte proizvoljni cijeli broj s tipkovnice. Implementirajte funkciju `izracun()` koja će učitanu vrijednost dohvatiti preko globalne varijable, a rezultat se vraća pomoću `return` naredbe. Kao rezultat potrebno je vratiti zbroj svih brojeva od 0 do unesenog broja. Na primjer, ako je unesena vrijednost -5, tada funkcija mora vratiti zbroj brojeva: - 5 - 4 - 3 - 2 - 1 = -15, ako pak je unesena vrijednost 4, tada funkcija mora vratiti zbroj brojeva: 4 + 3 + 2 + 1 = 10. Rezultat ispišite na zaslon.
6. * Kreirajte 4 funkcije koje implementiraju operacije: zbrajanja, oduzimanja, množenja i dijeljenja dvaju brojeva koji se u funkciju prenose kao parametri. Najprije od korisnika tražite da odluči želi li raditi zbrajanje, oduzimanje, množenje, dijeljenje ili prekidanje izvršavanja programa, a nakon toga omogućite unos dviju vrijednosti nad kojima će se napraviti željena operacija. Ovaj postupak ponavljati tako dugo dok se ne unese vrijednost za prekidanje programa. U nastavku slijedi primjer:

   Odaberite računsku operaciju:
   1 – zbrajanje
   2 – oduzimanje
   3 – množenje
   4 – dijeljenje
   0 – izlaz iz programa
   Unesite broj željene operacije: ***1*** *(ovo unosi korisnik preko tipkovnice)*
   Unesite prvu vrijednost: ***5*** *(ovo unosi korisnik preko tipkovnice)*
   Unesite drugu vrijednost: ***10*** *(ovo unosi korisnik preko tipkovnice)*
   Rezultat: 15
   {Nakon ispisa rezultata ovu sekvencu ponavljajte tako dugo dok korisnik ne unese 0}
7. * S tipkovnice učitajte proizvoljni niz znakova. Kreirajte novi niz znakova koji će sadržavati naizmjence velika i mala slova iz ulaznog niza, redom kako se pojavljuju u ulaznom nizu: prvo veliko slovo u ulaznom nizu, prvo sljedeće malo slovo u nastavku ulaznog niza,

prvo sljedeće veliko slovo u nastavku ulaznog niza itd. Novokreirani niz ispišite na zaslon. U nastavku se nalazi primjer:

Ulazni niz: ife**Fe**m**F**E**k**ej83**FkW**

Izlazni niz: FeFkFkW

8. * Napišite program koji s tipkovnice učitava cijeli broj `n` iz intervala [3, 20]. U slučaju da je unesena vrijednost neispravna, ispisati prikladnu poruku na ekran te zatražiti ponovni unos cijelog broja. Nakon učitane vrijednosti `n`, učitajte `n` parova cijelih brojeva. Nakon što je `n` parova brojeva učitano, ispišite parove brojeva koji imaju najveću sumu.
9. ** Napišite program koji s tipkovnice učitava proizvoljni niz znakova. Nad učitanim nizom znakova napravite analizu je li taj niz palindrom ili nije. Niz je palindrom ako se isto čita slijeva nadesno ili pak zdesna nalijevo. Na primjer, niz: "Ana voli Milovana." je simetričan niz.
10. ** S tipkovnice učitajte pozitivne realne brojeve `a`, `b` i cijeli broj `n`. Brojevi `a` i `b` predstavljaju početne članove nizova A i B (a_1 odnosno b_1), dok broj n predstavlja broj koraka izračunavanja. Član niza a_i izračunava se kao aritmetička sredina prethodnog člana niza A i prethodnog člana niza B, tj. $a_i = \frac{a_{i-1}+b_{i-1}}{2}$. Član niza b_i izračunava se kao geometrijska sredina prethodnog člana niza A i prethodnog člana niza B, tj. $\sqrt{a_{i-1} + b_{i-1}}$. Članove nizova A i B ispišite u skladu s oblikom ispisa prikazanog u nastavku. Prilikom ispisa, vrijednosti elemenata niza a_i i b_i zaokružite na dvije decimale.

    Unesite vrijednost a1: {vrijednost}

    Unesite vrijednost b1: {vrijednost}

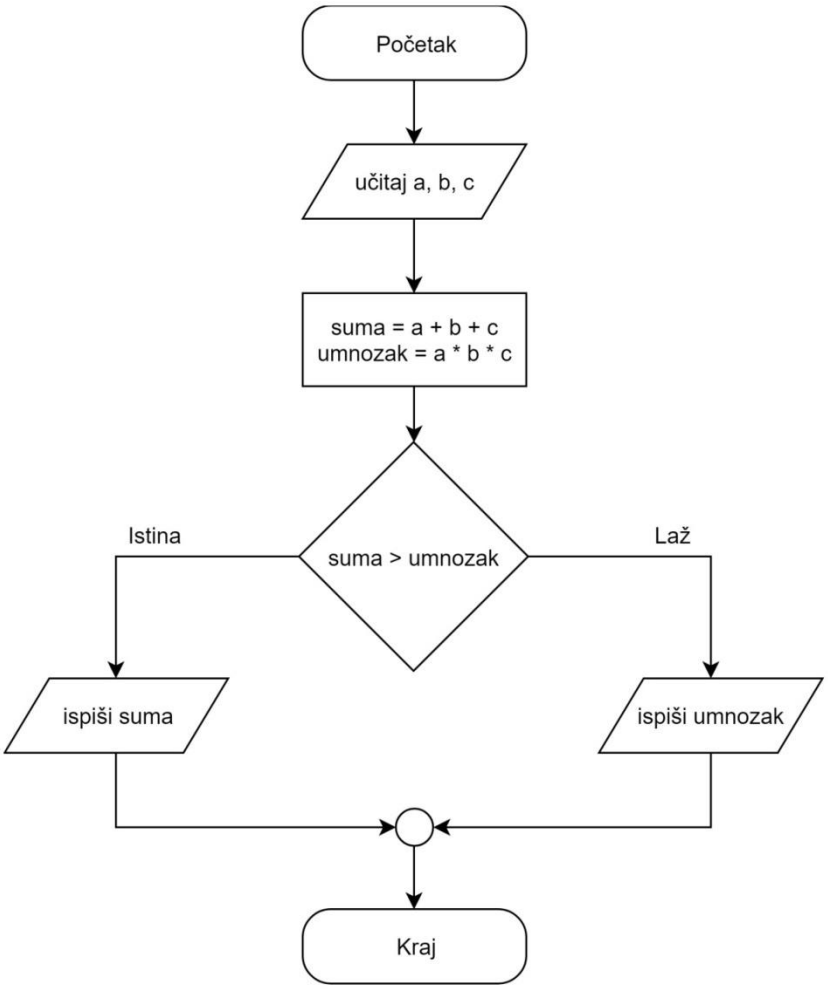
    Unesite vrijednost n: {vrijednost}

    A(1) = {vrijednost}, B(1) = {vrijednost}

    Napomena: prethodnu liniju potrebnu je ponoviti n puta, a kao vrijednosti ispišite vrijednosti dobivene prema zadanim formulama.

Dodatak: Rješenja vježbi

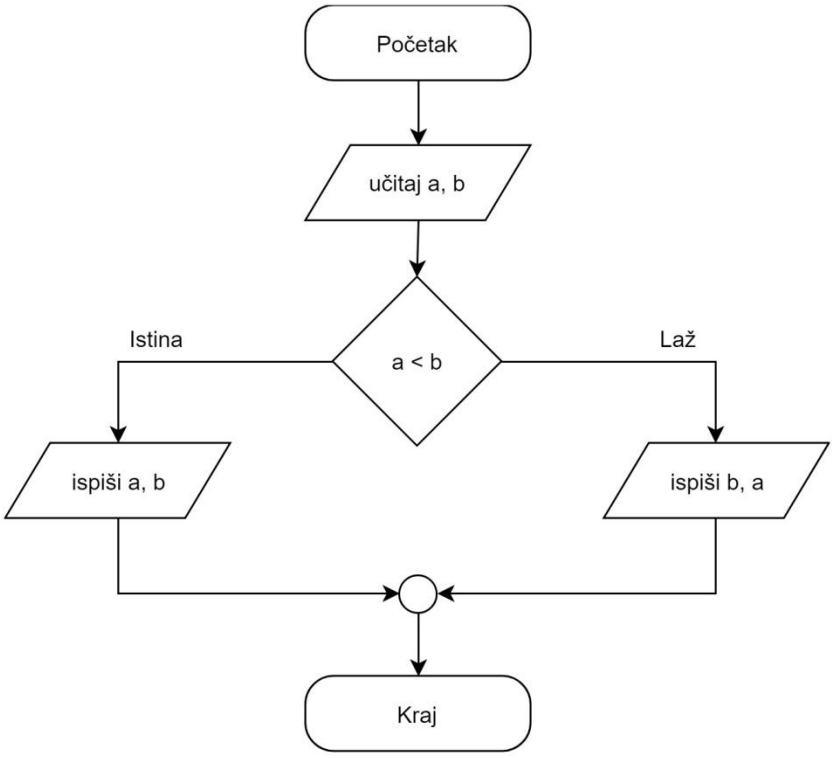
1.4.1.



1.4.2.

```
početak

učitaj(a, b, c)

izračunaj:
    suma := a + b + c
    umnozak := a * b * c

ako je suma > umnozak tada
    ispiši(suma)
inače
    ispiši(umnozak)

kraj
```

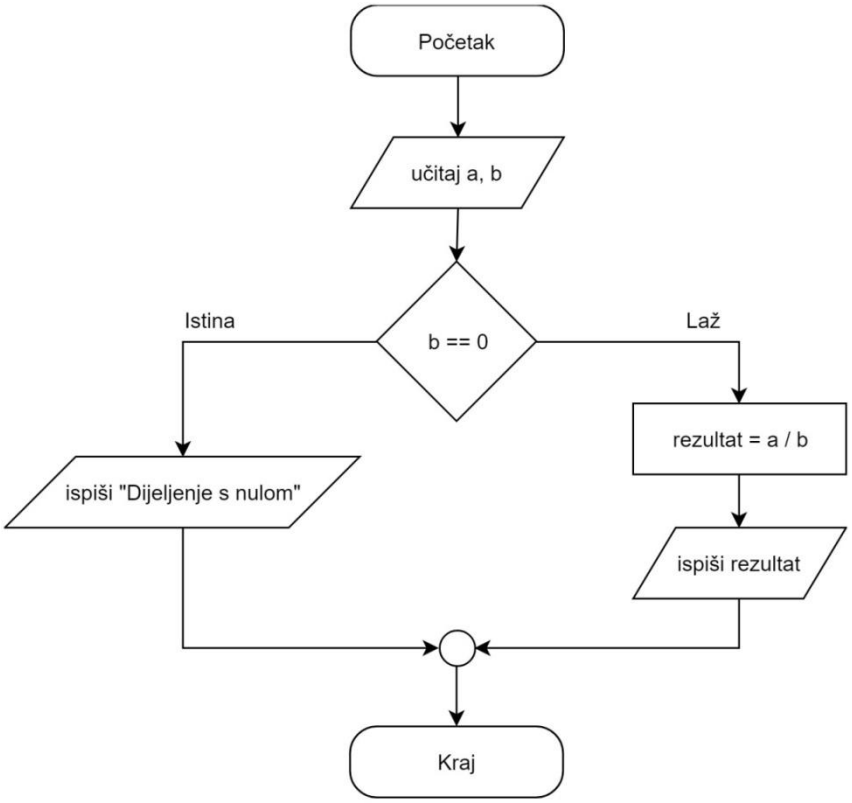
1.4.3.



1.4.4.

```
početak

učitaj(a, b)

ako je a < b tada
    ispiši(a, b)
inače
    ispiši(b, a)

kraj
```

1.4.5.



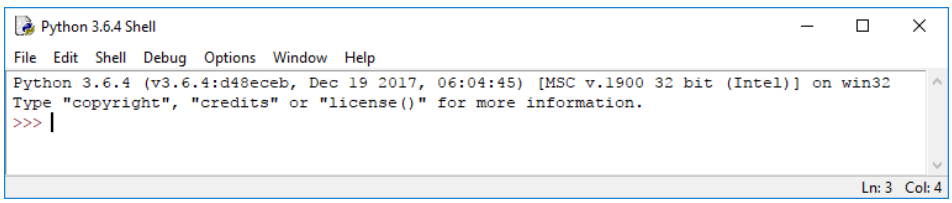
1.4.6.

```
početak

učitaj(a, b)

ako je b == 0 tada
    ispiši(Dijeljenje s nulom)
inače
    rezultat := a / b
    ispiši(rezultat)

kraj
```

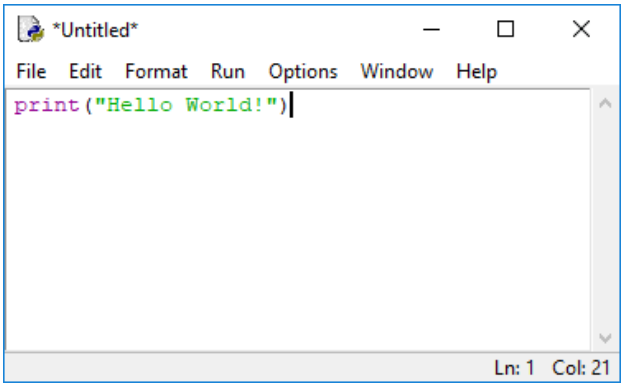
1.4.7.

Otvoriti *Python IDLE*:

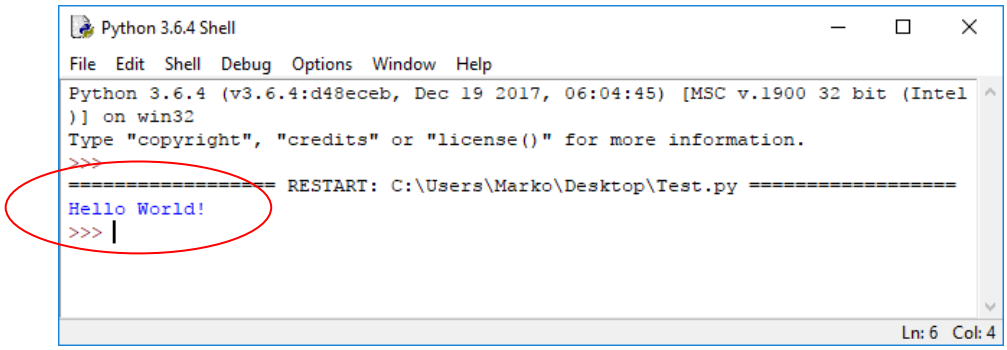
Start → Python 3.7 → IDLE (Python 3.7 32-bit)



Nakon toga kliknuti na **File** pa na **New File**. Prozor koji se otvori je editor unutar kojega je moguće unositi programski kôd:



Nakon što je programski kôd napisan, potrebno ga je spremiti. Programski kôd se sprema klikom na **File** pa na **Save**. Nakon što je kôd spremljen u datoteku, napisani kôd je moguće pokrenuti, to se radi tako da se u izborniku odabere **Run** pa **Run Module** . Ako je sve točno napisano, otvorit će se *Python Shell* i unutar njega će se ispisati niz znakova koji je napisan u funkciji `print()`.



1.5.1.

Za upis ulaznih / ispis izlaznih vrijednosti koristi se matematički oblik paralelogram.

1.5.2.

Glavna prednost pseudo-kôda pred dijagramom toka jest čitljivost prilikom opisivanja kompleksnijih algoritama.

2.9.1.

```
a = 10
b = 5

zbroj = a+b
razlika = a-b
umnozak = a*b
kolicnik = a/b
ostatak = a%b
potenciranje = a**b
cjelobrojnoDijeljenje = a//b

print(zbroj)
print(razlika)
print(umnozak)
print(kolicnik)
print(ostatak)
print(potenciranje)
print(cjelobrojnoDijeljenje)
```

```
Izlaz:
      15
      5
      50
      2.0
      0
      100000
      2
```

2.9.2.

```
a = 10
b = 5

a += b
print(a)

a = 10
a -= b
print(a)

a = 10
a *= b
print(a)

a = 10
a /= b
print(a)

a = 10
a %= b
print(a)

a = 10
a **= b
print(a)
```

```
a = 10
a //= b
print(a)
```

```
Izlaz:
      15
      5
      50
      2.0
      0
      100000
      2
```

2.9.3.

```
a = 51
b = 12

tmp = a
a = b
b = tmp

a = a%10
b %= 10

print(a, b)
```

```
Izlaz:
      2 1
```

2.9.4.

```
a = 55
b = 11

print(a < b)
print(a > b)
print(a <= b)
print(a >= b)
print(a == b)
print(a != b)
```

```
Izlaz:
      False
      True
      False
      True
      False
      True
```

2.9.5.

```
a = True
b = False
print(a and b)
print(a or b)
print(not a)
```

```
Izlaz:
      False
      True
      False
```

2.9.6.

```
'''
Ovo je
rješenje
zadatka
'''

a = True # Inicijalizacija varijable a
b = False # Inicijalizacija varijable b

print(a and b)
print(a or b)
print(not a)
```

```
Izlaz:
      False
      True
      False
```

2.9.7.

```
a = 5
b = 10
c = 15
d = 21

arSred = (a + b + c + d) / 4

print(arSred)
```

```
Izlaz:
      12.75
```

2.9.8.

```
a = 5
b = 10
c = 15
d = 21

arSred = (a + b + c + d) / 4

arSred = int(arSred)

kvadrat = arSred ** 2

print(kvadrat)
```

```
Izlaz:
      144
```

2.9.9.

```
a = 5
b = 10
c = 15
d = 21

arSred = (a + b + c + d) / 4

arSred = int(arSred)

kvadrat = arSred ** 2

kvadrat *= 100

print(kvadrat)
```

```
Izlaz:
      14400
```

2.9.10.

```
a = 5
b = 10
c = 15
d = 21

arSred = (a+b+c+d) / 4

arSred = int(arSred)

kvadrat = arSred ** 2

kvadrat *= 100

print(kvadrat < 500)
```

```
Izlaz:
      False
```

2.9.11.

```
niz = "I'm from Croatia."

print(niz)
```

```
Izlaz:
      I'm from Croatia.
```

2.9.12.

```
niz = "I'm from Croatia."

print(niz[4:8])
```

```
Izlaz:
      from
```

2.9.13.

```
niz1 = "Prvi"
niz2 = "Drugi"

print(niz1 + niz2)
print(niz1 * 3)
print(niz1[2])
print(niz1[2:4])
print(niz1[:3])
print('X' in niz1)
print('X' not in niz1)
```

```
Izlaz:
      PrviDrugi
      PrviPrviPrvi
      v
      vi
      Prv
      False
      True
```

2.9.14.

```
a = 10
b = 20
c = 30.5
print(int(c))
print(float(b))
print(complex(a, b))
print(hex(a))
print(oct(a))
```

```
Izlaz:
      30
      20.0
      (10+20j)
      0xa
      0o12
```

2.9.15.

```
niz = "Hello WORLD!"

print(niz.capitalize())
print(len(niz))
print(niz.lower())
print(niz.upper())
print(niz.title())

niz = "   Hello WORLD!   "
print(niz.lstrip(),".")
print(niz.rstrip(),".")
print(niz.strip(),".")
```

```
Izlaz:
      Hello world!
      12
```

```
      hello world!
      HELLO WORLD!
      Hello World!
      Hello WORLD!    .
         Hello WORLD! .
      Hello WORLD! .
```

2.9.16.

```
niz = "I'm from Croatia."

duljina = len(niz)

# int() se koristi jer duljina niza može biti
neparan broj.
print(niz[0:int(duljina/2)])
```

```
Izlaz:
      I'm from
```

2.9.17.

```
stupnjevi = 24

farenhajt = (stupnjevi * 9 / 4) + 32

print(farenhajt)
```

```
Izlaz:
      86.0
```

2.10.1.

Preciznost zapisa decimalnih brojeva razlikuje se od računala do računala.

2.10.2.

Skraćeni oblik aritmetičkog operatora ako želimo uvećati vrijednost neke varijable jest += .

2.10.3.

U *Pythonu* prilikom navođenja varijable NIJE potrebno eksplicitno navesti kojega je tipa neka varijabla.

3.5.1.

```
r = 5.5

if r > 0:
    volumen = 4 / 3 * r**3 * 3.1415
    print("Radijus iznosi: ", r)
    print("Volumen iznosi: ", volumen)
else:
    print("Radijus je neispravan.")
```

```
Izlaz:
     Radijus iznosi:  5.5
     Volumen iznosi:  696.8894166666666
```

3.5.2.

```
a = 500
b = 5

if a > 100 and b < 100 or a < 100 and b > 100:
    print("Jedna je veća, a druga je manja od
100.")
elif a > 100 and b > 100:
    print("Obje vrijednosti su veće od 100.")
elif a < 100 and b < 100:
    print("Obje vrijednosti su manje od 100.")
else:
    print("Obje vrijednosti su jednake.")
```

```
Izlaz:
     Jedna vrijednost je veća, a druga vrijednost
     je manja od 100.
```

3.5.3.

```
a = 200
b = 20

if a > (b + 50) and b % 2 == 0:
    print("Uvjeti su zadovoljeni.")
else:
    print("Uvjeti nisu zadovoljeni.")
```

```
Izlaz:
     Uvjeti su zadovoljeni.
```

3.5.4.

```
a = 200
b = 20

if a >= 5 and a <= 20 or b >= 5 and b <= 20:
    print("Zadovoljava.")
else:
    print("Ne zadovoljava.")
```

```
Izlaz:
     Zadovoljava.
```

3.5.5.

```
a = 55
b = 21
c = 55
d = 621
e = 123
brojac = 0

if a > 100:
    brojac += 1
if b > 100:
    brojac += 1
if c > 100:
    brojac += 1
if d > 100:
    brojac += 1
if e > 100:
    brojac += 1

if brojac >= 3:
    print("Zadovoljava.")
else:
    print("Ne zadovoljava.")
```

```
Izlaz:
     Ne zadovoljava.
```

3.5.6.

```
a1 = 5
a2 = 10

b1 = 6
b2 = 9

if a1 <= b1 and a2 >= b2:
    print("Zadovoljava.")
else:
    print("Ne zadovoljava.")
```

```
Izlaz:
     Zadovoljava.
```

3.5.7.

```
for n in range(1, 1001):
    if n%2 == 0 and n%5 == 0 and n%13 == 0:
        print(n)
```

```
Izlaz:
     130
     260
     390
     520
     650
```

```
780
910
```

3.5.8.

```
niz = "Ovo Je Neki Niz Anakova"

brojac = 0
pronadenoA = 0

for x in niz:
    if x >= "A" and x <= 'Z':
        brojac += 1

    if x == "A":
        pronadenoA = 1
        print("Veliko slovo A je pronađeno.")

if pronadenoA == 0:
    print(brojac)
```

```
Izlaz:
 Veliko slovo A je pronađeno.
```

3.5.9.

```
niz = "ABCDEFGHIJK"
duljina = len(niz)
n = 2

if duljina > n:
    indeks = 0
    for x in niz:
        if indeks%n == 0:
            print(x, end="")
        indeks += 1
else:
    print("Greska!")
```

```
Izlaz:
 ACEGIK
```

3.5.10.

```
brojac = 0
a = 2
b = 10

for n in range(a, b+1):
    jePrim = True
    for x in range(2, n):
        if n % x == 0:
            jePrim = False
            break
    if jePrim == True:
        brojac += 1

print(brojac)
```

```
Izlaz:
      4
```

3.5.11.

```
n = 7

for i in range(0,n): # Određuje redak
    for j in range(0,n): # Određuje stupac
        if i == j:
            print("1", end='')
        else:
            print("0", end='')
    print()
```

```
Izlaz:
      1000000
      0100000
      0010000
      0001000
      0000100
      0000010
      0000001
```

3.5.12.

```
n=5
x=1
y=1

for i in range(0,n): # Određuje redak
    for j in range(0,n): # Određuje stupac
        if i == y and j == x:
            print("X", end='')
        else:
            print("-", end='')
    print()
```

```
Izlaz:
      -----
      -X---
      -----
      -----
      -----
```

3.5.13.

```
n = 159
suma = 0

while n > 0:
    suma += n%10
    n //= 10

print(suma)
```

```
Izlaz:
      15
```

3.6.1.

`For` petlja iterira po danom nizu, dok se `while` petlja okreće tako dugo dok je dani uvjet zadovoljen.

3.6.2.

Naredba `break` prekida izvršavanje prve najbliže petlje, dok naredba `continue` preskače sav programski kôd koji se nalazi napisan ispod `continue` naredbe u pripadajućoj petlji te se izvršavanje programa vraća na početak.

4.6.1.

```
def ispis():
    print("Hello World!")

ispis()
```

```
Izlaz:
      Hello World!
```

4.6.2.

```
def izracun(a, b, c, d):
    print( ( (a*a) + (b*c) - d ) / 2)

izracun(1,2,3,4)
```

```
Izlaz:
      1.5
```

4.6.3.

```
def mnozenje(a, b=10):
    print(a*b)

izracun(10, 5)
izracun(10)
```

```
Izlaz:
      50
      100
```

4.6.4.

```
def izracun(a, b):
    return (a*a) + (b*b)

rezultat = izracun(10, 20)
print(rezultat)
```

```
Izlaz:
      500
```

4.6.5.

```
def zbroj():
    return a + b

a = 10
b = 20

rezultat = zbroj()

print(rezultat)
```

```
Izlaz:
      30
```

4.6.6.

```
def zbroj(a, b):
    return a+b

rezultat = zbroj(10, 20)

print(rezultat)
```

```
Izlaz:
     30
```

4.6.7.

```
n = 5
m = 10

def fakt(x):
    umnozak = 1

    while x > 1:
        umnozak *= x
        x -= 1

    return umnozak

if 0 <= n and n <= m:
    print(fakt(m) / (fakt(n) * fakt(m-n)))
else:
    print("Greška.")
```

```
Izlaz:
     252.0
```

4.7.1.

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije.

4.7.2.

Zaglavlje funkcije sastoji se od 3 elementa, a to su: ključna riječ `def`, ime funkcije i parametri funkcije.

4.7.3.

Nije potrebno navesti svih 5 parametara. U parametrima funkcije neki od parametara može se postaviti na predefiniranu vrijednost na način: `def funkcija(a, b, c, d, e=10)`. U ovom slučaju možemo pozvati funkciju na dva načina:

- `funkcija (5, 20, 30, 40)`
- `funkcija (5, 20, 30, 40, 100)`

4.7.4.

Naredba pomoću koje se iz funkcije vraća vrijednost u pozivajući dio programa zove se `return`.

4.7.5.

Tip varijable koja se nakon završetka funkcije "uništava" zove se lokalna varijabla.

4.7.6.

Globalne varijable iz glavnog dijela programa vidljive su u funkcijama.

5.3.1.

```
a = 1
b = 2
c = 3
d = 4
e = 5

print(a, b, c, d, e)
```

```
Izlaz:
      1 2 3 4 5
```

5.3.2.

```
a = 1
b = 2
c = 3
d = 4
e = 5

print(a, b, c, d, e, sep="\n")
```

```
Izlaz:
      1
      2
      3
      4
      5
```

5.3.3.

```
print("Hello World", end="!")
```

```
Izlaz:
      Hello World!
```

5.3.4.

```
var1 = input("Unesite prvi niz znakova: ")

var2 = input("Unesite drugi niz znakova: ")

print(var1, var2)
```

```
Izlaz:
      Unesite prvi niz znakova: AAA
      Unesite drugi niz znakova: BBB
      AAA BBB
```

5.3.5.

```
dan = input("Dan: ")
mjesec = input("Mjesec: ")
godina = input("Godina: ")

dan = int(dan)
mjesec = int(mjesec)
godina = int(godina)

if dan < 1 or dan > 31 or mjesec < 1 or mjesec > 12
or godina < 0 or godina > 2020:
    print("Greška")

else:
    print(dan, ". ", sep="", end="")
    if mjesec == 1:
       print("siječnja, ", end="")
    elif mjesec == 2:
       print("veljače, ", end="")
    elif mjesec == 3:
       print("ožujka, ", end="")
    elif mjesec == 4:
       print("travnja, ", end="")
    elif mjesec == 5:
       print("svibnja, ", end="")
    elif mjesec == 6:
       print("lipnja, ", end="")
    elif mjesec == 7:
       print("srpnja, ", end="")
    elif mjesec == 8:
       print("kolovoza, ", end="")
    elif mjesec == 9:
       print("rujna, ", end="")
    elif mjesec == 10:
       print("listopada, ", end="")
    elif mjesec == 11:
       print("studenog, ", end="")
    elif mjesec == 12:
       print("prosinca, ", end="")
    print(godina, ".", sep="")
```

```
Izlaz:
     Dan: 7
     Mjesec: 12
     Godina: 2018
     7. prosinca, 2018.
```

5.3.6.

```
while True:
    bodovi = input("Unesite broj bodova: ")
    bodovi = float(bodovi)

    if bodovi >= 0 and bodovi < 50:
        print("Nedovoljan!")
    elif bodovi >= 50 and bodovi < 62.5:
        print("Dovoljan!")
    elif bodovi >= 62.5 and bodovi < 75:
        print("Dobar!")
    elif bodovi >= 75 and bodovi < 87.5:
        print("Vrlo dobar!")
    elif bodovi >= 87.5 and bodovi <= 100:
        print("Odličan!")
    else:
        print("Uneseni broj bodova je neispravan!")
        break
```

```
Izlaz:
     Unesite broj bodova: 50
     Dovoljan!
     Unesite broj bodova: 60
     Dovoljan!
     Unesite broj bodova: 70
     Dobar!
     Unesite broj bodova: 80
     Vrlo dobar!
     Unesite broj bodova: 90
     Odličan!
     Unesite broj bodova: 100
     Odličan!
     Unesite broj bodova: 101
     Uneseni broj bodova je neispravan!
```

5.3.7.

```
while True:
    n = input("Unesite n: ")
    n = int(n)
    if n >= 3 and n <= 10:
        break

brojac = 1
for i in range(0, n): # Određuje redak
    for j in range(0, i): # Određuje prazan stupac
        print("   ", end="")
    for j in range(0, n-i): # Ispisuje broj
        if brojac >= 0 and brojac <= 9:
            print("  ", end="")
            print(brojac, end="")
        else:
            print(" ", end="")
            print(brojac, end="")
        brojac += 1
    print()
```

```
Izlaz:
     Unesite n: 10
       1  2  3  4  5  6  7  8  9 10
         11 12 13 14 15 16 17 18 19
            20 21 22 23 24 25 26 27
               28 29 30 31 32 33 34
                  35 36 37 38 39 40
                     41 42 43 44 45
                        46 47 48 49
                           50 51 52
                              53 54
                                 55
```

5.4.1.

Argumente `sep` i `end` nije potrebno prenositi prilikom pozivanja funkcije `print()`. Oni se prenose samo ako su potrebni.

5.4.2.

Ako se u pozivu funkcije ne prenese argument `sep`, on će poprimiti predefiniranu vrijednost *razmak*: `' '`.

5.4.3.

Automatski će se ispisati predefinirana vrijednost parametra `end`.

5.4.4.

Jedini mogući argument funkcije `input()` služi kao argument koji se ispisuje neposredno prije unosa teksta, a obično je to opis što se sljedeće unosi.

5.4.5.

Slanje teksta funkciji `input()` možemo izbjeći tako da prije poziva funkcije `input()` tekst ispišemo funkcijom `print()`.

6.2.1.

```
import math

rezultat = math.sin(2) + math.cos(1)

print(rezultat)
```

```
Izlaz:
      1.4495997326938215
```

6.2.2.

```
import math

broj = input("Unesite broj za korjenovanje: ")
rez = math.sqrt(int(broj))

print(rez)
```

```
Izlaz:
      Unesite broj za korjenovanje: 16
      4.0
```

6.2.3.

```
import math

print(math.pi)
```

```
Izlaz:
      3.141592653589793
```

6.2.4.

```
import math

rezultat = math.sin(2*math.pi) + math.cos(math.pi)

print(rezultat)
```

```
Izlaz:
      -1.0000000000000002
```

6.2.5.

```
from math import pow

a = int(input("Unesite a: "))
b = int(input("Unesite b: "))

rezultat = pow(a,2) + 2*a*b + pow(b,2)

print(rezultat)
```

```
Izlaz:
      Unesite a: 2
      Unesite b: 3
      25.0
```

6.2.6.

```
import math

broj = input("Unesite broj: ")
broj = float(broj)

f = math.floor(broj)
c = math.ceil(broj)

print(f, c, sep=", ")
```

```
Izlaz:
      Unesite broj: 5.6
      5, 6
```

6.3.1.

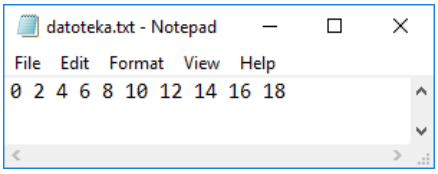
U paketima se uz funkcije nalaze i konstante poput: `pi`, `e`, `tau`, `inf`, `nan`.

6.3.2.

Da, potrebno je prilikom poziva funkcije napisati i ime paketa u kojem se pozivajuća funkcija nalazi, na primjer, `math.sqrt()`.

7.2.1.

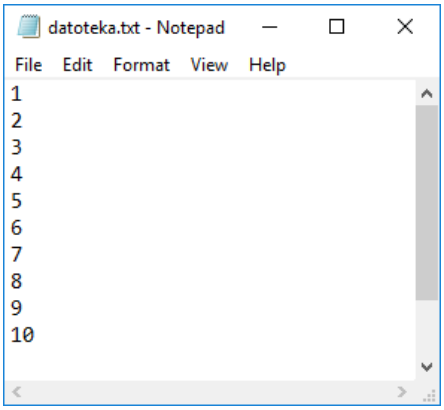
```
with open("datoteka.txt", "w") as f:
    for i in range(0, 20):
        if i%2 == 0:
            f.write(str(i) + " ")
```



7.2.2.

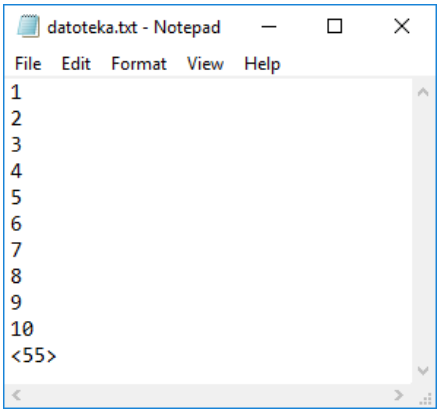
```
with open("datoteka.txt", "w") as f:
    for i in range(1, 11):
        f.write(str(i) + "\n")

suma = 0
with open("datoteka.txt", "r") as f:
    for i in f.readlines():
        suma += int(i)

print(suma)
```

```
Izlaz:
     55
```



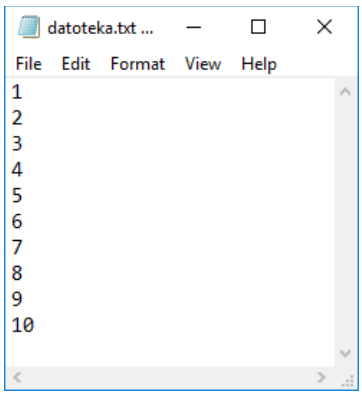
7.2.3.

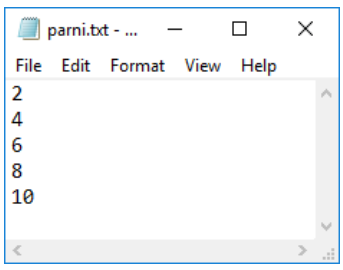
```
with open("datoteka.txt", "w") as f:
    for i in range(1, 11):
        f.write(str(i) + "\n")

suma = 0
with open("datoteka.txt", "r") as f:
    for i in f.readlines():
        suma += int(i)

with open("datoteka.txt", "a") as f:
    f.write("<" + str(suma) + ">")
```

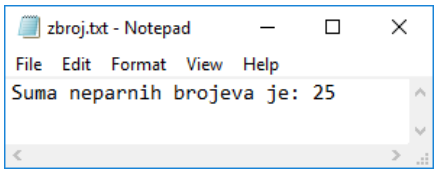


7.2.4.

```
ulaz = open("datoteka.txt", "r")
izlazParni = open("parni.txt", "w")
izlazZbroj = open("zbroj.txt", "w")

sumaBrojeva = 0
for line in ulaz.readlines():
    line = int(line)
    if line%2 == 0:
        izlazParni.write(str(line) + "\n")
    else:
        sumaBrojeva += line

izlazZbroj.write("Suma neparnih brojeva je: " +
str(sumaBrojeva))

ulaz.close()
izlazParni.close()
izlazZbroj.close()
```







7.3.1.

Mode koji služi samo za čitanje iz datoteke je `r`.

7.3.2.

Mode koji služi za prebrisavanje stare postojeće datoteke je `w`.

7.3.3.

Znak u načinu korištenja koji omogućava i čitanje iz datoteke, ali i pisanje u datoteku je `+`.

7.3.4.

Ako se želi obrađivati binarnu datoteku, potrebno je dodati u način korištenja znak `b`.

8.3.1.

```
a = input("Unesite prvi broj: ")
b = input("Unesite drugi broj: ")

rez = int(a) + int(b)
print(rez)
```

```
Izlaz:
      Unesite prvi broj: 3
      Unesite drugi broj: 3
      6
```

Ako program pokrenemo klikom na *.py datoteku, možemo unijeti dva broja, no naredbeni redak se automatski zatvara nakon što se program izvrši i ne vidimo rezultat.

Ako skriptu pokrenemo izravno iz naredbenog retka, možemo vidjeti i rezultat zbroja.

8.3.1.

```
with open("datoteka.txt", "a") as f:
    f.write("x")
```

9.1.10.1.

```
dani = ['ponedjeljak', 'utorak', 'srijeda',
'četvrtak', 'petak']

dani.append('subota')
dani.append('nedjelja')

print(dani)

#ispisuju se samo neradni dani, svaki u svom retku:
for element in dani[5:7]:
    print(element)
```

```
Izlaz:
     ['ponedjeljak', 'utorak', 'srijeda',
     'četvrtak', 'petak', 'subota', 'nedjelja']
     subota
     nedjelja
```

9.1.10.2.

```
dani = ['ponedjeljak', 'utorak', 'srijeda',
'četvrtak', 'petak', 'subota', 'nedjelja']

radniDani = dani.copy()
radniDani.pop()
radniDani.remove('subota')

print(radni_dani)
```

```
Izlaz:
     ['ponedjeljak', 'utorak', 'srijeda',
     'četvrtak', 'petak']
```

9.1.10.3.

```
dani = ['ponedjeljak', 'utorak', 'srijeda',
'četvrtak', 'petak', 'subota', 'nedjelja']
radniDani = ['ponedjeljak', 'utorak', 'srijeda',
'četvrtak', 'petak']
vikend = []

for element in dani:
    if element not in radniDani:
        vikend.append(element)
print(vikend)
```

```
Izlaz:
     ['subota', 'nedjelja']
```

9.1.10.4.

```
radniDani = ['ponedjeljak', 'utorak', 'srijeda',
'četvrtak', 'petak']
vikend = ['subota', 'nedjelja']

radniDani.extend(vikend)
```

```
print(radniDani)

print(vikend)
```

```
Izlaz:
      ['ponedjeljak', 'utorak', 'srijeda',
      'četvrtak', 'petak', 'subota', 'nedjelja']
```

9.1.17.1.

```
gradovi = ['Split', 'Zagreb', 'Osijek', 'Bjelovar',
'Koprivnica']
print(gradovi)

gradovi.reverse()
print(gradovi)

gradovi.sort()
print(gradovi)
```

```
Izlaz:
      ['Split', 'Zagreb', 'Osijek', 'Bjelovar',
      'Koprivnica']
      ['Koprivnica', 'Bjelovar', 'Osijek',
      'Zagreb', 'Split']
      ['Bjelovar', 'Koprivnica', 'Osijek', 'Split',
      'Zagreb']
```

9.1.17.2.

```
gradovi = ['Bjelovar', 'Koprivnica', 'Osijek',
'Split', 'Zagreb']
gradovi.insert(0, gradovi[4])
print(gradovi)

while 'Zagreb' in gradovi:
    gradovi.remove('Zagreb')

print(gradovi)
```

```
Izlaz:
      ['Zagreb', 'Bjelovar', 'Koprivnica',
      'Osijek', 'Split', 'Zagreb']
      ['Bjelovar', 'Koprivnica', 'Osijek', 'Split']
```

9.1.17.3.

```
gradovi = ['Bjelovar', 'Koprivnica', 'Osijek',
'Split']
for element in gradovi:
    print(element)
```

```
Izlaz:
      Bjelovar
      Koprivnica
      Osijek
      Split
```

9.2.5.1.

```
a = {1, 2, 3, 4, 5}
b = {4, 5, 6, 7, 8}
print(a | b)
print(a & b)
```

```
Izlaz:
      {1, 2, 3, 4, 5, 6, 7, 8}
      {4, 5}
```

9.3.6.1.

```
osoba = {'ime' : 'Alan', 'prezime' : 'Ford',
'godine' : 20}

print(osoba.values())

for element in osoba:
    print(osoba[element])
```

```
Izlaz:
      dict_values(['Alan', 'Ford', 20])
      Alan
      Ford
      20
```

9.3.6.2.

```
osoba = {'ime' : 'Alan', 'prezime' : 'Ford',
'godine' : 20}

osoba.update({'adresa' : '6. avenija, New York'})
print(osoba)
```

```
Izlaz:
      {'ime': 'Alan', 'prezime': 'Ford', 'godine':
      20, 'adresa': '6. avenija, New York'}
```

9.3.6.3.

```
osoba = {'ime' : 'Alan', 'prezime' : 'Ford',
'godine' : 20, 'adresa': '6. avenija, New York'}

del osoba['godine']
print(osoba)
```

```
Izlaz:
      {'ime': 'Alan', 'prezime': 'Ford', 'adresa':
      '6. avenija, New York'}
```

9.4.4.1.

```
samoglasnici = ('a', 'e', 'i', 'o' , 'u')
print(samoglasnici)
```

```
Izlaz:
      ('a', 'e', 'i', 'o', 'u')
```

9.4.4.2.

```
samoglasnici = ('a', 'e', 'i', 'o' , 'u')
samoglasnici[0] = 'x'
print(samoglasnici)
del samoglasnici[0]
print(samoglasnici)
```

```
Izlaz:
      Traceback (most recent call last):
        File "D:\Python\liste.py", line 87, in
      <module>
          samoglasnici[0] = 'x'
      TypeError: 'tuple' object does not support
      item assignment
```

9.5.1.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]

i=0

for element in lista:
    if i%2 == 0:
        print(element)
    i += 1
```

```
Izlaz:
      Ivan
      Nino
      Tin
```

9.5.2.

```
lista = []

brojac = 0
while True:
    broj = input("Unesite broj: ")
    broj = int(broj)

    if broj == 5:
        break
    else:
        brojac += 1
        lista.append(broj)

print(lista)
```

```
Izlaz:
      Unesite broj: 1
      Unesite broj: 2
      Unesite broj: 3
      Unesite broj: 4
      Unesite broj: 5
      [1, 2, 3, 4]
```

9.5.3.

```
lista = []

brojac = 0
while True:
    broj = input("Unesite broj: ")
    broj = int(broj)

    if broj == 5:
        break
    else:
        brojac += 1
        lista.append(broj)

if brojac < 6:
    while brojac < 6:
        lista.append(0)
        brojac += 1

print(lista)
```

```
Izlaz:
     Unesite broj: 1
     Unesite broj: 2
     Unesite broj: 3
     Unesite broj: 5
     [1, 2, 3, 0, 0, 0]
```

9.5.4.

```
lista = []

brojac = 0
while True:
    broj = input("Unesite broj: ")
    broj = int(broj)

    if broj == 5:
        break
    else:
        brojac += 1
        lista.append(broj)

if brojac < 6:
    while brojac < 6:
        lista.append(0)
        brojac += 1

i = 0
for element in lista:
    print("[", i, "] = ", element, sep='')
    i += 1
```

```
Izlaz:
     Unesite broj: 1
     Unesite broj: 2
     Unesite broj: 3
```

```
Unesite broj: 4
Unesite broj: 5
[0] = 1
[1] = 2
[2] = 3
[3] = 4
[4] = 0
[5] = 0
```

9.5.6.

```
rjecnik = {"stol" : "desk", "sat" : "clock",
"penkala" : "pencil", "remen" : "belt", "svjetlo" :
"light"}

while 1==1:
    rijec = input("Unesite rijec na hrvatskome: ")
    rijec = rijec.lower()
    if rijec == "x":
        break
    if rijec in rjecnik:
        print(rjecnik[rijec])
    else:
        print("Unesene rijeci nema u rjecniku!")
```

```
Izlaz:
 Unesite rijec na hrvatskome: stol
 desk
 Unesite rijec na hrvatskome: sat
 clock
 Unesite rijec na hrvatskome: nema
 Unesene rijeci nema u rjecniku!
 Unesite rijec na hrvatskome: remen
 belt
 Unesite rijec na hrvatskome: x
```

9.5.7.

```
def analiza(lista):
    for i in lista:
        if i%2 == 1:
            return False
    return True


lista = [2, 4, 6, 8, 9]

if analiza(lista) == True:
    print("Svi elementi su parni!")
else:
    print("Postoji barem jedan neparan broj!")
```

```
Izlaz:
 Postoji barem jedan neparan broj!")
```

9.5.8.

```
lista = []

while True:
    broj = input("Unesite broj: ")
    broj = int(broj)

    if broj == 5:
        break
    else:
        lista.append(broj)

print(lista)
lista.reverse()
print(lista)
```

```
Izlaz:
     Unesite broj: 10
     Unesite broj: 20
     Unesite broj: 30
     Unesite broj: 20
     Unesite broj: 30
     Unesite broj: 5
     [10, 20, 30, 20, 30]
     [30, 20, 30, 20, 10]
```

9.5.9.

```
povrce = {
     'krumpir' : ['bijeli', 'crveni', 'za salatu'],
     'luk' : ['bijeli', 'crveni']
}

for vrsta in povrce.keys():
    print(vrsta.title(), ":", len(povrce[vrsta]))
```

```
Izlaz:
     Krumpir : 3
     Luk : 2
```

9.5.10.

```
from math import pow

lista = []
for x in range(41):
    for y in range(41):
        for z in range(41):
            if (pow(x, 2) + pow(y, 2)) == z:
                tmp = [x, y, z]
                lista.append(tmp)

for i in lista:
    print(i)
```

```
Izlaz:
     [0, 0, 0]
     [0, 1, 1]
     [0, 2, 4]
```

```
      [0, 3, 9]
      [0, 4, 16]
      [0, 5, 25]
      [0, 6, 36]
      [1, 0, 1]
      [1, 1, 2]
      [1, 2, 5]
      [1, 3, 10]
      [1, 4, 17]
      [1, 5, 26]
      [1, 6, 37]
      [2, 0, 4]
      [2, 1, 5]
      [2, 2, 8]
      [2, 3, 13]
      [2, 4, 20]
      [2, 5, 29]
      [2, 6, 40]
      [3, 0, 9]
      [3, 1, 10]
      [3, 2, 13]
      [3, 3, 18]
      [3, 4, 25]
      [3, 5, 34]
      [4, 0, 16]
      [4, 1, 17]
      [4, 2, 20]
      [4, 3, 25]
      [4, 4, 32]
      [5, 0, 25]
      [5, 1, 26]
      [5, 2, 29]
      [5, 3, 34]
      [6, 0, 36]
      [6, 1, 37]
      [6, 2, 40]
```

9.6.1.

U načelu, n-terac se ponaša isto kao i lista, ali s jednom bitnom razlikom. Razlika je ta da je n-terac nepromjenjiv (engl. *immutable*) tip podatka, za razliku od liste koja je promjenjiva.

9.6.2.

Sve vrijednosti unutar jednoga skupa su jedinstvene.

9.6.3.

Svaki zapis možemo podijeliti na dva dijela: `ključ : vrijednost`.

9.6.4.

Da, u rječnik je moguće naknadno dodavati parove vrijednosti.

9.6.5.

N-terac se uništava pomoću ključne riječi `del`.

Dodatak: Završna vježba

1.

```
suma = 0

vrijednost = input("Unesite broj: ")
vrijednost = int(vrijednost)

suma += vrijednost

while True:
   prethodni = vrijednost

   vrijednost = input("Unesite broj: ")
   vrijednost = int(vrijednost)

   if vrijednost <= prethodni:
      break

   suma += vrijednost
   prethodni = vrijednost

print("Suma unesenih brojeva: ", suma)
```

```
Izlaz:
      Unesite broj: 1
      Unesite broj: 2
      Unesite broj: 3
      Unesite broj: 4
      Unesite broj: 5
      Unesite broj: 5
      Suma unesenih brojeva:  15
```

2.

```
niz1 = input("Unesite niz znakova: ")
niz2 = input("Unesite niz znakova: ")

duljina1 = len(niz1)
duljina2 = len(niz2)

if duljina1 < duljina2:
    duljina = duljina1
else:
    duljina = duljina2

i = 0
while i < duljina:
    if niz1[i].lower() == niz2[i].lower():
        print(i, niz1[i].lower())
    i += 1
```

```
Izlaz:
      Unesite niz znakova: Srce
      Unesite niz znakova: xRcx
      1 r
      2 c
```

3.

```
x = input("Unesite troznamenkasti broj: ")
x = int(x)

if x < 100 or x > 999:
   print("Uneseni broj nije troznamenkasti!")
else:
   while x<1000:
      prva = x // 100
      zadnja = x % 10

      if prva == zadnja:
         print(x)
         break
      x+=1
```

```
Izlaz:
      Unesite troznamenkasti broj: 119
      121
```

4.

```
def sumaZnamenki(x):
   suma = 0

   while x > 0:
      suma += x%10
      x /= 10
      x = int(x)

   return suma

najmanji = 0
najmanjaSuma = -1
while True:
   x = input("Unesite broj: ")
   x = int(x)

   if x <= 0:
      break

   s = sumaZnamenki(x)

   if najmanjaSuma == -1 or najmanjaSuma > s:
      najmanjaSuma = s
      najmanji = x

print("Broj: ", najmanji)
print("Najmanja suma znamenki: ",
int(najmanjaSuma))
```

```
Izlaz:
      Unesite broj: 159
      Unesite broj: 845
      Unesite broj: 144
      Unesite broj: 0
      Broj:  144
      Najmanja suma znamenki:  9
```

5.

```
def izracun():
    suma = 0
    tmp = x

    if tmp < 0:
        tmp *= -1

    while tmp > 0:
        suma += tmp
        tmp -= 1

    if x < 0:
        suma *= -1

    return suma


x = input("Unesite cijeli broj: ")
x = int(x)

suma = izracun()
print(suma)
```

```
Izlaz 1:
     Unesite cijeli broj: 10
     55
```

```
Izlaz 2:
     Unesite cijeli broj: -10
     -55
```

6.

```
def zbroj(a, b):
   return a+b

def razlika(a, b):
   return a-b

def umnozak(a, b):
   return a*b

def kolicnik(a, b):
   # Provjera nedozvoljenog dijeljenja
   if b == 0:
      return "Dijeljenje s nulom!"
   return a/b


info = ("Odaberite računsku operaciju:\n" +
         "1 - zbrajanje\n" +
         "2 - oduzimanje\n" +
         "3 - množenje\n" +
         "4 - dijeljenje\n" +
         "0 - izlaz iz programa")
```

```
while True:
   print(info)

   tip = input("Unesite broj željene operacije: ")
   tip = int(tip)

   if tip < 0 or tip > 4:
      print("Nepoznata operacija!\n")
      continue
   if tip == 0:
      print("Izlaz iz programa")
      break

   a = int(input("Unesite prvu vrijednost: "))
   b = int(input("Unesite drugu vrijednost: "))

   if tip == 1:
      rezultat = zbroj(a, b)
   elif tip == 2:
      rezultat = razlika(a, b)
   elif tip == 3:
      rezultat = umnozak(a, b)
   elif tip == 4:
      rezultat = kolicnik(a, b)

   print("Rezultat je:", rezultat, "\n")
```

```
Izlaz:
      Odaberite računsku operaciju:
      1 – zbrajanje
      2 – oduzimanje
      3 – množenje
      4 – dijeljenje
      0 – izlaz iz programa
      Unesite broj željene operacije: 1
      Unesite prvu vrijednost: 10
      Unesite drugu vrijednost: 20
      Rezultat je: 30

      Odaberite računsku operaciju:
      1 – zbrajanje
      2 – oduzimanje
      3 – množenje
      4 – dijeljenje
      0 – izlaz iz programa
      Unesite broj željene operacije: 0
      Izlaz iz programa
```

7.

```
niz = input("Unesite niz znakova: ")

izlazniNiz = ""

veliko = True

for e in niz:
   if e >= 'A' and e <= 'Z' and veliko == True:
      izlazniNiz += e
      veliko = False
   elif e >= 'a' and e <= 'z' and veliko == False:
      izlazniNiz += e
      veliko = True

print(izlazniNiz)
```

```
Izlaz 1:
      Unesite cijeli broj: 10
      55
```

```
Izlaz 2:
      Unesite niz znakova: ifeFemFEkej83FkW
      FeFkFkW
```

8.

```
while True:
   n = int(input("Unesite vrijednost: "))

   if n >= 3 and n <= 20:
      break
   else:
      print("Neispravna vrijednost!")

i = 0
lista = []
postavljeno = 0
najveci = 0

while i < n:
   print(i+1, ". par!", sep="")

   x = int(input("Unesite x: "))
   y = int(input("Unesite y: "))

   lista.append([x, y])

   if postavljeno == 0 or najveci < x+y:
      najveci = x+y
      postavljeno = 1

   i += 1

for e in lista:
   if e[0]+e[1] == najveci:
      print(e)
```

```
Izlaz:
      Unesite vrijednost: 3
      1. par!
      Unesite x: 1
      Unesite y: 2
      2. par!
      Unesite x: 3
      Unesite y: 4
      3. par!
      Unesite x: 5
      Unesite y: 2
      [3, 4]
      [5, 2]
```

9.

```
niz = input("Unesite niz znakova: ")

privremenNiz = ""

for e in niz:
   if e>='a' and e<='z' or e>='A' and e<='Z':
      privremenNiz += e.lower()

duljina = len(privremenNiz)

i=0
palindrom = True

while i < int(duljina/2):
   if privremenNiz[i] != privremenNiz[duljina-i-1]:
      palindrom = False
      break
   i += 1

if palindrom == True:
   print("Uneseni niz znakova je palindrom!")
else:
   print("Uneseni niz znakova nije palindrom!")
```

```
Izlaz 1:
      Unesite niz znakova: Ana voli Milovana
      Uneseni niz znakova je palindrom!
```

```
Izlaz 2:
      Unesite niz znakova: Ana voli Milovanaaaa
      Uneseni niz znakova nije palindrom!
```

10.

```
import math

a = int(input("Unesite vrijednost a1: "))
b = int(input("Unesite vrijednost b1: "))
n = int(input("Unesite vrijednost n: "))

print("A(1)=", round(a, 2), sep="", end=", ")
print("B(1)=", round(b, 2))

i = 2

while i <= n:
   aTmp = a
   bTmp = b

   a = (aTmp + bTmp) / 2
   b = math.sqrt(aTmp + bTmp)

   print("A(", i, ")=", round(a, 2),sep="",end=",
")
   print("B(", i, ")=", round(b, 2),sep="")

   i += 1
```

```
Izlaz:
      Unesite vrijednost a1: 5
      Unesite vrijednost b1: 10
      Unesite vrijednost n: 5
      A(1)=5, B(1)= 10
      A(2)=7.5, B(2)=3.87
      A(3)=5.69, B(3)=3.37
      A(4)=4.53, B(4)=3.01
      A(5)=3.77, B(5)=2.75
```

Literatura

1. A. B. Downey, *Think Python*, O'Reilly Media, 2012.
2. Aleksandar Stojanović, *Elementi računalnih programa*, Element, 2012.
3. David Beazley, Brian Jones, *Python Cookbook, 3rd Edition*, O'Reilly Media, 2013.
4. G. van Rossum, *Introduction to Python 3; Documentation for Python*, SoHoBooks, 2010.
5. John Paul Mueller, *Beginning Programming with Python For Dummies*, Wiley, 2014.
6. *Learn Python*, https://www.learnpython.org/, dohvaćeno 20.10.2018.
7. Leo Budin, Predrag Brođanac, Zlatka Markučič, Smiljana Perić, Dejan Škvorc, Magdalena Babić, *Računalno razmišljanje i programiranje u Pythonu*, Element, 2017.
8. Leo Budin, Predrag Brođanac, Zlatka Markučič, Smiljana Perić, *Napredno rješavanje problema programiranjem u Pythonu*, Element, 2013.
9. Leo Budin, Predrag Brođanac, Zlatka Markučič, Smiljana Perić, *Rješavanje problema programiranjem u Pythonu*, Element, 2012.
10. Mark Lutz, *Programming Python, 4th Edition*, O'Reilly Media, 2010.
11. Predrag Brođanac, Leo Budin, Zlatka Markučič, Smiljana Perić, *Izrada primjenskih programa u Pythonu*, Element, 2017.
12. *Programiz, https://www.programiz.com, dohvaćeno 20.10.2018.*
13. *Python 3.x Documentation*, https://docs.python.org/3/, dohvaćeno 20.10.2018.
14. *Python Notes for Professionals*, GoalKicker, https://goalkicker.com/PythonBook/
15. *Python Tutorials*, https://pythonspot.com/, dohvaćeno 20.10.2018.
16. *Real Python*, https://realpython.com/, dohvaćeno 20.10.2018.
17. Toma Rončević, *Uvod u programiranje*, 2016.
18. Zoran Kalafatić, Antonio Pošćić, Siniša Šegvić, Julijan Šribar, *Python za znatiželjne*, Element, 2016.

Bilješke: