Osnove programiranja (Python)

D450

OBRAZOVNI PROGRAMI srca

priručnik za polaznike

srce
Sveučilište u Zagrebu
Sveučilišni računski centar

Ovu verziju priručnika izradio je autorski tim Srca u sastavu:

Autor: Marko Hruška

Recenzentica: Vesna Zbodulja

Urednik: Dominik Kenđel

Lektorica: Ana Đorđević

Sveučilište u Zagrebu

Sveučilišni računski centar

Josipa Marohnića 5, 10 000 Zagreb

edu@srce.hr

Verzija priručnika D450-20230613

Sadržaj

Uvod

Trajanje poglavlja:

10 min

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

Preduvjet za pohađanje i razumijevanje ovoga tečaja jest poznavanje osnova rada na računalu i osnovnih matematičkih koncepata. Priručnik se sastoji od 10 poglavlja koja se odrađuju u četiri dana, po četiri školska sata dnevno. Uz teoretski dio gradiva, svako poglavlje sadrži 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. Naučeni osnovni koncepti 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 pretraživanja interneta i ostale literature. Nakon tečaja polaznici će posjedovati znanje za rješavanje lakših programskih zadataka u programskom jeziku Python.

Python podržava nekoliko različitih pristupa u pisanju programskog kôda, a to su imperativno, funkcijsko i objektno orijentirano programiranje [8].

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 (prevođenjem u izvršni oblik). Kako je Python interpreterski programski jezik, programi pisani u njemu sporije se izvršavaju za razliku od programa koji su pisani u programskim jezicima kao što su C, C++ itd.

U trenutku pisanja ovog priručnika aktivna verzija je Python 3. Također, postoji i Python 2, no ta verzija više nije aktivna, tj. nije podržana. Sintaksa Pythona 2 i Pythona 3 nije kompatibilna. Ovaj tečaj bazira se na verziji Python 3.

Napomena

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

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

Trajanje poglavlja:

35 min

Po završetku ovog poglavlja polaznik će moći:

- objasniti kako dijagram toka prikazuje algoritam
- nacrtati dijagram toka za jednostavan algoritam
- tumačiti pseudokôd
- napisati pseudokôd za jednostavan algoritam.

1.1. O programiranju

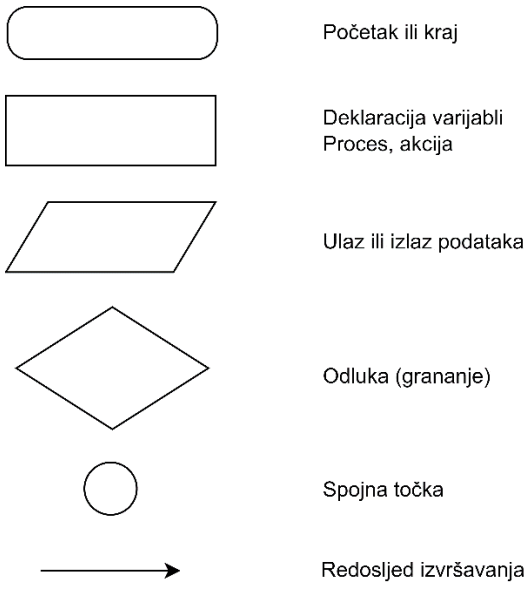
Programiranje je proces stvaranja računalnih programa koji izvršavaju željene funkcionalnosti. Postupak programiranja uključuje razumijevanje problema koji se pokušava riješiti, stvaranje algoritama koji opisuju korake za rješavanje problema i pisanje kôda koji implementira te korake. Programiranje također uključuje testiranje kôda kako bi se osiguralo da napisani programski kôd ispravno radi. Kao i kod svih vještina, programiranje zahtijeva vrijeme i trud kako bi se savladalo.

Algoritam je niz koraka kojima se opisuje kako riješiti neki problem. On predstavlja logički niz operacija koje se trebaju izvršiti kako bi se postigao određeni cilj. Algoritam može biti predstavljen dijagramom toka, pseudokôdom i programskim jezikom. U ovom poglavlju obradit ćemo dijagram toka i pseudokôd.

1.2. Dijagrami toka

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

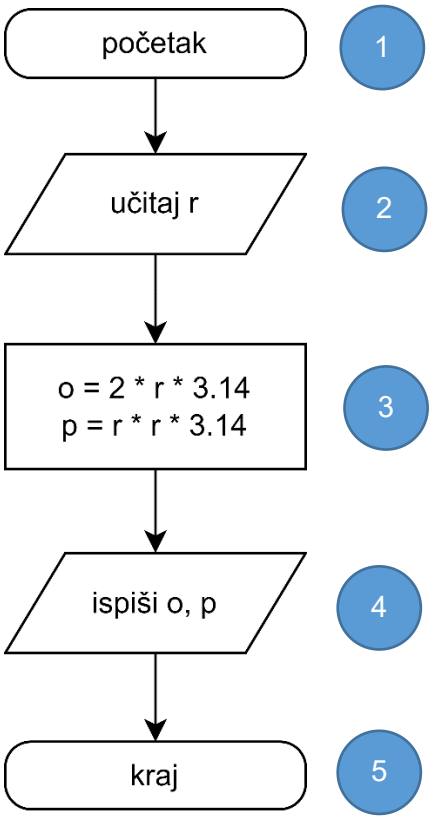
Osnovni elementi dijagrama toka:



Dijagrami toka pomažu programerima u vizualizaciji algoritma, ali i za bolje shvaćanje (vizualiziranje) problema koji rješavaju.

Varijable – ovaj pojam detaljnije je objašnjen u poglavlju *3. Osnove programskog jezika Python*. Varijable su osnovni elementi programa koji se koriste za spremanje vrijednosti (brojeva, nizova znakova itd.). 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.

Primjer 1: Program s tipkovnice neka učita jednu vrijednost koja predstavlja polumjer. Potrebno je izračunati opseg kruga (rezultat spremiti u varijablu imena `o`) te površinu kruga (rezultat spremiti u varijablu imena `p`). Tako dobiveni rezultat neka se ispiše.

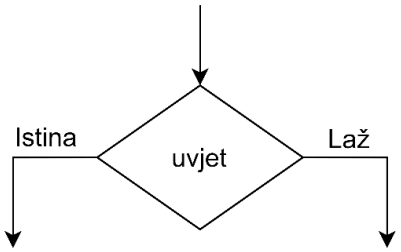


Obrazloženje rješenja po koracima*:*

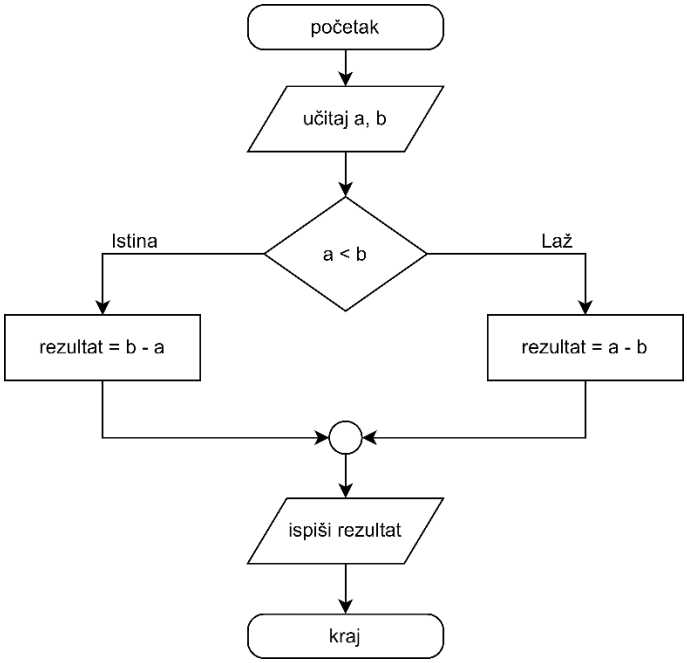
1. pravokutnik sa zaobljenim rubovima označava početak algoritma
2. paralelogram označava učitavanje polumjera s tipkovnice
3. unutar pravokutnika izračunava se opseg i površina kruga
4. paralelogram označava ispis izračunatog opsega i površine na ekranu
5. pravokutnik sa zaobljenim rubovima označava kraj algoritma.

Uvjetno gananje

Ako se želi ostvariti funkcionalnost da se ovisno o ulaznim podacima program odvija na različite načine, to je moguće napraviti pomoću uvjetnih grananja. Element koji se koristi za uvjetna grananja je romb. Unutar romba piše se izraz koji se provjera te se na temelju njega odlučuje hoće li se izvesti „lijeva grana“ ili „desna grana“ programa. U donjem primjeru „lijeva grana“ izvodi se u slučaju da je uvjet istinit, dok se „desna grana“ izvodi u slučaju da je uvjet laž.



Primjer 2: Program učitava dvije vrijednosti (učitane vrijednosti spremaju se u varijable). Od veće vrijednosti oduzima se manja učitana vrijednost. Tako dobiveni rezultat se ispisuje i on je uvijek veći ili jednak nuli (>= 0).



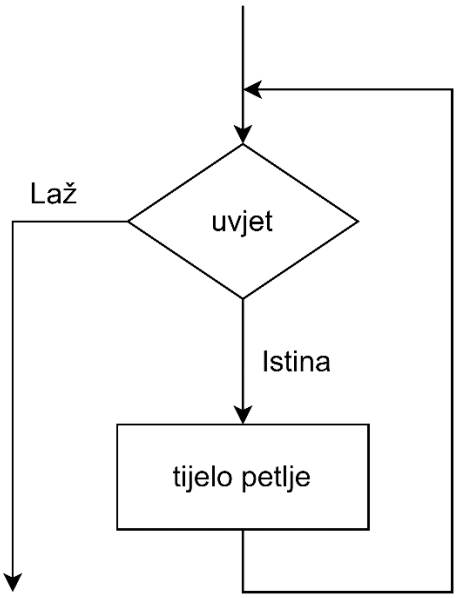
Napomena

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

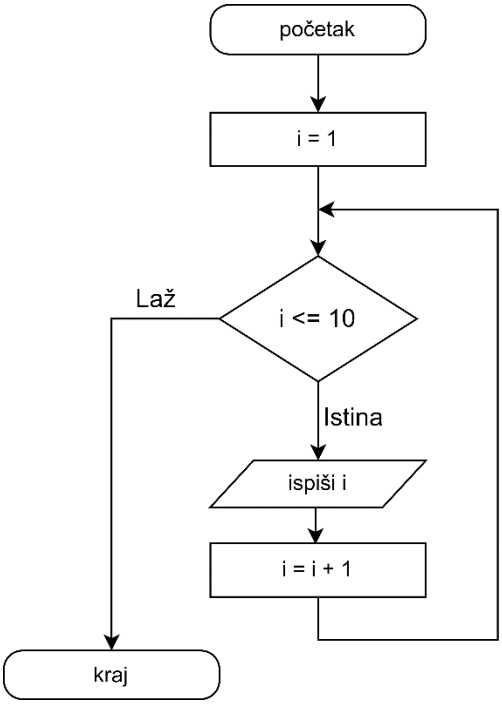
Obrazloženje rješenja*:* priloženi dijagram toka prikazuje algoritam koji učitava dva broja, a učitani brojevi spremaju se u varijable `a` i `b`. Nakon učitavanja vrijednosti slijedi odluka o tome je li vrijednost varijable `a` manja od vrijednosti varijable `b`. Ako je uvjet provjere istinit, u varijablu `rezultat` sprema se rezultat izraza `b-a`, dok se u suprotnom primjeru u varijablu `rezultat` sprema rezultat izraza `a-b`. Nakon izvršenog oduzimanja rezultat spremljen u varijablu `rezultat` ispisuje se na ekranu. Priloženi algoritam radi matematičku operaciju oduzimanja čiji rezultat nikada neće biti negativan, već će uvijek biti `x >= 0`.

Programske petlje

Programska petlja je konstrukt koji omogućava da se neki dio programskog kôda ponavlja toliko dugo dok je uvjet zadovoljen. Onog trenutka kada uvjet više nije zadovoljen izlazi se iz petlje te se nastavlja izvršavati programski kôd koji se nalazi neposredno u nastavku petlje.



Primjer 3: Program ispisuje sve brojeve od 1 do 10.



Obrazloženje rješenja*:* varijabla `i` postavlja se na početnu vrijednost 1. Ispis vrijednosti varijable `i` te njeno uvećavanje za 1 u svakom koraku petlje radi se toliko dugo dok je zadovoljen uvjet `i <= 10`; onog trenutka kada uvjet prestane biti zadovoljen, program završava.

1.3. Pseudokôd

Budući da su dijagrami toka prilikom opisivanja složenih algoritama izrazito nepregledni, bolja zamjena za njih je pseudokôd. Pseudokôd koristi termine govornoga jezika i on je puno prikladniji za opis složenijih algoritama. Prilikom opisa nekog algoritma pseudokôd u svojem sadržaju koristi izraze kojima se ljudi svakodnevno koriste. Na autoru pseudokôda je da odluči koliko će detaljno ići riječima opisivati neki algoritam (treba pronaći zlatnu sredinu jer se s previše detalja gubi smisao opisivanja algoritma pseudokôdom, a opet ako imamo premalo detalja, algoritam može biti nejasan, osobito početnicima). Pseudokôd mora biti napisan općenito tako da se ne vide detalji implementacije koji su ovisni o nekom konkretnom programskom jeziku, na primjer Pythonu. U nastavku su prikazani primjeri pseudokôda temeljeni na primjerima iz poglavlja *1.2. Dijagrami toka*.

Napomena

U knjigama se složeniji algoritmi opisuju pomoću pseudokôda.

Primjer 1: Program s tipkovnice čita jednu vrijednost koja predstavlja polumjer. Potrebno je izračunati opseg kruga (rezultat spremiti u varijablu imena `o`) te površinu kruga (rezultat spremiti u varijablu imena `p`). Tako dobiveni rezultat neka se ispiše.

```
početak

učitaj(r)
o = 2*r*3.14
p = r*r*3.14
ispiši(o, p)

kraj
```

Uvjetno grananje

```
ako je uvjet onda
      naredbe_za_istinit_uvjet
inače
      naredbe_za_lažan_uvjet
```

Primjer 2: Program učitava dvije vrijednosti (učitane vrijednosti spremaju se u varijable). Od veće vrijednosti oduzima se manja učitana vrijednost. Tako dobiven rezultat ispisuje se i on je uvijek veći ili jednak nuli (>= 0).

```
početak
učitaj(a, b)
ako je a < b tada
      x := b - a
inače
      x := a - b
ispiši(x)
kraj
```

Programske petlje

```
dok je uvjet raditi
      tjelo_petlje
```

Primjer 3: Program ispisuje sve brojeve od 1 do 10.

```
početak

i = 1

dok je i <= 10 raditi
      ispiši(i)
      i = i + 1

kraj
```

1.4. Vježba: Dijagram toka, pseudokôd

1. Kreirajte dijagram toka koji učitava dva broja te ih sprema u varijable imena `a` i `b`. Te dvije vrijednosti predstavljaju duljine stranica pravokutnika. Izračunajte i spremite u pripadajuće varijable opseg (2*a + 2*b) i površinu (a*b) pravokutnika te ispišite izračunate vrijednosti.
2. Riješite prethodni zadatak pomoću pseudokôda.
3. 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, a 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.
4. Riješite prethodni zadatak pomoću pseudokôda.
5. Kreirajte dijagram toka koji učitava jedan broj s tipkovnice. Ispišite sve brojeve od 1 do učitanog broja (uključujući njega samog).
6. Riješite prethodni zadatak pomoću pseudokôda.
7. * 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“.
8. * Riješite prethodni zadatak pomoću pseudokôda.
9. * 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.
10. * Riješite prethodni zadatak pomoću pseudokôda.

1.5. Pitanja za ponavljanje: Dijagram toka, pseudokôd

1. Kod dijagrama toka, koji oblik služi za upis ulaznih i ispis izlaznih vrijednost?
2. Kod dijagrama toka, koji oblik služi za uvjetno grananje?
3. Koja je prednost pseudokôda pred dijagramom toka?

2. Prvi program

Trajanje poglavlja:

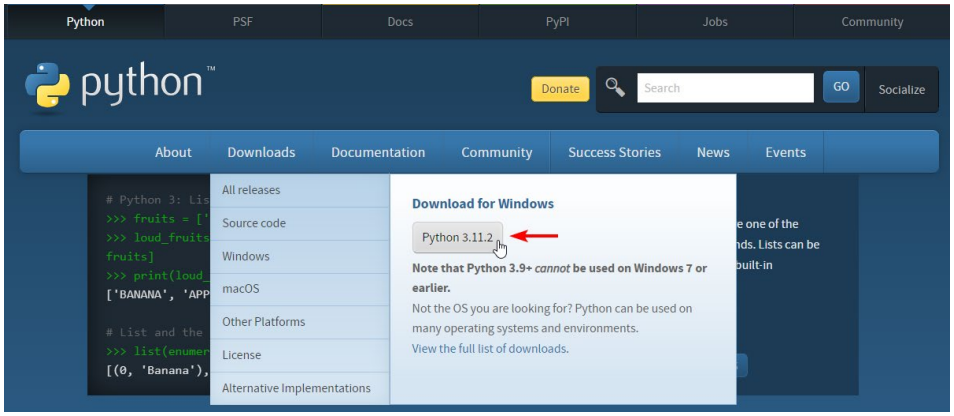
10 min

Po završetku ovog poglavlja polaznik će moći:

- pripremiti radno okruženje za rad s programskim jezikom Python
- napisati i pokrenuti svoj prvi program napisan u Pythonu.

2.1. Instalacija radnog okruženja

Na stranici https://www.python.org/ treba skinuti i instalirati zadnju dostupnu verziju programskog jezika Python 3. Minimalna potrebna verzija za ovaj tečaj je Python 3.6.0.



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

Python – verzije

Verzija Pythona je označena s A.B.C. Svako od tih triju 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.

Službena dokumentacija

Službena dokumetacija Pythona 3 [7] dostupna je na poveznici:

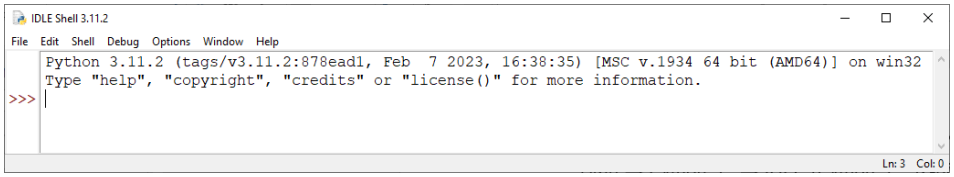
https://docs.python.org/3

2.2. Pisanje prvog programa

Pokrenuti IDLE koji dolazi s instalacijom Pythona.

Start → Python 3. → IDLE (Python 3.* 64-bit)*

Nakon pokretanja IDLE-a otvara se IDLE Shell [4]. Taj program možemo smatrati naredbenim retkom u kojem 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*).



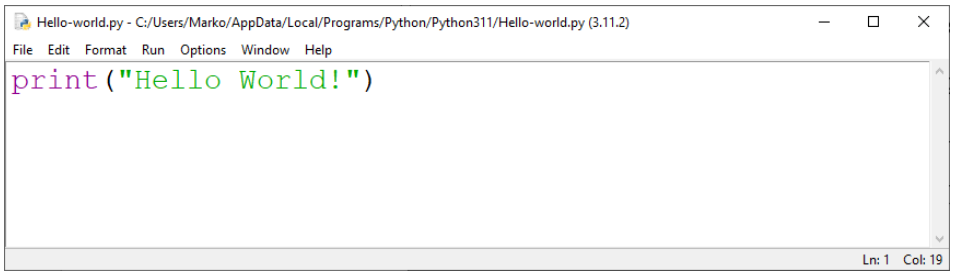
Ako želimo napisati program koji se sastoji od više linija, koji lagano možemo uređivati kao i svaki drugi dokument na računalu, najprikladnije

je da preko IDLE Shella otvorimo novi dokument u kojem ćemo pisati programski kôd. U IDLE Shellu potrebno je u izborniku odabrati:

File → New file

Primijetimo da nam se sada otvorio prozor za unos teksta. U ovom prozoru piše se programski kôd. U nastavku se nalazi linija programskog kôda koja će nam poslužiti za prvi primjer kôda pisanog u programskom jeziku Python.

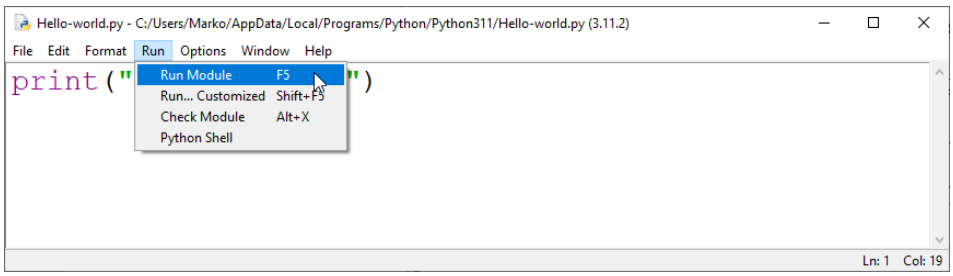
```
print("Hello World!")
```



Nakon što je gore navedena linija programskog kôda napisana, potrebno je pokrenuti program kako bismo unutar IDLE Shella dobili rezultat. Pokretanje napisanog programskog kôda moguće je napraviti na dva načina:

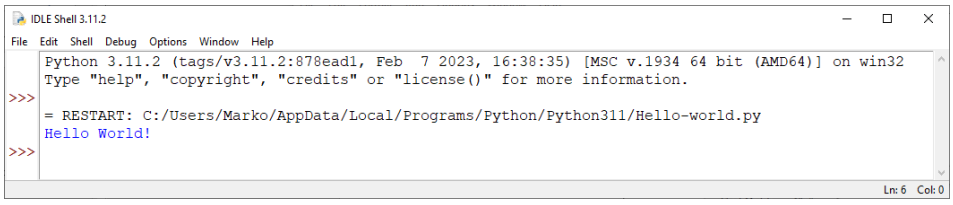
- Kroz izbornik:

  Run → Run module



- Pritiskom na tipku *F5* koja se nalazi na tipkovnici

Nakon pokretanja napisanog programskog kôda rezultat izvođenja programa (rečenica `"Hello World!"`) pojavit će se u prozoru koji smo prvo otvorili, tj. u IDLE Shellu.



Napomena: Funkcija `print()` služi za ispis teksta i ona je detaljnije objašnjenja u poglavlju *3.4. Funkcija* `print()`. Za sada uzmimo funkciju `print()` kao naredbu koju pozivamo kada želimo ispisati neku vrijednost, a vrijednost koju želimo ispisati stavljamo u oble zagrade.

2.3. Alternativna radna okruženja

Okruženja u kojima je moguće pisati programski kôd mogu se podijeliti u dvije osnovne skupine:

- uređivači kôda (engl. *Code editor*)
- *IDE* (engl. *Integrated development environment*).

Alati iz obje skupine pružaju programerima okruženje za pisanje programskog kôda. Uređivači kôda su jednostavnija okruženja od IDE-a.

Uređivači kôda

Iako se programski kôd može pisati u bilo kojem tekstualnom uređivaču (na primjer Notepadu), korištenje specijaliziranih alata, u ovom slučaju uređivača kôda, programerima se omogućava da brže i učinkovitije pišu programski kôd. Osnovne prednosti naspram običnih tekstualnih uređivača su: isticanje sintakse (bojanje ključnih riječi), olakšano formatiranje (uvlačenje), automatsko dovršavanje, podcrtavanje sintaksnih grešaka ako su napravljene i slično. Ove značajke uvelike pomažu programerima u pisanju kvalitetnijeg programskog kôda.

Najpopularniji besplatni uređivači kôda:

- Visual Studio Code
- Atom
- Eclipse
- Notepad++.

IDE

Uz sve prednosti uređivača kôda, IDE pruža dodatne funkcionalnosti koje programerima omogućavaju još jednostavniji razvoj programskih rješenja. Unutar alata IDE dolazi integriran Python interpreter za pokretanje programa, automatsko dovršavanje koje je prilagođeno za programski jezik za koji je IDE namijenjen, sustav za otklanjanje pogrešaka (engl. *debug*). Također, oni nude bolju organizaciju i pregled kôda te funkcionalnosti za upravljanje projektima.

Najpopularniji besplatni IDE-ovi za Python:

- IDLE (Integrated Development and Learning Environment)
- PyCharm
- Jupyter
- Spyder
- PyDev.

Za potrebe ovog tečaja zbog jednostavnosti koristit će se radno okruženje IDLE.

2.4. Vježba: Prvi program

1. Pokrenuti IDLE te ispisati niz znakova: „Hello World!“.

3. Osnove programskoga jezika Python

Trajanje poglavlja:

125 min

Po završetku ovog poglavlja polaznik će moći:

- koristiti jednolinijske i višelinijske komentare u kôdu
- koristiti varijable te znati raspoznati ključne riječi
- ispisivati vrijednosti na ekran
- koristiti aritmetičke operatore, operatore usporedbe, logičke operatore te sastavljati složene izraze pomoću njih
- primjenjivati tipove podataka: brojeve, nizove znakova, logičke vrijednosti
- koristiti ugrađene funkcije i metode za rad s brojevima i nizovima znakova.

3.1. Komentari

Kao i u drugim programskim jezicima, i u Pythonu postoje komentari koji programerima služe za opisivanje i objašnjavanje programskog kôda [1].

Komentari se ne izvršavaju, tj. interpretiraju, te oni ne utječu na rezultat programa (prilikom izvođenja programa komentari se zanemaruju, kao da ne postoje). Oni su namijenjeni programerima koji čitaju programski kôd kako bi im omogućili lakše snalaženje u kôdu.

Razlozi korištenja komentara:

- opisivanje namjene kôda: opisuje se što kôd radi, na primjer funkcionalnost nekog segmenta programskog kôda kao što je funkcija
- pojašnjenje algoritma: opisuje se algoritam (logika), tj. zašto je kôd napisan baš na taj način; ovo se obično koristi kada je programski kôd kompliciran te je teško pratiti na jednostavan i brz način zašto je baš tako napisan
- dokumentiranje kôda: programski kôd dokumentira se kako bi se drugim programerima olakšalo razumijevanje i rad na kôdu
- debugiranje (engl. *debugging*): komentari se mogu koristiti za isključivanje dijelova kôda tijekom procesa pronalaženja grešaka i nedostataka u programu
- izostavljanje kôda: ako je potrebno izostavili dijelove kôda koji se trenutno ne koriste, oni se mogu komentirati i na taj se način trenutno nepotrebne linije programskog kôda zadržavaju u kôdu za potencijalnu buduću uporabu.

Komentari mogu biti jednolinijski ili višelinijski.

Jednolinijski komentari

Jednolinijski komentari u Pythonu počinju znakom „hash“ – `#`. Sve što se nalazi nakon znaka za početak komentara pa do kraja linije bit će zanemareno od strane Python interpretera.

Primjer 1: Jednolinijski komentar sa sljedećim varijacijama:

1. linija – početak komentara je na početku linije
2. linija – komentar počinje nakon programskog kôda
3. linija – komentar počinje uvučeno od početka linije, no ispred njega ne nalazi se nikakav programski kôd.

Komentari

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

```
# Ovo je prvi jednolinijski komentar
var1 = 1  # ovo je drugi jednolinijski komentar
          # ... a ovo je treći jednolinijski komentar!

print(var1)
```

```
Izlaz:
      1
```

Višelinijski komentari

Višelinijski komentari protežu se kroz nekoliko linija, oni počinju i završavaju s `'''` ili `"""` (tri jednostruka ili tri dvostruka navodnika). Sve što se nalazi između ovih znakova bit će zanemareno od strane Python interpretera. Pomoću višelinijskih komentara možemo napisati neki opisni tekst kroz nekoliko linija ili pak komentirati cijeli odsječak programskog kôda (bilo da isti više nije potreban ili se privremeno želi obustaviti njegovo izvođenje).

Napomena: Znak za početak višelinijskog komentara (tri jednostruka ili tri dvostruka navodnika) mora biti napisan na početku retka. Ispred znaka za početak višelinijskog komentara ne smije se pisati nikakav programski kôd, a ne smiju se nalaziti ni praznine, tj. razmaci.

Primjer 2: Višelinijski komentar s jednostrukim navodnicima.

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

Primjer 3: Višelinijski komentar s dvostrukim navodnicima.

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

3.2. Varijable

Varijabla je osnovni koncept koji se koristi za čuvanje vrijednosti. Spremljenim vrijednostima (podacima) moguće je pristupiti iz programskog kôda. Osim što je u varijablu moguće spremiti željenu vrijednost, ta vrijednost može se promijeniti, a varijablu je također moguće i uništiti. Svaka varijabla određena je svojim imenom i memorijskom lokacijom na kojoj je zapisana njena vrijednost.

Varijable u programskom jeziku Python nije potrebno deklarirati (pridružiti tip podataka koji će biti spremljen u varijablu, na primjer cijeli broj, decimalni broj, niz znakova itd.). Pridruživanje tipa podataka varijabli Python radi automatikom u trenutku kada se varijabli pridružuje 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 drugim stvarima (drugim vrijednostima) itd. U varijable je moguće spremiti razne tipove podataka, no u ovom će se dijelu tečaja u varijable spremati samo brojevi radi lakšeg razumijevanja gradiva.

Vrijednost se nekoj varijabli pridružuju pomoću znaka jednakosti =. Ime varijable u koju želimo spremiti neku vrijednost nalazi se s lijeve strane znaka = , dok se s desne strane nalazi vrijednost koju želimo spremiti u varijablu.

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`.

Primjer 1: Varijabli imena `mojaCijelobrojnaVarijabla` pridružuje se vrijednost 5. U četvrtoj liniji pozivom funkcije `print(mojaCijelobrojnaVarijabla)` Python ispisuje vrijednost koja je spremljena u varijabli imena `mojaCijelobrojnaVarijabla`. Iz tog se razloga na zaslonu ispisuje vrijednost 5. Varijabla se može koristiti u bilo kojem dijelu programa na način da se upotrijebi njeno ime.

```
mojaCijelobrojnaVarijabla = 5
mojaDecimalnaVarijabla = 5.55
mojaZnakovnaVarijabla = "Srce"
print(mojaCijelobrojnaVarijabla)
print(mojaDecimalnaVarijabla)
print(mojaZnakovnaVarijabla)
```

```
Izlaz:
      5
      5.55
      Srce
```

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

Na primjer, ako je potrebno u varijablu spremiti trenutnu godinu, tu varijablu pametnije je nazvati `trenutnaGodina`, a ne `godina`.

Nazivi varijabli bitan su dio samodokumentirajućeg kôda jer jasno imenovane varijable mogu uvelike pomoći u razumijevanju kôda. Ako su varijable dobro imenovane, programer koji čita programski kôd može brzo shvatiti što se događa u njemu, bez potrebe za detaljnim proučavanjem. Na primjer, ako koristimo varijablu koja se zove `trenutnaGodina`, teško može doći do zabune o kojoj je godini u toj varijabli riječ, no ako bismo koristili varijablu imena `godina`, programer koji čita programski kôd morao bi pozornije popratiti je li u toj varijabli spremljena trenutna godina ili neka druga.

Primjeri ispravnog imenovanja varijabli:

- `var1`, `Var2`
- `var_1`, `Var_2`
- `a`, `B`
- `zbrojbrojeva`, `zbrojBrojeva`, `ZbrojBrojeva`
- `zbroj_brojeva`, `zbroj_Brojeva`, `Zbroj_Brojeva`.

Primjeri neispravnog imenovanja varijabli:

- u imenu varijable nalaze se nedozvoljeni znakovi (znakovi koji nisu slova, brojevi i znak _)
  - `a.1`
  - `a(1)`
  - `euro_€`
  - `a-b`
- prvi znak imena varijable **ne smije** biti broj.

Python je osjetljiv na velika i mala slova (engl. *case sensitive*). To znači da velika i mala slova u imenima varijabli, funkcija, metoda, ključnih riječi itd. moraju biti točno napisana kako bi se program ispravno izvršavao. Na primjer, `varijabla1` i `Varijabla1` dvije su različite varijable.

3.3. Ključne riječi

Ključne riječi su osnovni gradivni elementi programskog jezika, koje imaju unaprijed određeno značenje. Nazivi varijabli ne smiju biti identični ključnim riječima. Listu ključnih riječi moguće je dobiti pokretanjem naredbe: `help("keywords")` u IDLE Shellu. Tablica ključnih riječi:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	async
break	except	in	raise	await

3.4. Funkcija `print()`

Funkcije

Funkcije se dijele na:

- ugrađene
- korisnički definirane.

U ovom poglavlju obrađuju se isključivo ugrađene funkcije, dok se korisnički definirane funkcije obrađuju u poglavlju *6. Funkcije*.

Funkcija `print()` koristi se za ispis poruka odnosno podataka na ekranu. Funkcija `print()` može primiti nekoliko argumenata. U nastavku slijedi detaljan opis triju najčešće korištenih argumenata funkcije `print()`, a spomenut će se i dva dodatna argumenta čije se predefinirane vrijednosti u ovom tečaju neće mijenjati.

Prototip funkcije `print()`:

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

- `*objects` – argumenti (može ih biti više) u pozivu funkcije `print()` su vrijednosti ili 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 kod poziva funkcije `print()`, oni se međusobno odvajaju vrijednošću parametra `sep` i završavaju vrijednošću parametra `end`.
- `sep=' '` – ovaj parametar označava znak kojim su vrijednosti što se ispisuju međusobno odvojene. Ako ovaj argument nije zadan u funkciji `print()` prilikom poziva, 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()`, on poprima predefiniranu vrijednost, tj. znak `'\n'`. Znak `'\n'` predstavlja novi redak. Što bi značilo da se nakon što se sve vrijednosti ispišu, ispisuje i znak za novi redak.
- `file=None`
- `flush=False`

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 budući da nema predanih argumenata, to je predefinirana vrijednost `'\n'`, tj. novi redak.

Primjer 1: Funkciji `print()` predani su samo argumenti koji se žele ispisati, bez dodatnih argumenata – `sep` i `end`. Prilikom ispisa funkcija je između varijabli ispisala predefiniranu vrijednost, tj. razmak, a nakon ispisa svih varijabli ispisala je i znak `'\n'`, tj. ispis sljedećeg poziva funkcije `print()` nalazi se u novom retku.

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

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

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

Primjer 2: 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, sep=' --- ')
print("Novi redak.")
```

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

Primjer 3: U prvom pozivu funkcije `print()` uz varijable čije se vrijednosti žele ispisati, prenesen je i argument `end` čija je vrijednost zamijenila predefiniranu vrijednost parametra `end`, tj. vrijednost `'\n'`. Nakon ispisa svih prenesenih argumenata u funkciju `print()` ispisao se niz znakova `"...\n\n"`. U drugom pozivu funkcije `print()` ispisuje se predani niz znakova i potom predefinirana vrijednost parametra `end`, tj. vrijednost `'\n'`. U trećem pozivu funkcije `print()` opet se ispisuje predefinirana vrijednost parametra `end`, ali se u ovom primjeru parametar `sep` postavlja na ništa, čime se postiže da se s lijeve i desne strane broja 1244 ne nalaze dodatni razmaci.

```
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!
```

3.5. Vježba: Komentari, varijable, ključne riječi i funkcija `print()`

1. Napišite program koji se sastoji od pet varijabli proizvoljnih vrijednosti. Jednim pozivom funkcije `print()` ispišite svih pet vrijednosti.
2. Nastavno na prethodni zadatak, doradite poziv funkcije `print()` tako da se između svake pojedine varijable ispiše znak `#`.
3. Nastavno na prvi zadatak, doradite poziv funkcije `print()` tako da vrijednost svake pojedine varijable bude u novom redu.
4. Nastavno na 3. zadatak, nakon što su sve vrijednosti ispisane, svaka u svojem retku, u zadnjem novom retku ispišite još i niz znakova „Kraj“.
5. Prethodni zadatak uredite tako da isprobate korištenje jednolinijskih i višelinijskih komentara.

3.6. Operatori

Operatori su specijalni simboli koji se koriste u programiranju za izvršavanje različitih matematičkih, logičkih i drugih operacija nad vrijednostima podataka. Oni programerima omogućuju da stvaraju složene izraze i izvršavaju različite vrste operacija nad podacima. Operatori se klasificiraju prema svojoj funkciji, a u ovom poglavlju obradit će se sljedeće vrste operatora: aritmetički operatori, operatori usporedbe i logički operatori. Također, prikazat će se kako kreirati logičke izraze koji omogućavaju stvaranje složenih uvjeta.

3.6.1. Aritmetički operatori

Aritmetički operatori su matematički operatori koji se koriste za izvođenje osnovnih aritmetičkih operacija na brojevima. U nastavku je prikazana tablica u kojoj se nalazi popis aritmetičkih operatora.

Operator	**Naziv operatora**	**Izraz**	**Primjer** `a=10; b=3`
`+`	Zbrajanje (engl. *addition*)	`x = a+b`	`x = 13`
`-`	Oduzimanje (engl. *subtraction*)	`x = a-b`	`x = 7`
`*`	Množenje (engl. *multiplication*)	`x = a*b`	`x = 30`
`/`	Dijeljenje (engl. *division*)	`x = a/b`	`x = 3.333`
`%`	Ostatak dijeljenja (engl. *modulus*)	`x = a%b`	`x = 1`
`**`	Potenciranje (engl. *power*)	`x = a**b`	`x = 1000`
`//`	Cjelobrojno dijeljenje (engl. *flor division*)	`x = a//b`	`x = 3`

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	Izraz	Osnovni izraz	Primjer a=10; b=3
+=	a += b	a = a+b	a = 13
-=	a -= b	a = a-b	a = 7
*=	a *= b	a = a*b	a = 30
/=	a /= b	a = a/b	a = 3.333
%=	a %= b	a = a%b	a = 1
**=	a **=b	a = a**b	a = 1000
//=	a //= b	a = a//b	a = 3
=	*a = b*	*a = b*	*a = 3*

Kao što se može vidjeti iz gornje tablice, u usporedbi s osnovnim oblikom aritmetičkih operatora, skraćeni operatori smanjuju složenost izraza, tj. predstavljaju kraći zapis identičnog izraza. Uzmimo za primjer operator `+=`. Ako je varijablu `a` potrebno uvećati za vrijednost varijable `b`, skraćenim aritmetičkim operatorom to bi se zapisalo kao `a += b`. To je identičan izraz kao `a = a + b`, ali zapisan na drugačiji način. Pomoću ovih operatora programer može pojednostaviti izgled programskoga kôda i povećati njegovu čitljivost.

Prioritet izvršavanja aritmetičkih operatora

U programiranju se aritmetički operatori izvršavaju po redoslijedu prioriteta (kao i u matematici). Operatori s višim prioritetom (1) izvršavaju se prije operatora s nižim prioritetom (3). Ako unutar izraza ima više operatora s istim prioritetom, operatori se izvršavaju slijeva nadesno. U nastavku slijedi tablica prioriteta aritmetičkih operatora.

Prioritet	Operator
1	**
2	*, /, //, %
3	+, -

Primjer 1: Operator potenciranja ima najviši prioritet, pa se prvo izvršava operacija potenciranja, a nakon toga operacija množenja.

```
rezultat = 2 ** 3 * 4
print(rezultat)
```

```
Izlaz:
      32
```

Primjer 2: Svi korišteni operatori imaju isti prioritet te se iz tog razloga rezultat izračunava slijeva nadesno.

```
rezultat = 8 // 3 * 4 % 3
print(rezultat)

rezultat = 10 % 3 / 2 * 4
print(rezultat)
```

```
Izlaz:
      2
      2.0
```

Primjer 3: Operatori zbrajanja i oduzimanja imaju najniži prioritet, pa se izvršavaju nakon operacija potenciranja, množenja, dijeljenja i cjelobrojnog dijeljenja ostatka dijeljenja.

```
rezultat = 4 + 5 * 6 - 7 / 2
print(rezultat)

rezultat = 10 - 4 % 3 + 2 // 5
print(rezultat)
```

```
Izlaz:
      30.5
      9
```

Primjer 4: Korištenjem zagrada može se promijeniti predefinirani redoslijed izvršavanja aritmetičkih operatora. Izrazi u zagradama izvršavaju se prvi.

```
rezultat = (4 + 5) * (6 - 7) / 2
print(rezultat)

rezultat = (10 - 4) % (3 + 2) // 5
print(rezultat)
```

```
Izlaz:
      -4.5
      0
```

Primjer 5: Korištenje skraćenih aritmetičkih operatora. Skraćene aritmetičke operatore nije moguće pisati kao argumente funkcija, npr. `print(a += 1)`.

```
a = 5
print(a)

a += 1
print(a)

a *= 5
print(a)

a %= 2
print(a)
```

```
Izlaz:
      5
      4
      30
      0
```

Primjer 6: Praktična upotreba aritmetičkog operatora modulo `%` (operator modulo vraća ostatak dijeljenja cijelih brojeva) česta je kod detekcije parnosti brojeva. Na primjer, ako moramo detektirati je li neki broj paran, napravimo operaciju `%2` nad danim brojem i kao rezultat

dobit će se `0` ili `1`. Ako je rezultat `0` – broj je paran, ako je rezultat `1` – broj je neparan.

```
a = 1 % 2
print(a)

a = 2 % 2
print(a)

a = 11 % 2
print(a)

a = 12 % 2
print(a)
```

```
Izlaz:
      1
      0
      1
      0
```

3.6.2. Operatori usporedbe

Operatori usporedbe koriste se za uspoređivanje vrijednosti podataka. Rezultat operatora usporedbe uvijek je ili istina (engl. *True*) ili laž (engl. *False*). Operatori usporedbe najčešće se koriste za kontrolu toka programa, tj. u uvjetima provjere točnosti rezultata logičkih operatora. U tablici prikazanoj u nastavku nalazi se popis operatora usporedbe.

Operator	Naziv operatora	Izraz
==	Usporedba jednakosti	`a == b`
!=	Usporedba nejednakosti	`a != b`
>	Strogo veće od	`a > b`
<	Strogo manje od	`a < b`
>=	Veće ili jednako od	`a >= b`
<=	Manje ili jednako od	`a <= b`

U tablici koja se nalazi u nastavku prezentirani su svi operatori usporedbe i njihovi rezultati s tri kombinacije vrijednosti spremljene u varijablu `a` i u varijablu `b`. U prvoj kombinaciji vrijednost varijable `a` je manja od vrijednosti varijable `b`, u drugoj kombinaciji je vrijednost varijable `a` veća od vrijednosti varijable `b`, dok su u trećoj kombinaciji vrijednosti spremljene u varijablu `a` i u varijablu `b` jednake.

Izraz	Primjer a=5; b=10	Primjer a=10; b=5	Primjer a=5; b=5
a == b	False	False	True
a != b	True	True	False
a > b	False	True	False
a < b	True	False	False
a >= b	False	True	True
a <= b	True	False	True

Kod operatora usporedbe svi operatori imaju jednak prioritet. U nastavku slijedi tablica prioriteta operatora usporedbe.

Prioritet	Operator
1	<, <=, >, >=, !=, ==

Primjer 1: Vrijednost varijable `a` veća je od vrijednosti varijable `b`.

```
a = 5
b = 10
print(a == b)
print(a != b)
print(a < b)
print(a > b)
print(a <= b)
print(a >= b)
```

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

Primjer 2: Vrijednost varijable `a` jednaka je vrijednosti varijable `b`.

```
a = 5
b = 5

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

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

3.6.3. Logički operatori

Logički operatori koriste se za kreiranje složenijih logičkih izraza. Rezultat logičkih izraza je ili istina (engl. *True*) ili laž (engl. *False*).

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

Operator `and`

Logički operator `and` koristi se kada želimo provjeriti jesu li oba uvjeta istinita (uvjet s lijeve i desne strane logičkog operatora).

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

Tablica stanja operatora `and`:

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

Operator `or`

Logički operator `or` koristi se kada želimo provjeriti je li barem jedan uvjet istinit.

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

Tablica stanja operatora `or`:

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

Operator `not`

Logički operator `not` koristi se za invertiranje logičkog izraza. Ako je logički izraz `True`, `not` operator će ga pretvoriti u `False`, a ako je logički izraz `False`, `not` operator će ga pretvoriti u `True`.

Tablica stanja operatora `not`:

A	not A
False	True
True	False

Logički operatori također se izvršavaju po redoslijedu prioriteta. Operatori s višim prioritetom (1) izvršavaju se prije operatora s nižim prioritetom (3). U nastavku slijedi tablica prioriteta logičkih operatora.

Prioritet	Operator
1	`not`
2	`and`
3	`or`

Primjer 1: Jedna vrijednost varijable je `True`, a druga je `False`.

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

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

Primjer 2: Obje vrijednosti varijable su `True`.

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

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

Primjer 3: Obje vrijednosti varijable su `False`.

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

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

Napomena: Obratite pažnju na velika i mala slova kod pisanja logičkih operatora. Logički operatori moraju biti napisani u potpunosti malim slovima: `and`, `or`, `not`. `True` i `False` počinju velikim početnim slovom.

3.6.4. Složeni izrazi

Složeni izrazi sastoje se od više jednostavnih izraza koji su povezani operatorima. U ovom poglavlju prikazat će se kako kombinirati aritmetičke operatore, operatore usporedbe i logičke operatore. Složeni izrazi koriste se za opisivanje složenih procesa i algoritama. Važno je imati dobro razumijevanje operatora i pravila prioriteta operatora kako bi se složeni izrazi pravilno napisali te dobili ispravnu (željena) funkcionalnost. U prethodnim potpoglavljima prikazan je redoslijed izvođenja operacija za slučajeve kada se unutar izraza koristi samo jedna vrsta operatora. Ako se operatori kombiniraju, potrebno je obratiti pozornost na redoslijed izvođenja operacija unutar složenih izraza. U nastavku se nalazi tablica koja prikazuje prioritet izvođenja operatora po vrsti:

Prioritet	Vrsta operatora
1.	Aritmetička operacija
2.	Operatori usporedbe
3.	Logička operacija

U nastavku se nalazi tablica koja prikazuje prioritet izvođenja svih operatora koji postoje, uključujući operatore koji se na ovom tečaju niti ne obrađuju.

Prioritet	Operator
1.	`(expressions...), [expressions...], {key: value...}, {expressions...}`
2.	`x[index], x[index:index], x(arguments...), x.attribute`
3.	`await x`
4.	`**`
5.	`+x, -x, ~x`
6.	`*, @, /, //, %`
7.	`+, -`
8.	`<<, >>`
9.	`&`
10.	`^`
11.	`|`
12.	`in, not in, is, is not, <, <=, >, >=, !=, ==`
13.	`not x`
14.	`and`
15.	`or`
16.	`if – else`
17.	`lambda`
18.	`:=`

Primjer 1: Operatori usporedbe imaju viši prioritet od logičkog operatora. Također, naveden je primjer sa zagradama koji nedvosmisleno prikazuje redoslijed izvršavanja.

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

rezultat = a > b or a < c
print(rezultat)

rezultat = (a > b) or (a < c)
print(rezultat)
```

```
Izlaz:
      True
      True
```

Napomena

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

Primjer 2: Aritmetički operator `%` ima viši prioritet od operatora `==` te se iz tog razloga najprije izvršava `a%2`, a potom se uspoređuje je li dobiveni rezultat jednak broju 0. Nakon što je lijeva strana izraza razriješena, uspoređuje se je li vrijednost zapisana u varijabli `b` veća od broja 10. Na kraju se pomoću logičkog operatora `and` provjerava je li s lijeve strane i s desne strane operatora `and` istinit izraz.

```
a = 10
b = 15

rezultat = 0 == a % 2 and b > 10
print(rezultat)

rezultat = (0 == a % 2) and (b > 10)
print(rezultat)
```

```
Izlaz:
      True
      True
```

Primjer 3: Sve tri verzije složenih izraza izvršavaju se na potpuno jednak način. Jedina je razlika što je 2. i 3. složeni izraz prikazan sa zagradama kako bi se jasnije prikazao redoslijed izvršavanja.

```
rezultat = 5 + 8 > 20 / 2 and 100 / 2 > 10
print(rezultat)

rezultat = (5 + 8) > (20 / 2) and (100 / 2) > 10
print(rezultat)

rezultat = ((5 + 8) > (20 / 2)) and ((100 / 2) > 10)
print(rezultat)
```

```
Izlaz:
      True
      True
      True
```

3.7. Vježba: Operatori

1. Kreirajte dvije varijable i pridijelite im proizvoljne cjelobrojne vrijednosti te uz pomoć aritmetičkih operatora (zbrajanje, oduzimanje, množenje, dijeljenje, ostatak dijeljenja, potenciranje, cjelobrojno dijeljenje) ispišite rezultate.
2. Kreirajte dvije varijable i pridijelite im proizvoljne cjelobrojne vrijednosti te uz pomoć skraćenih aritmetičkih operatora (zbrajanje, oduzimanje, množenje, dijeljenje, ostatak dijeljenja, potenciranje, cjelobrojno dijeljenje) ispišite rezultate.
   Napomena 1: Nakon svakog korištenja skraćenog aritmetičkog operatora potrebno je varijablu čija se vrijednost mijenja postaviti na početnu vrijednost.
   Napomena 2: Skraćene aritmetičke operatore nije moguće pisati unutar poziva funkcije.
3. Kreirajte dvije varijable i pridijelite im proizvoljne cjelobrojne vrijednosti te ispišite rezultat operatora usporedbe (usporedba jednakosti, nejednakosti, strogo veće, strogo manje, veće ili jednako, manje ili jednako).
4. Kreirajte dvije varijable i pridijelite im vrijednosti `True` ili `False`. Pomoću varijabli testirajte logičke operatore (*`and`, `or`, `not`*).
5. Napišite program koji će u varijablu spremiti proizvoljnu vrijednost temperature (izražene u Celzijevim stupnjevima). Na ekranu treba ispisati vrijednost temperature u Fahrenheitovim stupnjevima. Formula: $(x \times \frac{9}{5}) + 32$, `x` je vrijednost izražena u Celzijevim stupnjevima.
6. 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.
   Napomena: Aritmetička sredina računa se tako da se sve dane vrijednosti zbroje te se rezultat podijeli brojem varijabli.
7. 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.
8. Korištenjem skraćenih aritmetičkih operatora rezultat dobiven iz prethodnog zadatka pomnožite sa 100 i ispišite dobivenu vrijednost.
9. 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`).
10. 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 1: 159 % 10 = 9, 159 % 100 = 59.
    Napomena 2: Za zamjenu vrijednosti dviju varijabli potrebno je uvesti treću, pomoćnu varijablu.

3.8. Brojevi

Python razlikuje tri osnovna tipa brojeva:

- cijeli brojevi
- realni brojevi
- kompleksni brojevi.

Cijeli brojevi

Cijeli brojevi predstavljaju se cjelobrojnim tipom `int` (engl. *integer*). Skup svih cijelih brojeva čine: pozitivni cijeli brojevi, nula i negativni cijeli brojevi. Skup cijelih brojeva označava se oznakom Z.

$$Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$$

Python je dinamički tipiziran jezik (engl. *dynamically typed language*), što znači da tip varijable nije određen prilikom definicije varijable, već se tip određuje prilikom dodjele vrijednosti varijabli tijekom izvođenja programa. Varijable u Pythonu mogu mijenjati svoj tip podataka tijekom izvođenja programa. Ako se želi dohvatiti tip objekta, može se koristiti funkcija imena `type()`.

Primjer 1: Korištenje funkcije imena `type()` nad varijablom u kojoj je spremljena cjelobrojna vrijednost.

```
a = 10
print(type(a))
```

```
Izlaz:
      <class 'int'>
```

Primjer 2: U memoriji računala ovi brojevi pohranjuju se u niz osnovnih memorijskih jedinki (procesorskih riječi) te iz tog razloga raspon vrijednosti koji se može zapisati u varijablu ovog tipa podataka nije ograničen procesorom već radnom memorijom. Zaključak je da se kod ovog tipa podataka ne događa gubitak preciznosti.

```
a = 100000000000000000000
b = a + 1

print(b)
```

```
Izlaz:
      100000000000000000001
```

Realni brojevi

Realni brojevi predstavljaju se tipom podataka `float` (engl. *floating point number*), tj. broj s pomičnim zarezom. Ovi brojevi prikazuju se s decimalnim zarezom, a skup realnih brojeva označava se oznakom **R**.

Primjer 3: Korištenje funkcije imena `type()` nad varijablom u kojoj je spremljena realna vrijednost.

```
a = 10.55
print(type(a))
```

```
Izlaz:
      <class 'float'>
```

Za razliku od cijelih brojeva, tip podataka `float` u Pythonu nije uvijek precizan i može doći do odstupanja od točne vrijednosti. Razlog odstupanja je što se ovaj tip podataka pohranjuje u memoriji u konačnom prostoru. Ovi brojevi zapisuju se u memoriju računala koristeći standard IEEE 754-1985[3] s pomičnim zarezom i on ne može točno prikazati sve decimalne vrijednosti. Zaključak: kod ovog tipa podataka može se dogoditi gubitak preciznosti. Ako je potrebna velika preciznost u radu s decimalnim brojevima, preporučuje se koristiti modul `decimal`. Taj modul dio je standardne instalacije Pythona i tip podataka definiran u njemu omogućava korištenje realnih brojeva bez gubitka preciznosti.

Primjer 4: Gubitak preciznosti.

```
a = 0.1

b = 0.18
c = a + b
print(c)

b = 0.19
c = a + b
print(c)

b = 0.20
c = a + b
print(c)

b = 0.21
c = a + b
print(c)
```

```
Izlaz:
      0.28
      0.29000000000000004
      0.30000000000000004
      0.31
```

Kompleksni brojevi

Kompleksni brojevi definirani su tipom podataka `complex`. Kompleksni brojevi sastoje se od realnog i imaginarnog dijela, a svaki od njih prikazan je jednim decimalnim brojem. Kako bi se dohvatio realni i/ili imaginarni dio iz kompleksnog broja spremljenog u varijabli `z` u donjem primjeru, mogu se koristiti podatkovni članovi `real` i `imag`.

Primjer 5: Korištenje funkcije `complex()` za rad s kompleksnim brojevima.

```
z = complex(1.11, 2.22)

print(z)
print(z.real)
print(z.imag)
print(type(z))
```

```
Izlaz:
      (1.11+2.22j)
      1.11
      2.22
      <class 'complex'>
```

3.9. Nizovi znakova

Za prikaz teksta koristi se poseban tip podataka koji se zove znakovni niz te se predstavlja tipom podataka `str` (engl. *string*). Niz znakova može se zapisati korištenjem jednostrukih ili dvostrukih navodnika.

Navodnici u C-u

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

Jednostruki navodnici

Omogućavaju uključivanje dvostrukih navodnika u niz znakova.

Primjer 1: Pisanje niza znakova pomoću jednostrukih navodnika.

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

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

Dvostruki navodnici

Omogućavaju uključivanje jednostrukih navodnika u niz znakova.

Primjer 2: Pisanje niza znakova pomoću dvostrukih navodnika.

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

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

Python na jednak način tretira jednostruke i dvostruke navodnike, osim što unutar niza znakova omogućava korištenje navodnika druge vrste.

Tri uzastopna jednostruka ili dvostruka navodnika

Omogućavaju protezanje niza znakova kroz nekoliko linija. Svi uneseni razmaci uključeni su u niz znakova.

Primjer 3: Pisanje niza znakova pomoću triju uzastopnih jednostrukih ili dvostrukih navodnika.

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

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

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

Primjer 4: Korištenje funkcije `type()` ako je kao argument predan objekt u kojem se nalazi niz znakova.

```
nizZnakova = 'Novi niz znakova!'

print(type(nizZnakova))
```

```
Izlaz:
      <class 'str'>
```

Posebni znakovi

Posebni znakovi u nizovima podataka su znakovi koji imaju posebno značenje [3]. Oni se ne mogu prikazati pomoću običnih znakova. Najčešće korišteni posebni znakovi:

Posebni znak	Značenje
\n	Novi redak
\t	Horizontalni tablični pomak
\\	Silazna crta (engl. *backslash*)
\'	Jednostruki navodnik
\"	Dvostruki navodnik

Primjer 5: Korištenje znakova za novi redak i horizontalni tablični pomak.

```
# Novi redak
niz1 = "Prvi redak.\nDrugi redak.\nTreći redak."
print(niz1)

# Horizontalni tablični pomak i novi redak
niz2 = "10\t20\n30\t40"
print(niz2)
```

```
Izlaz:
      Prvi redak.
      Drugi redak.
      Treći redak.
      10    20
      30    40
```

Primjer 6: Korištenje znakova za silaznu crtu te jednostruki i dvostruki navodnik.

```
# Silazna crta
niz1 = 'Silazna crta: \\'
print(niz1)

# Jednostruki navodnik
niz2 = 'I\'m here'
print(niz2)

# Dvostruki navodnik
niz3 = "On je rekao: \"Idemo tamo!\""
print(niz3)
```

```
Izlaz:
      Silazna crta: \
      I'm here
      On je rekao: "Idemo tamo!"
```

Dohvaćanje vrijednosti unutar niza znakova

Kod nizova znakova svaki znak ima svoju poziciju, a pozicija na kojoj se neko slovo nalazi zove se **indeks**. Ako se želi dohvatiti prvi znak u nizu, na primjer u riječi „Zagreb“ želi se dohvatiti znak „Z“, to slovo u logici programiranja nema poziciju 1, već ima poziciju, tj. indeks, 0, a slovo „b“ nalazi se na indeksu 5.

Primjer 7: Ako se želi dohvatiti element na nekom točno određenom indeksu, to se radi tako da se nakon imena varijable u kojoj se nalazi niz znakova napišu uglate zagrade i u njima indeks pozicije s koje se želi dohvatiti znak.

```
nizZnakova = 'Zagreb'
print(nizZnakova[0])
print(nizZnakova[2])
```

```
Izlaz:
      Z
      g
```

Primjer 8: Prikaz konvencije preko koje se dohvaćaju elementi od jednog do drugog indeksa. Ako se želi dohvatiti znakove od, na primjer, 1. do 4. indeksa (slova „agre“ u riječi „Zagreb“), to će se napraviti tako da se napiše `nizZnakova[1:5]`. Sintaksa i detaljniji opis prikazani su u nastavku.

```
nizZnakova[start:stop]
```

- `start` – indeks od kojeg kreće ispisivanje
- `stop` – vrijednost za 1 broj veća od indeksa zadnjeg elementa koji se ispisuje

Moguće je izostaviti vrijednost `start` ili `stop`. U nastavku se nalazi opis što se dogodi ako se izostavi `start`, a što se dogodi ako se izostavi vrijednost `stop`.

- `start` – dohvaćanje znakova kreće od nultog indeksa
- `stop` – dohvaćanje znakova ide do kraja niza znakova.

```
nizZnakova = 'Zagreb'

print(nizZnakova[1:4])
print(nizZnakova[:3])
print(nizZnakova[2:])
```

```
Izlaz:
      agr
      Zag
      greb
```

Nadovezivanje i ponavljanje

Kao i kod brojeva, tako i kod nizova znakova postoje operatori `+` i `*`. Nadovezivanje (engl. *concatenate*) izvodi se operatorom `+`, dok se ponavljanje (engl. *repetition*) izvodi operatorom `*`.

Operator	Opis
+	Nadovezivanje
*	Ponavljanje

Primjer 9: Nadovezivanje i ponavljanje nizova znakova.

```
a = "abc"
b = "def"

c = a + b
d = a * 2

print(c)
print(d)
```

```
Izlaz:
      abcdef
      abcabc
```

Provjera članstva u nizu znakova

Za provjeru članstva nekog elementa u nizu znakova mogu se koristiti operator `in` i `not in`. Rezultat ovih operatora je logička vrijednost `True` ili `False`.

Operator	Opis
in	Provjerava nalazi li se dani znak u nizu
not in	Provjerava NE nalazi li se dani znak u nizu

Primjer 10: Provjera članstva u nizu znakova.

```
a = "abc"

print("a" in a)
print("a" not in a)
print("c" in a)
print("c" not in a)
```

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

Napomena – razmaci u kôdu

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

3.10. ASCII tablica

ASCII (engl. *American Standard Code for Information Interchange*) je standardizirana tablica koja prikazuje kôdove znakova. Budući da se znakovi kao takvi ne mogu pohraniti u radnoj memoriji (ili bilo kojoj drugoj memoriji), ova se tablica koristi za pretvaranje znakova u pripadne brojeve koje računalo može obrađivati. Na primjer, slovo `A` u ASCII tablici ima broj 65, što znači da se u računalnom sustavu može predstaviti binarnim brojem 01000001. ASCII tablica sastoji se od 128 znakova, a postoji i proširena ASCII tablica koja se sastoji od 256 znakova. Uz ASCII postoji i UNICODE, koji je noviji industrijski standard za kodiranje znakova. Standard UNICODE sadrži znakove gotovo svih pisama (latinično, kinesko, arapsko pismo itd.).

Primjer 1: Dekadske vrijednosti znakova `0`, `A`, `Z`, `a` koje su dobivene pozivom funkcije `ord()`.

```
print("0:", ord("0"))
print("A:", ord("A"))
print("a:", ord("Z"))
print("z:", ord("a"))
```

```
Izlaz:
      0: 48
      A: 65
      z: 90
      a: 97
```

Python omogućuje korištenje usporednih operatora nad znakovima. U nastavku se nalazi nekoliko primjera usporedbe. Usporedbe iz primjera u nastavku:

1. Je li slovo `a` (97) manje od slova `b` (98), rezultat je `True`
2. Je li slovo `a` (97) veće od slova `b` (98), rezultat je `False`
3. Je li slovo `a` (97) manje od slova `A` (65), rezultat `False`.

Primjer 2: Operatori usporedbe nad znakovima.

```
print("a" < "b")
print("a" > "b")
print("a" < "A")
```

```
Izlaz:
      True
      False
      False
```

3.11. Logičke vrijednosti

Logičke vrijednosti su vrijednosti koje se koriste kako bi se izrazila istinitost izraza (uvjeta) [5]. Postoje dvije logičke vrijednosti, a to su:

- Istina (engl. *true*) – `True`
- Laž (engl. *false*) – `False`.

Python za logičke vrijednosti ima definiran ugrađeni tip podataka `bool` (engl. *boolean*). Operatori usporedbe i logički operatori kao rezultat izraza daju tip podataka `bool`.

Primjer 1: Logičke vrijednosti.

```
print(0 < 1)
print(1 < 0)
a = True
print(type(a))
```

```
Izlaz:
      True
      False
      <class 'bool'>
```

Primjer 2: Ovaj tip podataka ujedno je i podtip cijelih brojeva te se istina prilikom aritmetičkih operacija `True` ponaša kao 1, a `False` kao 0.

```
print(True + 0)
print(True + 1)
print(False + 0)
print(False + 1)
```

```
Izlaz:
      1
      2
      0
      1
```

3.12. Ugrađene funkcije za rad s brojevima

U ovom poglavlju bit će opisane najčešće korištene funkcije za rad s brojevima. Za korištenje ovih funkcija nije potrebno uključivati dodatne module, npr. modul `math`.

Naziv	Opis
`int(x)`	Pretvara vrijednost `x` u cijeli broj
`float(x)`	Pretvara vrijednost `x` u decimalni broj
`abs(x)`	Apsolutna vrijednost broja `x`
`max()`	Vraća najveći predani broj
`min()`	Vraća najmanji predani broj

Primjer 1: Ugrađene funkcije za rad s brojevima.

```
print(int(5.5))
print(float("5.55"))
print(abs(-5.55))
print(max(-5.55, -1, 5.55, 10))
print(min(-5.55, -1, 5.55, 10))
```

```
Izlaz:
      5
      5.55
      5.55
      10
      -5.55
```

3.13. Ugrađene funkcije i metode za rad s nizovima znakova

U ovom poglavlju bit će objašnjene najčešće korištene funkcije i metode za rad s nizovima znakova.

Često korištena funkcija za rad s nizovima znakova je funkcija koja vraća duljinu niza znakova, a također se često koristi i funkcija koja prima neku vrijednost, npr. cjelobrojnu vrijednost, i kao rezultat vraća niz znakova te cjelobrojne vrijednosti, na primjer, pretvorba iz tipa podataka `int` (ili nekog drugog tipa podataka) u tip podataka `str`.

Naziv	Opis
`len(niz)`	Vraća duljinu niza znakova
`str()`	Prima objekt i vraća niz znakova.

Primjer 1: U varijablu imena `a` spremljen je niz znakova `Zagreb`, a funkcija `len()` izračunava duljinu niza znakova spremljenog u varijablu `a` te kao povratnu vrijednost vraća cijeli broj.

```
a = "Zagreb"
n = len(a)
print(n)
```

```
Izlaz:
      6
```

Primjer 2: U varijablu `a` pohranjena je cjelobrojna vrijednost 10, a potom se pozivom funkcije `str()` ta cjelobrojna vrijednost tipa podataka `int` pretvara u tip podataka `str`.

```
a = 10
niz = str(a)
print(type(niz))
print(niz)
```

```
Izlaz:
      <class 'str'>
      10
```

Razlika u načinu pozivanja između funkcija i metoda

Kako bismo mogli obraditi metode za rad s nizovima znakova, u nastavku će biti objašnjena razlika između poziva funkcije i metode.

- 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 poziva preko svojeg imena, no ona je povezana s objektom na temelju kojega se metoda poziva, prijenos vrijednosti se vrši implicitno. 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()
  ```

Velika i mala slova

Metode u narednoj tablici služe za manipulaciju veličine slova unutar znakovnih nizova.

Naziv	Opis
`capitalize()`	U danom nizu prvo slovo stavlja u veliko slovo, a sva ostala slova u mala.
`title()`	Sve riječi počinju velikim slovom, a ostatak riječi napisan je malim slovima.
`lower()`	Pretvara cijeli niz znakova u mala slova.
`upper()`	Pretvara cijeli niz znakova u velika slova.

Primjer 3: Metode za manipulaciju veličinom slova.

```
nizZnakova = "hello WORLD"

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

```
Izlaz:
      Hello world
      Hello World
      hello world
      HELLO WORLD
```

Uklanjanje praznina s početka i kraja niza znakova

Metode u narednoj tablici služe za uklanjanje praznina koje se nalaze na početku i/ili kraju niza znakova nad kojim se ove metode pozivaju.

Naziv	Opis
`lstrip()`	Uklanja sve praznine s lijeve strane niza.
`rstrip()`	Uklanja sve praznine s desne strane niza.
`strip()`	Uklanja sve praznine s desne i lijeve strane niza.

Primjer 4: Prilikom inicijalizacije varijable imena `nizZnakova` praznine su radi jasnoće markirane sivom bojom (unesene su po 4 praznine). Također, u izlazu samog programa sve praznine koje se ispisuju markirane su sivom bojom.

```
nizZnakova = "    hello WORLD    "
print(nizZnakova)
print(nizZnakova.lstrip())
print(nizZnakova.rstrip())
print(nizZnakova.strip())
```

```
Izlaz:
      ␣␣␣␣hello WORLD␣␣␣␣
      hello WORLD␣␣␣␣
      ␣␣␣␣hello WORLD
      hello WORLD
```

3.14. Vježba: Brojevi, nizovi znakova i logičke vrijednosti

1. Spremite niz znakova u varijablu i ispišite taj niz: `I'm from Croatia!`
2. Iz niza znakova u prethodnom zadatku dohvatite i ispišite riječ `Croatia`.
3. 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).
4. Spremite u tri varijable brojčane vrijednosti (cjelobrojnu, realnu i cjelobrojnu negativnu) te pomoću njih isprobajte funkcije za rad s brojevima (`int()`, `float()`, `abs()`, `max()`, `min()`).
5. Kreirajte varijablu te u nju spremite proizvoljni niz znakova. Tu varijablu upotrijebite kako biste isprobali korištenje funkcija i metoda za rad s nizovima znakova (`len()`, `capitalize()`, `title()`, `lower()`, `upper()`, `lstrip()`, `rstrip()`, `strip()`).

Napomena

* Zadaci označeni zvjezdicom ne rješavaju se na predavanjima. Ovako označeni zadaci namijenjeni su za samostalnu vježbu.

6. * U varijablu upišite 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`).

3.15. Pitanja za ponavljanje: Osnove programskog jezika Python

1. Koje sve vrste komentara postoje?
2. Može li se ključna riječ koristiti kao ime varijable?
3. Može li doći do gubitka preciznosti kod cijelih brojeva?
4. Može li doći do gubitka preciznosti kod realnih brojeva?
5. Je li potrebno prenijeti argumente `sep` i `end` u pozivu funkcije `print()`?
6. Ako se u pozivu funkcije `print()` ne prenese argument `sep`, koja će se vrijednost ispisati između dvaju objekata?
7. Ako je poziv funkcije `print()` oblika: `print("Hello World!")`, hoće li se automatikom ispisati predefinirana vrijednost parametra `sep` ili `end`?
8. Koji je skraćeni oblik aritmetičkog operatora ako neku varijablu želimo uvećati za neki broj?
9. Je li u Pythonu kod navođenja varijable potrebno eksplicitno navesti kojeg je tipa neka varijabla, na primjer: *`int, float, str`*?
10. Kako se može ispitati parnost cijelog broja?

4. Upravljanje tokom programa

Trajanje poglavlja:

90 min

Po završetku ovog poglavlja polaznik će moći:

- kontrolirati tok programa odlukama `if`, `elif`, `else`
- kontrolirati tok programa petljama `while`, `for`.

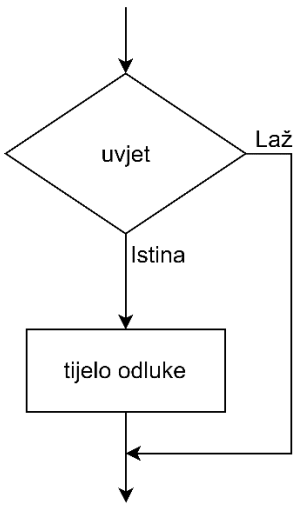
Jedan od najvažnijih segmenata programiranja jest kontrola toka programa. Kontrola toka programa omogućava da odlučujemo na koji se način program ponaša u zadanim okolnostima, tj. u zadacima koji se moraju odraditi ovisno o ulaznim vrijednostima.

U Pythonu postoje dva osnovna elementa za kontrolu toka, a to su odluke i petlje. Odluke nam pomažu odlučivati koji je dio programskoga kôda potrebno izvršiti, a koji dio programskoga kôda je za ulazne vrijednosti nebitan. Petlje olakšaju ponavljanje akcija, 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 te sam program ne bi mogao biti univerzalan za bilo koju unesenu vrijednost. Bolji način je da se ispis odradi pomoću petlje čija će početna vrijednost biti 1, završna vrijednost 1000, a faktor uvećanja 1. Na ovaj se način s nekoliko linija programskoga kôda može ispisati 1000 brojeva, a prema potrebi količina ispisanih brojeva može se jednostavno povećati ili smanjiti.

4.1. Uvjetno izvođenje

Uvjetna naredba `if`

Uvjetna 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`, uvjeta i dvotočke, dok se tijelo odluke sastoji od jedne ili više uvučenih naredbi koje rade neku korisnu akciju. U nastavku slijedi dijagram toka.



Opis dijagrama: ako je uvjet istinit (`True`), izvršava se tijelo odluke, no ako izraz nije zadovoljen, tijelo odluke neće se izvršiti. Sintaksa `if` odluke može se vidjeti u nastavku.

Sintaksa zahtijeva da se napiše ključna riječ `if`, a nakon toga logički izraz i potom dvotočka koja interpreteru označava da slijedi tijelo odluke. Sve linije tijela odluke moraju biti uvučene jedan korak udesno od zaglavlja.

```
if <uvjet>:
        <tijelo_odluke>
```

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

Naredbe u tijelu odluke uvijek moraju biti uvučene za isti broj mjesta udesno. Samo uvlačenje postiže se ili korištenjem razmaka ili pak korištenjem tabulatora. Bitno je naglasiti da miješanje razmaka i tabulatora nije dozvoljeno.

Primjer 1: Rezultat logičkog izraza naredbe `if` je `True`.

```
a = 5

if a == 5:
    print("Istinit uvjet!")
    print(5)

print("Kraj!")
```

```
Izlaz:
      Istinit uvjet!
      5
      Kraj!
```

Primjer 2: Rezultat logičkog izraza naredbe `if` je `False`.

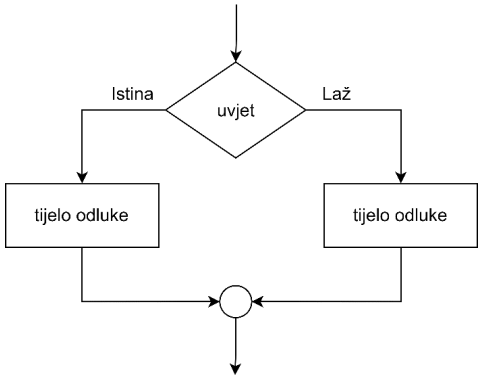
```
a = 0

if a == 5:
    print("Istinit uvjet!")
    print(5)

print("Kraj!")
```

```
Izlaz:
      Kraj!
```

Uvjetna naredba `if-else`

Samom naredbom `if` moguće je napraviti puno stvari, no kako bi se programerima što više olakšao posao, uvedena je i naredba `else`, koja

zajedno s naredbom `if` čini odluku `if-else`. Kod ovakvog tipa odluke uvijek će biti izvršen jedan dio. Ako je rezultat uvjeta istina (`True`), izvršava se tijelo odluke za istinu, a ako je rezultat uvjeta laž (`False`), izvršava se tijelo odluke za laž.



Sintaksa zahtijeva da se napiše ključna riječ `if`, a nakon toga logički izraz i potom dvotočka koja interpreteru označava da slijedi tijelo odluke. Sve linije tijela odluke moraju biti uvučene jedan korak udesno od zaglavlja. Alternativni stavak počinje ključnom riječi `else`, potom ide dvotočka te sve linije alternativnog stavka opet moraju biti uvučene jedan korak udesno.

```
if <uvjet>:
        <tijelo_odluke>
else:
        <tijelo_odluke>
```

Primjer 3: Testiranje parnosti broja u varijabli `a`. Rezultat logičkog izraza naredbe `if` je `False`.

```
a = 1

if a % 2 == 0:
    print("Broj je paran!")
else:
    print("Broj je neparan!")
```

```
Izlaz:
      Broj je neparan!
```

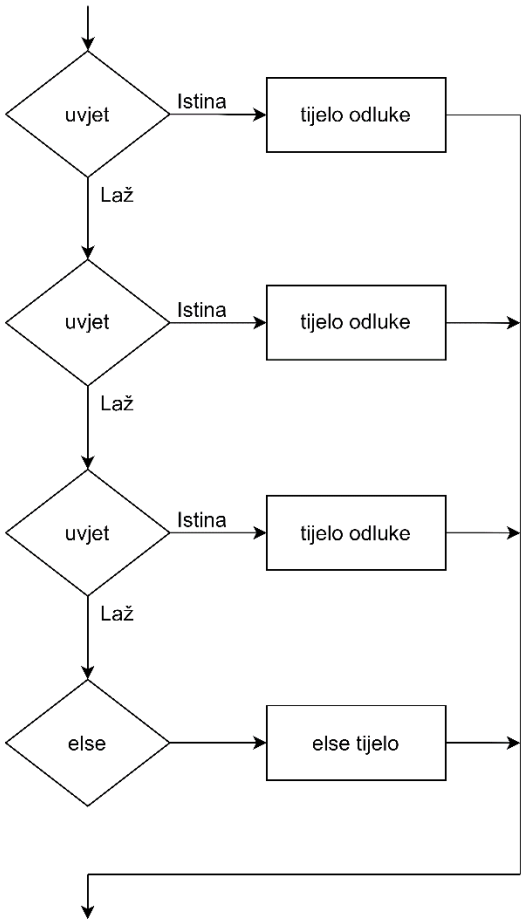
Primjer 4: Testiranje parnosti broja u varijabli `a`. Rezultat logičkog izraza naredbe `if` je `True`.

```
a = 2

if a % 2 == 0:
    print("Broj je paran!")
else:
    print("Broj je neparan!")
```

```
Izlaz:
      Broj je paran!
```

Uvjetna naredba `if-elif-else`

Naredbama `if` i `if-else` pridodaje se još i uvjet `elif` te tako dobivamo `if-elif-else`. Naredba `elif` služi za „neograničen“ broj uvjeta, tj. provjera. U nastavku se nalazi dijagram toka koji prikazuje kako se koriste višestruki uvjetni stavci.



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

```
if <uvjet>:
        <tijelo_odluke>
elif <uvjet>:
        <tijelo_odluke>
elif <uvjet>:
        <tijelo_odluke>
else:
        <else_tijelo>
```

Primjer 5: Zadovoljen je uvjet u bloku `if`.

```
a = -5

if a < 0:
    print("Broj je negativan.")
elif a == 0:
    print("Broj je nula.")
elif a > 0 and a < 100:
    print("Broj je veći od 0 i manji od 100.")
else:
    print("Broj je veći od 100.")
```

```
Izlaz:
      Broj je negativan.
```

Primjer 6: Zadovoljen je uvjet u bloku `elif`.

```
a = 50

if a < 0:
    print("Broj je negativan.")
elif a == 0:
    print("Broj je nula.")
elif a > 0 and a < 100:
    print("Broj je veći od 0 i manji od 100.")
else:
    print("Broj je veći od 100.")
```

```
Izlaz:
      Broj je veći od 0 i manji od 100.
```

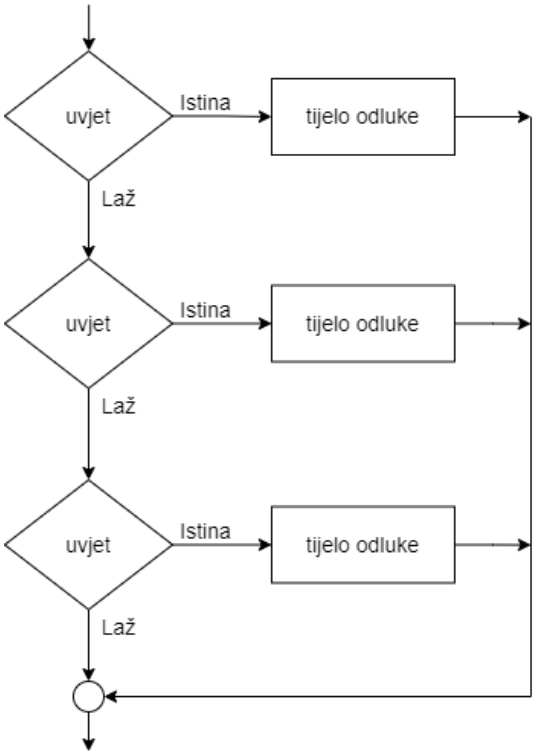
Primjer 7: Izvršava se tijelo bloka `else`.

```
a = 500

if a < 0:
    print("Broj je negativan.")
elif a == 0:
    print("Broj je nula.")
elif a > 0 and a < 100:
    print("Broj je veći od 0 i manji od 100.")
else:
    print("Broj je veći od 100.")
```

```
Izlaz:
      Broj je veći od 100.
```

Uvjetna naredba `if-elif-elif`

Također je moguć slučaj da se blok `else` izostavi. U tom slučaju nastavit će se izvršavati programski kôd koji se nalazi u nastavku bloka `if-elif-elif-...`



Primjer 8: Nijedan od logičkih uvjeta nije istinit, no kako nema bloka `else`, program će nastaviti izvršavati programski kôd nakon bloka `if-elif-elif`.

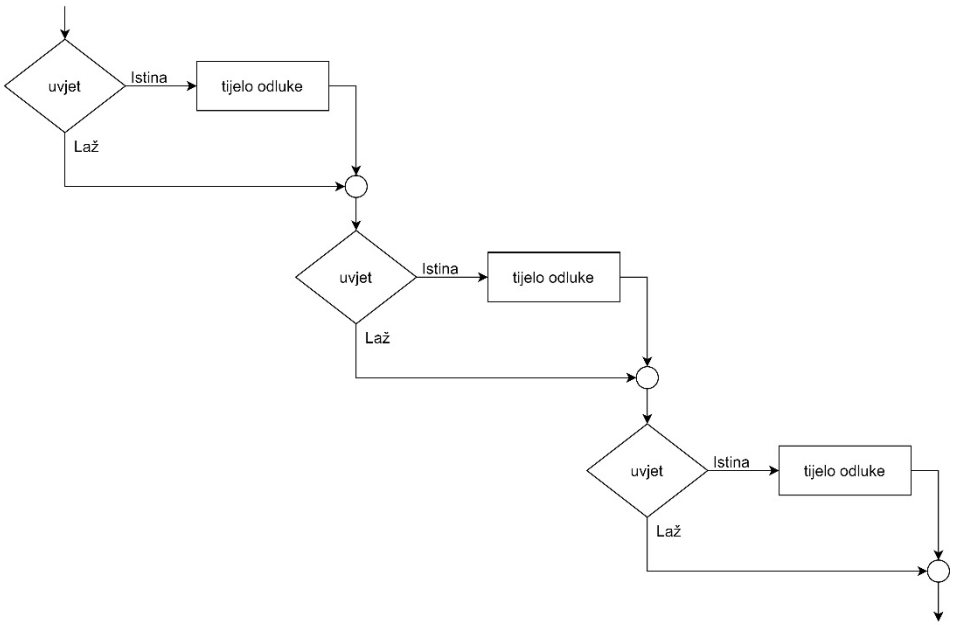
```
a = 500

if a < 0:
    print("Broj je negativan.")
elif a == 0:
    print("Broj je nula.")
elif a > 0 and a < 100:
    print("Broj je veći od 0 i manji od 100.")

print("Kraj!")
```

```
Izlaz:
      Kraj
```

Uvjetna naredba `if-if-if-...`

Niz naredbi `if` međusobno nije ovisan, tj. svaka naredba `if` zasebno testira svoj uvjet i ulazak u tijelo jednog segmenta neće preskočiti sve ostale naredbe `if`.

```
if <uvjet>:
        <tijelo_odluke>
if <uvjet>:
        <tijelo_odluke>
if <uvjet>:
        <tijelo_odluke>
```



Primjer 9: Svaki od logičkih uvjeta rezultirati će istinom, a budući da niz naredbi `if` međusobno nije ovisan, izvršit će se svi segmenti.

```
a = 75

if a > 0:
    print("Broj je veći od 0.")

if a > 0 and a < 100:
    print("Broj je veći od 0 i manji od 100.")

if a > 50 and a < 150:
    print("Broj je veći od 50 i manji od 150.")
```

```
Izlaz:
      Broj je veći od 0.
      Broj je veći od 0 i manji od 100.
      Broj je veći od 50 i manji od 150.
```

4.2. Vježba: Uvjetno izvođenje

Napomena

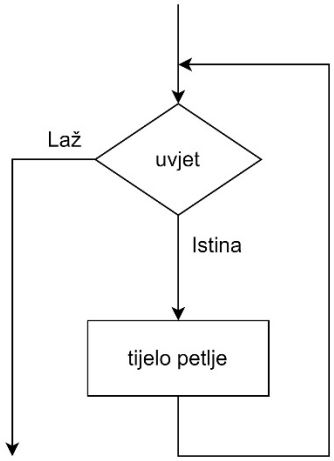
$\pi = 3.1415$

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, $V = \frac{4}{3} \times r^3 \times \pi$. U suprotnom 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, 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. Razmislite je li ostao još neki slučaj koji nije obrađen i, ako jest, obradite ga na pripadajući način.
3. U varijable `a` i `b` spremite dva broja. Ako je vrijednost varijable `a` barem za 50 veća od vrijednosti varijable `b`, a uz to je vrijednost varijable `b` parna, 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], a 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, ispišite poruku „Zadovoljava.“, a 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.“.

4.3. Petlja `while`

Ako se želi neki dio programskog kôda ponoviti određeni broj puta (bez kopiranja linija programskog kôda), to je moguće napraviti na elegantan način koristeći programske petlje. U ovom poglavlju bit će obrađena petlja `while`. Ova petlja prije izvršavanja programskog kôda koji joj pripada (tijelo petlje) ispituje je li uvjet istinit. Ako je rezultat logičkog uvjeta `True`, izvršit će se tijelo petlje, dok se u suprotnom (rezultat logičkog uvjeta je `False`) tijelo petlje preskače i izvršava se programski kôd napisan ispod petlje. Sintaksa petlje `while` sastoji se od ključne riječi `while`, a slijedi logički izraz te dvotočka koja interpreteru označava da nakon nje slijedi tijelo petlje.

```
while <uvjet>:
        <tijelo_petlje>
```



Primjer 1: Programski kôd koji ispisuje brojeve od 1 do 9, bez petlje.

```
print("1")
print("2")
print("3")
print("4")
print("5")
print("6")
print("7")
print("8")
print("9")
```

```
Izlaz:
      1
      2
      3
      4
      5
      6
      7
      8
      9
```

Promjenom zahtjeva da se moraju ispisati svi brojevi od 1 do 20 programer bi morao napisati dodatnih 11 linija programskog kôda koji su u suštini identični, jedino se razlikuju u vrijednosti koju ispisuju. Zaključak koji se nameće je da je ovakav način neefikasan i u sljedećem primjeru on će biti zamijenjen implementacijom pomoću petlje `while`.

Primjer 2: Uvjet izvođenja petlje u svakom koraku ispituje je li zadovoljen logički izraz `i < 10`, toliko dugo dok je logički izraz zadovoljen tijelo petlje `while` će se izvršavati. Varijabla `i` je prije početka izvođenja petlje postavljena na 1 i njena vrijednost se u svakom koraku petlje `while` ispisuje i potom uvećava za 1.

```
i = 1

while i < 10:
    print(i)
    i += 1
```

```
Izlaz:
      1
      2
      3
      4
      5
      6
      7
      8
      9
```

Beskonačna petlja: potrebno je obratiti pozornost da se ne izazove beskonačna petlja. Beskonačna petlja će se izazvati ako uvjet provjere, tj. logički izraz, zauvijek bude istinit.

Primjer 3: U nastavku slijedi prepravljeni primjer prethodnog programskog kôda gdje je izostavljena linija `i += 1` čime se izazvalo to da se vrijednost 1 ispisuje beskonačno puta. Ovaj se program nikad neće završiti, osim ako ga prekinemo pritiskom kombinacije tipki Ctrl i C.

```
i = 1

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

```
Izlaz:
      1
      1
      1
      1
      1
      1
      1
      1
      1
      # Prekid pomoću kombinacije tipki Ctrl i c
```

4.4. Petlja `for`

U programskom jeziku Python petlja `for` koristi se na drugačiji način nego kod programskih jezika kao što su C, C++, Java i tako dalje. Kod Pythona petlja `for` nema početnu ni završnu vrijednost niti aritmetički faktor uvećanja. Petlja `for` u Pythonu iterira kroz elemente zadane sekvence (objekt po kojem petlja `for` radi iteraciju mora biti pobrojiv ili iterabilan). Zadana sekvenca može biti na primjer lista ili niz znakova. Sintaksa petlje `for` sastoji se od ključne riječi `for`, potom od upravljačke varijable proizvoljnog imena, ključne riječi `in` te pobrojivog objekta i dvotočke, dok je tijelo petlje `for` uvučeno u novoj liniji udesno.

```
for <upravljačka_varijabla> in <pobrojivi_objekt>:
        <tijelo_for_petlje>
```

Primjer 1: Prikazuje kako korištenjem petlje `for` možemo iterirati, tj. dohvaćati element po element iz varijable u koju je spremljen niz znakova. U svakom koraku u upravljačku varijablu imena `e` petlja `for` stavlja po jedan znak iz varijable imena `niz`.

```
niz = "Zagreb"

for e in niz:
    print(e)
```

```
Izlaz:
      Z
      a
      g
      r
      e
      b
```

Drugi primjer korištenja petlje `for` je za iteraciju po listi (ili nekom drugom pobrojivom objektu). Budući da liste dosad nisu obrađene, u nastavku će biti prikazano korištenje funkcije `range()`.

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)
```

kreira se lista od 10 elemenata. Prvi element poprima vrijednost 0, dok zadnji element poprima vrijednost 9. Tako kreirana 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. Tako kreirana lista izgleda ovako:

10	11	12	13	14

U slučaju da je druga vrijednost koju prima funkcija `range()` manja ili jednaka prvoj vrijednosti, rezultat će biti prazna lista. U tom slučaju petlja `for` neće imati elemente po kojima bi mogla iterirati.

```
range(10, 10)        range(10, 9)
```

Primjer 2: Lista koja je kreirana pozivom funkcije `range(1, 10)`. Riječi `for` i `in` su ključne riječi. Ova će se petlja okrenuti 9 puta, što je određeno funkcijom `range()`. Upravljačka varijabla `e` pri svakoj iteraciji poprima sljedeću vrijednost koja se nalazi u listi koju je funkcija `range()` kreirala te se tako poprimljena vrijednost ispisuje u tijelu petlje.

```
for e in range(1, 10):
    print(e)
```

```
Izlaz:
      1
      2
      3
      4
      5
      6
      7
      8
      9
```

Primjer 3: Lista koja je kreirana pozivom funkcije `range(10, 15)`.

```
for e in range(10, 15):
    print(e)
```

```
Izlaz:
      10
      11
      12
      13
      14
```

Primjer 4: Računanje 5! (pet faktorijela).

```
a = 5
rezultat = 1

for e in range(1, a + 1):
    rezultat *= e
print(rezultat)
```

```
Izlaz:
      120
```

Primjer 5: Određuje se 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, 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 e in range(2, n):
    if n % e == 0:
        jePrim = False

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

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

Primjer 6: Ako se želi ispisati lista prostih brojeva iz intervala [2, 10], najprije se pomoću petlje generira popis svih brojeva, neovisno o tome je li generirani broj prost ili nije. U nastavku slijedi primjer koji ispisuje sve brojeve iz intervala [2, 10].

```
for n in range(2, 11):
    print(n)
```

```
Izlaz:
      2
      3
      4
      5
      6
      7
      8
      9
      10
```

Primjer 7: Ako se žele ispisati samo prosti brojevi, potrebno je funkciju `print(n)` zamijeniti implementacijom koja radi detekciju je li neki broj prost. U nastavku slijedi primjer kôda koji ispisuje samo proste brojeve iz intervala [2, 10].

```
for n in range(2, 11):
    jePrim = True
    for e in range(2, n):
        if n % e == 0:
            jePrim = False

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

```
Izlaz:
      Broj 2 je prost!
      Broj 3 je prost!
      Broj 5 je prost!
      Broj 7 je prost!
```

U ovom primjeru vidi se da je petlje osim pojedinačnog korištenja moguće i grupirati, tj. unutar jedne petlje staviti drugu petlju (neovisno o kombinaciji petlje `for` ili `while`). Nema ograničenja razine grupiranja petlji unutar petlje.

4.5. Naredbe `break` i `continue`

Naredba `break`

Naredba `break` služi za prekid petlje `for` ili `while`. Kako petlje mogu biti jedna unutar druge, treba obratiti pozornost na to da naredba `break` prekida izvođenje isključivo petlje unutar koje je pozvana te se potom nastavlja izvođenje programskog kôda koji se nalazi ispod petlje iz koje smo izašli. Ova naredba često se koristi za prekidanje beskonačnih petlji. Upravo takav primjer prikazan je u nastavku.

Primjer 1: Uvjet petlje `while` postavljen je tako da zauvijek bude istinit, a potom se unutar tijela petlje `while` naredbom `if` testira rezultat logičkog izraza. Toliko dugo dok je rezultat logičkog izraza laž, ispisuje se vrijednost koja se nalazi u varijabli imena `e`, a onog trenutka kada rezultat logičkog izraza postane istina, ulazi se u tijelo `if` bloka. U tijelu `if` bloka najprije se ispisuje niz znakova predan funkciji `print()`, a potom se pozivom naredbe `break` prekida petlja. Funkcija `print()`, koja se nalazi nakon naredbe `break`, ne izvršava se, već se izvršava poziv funkcije `print()` koji se nalazi nakon petlje `for`.

```
e = 0
while True:
    if e == 5:
        print("Prije naredbe break!")
        break
        print("Nakon naredbe break!")
    print(e)
    e += 1

print("Kraj!")
```

```
Izlaz:
      0
      1
      2
      3
      4
      Prije naredbe break!
      Kraj!
```

Primjer 2: Poboljšani primjer programskog kôda koji određuje je li broj prost ili nije. Za razliku od primjera ovog istog zadatka iz prethodnog poglavlja, 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 petlja `for` u tijelu `if` sadržava ključnu riječ `break`. `break` nam u ovom slučaju omogućava da prekinemo izvršavanje unutarnje petlje `for`.

```
for n in range(2, 11):
    jePrim = True

    for e in range(2, n):
        if n % e == 0:
            jePrim = False
            break

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

```
Izlaz:
      Broj 2 je prost!
      Broj 3 je prost!
      Broj 5 je prost!
      Broj 7 je prost!
```

Naredba `continue`

Naredba `continue` koristi se za preskakanje linija programskog kôda koje se nalaze od njenog poziva pa do kraja tijela petlje unutar koje je naredba `continue` pozvana te se odlazi na sljedeću iteraciju petlje. Za razliku od naredbe `break`, naredba `continue` ne prekida izvođenje.

Primjer 3: Ovaj primjer ispisuje neparne brojeve u intervalu [1, 10], u uvjetu `if` testiramo je li broj paran i, ako je, pozivom naredbe `continue` preskaču se sve naredbe do kraja petlje i odlazi se na sljedeću iteraciju petlje.

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

```
Izlaz:
      1
      3
      5
      7
      9
```

Primjer 4: Identičnu funkcionalnost iz gornjeg primjera moguće je postići korištenjem samo naredbe `if`. Slijedi primjer koji pruža identičnu funkcionalnost kao gornji primjer s naredbom `continue`. Ovo su jednostavni slučajevi, pa korištenje naredbe `continue` ne dolazi do izražaja, ali u složenijim programima može biti jako korisna.

```
for i in range(1, 11):
    if i % 2 != 0:
        print(i)
```

```
Izlaz:
      1
      3
      5
      7
      9
```

4.6. Vježba: Petlje

1. Koristeći petlju `for` ispišite sve parne brojeve između 1 i 1000 koji su istovremeno djeljivi i s 5 i s 13.
2. Prethodni zadatak riješite pomoću petlje `while`.
3. Napišite program koji sadrži varijablu u kojoj je upisan proizvoljan niz znakova velikih i malih slova. Korištenjem petlje `for` prebrojite koliko ima velikih slova u nizu znakova te detektirajte postoji li u nizu znakova veliko slovo „A“. Rezultate ispišite na ekranu (broj velikih slova i informaciju postoji li veliko slovo „A“ u nizu).
4. Prethodni zadatak riješite pomoću petlje `while`.
5. * Napišite program koji sadrži varijablu u kojoj je upisan proizvoljni niz znakova i proizvoljna cjelobrojna vrijednost spremljena u varijablu `n`. Provjerite je li vrijednost varijable `n` strogo manja od broja znakova u nizu i u tom slučaju ispišite iz niza znakova svako `n`-to slovo. Na primjer, ulazni niz je „ABCDEFGH“, `n` je 2, a tada je izlaz „ACEG“. Ako je vrijednost varijable `n` veća ili jednaka od broja znakova u nizu, ispišite informaciju o grešci.
6. * Napišite program koji ispisuje koliko ima prostih brojeva između dvaju proizvoljnih brojeva.
7. * Napišite program koji će inicijalizirati varijablu `n` na proizvoljnu cjelobrojnu vrijednost. Vrijednost varijable `n` neka predstavlja red tablice. Ispišite 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 vrijednosti, tako da se nakon ispisa vrijednosti predane funkciji `print()` ne ispiše novi redak, treba koristiti sljedeću sintaksu: `print(vrijednost, end='')`.

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

Primjer 1.: T=(1, 1)

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

Primjer 2.: T=(4, 6)

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

Napomena

Uvjet provjere ako je u varijablu upisano malo slovo:

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

9. * Napišite program koji ispisuje sumu znamenaka nekog višeznamenkastog broja. Program mora biti napisan univerzalno za sve brojeve, a ne samo za npr. troznamenkaste. Na primjer, suma broja 159 iznosi 15. Primjer: 159%10 = 9, 159 // 10 = 15.

4.7. Pitanja za ponavljanje: Upravljanje tokom programa

1. Što omogućava uvjetno izvođenje?
2. Što omogućavaju petlje?
3. Kako funkcionira petlja `while`?
4. Kako funkcionira petlja `for`?
5. Što omogućuje naredba `break`?
6. Što omogućuje naredba `continue`?

5. Unos podataka s tipkovnice

Trajanje poglavlja:

15 min

Po završetku ovog poglavlja polaznik će moći:

- koristiti funkciju `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 zaslonu, dok ulazne funkcije, (na primjer, funkcija `input()`), omogućavaju unos podataka preko tipkovnice u sâm program.

U ovom poglavlju obrađuju se funkcija `input()`. Uz ulazno-izlazne funkcije `print()` i `input()` postoje i druge ulazno-izlazne funkcije, poput funkcija za čitanje i pisanje u datoteke. Takve funkcije bit će obrađene u poglavlju *8. Datoteke*.

5.1. 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, 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 tako da čita s tipkovnice znakove toliko dugo dok se ne pritisne tipka *Enter*. Nakon pritiska tipke *Enter* čitanje se završava te funkcija konvertira pročitane podatke u tip podataka *str*, tj. niz znakova, te tako učitane znakove vraća preko povratne vrijednosti funkcije.

Primjer 1: Učitavanje znakova s tipkovnice.

```
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 završenog čitanja znakova s tipkovnice funkcija `input()` vraća učitani niz znakova te se taj niz znakova sprema u varijablu `tekst`. U drugoj liniji funkcija `print()` ispisuje predane argumente. Funkciji `input()` možemo poslati jedan argument. Sadržaj tog argumenta ispisuje se prije nego što funkcija krene čitati znakove koji se unose s tipkovnice. Ovaj argument nije obavezan i on se može zamijeniti jednostavnim ispisom željenoga teksta pomoću funkcije `print()`.

Primjer 2: Prijenos argumenta funkciji `input()`.

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

```
Izlaz:
      Unesite tekst: Ogledna recenica!
      Unesen je tekst: Ogledna recenica!
```

U primjeru pozvana je funkcija `input("Unesite tekst: ")`, preneseni argument vrijednosti `"Unesite tekst: "` ispisuje se prije nego što se omogućava unos 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`.

Primjer 3: Kao što je na početku ovog poglavlja navedeno, povratni tip podataka funkcije `input()` je *string*, tj. niz znakova, neovisno o vizualnom prikazu unesenih brojeva.

```
text = input("Unesite broj: ")

print("Broj:", text)
print("Tip podataka:", type(text))
```

```
Izlaz:
      Unesite broj: 5
      Broj: 5
      Tip podataka: <class 'str'>
```

Primjer 4: Ako se želi baratati brojčanim vrijednostima kao što su cjelobrojne ili realne vrijednosti, potrebno je koristiti funkcije `int()` ili `float()` za konverziju niza znakova u željeni brojčani tip podataka. U nastavku je prikazan primjer gdje se unesena vrijednost preko tipkovnice pomoću funkcije `input()` odmah konvertira u brojčanu vrijednost pomoću funkcije `int()` ili `float()`.

```
broj1 = int(input("Unesite cijeli broj: "))
broj2 = float(input("Unesite realni broj: "))

print("Broj 1:", broj1)
print("Broj 2:", broj2)

print("Tip podataka 1:", type(broj1))
print("Tip podataka 2:", type(broj2))
```

```
Izlaz:
      Unesite cijeli broj: 5
      Unesite realni broj: 5.55
      Broj 1: 5
      Broj 2: 5.55
      Tip podataka 1: <class 'int'>
      Tip podataka 2: <class 'float'>
```

5.2. Vježba: Unos podataka s tipkovnice

1. Napišite program koji se sastoji od dviju varijabli. Vrijednosti tih varijabli učitajte preko tipkovnice te ih ispišite na ekranu. Prilikom upisivanja vrijednosti u varijable obavijestite korisnika da mora unijeti neku vrijednost, npr. „Unesite prvi niz znakova: “. To napravite bez korištenja funkcije `print()`.
2. 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, 2024]). Za ulazne podatke 20, 3, 2023, ispišite datum u sljedećem formatu:

   20. ožujka, 2023.
3. 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.
4. * Učitajte s tipkovnice cjelobrojnu vrijednost `n` iz intervala [3, 10]. Učitavanje treba ponavljati toliko dugo dok se ne učita ispravna vrijednost. U slučaju neispravne vrijednosti treba ispisati obavijest da je vrijednost neispravna. Treba 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
```

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*.

5.3. Pitanja za ponavljanje: Unos podataka s tipkovnice

1. Za što služi jedini mogući argument funkcije `input()`?
2. Na koji način možemo izbjeći slanje funkciji `input()` argument tipa `str`, da se dobije identična funkcionalnost kao da smo taj argument prenijeli?
3. Koji je povratni tip podataka funkcije `input()`?

6. Funkcije

Trajanje poglavlja:

75 min

Po završetku ovog poglavlja polaznik će moći:

- definirati i pozivati funkcije
- koristiti argumente funkcija
- koristiti naredbu `return`
- razlikovati lokalne i globalne varijable.

Funkcije

Funkcije se dijele na:

- ugrađene
- korisnički definirane.

U ovom poglavlju obrađuju se korisnički definirane 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 može se iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u nekom trenutku potrebna. Ovakav pristup omogućava nam 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 omogućuje nam 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 modula. Funkcije iz modula detaljnije se obrađuju u poglavlju *9. Moduli i paketi*.

6.1. Definicija funkcije

Kako bismo funkciju uopće mogli pozivati, ona mora biti definirana (implementirana). Funkcija se sastoji od zaglavlja funkcije i tijela funkcije. U nastavku slijedi primjer definicije funkcije:

```
def imeFunkcije(<argumenti>):
    <tijelo_funkcije>
```

Svaka funkcija koju želimo implementirati započinje ključnom riječju ***def*** nakon čega slijedi ime funkcije. Ime funkcije može biti proizvoljno, no treba pripaziti da se funkcija ne zove poput neke od ključnih riječi u Pythonu ili pak imenom neke druge funkcije ili varijable. Nakon imena funkcije obavezne su zagrade u koje se stavljaju potencijalni argumenti (u ovom potpoglavlju zagrade ostavljamo prazne), nakon zagrada navodi se dvotočka. Tijelo funkcije mora biti uvučeno jedan korak udesno.

Primjer 1: Implementacija funkcije koja radi ispis fraze „Prva funkcija!“. No, ako se pobliže promotri ispis, vidimo da tamo nema nikakvog ispisa.

Razlog je što ova funkcija nije nigdje pozvana. Poziv funkcije obrađuje se u sljedećem potpoglavlju.

```
def prvaFunkcija():
    print("Prva funkcija!")
```

```
Izlaz:
```

6.2. Poziv funkcije

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

Sintaksa poziva funkcije je:

```
imeFunkcije()
```

```
def imeFunkcije(<agumenti>):
    <tijelo_funkcije>


imeFunkcije()     # Ovo je poziv funkcije
```

Primjer 1: Poziv implementirane funkcije.

```
def prvaFunkcija():
    print("Prva funkcija!")


prvaFunkcija()
```

```
Izlaz:
      Prva funkcija!
```

Kao što se može vidjeti iz gornjeg primjera, najprije je definirana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izveden sâm poziv funkcije `prvaFunkcija()`. Bitno je primijetiti da definicija funkcije mora biti napisana ispred samog poziva funkcije.

Primjer 2: 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.

```
prvaFunkcija()


def prvaFunkcija():
    print("Prva funkcija!")
```

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

6.3. Argumenti funkcije

Da bismo funkcijama mogli slati različite vrijednost na temelju kojih će one raditi neku obradu, u zaglavlju funkcije u oblim zagradama upisuju se argumenti koje funkcija prima. Vrijednosti argumenata funkcije mogu se mijenjati ovisno o potrebi. U istom programu možemo nekoliko puta pozvati jednu te istu funkciju, recimo funkciju `kvadrat(x)`.

Primjer 1: Funkcija `kvadrat(x)` prima samo jedan argument i ovisno o vrijednosti toga argumenta ona izračunava kvadrat unesenoga broja. Prvi put zovemo funkciju `kvadrat(3)`, a drugi put `kvadrat(5)`. Primijetimo da je rezultat funkcije prilikom prvog poziva 9, a rezultat funkcije prilikom drugog poziva 25, tj. da rezultat ovisi o vrijednostima koje je funkcija primila.

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


kvadrat(3)
kvadrat(5)
```

```
Izlaz:
      9
      25
```

Formalni argumenti – koriste se isključivo unutar definicije funkcije.

Stvarni argumenti – koriste se isključivo u kôdu koji poziva funkciju.

Gore je prikazan primjer definicije funkcije `kvadrat(x)`. Iz priloženoga se vidi da funkcija prima jedan formalni argument imena `x`. U tijelu funkcije na temelju vrijednosti u formalnom argumentu `x` izračunava se i ispisuje kvadrat prenesene vrijednosti. Kako funkcija ima jedan formalni argument, prilikom poziva funkcije `kvadrat()` iz glavnog dijela programa potrebno je predati vrijednosti koje će se mapirati u varijablu `x` funkcije `kvadrat()`. Tako predana vrijednost zove se stvarni argument. Prilikom poziva funkcije vrijednosti se mogu predati direktno, a također ih je moguće predati preko varijabli.

Funkcije mogu primati i više argumenata.

Primjer 2: Ako funkcija mora vratiti sumu triju brojeva, a sve tri vrijednosti su funkciji predane kao argumenti, definicija funkcije izgleda ovako:

```
def suma(a, b, c):
    print(a + b + c)


suma(5, 10, 15)
```

```
Izlaz:
      30
```

U gornjem primjeru vidimo definiciju funkcije `suma()` s trima formalnim argumentima `a`, `b`, `c`. Nakon same definicije funkcije (zaglavlja i tijela funkcije) nalazi se poziv funkcije `suma(5, 10, 15)`. U pozivu prenosimo u funkciju vrijednosti 5, 10 i 15. Vrijednost 5 sprema se u formalni argument `a`, vrijednost 10 sprema se u formalni argument `b`, dok se vrijednost 20 sprema u formalni argument `c`.

Primjer 3: U ovom primjeru stvarni argumenti predani su preko varijabli `x`, `y` i `z`.

```
def suma(a, b, c):
    print(a + b + c)


x = 5
y = 10
z = 15
suma(x, y, z)
```

```
Izlaz:
      30
```

6.4. Predefinirani argumenti funkcije

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

Primjer 1: U nastavku je prikazan način pomoću kojega se postavlja argument funkcije na unaprijed zadanu vrijednost. U donjem slučaju ta unaprijed zadana vrijednost pridružena je formalnom argumentu `c` na vrijednost 0. Formalni argument `c` će vrijednost 0 poprimiti samo ako se prilikom poziva funkcije prenesu 2 stvarna argumenta, a ne sva 3 stvarna argumenta, koliko ih maksimalno može biti. Ako se prenesu 3 stvarna argumenta, formalni argument `c` poprima vrijednost trećega prenesenog stvarnog argumenta.

```
def suma(a, b, c = 0):
    print(a + b + c)


suma(5, 10)
```

```
Izlaz:
      15
```

U gornjem primjeru prilikom poziva funkcije `suma()` prenesene su vrijednosti 5 i 10, tj. prenesena su samo 2 stvarna argumenta. U ovom slučaju varijabla `c` poprima predefiniranu vrijednost, a to je vrijednost 0. Varijable `a` i `b` poprimaju vrijednost iz poziva funkcije – 5 i 10.

Primjer 2: U ovom primjeru prenose se 3 stvarna argumenta. Funkcija `suma()` ima 3 formalna argumenta te iz tog razloga formalni argument `c` neće poprimiti predefiniranu vrijednost 0, već će poprimiti vrijednost trećega prenesenog stvarnog argumenta iz poziva funkcije `suma()`, tj. vrijednost 15.

```
def suma(a, b, c = 0):
    print(a + b + c)


suma(5, 10, 15)
```

```
Izlaz:
      30
```

Primjer 3: Argumenti s predefiniranim vrijednostima moraju uvijek biti navedeni na kraju liste argumenata, u suprotnom će Python prijaviti grešku.

```
def suma(a, b=0, c, d=0):
    print(a + b + c + d)


suma(5, 10, 15)
```

```
Izlaz:
      SyntaxError: non-default argument follows default
      argument
```

Primjer 4: Ako postoji više predefiniranih argumenata i želi se promijeniti predefinirani argument koji nije prvi u toj listi (u donjem primjeru vrijednost prvog i drugog predefiniranog argumenta ostaje nepromijenjena), to se radi tako da se prilikom poziva funkcije eksplicitno navede kojem se formalnom argumentu pridružuje vrijednost.

```
def suma(a, b=0, c=0, d=0):
    print(a + 2 * b + 3 * c + 4 * d)


suma(1, d=4)
```

```
Izlaz:
      17
```

6.5. Naredba `return`

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

Primjer 1: Funkcija `kvadriranje()` 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 `a`, a kod drugog poziva funkcije povratna vrijednost sprema se u varijablu `b`.

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

a = kvadriranje(3)
b = kvadriranje(4)
print(a + b)
```

```
Izlaz:
      25
```

Primjer 2: U jednoj liniji poziva se funkcija za kvadriranje i nakon dobivanja povratne vrijednosti ista se pomoću funkcije `print()` odmah i ispisuje.

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

print(kvadriranje(3))
```

```
Izlaz:
      9
```

Primjer 4: Naredba `return` može se pozvati na način da se njoj zdesna ne nalaze nikakvi izrazi, varijable ili pak vrijednosti koje mora vratiti. Na ovakav način povratna vrijednost bit će `None`.

```
def fZaTest():
    print("Test!")
    return

print(fZaTest())
```

```
Izlaz:
      Test!
      None
```

Primjer 5: U jednom programu može biti više poziva naredbe `return`. Sve naredbe koje se nalaze nakon naredbe `return` bit će zanemarene.

```
def punoljetnost(god):
    if god < 18:
        return "Maloljetan!"
    return "Punoljetan!"

print(punoljetnost(15))
print(punoljetnost(19))
```

```
Izlaz:
      Maloljetan!
      Punoljetan!
```

6.6. Lokalne i globalne varijable

Vidljivost varijable je pojam koji određuje iz kojih dijelova programa je neka varijabla dohvatljiva, tj. dostupna. Varijable mogu biti globalne ili lokalne te će u nastavku biti pobliže objašnjene.

6.6.1. Globalna varijabla

Globalne varijable su varijable koje su definirane izvan funkcija.

Vidljivost globalnih varijabli – globalnim varijablama moguće je pristupati iz bilo kojeg dijela programa, a također i iz funkcija.

Životni vijek globalnih varijabli – ovim varijablama moguće je pristupati sve do završetka programa unutar kojeg su definirane.

Primjer 1: U programu su definirane dvije globalne varijable imena `a` i `b`, pridružene su im vrijednosti 10 i 20. Nakon toga u sljedećoj liniji pozvana je funkcija `test()`. Funkcija `test()` ispisuje sadržaj varijable `a` i `b`, no primijetimo da funkcija nema ni formalnih argumenata ni varijabli koje bi bile definirane u njoj, a da svejedno uspijeva ispisati vrijednost varijable `a` i `b`. Razlog tome je što su varijable `a` i `b` globalne varijable, koje su vidljive, tj. dohvatljive, iz svih dijelova programa (što uključuje i funkcije).

```
def test():
    print("test(), varijabla a: ", a)   # globalna
    print("test(), varijabla b: ", b)   # globalna


a = 10   # globalna
b = 20   # globalna

test()

print("varijabla a: ", a)   # globalna
print("varijabla b: ", b)   # globalna
```

```
Izlaz:
      10
      10
```

6.6.2. Lokalna varijabla

Lokalne varijable su varijable koje su definirane unutar funkcija. U ovu kategoriju svrstavaju se i formalni argumenti funkcija.

Vidljivost lokalnih varijabli – lokalne varijable vidljive su samo u tijelu funkcije gdje su definirane. Definicijom se smatra izraz pridruživanja vrijednosti varijabli, na primjer, `a = 10`.

Životni vijek lokalnih varijabli – formalni argumenti funkcije ili pak varijable definirane unutar funkcije, onog trenutka kada izvođenje funkcije završi (bilo pomoću naredbe `return`, bilo završetkom tijela funkcije) bit će trajno uništene. Što znači da sljedeći poziv iste funkcije više ne može vidjeti vrijednost varijable iz prethodnog poziva.

Primjer 1: U glavnom dijelu programa pozvana je funkcija `test()`, unutar funkcije `test()` definirana je varijabla `a` s vrijednošću 10. Nakon izlaska iz funkcije pokušava se dohvatiti vrijednost varijable `a`, no varijabla `a` nakon izlaska iz funkcije više ne postoji jer je ona lokalna varijabla koja je uništena nakon završetka funkcije `test()`. Njoj je bilo moguće pristupati isključivo unutar funkcije.

```
def test():
    a = 10


test()
print(a)
```

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

Primjer 2: Varijable `a` i `b` iz glavnog dijela programa su globalne varijable i one se nalaze u prostoru imena za globalne varijable. Unutar funkcije `test()` definira se nova varijabla istog imena `a`, no ta varijabla je lokalna varijabla jer je definirana unutar funkcije, dok je varijabla `b` globalna varijabla.

```
def test():
    a = 20  # lokalna
    print("test(), varijabla a: ", a)  # lokalna
    print("test(), varijabla b: ", b)  # globalna


a = 10  # globalna
b = 20  # globalna

test()

print("varijabla a: ", a)  # globalna
print("varijabla b: ", b)  # globalna
```

```
Izlaz:
      test(), varijabla a:  20
      test(), varijabla b:  20
      varijabla a:  10
      varijabla b:  20
```

Primjer 2: Ako je globalnim varijablama unutar funkcije potrebno mijenjati vrijednosti, to omogućava naredba `global`.

```
def test():
    global a  # globalna
    a = 20  # globalna
    print("test(), varijabla a: ", a)  # globalna
    print("test(), varijabla b: ", b)  # globalna


a = 10  # globalna
b = 20  # globalna
test()
print("varijabla a: ", a)  # globalna
print("varijabla b: ", b)  # globalna
```

```
Izlaz:
      test(), varijabla a:  20
      test(), varijabla b:  20
      varijabla a:  20
      varijabla b:  20
```

6.7. Vježba: Funkcije

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 parametra. Funkcija mora ispisati rezultat matematičke formule:

   ((a*a) + (b*c) – d) / 2

   U glavnom dijelu programa pozovite napisanu funkciju.
3. Prethodni zadatak prepravite na način da su formalni argumenti `b`, `c` i `d` predefinirani, a vrijednosti im odredite sami. Pozovite funkciju na način da joj šaljete jedan, dva, tri i četiri stvarna argumenta, a također isprobajte kako povezati stvarni i formalni argument na način da se ne prati poredak formalnih argumenata.
4. Napišite funkciju koja prima 2 parametra. Rezultat izračuna funkcije ovog se puta 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. Napišite funkciju `suma()` koja ne prima niti jedan parametar, ali mora izračunati i ispisati zbroj dvaju brojeva. Brojevi neka se dohvate iz glavnog dijela programa preko globalnih varijabli.
6. Prethodni zadatak prepravite na način da u glavnom dijelu programa definirate varijablu `rezultat`, pozovite funkciju `suma()` bez argumenata te bez korištenja naredbe `return`. Funkcija `suma()` mora globalnoj varijabli imena `rezultat` pridružiti sumu brojeva koji su dohvaćeni iz glavnog dijela programa preko globalnih varijabli.
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, 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! izračunava se 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)!`

6.8. Pitanja za ponavljanje: Funkcije

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 formalnih argumenata, je li prilikom poziva funkcije potrebno navesti svih 5 stvarnih argumenata ili postoji način da se neki stvarni argumenti izostave. Ako da, na koji način?
4. Kako se zove naredba pomoću koje funkcija vraća izračunatu vrijednost u pozivajući dio programa?
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?

7. Formatiranje ispisa podataka

Trajanje poglavlja:

90 min

Po završetku ovog poglavlja polaznik će moći:

- koristiti napredne mogućnosti za formatiranje ispisa podataka u željenom obliku
- koristiti sljedeće načine formatiranja: operator formatiranja `%`, formatiranje pomoću metode `format()` i pomoću f-stringova.

U dosadašnjem dijelu gradiva obrađena je funkcija `print()` i njeni parametri `sep` i `end`. Samo korištenje funkcije `print()` je izrazito jednostavno, no upravo zbog toga ta funkcija ima svoja ograničenja kada se žele postići složenija formatiranja.

Formatiranje ispisa ključno je za svaki program jer ono omogućava da rezultati budu vizualno pregledni i lakše čitljivi. Python nudi nekoliko različitih načina formatiranja ispisa. U ovom poglavlju obrađuju se sljedeći načini formatiranja ispisa podataka:

- operator `%`
- metoda `format()`
- f-strings.

7.1. Operator formatiranja `%`

Operator formatiranja `%` koristi se za formatiranje nizova znakova. Ovaj operator koristi se za umetanje vrijednosti u nizove znakova uz pomoć posebnih oznaka. Sintaksno gledajući ovakav način formatiranja nizova znakova sastoji se od dvaju dijelova. S lijeve strane operatora formatiranja `%` nalazi se formatirani niz, a s desne strane unutar oblih zagrada navode se vrijednosti koje će biti umetnute u formatirani niz znakova.

```
"formatiraniNiz" % (vrijednost1, vrijednost2, ...)
```

U formatiranom nizu znakova kontrola sadržaja obavlja se pomoću posebnih oznaka pri čemu svaka od njh ima svoje zasebno značenje.

Najčešće korištene oznake:

- `%s` – oznaka za znakovni niz
- `%d` – oznaka za cjelobrojne vrijednosti
- `%f` – oznaka za vrijednosti s pomičnim zarezom
- `%e` – oznaka za prikaz broja s eksponencijalom
- `%c` – oznaka za jedan znak
- `%g` – oznaka za automatski odabir najprikladnijeg načina ispisa broja.

Primjer 1: Korištenje oznake `%s` bez dodatnih mogućnosti.

```
zg = "Zagreb"
st = "Split"
ri = "Rijeka"
os = "Osijek"

print("%s %s %s %s" % (zg, st, ri, os))
print("---------------")
print("%s\n%s\n%s\n%s" % (zg, st, ri, os))
print("---------------")
print("Gradovi: %s > %s > %s > %s" % (zg,st,ri,os))
```

```
Izlaz:
      Zagreb Split Rijeka Osijek
      ---------------
      Zagreb
      Split
      Rijeka
      Osijek
      ---------------
      Gradovi: Zagreb > Split > Rijeka > Osijek
```

Primjer 2: Oznaku `%s` moguće je koristiti i s dodatnim mogućnostima, a to je određivanje duljine niza koji će se ispisati. Ova mogućnost obično se koristi kada se želi postići stupčasto formatiranje ispisa. Postoje dvije mogućnosti ispisa:

- Desno poravnanje: ako se navede samo brojčana vrijednost, na primjer `%20s`, ispisat će se cijeli niz znakova s desnim poravnanjem
- Lijevo poravnanje: za ovu vrstu poravnanja potrebno je koristiti dodatak `-`, na primjer: `%-20s`.

```
zg = "Zagreb"
st = "Split"
ri = "Rijeka"

i = 0
while i < 45:
    print(i % 10, end="")
    i += 1
print()

print("%15s%15s%15s" % (zg, st, ri))

print("%-15s%-15s%-15s" % (zg, st, ri))
```

```
Izlaz:
      012345678901234567890123456789012345678901234
                Zagreb          Split         Rijeka
      Zagreb          Split           Rijeka
```

Primjer 3: Korištenje oznake `%d`. Kao i kod prethodno objašnjene oznake, i kod `%d` moguće je koristiti opcije za kontrolu broja pozicija ispisa, dok se s opcijom `-` ostvaruje lijevo poravnanje.

```
a = 1
b = 22
c = 333
d = 4444
e = 55555

print("%d %d %d %d %d" % (a, b, c, d, e))

print("%-7d %-7d %-7d %-7d %-7d" % (a, b, c, d, e))

print("%7d %7d %7d %7d %7d" % (a, b, c, d, e))
print("%7d %7d %7d %7d %7d" % (e, d, c, b, a))
```

```
Izlaz:
      1 22 333 4444 55555
      1       22      333     4444    55555
            1       22      333     4444    55555
        55555     4444      333       22       1
```

Primjer 4: Korištenje oznake `%f`. Prilikom korištenja ove oznake, ako se ne zadaju drugačije vrijednosti, uvijek će se ispisati 6 decimala (pa makar sve bile 0). Za kontrolu broja ispisanih elemenata koristi se sintaksa sljedećeg oblika: `%10.2f`. Ovako zadana sintaksa označava da će se ukupno ispisati 10 elemenata, od kojih je u prvih 8 uključena cjelobrojna vrijednost kojoj prethode razmaci, nakon toga decimalna točka i jedna decimala. Vrijednost s lijeve strane točke također se može i izostaviti. Za kontrolu poravnanja, kao i u prethodnim primjerima, koristi se opcija `-` kojom se ostvaruje lijevo poravnanje (predefinirano poravnanje je desno).

```
a = 1.1
b = 22.22
c = 333.333
d = 4444.4444

print("%f_%f_%f_%f" % (a, b, c, d))

print("%.2f_%.2f_%.2f_%.2f" % (a, b, c, d))

print("%9.2f_%9.2f_%9.2f_%9.2f" % (a, b, c, d))
print("%-9.2f_%-9.2f_%-9.2f_%-9.2f" % (a, b, c, d))
```

```
Izlaz:
      1.100000_22.220000_333.333000_4444.444400
      1.10_22.22_333.33_4444.44
           1.10_    22.22_   333.33_  4444.44
      1.10      _22.22    _333.33   _4444.44
```

Primjer 5: Korištenje oznaka `%c`, `%e` i `%g`.

```
a = "Zagreb"
b = 0.000123
c = 0.0000123

print("Oznaka c: %c" % (a[0]))
print("Oznaka s: %s" % (a[0]))

print("Oznaka e: %e" % (b))
print("Oznaka g: %g" % (b))
print("Oznaka g: %g" % (c))
```

```
Izlaz:
      Oznaka c: Z
      Oznaka s: Z
      Oznaka e: 1.230000e-04
      Oznaka g: 0.000123
      Oznaka g: 1.23e-05
```

Primjer 6: Ako se operator formatiranja `%` koristi za umetanje više vrijednosti u niz znakova, sve vrijednosti moraju se navesti u oblim zagradama, tj. n-torki. Ako je potrebno umetnuti samo jednu vrijednosti, ona se može navesti i bez korištenja zagrada.

```
a = "Broj:"
b = 55

print("%s %d" % (a, b))
print("Broj: %d" % b)
```

```
Izlaz:
      Broj: 55
      Broj: 55
```

Operator formatiranja `%` je stariji način formatiranja nizova znakova u Pythonu te se u novijim verzijama preporučuje korištenje načina koji će biti obrađeni u nastavku, a to su metoda `format()` i f-strings.

7.2. Vježba: Operator formatiranja `%`

1. U varijablu imena `pi` spremite vrijednost 3.1415926. Korištenjem operatora formatiranja `%` ispišite tu vrijednost s trima decimalama.
2. Kreirajte 3 varijable koje neka se zovu: `ime`, `prezime`, `godine`. U te varijable spremite proizvoljne vrijednosti te ih ispišite prema niže prikazanom primjeru koristeći operator formatiranja `%`.

```
012345678901234567890123456789
      Pero      Perić         19
```

3. U nastavku se nalazi tablica nogometne lige. Pomoću operatora formatiranja `%` ispišite tablicu na ekranu. Vrijednosti nije potrebno spremati u zasebne varijable, već se mogu direktno predati u oblim zagradama. U prvom retku nalaze se brojevi koji olakšavaju detektiranje koje je oznake za formatiranje potrebno koristiti.

```
012345678901234567890123456789012345
Naziv     OS   P    N    I    B
K1        26   10   5    12   35
K2        26   9    5    12   32
K3        26   4    12   10   24
```

7.3. Metoda `format()`

Metoda `format()` noviji je način formatiranja nizova znakova. Ova metoda poziva se nad formatnim nizom. Formatni niz konstruira se na sličan način kao kod operatora formatiranja `%`. U ovom slučaju oznake se označavaju vitičastim zagradama `{}`. Unutar vitičastih zagrada može se navoditi:

- ime argumenta
- redni broj argumenta koji dolazi na to mjesto (indeksi argumenta kreću od 0)
- ništa – ako je broj vitičastih zagrada jednak broju predanih argumenata, unutar vitičastih zagrada nije potrebno navoditi nikakve vrijednosti.

Primjer 1: Korištenje metode `format()`.

```
a = 10
b = 20

print("{} * {} = {}".format(a, b, a * b))
print("{0} * {1} = {2}".format(a, b, a * b))
print("{a} * {b} = {r}".format(a=a, b=b, r=a * b))
```

```
Izlaz:
      10 * 20 = 200
      10 * 20 = 200
      10 * 20 = 200
```

Primjer 2: Korištenjem metode `format()` vrijednosti koje se ponavljaju u ispisu ne moraju se više puta stavljati kao argumenti koji se predaju funkciji `format()`, već ih je moguće jednostavno pozvati preko indeksa ili pak imena argumenta.

```
a = 1
b = 2

print("a: {0}{0}{0}, b:{1}{1}{1}".format(a, b))
print("a: {a}{a}{a}, b:{b}{b}{b}".format(a=a, b=b))
```

```
Izlaz:
      a: 111, b:222
      a: 111, b:222
```

Kod metode `format()` također je moguće kontrolirati širinu i poravnanje.

Širina se navodi unutar vitičastih zagrada, na primjer: `{:5s}`, `{0:5s}`, `{a:5s}`. Poravnanje se navodi oznakama:

- < za lijevo poravnanje (predefinirano poravnanje za nizove znakova)
- > za desno poravnanje (predefinirano poravnanje za brojeve)
- ^ za centriranje poravnanja.

Primjer 3: Formatiranje cjelobrojnih vrijednosti.

```
a = 10
b = 20

print("01234567890123456789")
print("{:5d} * {:5d}|".format(a, b))
print("{0:5d} * {1:5d}|".format(a, b))
print("{a:5d} * {b:5d}|".format(a=a, b=b))
print("{:<5d} * {:<5d}|".format(a, b))
print("{:^5d} * {:^5d}|".format(a, b))
```

```
Izlaz:
      01234567890123456789
         10 *    20|
         10 *    20|
         10 *    20|
      10    * 20   |
       10   *  20  |
```

Primjer 4: Formatiranje nizova znakova.

```
a = "Srce"

print("01234567890123456789")
print("{:20s}|".format(a))
print("{:>20s}|".format(a))
print("{:^20s}|".format(a))
```

```
Izlaz:
      01234567890123456789
      Srce                |
                      Srce|
               Srce       |
```

Primjer 5: Formatiranje realnih brojeva.

```
a = 10.12345

print("01234567890123456789")
print("{:.2f}|".format(a))
print("{:5.2f}|".format(a))
print("{:10.2f}|".format(a))
print("{:<10.2f}|".format(a))
print("{:^10.2f}|".format(a))
```

```
Izlaz:
      01234567890123456789
      10.12|
      10.12|
           10.12|
      10.12     |
        10.12   |
```

7.4. Vježba: Metoda `format()`

1. U varijablu imena `pi` spremite vrijednost 3.1415926. Korištenjem metode `format()` ispišite tu vrijednost s trima decimalama.

2. Kreirajte 3 varijable koje neka se zovu: `ime`, `prezime`, `godine`. U te varijable spremite proizvoljne vrijednosti te ih ispišite prema niže prikazanom primjeru koristeći metodu `format()`.

```
01234567890123456789012345678 9
      Pero     Perić        19
```

3. U nastavku se nalazi tablica nogometne lige. Pomoću metode `format()` ispišite tablicu na ekranu. Vrijednosti nije potrebno spremati u zasebne varijable, već se mogu direktno predati kao stvarni argumenti metode `format()`. U prvom retku nalaze se brojevi koji olakšavaju detektiranje koje je oznake za formatiranje potrebno koristiti.

```
012345678901234567890123456789012345
Naziv     OS   P    N    I    B
K1        26   10   5    12   35
K2        26   9    5    12   32
K3        26   4    12   10   24
```

7.5. f-strings

f-strings (engl. *Formatted string literals*) je način formatiranja nizova znakova u Pythonu. Ovaj način uveden je u verziji Python 3.6, što pak znači da u starijim verzijama Pythona način formatiranja f-strings nije moguće koristiti te je potrebno instalirati verziju Python 3.6 ili noviju verziju. Ovaj način formatiranja omogućuje jednostavnije i preglednije umetanje vrijednosti varijabli u nizove znakova. Sintaksa korištenja načina formatiranja f-strings je da se ispred navodnika stavi slovo `f` ili `F`, a same varijable stavljaju se u vitičaste zagrade `{}`. U vitičaste zagrade mogu se stavljati i izrazi koji se izračunavaju prije samog ispisa.

Primjer 1: Korištenje načina formatiranja f-strings.

```
a = 10
b = 20

print(f"{a} * {b} = {a * b}")
```

```
Izlaz:
      10 * 20 = 200
```

Metoda `format()` i f-strings, osim što formatiranje rade na sličan način koristeći vitičaste zagrade, također omogućuju i detaljnija formatiranja kao što je kontrola širine ispisa i poravnanja te ih rade na identičan način. Kod f-strings najprije se u vitičastim zagradama piše ime varijable, potom dvotočka `:` i nakon toga oznake samog formatiranja koje su obrađene u poglavlju *7.4. Metoda `format()`*.

Primjer 2: Formatiranje cjelobrojnih vrijednosti.

```
a = 10
b = 20

print("01234567890123456789")
print(f"{a:5d} * {b:5d}|")
print(f"{a:<5d} * {b:<5d}|")
print(f"{a:^5d} * {b:^5d}|")
```

```
Izlaz:
      01234567890123456789
         10 *    20|
      10    * 20   |
       10   *  20  |
```

Primjer 3: Formatiranje nizova znakova.

```
a = "Srce"

print("01234567890123456789")
print(f"{a:20s}|")
print(f"{a:>20s}|")
print(f"{a:^20s}|")
```

```
Izlaz:
      01234567890123456789
      Srce                |
                      Srce|
               Srce       |
```

Primjer 4: Formatiranje realnih brojeva.

```
a = 10.12345

print("01234567890123456789")
print(f"{a:.2f}|")
print(f"{a:5.2f}|")
print(f"{a:10.2f}|")
print(f"{a:<10.2f}|")
print(f"{a:^10.2f}|")
```

```
Izlaz:
      01234567890123456789
      10.12|
      10.12|
           10.12|
      10.12     |
        10.12   |
```

7.6. Vježba: f-strings

1. U varijablu imena `pi` spremite vrijednost 3.1415926. Korištenjem načina formatiranja f-strings ispišite tu vrijednost s trima decimalama.
2. Kreirajte 3 varijable koje neka se zovu: `ime`, `prezime`, `godine`. U te varijable spremite proizvoljne vrijednosti te ih ispišite prema niže prikazanom primjeru koristeć način formatiranja i f-strings.

```
01234567890123456789012345678901234567890123456789
      Pero     Perić        19
```

3. U nastavku se nalazi tablica s brojevima od 95 do 104. Proučite način formatiranja tablice te osmislite logiku kojom ćete pomoću načina formatiranja f-strings ispisati tablicu na ekranu. Zadatak ne smije biti riješen tako da se konkretni brojevi direktno ispisuju na ekranu, već je zadatak potrebno riješiti pomoću petlje.

```
01234567890123456789
   95   96   97   98
   96   97   98   99
   97   98   99  100
   98   99  100  101
   99  100  101  102
  100  101  102  103
  101  102  103  104
```

7.7. Pitanja za ponavljanje: Formatiranje ispisa podataka

1. Koji načini formatiranja ispisa podataka postoje?
2. Koje se oznake kod operatora formatiranja % najčešće koriste?
3. Koja se oznaka koristi za ispis samo dviju decimala (navesti za sva 3 načina formatiranja)?
4. Koje se oznake koriste za poravnanje nizova znakova (navesti za sva 3 načina formatiranja)?

8. Tekstualne datoteke

Trajanje poglavlja:

60 min

Po završetku ovog poglavlja polaznik će moći:

- kreirati, čitati, promijeniti i brisati tekstualne datoteke.

Za vrijeme izvršavanja Python skripti svi podaci koje skripta izračuna nalaze se u radnoj memoriji – RAM memorija (engl. *Random Access Memory*). Tim podacima nakon završetka izvođenja skripte više nije moguće pristupiti. Sve podatke koji se žele trajno spremiti za kasnije korištenje potrebno 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. Tako spremljeni podaci u trajnoj memoriji ostaju zapisani i nakon isključivanja računala.

U ovom poglavlju opisane su osnovne operacije za rad s tekstualnim datotekama, a to su:

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

Ponegdje se ove četiri operacije spominju i pod nazivom *CRUD* (engl. *CreateReadUpdateDelete).*

U trajnoj memoriji podaci su spremljeni u datoteke. Svaka datoteka ekstenzijom je opisana 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 pozornost da, na primjer, tipovi datoteka `*.txt` i `*.doc` nisu identične strukture, što znači da, ako želimo skriptom spremati podatke u `*.doc` tip datoteke, potrebno je proučiti na koji način se takva datoteka gradi. Najjednostavniji tip datoteke za obradu je tekstualna datoteka tj. `*.txt` jer ona ne zahtijeva nikakvu dodatnu strukturu te će se ona koristiti u ovom poglavlju.

Uz datoteke bitni su i direktoriji, tj. mape (engl. *folder*).

Direktoriji olakšavaju pregled i pronalaženje datoteka. Direktoriji se slažu u hijerarhije po smislenoj strukturi. Za razdjeljivanje nivoa direktorija koristi se znak: `\` ili `/` ovisno o operacijskom sustavu.

Operacijski sustav Windows:

```
C:\Users\Korisnik\Documents\datoteka.txt
```

Operacijski sustav UNIX:

```
/home/korisnik/Documents/datoteka.txt
```

Hijerarhija mape koja vodi sve do neke određene datoteke zove se putanja (engl. *path*). Putanje možemo podijeliti na dvije vrste, a to su apsolutne i relativne putanje.

Apsolutna putanja – cijeli put od vršnog (korijenskog) direktorija pa do tražene datoteke ili direktorija.

Relativna putanja – putanja koja nije potpuna, već je to putanja do datoteke ili direktorija krenuvši od direktorija u kojem u kojem je korisnik trenutno pozicioniran (npr. direktorija u kojem se nalazi skripta koja se pokreće).

Primjer strukture direktorija:

```
C:\Users\Ime\Dokumenti (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 se pak u skripti `skripta.py` za dohvaćanje datoteke `Podaci.txt` želi koristiti apsolutna putanja, potrebno je napisati:

```
C:\Users\Ime\Dokumenti\Program\Datoteke\Podaci.txt
```

Postoje dvije vrste datoteka, a to su tekstualne i binarne datoteke, ovaj tečaj obrađuje isključivo tekstualne datoteke.

8.1. Otvaranje datoteke

Prije čitanja iz datoteke ili pak pisanja u datoteku istu je potrebno otvoriti [2]. Datoteka se otvara pozivom funkcije `open()`. Funkcija `open()` vraća datotečni objekt (engl. *file object*). U nastavku se nalazi funkcija `open()` i svi argumenti koje ona može primiti:

```
open(file, mode='r', buffering=- 1, encoding=None,
errors=None, newline=None, closefd=True, opener=None)
```

Na ovom tečaju obradit će se argumenti `file` i `mode`.

- `file` – ime datoteke (može se koristiti i kao apsolutna i kao relativna putanja) koja će se kreirati / čitati / mijenjati
- `mode` – način korištenja (engl. *mode*) – prava koja datotečni objekt ima za obradu datoteke. Tablica svih opcija koje ovaj argument može poprimiti nalazi se u nastavku.

Napomena

Uz tekstualni format datoteka postoji i binarni format zapisa podataka u datoteke.

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

U gornjem primjeru argument `datoteka.txt` je naziv datoteke koja se želi koristiti, dok je drugi argument `w` pravo s kojim kreirani datotečni objekt može koristiti datoteku.

Tablica načina korištenja datoteka:

Način	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 upisuju se na kraj datoteke. Nije dopušteno čitanje iz datoteke. Pokazivač za pisanje u datoteku nalazi se na kraju datoteke.
`+`	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 opcijski. Ako se ovaj parametar izostavi, predefinirana vrijednost je `'r'` i `'t'`.

8.2. Zatvaranje datoteke

Svaki datotečni objekt koji se više ne planira koristiti potrebno je zatvoriti.

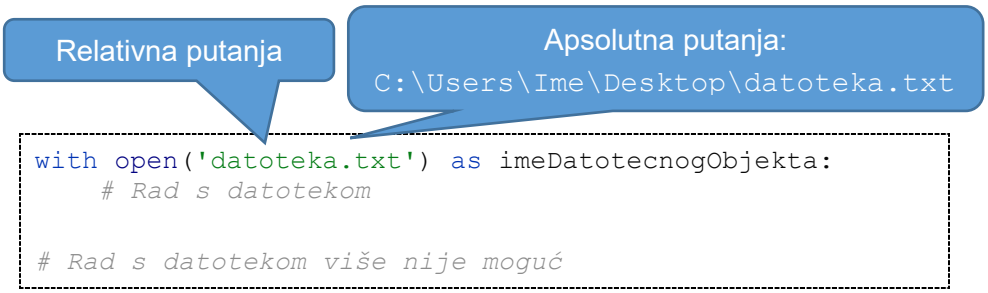
Razlozi za zatvaranje datotečnog objekta:

- optimizacija korištenja resursa operacijskog sustava
- povećanje vremena raspoloživosti datoteke drugim programima koji moraju koristiti tu datoteku
- zapisivanje podataka u datoteku – podaci se ne zapisuju direktno u datoteku, već se zapisuju u memorijski međuspremnik. Podaci će biti zapisani u datoteku tek kada se memorijski međuspremnik napuni, prilikom zatvaranja datotečnog objekta ili pak pozivanjem metode `flush()`. Za tekstualne datoteke memorijski međuspremnik nije moguće isključiti, može se jedino postaviti da se on koristi za najmanje jedan redak teksta. Memorijski međuspremnik koristi se za poboljšanje performansi pristupa disku.

Sintaksa korištenja metode `close()` za zatvaranje datoteke:

```
imeDatotecnogObjekta.close()
```

Primjer 1: Kod rada s datotekama dobra praksa je koristiti naredbu `with`. Po završetku bloka naredbi unutar tijela `with` datotečni objekt bit će automatski zatvoren bez potrebe za pozivanjem metode `close()`. U nastavku je prikazana sintaksa korištenja prethodno opisanoga načina.




```
with open('datoteka.txt') as imeDatotecnogObjekta:
    # Rad s datotekom

# Rad s datotekom više nije moguć
```

8.3. Pisanje u datoteku

Nakon što je datotečni objekt kreiran, u datoteku je moguće zapisivati željeni sadržaj. Za pisanje u datoteku koristi se metoda `write(str)`. Ova metoda u datoteku zapisuje niz znakova koji je spremljen u varijabli tipa podataka `str` ili je pak direktno predan (vrijednost obavezno mora biti tipa podataka `str`).

Sintaksa korištenja metode `write()`:

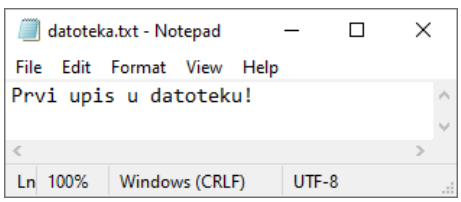
```
imeDatotecnogObjekta.write(str)
```

Povratna vrijednost ove metode jest broj znakova koji je uspješno zapisan u datoteku. Ako u datoteku želimo zapisivati niz znakova, prilikom kreiranja datotečnog objekta potrebno je prenijeti i parametar `mode`. Kao što je prethodno navedeno, ako se parametar `mode` ne prenese, on poprima predefiniranu vrijednost, a to je dozvola samo za čitanje iz datoteke.

Primjer 1: U datoteku imena `datoteka.txt` upisuje se niz znakova „Prvi upis u datoteku!“.

```
with open('datoteka.txt', 'w') as f:
    n = f.write('Prvi upis u datoteku!')
    print(n)
```

```
Izlaz:
      21
```



Primjer 2: Pokušaj upisivanja vrijednosti tipa podataka – `int` u datoteku.

```
a = 10

print(type(a))

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

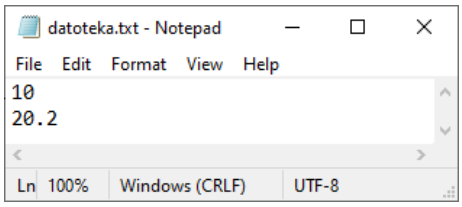
```
Izlaz:
      <class 'int'>
      TypeError: write() argument must be str, not int
```

Kao što se vidi iz gornjeg primjera, pokušaj upisa cjelobrojne vrijednosti nije uspješan. Ako je u datoteku potrebno spremiti vrijednost nekog drugog tipa podataka, na primjer: cjelobrojnu vrijednost – `int`, realnu vrijednost – `float` i tako dalje, to je također moguće napraviti, no prije samog upisa u datoteku te vrijednosti potrebno je pretvoriti u tip podataka niz znakova – `str`.

Primjer 3: Pretvaranje neke vrijednosti u niz znakova, obrađuje se pozivom funkcije `str()`.

```
a = 10

print(type(a))
print(type(20.20))

with open('datoteka.txt', 'w') as f:
    f.write(str(a) + "\n")
    f.write(str(20.20))
```

```
Izlaz:
      <class 'int'>
      <class 'float'>
```



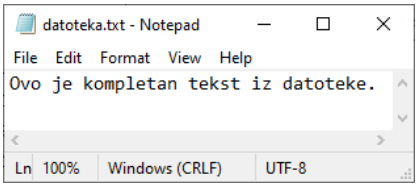
8.4. Čitanje sadržaja datoteke

Za čitanje sadržaja datoteke, potrebno je pozvati metodu `read(size)`. Ova metoda čita podatke iz datoteke i vraća ih kao niz znakova. Parametar `size` je opcijski argument. U slučaju da je parametar `size` izostavljen, čita se cijela datoteka. Sintaksa korištenja metode `read(size)`:

```
sadrzaj = imeDatotecnogObjekta.read(size)
```

Programer mora paziti da veličina datoteke koja se čita ne bude veća od radne memorije računala na kojem se skripta izvršava. U slučaju kada metoda `read()` nema znakove za čitanje zato što je datotečna kazaljka (engl. *file pointer*) na kraju datoteke metoda `read()` vraća prazan niz znakova.

Primjer 1: U datoteci `datoteka.txt` nalazi se niz znakova sadržaja: „Ovo je sav tekst iz datoteke.“. U nastavku se nalazi prikaz izgleda datoteke i primjer programskog kôda koji čita sadržaj datoteke.




```
with open('datoteka.txt') as f:
    sadrzajDatoteke = f.read()
    print(sadrzajDatoteke)
    sadrzajDatoteke = f.read()
    print(">" + sadrzajDatoteke + "<")
```

```
Izlaz:
      Ovo je kompletan tekst iz datoteke.
      ><
```

Metoda `f.read()` poziva se dva puta. Prilikom prvoga poziva učitao se kompletan tekst koji se nalazi u datoteci `datoteka.txt`, a 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 i datotečna kazaljka se sada nalazi na kraju datoteke, u tom koraku nije ostalo ništa više za pročitati i u varijablu `cijelaDatoteka` vraćen je prazni niz znakova.

Primjer 2: Metodi `read()` prenosi se i parametar `size`. U ovom slučaju ovaj parametar označava koliko se znakova želi pročitati. U trima čitanjima čita se po 5 znakov i, kao što se može vidjeti iz ispisa, kod prvog čitanja pročitan je niz znakova „`Ovo j`“, kod drugog čitanja pročitan je niz znakova: „`e sav`“, a kod trećeg: „`pleta`“.

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

```
Izlaz:
      Ovo j
      e kom
      pleta
```

8.5. Čitanje sadržaja datoteke – liniju po liniju

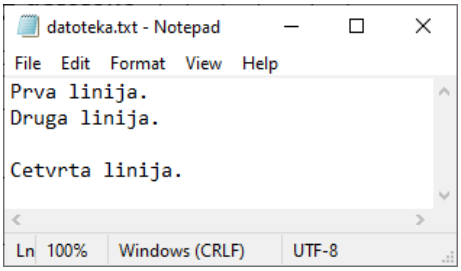
Čitanje sadržaja datoteke liniju po liniju moguće je napraviti na nekoliko načina koji su objašnjeni u nastavku:

- korištenjem metode `f.readline()`
- korištenjem metode `f.readlines()`
- korištenjem petlje `for` (preporučeni način).

Metoda `f.readline()`

Prilikom svakog poziva ova metoda pročita po jednu liniju iz datoteke. U jednu liniju ulaze svi znakovi do uključujući znak za novu liniju `'\n'` (svaka linija završava znakom `'\n'`). U slučaju da je negdje u datoteci pročitana prazna linija, tj. `'\n'`, u povratnoj vrijednosti niza znakova nalazi se samo znak `'\n'`. Ako metoda `f.readline()` vrati prazan niz znakova, to znači da je čitanje došlo do kraja datoteke.

Sadržaj datoteke `datoteka.txt` je:



Primjer 1: Čitanje liniju po liniju 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.
```

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()` iz datoteke uz tekst pročitala i znak `'\n'` koji označava novi redak. Prilikom ispisa niza znakova ispisana su dva nova retka, 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`, a predefinirana vrijednost argumenta `end` je novi redak, tj. `'\n'`. Ovu metodu nije moguće koristiti uz petlju `for`.

Metoda `f.readlines()`

Učitava cijeli sadržaj datoteke u radnu memoriju. Zbog učitavanja cjelokupnog sadržaja u radnu memoriju ovaj način nije preporučljiv jer prilikom obrade velikih datoteka može doći do degradacije performansi.

Primjer 2: Ovaj programski kôd prikazuje način kako iterirati po cijeloj datoteci koja je učitana u radnu memoriju.

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

```
Izlaz:
      Prva linija.
      Druga linija.

      Cetvrta linija.
```

Primjer 3: Uz gore prikazan način, za čitanje sadržaja cijele datoteke u radnu memoriju može se koristiti i funkcija `list(f)`.

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

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

Čitanje sadržaja datoteke liniju po liniju pomoću petlje `for`

Ovo je preporučeni način jer ne učitava cijeli sadržaj datoteke u radnu memoriju. Ovim načinom možemo obrađivati datoteke neovisno o njihovoj veličini. Ovo je moguće napraviti jer je datotečni objekt pobrojiv.

Primjer 4: Čitanje sadržaja datoteke pomoću petlje `for`.

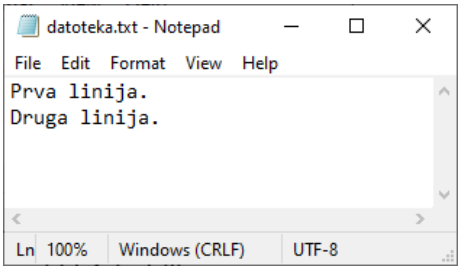
```
with open('datoteka.txt') as f:
    for linija in f:
        print(linija, end="")
```

```
Izlaz:
      Prva linija.
      Druga linija.

      Cetvrta linija.
```

8.6. Pisanje na kraj datoteke

Ako se u tekstualnu datoteku u kojoj otprije postoji sadržaj želi upisati novi tekst, na kraj datoteke (bez da se prebriše postojeći sadržaj) potrebno je otvoriti datotečni objekt željene datoteke s modom korištenja `'a'`.

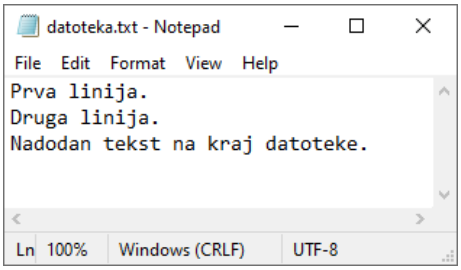
Početni sadržaj datoteke `datoteka.txt` je:



Primjer 1: Nakon pokretanja programskog kôda:

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

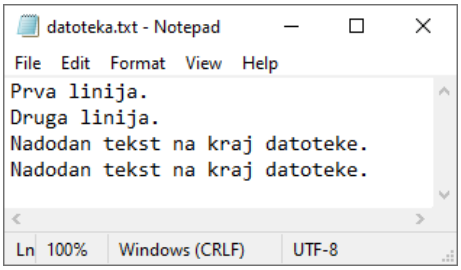
Rezultat upisa u tekstualnu datoteku je:



Primjer 2: Ako je datotečni objekt otvoren modom korištenja tipa `'a'` ili `'a+'`, svaki put kada se pomoću metode `write()` upisuje novi tekst u datoteku, taj tekst sprema se isključivo na kraj datoteke. Pa čak i u slučaju ako se datotečna kazaljka pozivom metode `f.seek(0, 0)` prebaci na početak datoteke.

```
with open('datoteka.txt', 'a') as f:
    f.write("\nNadodan tekst na kraj datoteke.")
    f.seek(0, 0)
    f.write("\nNadodan tekst na kraj datoteke.")
```

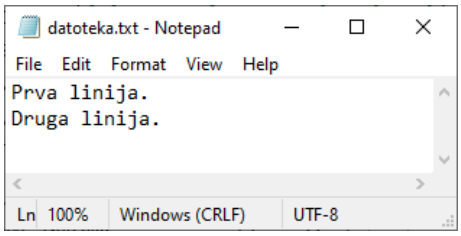
Rezultat upisa u tekstualnu datoteku je:



8.7. Pisanje na početak datoteke

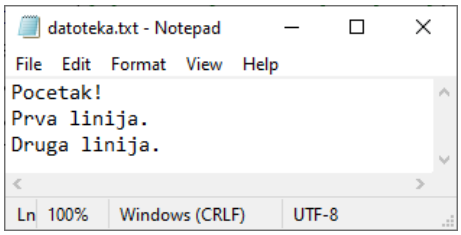
Za razliku od pisanja na kraj datoteke, pisanje na početak datoteke nije trivijalno jer ne postoji ugrađen mehanizam za pisanje na početak datoteke. U ovom poglavlju bit će prikazan jedan potencijalan način kako na početak datoteke upisati željeni sadržaj. Sadržaj datoteke cijeli se učita u radnu memoriju, potom se u radnoj memoriji sadržaj varijable uređuje te se vraća u datoteku.

Primjer 1: Pisanje novog sadržaja datoteke na početak.




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

Ako se sadržaj datoteke `datoteka.txt` želi urediti tako da se na sâm početak datoteke doda novi tekst, najprije je potrebno cijeli sadržaj datoteke učitati u radnu memoriju. Nakon što je sadržaj datoteke učitan u varijablu imena `sadrzaj`, željeni početni tekst stavlja se na početak varijable. Rezultat gornjega programskog kôda je:



U ovom programskom kôdu datotečni objekt 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 datotečnog objekta.

Napomena: iako nije razrađivano u ovom tečaju, ovakav postupak može se iskoristiti ako se želi umetnuti novi sadržaj datoteke na bilo koju lokaciju u datoteci (osim na kraj datoteke, to se radi načinom korištenja `'a'`). Također, ovakvim postupkom može se neki segment datoteke, nakon što je ona učitana u radnu memoriju, izbaciti i potom ponovo upisati u tekstualnu datoteku. Prilikom ovakvog rada potrebno je paziti na sljedeće stvari:

- veličinu datoteke koja se učitava u radnu memoriju

- ako novi sadržaj ima manje znakova nego stari sadržaj datoteke, na kraju datoteke ostat će „stari“ znakovi koji nisu zamijenjeni novim sadržajem (takve znakove potrebno je ukloniti).

8.8. Datotečna kazaljka

U gornjem primjeru programskoga kôda pojavljuje se poziv metode `seek()`. Metoda `seek()` služi za pomicanje datotečne kazaljke unutar datoteke. Sintaksa poziva te metode je:

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

Parametar `offset` je pozicija koja je izračunata u odnosu na referentnu točku. Referentna točka je definirana pomoću argumenta `from_what`. Donja tablica opisuje vrijednosti argumenta `from_what`.

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

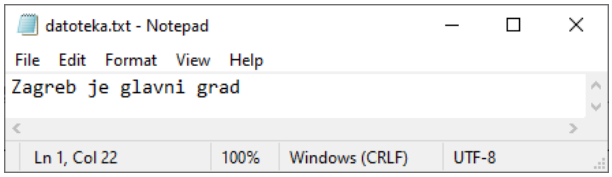
Dodatno objašnjenje korištenja metode `seek()`: u primjeru programskog kôda u *poglavlju 8.7*. najprije je pomoću metode `read()` pročitana cijela datoteka. Nakon čitanja cijele datoteke datotečna kazaljka pokazuje na kraj datoteke. Kako je sljedeći korak zapisivanje teksta natrag u datoteku, potrebno je pomoću metode `seek()` pozicionirati datotečnu kazaljku natrag na početak tekstualne datoteke odakle će krenuti zapisivanje novoga teksta natrag u datoteku. Ako se to ne napravi, zapisivanje sadržaja pozivom metode `write()` krenut će od pozicije datotečne kazaljke, a ona se nalazi na kraju datoteke.

Napomena: ako je datotečni objekt otvoren modom korištenja `'a'`, u tom slučaju metoda `write()` piše isključivo na kraj datoteke (neovisno o trenutnoj poziciji datotečne kazaljke) jer se prije početka zapisivanja datotečna kazaljka uvijek automatikom postavlja na kraj datoteke.

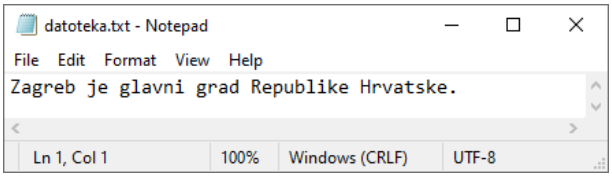
Također, postoji metoda `tell()` koja vraća trenutnu poziciju datotečne kazaljke u datoteci. Trenutačna pozicija je broj bajtova od početka datoteke.

Primjer 1: U nastavku se nalazi programski kôd u kojem se može vidjeti funkcioniranje metoda `tell()` i `seek()` u kombinacijama s metodama `read()` i `write()`.

Sadržaj datoteke na početku:



Sadržaj datoteke nakon izvršavanja kôda koji se nalazi u nastavku:




```
with open('datoteka.txt', 'r+') as f:
    print(f.tell())  # Pozicija na početku

    sadrzaj = f.read(5)
    print(f.tell())  # Pozicija nakon 1. čitanja

    sadrzaj = f.read(5)
    print(f.tell())  # Pozicija nakon 2. čitanja

    f.seek(0, 2)
    print(f.tell())  # Pozicija na kraju datoteke

    f.write(" Republike Hrvatske.")
    print(f.tell())  # Pozicija nakon upisa
```

```
Izlaz:
      0     # Pozicija na početku
      5     # Pozicija nakon 1. čitanja
      10    # Pozicija nakon 2. čitanja
      21    # Pozicija na kraju datoteke
      41    # Pozicija nakon upisa
```

8.9. Vježba: Tekstualne datoteke

Napomena

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

1. Napišite program koji će kreirati praznu tekstualnu datoteku naziva `datoteka.txt` (ako datoteka identičnog imena već postoji, potrebno ju je „pregaziti“ 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 zaslonu.
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 ju zapišite u treću datoteku.

8.10. Pitanja za ponavljanje: Tekstualne datoteke

1. Koji način korištenja služi samo za čitanje iz datoteke?
2. Koji način korištenja služi za brisanje sadržaja postojeće datoteke?
3. Koji način korištenja služi za čitanje i pisanje u datoteku uz postojeći sadržaj?
4. Koji način korištenja služi za pisanje novog sadržaja isključivo na kraj datoteke?

9. Moduli i paketi

Trajanje poglavlja:

30 min

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, organizirani su u smislene cjeline radi što lakšeg snalaženja, ne samo prilikom korištenja, već i radi lakšeg pretraživanja dokumentacije.

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`.

9.1. Rad s modulima i paketima

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

- najavljivanjem korištenja
- izravnim uključivanjem u program.

Napomena: radi preglednosti sadržaja tečaja, u nastavku će se spominjat samo funkcije unutar modula, no podrazumijeva se da su dostupni i razredi te konstante.

Najavljivanje korištenja

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

```
import <imeModula>
```

Ako se korištenje funkcija iz modula najavi, u svakom se dijelu programa gdje se želi koristiti funkcija iz najavljenog modula mora pisati ime modula te potom ime funkcije. Ova imena moraju biti međusobno odvojena točkom. Sintaksa ovakvoga načina korištenja najavljenih funkcija jest:

```
imeModula.imeFunkcije()
```

Primjer 1: Korištenje naredbe `import` i modula `random`. Kod ovakvog načina korištenja naredbe `import` najavljuje se korištenje **svih** funkcija iz nekog modula. Metoda `random()` vraća nasumični realni broj u rasponu 0.0 <= X < 1.0.

```
import random

print(random.random())
```

```
Izlaz:
      0.2950123945997093
```

Izravno uključivanje u program

Ako se želi izbjeći gornja sintaksa i koristiti samo ime funkcije kao poziv, bez pisanja imena modula ispred funkcije koja se želi pozvati, potrebno je željene funkcije izravno uključiti u program. Uključivanje funkcija u program radi se korištenjem ključnih riječi `from` i `import`. Sintaksa takvoga načina korištenja jest:

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

Ključna riječ `from` govori koji će se modul koristiti, dok ključna riječ `import` govori koje se sve funkcije, razredi ili konstante uključuju 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 se modulu nalazi korištena funkcija.

Primjer 2: U ovom primjeru prikazana je sintaksa korištenja funkcija tako da se one izravno uključuju u programski kôd. U program su uključene funkcije koje će se koristiti u programskom kôdu, a to su funkcije `random()` i `randrange()`.

```
from random import random, randrange


print(random())
print(randrange(10))
```

```
Izlaz:
      0.6243677432699887
      4
```

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 da se prilikom uključenja više različitih modula može dogoditi kolizija ako se u dvama 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*.

Primjer 3: Najavljivanje korištenja svih funkcija i konstanti modula `math`.

```
import math

print(math.pi)
```

```
Izlaz:
      3.141592653589793
```

Primjer 4: Uključivanje u program samo konstante `pi` iz modula `math`.

```
from math import pi

print(pi)
```

```
Izlaz:
      3.141592653589793
```

9.2. Vježba: Moduli i paketi

1. Učitajte s tipkovnice cjelobrojnu vrijednost koja predstavlja polumjer kruga te izračunajte i ispišite njegov opseg i površinu. Konstantu π (pi) dohvatite iz modula `math` najavljivanjem korištenja.

$$O = 2 * r * \pi$$

$$P = r^2 * \pi$$

2. Prethodni zadatak riješite tako da konstantu π izravno uključite u program.
3. Pronađite na internetu u službenoj dokumentaciji Pythona 3 kako se zove funkcija koja se nalazi u modulu `math` koja se koristi za

korjenovanje. Putem tipkovnice unesite cjelobrojnu vrijednost te pomoću funkcije za korjenovanje iz modula `math` izračunajte i ispišite njegov korijen. Zadatak riješite ili najavljivanjem korištenja ili izravnim uključenjem u program.

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

4. U službenoj dokumentaciji Pythona 3 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.
5. * U službenoj dokumentaciji Pythona 3 pronađite imena funkcija koje služe 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 ekranu mora biti sadržaja: „5, 6“. U ovom zadatku najavite korištenje funkcija, no nemojte ih uključiti.

9.3. Pitanja za ponavljanje: Moduli i paketi

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 se modulu pozvana funkcija nalazi?

10. Kolekcije objekata

Trajanje poglavlja:

125 min

Po završetku ovog poglavlja polaznik će moći:

- koristiti kolekcije objekata: liste, skupove, rječnike i n-terce.

Kolekcije objekata u Pythonu su tipovi podataka koji se sastoje od skupa raznih vrijednosti spremljenih u jednu varijablu ili objekt. Kolekcije omogućavaju programerima učinkovitu organizaciju i manipulaciju velikim količinama podataka. Kolekcije objekata ključne su za svakodnevni rad s podacima. U ovom tečaju obrađivat će se 4 ugrađene kolekcije objekata, a to su: liste (engl. *list*), n-torke (engl. *tuple*), rječnici (engl. *dictionary*) i skupovi (engl. *set*).

Postoje dvije osnovne vrste kolekcija objekata: sljedne i asocijativne kolekcije.

U **slijedne kolekcije** svrstavaju se znakovni nizovi, liste i n-torke. Elementima slijednih kolekcija pristupa se putem indeksa. Prednost korištenja sljednih kolekcija je jednostavnost i iznimno brz pristup elementima preko njihova indeksa, dok je nedostatak činjenica da je potrebno izbjegavati postupke koji zahtijevaju puno umetanja, brisanja i pridruživanja.

U **asocijativne kolekcije** svrstavaju se rječnici i skupovi. Elementima asocijativnih kolekcija pristupa se preko ključa. Prednost korištenja asocijativnih kolekcija je konstantno prosječno vrijeme pristupa, umetanja i brisanja, dok je nedostatak što redoslijed obilaska elemenata nije unaprijed definiran niti predvidiv.

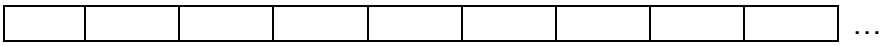
10.1. Lista (engl. *List*)

Liste pripadaju u slijedne kolekcije i one su izmjenjive (engl. *mutable*), što znači da je elemente liste moguće mijenjati, brisati i dodavati [6].

Liste možemo smatrati strukturama koje nam služe za organiziranje podataka. 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 itd.), a u drugoj listi nalaze se stvari za kupnju u trgovini tehničke robe.

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 s tipkovnice tako dugo dok se 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). 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 spremamo vrijednosti jednu iza druge. 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.

Svaka pohranjena vrijednost u listi dobiva svoj indeks te se dodavanjem svakoga novog elementa indeks povećava za 1. **Indeksi uvijek kreću 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 5. mjestu i dobit će indeks 4.

Za razliku od nekih drugih programskih jezika, na primjer programskog jezika C, kod Pythona nije potrebno definirati koji će tip vrijednosti 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 vidljivo je da se na 0. indeksu nalazi niz znakova, na 1. indeksu nalazi se cijeli broj, a na 2. indeksu nalazi se realni 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, prije svake akcije nad nekom vrijednosti preporučljivo je provjeriti koji tip podatka se obrađuje. Prilikom osmišljavanja logike programa potrebno je osmisliti dobru organizaciju podataka.

10.1.1. Izrada liste

Lista s definiranim početnim vrijednostima kreira se tako da se vrijednosti međusobno odvojene zarezom pišu unutar uglatih zagrada (otvorena i zatvorena uglata zagrada) tj. `[` i `]`. Sintaksa definiranja liste s vrijednostima:

```
imeListe = [vrijednost1, vrijednost2, …]
```

Često se definira prazna lista koja se naknadno koristi za spremanje vrijednosti koje program izračunava ili pak za spremanje vrijednosti koje se čitaju s tipkovnice ili pak nekog drugog izvora. Prazna lista definira se praznim otvorenim-zatvorenim uglatim zagradama.

```
imeListe = []
```

Primjer 1: Kreiranje liste od 3 elementa, od kojih je svaki od elemenata drugog tipa podataka.

```
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 realni broj). Elementi liste stavljaju se unutar uglatih zagrada. Ako se želi kreirati prazna lista, može se 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.

10.1.2. Dohvaćanje elemenata liste

Ako u listi postoje pohranjene vrijednosti, pretpostavka je da im je potrebno i pristupati, bilo na način da se iz liste uzima po jedan element, po nekoliko elemenata ili pak svi elementi liste. 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. Pomoću indeksa moguće je pristupati elementima liste na točno željenim pozicijama. Pravila vezana za indekse koja su prethodno obrađena u poglavlju *3.9. Nizovi znakova* primjenjuju se i na pravila indeksa u listama. Razlika je u tome što se kod nizova znakova svako slovo nalazi na svom indeksu, dok se kod lista na jednom indeksu nalazi po jedna vrijednost. Jedna vrijednost može biti jedan broj, niz znakova, druga lista itd.

Primjer 1: 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

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

```
Izlaz:
      Luka
```

U drugoj liniji programskog kôda poziva se funkcija `print()` kojoj se kao argument predaje vrijednost koja se nalazi u listi na indeksu 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.

Primjer 2: Kao što je objašnjeno u poglavlju s nizovima znakova, i kod lista moguće je dohvatiti vrijednosti od indeksa do indeksa. Sintaksa i detaljniji opis je prikazani su u nastavku.

`imeListe[start:stop]`

- `start` – indeks od kojeg kreće ispisivanje
- `stop` – vrijednost za 1 broj veća od indeksa zadnjeg elementa koji se ispisuje

Moguće je izostaviti vrijednost `start` ili `stop`. U nastavku se nalazi opis što se dogodi ako se izostavi `start`, a što se dogodi ako se izostavi vrijednost `stop`:

- `start` – dohvaćanje znakova kreće od nultog indeksa
- `stop` – dohvaćanje znakova ide do kraja niza znakova.

```
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']
```

10.1.3. Iteriranje po elementima liste

Postoji više načina iteracije po elementima liste, a u ovom poglavlju bit će prikazana 2 načina prolaska po elementima liste. Najjednostavniji način je korištenjem petlje `for`, dok je drugi način korištenje petlje `while`.

Petlja `for`

Petlja `for` kao pobrojivi objekt prima listu te ona potom dohvaća element po element iz liste i stavlja ga u upravljačku varijablu proizvoljnog imena. U donjem primjeru upravljačka varijabla nazvana je imenom `e`. Potom u tijelu petlje možemo raditi obradu dobivene vrijednosti iz liste.

Primjer 1: Korištenje petlje `for`.

```
lista = ["Ivan", "Luka", "Nino", "Jan", "Tin"]
for e in lista:
    print(e)
```

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

Petlja `while`

Petlja `while` nema mogućnost direktnog iteriranja po elementima liste, već se za dohvaćanje elemenata koriste indeksi.

Primjer 1: U ovom primjeru programskog kôda varijabla `i` predstavlja indeks elementa koji se želi dohvatiti i ona se na početku postavlja na 0 (kreće se od nultog indeksa). Uvjet petlje `while` daje rezultat `True` toliko dugo dok je vrijednost u varijabli `i` manja od broja elemenata liste. Broj elemenata liste moguće je dobiti korištenjem funkcije `len()`. U svakom koraku petlje varijablu `i` potrebno je uvećati za 1 ili neki drugi broj, na primjer 2 ako se želi dobiti svaki drugi element, 3 za svaki treći element itd.

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

i = 0
while i < len(lista):
    print(lista[i])
    i += 1
```

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

10.1.4. Korištenje dokumentacije za rad s listom

Osim često korištenih metoda za rad s listama, ponekad su potrebne i ugrađene metode koje programer ne koristi često i jednostavno ih smetne s uma. Postoje ugrađene funkcije koje se koriste za pružanje informacija o objektima i njihovim metodama.

Funkcija `dir()` koristi se za prikazivanje popisa metoda koje su dostupne za objekt koji se funkciji prosljeđuje kao argument. Funkciji se kao argument prosljeđuje tip podataka za koji se želi dobiti više informacija.

```
dir(<tipPodataka>)
```

Funkcija `help()` koristi se za dobivanje detaljnih informacija. Funkciji se kao argument prosljeđuje tip podataka za koji se želi dobiti više informacija.

```
help(<tipPodataka>)
```

Ako se funkcija `dir()` pozove unutar samog programskog kôda (skripte), na zaslonu se neće ispisati ništa. Funkcija `dir()` mora se pozvati unutar IDLE Shella. Funkcija `help()` nema prethodno navedeno ograničenje.

Kolekcija objekata lista je tip podataka `list`, tako da se ovim funkcijama može proslijediti ili objekt koji sadrži tip podataka lista ili pak `list` kako bi se dobilo više informacija.

Primjer 1: Popis metoda liste dobiven korištenjem funkcije `dir()`.

```
>>> dir(list)
['__add__', '__class__', '__class_getitem__',
'__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getstate__',
'__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', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

Iz gornjeg popisa u ovom trenutku bitne su nam samo metode koja ne počinju i završavaju s __, a to su metode: *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 broj pojavljivanja predane vrijednosti u listi
- `extend()` – proširuje listu elementima druge liste
- `index()` – vraća indeks prvog pojavljivanja predane vrijednosti
- `insert()` – umeće vrijednost u listu na zadani indeks
- `pop()` – uklanja i vraća element iz liste
- `remove()` – uklanja prvo pojavljivanje određene vrijednosti
- `reverse()` – obrće redoslijed elemenata u listi
- `sort()` – metoda koja sortira elemente liste.

Primjer 2: Korištenje funkcije `help()`.

```
help(list)
```

```
Izlaz:
      ...
       |  append(self, object, /)
       |      Append object to the end of the list.
       |
       |  clear(self, /)
       |      Remove all items from list.
       |
       |  copy(self, /)
       |      Return a shallow copy of the list.
```

```
        |   ...
```

10.1.5. Dodavanje novog elementa u listu

U gornjim primjerima lista je popunjavana vrijednostima odmah prilikom deklaracije. U praksi, gotovo uvijek je potrebno naknadno dodavati nove vrijednosti u listu. Za to se koriste ugrađene metode. Za dodavanje nove vrijednosti na kraj liste koristi se metoda `append()`. Sintaksa dodavanja nove vrijednosti je:

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

Primjer 1: Dodavanje novih elemenata u listu.

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

print(lista)
lista.append("Jan")
lista.append("Tin")
print(lista)
```

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

Ako je u listu potrebno dodati više elemenata, potrebno je više puta pozvati metodu `append()` ili pak koristiti ugrađenu metodu `extend()` koja će biti objašnjena u nastavku.

10.1.6. Uklanjanje svih elemenata iz liste

Ako je iz liste potrebno izbaciti sve vrijednosti, to je moguće napraviti pomoću metode `clear()`. Nakon poziva ove metode nad listom lista postaje prazna. Sintaksa za uklanjanje svih elemenata iz liste:

```
imeListe.clear()
```

Primjer 1: Uklanjanje svih elemenata iz liste.

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

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

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

10.1.7. Kopiranje svih elemenata liste u novu listu

Pomoću metode `copy()` stvara se i vraća nova kopija liste nad kojom je ova metoda pozvana. Ova metoda stvara novu listu s jednakim

vrijednostima kao u originalnoj listi i te dvije liste međusobno su neovisne. Sintaksa kopiranja elemenata liste u novu listu je:

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

Primjer 1: Kopiranje svih elemenata liste u novu listu.

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

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

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

Ovaj način kopiranja elemenata liste, u novu neovisnu listu, izrazito je jednostavan, no ako se ne žele kopirati svi elementi u novu listu, već samo određeni, ovaj način rada ne može se koristiti. U tom slučaju to se radi pomoću petlje kojom se iterira po elementima liste te se elementi koji moraju biti u destinacijskoj listi dodaju u nju pomoću metode `append()`.

10.1.8. Broj pojavljivanja neke vrijednosti u listi

Ako je potrebno dobiti broj pojavljivanja neke određene vrijednosti u listi, može se koristiti metoda `count()`. Povratna vrijednost ove metode je broj pronađenih elemenata predane vrijednosti metodi `count()`. Sintaksa koja vraća broj elemenata određene vrijednosti je:

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

Primjer 1: Broj pojavljivanja vrijednosti „Luka“ u listi.

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

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

```
Izlaz:
      2
```

10.1.9. Proširivanje liste elementima druge liste

Metoda `extend()` proširuje listu nad kojom je pozvana elementima liste koji su joj predani. Sintaksa korištenja je sljedeća:

```
listaKojaSeProsiruje.extend(lista)
```

Primjer 1: Lista imena `lista` proširena je pozivom metode `extend()` novim trima elementima.

```
lista = [1, 2, 3]
print(lista)
lista.extend([4, 5, 6])
print(lista)
```

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

Na ovakav način u listu se može dodati više vrijednosti istovremeno, umjesto da se više puta uzastopno poziva metoda `append()`.

Primjer 2: U nastavku se nalazi primjer koji postojeću listu proširuje elementima druge postojeće liste.

```
lista1 = ["Ivan", "Luka", "Nino", "Jan"]
lista2 = [123, 456]

print("Prije poziva metode extend():")
print(lista1)
print(lista2)

lista2.extend(lista1)

print("Nakon poziva metode extend():")
print(lista1)
print(lista2)
```

```
Izlaz:
      Prije poziva metode extend():
      ['Ivan', 'Luka', 'Nino', 'Jan']
      [123, 456]
      Nakon poziva metode extend():
      ['Ivan', 'Luka', 'Nino', 'Jan']
      [123, 456, 'Ivan', 'Luka', 'Nino', 'Jan']
```

10.1.10. Traženje elementa

Metoda `index()` vraća poziciju, tj. indeks na kojem se tražena vrijednost nalazi. Ako u listi ima više elemenata s traženom vrijednošću, vratit će se indeks prve pronađene vrijednosti. Sintaksa korištenja je sljedeća:

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

Primjer 1: Traženje pozicije elementa „Luka“.

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

```
Izlaz:
      1
```

Primjer 2: Ako tražena vrijednost ne postoji u listi, metoda `index()` će izbaciti iznimku `ValueError` (iznimke se u ovom tečaju ne obrađuju).

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

```
Izlaz:
      ValueError: 'Ivo' is not in list
```

10.1.11. Dodavanje novog elementa na točno određenu poziciju

Pomoću prethodno obrađene metode `append()` u listu dodajemo novu vrijednost, no ta nova vrijednost uvijek se nalazi na zadnjoj poziciji. Ako se nova vrijednost želi staviti na točnu određenu poziciju, tj. indeks, koristi se metoda `insert()`. Sintaksa korištenja metode `insert()` je:

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

Primjer 1: Dodavanje novog elementa na točno određenu poziciju unutar liste.

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

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

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

10.1.12. Uklanjanje elementa iz liste

Za uklanjanje i vraćanje elementa u listu koristi se metoda `pop()`. Ova metoda može se koristiti na dva načina: prvi je da uklanja i vraća zadnji element iz liste, a drugi je da uklanja i vraća element iz liste na predanom indeksu.

Zadnji element u listi uklanja se tako da se metodi `pop()` ne predaje nikakav argument. Sintaksa uklanjanja zadnjeg elementa iz liste je:

```
lista.pop()
```

Primjer 1: Uklanjanje zadnjeg elementa iz liste.

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

print(lista)
zadnji = lista.pop()
print(lista)

print("Uklonjeni element je:", zadnji)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Luka']
      ['Ivan', 'Luka', 'Nino']
      Uklonjeni element je: Luka
```

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

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

Primjer 2: Uklanjanje elementa iz liste na željenom indeksu.

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

print(lista)
e = lista.pop(2)
print(lista)

print("Uklonjeni element je:", e)
```

```
Izlaz:
      ['Ivan', 'Luka', 'Nino', 'Luka']
      ['Ivan', 'Luka', 'Luka']
      Uklonjeni element je: Nino
```

10.1.13. Uklanjanje elementa određene vrijednosti

Za uklanjanje elementa točno određene vrijednosti moguće je koristiti metodu `remove()`. Sintaksa je sljedeća:

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

Primjer 1: Uklanjanje elementa iz liste na željenom indeksu.

```
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()` toliko dugo dok „Luka“ postoji u listi. Broj pozivanja metode `remove()` moguće je kontrolirati pomoću petlje `while`. Petlja se mora okretati toliko 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']
```

10.1.14. Okretanje redoslijeda elemenata u listi

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()
```

Primjer 1: Okretanje redoslijeda elemenata u listi.

```
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]
```

10.1.15. Sortiranje elemenata u listi

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

```
lista.sort()
```

Primjer 1: Sortiranje elemenata u listi.

```
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]
```

10.2. Vježba: Lista

1. Napišite program koji se sastoji od liste. Elementi liste neka budu proizvoljnih vrijednosti. Ispišite elemente liste koji se nalaze na parnim indeksima.
2. Kreirajte praznu listu te s tipkovnice unosite vrijednosti. Tako pročitane vrijednosti potrebno je spremiti u listu. Učitavanje prekinuti onoga trenutka kada korisnik unese broj 5 (5 ne ulazi u listu).
3. Nadogradite prethodni zadatak tako da se, 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 postave na predefiniranu vrijednost, a to je 0.
4. Prethodni zadatak nadogradite tako da nakon završetka upisivanja vrijednosti u listu ispišete sve elemente liste zajedno s njihovim pripadajućim indeksima na temelju sljedećeg primjera ispisa:

```
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 listu imena `dani` s popisom radnih dana u tjednu (od ponedjeljka do petka). U listu unesite dane koji nedostaju (subota i nedjelja) – unos napravite na dva načina: pomoću metode `append()` i metode `extend()`.
6. Napravite dvije liste, jednu imena `radniDani` (od ponedjeljka do petka) te drugu listu imena `dani` (od ponedjeljka do nedjelje). Na temelju tih dviju lista detektirajte dane koji pripadaju u vikend (subota i nedjelja) te njima popunite listu imena `vikend`.
7. Napišite listu `gradovi` s barem pet gradova te ju ispišite. Programski odredite ime grada koji se nalazi na kraju liste. Tako određen grad dodajte na prvu, drugu i treću poziciju (0., 1. i 2. indeks) u listi `gradovi`. Ispišite listu.
8. Nadogradite prethodni zadatak tako da iz liste `gradovi` izbacite sva pojavljivanja zadnjeg grada te ponovo ispišite listu.
9. Napišite program koji učitava elemente liste s tipkovnice. Učitavanje treba prekinuti onoga trenutka kada korisnik unese broj 5. Nakon što je lista učitana, program mora zamijeniti prvi element posljednjim, drugi element pretposljednjim i tako redom.
10. Napišite funkciju koja prima listu. Funkcija vraća `bool` vrijednost istina ako su svi elementi liste parni brojevi, a `bool` vrijednost laž ako postoji barem jedan broj koji nije paran.

10.3. N-torka (engl. *Tuple*)

N-torke pripadaju u slijedne kolekcije i one su neizmjenjive (engl. *immutable*), što znači da n-torka, jednom kada se kreira, svoje elemente više ne može mijenjati, brisati i dodavati. Ova kolekcija objekata koristi se kada se želi postići neizmjenljivost podataka.

U načelu n-torka se ponaša na isti način kao i lista, ali s razlikom da je n-torka nepromjenjiva. Također, liste koriste uglate zagrade, dok se kod pridruživanja vrijednosti n-torci koriste oble zagrade. Unutar n-torke vrijednosti se odvajaju zarezom. Sintaksa za definiranje n-torke je:

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

Ako se želi definirati prazna n-torka, to se radi na sljedeći način:

```
imeNtorke = ()
```

Jedan od češćih primjera korištenja prazne n-torke je kada se očekuje povratna vrijednost neke funkcije u obliku n-torke. Ako funkcija nema elemenata za vraćanje, vraća se prazna n-torka.

Primjer 1: Stvaranje jedne n-torke.

```
ntorka = (False, 1, 'Dva', 3.33)
print(ntorka)
```

```
Izlaz:
      (False, 1, 'Dva', 3.33)
```

Primjer 2: U radu s n-torkama moguće je koristiti funkciju `len()` za dohvaćanje broja elemenata u n-torci, a također i naredbe `in` i `not in`.

```
ntorka = (False, 1, 'Dva', 3.33)

print("Broj elemenata:", len(ntorka))
print("Ima li 'Dva' u ntorci:", "Dva" in ntorka)
```

```
Izlaz:
      Broj elemenata: 4
      Ima li 'Dva' u ntorci: True
```

Primjer 3: N-torke se mogu nadovezivati poput nizova znakova operatorom +=. U narednom primjeru programskog kôda nakon svakog korištenja operatora += ime n-torke `ntorka1` svaki put se povezuje s potpuno novom n-torkom u koju se kopiraju njene prethodne vrijednosti, a potom i nove vrijednosti iz n-torke koja se pridružuje.

```
ntorka1 = (1, 2, 3)
ntorka2 = (4, 5, 6)
ntorka3 = (7, 8, 9)

print(ntorka1)
```

```
ntorka1 += ntorka2
print(ntorka1)

ntorka1 += ntorka3
print(ntorka1)
```

```
Izlaz:
      (1, 2, 3)
      (1, 2, 3, 4, 5, 6)
      (1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Ako se u n-torku želi nadovezati nova vrijednost (na primjer cijeli broj), može se koristiti sljedeća sintaksa: `ntorka += (1, )`. Važno je napomenuti da vrijednost koja se nadovezuje mora biti u oblim zagradama i da se nakon vrijednosti mora staviti zarez, čak i ako se dodaje samo jedna vrijednost. Ovo je zato što Python prepoznaje zarez kao operator za stvaranje n-torke, pa bez zareza Python neće prepoznati da stvarate n-torku, a nadovezivanje je moguće isključivo s istim tipom podataka, u ovom slučaju s n-torkama.

10.3.1. Dohvaćanje vrijednosti iz n-torke

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

```
imeNtorke[indeks]
```

Primjer 1: Dohvaćanje vrijednosti iz n-torke.

```
ntorka = (False, 1, 'Dva', 3.33)
print(ntorka[2])
```

```
Izlaz:
      Dva
```

Primjer 2: Kao što je objašnjeno i u poglavlju s listama, i kod n-torki je moguće dohvatiti vrijednosti od indeksa do indeksa. Sintaksa i detaljniji opis prikazani su u nastavku.

`imeNtorke[start:stop]`

- `start` – indeks od kojeg kreće ispisivanje
- `stop` – vrijednost za 1 broj veća od indeksa zadnjeg elementa koji se ispisuje

Moguće je izostaviti vrijednost `start` ili `stop`. U nastavku se nalazi opis što se dogodi ako se izostavi `start`, a što se dogodi ako se izostavi vrijednost `stop`:

- `start` – dohvaćanje znakova kreće od nultog indeksa
- `stop` – dohvaćanje znakova ide do kraja niza znakova.

```
ntorka = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h')
print(ntorka[1:5])
print(ntorka[:5])
print(ntorka[5:])
```

```
Izlaz:
      ('b', 'c', 'd', 'e')
      ('a', 'b', 'c', 'd', 'e')
      ('f', 'g', 'h')
```

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

```
var1, var2, var3 = nTorka
```

Primjer 3: Raspakiravanje n-torke u zasebne varijable.

```
nTorka = (11, 22, 33)
var1, var2, var3 = nTorka
print(var1, var2, var3)
```

```
Izlaz:
      11 22 33
```

U gornjem primjeru unutar n-torke nalaze se tri vrijednosti. Te vrijednosti spremaju se u tri varijable: `var1`, `var2`, `var3`. Vrijednosti se u varijable spremaju identičnim redoslijedom kako se nalaze unutar n-terca.

10.3.2. Promjena vrijednosti unutar n-torke

Kao što je u uvodu navedeno, n-torka je neizmjenjiva kolekcija objekata. To znači da nakon kreiranja n-torke više nije moguće mijenjati vrijednosti koje se nalaze unutar nje.

Primjer 1: Prikaz kôda u kojem se pokušava promijeniti jedan element n-torke. U izlazu pokrenutoga kôda vidi se da to nije moguće napraviti te da je podignuta iznimka `TypeError`.

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

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

10.3.3. Brisanje n-torke

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

Moguće je „uništiti“ cijelu n-torku. N-torka se uništava pomoću ključne riječi `del`.

Primjer 1: Uništavanje n-torke.

```
nTorka = (11, 22, 33)
print(nTorka)
del nTorka
print("N-torka nakon brisanja nije dohvatljiva!")
print(nTorka)
```

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

U gore napisanom kôdu nad varijablom `nTorka` pozvana je naredba `del`. Nakon brisanja n-torke ta ista n-torka imena `nTorka` pokušava se ispisati, ali to nije moguće jer `nTorka` više ne postoji te se podiže iznimka `NameError`.

10.3.4. Broj pojavljivanja neke vrijednosti u n-torci

Ako je potrebno dobiti broj pojavljivanja neke određene vrijednosti u n-torki, može se koristiti metoda `count()`. Povratna vrijednost ove metode je broj pronađenih elemenata predane vrijednosti metodi `count()`. Sintaksa koja vraća broj elemenata određene vrijednosti je:

```
imeNtorke.count(vrijednost)
```

Primjer 1: Broj pojavljivanja neke vrijednosti u n-torci.

```
nTorka = (1, 2, 3, 3, 3, 4, 4, 5)
print(nTorka.count(1))
print(nTorka.count(3))
```

```
Izlaz:
      1
      3
```

10.3.5. Traženje elementa

Metoda `index()` vraća poziciju, tj. indeks prvog pojavljivanja tražene vrijednosti. Potrebno je obratiti pozornost na to da će, ako u n-torki ima više elemenata s traženom vrijednošću, ova metoda vratiti indeks prve pronađene vrijednosti. Sintaksa korištenja je sljedeća:

```
imeNtorke.index(trazenaVrijednost)
```

Primjer 1: Traženje željenog elementa u n-torci.

```
nTorka = (1, 2, 3, 3, 3, 4, 4, 5)
print(nTorka.index(1))
print(nTorka.index(3))
print(nTorka.index(4))
```

```
Izlaz:
      0
      2
      5
```

Primjer 2: Ako tražena vrijednost ne postoji u n-torki, metoda `index()` izbacit će iznimku `ValueError` (iznimke se u ovom tečaju ne obrađuju).

```
nTorka = (1, 2, 3, 3, 3, 4, 4, 5)

print(nTorka.index(0))
```

```
Izlaz:
      ValueError: tuple.index(x): x not in tuple
```

10.4. Vježba: N-torka

1. Napravite n-torku u kojoj su pohranjeni samoglasnici. Ispišite prva tri samoglasnika.
2. Probajte promijeniti jedan element n-torke.
3. Napišite funkciju koja prima listu cijelih brojeva i vraća n-torku koja se sastoji samo od parnih brojeva. Ako u listi ne postoji nijedan parni broj, funkcija mora vratiti praznu n-torku. U glavnom dijelu programa ispišite dobivenu vrijednost.
4. Napišite funkciju koja prima listu što se sastoji od nekoliko n-torki. U nastavku slijedi primjer takve liste:
   `[('a', 'b', 'c'),( 'a', 'i', 'o'),( 'd', 'e')]`

   Funkcija mora vratiti prvu n-torku iz liste u kojoj se nalaze isključivo samoglasnici (ne moraju se pojaviti svi samoglasnici).
5. * Napišite funkciju koja prima n-torku tako da predani argument n-torke mora sadržavati barem 3 cijela broja. Funkcija vraća n-torku koja se sastoji od najmanje i najveće vrijednosti iz predane n-torke.

10.5. Skup (engl. *Set*)

Skup je asocijativna kolekcija u kojoj su vrijednosti ujedno i ključevi. Unutar skupa svi su elementi jedinstveni tj. skup ne može sadržavati duple objekte. Elementi skupa ne mogu se dohvaćati pomoću indeksnog operatora `[]`.

Ova kolekcija 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 koji su pohranjeni unutar skupa. Postoje dva podatkovna tipa skupova: razred `set` za promjenjive skupove i razred `frozenset` za nepromjenjive skupove. U ovom tečaju obrađuje se razred `set`.

Skup se definira navođenjem elemenata koji su odvojeni zarezom u vitičastim zagradama `{}`. Sintaksa definiranja skupa je:

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

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

```
imeSkupa = set()
```

Primjer 1: U nastavku se nalazi primjer stvaranja jednog skupa.

```
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.

Primjer 2: Petlja `for` može se koristiti za prolazak po svim elementima skupa na identičan način kao što se koristi kod npr. liste.

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

for e in skup:
    print(e)
```

```
Izlaz:
      1
      2
      3
      4
      5
      7
      8
```

Za rad sa skupovima također postoje implementirane metode koje olakšavaju izradu programske logike, a to su metode: `add()`, `clear()`, `copy()`, `difference()`, `difference_update()`, `discard()`, `intersection()`, `intersection_update()`, `isdisjoint()`, `issubset()`, `issuperset()`, `pop()`, `remove()`, `symmetric_difference()`, `symmetric_difference_update()`, `union()`, `update()`. Od ovih metoda u nastavku se obrađuju neke najčešće korištene.

10.5.1. Dodavanje novog elementa u skup

Za dodavanje nove vrijednosti u skup koristi se metoda `add()`. Ako element koji se želi dodati već od prije postoji u skupu, poziv ove metode neće napraviti nikakvu promjenu nad skupom. Sintaksa dodavanja nove vrijednosti je:

```
imeSkupa.add(novaVrijednost)
```

Primjer 1: Ako se u skup želi dodati više elemenata, potrebno je više puta pozvati metodu `add()`.

```
skup = {1, 2, 3, 4}

skup.add(5)
skup.add(6)
skup.add(7)

print(skup)
```

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

10.5.2. Uklanjanje svih elemenata iz skupa

Ako je iz skupa potrebno izbaciti sve vrijednosti, to je moguće napraviti pomoću metode `clear()`. Nakon poziva ove metode nad skupom skup postaje prazan. Sintaksa za uklanjanje svih elemenata iz skupa:

```
imeSkupa.clear()
```

Primjer 1: Uklanjanje svih elemenata iz skupa.

```
skup = {1, 2, 3, 4}

print(skup)
skup.clear()
print(skup)
```

```
Izlaz:
      {1, 2, 3, 4}
      set()
```

10.5.3. Kopiranje svih elemenata skupa u novi skup

Pomoću metode `copy()` stvara se i vraća nova kopija skupa nad kojim je ova metoda pozvana. Ova metoda stvara novi skup s jednakim vrijednostima kao u originalnom skupu i ta dva skupa međusobno su neovisna. Sintaksa kopiranja elemenata skupa u novi skup je:

```
noviSkup = skup.copy()
```

Primjer 1: Kopiranje svih elemenata skupa u novi skup.

```
skup = {1, 2, 3, 4}

noviSkup = skup.copy()
noviSkup.add(5)

print(skup)
print(noviSkup)
```

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

10.5.4. Uklanjanje elementa iz skupa

Za uklanjanje i vraćanje nasumično odabranog elementa iz skupa koristi se metoda `pop()`. Sintaksa uklanjanja nasumičnog elementa iz skupa:

```
imeSkupa.pop()
```

Primjer 1: Uklanjanje nasumično odabranog elementa iz skupa.

```
skup = {1, 2, 3, 4}

x = skup.pop()

print(x)
```

```
Izlaz:
      1
```

10.5.5. Uklanjanje elementa određene vrijednosti

Za uklanjanje elementa točno određene vrijednosti moguće je koristiti dvije metode, metodu `discard()` i metodu `remove()`.

Metoda `discard()` uklanja element predane vrijednosti iz skupa. Ako element koji se želi izbaciti iz skupa ne postoji, ova metoda neće podići iznimku. Sintaksa je sljedeća:

```
imeSkupa.discard(vrijednost)
```

Primjer 1: Uklanjanje elementa iz skupa točno određene vrijednosti.

```
skup = {1, 2, 3, 4}

skup.discard(2)
skup.discard(5)

print(skup)
```

```
Izlaz:
      {1, 3, 4}
```

Metoda `remove()` uklanja element predane vrijednosti iz skupa. Ako element koji se želi izbaciti iz skupa ne postoji, ova metoda izbacuje iznimku `KeyError`. Sintaksa je sljedeća:

```
imeSkupa.remove(vrijednost)
```

Primjer 2: Uklanjanje elementa iz skupa točno određene vrijednosti, ako predani element ne postoji, izbacuje se iznimka.

```
skup = {1, 2, 3, 4}

skup.remove(2)
print(skup)
skup.remove(5)
```

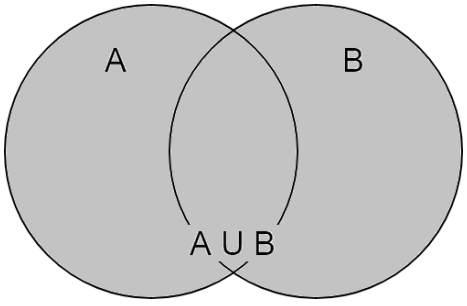
```
Izlaz:
      {1, 3, 4}
      Traceback (most recent call last):
        File "C:\skup.py", line 5, in <module>
          skup.remove(5)
      KeyError: 5
```

10.5.6. Unija

Unija elemenata je skup svih elemenata koji su članovi ili skupa A ili skupa B (ili su članovi obaju skupova). Matematička oznaka unije je A U B.

Unija skupova može se izračunati na dva načina, koristeći metodu `union()` ili koristeći operator `|`.

```
unija = imeSkupa1.union(imeSkupa2)
unija = imeSkupa1 | imeSkupa2
```



Primjer 1: Unija dvaju skupova.

```
# definiranje skupova
skup1 = {1, 2, 3}
skup2 = {3, 4, 5}

# izračun unije skupova - metoda union()
unija = skup1.union(skup2)
print(unija)

# izračun unije skupova - operator |
unija = skup1 | skup2
print(unija)
```

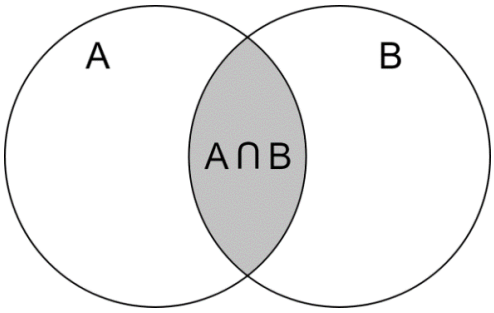
```
Izlaz:
      {1, 2, 3, 4, 5}
      {1, 2, 3, 4, 5}
```

10.5.7. Presjek

Presjek elemenata je skup elemenata koji su članovi i skupa A i skupa B. Matematička oznaka presjeka je A ∩ B.

Presjek skupova može se izračunati na dva načina, koristeći metodu `intersection()` ili koristeći operator `&`.

```
presjek = imeSkupa1.intersection(imeSkupa2)
presjek = imeSkupa1 & imeSkupa2
```



Primjer 1: Presjek dvaju skupova.

```
# definiranje skupova
skup1 = {1, 2, 3, 4}
skup2 = {3, 4, 5, 6}

# izračun presjeka - metoda intersection()
presjek = skup1.intersection(skup2)
print(presjek)

# izračun presjeka - operator &
presjek = skup1 & skup2
print(presjek)
```

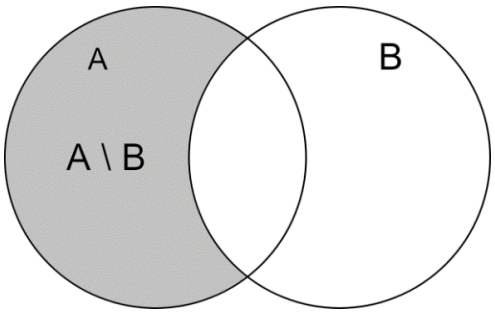
```
Izlaz:
      {3, 4}
      {3, 4}
```

10.5.8. Skupovna razlika

Skupovna razlika skupova A i B je skup svih elemenata koji su članovi skupa A, ali nisu članovi skupa B. Matematička oznaka skupovne razlike je A \ B.

Skupovna razlika može se izračunati na dva načina, koristeći metodu `difference()` ili koristeći operator `-`.

```
skupovnaRazlika = imeSkupa1.difference(imeSkupa2)
skupovnaRazlika = imeSkupa1 - imeSkupa2
```



Primjer 1: Skupovna razlika dvaju skupova.

```
# definiranje skupova
skup1 = {1, 2, 3, 4}
skup2 = {3, 4, 5, 6}

# izračun skupovne razlike - metoda difference()
razlika = skup1.difference(skup2)
print(razlika)
# izračun skupovne razlike - operator -
razlika = skup1 - skup2
print(razlika)
```

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

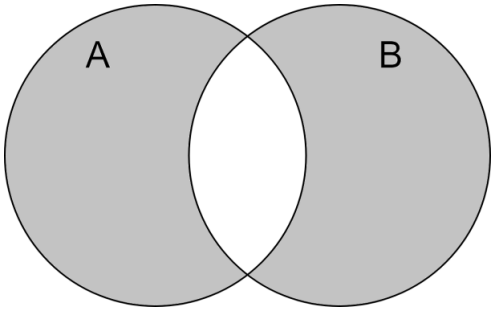
Kod ispisa skupovne razlike (`skup1 - skup2`) vidi se da su rezultat brojevi `{1, 2}`, a to su brojevi koji su se pojavili u skupu imena `skup1`, a nisu se pojavili u skupu imena `skup2`.

10.5.9. Komplement presjeka

Komplement presjeka je skup elemenata koji se nalaze u skupu A ili skupu B, ali se istovremeno ne nalaze i u skupu A i u skupu B. Matematička oznaka komplementa presjeka je (A ∩ B)'.

Skupovna razlika može se izračunati na dva načina, koristeći metodu `symmetric_difference()` ili koristeći operator `^`.

```
kp = imeSkupa1.symmetric_difference(imeSkupa2)
kp = imeSkupa1 ^ imeSkupa2
```



Primjer 1: Komplement presjeka dvaju skupova.

```
# definiranje skupova
skup1 = {1, 2, 3, 4}
skup2 = {3, 4, 5, 6}

# izračun kompelemnta presjeka pomoću metode
kp = skup1.symmetric_difference(skup2)
print(kp)

# izračun kompelemnta presjeka - operator ^
kp = skup1 ^ skup2
print(kp)
```

```
Izlaz:
      {1, 2, 5, 6}
      {1, 2, 5, 6}
```

10.6. Vježba: Skup

1. Napravite 2 skupa podataka i napunite ih proizvoljnim vrijednostima (svaki skup neka se sastoji od minimalno 4 vrijednosti). Naknadno dodajte po još jednu vrijednost u svaki skup. Ispišite tako kreirane skupove
2. Nad prethodno kreiranim skupovima napravite uniju, presjek, skupovnu razliku i komplement presjeka.

10.7. Rječnik (engl. *Dictionary*)

Rječnik je asocijativna kolekcija u kojoj elementi nisu indeksirani ni poredani. Rječnici u Pythonu funkcioniraju po principu ključa. Svaki element izražava se parom: `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 se unutar vitičastih zagrada navode parovi `ključ:vrijednost`, a 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 = {}
```

Primjer 1: Definiranje rječnika.

```
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 skup, osim što se elementi kod rječnika sastoje od para `ključ:vrijednost`.

Potrebno je obratiti pozornost na sljedeće stvari kod rječnika:

- 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 tip podataka koji je neizmjenjiv (engl. *immutable*), na primjer: broj, niz znakova, n-torka (n-torka unutar sebe ne smije sadržavati druge izmjenjive vrijednosti). Izmjenjivi tipovi podataka poput listi ne mogu se koristiti kao ključevi. U jednom te istom rječniku kao ključ moguće je kombinirati različite tipove podataka.
- Rječnik je asocijativna kolekcija te zbog toga redoslijed spremanja ključeva nije predvidiv.

10.7.1. Dohvaćanje vrijednosti iz rječnika

Dohvaćanje vrijednosti iz rječnika radi se tako da se najprije napiše naziv rječnika i zatim se u uglatim zagradama stavi ključ vrijednosti koja se želi dohvatiti. Sintaksa:

```
nazivRjecnika[kljuc]
```

Primjer 1: Dohvaćanje željenih vrijednosti iz rječnika.

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

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

```
Izlaz:
      Veljača
      Lipanj
```

Elementi u rječniku nisu indeksirani zato jer je rječnik asocijativna kolekcija.

10.7.2. Dohvaćanje popisa ključeva

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

```
nazivRjecnika.keys()
```

Primjer 1: Dohvaćanje popisa ključeva 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])
```

10.7.3. Dodavanje novoga para

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

```
nazivRjecnika.update({kljuc:vrijednost})
```

Primjer 1: Dodavanje novog para u rječnik.

```
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])
```

10.7.4. Dohvaćanje popisa vrijednosti

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

```
nazivRjecnika.values()
```

Primjer 1: Dohvaćanje popisa svih vrijednosti spremljenih u rječniku.

```
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'])
```

10.7.5. Brisanje vrijednosti iz rječnika

Ako se neki par `ključ:vrijednost` želi izbrisati iz rječnika, sintaksa je:

```
del nazivRjecnika[kljuc]
```

Primjer 1: Brisanje para vrijednosti iz rječnika.

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

del rjecnik[1]
del rjecnik[2]
del rjecnik[3]
print(rjecnik)
```

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

10.7.6. Dohvaćanje broja elemenata u rječniku

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

Primjer 1: Dohvaćanje broja parova elemenata u rječniku.

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

print(len(rjecnik))
```

```
Izlaz:
      6
```

10.7.7. Iteriranje po elementima rječnika

Po elementima rječnika moguće je iterirati koristeći petlju `for`. Iteracija ide po ključevima te je preko ključa potrebno dohvaćati vrijednosti.

Primjer 1: Iteriranje po elementima rječnika.

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

for kljuc in rjecnik:
    print(rjecnik[kljuc])
```

```
Izlaz:
      Siječanj
      Veljača
      Ožujak
      Travanj
      Svibanj
      Lipanj
```

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 u službenoj dokumentaciji Pythona ili pak pretraživanjem interneta.

10.8. Vježba: Rječnik

1. Napravite rječnik `osoba` s ključevima `ime`, `prezime`, `godine`. Vrijednosti pridružene ključevima odredite sami. Ispišite samo vrijednosti u rječniku koristeći petlju `for`.
2. U rječnik kreiran u prethodnom zadatku dodajte novi par `{ključ:vrijednost}`. Ključ može biti adresa, OIB ili nešto slično.
3. Iz rječnika korištenog u prethodnom zadatku izbrišite element `godine` te ispišite rječnik.
4. Napravite hrvatsko-engleski rječnik. Ključ podataka neka bude hrvatska riječ, a vrijednost toga ključa neka bude engleska riječ. Napunite rječnik s 5 elemenata. Napravite beskonačnu petlju koja s tipkovnice učitava hrvatske riječi. Za svaku učitanu riječ (ako prijevod postoji) treba ispisati prijevod, a ako tražena riječ ne postoji, ispisati poruku da ta riječ ne postoji u rječniku. Učitavanje treba raditi toliko dugo dok se ne unese znak „x“. Potrebno je obratiti pozornost na mala/velika slova. Prijedlog je pretvarati sve u mala slova.
5. 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.

10.9. Pitanja za ponavljanje: Kolekcije objekata

1. Koje kolekcije objekata pripadaju u slijedne kolekcije?
2. Koje kolekcije objekata pripadaju u asocijativne kolekcije?
3. Koja je razlika između liste i n-torke?
4. Na koji je način moguće proširiti n-torku?
5. Mogu li u skupu postojati dva elementa identične vrijednosti?
6. Od čega se sastoji svaki element u rječniku?

Dodatak: Završna vježba

1. * S tipkovnice učitavajte cijele brojeve. Prvi upisani broj može biti bilo koji cijeli broj. Učitavanje treba ponavljati dok god je upisani broj strogo veći od prethodno upisanog broja. Treba ispisati sumu svih učitanih brojeva osim broja zbog kojeg je prekinuto učitavanje.
2. * Učitajte s tipkovnice 2 niza znakova i 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 troznamenkast, 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 naredbe `return`. Kao rezultat potrebno je vratiti zbroj svih brojeva od 0 do unesenog broja. Na primjer, ako je unesena vrijednost -5, funkcija mora vratiti zbroj brojeva: - 5 - 4 - 3 - 2 - 1 = -15, ako je pak unesena vrijednost 4, funkcija mora vratiti zbroj brojeva: 4 + 3 + 2 + 1 = 10. Rezultat ispišite na zaslonu.
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 treba ponavljati toliko 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 toliko 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 zaslonu. 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 treba ispisati prikladnu poruku na ekran te zatražiti ponovni unos cijelog broja. Nakon učitavanja 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. 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}

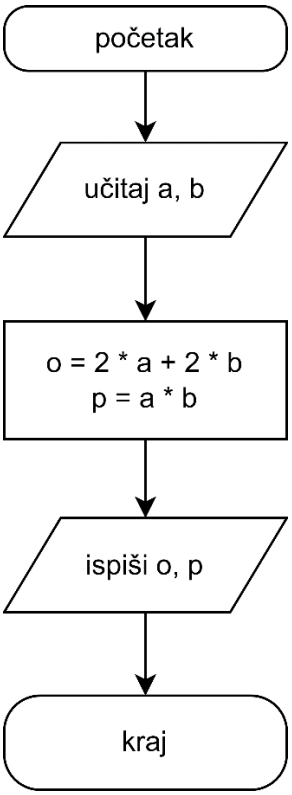
    Unesite vrijednost n: {vrijednost}

    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

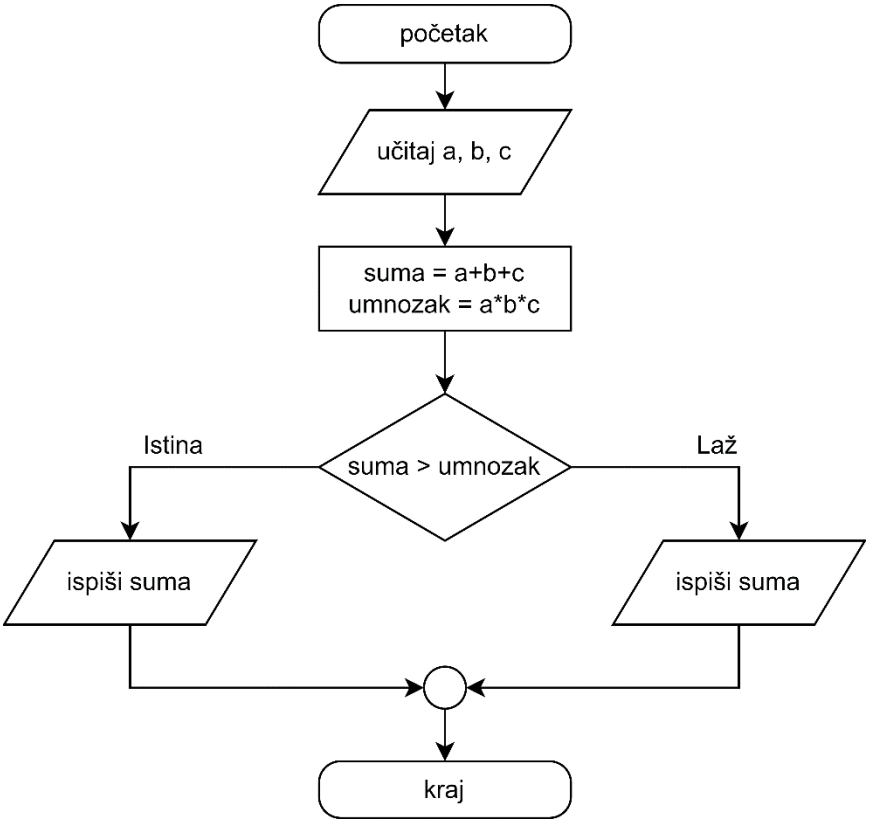
1.4.1.



1.4.2.

```
početak

učitaj(a, b)

o = 2 * a + 2 * b
p = a * b

ispiši(o, p)

kraj
```

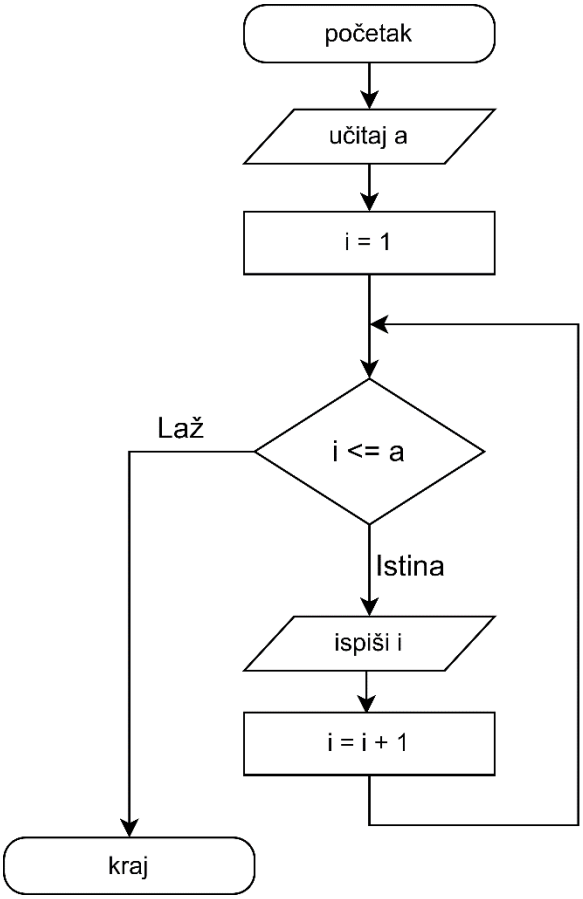
1.4.3.



1.4.4.

```
početak

učitaj(a, b, c)

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

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

kraj
```

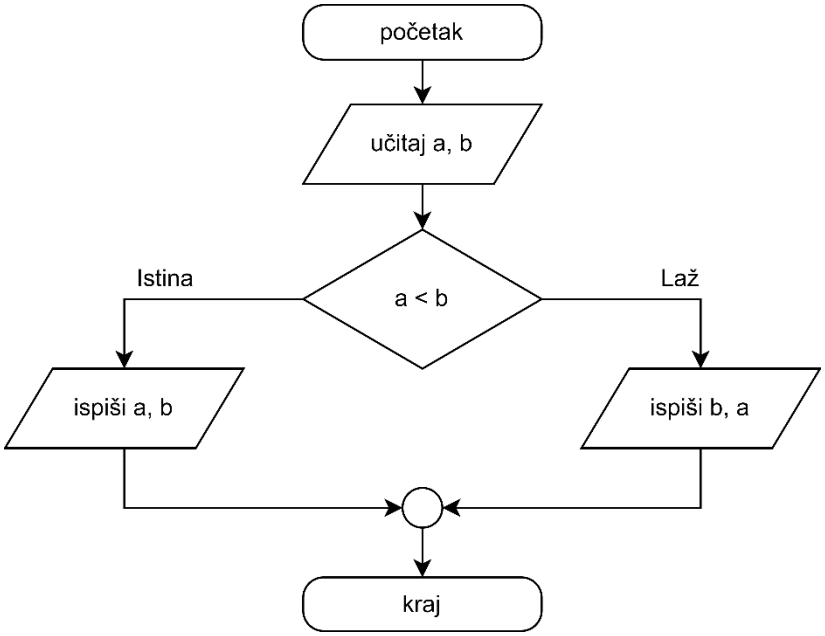
1.4.5.



1.4.6.

```
početak

učitaj(a)

i = 1

dok je i <= a raditi
      ispiši(i)
      i = i + 1

kraj
```

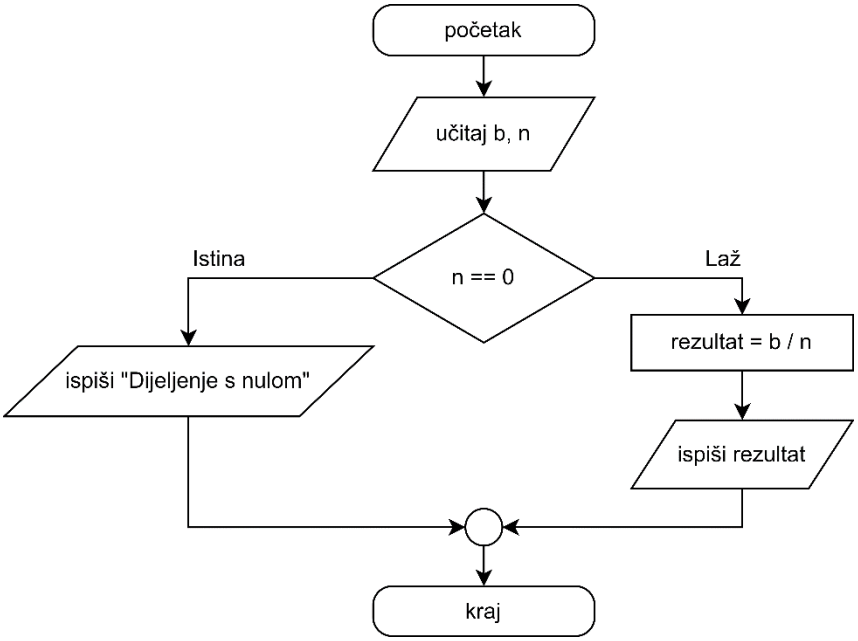
1.4.7.



1.4.8.

```
početak

učitaj(a, b)

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

kraj
```

1.4.9.



1.4.10.

```
početak

učitaj(a, b)

ako je n == 0 tada
    ispiši("Dijeljenje s nulom")
inače
    rezultat = b / n
    ispiši(rezultat)

kraj
```

1.5.1.

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

1.5.2.

Za uvjetno grananje koristi se romb.

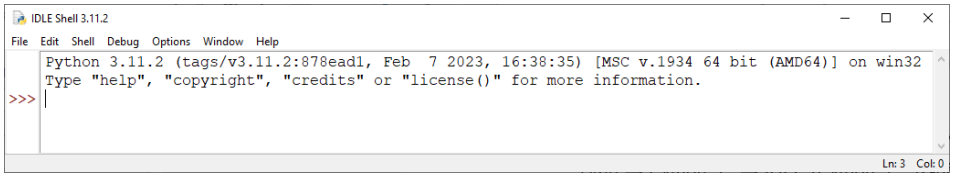
1.5.3.

Glavna prednost pseudokôda pred dijagramom toka jest čitljivost prilikom opisivanja kompleksnijih algoritama.

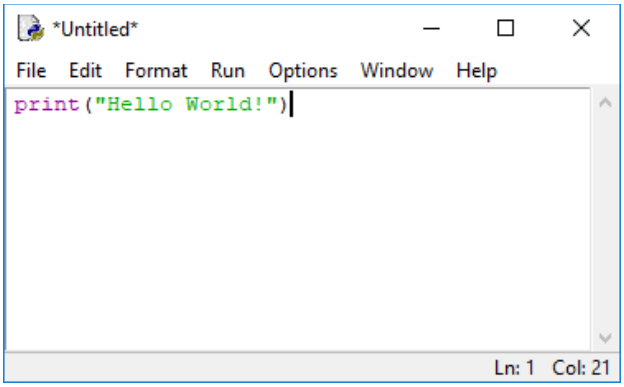
2.4.

Otvoriti IDLE:

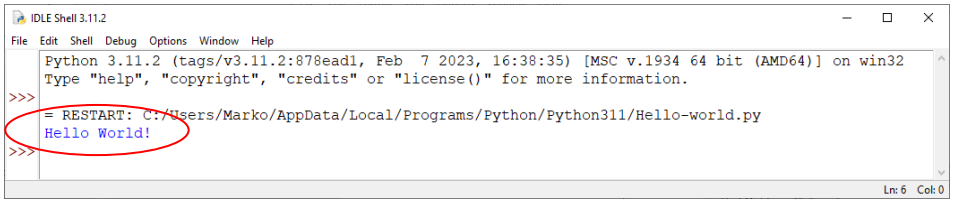
Start → Python 3. → IDLE (Python 3.7 64-bit)*



Nakon toga pritisnuti ***File*** pa ***New File***. Unutar prozora koji se otvori moguće je pisati programski kod.



Nakon što je programski kôd napisan, potrebno ga je spremiti. Programski kôd sprema se pritiskom na ***File*** pa na ***Save***. Nakon što je kôd spremljen u datoteku napisani kôd moguće je pokrenuti, a 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()`.



3.5.1.

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

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

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

3.5.2.

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

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

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

3.5.3.

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

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

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

3.5.4.

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

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

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

3.5.5.

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

"""
Višelinijski komentar
Komentar
kroz
više
linija
"""

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

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

3.7.1.

```
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
```

3.7.2.

```
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
```

3.7.3.

```
a = 10
b = 15

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

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

3.7.4.

```
a = True
b = False

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

```
Izlaz:
      False
      True
      False
```

3.7.5.

```
stupnjevi = 24

farenhajt = (stupnjevi * 9 / 4) + 32

print(farenhajt)
```

```
Izlaz:
      86.0
```

3.7.6.

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

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

print(arSred)
```

```
Izlaz:
      12.75
```

3.7.7.

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

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

arSred = arSred // 1

kvadrat = arSred ** 2

print(kvadrat)
```

```
Izlaz:
      144.0
```

3.7.8.

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

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

arSred = arSred // 1

kvadrat = arSred ** 2
```

```
kvadrat *= 100

print(kvadrat)
```

```
Izlaz:
      14400.0
```

3.7.9.

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

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

arSred = arSred // 1

kvadrat = arSred ** 2

kvadrat *= 100

print(kvadrat < 500)
```

```
Izlaz:
      False
```

3.7.10.

```
a = 12
b = 34

tmp = a
a = b
b = tmp

a = a % 10
b %= 10

print(a, b)
```

```
Izlaz:
      4 2
```

3.14.1.

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

print(niz)
```

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

3.14.2

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

print(niz[9:16])
```

```
Izlaz:
      Croatia
```

3.14.3.

```
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
```

3.14.4.

```
a = 10
b = 30.5
c = -10

print(int(b))
print(float(a))
print(abs(c))
print(max(a, b, c))
print(min(a, b, c))
```

```
Izlaz:
      30
      10.0
      10
      30.5
      -10
```

3.14.5.

```
niz = "Hello WORLD!"
print(len(niz))
print(niz.capitalize())
print(niz.title())
print(niz.lower())
print(niz.upper())

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

```
Izlaz:
      12
      Hello world!
      Hello World!
      hello world!
      HELLO WORLD!
      Hello WORLD!   .
         Hello WORLD!.
      Hello WORLD!.
```

3.14.6.

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

duljina = len(niz)

# start i stop moraju biti cijeli brojevi
print(niz[0:int(duljina / 2)])
```

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

3.15.1.

Postoje jednolinijski i višelinijski komentari.

3.15.2.

Ključna riječ ne može se koristiti kao ime varijable

3.15.3.

Kod cjelobrojnih vrijednosti u Pythonu ne dolazi do gubitka preciznosti.

3.15.4.

Kod realnih vrijednosti u Pythonu može doći do gubitka preciznosti.

3.15.5.

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

3.15.6.

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

3.15.7.

Automatski će se ispisati predefinirana vrijednost parametra `end`, a to je novi redak.

3.15.8.

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

3.15.9.

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

3.15.10.

Nad cjelobrojnom vrijednosti napravi se operacija %2 i, ako je rezultat 0, broj je paran, a ako je rezultat 1, broj je neparan.

4.2.1.

```
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
```

4.2.2.

```
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.")
elif a == 100 and b == 100:
    print("Obje vrijednosti su jednake 100.")
```

```
else:
    print("Jedna je 100, druga je manja ili veća od
100.")
```

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

4.2.3.

```
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.
```

4.2.4.

```
a = 200
b = 20

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

```
Izlaz:
     Zadovoljava.
```

4.2.5.

```
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.
```

4.2.6.

```
a1 = 5
a2 = 10

b1 = 6
b2 = 9

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

```
Izlaz:
     Zadovoljava.
```

4.6.1.

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

```
Izlaz:
     130
     260
     390
     520
     650
     780
     910
```

4.6.2.

```
i = 1
while i <= 1000:
    if i % 2 == 0 and i % 5 == 0 and i % 13 == 0:
        print(i)
    i += 1
```

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

```
780
910
```

4.6.3.

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

brojac = 0
pronadenoA = False

for e in niz:
    if e == "A":
        pronadenoA = True

    if e >= "A" and e <= 'Z':
        brojac += 1

if pronadenoA == True:
    print("U nizu se nalazi veliko slovo A!")

print("Broj velikih slova u nizu:", brojac)
```

```
Izlaz:
U nizu se nalazi veliko slovo A!
Broj velikih slova u nizu: 6
```

4.6.4.

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

brojac = 0
pronadenoA = False
i = 0

while i < len(niz):
    if niz[i] == "A":
        pronadenoA = True

    if niz[i] >= "A" and niz[i] <= 'Z':
        brojac += 1

    i += 1

if pronadenoA == True:
    print("U nizu se nalazi veliko slovo A!")

print("Broj velikih slova u nizu:", brojac)
```

```
Izlaz:
U nizu se nalazi veliko slovo A!
Broj velikih slova u nizu: 6
```

4.6.5.

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

if n < duljina:
    i = 0
    while i < duljina:
        print(niz[i], end="")
        i += n
else:
    print("Greska!")
```

```
Izlaz:
     ACEGIK
```

4.6.6.

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

for n in range(a, b + 1):

    jePrim = True

    for e in range(2, n):
        if n % e == 0:
            jePrim = False
            break

    if jePrim == True:
        brojac += 1

print(brojac)
```

```
Izlaz:
     4
```

4.6.7.

```
n = 5

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:
     10000
     01000
     00100
     00010
     00001
```

4.6.8.

```
n = 10
x = 4
y = 6

for i in range(1, n + 1):  # Određuje redak

    for j in range(1, n + 1):  # Određuje stupac
        if i == y and j == x:
            print("X", end='')
        else:
            print("-", end='')

    print()
```

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

4.6.9.

```
broj = 159
suma = 0

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

print(suma)
```

```
Izlaz:
     15
```

4.7.1.

Uvjetno izvođenje omogućava, ovisno o rezultatu logičkog izraza, izvršavanje željenog segmenta programskog kôda.

4.7.2.

Petlje omogućavaju uzastopno ponavljanje nekog skupa naredbi.

4.7.3.

Petlja `while` okreće se toliko dugo dok je logički uvjet zadovoljen.

4.7.4.

Petlja `for` iterira kroz elemente zadane sekvence (objekt po kojem petlja `for` radi iteraciju mora biti pobrojiv ili iterabilan).

4.7.5.

Naredba `break` prekida izvršavanje prve najbliže petlje.

4.7.6.

Naredba `continue` preskače sav programski kôd koji se nalazi napisan ispod naredbe `continue` u petlji te se odlazi na sljedeću iteraciju petlje.

5.2.1.

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

print(a, b)
```

```
Izlaz:
      Unesite prvi niz znakova: Grad
      Unesite drugi niz znakova: Zagreb
      Grad Zagreb
```

5.2.2.

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

if dan >= 1 and dan <= 31 and mjesec >= 1 and
mjesec <= 12 and godina >= 0 and godina <= 2024:
    if mjesec == 1:
        mjesecNaziv = "siječnja"
    elif mjesec == 2:
        mjesecNaziv = "veljače"
    elif mjesec == 3:
        mjesecNaziv = "ožujka"
    elif mjesec == 4:
        mjesecNaziv = "travnja"
    elif mjesec == 5:
        mjesecNaziv = "svibnja"
    elif mjesec == 6:
        mjesecNaziv = "lipnja"
    elif mjesec == 7:
        mjesecNaziv = "srpnja"
    elif mjesec == 8:
        mjesecNaziv = "kolovoza"
    elif mjesec == 9:
        mjesecNaziv = "rujna"
    elif mjesec == 10:
        mjesecNaziv = "listopada"
    elif mjesec == 11:
        mjesecNaziv = "studenog"
    elif mjesec == 12:
        mjesecNaziv = "prosinca"

    print(dan, ". ", mjesecNaziv, ", ", godina,
".", sep="")
else:
    print("Greška")
```

```
Izlaz:
      Dan: 20
      Mjesec: 3
      Godina: 2023
      20. ožujka, 2023.
```

5.2.3.

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

    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:
     Broj bodova: 50
     Dovoljan!
     Broj bodova: 87.5
     Odličan!
     Broj bodova: 100
     Odličan!
     Broj bodova: 101
     Uneseni broj bodova je neispravan!
```

5.2.4.

```
while True:
    n = int(input("Unesite 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
```

5.3.1.

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

5.3.2.

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

5.3.3.

Povratni tip podataka funkcije `input()` je `str`, tj. niz znakova.

6.7.1.

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


ispis()
```

```
Izlaz:
      Hello World!
```

6.7.2.

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


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

```
Izlaz:
      1.5
```

6.7.3.

```
def izracun(a, b=20, c=30, d=0):
    print(((a * a) + (b * c) - d) / 2)


izracun(1)
izracun(1, 2)
izracun(1, 2, 3)
izracun(1, 2, 3, 4)
izracun(1, c=5)
izracun(1, c=1, b=2, d=5)
```

```
Izlaz:
      300.5
      30.5
      3.5
      1.5
      50.5
      -1.0
```

6.7.4.

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

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

```
Izlaz:
      500
```

6.7.5.

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


a = 10
b = 20

rezultat = suma()
print(rezultat)
```

```
Izlaz:
      30
```

6.7.6.

```
def suma():
    global rezultat
    rezultat = a + b


a = 10
b = 20

rezultat = 0
suma()
print(rezultat)
```

```
Izlaz:
      30
```

6.7.7.

```
def fakt(x):
    umnozak = 1

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

    return umnozak


n = 5
m = 10

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

```
Izlaz:
      252.0
```

6.8.1.

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije.

6.8.2.

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

6.8.3.

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)`

6.8.4.

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

6.8.5.

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

6.8.6.

Globalne varijable iz glavnog dijela programa vidljive su u funkcijama.

7.2.1.

```
pi = 3.1415926

print("%.3f" % pi)
```

```
Izlaz:
      3.141
```

7.2.2.

```
ime = "Pero"
prezime = "Perić"
godine = 19

print("01234567890123456789012345678 9")
print("%10s%10s%10s" % (ime, prezime, godine))
```

```
Izlaz:
      01234567890123456789012345678 9
            Pero      Perić         19
```

7.2.3.

```
print("012345678901234567890123456789012345")
print("Naziv     OS    P     N     I     B")
print("%-10s%-5d%-5d%-5d%-5d%-5d" % ("K1", 26, 10, 5, 12, 35))
print("%-10s%-5d%-5d%-5d%-5d%-5d" % ("K2", 26, 9, 5, 12, 32))
print("%-10s%-5d%-5d%-5d%-5d%-5d" % ("K3", 26, 4, 12, 10, 24))
```

```
Izlaz:
      01234567890123456789012345678901234 5
      Naziv     OS    P     N     I     B
      K1        26    10    5     12    35
      K2        26    9     5     12    32
      K3        26    4     12    10    24
```

7.4.1.

```
pi = 3.1415926

print("{:.3f}".format(pi))
```

```
Izlaz:
      3.141
```

7.4.2.

```
ime = "Pero"
prezime = "Perić"
godine = 19

print("01234567890123456789012345678 9")
```

```
print("{:>10s}{:>10s}{:10d}"
          .format(ime, prezime, godine))
```

```
Izlaz:
      01234567890123456789012345678 9
            Pero      Perić         19
```

7.4.3.

```
print("012345678901234567890123456789012345")
print("Naziv      OS    P     N     I     B")
print("{:10s}{:<5d}{:<5d}{:<5d}{:<5d}{:<5d}"
          .format("K1", 26, 10, 5, 12, 35))
print("{:10s}{:<5d}{:<5d}{:<5d}{:<5d}{:<5d}"
          .format("K2", 26, 9, 5, 12, 32))
print("{:10s}{:<5d}{:<5d}{:<5d}{:<5d}{:<5d}"
          .format("K3", 26, 4, 12, 10, 24))
```

```
Izlaz:
      012345678901234567890123456789012345
      Naziv      OS    P     N     I     B
      K1         26    10    5     12    35
      K2         26    9     5     12    32
      K3         26    4     12    10    24
```

7.6.1.

```
pi = 3.1415926

print(f"{pi:.3f}")
```

```
Izlaz:
      3.141
```

7.6.2.

```
ime = "Pero"
prezime = "Perić"
godine = 19

print("01234567890123456789012345678 9")
print(f"{ime:>10s}{prezime:>10s}{godine:10d}")
```

```
Izlaz:
      01234567890123456789012345678 9
            Pero      Perić         19
```

7.6.3.

```
print("0123456789012345678 9")


for i in range(95, 102):
    print(f"{i:5d}{i + 1:5d}{i + 2:5d}{i + 3:5d}")
```

```
Izlaz:
     01234567890123456789
        95   96   97   98
        96   97   98   99
        97   98   99  100
        98   99  100  101
        99  100  101  102
       100  101  102  103
       101  102  103  104
```

7.7.1.

Uz korištenje funkcije `print()`, postoji operator formatiranja `%`, metoda `format()` i način formatiranja *f-strings*.

7.7.2.

Kod operatora formatiranja `%` najčešće se koriste oznake: `%s`, `%d`, `%f`, `%e`, `%c`, `%g`.

7.7.3.

Za ispis realnog broja na dvije decimale koristi se sljedeća oznaka:

- operator formatiranja `%`: `%.2f`

- metoda `format()`: `{:.2f}`

- f-strings: `{varijabla:.2f}`

7.7.4.

Za poravnanje nizova znakova koriste se sljedeće oznake:

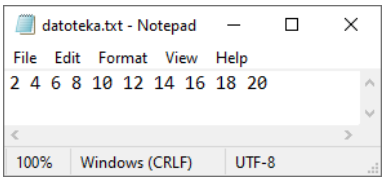
- operator formatiranja `%`: `-` (za formatiranje ulijevo)

- metoda `format()`: `<`, `>`, `^`

- f-strings: `<`, `>`, `^`

8.9.1.

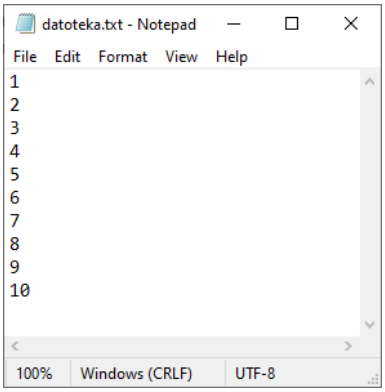
```
with open("datoteka.txt", "w") as f:
    i = 2
    while i <= 20:
        f.write(str(i) + " ")
        i += 2
```



8.9.2.

```
with open("datoteka.txt", "w+") as f:
    i = 1
    while i <= 10:
        f.write(str(i) + "\n")
        i += 1

    f.seek(0, 0)
    suma = 0

    for i in f:
        suma += int(i)

    print(suma)
```

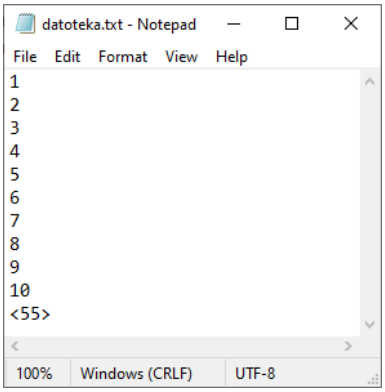
```
Izlaz:
     55
```



8.9.3.

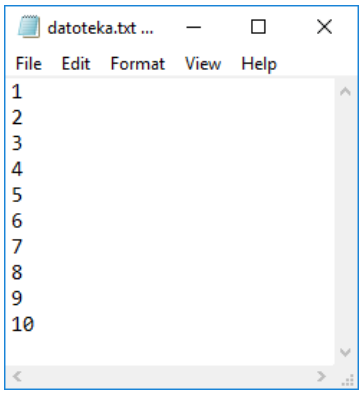
```
with open("datoteka.txt", "w+") as f:
    i = 1
    while i <= 10:
        f.write(str(i) + "\n")
        i += 1

    f.seek(0, 0)
    suma = 0
```

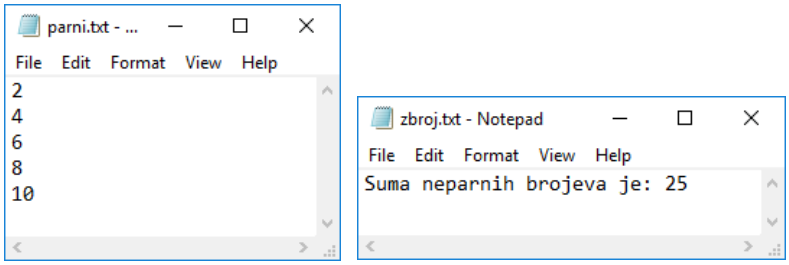
```
    for i in f:
        suma += int(i)

    f.write("<" + str(suma) + ">")
```



8.9.4.

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

sumaNeparnihBrojeva = 0

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

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

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





8.10.1.

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

8.10.2.

Mode koji služi za brisanje sadržaja postojeće datoteke je `w`.

8.10.3.

Mode koji služi za čitanje i pisanje u datoteku uz postojeći sadržaj je `r+`.

8.10.4.

Mode koji služi za pisanje novog sadržaja isključivo na kraj datoteke je `a`.

9.2.1.

```
import math

r = int(input("Polumjer (mm): "))

o = 2 * r * math.pi
p = r ** 2 * math.pi

print("Opseg (mm) =", o)
print("Površina (mm^2) =", p)
```

```
Izlaz:
      Polumjer (mm): 10
      Opseg (mm) = 62.83185307179586
      Površina (mm^2) = 314.1592653589793
```

9.2.2.

```
from math import pi

r = int(input("Polumjer (mm): "))

o = 2 * r * pi
p = r ** 2 * pi

print("Opseg (mm) =", o)
print("Površina (mm^2) =", p)
```

```
Izlaz:
      Polumjer (mm): 10
      Opseg (mm) = 62.83185307179586
      Površina (mm^2) = 314.1592653589793
```

9.2.3.

```
import math

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

print(rez)
```

```
Izlaz:
      Broj za korjenovanje: 16
      4.0
```

9.2.4.

```
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: 5
      49.0
```

9.2.5.

```
import math

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

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

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

```
Izlaz:
      Broj: 5.67
      5, 6
```

9.3.1.

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

9.3.2.

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

10.2.1.

```
lista = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

i = 0
while i < len(lista):
    print(lista[i])
    i += 2
```

```
Izlaz:
     a
     c
     e
     g
```

10.2.2.

```
lista = []

while True:
    broj = int(input("Unesi broj: "))

    if broj == 5:
        break
    else:
        lista.append(broj)

print(lista)
```

```
Izlaz:
     Unesi broj: 1
     Unesi broj: 2
     Unesi broj: 3
     Unesi broj: 4
     Unesi broj: 5
     [1, 2, 3, 4]
```

10.2.3.

```
lista = []

while True:
    broj = int(input("Unesi broj: "))

    if broj == 5:
        break
    else:
        lista.append(broj)

while len(lista) < 6:
    lista.append(0)

print(lista)
```

```
Izlaz:
```

```
Unesi broj: 1
Unesi broj: 3
Unesi broj: 5
[1, 3, 0, 0, 0, 0]
```

10.2.4.

```
lista = []

while True:
    broj = int(input("Unesi broj: "))

    if broj == 5:
        break
    else:
        lista.append(broj)

while len(lista) < 6:
    lista.append(0)

i = 0
while i < 6:
    print("[", i, "] = ", lista[i], sep='')
    i += 1
```

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

10.2.5.

```
dani = ['PON', 'UTO', 'SRI', 'ČET', 'PET']

dani.append('SUB')
dani.append('NED')
print(dani)

dani = ['PON', 'UTO', 'SRI', 'ČET', 'PET']
dani.extend(['SUB', 'NED'])
print(dani)
```

```
Izlaz:
['PON', 'UTO', 'SRI', 'ČET', 'PET', 'SUB',
'NED']
['PON', 'UTO', 'SRI', 'ČET', 'PET', 'SUB',
'NED']
```

10.2.6.

```
dani = ['PON', 'UTO', 'SRI', 'ČET', 'PET', 'SUB',
'NED']
radniDani = ['PON', 'UTO', 'SRI', 'ČET', 'PET']
vikend = []

for dan in dani:
    if dan not in radniDani:
        vikend.append(dan)

print(vikend)
```

```
Izlaz:
     ['SUB', 'NED']
```

10.2.7.

```
lista = ["OS", "ST", "DU", "RI", "ZG"]
print(lista)

n = len(lista)
zadnji = lista[n - 1]
lista.insert(0, zadnji)
lista.insert(1, zadnji)
lista.insert(2, zadnji)
print(lista)
```

```
Izlaz:
     ['OS', 'ST', 'DU', 'RI', 'ZG']
     ['ZG', 'ZG', 'ZG', 'OS', 'ST', 'DU', 'RI', 'ZG']
```

10.2.8.

```
lista = ["OS", "ST", "DU", "RI", "ZG"]
print(lista)

n = len(lista)
zadnji = lista[n - 1]
lista.insert(0, zadnji)
lista.insert(1, zadnji)
lista.insert(2, zadnji)
print(lista)

while zadnji in lista:
    lista.remove(zadnji)

print(lista)
```

```
Izlaz:
     ['OS', 'ST', 'DU', 'RI', 'ZG']
     ['ZG', 'ZG', 'ZG', 'OS', 'ST', 'DU', 'RI', 'ZG']
     ['OS', 'ST', 'DU', 'RI']
```

10.2.9.

```
lista = []

while True:
    a = int(input("Unesi broj: "))
    if a == 5:
        break
    else:
        lista.append(a)

print(lista)

lista.reverse()
print("Okrenuta lista:", lista)

lista.sort()
print("Sortirana lista:", lista)
```

```
Izlaz:
     Unesi broj: 1
     Unesi broj: 9
     Unesi broj: 2
     Unesi broj: 8
     Unesi broj: 3
     Unesi broj: 7
     Unesi broj: 4
     Unesi broj: 6
     Unesi broj: 5
     [1, 9, 2, 8, 3, 7, 4, 6]
     Okrenuta lista: [6, 4, 7, 3, 8, 2, 9, 1]
     Sortirana lista: [1, 2, 3, 4, 6, 7, 8, 9]
```

10.2.10.

```
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!
```

10.4.1.

```
samoglasnici = ('a', 'e', 'i', 'o', 'u')

print(samoglasnici[:3])
```

```
Izlaz:
      ('a', 'e', 'i')
```

10.4.2.

```
samoglasnici = ('a', 'e', 'i', 'o', 'u')

samoglasnici[0] = 'x'
```

```
Izlaz:
      TypeError: 'tuple' object does not support
      item assignment
```

10.4.3.

```
def parniBrojevi(l):
    ntorkaParnih = ()
    for e in l:
        if e % 2 == 0:
            ntorkaParnih += (e,)

    return ntorkaParnih


lista = [1, 2, 3, 4, 5, 6, 7, 8, 9]
parni = parniBrojevi(lista)
print(parni)
```

```
Izlaz:
      (2, 4, 6, 8)
```

10.4.4.

```
def samoSamoglasnici(l):
    for nTorka in l:
        zastavica = True
        for e in nTorka:
            if e != 'a' and e != 'e' and e != 'i'
              and e != 'o' and e != 'u':
                zastavica = False
                break
        if zastavica == True:
            return nTorka

    return ()
```

```
lista = [('a', 'b', 'c'),('a', 'i', 'o'),('d',
'e')]
samoglasnici = samoSamoglasnici(lista)
print(samoglasnici)
```

```
Izlaz:
     ('a', 'i', 'o')
```

10.4.5.

```
def detekcija(l):
    if len(l) < 3:
        print("Lista ima manje od 3 elementa.")
        return ()

    najmanji = l[0]
    najveci = l[0]

    for e in l:
        if e < najmanji:
            najmanji = e
        if e > najveci:
            najveci = e

    return (najmanji, najveci)


lista = [1, 0, 2, 3, 4, 5, 6, 7, 8, 10, 9]
najmanjiNajveci = detekcija(lista)
print(najmanjiNajveci)
```

```
Izlaz:
     (0, 10)
```

10.6.1.

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

skup1.add(5)
skup2.add(8)

print(skup1)
print(skup2)
```

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

10.6.2.

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

unija = skup1.union(skup2)
presjek = skup1.intersection(skup2)
razlika = skup1.difference(skup2)
komplementPresjeka = skup1.symmetric_difference(skup2)

print("Unija:", unija)
print("Presjek:", presjek)
print("Razlika:", razlika)
print("Komplement presjeka:", komplementPresjeka)
```

```
Izlaz:
      Unija: {1, 2, 3, 4, 5, 6, 7, 8}
      Presjek: {4, 5}
      Razlika: {1, 2, 3}
      Komplement presjeka: {1, 2, 3, 6, 7, 8}
```

10.8.1.

```
osoba = {'ime': 'Pero', 'prezime': 'Horvat',
         'godine': 20}

for e in osoba.values():
    print(e)
```

```
Izlaz:
      Pero
      Horvat
      20
```

10.8.2.

```
osoba = {'ime': 'Pero', 'prezime': 'Horvat',
         'godine': 20}

osoba.update({'oib': '12345678900'})

for e in osoba.values():
    print(e)
```

```
Izlaz:
     Pero
     Horvat
     20
     12345678900
```

10.8.3.

```
osoba = {'ime': 'Pero', 'prezime': 'Horvat',
         'godine': 20}

osoba.update({'oib': '12345678900'})

del osoba['godine']

for e in osoba.values():
    print(e)
```

```
Izlaz:
     Pero
     Horvat
     12345678900
```

10.8.4.

```
rjecnik = {"stol": "desk", "sat": "clock",
           "penkala": "pencil", "remen": "belt",
           "svjetlo": "light"}

while True:
    rijec = input("HR: ")
    rijec = rijec.lower()

    if rijec == "x":
        print("Kraj programa.")
        break

    if rijec in rjecnik:
        print("EN:", rjecnik[rijec])
    else:
        print("EN: NEMA!")
```

```
Izlaz:
```

```
      HR: stol
      EN: desk
      HR: SAT
      EN: clock
      HR: vlak
      EN: NEMA!
      HR: x
      Kraj programa.
```

10.8.5.

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

for povrtnica in povrce.keys():
    print(povrtnica.title(), ":",
len(povrce[povrtnica]))
```

```
Izlaz:
      Krumpir : 3
      Luk : 2
```

10.9.1.

U sljedne kolekcije objekata pripadaju lista i n-terac.

10.9.2.

U asocijativne kolekcije objekata pripadaju skup i rječnik.

10.9.3.

U načelu, n-torka se ponaša isto kao lista, ali s jednom bitnom razlikom. Razlika je ta da je n-torka nepromjenjiva (engl. *immutable*), za razliku od liste koja je promjenjiva.

10.9.4.

N-torka se može proširiti dodavanjem pomoću operatora `+=`, no treba imati na umu da takvo proširivanje ne mijenja postojeću n-torku, već kreira potpunu novu n-torku.

10.9.5.

U skupu ne mogu postojati dva elementa identične vrijednosti. Svi elementi u skupu su jedinstveni.

10.9.6.

Svaki element sastoji se od para: `ključ:vrijednost`.

Dodatak: Završna vježba

1.

```
vrijednost = int(input("Unesite broj: "))
suma = vrijednost

while True:
    prethodni = vrijednost

    vrijednost = int(input("Unesite broj: "))

    if vrijednost <= prethodni:
        break

    suma += 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 = int(input("Unesite troznamenkasti broj: "))

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

    return suma


najmanji = 0
najmanjaSuma = -1

while True:
    x = int(input("Unesite broj: "))

    if x <= 0:
        break

    s = sumaZnamenki(x)
    if najmanjaSuma == -1 or najmanjaSuma > s:
        najmanjaSuma = s
        najmanji = x

print("Broj: ", najmanji)
print("Najmanja suma znamenki: ", 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 = int(input("Unesite cijeli broj: "))
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 = int(input("Unesite broj operacije: "))

    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: Sir ima miris
     Uneseni niz znakova je palindrom!
```

```
Izlaz 2:
     Unesite niz znakova: Sir ima mirisssss
     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, 2nd Edition*, O'Reilly Media, 2015.
2. David Beazley, Brian Jones, *Python Cookbook, 3rd Edition*, O'Reilly Media, 2013.
3. John Paul Mueller, *Beginning Programming with Python For Dummies*, 2nd Edition, Wiley, 2018.
4. Leo Budin, Predrag Brođanac, Zlatka Markučič, Smiljana Perić, Dejan Škvorc, Magdalena Babić, *Računalno razmišljanje i programiranje u Pythonu*, Element, 2017.
5. Leo Budin, Predrag Brođanac, Zlatka Markučič, Smiljana Perić, *Rješavanje problema programiranjem u Pythonu*, Element, 2012.
6. Mark Lutz, *Programming Python, 4th Edition*, O'Reilly Media, 2010.
7. *Python 3 Documentation*, https://docs.python.org/3/, dohvaćeno 28.04.2023.
8. Zoran Kalafatić, Antonio Pošćić, Siniša Šegvić, Julijan Šribar, *Python za znatiželjne*, Element, 2016.

Bilješke: