Priručnik

„Programiranje”

Zagreb, 2018. godina

e-Škole
USPOSTAVA SUSTAVA RAZVOJA DIGITALNO ZRELIH ŠKOLA
(PILOT PROJEKT)

CARNET
znanje povezuje

Europska unija
Zajedno do fondova EU

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Sadržaj:

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Europska unija
Zajedno do fondova EU

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Značenje oznaka u tekstu:

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Sažetak

Algoritmi i algoritamsko razmišljanje osnova su programiranja. Jedini je način za učenje algoritama njihova razrada u nekom programskom jeziku jer se većina pojmova ne može naučiti iz teorije i „na papiru“. U našoj e-radionici oslanjamo se na Python kao programski jezik te primjerima i pojašnjenjima povezujemo algoritme i osnovne strukture podataka s načinom na koji su oni realizirani u Pythonu. Ovakvim će pristupom polaznik dobiti pregled onoga što ga očekuje u svijetu programiranja, ali i ideju kako Python upotrijebiti ne samo za učenje nego i kasnije, u trenutku kada mu zatreba alat da svoje ideje pretvori u funkcionalni programski kod.

Ovaj priručnik izrađen je za potrebe radionice „Programiranje”, izrađene u sklopu projekta „e-Škole: Uspostava sustava razvoja digitalno zrelih škola (pilot projekt).

Europska unija
Zajedno do fondova EU

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Uvod

Ovaj je priručnik na prvi pogled priručnik o programskom jeziku Python, a tek onda priručnik o programiranju. Svjesni smo toga. No, osnovna nam je namjera naučiti vas programiranju, a ne Pythonu. Kako smo to zamislili? Programiranje je iznimno složena disciplina, a najgori je dio učenja programiranja prvi susret s njime. Mi u ovome slučaju koristimo Python da bismo vas upoznali s velikim brojem različitih ideja i koncepata, ne zato što smatramo da morate učiti Python, nego zato što Python smatramo toliko dobro osmišljenim jezikom da njegova struktura ne stoji na putu učenju.

Postoji nekoliko razloga koji Python čine idealnim jezikom za učenje programiranja. Čitljivost je onaj prvi razlog, koji se svima nameće, karakteristika koja je utkana u samu osnovu jezika Python. Čitljivost je za početnike u programiranju iznimno važna jer omogućava da se u učenju posvetimo logičkim strukturama i algoritmima, a ne čitanju teksta. Jasno strukturirani prikaz neke funkcije bit će nam jednostavnije shvatiti ako se ne moramo istovremeno koncentrirati na to kako funkcija radi i na koji je način ona napisana. Pythonova je struktura kao stvorena za to – od vrlo rigidnog načina na koji moramo formatirati blokove naredbi pa do minimalnog korištenja posebnih znakova i interpunkcije. Programi napisani u Pythonu prvenstveno su pregledni i čitljivi, a uloži li korisnik – programer minimalni trud, programi su onda i razumljivi.

Drugo, Python je najvećim dijelom potpuno nezavisan od platforme na kojoj se pokreće, tako da je programe u velikom broju slučajeva moguće pokrenuti na različitim operacijskim sustavima bez ikakvih promjena ili uz minimalne promjene. Tu je i jednostavnost instalacije – većina osnovnih alata dio je same instalacije, dok je većina dodatnih funkcionalnosti, koje bi nam mogle zatrebati kod ozbiljnijih programa, dostupna preko javno dostupnih i besplatnih skupova funkcija.

Fleksibilnost je također jedna od karakteristika koju ima Python – zbog načina na koji se radi s tipovima i varijablama, Pythonove se skripte lako pišu i razumiju. Kao viši programski jezik, Python neke stvari, za koje u drugim jezicima treba posebno paziti, rješava potpuno transparentno za korisnika. Najbolji je primjer način na koji se radi s velikim brojevima. Dok je u drugim jezicima uglavnom potrebno unaprijed znati ne samo s kojim brojevima radimo nego i koje će veličine biti krajnji rezultat kako bismo ga mogli ispravno pohraniti, Python će se pobrinuti da ispravno pohrani vrijednost koju pospremimo u varijablu, bez ikakvih ograničenja.

Postoji još jedna svakodnevna dvojba s kojom se susrećem svi koji biraju prvi „ozbiljniji“ programski jezik, dvojba o tome hoće li im odabrani jezik biti dovoljno brz ili dovoljno fleksibilan ne samo za prve korake nego i kasnije, kada se odluče na ozbiljnije programiranje. Iako je riječ o skriptnom jeziku, Python je koncipiran tako da se aplikacije napisane u njemu mogu vrlo brzo izvršavati, a podržan je i kompletni model objektnog programiranja bez obzira na to što prve korake radimo u vrlo jednostavnom funkcijskom okruženju. O brzini i kompatibilnosti Pythona govori i činjenica da je u njemu napisan YouTube, najveći svjetski videoservis, veliki dio Googleova pretraživača te Instagram, jedan od najvećih servisa za razmjenu slika.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

1. poglavlje: **Osnove Pythona**

U ovom poglavlju naučit ćete:

- ☑ što je to okruženje za rad
- ☑ kako ga instalirati
- ☑ što je to interaktivni rad
- ☑ što je to IDLE
- ☑ što je to sintaksa
- ☑ kako izgleda sintaksa u Pythonu.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

1.1 Okruženje za rad

Python je, kao programski jezik ili kao skriptni jezik, jedan od najbrže rastućih jezika dostupnih na tržištu.

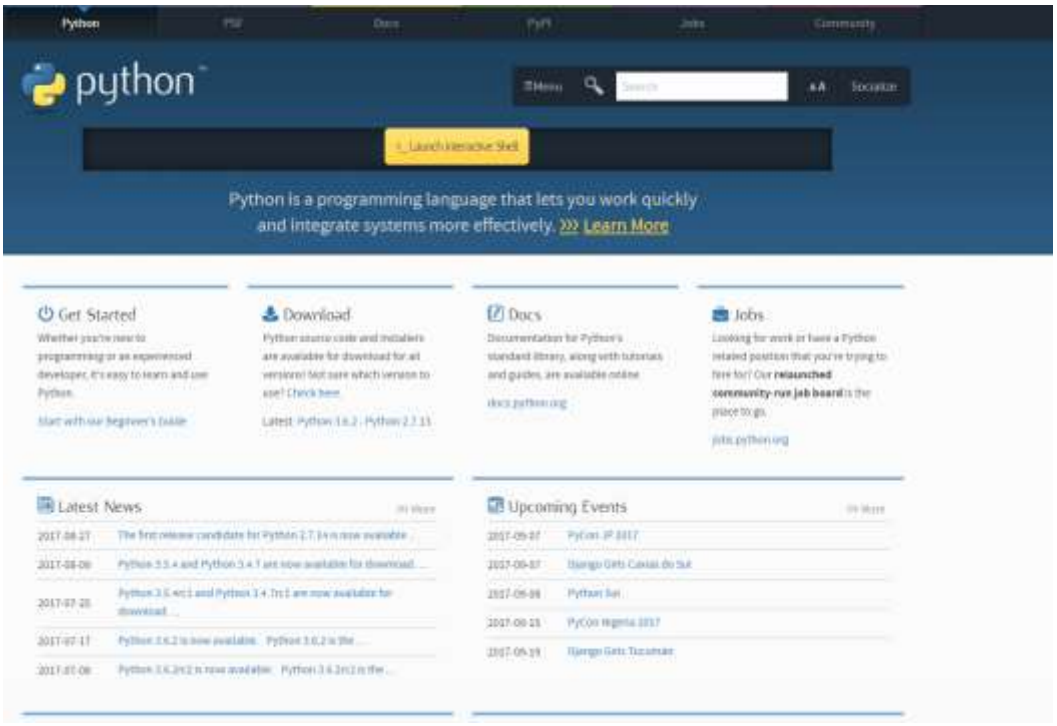
Ovaj dio materijala pokriva osnove Pythona i njegova okruženja koje su neophodne kako biste mogli početi raditi u ovome programskom jeziku, ali nije ni izbliza dovoljan da se svlada sve što Python čini zanimljivim. Zato preporučujemo da prvenstveno pročitate dokumentaciju koja prati Python, koja je besplatna i dostupna na internetu, te stotine primjera i materijala za učenje Pythona. Nemojte zaboraviti da se mi ovdje bavimo Pythonom samo kao načinom na koji je moguće brzo i efikasno naučiti osnove programiranja, dok je njegova dubina značajno veća.

Prva stvar koju je potrebno odabrati da bismo mogli ugodno (ili uopće) raditi u Pythonu je njegova inačica te osnovno okruženje u kojemu namjeravamo raditi. Izbora imamo mnogo, ne samo ako govorimo o okruženju ili uređivaču teksta u kojem namjeravamo pisati naše skripte. Čak ni verzija Pythona nije jednoznačna, jer se u Pythonovu svijetu istovremeno razvijaju dvije inačice 2.x i 3.x. U ovome priručniku označavat ćemo ih kao Python 2 i Python 3 jer stvarna oznaka ovisi o trenutnoj inačici dostupnoj na *webu*. U trenutku pisanja to bi bile 3.6.2 i 2.7.3. Trenutno dostupnu inačicu možete provjeriti i preuzeti na www.python.org.

Ovisno o tome na kojoj platformi (Linux, Windows) radite, odlučit ćete se za okruženje u kojem se osjećate najugodnije. Python je, kao interpreter, dostupan za sve velike, a i većinu malih platformi, tako da aplikacije možete pokretati ne samo pod Windowsima nego i, primjerice, na mobilnim telefonima. Sve što pokriva ova radionica radit će na svim trenutno aktualnim inačicama Pythona, tako da ne morate birati hoćete li instalirati inačicu 2 ili 3. No, preporučujemo da odaberete Python 3 koji predstavlja budućnost ovoga jezika.

1.2 Instalacija

Sama je instalacija jednostavna, Windowsi imaju klasičnu instalacijsku datoteku, dok je unutar Linuxa Python najjednostavnije potražiti kao dio „standardnog paketa aplikacija“ u vašem omiljenom *package manageru*.



Slika 1.: Python.org mjesto je na kojemu možete pronaći sve o Pythonu

Dovoljno je otići na www.python.org i preuzeti posljednju inačicu Pythona za vaš operacijski sustav (Python Software Foundation, 2018). Radite li sa sustavom Windows, to je instalacijska datoteka kod koje je dovoljno nekoliko puta pritisnuti „Next“. Pod Linuxom imate dvije mogućnosti: ili preuzeti instalaciju s *weba*, ili korištenjem svojeg paketnog menadžera instalirati Python izravno u svojoj distribuciji.

U nastavku demonstriramo kako izgleda instalacija u sustavu Windows, s nekoliko kratkih napomena. Prva se odnosi na instalacijski mehanizam jer možete odabrati datoteku koja u sebi sadrži sve potrebno za instalaciju, ili, takozvani, „web installer“ koji u sebi sadrži samo osnovne pakete, a ostale preuzima s interneta prema potrebi, odnosno prema tome što odaberete pri instalaciji. Trenutno je cijela instalacija relativno mala, oko 30 megabajta, pa je razlika između dvije verzije zapravo kozmetička, no u budućnosti se očekuje da će instalacija nuditi i niz dodatnih paketa pa će imati smisla ne preuzimati ono što nam nije nužno.

Instalaciju je moguće napraviti s predefiniranim postavkama ili se odlučiti za podešavanje svih opcija onako kako nama odgovara. Iako je i osnovna instalacija potpuno u redu, mi smo se odlučili za punu fleksibilnost odluka koje nam je ponudio instalacijski paket.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Slika 2.: Odabir lokacije instalacije

Preporučujemo da instalirate sve značajke jer među njima zapravo i nema nekog suvišnog materijala.

Dokumentacija je jasna, sadrži dovoljno informacija za svakodnevno pozivanje na najčešće korištene funkcije i sintaksu.

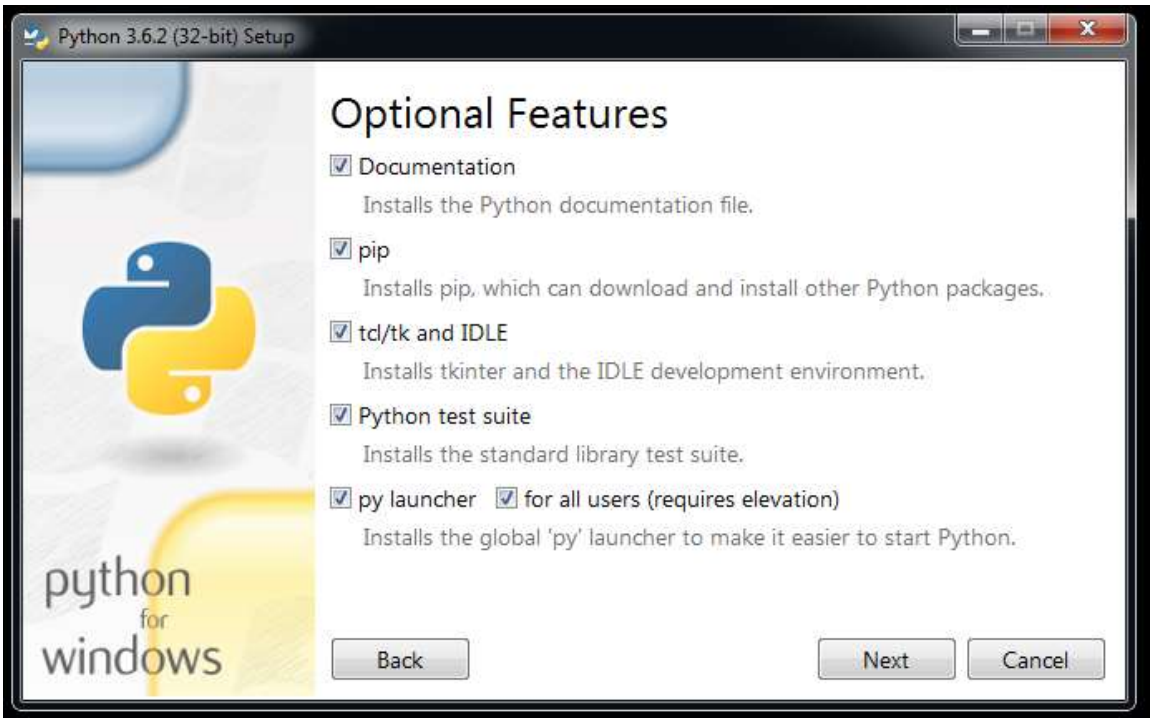
Pip je aplikacija koja omogućava da iznimno jednostavno instalirate bilo koji od dodatnih Pythonovih paketa i aplikacija, te je, također, gotovo nužan ako želite ozbiljnije raditi jer drastično skraćuje vrijeme potrebno da pronađete i instalirate dodatne biblioteke.

IDLE je, uz biblioteke tcl/tk, osnova bilo kakvog ozbiljnijeg rada u Pythonu, barem za početnike, jer je to ujedno i prvo integrirano okruženje za razvoj koje ćete vidjeti i koristiti.

Python test suite je jedina stvarno opcionalna stvar pri instalaciji jer je riječ o bibliotekama koje olakšavaju testiranje i prve korake u Pythonu.

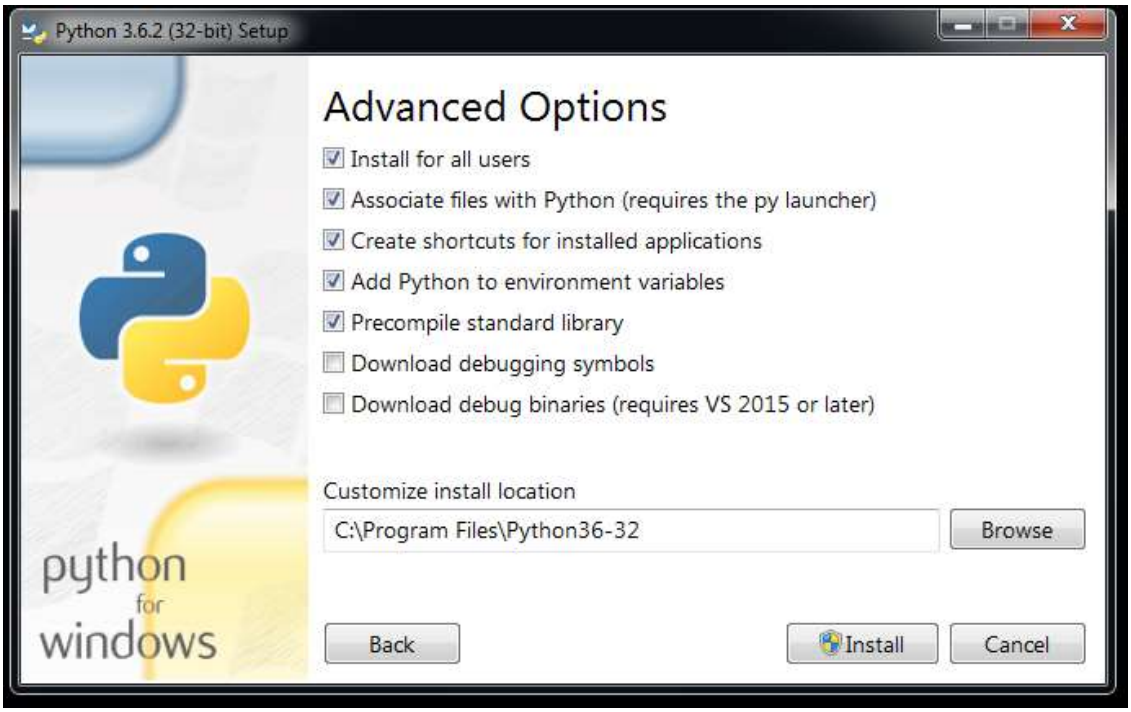
Py launcher je specifičan za operacijski sustav Windows i riječ je o posebnoj aplikaciji koja, po potrebi, pokreće Python unutar Windowsa. Iako zvuči nepotrebno, Py launcher je u stanju identificirati koju verziju Pythona koristi aplikacija te može odabrati i pokrenuti ispravno okruženje. Ovo je posebno zanimljivo u situacijama kada istovremeno imamo više instaliranih verzija, što je kod Pythona česta pojava, posebno ako se bavimo nadogradnjom starijih aplikacija.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr



Slika 3.: Instalacija – odabir željenih funkcionalnosti

Naš je prijedlog da Python instalirate za sve korisnike s podrškom za Py launcher. Time ćete dobiti mogućnost pokretanja svojih programa izravno klikom miša.



Slika 4.: Instalacija – ostale mogućnosti

Ovime je instalacija gotova i Python možete koristiti.

1.3 IDLE

Osnovno korisničko sučelje koje se koristi za Python, ako se želimo baviti nekim oblikom razvoja, najčešće je **IDLE**, zbog niza razloga. Glavni je razlog da IDLE dolazi kao dio osnovne instalacije Pythonova programskog paketa na svim platformama, te je izravno dostupan odmah nakon instalacije, malen je i brz, tako da za većinu korisnika predstavlja prvo sučelje koje uopće koriste da bi radili u Pythonu.

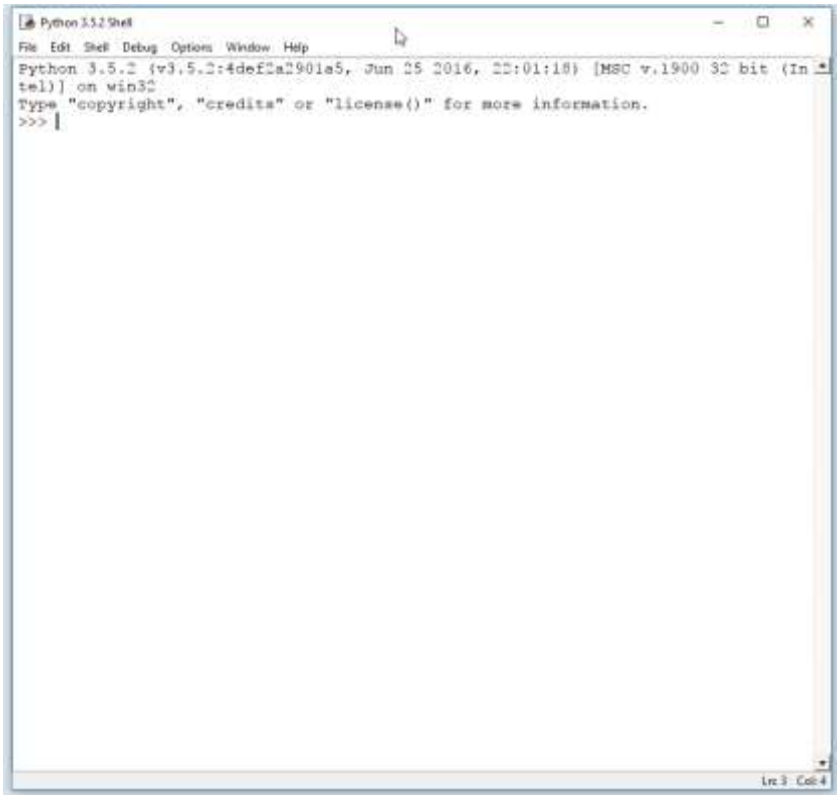
IDLE je i sam napisan u Pythonu, koristeći Tkinter Toolkit, posebni skup funkcija koje omogućavaju rad s prozorima neovisno o platformi na kojoj je aplikacija pokrenuta. U osnovi ovo znači da je IDLE uz minimalne izmjene dostupan i na operacijskim sustavima Windows, Linux i OSX. IDLE je, od svih načina rada s Pythonom, onaj najrudimentarniji, predviđen za početnike, ali i za profesionalce, no njegovo je sučelje prvenstveno zamišljeno za one koji su se s Pythonom već susretali. Mi na radionici, sve primjere i sve što radimo, radimo ili u interaktivnom okruženju ili unutar IDLE-a.

Vježbate li izvan IDLE-a, manje ili više, podrške za rad u Pythonu, ali i velikom broju drugih jezika, pružaju programerski uređivači teksta, posebna klasa programa za obradu teksta koja je bogata funkcijama koje su potrebne programerima, od označavanja dijelova programskog koda, integracije s različitim kompajlerima i interpreterima, pa do specifičnih modula za traženje pogrešaka i brigu o programskom kodu.

Oni koje bismo naveli kao trenutno najpopularnije su **Notepad++, Atom i Sublime**, a svaki je od njih na svoj način poseban i svaki od njih ima neku svoju prednost kada govorimo o svakodnevnom radu u Pythonu i drugim programskim jezicima.

Osim navedenog, oko Pythona postoji još i niz alata koji proširuju i olakšavaju svakodnevni rad, ili nude neke dodatne funkcije. Bilo da je riječ o proizvodnji aplikacija koje je na kraju moguće izvršavati potpuno samostalno, o osnovnom Python interpreteru ili o uređivanju gotovog koda tako da zadovoljava Python standarde, broj je alata koji su dostupni na internetu ogroman.

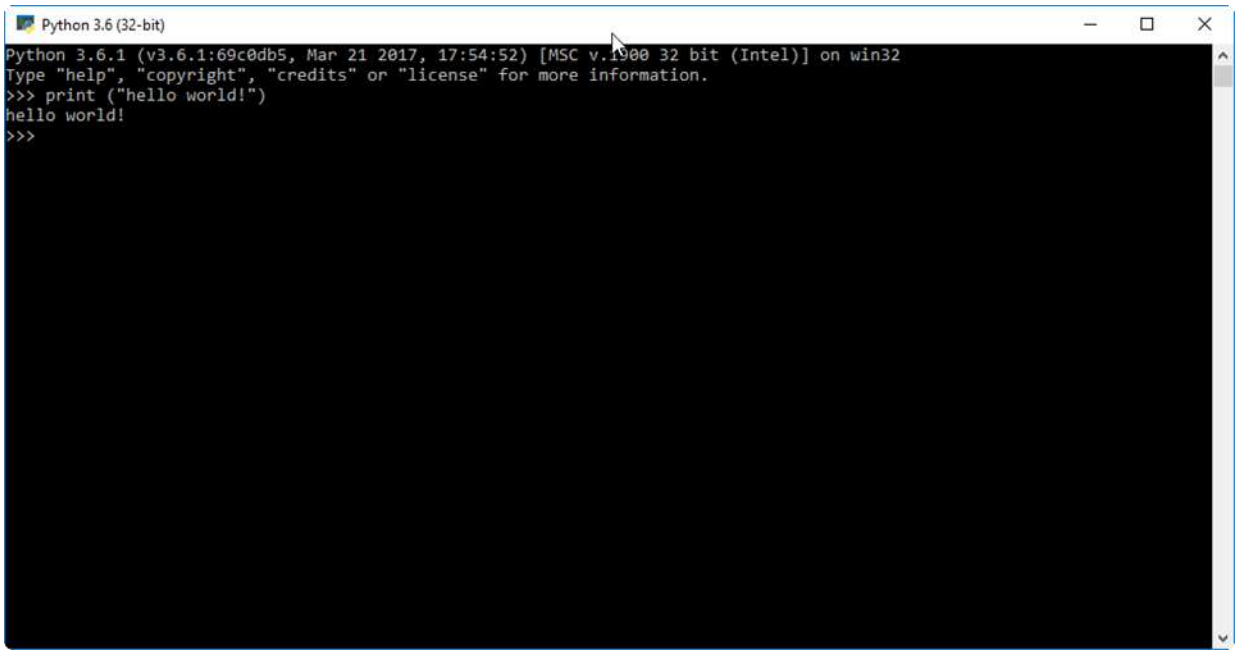
Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr



Slika 5.: Sučelje Pythonove ljuske

1.4 Interaktivni rad

Najjednostavniji je način, za brz razvoj i učenje korištenja novih funkcija, interaktivni rad, nešto po čemu je Python specifičan, jer je riječ o interpreteru. Pokrenete li samu aplikaciju Python, dočekat će vas sljedeći ekran:



Slika 6.: Izgled Pythonove interaktivne ljuske

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Naredbe je moguće pisati izravno, liniju po liniju, uz neke specifičnosti. Interaktivni rad često koristimo kad je potrebno provjeriti kako se neka funkcija ponaša, ili isprobati neki jednostavniji poziv vanjske biblioteke.

Unutar ove knjižice često ćemo koristiti interaktivni način rada da bismo demonstrirali pojedinačne funkcije. Na radionici ćemo primjere pokretati u IDLE okruženju, a interaktivni će se način rada koristiti uglavnom za demonstracije pojedinačnih jednostavnih funkcionalnosti. Razlog je jednostavan. Iako je unutar interaktivnog rada moguće koristiti i petlje i grananja, mnogo ih je lakše koristiti unutar zasebno napisanih programa, a ne ovako interaktivno.

Prije nego započnete bilo što ozbiljnije raditi, moramo spomenuti dvije najvažnije naredbe u interaktivnom radu: help() i exit(). Exit() je najjednostavnija naredba kojom se izlazi iz programa, ali i iz interaktivnog okruženja.

Help() je koristan za brzo podsjećanje o značenju koje ključne riječi, atributa ili funkcije. Ugrađene naredbe imaju jako dobar sustav pomoći pa se isplati baciti pogled ako se ne možemo sjetiti koja je sintaksa neke od naredbi. Primjerice:

```
>>>help(help)
Help on _Helper in module _sitebuiltins object:
class _Helper(builtins.object)
 |  Define the builtin 'help'.
 |
 |  This is a wrapper around pydoc.help that provides a helpful message
 |  when 'help' is typed at the Python interactive prompt.
 |  Calling help() at the Python prompt starts an interactive help session.
 |  Calling help(thing) prints help for the python object 'thing'.
 |  Methods defined here:
 |
 |  __call__(self, *args, **kwds)
 |      Call self as a function.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
>>>
```

Interaktivni način rada ujedno je i način na koji je nastala većina, ako ne i svi primjeri u ovome materijalu. Razloga je nekoliko, no glavni je brza provjera neke funkcije jezika ili funkcije nekog modula ili funkcije kod koje će najbrže biti otvoriti interpreter i krenuti izravno pisati naredbe, te vidjeti njihove rezultate u realnom vremenu. Interaktivni rad nije rješenje za sve, no postoji niz stvari koje u njemu možemo riješiti bez pisanja cijele aplikacije, posebno u trenutku kada se upoznajemo s nekim novim modulom ili nekim novim programskim konceptom.

1.5 Sintaksa Pythona

Pojedinačni izrazi bave se jednostavnim ili manje jednostavnim objektima i nad njima vrše neke radnje. Kombiniramo li nekoliko izraza, dobit ćemo naredbe koje onda možemo strukturirati u funkcije, klase ili cijele module.

Svaki jezik ima vlastita pravila o tome kako bi trebala ili morala izgledati sintaksa svih ovih logičkih struktura, pa ni Python nije nimalo drukčiji. Povijesno, Python se pridržava uputa i pravila koje su nazvane PEP, odnosno Python Enhancement Proposal. Riječ je o nizu dokumenata koji definiraju sve vezano uz Python, od strukture i načina pisanja Pythonova programa pa do novih i starih funkcija unutar Pythona. PEP je način na koji tvorci Pythona komuniciraju sa širom programerskom zajednicom, jer je prijedloge, koji će ući u ove dokumente, moguće izravno uputiti osobama zaduženima za održavanje nekog segmenta sustava Python.

Što se sintakse tiče, trebali bismo se pridržavati PEP-a 8, poznatog i kao Python Style Guide. Sam dokument nastao je 2001. godine i nadopunjavan je novostima i novim dogovorima oko ključnih elemenata pisanja skripata i aplikacija te ga vrijedi pročitati jer će uvelike olakšati rad.

Autor je ovoga dokumenta Guido van Rossum, autor cijelog Pythona, dok je razlog njegova nastajanja opservacija koju je Guido van Rossum imao pri samom stvaranju jezika – programe i aplikacije, odnosno njihov programski kod, ljudi će češće morati pročitati nego što će ga morati pisati. Samim time uvedena su pravila koja olakšavaju čitljivost i razumljivost koda.

Osnovna sintaksa Pythonovih aplikacija svodi se na poštivanje nekoliko jednostavnih pravila.

Prvo i osnovno pravilo je „Readability counts“, odnosno, čitljivost je bitna. Kod mora biti takav da ga je lako ne samo pročitati nego i shvatiti. Za čitljivost je važna dosljednost, odnosno, konzistentnost pisanja, pa je jedno posljedica drugoga. Potrudimo li se da svoj kod pišemo dosljedno, bit će ga lakše čitati.

Jedina iznimka koju upute priznaju je onda kada bi dosljednost u formatiranju koda stvarala probleme s radom ili mijenjala smisao izvršavanja određenog dijela koda.

Python je posebno osjetljiv na uvlačenje koda i na ispravno korištenje razmaka, a neka će razvojna okruženja čak prijavljivati greške ako je neka naredba krivo uvučena ili ne počinje na početku retka. Preporučuje se da naredbe budu napisane od početka retka. Ako je potrebno uvlačenje, ono bi moralo biti za 4 razmaka po svakom stupnju uvlačenja. Uvlačenje treba koristiti kako bi se omogućila čitljivost neke strukture ili kako bi se lakše razumjele petlje i grananja.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
ESF ULAGANJE U LJUDSKE POTENCIJALE
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

U pisanje programa i aplikacija za uvlačenje koristite razmake, a ne tabulatore. Ako ste već i počeli koristiti tabulatore, promijenite ih u razmake.

Sve linije unutar programa ograničite na 79 znakova. Razloga je nekoliko, a najvažniji je da u slučaju kada uspoređujemo više datoteka, ovakve kraće linije olakšavaju usporedbu napisanog i otvaranje različitih programa na istome monitoru. Ovo se posebno odnosi na komentare jer je važno lakše čitati komentare u više redova nego u jednom dugačkom redu.

Pri pozivanju modula, svaki modul pozovite u odvojenoj naredbi. Primjerice, umjesto

```
Import os, sys, math
```

napišite

```
Import os
Import sys
Import math
```

Ovako napisan kod čitljiviji je i jasno daje do znanja koji su moduli potrebni za funkcioniranje aplikacije.

Isto tako, izbjegavajte korištenje suvišnih razmaka u naredbama ili u deklaracijama, no koristite razmake tamo gdje oni olakšavaju čitanje koda.

Napomena: Nećemo previše ulaziti u detalje i primjere različitih savjeta koji su dani u ovome dokumentu, no, ako se mislite ozbiljnije baviti Pythonom, obvezno ga pročitajte. Nalazi se na lokaciji https://www.python.org/dev/peps/pep-0008/ i ažurira se redovito.

Kako preporuke za pisanje Pythonova programa govore o potrebi da pazite na dužinu linije, moramo spomenuti još nekoliko sitnica.

Znak točke sa zarezom služi za spajanje naredbi i primjenjiv je samo na jednostavne naredbe, kao što su dodjele vrijednosti ili ispis. Primjerice:

```
a = 2; b = 3; print(a + b)
```

Zagrade su univerzalno sredstvo za nastavljanje linija koda. Na bilo kojem mjestu na kojemu otvorimo bilo koju od zagrada, Python će automatski nastaviti tražiti ostatak izraza dok ne naiđe na zatvorenu zagradu. Dajemo nekoliko primjera.

```
>>> a=[1,
... 4,
... 5,
... 6]
>>> a
[1, 4, 5, 6]
>>>


>>> a=(2+
... 3-2*(
... 2
```

```
... +2)
... -3)
>>> a
-6
>>>
```

Funkcionalno, vidimo da Python izraze tumači prilično fleksibilno, no, dok pišete vlastite programe, nemojte koristiti ovu fleksibilnost, nego pokušajte pisati jasno i pregledno, sa zagradama koje odmah daju do znanja koji se izraz u njima nalazi. Osim što tako izbjegavate pogreške, olakšavate rad sebi i drugima u trenutku kada će vaš program trebati pogledati nakon nekog vremena.

1.6 Komentari

U svakom su programskom kodu komentari jedna od ključnih stvari. Komentari su upravo to, dijelovi teksta koji komentiraju neku funkcionalnost, koncept ili rješenje koje se nalazi u tom dijelu programskog koda. Korištenje komentara dokazuje da ste kvalitetan programer i značajno olakšava rad svima koji s vama moraju surađivati na nekom programu ili dijelu programa. Razloga je nekoliko. Onaj najvažniji je da komentari čine programski kod čitljivijim, a osobi koja ga analizira daju osnovnu informaciju što kod zapravo radi. Istovremeno, ako čitate programe, komentari vam daju priliku da odahnete i da vaš mozak lakše obradi podatke koji se nalaze ispred komentara. Ovo možda nije toliko važno kod programa koji se sastoje od desetak linija koda, no ispravno je komentiranje potrebno naučiti upravo na malim programima kako bismo sve što napišemo komentirali na isti način.

Komentari se u kod umeću iznimno jednostavno, korištenjem znaka „#“ što ćemo prikazati primjerom.

```
#komentar na početku programskog koda
A=1
B=2    #ovaj komentar nalazi se u liniji i opisuje nešto specifično
C=3


# komentare je moguće protegnuti i na nekoliko redova
# kako bi lakše objasnili kompleksnije pojmove ili funkcije
# koje smo realizirali u nekom dijelu aplikacije
```

Svi koji ulaze u svijet programiranja komentare smatraju potpuno nevažnima te ih na početku ignoriraju. Ovo je razumljivo, jer na početku rijetki pišu programe duže od nekoliko desetaka linija, a najčešće koriste gotove algoritme koji su im jasni. Jedino što možemo reći je to da naučite komentirati kod i da komentare koristite čak i ako ih smatrate nepotrebnima ili čak smiješnima jer tako štedite vrijeme nekome tko će vaš kod čitati kasnije. A možda ste taj netko upravo vi.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

2. poglavlje: **Tipovi podataka**

U ovom poglavlju naučit ćete:

- ☑ koji su osnovni tipovi podataka u Pythonu
- ☑ što su brojevi
- ☑ kako se radi s razlomcima
- ☑ što su to binarni tipovi
- ☑ što su to stringovi
- ☑ što su liste
- ☑ što su skupovi.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

2.1 Što su tipovi podataka?

Kada se govori o tipovima podataka, u svih jezika koji uopće barataju podatcima postoje dvije jasne skupine tipova podataka – oni ugrađeni i oni kompleksni, koje korisnik sam stvara. Kako je ovo uvod u programiranje, mi ćemo se gotovo isključivo baviti ugrađenim tipovima podataka zato što Python ima velik broj iznimno korisnih ugrađenih tipova podataka pa gotovo i nema potrebe za dodatnim definiranjem nekih drugih, vlastitih.

Istovremeno, neke ćemo tipove podataka samo spomenuti jer za njih u ovako vremenski ograničenoj radionici jednostavno nema mjesta, no, preporučujemo da više informacija potražite u literaturi jer je poznavanje tipova podataka vrlo korisno za lakše rješavanje algoritamskih i ostalih problema.

Prije svega, treba pojasniti što je zapravo tip podataka. Odgovor je prvenstveno intuitivan. Čak ćemo i u svakodnevnom razgovoru ponekad reći: „Daj, zapiši broj.“ ili „Prepiši ove tri rečenice.“ U ovim slučajevima našem sugovorniku jasno dajemo do znanja koje informacije slijede pa će, ako kažemo da zapiše broj, on automatski pretpostaviti da će ono što izgovorimo nakon toga biti neki niz znamenaka, a ne rečenica. Uz tu pretpostavku istovremeno ide i još jedna informacija o kojoj gotovo nikada ne razmišljamo svjesno, a to je informacija o tome koliko će nam prostora za zapisivanje trebati. Ako vas netko nazove i kaže: „Zapiši broj.“, nije isto kao i kada vas netko nazove i kaže: „Izdiktirat ću ti 100 brojeva.“ Za jedno je dovoljan papirić, za drugo je potrebno pronaći neku manju bilježnicu. Slična je situacija i s računalima.

U svakom programskom jeziku, pa tako i u Pythonu, neophodno je, na neki način, u trenutku kada neku informaciju namjeravamo pospremiti, računalu dati do znanja o kakvoj je vrsti informacije riječ, kako će ona izgledati i na koji je način najjednostavnije sačuvati. Klasični tipovi s kojima se susrećemo u programskim jezicima su slovo, *string* ili niz znakova, cijeli broj, broj s pomičnim zarezom i slično. Python, osim ovih tipova, barata i nizom drugih kompleksnijih struktura koje služe kako bismo brže i efikasnije mogli programirati i pisati programe.

Važna je stvar, koja kod Pythona ne dolazi do izražaja na prvi pogled, činjenica da kod svih tipova koje ćemo spomenuti gotovo da i nema ograničenja. Python se poprilično dobro snalazi s nizom znakova dugačkim desetak znakova ili desetak milijuna znakova, što je značajni odmak od onoga na što smo godinama gledali u drugim programskim jezicima. Upravo je zbog toga Python posebno zgodan za brzi rad s velikim skupovima podataka pa ga, osim kod programiranja, često viđamo kao osnovni programski jezik u radu koji nazivamo „big data“ ili „data mining“.

Prije nego prijeđemo na konkretne tipove podataka, važno je navesti još dva razloga zašto, posebno u Pythonu, uvijek nastojimo koristiti ugrađene tipove podataka. Prvi je razlog da je program, koji je jasno strukturiran i koji ima jasno definirane podatke, bitno lakši za čitanje i razumijevanje u slučaju da smo prisiljeni dodavati i mijenjati tuđe programe. Postoji izreka da svaki program treba pisati kao da je osoba koja će vas naslijediti serijski ubojica koji zna vašu kućnu adresu – ne postoji ništa gore od trenutka kada kao korisnik morate nastaviti tuđi posao u pisanju nekog programa, a ne postoje komentari koji bi vam pojasnili kako program funkcionira ili, ponekad, čak ni to što konkretno radi.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Drugi, možda i važniji razlog već smo spomenuli. Ugrađeni su tipovi nešto oko čega je nastao kompletan programski kod iza Pythona, te on s njima radi iznimno efikasno. Ovo znači brza pretraživanja, jednostavnije snalaženje u tablicama, brži ispis, te niz prednosti ako morate raditi s podatcima koji su nestrukturirani i kojih ima puno. To sve ne znači da nisu dozvoljeni neki vlastiti tipovi podataka, no oni ugrađeni gotovo će uvijek biti brži za rad.

Jedan od najvažnijih koncepata u Pythonu je da Python zapravo ne zanima kojeg je tipa sadržaj koji spremamo u varijablu. Upravo zato, u svakom primjeru koji vidite u ovoj knjizi, možete primijetiti da ni jednu varijablu nismo unaprijed definirali po tipu, nego smo joj samo dodijelili vrijednost. Python, na osnovi vrijednosti koja je dodijeljena, sam odlučuje koji će tip podataka iskoristiti za pohranu. Ovo će zbuniti sve one koji su se ikada bavili nekim od jezika koji imaju statičke tipove podataka kao što su Java, C ili C++. Kod tih jezika svaku varijablu prije korištenja moramo definirati i zadati joj tip, a, ako slučajno za vrijeme rada varijabli pridružimo vrijednost koja ne odgovara tipu koji smo unaprijed definirali, program će javiti grešku.

Kod Pythona, varijabli jednostavno pridružimo vrijednost i to je to. Naravno, neke operacije nije moguće napraviti nad nekim tipovima i neki tipovi imaju svoje specifičnosti što se tiče načina na koji referenciramo vrijednosti, no Python velikim dijelom razmišlja za nas. Druga je prednost ovakvog načina rada i dizajna samog sustava ta da korisnik ne mora razmišljati o veličinama i tipovima podataka koje sprema u smislu optimizacije. Sve interne funkcije optimizirane su tako da mogu optimalno raditi s malim, ali i s velikim skupovima podataka. Dok, primjerice, u jeziku C morate paziti na dužinu *stringa* jer o njoj ovisi alocirana količina memorije. Python takve stvari rješava transparentno za korisnika.

U Pythonu, tip podataka vezan je za konkretan objekt, te se, kada definiramo varijablu i njezinu vrijednost, zapravo radi o povezivanju varijable s objektom, koji onda opet nosi neki tip. Više riječi o tome bit će na vježbama.

Od ugrađenih tipova podataka koji su raspoloživi u Pythonu mi ćemo najčešće koristiti:

brojeve

stringove

liste

rječnike

tuplese

datoteke

skupove

razlomke

binarne tipove.

Neke ćemo od ovih tipova i detaljnije opisati. One koje ne spominjemo u ovoj radionici smatramo naprednim tipovima, pa preporučujemo da informacije o njima potražite u priloženoj literaturi.

2.2 Brojevi

Brojevi su najjednostavniji i većini korisnika najjasniji tipovi podataka, ako izuzmemo *stringove,* odnosno nizove znakova. Python prepoznaje nekoliko tipova brojeva:

- **Integer,** odnosno cijeli brojevi, brojevi koji nemaju decimalni dio.
- **Floating point,** brojevi s decimalnom točkom i varijabilnom preciznošću prikaza, odnosno brojem znamenki.
- **Complex,** kompleksni brojevi koji, osim realnog, imaju i imaginarni dio.

Tu su još i **skupovi (setovi)**, kojima samo ime govori o čemu se radi.

Za brojeve je najjednostavnije dati neke osnovne primjere izravno u interaktivnom okruženju Pythona, a vrijedi spomenuti da je Python u ovome načinu rada zapravo iznimno koristan i fleksibilan kalkulator.

```
>>> 2+2
4
>>> 2*2
4
>>> 2**2
4
>>> 2**3
8
>>> 2*3
6
>>> 2**10
1024
>>> 2*1000
2000
>>> 2**1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376
```

U ovim smo primjerima koristili nekoliko operatora. Osim uobičajenih „+“ i „*“, tu je i „**“ koji predstavlja eksponent. Dakle, ako napišemo 2**10, mislimo zapravo dva na desetu potenciju.

Moramo spomenuti i fleksibilnost računanja koju posjeduje Python, a to je automatsko povećavanje preciznosti. U većini programskih jezika za posljednju naredbu (2**1000) bilo bi potrebno unaprijed definirati da očekujemo vrlo veliki rezultat, kako bismo u varijabli koja ga prihvaća imali dovoljno mjesta za prikaz i izbjegavanje zaokruživanja. U Pythonu ne moramo napraviti ništa jer se sam jezik brine da podatci budu sačuvani i ispravno prikazani. Osim za precizno računanje, ovakav odnos prema velikim brojevima ima i druge zanimljive posljedice. Jedan od

primjera koji se često spominje je mogućnost da saznamo koliko znamenaka ima rezultat nekog izraza, primjerice:

```
>>> import math
>>> math.factorial(10)
3628800
>>> len(str(math.factorial(100000)))
456574
```

Razjasnimo što smo napravili. Prvo smo iskoristili dodatni modul koji se zove **math** kako bismo učinili dostupnima neke napredne matematičke funkcije. Više o modulima pročitajte u poglavlju koje se bavi upravo njima.

Vratimo se primjeru. Nakon što smo pozvali modul **math**, iskoristili smo njegovu funkciju **factorial** koja vraća *faktorijel* za zadani broj. *Faktorijel* je po definiciji umnožak svih cijelih brojeva od jedan pa do zadanog broja, uključujući i njega. Klasični matematički zapis bio bi 10!, no u Pythonu za to koristimo math.factorial(10). Krajnji je rezultat jednak umnošku 1*2*3*4*5*6*7*8*9*10.

Zanimljiv dio je onaj iza toga. Koristeći dvije funkcije za rad sa *stringovima*, ovdje smo pokazali kako pogledati koliko je velik rezultat, bez prikaza samog rezultata. Broj koji je prikazan (456574) nije *faktorijel* broja 1000, nego **broj znamenaka rezultata**. Kako smo izračunali broj znamenaka? Jednostavno, prateći zagrade.

Prvo izračunavamo sam *faktorijel*, no ne prikazujemo ga, nego ga prosljeđujemo funkciji str() koja ga pretvara u niz znakova. Taj niz znakova onda prosljeđujemo funkciji len() koja ih broji i daje nam krajnji rezultat.

Prevedeno na pseudojezik, mogli bismo reći da smo ovom funkcijom rekli „prebroji mi (niz znakova (faktorijel od 1000))“.

Modul **math** sadrži i neke konstante kao što su **pi** i **tau**.

```
>>> math.pi
3.141592653589793
>>> math.tau
6.283185307179586
>>>
```

Primijetite da je ispis rezultata iznimno jednostavan u interaktivnom načinu rada pa je dovoljno zadati izraz. Za ispis rezultata u programima morat ćete koristiti funkciju print().

2.2.1 Operatori

Spomenuvši osnovne numeričke tipove i način na koji možemo raditi s brojevima, moramo se malo detaljnije pozabaviti i operatorima te samim načinom pisanja brojeva. Operatori nisu tip podataka, no ključni su za njihovo razumijevanje jer definiraju prioritete u obradi podataka, te jednoznačno definiraju redoslijede obrade.

Dok je za cijele i decimalne brojeve način na koji ih pišemo manje-više jasan, jer prati ono na što smo navikli u svakodnevnom pisanju brojeva, postoje neke pojedinosti koje Python jako razlikuju od ostalih jezika.

Prva je preciznost. Iako nominalno postoje normalni i dugi zapisi cijelih brojeva, što zapravo opisuje količinu memorije koja je dodijeljena određenom broju, Python se automatski brine da u svojem funkcioniranju ne uzrokuje gubitak podataka. Iako nominalno možemo „dugi“ zapis označiti slovom L kada definiramo neki broj (L kao engl. *long*), Python će automatski brojevima dodijeliti 32 bita preciznosti ako je to dovoljno, a ako nije, automatski iskoristiti onoliko prostora u memoriji koliko je potrebno da se broj zapiše bez gubitka vrijednosti. Samim time moguće je bez razmišljanja raditi i neke stvari koje nam u drugim programskim jezicima ne bi pale na pamet.

Primjerice, možemo bez razmišljanja u varijablu pohraniti broj bitno veći od onoga koji stane u 32-bitni zapis.

```
>>> math.factorial(150)
57133839564458545904789328652610540031895535786011264182548375833179829124845398393126574488675311145377107878746854204162666250198684504466355949195922066574942592095735778929325357290444962472405416790722118445437122269675520000000000000000000000000000000000000
>>> A=math.factorial(150)
>>> A
57133839564458545904789328652610540031895535786011264182548375833179829124845398393126574488675311145377107878746854204162666250198684504466355949195922066574942592095735778929325357290444962472405416790722118445437122269675520000000000000000000000000000000000000
```

Python to radi potpuno transparentno za korisnika. Jedini realni problem može nastati ako vaša aplikacija počne raditi nešto i krivo računati neku vrijednost. Python će alocirati memoriju i pokušavati zapisati broj dok mu ne ponestane memorije.

No, osim veličine zapisa i preciznosti, Python kao programski jezik omogućava izravno baratanje i s brojevima u drugim brojevnim sustavima, dakle s drugim bazama. Tako je moguće cijele brojeve, osim kao decimalne, zapisati i kao heksadecimalne (u bazi 16), oktalne(u bazi 8) ili binarne (u bazi 2).

Brojevni sustav, koji želimo koristiti, odabiremo korištenjem prefiksa – decimalni brojevi ga po dogovoru nemaju, no, ako zapisujemo heksadecimalne brojeve, ispred broja pišemo 0x ili 0X. Za oktalne brojeve rezerviran je prefiks 0o ili 0O (znamenka nula i slovo o, veliko ili malo). Binarni brojevi, slijedeći logiku, počinju s 0b ili 0B. Brojevi su u memoriji zapisani kao cijeli brojevi, a postoje i funkcije hex(), bin() i oct() za pretvaranje u neku od spomenutih brojevnih baza. Baze i konverzija brojevnih sustava ne spadaju u program radionice, no spominjemo je zbog potpunosti priče o tome što je moguće u Pythonu.

Priča o operatorima nije vezana samo uz brojeve jer postoji još nekoliko operatora koje koristimo često, a operandi su im neki drugi tipovi.

Možda najčešće korišteni operator je „in“, koji ispituje pripadnosti skupu. Koristimo ga ako želimo provjeriti postoji li neki element u nekom skupu, ali i primjerice u FOR-ovoj petlji kada definiramo broj ponavljanja petlje.

Drugi često korišteni operator je negacija operatora „in“ koji logički glasi „not in“ i vraća logičku istinu ako neki element NIJE član nekog skupa.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Tablica 1.: Prioritet operatora u Pythonu (od najnižeg prema najvišem)

Yield x	Funkcija generatora
Lambda	Generator anonimnih funkcija
X if y else z	Ternarna evaluacija (x se evaluira samo ako je y)
X or y	Logičko ILI
X and y	Logičko I
Not x	Logička negacija
X in y, x not in y	Pripadnost skupu
X is y, x is not y	Test identiteta objekta (je li x y ili nije)
X < y, x > y, x <= y, x >= y	Usporedba veći – manji
X == y, x != y	Usporedba identičnosti vrijednosti
X | Y	Unija skupova; ILI kod operacija s bitovima
X ^ Y	Ekskluzivno ILI kod operacija s bitovima; razlika skupova
X & y	ILI kod operacija s bitovima, presjek skupova
X >> y, x << y	Pomak lijevo ili desno za y bitova
X + y	Zbrajanje
X – y	Oduzimanje
X * y	Množenje, ponavljanje
X % y	Ostatak, formatiranje
X / y, x // y	Dijeljenje, funkcija *floor*
-x, +x	Negacija, identiteta
~x	Inverzija bitova
X ** y	Eksponencijalna funkcija
X [i]	Indeksiranje (pozivanje indeksa)
X[i:j:k]	Rezanje (engl. *slicing*)
X(…)	Poziv funkcije, metode, klase…
x.attr	Referenciranje atributa
(…)	*Tuple*, izraz, generator
[…]	Lista
{…}	Rječnik, skup

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Python je pun implicitnih pretvorbi i pravila pa moramo navesti još dva. Logično je, ali moramo naglasiti da, u slučaju kada zbrajamo dvije vrijednosti različitih tipova, rezultat preuzima tip koji je najsloženiji. Primjerice, ako zbrojimo 1 + 1.1 rezultat će biti 2.1 i to tipa decimalnog broja, bez obzira na to što je prvi operand cijeli broj.

Isto tako, u tablici koju smo pokazali nije bitno samo što radi koji operator, nego i kojim su redom operatori navedeni u tablici. Naime, kao generalno pravilo, ako u nekom izrazu imamo više operatora, oni se evaluiraju tako da viši prioritet ima operator koji je niže u tablici. Tako će, recimo, poziv funkcije imati viši prioritet od dijeljenja, a množenje od zbrajanja. U slučaju nedoumice koristite zagrade, jer one poništavaju ovaj ugrađeni prioritet te se prvo evaluira ono u zagradama, slijeva nadesno, od unutrašnjih zagrada prema vanjskima.

2.2.2 Prikaz brojeva

U ovome materijalu na nekoliko mjesta spominjemo način prikaza, jer je on specifičan za pojedine tipove podataka. Ista je stvar i s brojevima.

Definirajmo dva broja.

```
>>> a=3
>>> b=4
```

Podijelimo ih uz korištenje zagrada da demonstriramo prioritete.

```
>>> b/(2.0+a)
0.8
>>> broj=1/3.0
>>> broj
0.3333333333333333
>>> broj = 1/3
>>> broj
0.3333333333333333
>>> print (broj)
0.3333333333333333
>>> print (1/3.0)
0.3333333333333333
```

Ovo su sve načini na koje je moguće ispisati vrijednost, odnosno rezultat. Ispis kod kojeg samo navodimo ime varijable skraćeni je način na koji varijable možemo ispisati u interaktivnom načinu rada. U programima MORAMO koristiti naredbu print().

2.2.3 Često korišteni operatori

Osim standardnih (zbrajanja, oduzimanja, dijeljenja i množenja) vrijedi spomenuti i neke često korištene operacije koje nam mogu poslužiti u svakodnevnom radu.

Što se brojeva tiče, često nam je potrebno zaokruživanje, a Python (očekivano) ima dvije odvojene funkcije. Jedna je klasično zaokruživanje trunc() koja daje rezultat zaokružen na ***najbliži cijeli broj bez decimale***. Druga je floor() koja također zaokružuje, ali na ***najbliži manji cijeli broj***. Ovo je posebno zanimljivo kod negativnih cijelih brojeva.

Usput ćemo ponoviti i način na koji Python radi s brojevima. Primjerice, ako dijeljenje dvaju broja zadamo ovako, mi zapravo dijelimo dva cijela broja, pa je rezultat također cijeli broj, zaokružen na nižu cijelu vrijednost:

```
>>> 2/3
0
>>> 3/2
1
```

Želimo li vidjeti i decimalni dio rezultata, nužno je da neki od operanada bude decimalni broj.

```
>>> 3.0/2
1.5
```

Floor, odnosno zaokruživanje na najbliži manji cijeli broj, ima svoju kraću oznaku sličnu dijeljenju: „//“. Primijetite da je i rezultat decimalni broj, samo zaokružen.

```
>>> 3.0//2
1.0
```

Želimo li koristiti funkcije za zaokruživanje, potreban nam je modul **math**.

```
>>> import math

>>> math.trunc(2.5)
2
>>> math.trunc(-2.5)
-2
>>> math.floor(-2.5)
-3.0
>>> math.floor(2.5)
2.0
>>>
```

Dodatno zbunjuje to što se ovo ponašanje razlikuje u Pythonu 2 i 3. U novijoj verziji Pythona svako zaokruživanje poštuje funkciju floor(), dok je kod starog Pythona zaokruživanje bilo jednostavnim odbacivanjem decimalnog dijela broja. Ovo je posebno važno kod negativnih brojeva, primjerice, -5 / 2 je u Pythonu 3 jednako -3

jer je to prvi manji cijeli broj, dok je u Pythonu 2 rezultat -2, jer je to cijeli dio decimalnog broja -2.5.

2.2.4 Razlomci

Osim skupova, Python odnedavno podržava i razlomke i decimalne brojeve koje izdvajamo u posebnu skupinu, iako bi nominalno pripali pod tip brojeva. To znači da je napredna matematika još jednostavnija.

```
>>> (1/2)+(2/4)
1.0
>>> (2/3)
0.6666666666666666
>>> (2/3)+1
1.6666666666666665
>>>
>>> from fractions import Fraction
>>> f=Fraction(2,3)
>>> f+2
Fraction(8, 3)
```

Kao što vidimo, ako odlučimo zapisati razlomke u klasičnom obliku (x/y), Python za sve vrijednosti radi izračun vrijednosti i zaokružuje rezultat kao decimalni broj.

Odlučimo li koristiti **fraction** kao funkciju, rezultat će ostati razlomak, bez zaokruživanja.

2.3 Binarni tipovi

Binarni tipovi podataka imaju dvije vrijednosti, 1 i 0. Kako je uobičajeno, te dvije vrijednosti ujedno definiraju i logičke vrijednosti *True* (točno) i *False* (lažno). No Python vrijednosti *True* i *False* dodjeljuje i drugim objektima pa svaki objekt zapravo može biti *True* ili *False*.

Općenito, bilo koji broj koji je različit od nule tretira se kao *True*. 0 se po definiciji definira kao *False*.

Bilo koji objekt koji je prazan je *False*, inače se smatra da je *True*.

Pitate se zašto je ovo važno? Programeri u Pythonu često koriste ovu karakteristiku objekata kako bi pojednostavnili grananja. Primjerice, ako imamo niz znakova, možemo provjeravati ima li u njemu znakova funkcijom:

„if niz != "":""

Ali možemo i jednostavnije pisati:

„if niz:“

```
>>> a=''
>>> if a:
...     print ('String je pun')
```

```
... else:
...     print('String je prazan')
... String je prazan


>>> a='Test'
>>> if a:
...     print ('String je pun')
... else:
...     print('String je prazan')
...
String je pun
>>>
```

2.4 Stringovi

Stringovi ili nizovi znakova jedan su od najčešće korištenih tipova podataka, i u Pythonu i općenito. Python u njima čuva ne samo tekst, kao što bismo to mogli očekivati od klasične varijable ovoga tipa, nego i bilo koji niz bajtova, ono što bismo inače nazivali binarnim tipom podataka. Mi ćemo se u ovoj radionici baviti samo tekstom.

Niz znakova najjednostavnije je definirati korištenjem jednostrukih navodnika, dok ćemo u našim programima često nizove dobivati ili čitanjem iz datoteke ili tako što ćemo zatražiti neki podatak od korisnika.

Operatori uglatih zagrada koriste se za označavanje položaja u *stringu*, a kako imaju nekoliko vrlo zanimljivih karakteristika, odmah smo ih stavili u primjer.

```
>>> V='e-radionica Programiranje'   #definiramo varijablu
>>> V                #ispišemo varijablu
'e-radionica Programiranje'

>>> V[0]

'e'

>>> V[1]

'-'

>>> V[1-2]

'e'

>>> V[1+2]

'a'

>>> V[-1]

'e'
```

Primijetimo nekoliko stvari. Prvo, Python indekse označava od nule, dakle prvi znak u *stringu* označen je kao V[0]. Pripazite, ovo je česta greška kod računanja pozicije i ispisa u programima. Znak s indeksom 1 zapravo je drugi znak *stringa.*

Drugo, unutar indeksa moguće je koristiti aritmetičke operacije, kao, uostalom, i bilo gdje drugdje na mjestima na kojima Python očekuje numeričku vrijednost. Tako V[1+2] zapravo ispisuje slovo na indeksu 3, odnosno četvrto slovo niza.

Treće, indeksi mogu biti negativni. I to je način na koji Pythonu kažemo da želimo krenuti od kraja *stringa*, zdesna nalijevo. Zadnji znak ima indeks -1, predzadnji -2 i tako dalje. U nekim bismo drugim jezicima za ovakve stvari bili prisiljeni koristiti funkcije za izračun pozicije, dok je kod Pythona sve riješeno samim pozivanjem *stringa* s negativnim parametrima.

2.4.1 Rezovi

Iduća zanimljiva stvar su takozvani rezovi ili *slices*. Obično ih koristimo u jednini, kao jedan rez ili jedan *slice.*

```
>>> V[1:3]
'-r'
>>> V[0:7]
'e-radio'
>>>
```

Prvi rez govori da želimo znakove od 1 do 3, što ne uključuje desni indeks, dakle kao rezultat ćemo dobiti znakove s indeksima 1 i 2. Drugi primjer ispisuje prvih 7 znakova niza. Početak i kraj reza u ovakvom ispisu možemo izostaviti i onda se pretpostavlja da mislimo na kraj ili početak *stringa*.

```
>>> V[:11]
'e-radionica'
>>> V[12:]
'Programiranje'
>>>
```

Ovo nam omogućava neke vrlo jednostavne, lako čitljive i korisne stvari. Primjerice, ako želimo s kraja *stringa* odrezati zadnja tri znaka, ne moramo, kao u drugim programskim jezicima, prvo gledati koliko je *string* dugačak, nego je dovoljno jednostavno iskoristiti negativni indeks.

```
>>> V[:-5]
```

```
'e-radionica Programi'
```

Još jedna od zanimljivih karakteristika *stringova* u Pythonu je njihov odnos prema numeričkim operacijama. Pokažimo jednostavno zbrajanje i množenje.

```
>>> V+' 1'
'e-radionica Programiranje 1'
>>> V+V
'e-radionica Programiranjee-radionica Programiranje'
>>> V
'e-radionica Programiranje'
>>> V*3
'e-radionica Programiranjee-radionica Programiranjee-radionica Programiranje'
>>>
```

U prvom primjeru – *stringu* jednostavno smo pribrojili razmak i broj jedan. Primijetite da ni u ovom, a ni u idućim primjerima nismo promijenili vrijednost varijable, nego samo ispisali krajnji rezultat.

Drugi primjer je također bilo zbrajanje. Jednostavno smo dva puta ispisali istu varijablu. Između nedostaje razmak jer ga ni originalna varijabla nema.

Treći primjer je množenje, koje je po definiciji ekvivalentno ponovljenom zbrajanju pa zato varijablu ispisujemo triput.

Operacije možemo i kombinirati, pri čemu funkcioniraju onako kako bismo i očekivali u matematičkom smislu.

```
>>> V+' '*3
'e-radionica Programiranje   '
>>> (V+' ')*3
'e-radionica Programiranje e-radionica Programiranje e-radionica Programiranje '
```

U prvom primjeru množenje ima prioritet pa se ispisuju tri razmaka. U drugom smo prioritet definirali zagradama, pa smo zapravo rekli: „Ispiši nam ovaj zbroj tri puta.“

Stringovi se u Pythonu ne mogu mijenjati. To u osnovi znači da, kada *string* jednom definiramo, u njemu više ne možemo raditi izmjene. Možemo samo zamijeniti cijeli *string* i dati mu isto ime.

Pokušajmo izmijeniti znak na poziciji 3.

```
>>> V[3]='t'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'str' object does not support item assignment
>>>
```

No, ako stvorimo novu varijablu preslagivanjem stare, to će funkcionirati tako da napravimo dva reza u „staroj“ varijabli te spajanjem stringova stvorimo „novu“ varijablu istog imena. Tiskarska je pogreška u ovome slučaju namjerna:

```
>>> V=V[0:3]+'t'+V[4:]
>>> V
'e-rtdionica Programiranje'
>>>
```

Spomenuvši da su *stringovi* zapravo sekvence, ujedno smo definirali i da je moguće koristiti indekse kako bismo pristupali dijelovima sekvenci. Osim ovih operacija, koje je zapravo moguće raditi nad drugim tipovima podataka koje možemo smatrati sekvencama, *stringovi* imaju i svoj skup metoda koje možemo primijeniti samo na njih. Primjer su, recimo, **find** i **replace** koji se u Pythonu rade na sljedeći način:

```
>>> V='e-radionica Programiranje'
>>> V
'e-radionica Programiranje'
>>> V.find('P')
12
>>> V.replace('-',' ')
'e radionica Programiranje'
>>> V
'e-radionica Programiranje'
>>>
>>>
```

Iznenađuje li vas rezultat? U prvome smo primjeru tražili znak „P“ unutar varijable V. Rezultat je bio 12, jer je to indeks na kojem je taj znak prvo pronađen. No, kada smo napravili **replace** u kojem smo rekli da želimo znak „-“ zamijeniti razmakom („ “), Python je izmijenio znak, ali je u skladu s nepromjenjivosti varijabli, originalna vrijednost varijable ostala ista, a izmijenjen je ispis.

Demonstrirat ćemo još neke metode.

Želimo li ispisati cijeli *string* velikim slovima, koristimo metodu upper().

```
>>> V.upper()
'E-RADIONICA PROGRAMIRANJE'
```

Želimo li pretvoriti *string* u listu uz korištenje nekog znaka kao separatora, funkcija se zove split(). O listama će biti više riječi kasnije.

```
>>> V.split()
['e-radionica', 'Programiranje']
```

Vrijedi spomenuti i specijalne znakove koje možemo upisivati ili kao \xNN heksadecimalne vrijednosti ASCII znakova, ili putem njihovih logičkih imena. Primjerice, uobičajeno je da je \n znak za kraj reda, dok je \t tabulator.

Slova su hrvatske latinice u Pythonu u potpunosti podržana jer je Python u potpunosti kompatibilan s Unicode znakovima.

2.5 Liste

Lista je objekt, odnosno tip podataka koji je svojstven Pythonu, i većina dugogodišnjih korisnika ovoga programskog jezika ne može zamisliti Python bez njih. Riječ je o uređenim skupovima objekata proizvoljnog tipa koji nemaju fiksnu duljinu.

Razmislimo malo o tome. Kada kažemo „proizvoljni objekti“ mislimo na to da lista može biti skup slova, *stringova*, brojeva, ili svega spomenutog. Lista može biti i skup lista, dakle lista lista. Lista je uređena pa za svaki objekt znamo njegov položaj u listi, slično kao što i za svako slovo znamo njegov položaj u riječi. Ako promijenimo raspored slova, to više nije ista riječ. Isto tako, promijenimo li raspored objekata u listi, to više nije ista lista.

Napomena: Liste, za razliku od *stringova*, možemo mijenjati izravno dodjeljivanjem novih vrijednosti objektima u listi. Liste su vrlo važan i snažan tip podataka.

Liste je najjednostavnije shvatiti primjerom pa definirajmo nekoliko lista.

Prvo definiramo i ispisujemo dužinu liste koja se sastoji od nekoliko znakova, riječi i brojki.

```
>>> L=[123,'lista','element',3.14]
>>> len(L)
4
```

Drugi primjer je lista sastavljena samo od brojeva.

```
>>> lista=[1,2,3,4,1,2,3,4]
>>> lista
[1, 2, 3, 4, 1, 2, 3, 4]
>>>
```

Primijetimo da listu definiramo uglatim zagradama, dok njezine elemente možemo definirati na isti način na koji bismo ih pridruživali varijablama. Elementi liste mogu biti bilo kojeg tipa, pa i same liste. Prisjetimo se rezova i indeksa koje smo koristili u

stringovima; svi oni vrijede i u listama te nam omogućuju da izravno dohvatimo element ili nekoliko elemenata liste.

```
>>> L[1]
'lista'
>>> L[:-2]
[123, 'lista']
>>> L=L+L
>>> L
[123, 'lista', 'element', 3.14, 123, 'lista', 'element', 3.14]
```

Moramo napomenuti da je kod lista potrebno pažljivo dodavati nove elemente, korištenjem odgovarajuće naredbe, jer će pokušaj da dodamo novi element s indeksom koji bi bio veći od zadnjega rezultirati greškom.

2.5.1 Dodavanje elemenata u listu – *append*

Dodavanje elemenata u listu može se napraviti na nekoliko načina. U gornjem primjeru pokazali smo to stvaranjem nove liste istog imena, no elemente je moguće dodavati i izravno jer je lista promjenjiva. Neke od češće korištenih funkcija su pop() i append(). **Pop** služi kako bismo iz liste izbacili određeni element. **Append** u listu dodaje element i to uvijek na kraj liste.

```
>>> L.pop(1)
'lista'
>>> L
[123, 'element', 3.14, 123, 'lista', 'element', 3.14]
>>> L.append('jos jedan element')
>>> L
[123, 'element', 3.14, 123, 'lista', 'element', 3.14, 'jos jedan element']
>>>
```

2.5.2 Dijeljenje niza u listu – split()

Jedan je od najkorisnijih načina na koji možemo upotrijebiti liste onaj za rad s grupama podataka, primjerice s tekstom. Metoda split(), koju koristimo u ovome primjeru, jedna je od najkorisnijih funkcija upravo ako moramo nešto napraviti s tekstualnim varijablama ili bilo kojim oblikom teksta. Pozovemo li je bez argumenata, ona znakovni niz dijeli na niz objekata koji onda možemo spremiti u listu ili ispisati. Kako se kao separator koristi znak razmaka, krajnji je rezultat tekst podijeljen u pojedinačne riječi.

```
>>> tekst='Danas je četvrti dan radionice programiranja i polako počinjemo razumjeti što nam predavač objašnjava.'
>>> a=tekst.split()
>>> a
```

```
['Danas', 'je', 'četvrti', 'dan', 'radionice', 'programiranja', 'i', 'polako', 'počinjemo', 'razumjeti', 'što', 'nam', 'predavač', 'objašnjava.']
```

Nakon toga, nad novonastalom listom možemo raditi sve operacije koje smo koristili i nad bilo kojom drugom listom. Primjerice, možemo ispisati prve četiri riječi.

```
>>> a[0:4]
['Danas', 'je', 'četvrti', 'dan']
>>>
```

2.5.3 Traženje elementa u listi – index()

Još jedna vrlo korisna metoda služi kako bismo na brzinu pronašli prvo pojavljivanje nekog elementa u listi. Metoda se zove index() te vraća indeks, odnosno položaj elementa, koji navedemo kao argument poziva metode. Navedimo primjer.

```
voce = ['mango', 'ananas', 'jabuka', 'jagoda', 'banana', 'naranča']
print(voce)
v = input('Upišite naziv voća: ')
if v in voce:
    print('Voće se nalazi na poziciji  ', voce.index(v), ' u listi')
else:
    print('Nema tog voća u listi!')
```

Ovaj primjer ima niz zanimljivih funkcija i struktura koje moramo dodatno pojasniti. Prva je stvar, naravno, kreiranje liste koja se sastoji od šest elemenata, i svi su različiti. Nakon toga ispisujemo listu, te od korisnika tražimo da unese ime voća koje želi pronaći u listi.

Zanimljivi dio slijedi u naredbi „if“, koja predstavlja uvjetno grananje. Korisnik je naime unio neku riječ koja je pohranjena u varijablu v. Mi ne znamo je li korisnik uopće unio ime nekog voća pa prvo moramo provjeriti ima li smisla tražiti taj pojam u listi. Ovo provjeravamo korištenjem operatora „in“ koji će nam vratiti logičku istinu ako je riječ koja se nalazi u varijabli v jedan od elemenata naše liste.

Nakon toga ispisujemo poruku, ako je riječ koju smo unijeli element u listi, ispisujemo njezin indeks, dakle poziciju na kojoj se nalazi, a ako riječ nije u listi, to i ispišemo.

2.6 Skupovi

Postoji još tipova podataka koji su ugrađeni u Python, a koji nam mogu biti korisni, iako se ne koriste toliko često kao oni koje smo dosad spomenuli. Postoje, primjerice, skupovi ili setovi, kako ih službeno nazivamo. Skup je upravo to, skup jedinstvenih i nepromjenjivih objekata. Nad skupovima možemo raditi presjeke, unije i razlike, pa mogu korisno poslužiti u obradi podataka.

Navedimo jedan primjer.

```
>>> cetveronosci={'psi','macke','kojoti'}
>>> domacezivotinje={'kanarinac','psi','macke','ribe'}
```

Presjek dvaju skupa daje one objekte koji se nalaze u oba istovremeno.

```
>>> domacezivotinje | cetveronosci
{'kojoti', 'macke', 'psi', 'ribe', 'kanarinac'}
```

Unija skupova daje sve elemente iz oba skupa. Primijetite da je Python automatski izbacio duplikate jer se isti element u skupu može pojaviti samo jednom.

```
>>> domacezivotinje – cetveronosci
{'ribe', 'kanarinac'}
>>>
```

Razlika dvaju skupova daje elemente koji se nalaze u samo jednom od njih, s time da je važan redoslijed kojim skupove navodimo.

```
>>> cetveronosci – domacezivotinje
{'kojoti'}
>>>
```

U prvom slučaju razlika nam je dala one životinje koje su domaće životinje, a nisu četveronošci. U drugom slučaju ispisali smo životinje koje imaju četiri noge, ali nisu domaće životinje.

Spomenuli samo da se Python brine da skup ima jedinstvene elemente pa ga ponekad koristimo kako bismo eliminirali duplikate u nekoj listi ili popisu.

```
>>> S={1,2,3,1,1,1,2,1,1,1,3,4,1,3,4}
>>> S
{1, 2, 3, 4}
>>>
```

2.7 Liste i skupovi – različitosti i sličnosti

U svakodnevnom programiranju i radu nakon znakovnih nizova i brojeva uvjerljivo ćete najčešće koristiti liste, a tek onda skupove, no to ne znači da ova dva naoko potpuno različita tipa podataka nisu vrlo zanimljiva. Ono najvažnije možemo pokazati na jednostavnom primjeru. Definirajmo prvo dvije varijable. Varijabla a je lista, varijable b je skup. Usput ćemo pokazati još jedan način na koji možemo definirati da je neka varijabla lista ili skup.

Definirajmo prvo našu listu i dodijelimo joj neke vrijednosti, što činimo metodom append(x) gdje je x neki element koji želimo dodati u listu. U ovome primjeru kao elemente liste i skupa koristit ćemo samo brojeve, iako lista i skup mogu sadržavati elemente bilo kojeg tipa.

```
>>> a=list()
>>> a.append(1)
>>> a.append(2)
>>> a.append(3)
>>> a.append(4)
```

Ispišimo sada krajnji rezultat našeg dodavanja i vidimo da su elementi ispisani onim redoslijedom kojim su i dodani u listu.

```
>>> a [1, 2, 3, 4]
```

Elemente je u listu moguće dodati i na drugi način, jednostavnim zbrajanjem postojeće i neke nove liste. U našem slučaju zbrojit ćemo postojeću listu i novu listu od četiri elementa te ispisati rezultat.

```
>>> a=a+[5,6,7,8]
>>> a [1, 2, 3, 4, 5, 6, 7, 8]
```

Nakon što smo napravili listu, napravimo na sličan način i naš skup, koristeći ključnu riječ set().

```
>>> b=set()
```

Slično kao i kod liste, dodajmo neke elemente, no drugačijim rasporedom, i nekoliko puta.

```
>>> b.add(1)
>>> b.add(2)
>>> b.add(1)
>>> b.add(4)
>>> b.add(3)
```

Ispišimo sada skup.

```
>>> b {1, 2, 3, 4}
```

Primijetite da su elementi sortirani te da se pojavljuju samo jednom, bez obzira na to što smo u skup dodali nekoliko identičnih elemenata.

Što će se dogoditi ako pokušamo identične elemente dodati u našu listu?

```
>>> a.append(1)
>>> a.append(1)
>>> a.append(1)
>>> a.append(1)
```

```
>>> a.append(1)
```

Ili u naš skup?

```
>>> b.add(1)
>>> b.add(1)
>>> b.add(1)
>>> b.add(1)
>>> b.add(1)
```

Ispišimo obje varijable.

```
>>> a [1, 2, 3, 4, 5, 6, 7, 8, 1, 1, 1, 1, 1]
>>> b {1, 2, 3, 4} >>>
```

Očekivano, lista sprema sve elemente koje dodajemo u nju jer je element definiran svojom vrijednošću, ali i svojim položajem u listi. Skupovi su potpuno različiti, njihovi elementi su unikatni, a poredak nije sačuvan, nego se skup ispisuje onako kako je pohranjen, u ovom slučaju sortirano.

Sve je ovo važno ako želimo napraviti neke složenije operacije nad grupama objekata. Koristimo li liste, to će nam jamčiti da su svi objekti koje pohranjujemo članovi liste te da je sačuvan njihov raspored. Pospremimo li iste objekte u skup, sigurni smo da će sve različite vrijednosti biti članovi skupa, no samo jednom, pa se ne moramo brinuti o duplikatima.

2.8 Konverzije među tipovima

Python potpuno automatski dodjeljuje tipove podatcima, no ponekad je potrebno da neki od podataka upotrijebimo na drukčiji način, ili zbog operacije koju provodimo moramo biti sigurni kojeg je tipa podatak. Najbolji su primjeri pretvaranje broja u *string* i obrnuto, situacija kada smo odnekud pročitali niz znakova, no mi očekujemo broj.

U takvim slučajevima postoji niz funkcija koje se ne bave sadržajem neke varijable, nego njezinim tipom. Mi ćemo spomenuti one najčešće korištene, počevši od funkcije type() koja vraća podatak o tome kojeg je tipa neki podatak ili varijabla.

Primjerice:

```
>>> type('abc')
<class 'str'>
>>> type(123)
<class 'int'>
>>> type(1.2)
```

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

```
<class 'float'>
>>> a='123'
>>> b=123
>>> type(a)
<class 'str'>
>>> type(b)
<class 'int'>
```

Vidimo da smo u prva tri primjera redom provjerili svaku od najčešće korištenih klasa podataka, i da je funkcija vratila očekivani rezultat. Zanimljiv dio je onaj kada provjeravamo tip podataka za naše dvije varijable, a i b. Iako one površno gledano imaju istu vrijednosti, varijablu a definirali smo koristeći navodnike, dok je varijabla b definirana kao broj, izravnim pridruživanjem. Python ovo shvaća kao da želimo da varijabla a bude niz znakova, bez obzira na to što su ti znakovi brojevi, pa i funkcija type() vraća njezin tip kao „str“, dakle *string*, odnosno niz znakova. Tip varijable b je očekivani broj, i to cijeli.

Funkcija int() namijenjena je pretvaranju brojeva s pomičnim zarezom i nizova znakova u cjelobrojni tip. Ova, kao i sve slične funkcije, pokušat će pretvorbu tipova napraviti ako je to moguće, no jasno je da neke stvari jednostavno nisu izvedive. Primjerice, iako int() kao argumente prima *stringove*, odnosno nizove znakova, ako oni sadrže i slova, funkcija će vratiti grešku.

Pokažimo nekoliko primjera.

```
>>> int('abc')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
>>> int('123')
123
>>> int('1.3')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1.3'
>>> int(1.3)
1
```

I ovdje vrijedi naglasiti neke stvari oko konverzije. Vidimo da je konverzija niza znakova „123“ očekivano uspjela. No, zanimljivo je da nije uspjela konverzija iz niza „1.3“ dok funkcija zna konvertirati broj 1.3 iz pomičnog zareza u cijeli broj, uz odgovarajuće zaokruživanje.

Slične su i druge dvije funkcije iz ove grupe, float() koja pretvara cijeli broj ili niz znakova u broj s pomičnim zarezom, te str() koja pretvara bilo koji broj u niz znakova.

```
>>> float('1.3')
1.3
>>> float('123')
123.0
>>> float(23)
23.0
>>> str(123)
'123'
>>>
```

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

3. poglavlje: **Algoritmi, strukture i kodiranje**

U ovom poglavlju naučit ćete:

- ☑ što su to varijable
- ☑ što su to naredbe
- ☑ što su to programske strukture
- ☑ što je to kodiranje algoritama.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

3.1 Što je to algoritam?

Algoritam je precizno opisan postupak rješenja nekog problema, odnosno uputa kako jasno definiranim postupcima riješiti problem. Ključna je stvar da je do rješenja moguće doći, i to u konačnom broju koraka. Algoritmi i algoritamsko razmišljanje nisu ograničeni samo na matematiku, već je riječ jednostavno o posebnom načinu strukturiranog definiranja problema i njegova rješenja. Primjera je mnogo jer ljudi vole jasne upute, a bilo koji dobro razrađeni algoritam zapravo je jasna uputa.

Primjerice, bilo koji kuharski recept zapravo je algoritam koji definira kako od osnovnih sastojaka konačnim koracima doći do krajnjeg rezultata, gotovog jela. Upute za navigaciju također možemo smatrati algoritmom za dolazak do cilja. Primjera je mnogo.

Što čini algoritam:

- Algoritam ima konačni niz koraka.
- Svaki je korak akcija koju treba napraviti.
- Promjena poretka izvođenja koraka može dovesti do nerješavanja problema.
- Koliko ima načina rješavanja problema, toliko ima i algoritama.
- U svakom koraku zapisano je što radimo (akcija) i nad čim (objekt) izvodimo akciju.

Dajmo primjer algoritma koji je primjenjiv i u računarstvu, ali i u stvarnome životu. Odlučili smo zamijeniti sadržaj dviju čaša, A i B. Potrebna nam je još jedna čaša P, i sljedeći slijed akcija:

- Sadržaj čaše A prebaciti u „pomoćnu“ čašu P.
- Sadržaj čaše B prebaciti u čašu A.
- Sadržaj čaše P prebaciti u čašu B.

Zamislimo sada da ne govorimo o tekućini i čašama, nego o podatcima spremljenima na lokacijama A i B.

Kako bismo zamijenili podatke, potrebna je pomoćna memorijska lokacija (neka se zove P).

- Sadržaj memorijske lokacije A spremiti u memorijsku lokaciju P.
- Sadržaj memorijske lokacije B spremiti u memorijsku lokaciju A.
- Sadržaj pomoćne memorijske lokacije P spremiti u lokaciju B.

Vidimo da je slijed koraka i logika kojom se vodimo da bismo riješili problem identična, iako naš inicijalni konkretni problem s čašama nije povezan s našim drugim problemom s podatcima.

3.1.1 Što je to pseudokod?

Korake u algoritmu potrebno je na neki način jasno i nedvosmisleno prikazati. Grafički, to činimo dijagramom tijeka. Mi se u ovoj radionici nećemo baviti njime, jer

bi nam za to trebalo dosta vremena. Koncentrirat ćemo se na drugi način prikaza, pseudokod.

Pseudokod je način na koji bilo koji algoritam možemo opisati koristeći riječi prirodnog jezika. Mi zapravo riječima opisujemo naredbe i korake algoritma u posebnoj strukturi koja nam olakšava praćenje koraka. Naredbe ne moraju biti pisane govornim jezikom, a umjesto nekih naredbi možemo pisati znakove, primjerice, za matematičke operacije. Ključno je da svaki korak bude jasno i nedvosmisleno definiran.

Ovdje odmah možemo prijeći na Python jer je on zapravo svojevrsni pseudokod koji je postao pravi programski kod.

Dajmo primjer koji smo gore opisali riječima, zamjenu dviju varijabli.

```
p = a
a = b
b = p
```

3.2 Što je to programiranje?

Programiranje je stvaranje novih računalnih programa, odnosno stvaranje slijeda instrukcija koje služe obradi nekog skupa podataka i kao svoj konačni rezultat transformiraju taj skup na način na koji smo to željeli napraviti. Programiranje neki nazivaju dosadnim poslom, neki umjetnošću, no sigurno je samo da je programiranje vrlo specifična disciplina koja razvija analitičko i proceduralno razmišljanje.

> „Učenje programiranja potiče razvitak računalno-algoritamskog načina razmišljanja (engl. *Computational thinking*) koje omogućuje razumijevanje, analizu i rješavanje problema odabirom odgovarajućih strategija i programskih rješenja.
>
> Takav način razmišljanja nadovezuje se na matematički način razmišljanja (engl. *Mathematical thinking*) koji se sustavno mora razvijati u matematici.
>
> Takav se način razmišljanja mora prenositi i u druga područja, posebice u područje prirodoslovlja, kao i u praktični život.“ (Budin, L. – Povjerenstvo za uvođenje Informatike kao obaveznog predmeta u osnovnoškolskom odgoju i obrazovanju, 2018.)

Poznavanje programiranja ne mora nužno značiti i duboko poznavanje nekog konkretnog računala ili njegova funkcioniranja. Programiranje je prvenstveno temeljeno na razumijevanju načina na koji je moguće podatke prihvatiti, obraditi i isporučiti korisniku, te kako tim podatcima baratati na efikasan način.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Programiranje je pisanje niza uputa za rješavanje problema uz pomoć računala, s time da se te upute pišu koristeći neki programski jezik te ih nazivamo računalni program.

Računalni program (program) je organizirani skup naredbi koje se izvode određenim redoslijedom i sa zadanim ciljem, a sastoji se od niza naredbi koje mogu biti grupirane u blokove, ali mogu biti i pojedinačne. Naredbe govore što treba napraviti s podatcima koji su zapisani u varijablama, ili ih treba dohvatiti ili ispisati.

3.3 Varijable

Varijable smo već koristili u ovome priručniku jer bez njih nije moguće pokazati ni jedan jedini razumljivi primjer, no moramo ih i formalno definirati i pojasniti.

Varijable su jedna od najčešće korištenih struktura unutar bilo kojeg programa. Varijabla je zapravo oznaka iza koje se krije vrijednost. Način na koji to funkcionira u Pythonovim programima je jednostavan jer ga Python čini jednostavnim. Sve što mi kao programeri moramo napraviti je dodijeliti određenu vrijednost određenoj varijabli. Primjerice, a = 1. Prije ove dodjele nije potrebno raditi ništa drugo, samo dodijeliti vrijednost. Načelno nije bitno ni kolika je vrijednost u slučaju, primjerice, niza znakova ili nekog brojevnog tipa, ni koliko objekata sadrži ako govorimo o nekom od kompleksnijih tipova varijabli.

Napomena: Python u svemu razlikuje mala i velika slova, pa tako i u imenima varijabli. Varijabla „a“ i varijabla „A“ nisu iste varijable, isto kao što nisu iste ni „Varijabla“ i „varijabla“. Pripazite na korištenje velikih slova u aplikaciji jer je ovo jedna od najčešćih pogrešaka.

Varijabla se stvara kad joj je prvi put dodijeljena vrijednost. Nakon toga, svako iduće pozivanje varijable zapravo mijenja njezinu vrijednost. Varijablu nazivamo i objektom koji ima svoju vrijednost i svoj tip.

Tip varijable nikada nije unaprijed definiran, nego se to određuje automatski prema vrijednosti koju pridružimo varijabli.

Način na koji se definiraju varijable mogli ste vidjeti u većini primjera. Ovdje ćemo se i formalno njima kratko pozabaviti.

Varijable su vjerojatno najjednostavnije. Kako je Python potpuno oslobodio programera razmišljanja o tipu i vrsti varijabli u trenutku dodjele, varijabli se vrijednost dodjeljuje jednostavnim znakom jednakosti, bez obzira na tip objekta.

```
A=2
B=[1,2,3]
C={(1,2),(3,4)}
```

Kada koristimo izraze i njihovu evaluaciju, najjednostavniji je način demonstracije pomoću primjera.

```
i = i + 1       #uvećava i za 1
```

```
brojac += 1 #uvećava brojač za jedan
x = x*2 - 1   #množi x s 2 i oduzima 1. Prioritet množenja je veći od oduzimanja
KvadratHipotenuze = x*x + y*y     #računa kvadrat hipotenuze kao zbroj kvadrata kateta.
c = (a+b) * (a-b) #koristi zagrade kako bi osigurao da zbrajanje bude izvršeno prije množenja
```

3.4 Naredbe

Naredba je drugi element koji je apsolutno neizbježan u definiranju bilo kojeg programskog jezika. Naredba (engl. *command* ili *statement*) je bilo koja struktura u programskom jeziku koja izravno ima neku posljedicu, obično izvršavanje neke akcije nad podatcima ili neku drugu promjenu stanja sustava. Ovakav je opis vrlo suhoparan, no, kada kažemo naredba, možemo misliti na niz različitih stvari. Ponekad mislimo na neku jednostavnu naredbu, primjerice print(a) koja će ispisati vrijednost varijable a. No naredba, isto tako, može biti neka funkcija, ili ključna riječ, odnosno, ako govorimo o Pythonu i interpreteru, čak je i ime varijable naredba, jer govori interpreteru da ispiše njezinu vrijednost. U ovome tečaju, a i u većini materijala na koje ćete naići čitajući ne samo o Pythonu nego i o programiranju općenito, svako spominjanje termina naredba zapravo se oslanja na naš intuitivni dojam o čemu se zapravo radi i na što konkretno mislimo. Za potrebe ove knjižice stanimo ovdje, no, ako ste zainteresirani doznati više, obavezno pogledajte u literaturu.

3.5 Sintaksa

Sintaksa je skup pravila koja jednoznačno definiraju kako se pišu naredbe i svi ostali elementi jezika te kako se ti napisani elementi onda kasnije tumače pri izvršavanju. Kako bi se očuvala ta jednoznačnost, svi programski jezici u manjoj ili većoj mjeri prisiljavaju korisnika da ne krši pravila, te njihovo nepoštivanje obično završava greškom pri izvršavanju. Sintaksa je definirana za određenu verziju nekog jezika, te je za Python dostupna u internetskoj dokumentaciji, s preporukama kako pisati programe. Mi smo se time više pozabavili u prvome poglavlju ovoga priručnika.

4. poglavlje: **Grananje i petlje**

U ovom poglavlju naučit ćete:

- ☑ što su to uvjetna grananja
- ☑ što su to petlje bez uvjeta
- ☑ koje se petlje koriste u Pythonu
- ☑ for petlju
- ☑ while petlju.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

4.1 Upravljanje tijekom programa

Osnovni način izvršavanja svakog programa možemo smatrati slijedno izvršavanje. Slijedno je izvršavanje naredbi u programu nešto savršeno prirodno i logično, no ono krije problem. Ako je jedini način na koji se program može izvršiti onaj u kojemu se uvijek na isti način izvršava isti skup naredbi istim redoslijedom, program će nužno uvijek imati isti ishod. Zato u programiranju imamo niz načina na koje taj linearni slijed izvršavanja naredbi možemo promijeniti, ili ponavljanjem blokova naredbi, ili izvršavanjem funkcija koje opet predstavljaju neke nove blokove naredbi, ili jednostavnim preskakanjem određenih naredbi ako se ispune zadani uvjeti. U ovome se poglavlju bavimo upravo načinima na koje je moguće izmijeniti tijek programa, što znači da govorimo o uvjetnim i bezuvjetnim grananjima, petljama te posebnim funkcijama koje služe prekidanju izvršavanja nekog bloka naredbi.

4.2 Uvjetna grananja

U većini, ako ne i u svim programskim jezicima, uvjetno je grananje jedna od najvažnijih funkcija. Mogućnost odlučivanja i promjene programskog tijeka, koju nam nude grananja, osnova je programiranja. Pojam grananja lako je objasniti: unutar aplikacije stalno odlučujemo želimo li na osnovi nekog ulaznog parametra izvršiti jedan ili drugi blok naredbi. Blok naredbi može biti jedna, više njih ili cijela zasebna aplikacija, no ključna riječ je odlučivanje. Uvjetna nam grananja daju mogućnost da na osnovi uvjeta odlučimo kako će se dalje odvijati naš program i, u konačnici, kako će izgledati krajnji rezultat.

Python, naravno, omogućava uvjetno grananje korištenjem ključne riječi **if**, uz neke specifičnosti.

U Pythonu grananje realiziramo vrlo jednostavno, primjerice, ako želimo da varijabla b dobije vrijednost 3 ako je varijabla a manja od 1, potrebno je napisati:

```
If a<1:
    b=3
```

Vrijedi primijetiti dvije stvari: dvotočku, koja je obvezna, i uvlačenje naredbi, koje je također obvezno. U osnovi, dvotočka označava kraj logičkog testa koji treba zadovoljiti da bi se izvršio blok naredaba uvučenih ispod reda koji završava dvotočkom. Dvotočka, uvlačenje naredaba i kreiranje blokova karakteristike su po kojima je Python specifičan, i tu će se u početku događati najveći broj pogrešaka u vašim programima. Razlog zašto je uvlačenje odabrano kao način na koji se formiraju blokovi naredbi je čitljivost krajnjeg koda: naredbe moraju biti ispravno formatirane, a programer ne mora „loviti“ zagrade kroz kod kako bi povezao s kojim je dijelom naredbe **if** vezan koji dio koda.

Govorimo li o ovako realiziranom grananju, uočit ćete jedan mali problem. Mi u ovome slučaju programu kažemo: „Ako je zadovoljena sljedeća logička izjava, izvrši ovaj dio koda.“ no nigdje ne postoji uputa što program treba napraviti u slučaju da uvjet nije zadovoljen. Naravno, to je moguće riješiti slijednim navođenjem nekoliko uvjetnih grananja, i tako realizirano odlučivanje radit će ispravno ako dobro

definiramo uvjete. S gledišta čitljivosti ovakav program nije dobro rješenje. Zato uvodimo još jedan dodatak u osnovnu **if** naredbu, posebni blok naredbi koji se izvršava ako osnovni uvjet nije zadovoljen. Python (a i većina ostalih jezika) tako poznaje ključnu riječ **else**, pa u konačnici struktura naredbe **if** izgleda ovako:

```
If a:
    Naredba()
Else:
    Naredba2()
```

Ovo stvara jako čitljiv i jasan programski kod, a i prisiljava nas na ljepše formatiranje, posebno ako moramo ugnijezditi nekoliko naredbi **if**.

```
If a:
    If b:
        Naredba1()
    Else:
        Naredba2()
Else:
    Naredba3()
```

Ovako napisano i formatirano, odmah je jasno na koji se **if** odnosi koji **else**, i koja će se naredba izvršiti u kojem slučaju.

Protumačimo to. U konkretnom primjeru prvo provjeravamo uvjet a. Ako je on zadovoljen, program nastavlja izvršavati novi blok naredbi i provjerava uvjet b. Ovo zapravo znači da naš program ima tri moguća ishoda. Jedan je, ako su zadovoljeni uvjeti a i b, u kojem će slučaju biti izvršena naredba1. Dogodi li se da je zadovoljen uvjet a, ali ne i uvjet b, izvršit će se naredba2. Na kraju, ako uvjet a nije zadovoljen, izvršit će se naredba3. Primijetite da, u slučaju da uvjet a nije zadovoljen, ne provjeravamo što se događa s uvjetom b. Bez obzira na njegovo stanje, uvijek izvršavamo naredbu naredba3.

Uvjet može biti bilo koji logički uvjet, dakle, usporedba vrijednosti neke varijable s nekom unaprijed definiranom vrijednosti, usporedba dvije ili više varijabli, neka logička funkcija ili bilo koji drugi logički izraz koji na kraju daje rezultat točno (True) ili netočno (False).

Bilo koji ozbiljniji alat koji se danas nalazi na tržištu, automatski prepoznaje način na koji se formatira programski kod u Pythonu, pa će vrlo vjerojatno veliki dio posla oko ispravnog uvlačenja blokova naredbi biti već odrađen za vas, ali to ne znači da se toga ne morate pridržavati. Neispravno formatiran kod javit će grešku prilikom izvršavanja.

4.3 Što su to petlje?

Petlja je programska struktura kojoj osnovni smisao opisuje i samo ime. Riječ je o strukturi koja određeni slijed naredbi ponavlja na određeni način te nam tako omogućava da neki postupak ponavljamo proizvoljni broj puta. To je ponavljanje jedan od ključnih elemenata programiranja jer nam daje mogućnost da jednostavnim manipulacijama obradimo veliki broj sličnih podataka. Istovremeno, petlje značajno skraćuju potrebni kod, te olakšavaju njegovu čitljivost.

Drukčiji način korištenja **for** petlje je ako u petlji provjeravamo uvjet

Postoje dvije vrste petlji, bezuvjetne i petlje s uvjetima. One prve, bezuvjetne, kako im i samo ime govori, jednostavno su strukturirane tako da program petljom prolazi neki fiksni broj puta, te se pritom ne oslanja na neki uvjet koji bi se mogao promijeniti dok se petlja izvršava. Uvjetne petlje ponašaju se upravo suprotno. Nakon što uđemo u petlju, u svakom prolasku provjeravamo treba li iz petlje izići, te tome prilagođavamo izvršavanje.

Python poznaje dvije vrste petlji: **while** i **for**.

4.3.1 For

Već spomenuti **for** jedna je od najčešće korištenih petlji i postoji u gotovo svim programskim jezicima. Iako jednostavne strukture, FOR petlje iznimno su snažan alat ako ga ispravno iskoristimo.

Tipičan primjer petlje ponavlja se u svim programskim jezicima. Pokažimo primjerom kako bismo ispisali sve brojeve između 0 i 10.

```
for broj in range(0,10):
    print(broj)
```

Kratko je pojašnjenje neophodno zbog korištenja skupova, odnosno funkcije range(). Većina programskih jezika, naime, jednostavnu **for** petlju realizira bez korištenja dodatnih funkcija, dok je Python pomalo specifičan. Funkcija range() vrlo je jednostavna. Ako je pozovemo, vratit će sve brojeve između dva broja koja smo zadali, uključujući i donju granicu, ali bez gornje granice. Pokrenemo li ovu petlju, bit će nam jasno i kako radi i koje vrijednosti vraća range().

```
>>> for a in range(0,10):
...     print(a)
...
0
1
2
3
4
5
```

```
6
7
8
9
```

Primjerice, sljedeći kod ispisuje rečenicu unutar navodnika, ali svako slovo u novom redu. U ovome se slučaju svako slovo tretira kao zasebni element te se odvojeno ispisuje. Petlja zapravo rečenicu tretira kao niz slova, dakle *string* (više informacija o *stringovima* potražite u poglavlju o tipovima podataka).

```
>>> for i in 'Testna rečenica':
...     print(i)
...
T
e
s
t
n
a

r
e
č
e
n
i
c
a
```

Napomena: **For** petlja na početku izvršavanja petlje analizira uvjet koji smo zadali i definira broj ponavljanja. To znači da broj ponavljanja nije moguće mijenjati za vrijeme izvršavanja petlje, a jedino što je moguće je izići iz petlje korištenjem naredbe **break** o kojoj više informacija potražite na kraju ovoga poglavlja.

4.3.2 While

Petlja **while** je klasična petlja s provjerom istinitosti na početku petlje. Dakle, petlja će se izvršavati dok je zadovoljen uvjet koji dolazi neposredno iza ključne riječi **while**. Krenimo primjerom:

```
>>> a=3

>>> while a>1:
```

```
...     print ('Ispisujem nesto')
...     a=a-1
...
Ispisujem nesto
Ispisujem nesto
>>>
```

Jasno je vidljivo što petlja radi: unaprijed definiranu vrijednost varijable **a** na početku bloka uspoređuje s uvjetom i, ako je uvjet zadovoljen, izvršava blok. U našemu je konkretnom slučaju varijabla **a** imala unaprijed zadanu vrijednost 3, pa se blok izvršio dva puta.

Pripazite na to da vrijednost koju koristite u uvjetu petlje unaprijed definirate jer o njoj ovisi izvršavanje same petlje. Isto tako, važno je primijetiti da se uvjet provjerava u svakom izvršavanju prije nego se kod uopće izvrši, dakle, ako na početku nije ispunjen uvjet, cijeli blok naredbi nikada neće biti izvršen.

4.3.3 Prekid izvršavanja – break i continue

Postoje jasne situacije u kojima je neophodno prekinuti neku petlju ili neki programski blok jer se dogodilo nešto što nas sprečava da nastavimo dalje izvršavati taj dio programa. Ovo nije toliko čest slučaj, no ponekad je korištenje ovakve naredbe jedino rješenje za razumljiv kod.

U slučaju petlji Python je tako usvojio dvije naredbe koje su došle iz drugih programskih jezika: **break** i **continue**. Obje se bave promjenom redoslijeda izvršavanja te imaju gotovo suprotna značenja.

Break je naredba koja programu govori da odmah iziđe iz bloka unutar petlje u kojem se trenutno nalazi, te da izvršavanje nastavi tako da će se aplikacija ponašati kao da je petlja završila.

Continue je drugačiji. On isto tako izaziva ispadanje iz petlje, no petlja ne završava, nego se nastavlja izvršavanje i pokreće idući prolaz kroz petlju.

4.3.4 Petlja – korak po korak

Vježba:

Najjednostavniji način, kojim možemo objasniti sve elemente koje smo spomenuli u ovome poglavlju, konkretan je primjer, koji ćemo razraditi korak po korak.

Primjer je s predavanja. Pišemo kratki program koji traži da korisnik pogodi broj koji je računalo „zamislilo“. Korisnik ima pet pokušaja pogađanja broja, a računalo će mu, kao pomoć, reći je li broj koji je korisnik upisao manji ili veći od zamišljenog broja.

Definirajmo zadatak:

Napisati program koji će odigrati kratku igru s korisnikom. Računalo će odabrati jedan broj u rasponu od 1 do 10 te korisniku omogućiti maksimalno 5 pogađanja

zadanog broja. Unese li korisnik broj koji je računalo odabralo, ispisat će mu se poruka i program treba završiti. Odabere li korisnik broj koji je veći ili manji od zadanog, isto tako će mu se ispisati odgovarajuća poruka. Nakon petog pokušaja, ako korisnik nije pogodio program, treba ispisati broj koji je odabrao i prekinuti izvršavanje.

Umjesto da pokažemo cijeli program, pokušat ćemo korak po korak objasniti što nam je sve potrebno, pa onda to uklopiti u jednu cjelinu.

Prvi je korak, očito, osmisliti pseudokod koji će definirati osnovnu logiku programa.

Pseudokod:

Odaberimo broj.

Definirajmo da je trenutno broj pokušaja jednak nuli.

Ako je broj pokušaja manji od 5:

zatražimo od korisnika da unese broj

usporedimo uneseni broj s odabranim brojem

ako je uneseni broj manji, ispišimo poruku „manji od zamišljenog“

ako je uneseni broj veći, ispišimo poruku „veći od zamišljenog“

ako je broj jednak zamišljenom, ispišimo poruku i završimo program

povećajmo broj pokušaja za jedan.

Nismo pogodili broj u zadanom broju pokušaja, ispišimo poruku.

Očito je da je prva stvar koju moramo napraviti odabrati broj koji će korisnik morati pogoditi. Broj mora biti u intervalu od 1 do 10, te mora biti isti za vrijeme cijelog ponavljanja petlje.

Iskoristit ćemo funkciju randint(a,b) iz modula random koja radi upravo ono što želimo – odabire jedan cijeli broj iz intervala zadanog argumentima a i b.

```
from random import *
broj = randint(1, 10)
```

Nakon što smo odabrali broj, moramo definirati osnovno stanje, a to je da je ovo prvi korisnikov pokušaj. Za to koristimo drugu varijablu koju ćemo nazvati broj_pokusaja. U zadatku je zadano da je broj pokušaja pet pa se naša petlja treba izvršiti pet puta, ili prekinuti izvršavanje ako korisnik pogodi broj.

Postoje dva načina na koja možemo brojiti unutar petlje i svode se na to smatramo li da treba brojiti od nule ili od jedinice. Većina programera broji od 0, te ćemo i mi to napraviti te proglasiti da je prvi pokušaj zapravo nulti.

```
broj_pokusaja = 0
```

Nakon toga stvaramo petlju. Očito je da je uvjet broj pokušaja. Ukupno ih je 5, a mi smo krenuli brojiti od 0, tako da u uvjet stavljamo da broj_pokusaja mora biti strogo manji od 5. Time smo definirali da ćemo petljom proći ako je broj_pokusaja jednak 0, 1, 2, 3 ili 4. i unutar petlje ćemo vrijednost broj_pokusaja povećati za jedan. Time zapravo garantiramo da ćemo nakon odgovarajućeg broja ponavljanja izaći iz petlje.

Varijablu broj_pokusaja nazivamo i brojačem, jer se u njoj nalazi numerički podatak o tome koliko smo puta prošli petljom.

Naša petlja sada izgleda ovako.

```
broj_pokusaja=0
while broj_pokusaja < 5:
    broj_pokusaja = broj_pokusaja + 1
```

Ovo je ispravno, ali nema nikakvog smisla jer u petlji moramo nešto i napraviti.

Razvijmo logiku koja provjerava uneseni broj. Očito je da ćemo imati niz uvjeta. Nakon što unesemo broj, on može biti veći, manji ili jednak zamišljenom broju pa pišemo sva tri uvjeta s odgovarajućim porukama za korisnika.

```
broj_p = int(input('Probajte pogoditi broj: '))
if broj_p < broj:
    print('Zamišljeni broj je veći!\n')
if broj_p > broj:
    print('Zamišljeni broj je manji!\n')
if broj_p == broj:
    print('Bravo pogodili ste broj!')
```

Naravno, da bi sve to imalo smisla, treba to uklopiti u petlju kako bi korisnik imao mogućnost pogađanja više od jednog puta.

Program trenutno dakle izgleda:

```
from random import *
broj = randint(1, 10)
broj_pokusaja = 0
```

```
while broj_pokusaja < 5:
    broj_p = int(input('Probajte pogoditi broj: '))
    broj_pokusaja = broj_pokusaja + 1
    if broj_p < broj:
        print('Zamišljeni broj je veći!\n')
    if broj_p > broj:
        print('Zamišljeni broj je manji!\n')
    if broj_p == broj:
        print('Bravo pogodili ste broj!')
```

Ovo još nije točno rješenje, postoji jedan veliki problem. Bez obzira na to unese li korisnik ispravni, veći ili manji broj, ispisat će mu se odgovarajuća poruka, no petlja će se nastaviti izvršavati. U slučaju da broj nije identičan zamišljenom, to je u redu, no ako smo u nekom od pokušaja pogodili broj, osim što to moramo dojaviti korisniku, moramo i prekinuti izvršavanje petlje. Za to koristimo naredbu **break**. Kada je pozovemo, program će izići iz petlje te nastaviti normalno izvršavanje prvom idućom naredbom iza petlje.

Prije nego sastavimo cijeli program, razmislimo o tome što nam još nedostaje, odnosno koja naredba treba slijediti petlju. Razmislimo konkretno o stanju naše male aplikacije i njezinih varijabli nakon što petlja završi. Postoje dva ishoda. Jedan je da je petlja dosegla maksimalni broj pokušaja i korisnik nije uspio pogoditi broj. Drugi ishod je da je petlja završila prije postizanja maksimalnog broja pokušaja jer je korisnik pogodio broj i aktivirala se naredba **break**.

To znači da je pogađanje gotovo i da mi samo moramo postaviti zadnji uvjet. Ako je broj koji je unesen različit od broja koji je zamišljen, onda moramo ispisati poruku da broj nije bio pogođen. Ako je uneseni broj identičan zamišljenom, odgovarajuću smo poruku već ispisali unutar petlje, tako da samo završavamo program.

```
from random import *
broj = randint(1, 10)
broj_pokusaja = 0
while broj_pokusaja < 5:
    broj_p = int(input('Probajte pogoditi broj: '))
    broj_pokusaja = broj_pokusaja + 1
    if broj_p < broj:
        print('Zamišljeni broj je veći!\n')
    if broj_p > broj:
        print('Zamišljeni broj je manji!\n')
    if broj_p == broj:
        print('Bravo pogodili ste broj!')
```

```
        break

if broj_p != broj:
                print('Žao mi je. Niste pogodili broj. Zamišljen je broj ', broj)
```

To je ujedno i rješenje našega zadatka.

Pokušajte zadatak modificirati tako da je raspon brojeva od 1 do 100, i da korisnik na raspolaganju ima 10 pokušaja!

5. poglavlje: **Funkcije**

U ovom poglavlju naučit ćete:

- ☑ što su funkcije
- ☑ kako koristiti ugrađene funkcije
- ☑ kako kreirati svoje funkcije.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

5.1 Što je funkcija?

Funkcije su u Pythonu jedan od fundamentalnih dijelova jezika. U ovim smo ih materijalima već koristili, iako nismo o njima pričali kao o funkcijama jer ih je lako zamijeniti za naredbe, a i nije bilo potrebe za nekim prevelikim formalizmom.

Funkcija je po definiciji blok naredbi organiziranih u cjelinu koju možemo pozivati po potrebi, prosljeđivati joj parametre i dobivati neki rezultat koji onda vraćamo na mjesto na kojemu smo pozvali funkciju. Osnovna je ideja funkcija i njihova korištenja vrlo slična onome za što koristimo i module: jednostavnije je, preglednije, ali i znatno sigurnije definirati jedan blok koda na koji se onda pozivamo više puta, nego svaki odsječak kopirati i ponavljati unutar našeg programa.

5.2 Kako koristiti funkcije?

Poziv funkcije je u Pythonu vrlo jednostavan. Obično se funkcija poziva po imenu, primjerice:

```
Funkcija(a,b,c)
```

Vrijednosti koje prosljeđujemo u zagradama zovemo argumentima, a njihov broj ovisi o funkciji i njezinoj namjeni. Postoje funkcije koje ne koriste argumente, a takvu bismo funkciju pozvali s:

```
Funkcija()
```

Osim vrijednosti koje prosljeđujemo funkciji, funkcije mogu vrijednost i vraćati kao rezultat svojeg rada. Postoje dva načina na koja se to može realizirati. Jedan je da se rezultat obradi u tijelu funkcije i spremi u neku globalnu varijablu ili ispiše korisniku. Drugi način je da se rezultat vrati korisniku korištenjem naredbe **return** o kojoj više pročitajte nešto niže u ovome poglavlju. Ono što moramo reći je da funkcije koje vraćaju vrijednost moramo i pozvati na odgovarajući način, kako bi se vrijednost koju je funkcija vratila mogla iskoristiti u ostatku programa. Za to je dovoljna jednostavna dodjela vrijednosti u varijablu.

```
Vrijednost=Funkcija(a,b)
```

Što smo ovime napravili? Jednostavno smo pozvali funkciju Funkcija, proslijedili joj dvije vrijednosti (a i b), te njezin rezultat pospremamo u varijablu „Vrijednost“.

Konkretni primjer mogu nam biti funkcije print() i input().

Funkcija print() je sasvim sigurno najčešće korištena funkcija u većini programa. Funkcija služi ispisu teksta ili neke vrijednosti varijable, bez obzira na njezin tip. Poziva se jednostavno, print(argument), pri čemu je argument ono što ispisujemo.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Cijela ova knjiga prepuna je primjera ispisa, tako da za funkciju print() nećemo navoditi primjere, ali za funkciju input() hoćemo.

Nužna posljedica interakcije s korisnicima unutar programa je da u nekom trenutku moramo od korisnika doznati neki podatak. Obično je to u formi nekog pitanja poput „unesite svoje ime“ ili slično. Unos podataka osnovni je smisao funkcije input(). Pokažimo jednostavni primjer.

```
>>> brojka=input('Unesite brojku ')
Unesite brojku 123
>>> print(brojka)
 123
>>>
```

Poziv funkcije je jednostavan, argument je tekst koji će se ispisati, dok je njezin rezultat ono što je korisnik unio, te ga možemo izravno pohraniti u varijablu.

5.3 Definicija vlastite funkcije

Python podržava nekoliko načina definicije funkcija, od kojih ćemo se mi pozabaviti samo nekima, dok ostale prepuštamo vama i dodatnom učenju.

Osnovna funkcija u Pythonu definira se na sljedeći način:

```
def funkcija(argument1,argument2….):
    naredba
    naredba
    return vrijednost
```

Ključna riječ **def** zapravo daje do znanja Pythonu da je blok koda koji slijedi funkcija. Nakon toga ide ime funkcije i njezini argumenti. Naredbe unutar bloka, koji čini samu funkciju, u potpunosti su proizvoljne. Naredba **return** je također opcionalna, i služi kako bismo vratili vrijednost, odnosno krajnji rezultat rada funkcije. Argumenti su u Pythonu malo drukčiji nego u nekim drugim programskim jezicima, jer se mogu prenijeti prema poziciji, dakle redom kako su navedeni u pozivu funkcije, ali i izravno, povezivanjem imena argumenta i vrijednosti, pri čemu se vrijednost onda dodjeljuje prema imenu, a ne poziciji.

Dajmo primjer najjednostavnije funkcije.

```
>>> def umnozak(a,b):
...     return a*b
...
>>> umnozak(2,3)
6
```

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

```
>>>
```

Kao što vidimo, prvo smo definirali samu funkciju koja prima dva parametra, a u našem slučaju implicitno podrazumijevamo da su u pitanju brojevi. Prva (i jedina) naredba u našoj funkciji je **return**, koja vraća umnožak parametara. Primijetite da funkcija ne provjerava apsolutno ništa vezano za sadržaj ili vrijednost parametara. Proslijedimo li joj, primjerice, neki od tipova koji ne podržavaju množenje, sam će poziv funkcije javiti grešku. Istovremeno, kako je, primjerice, moguće množiti *stringove* i brojke, ovakav poziv imat će valjan rezultat.

```
>>> umnozak('Riječ ',4)
'Riječ Riječ Riječ Riječ '
>>>
```

Ovakvo ponašanje nazivamo polimorfizmima, jer smisao operacije ovisi o operandima i operatoru, a znak za množenje može davati niz različitih rezultata, ovisno o tome kojeg su tipa operandi nad kojima ga primjenjujemo. U prethodnim je primjerima prva operacija bila množenje, druga je bila ponavljanje, što nam je razumski jasno, no i dalje zahtijeva određeno pojašnjenje. Polimorfizmi zapravo znače da bi naš program trebao biti sposoban raditi s različitim tipovima podataka, ili striktno provjeravati tipove podataka argumenata poziva funkcije, što je onda opet u suprotnosti s filozofijom rada Pythona koja kaže da su tipovi vezani uz objekte, a ne uz konkretne varijable.

5.3.1 Globalne i lokalne varijable

Govoreći o funkcijama, moramo spomenuti i klasičnu priču o globalnim i lokalnim varijablama. Globalne su one koje vrijede unutar cijele skripte ili nekog većeg bloka koda, dok su lokalne one koje vrijede samo unutar funkcije. Funkcija može koristiti imena varijabli koje su globalne, dok god to eksplicitno ne specificiramo. Ako nekoj varijabli lokalno dodijelimo vrijednost, ona ostaje lokalna, čak i ako ima isto ime kao neka globalna varijabla. Primjerice:

```
>>> vrijednost=10
```

Definirali smo novu globalnu varijablu vrijednost i dodijelili joj vrijednost 10. Nakon toga definiramo funkciju u kojoj definiramo lokalnu varijablu istog imena. Funkcija joj dodjeljuje vrijednost argumenta kojim je funkcija pozvana te ispisuje vrijednost argumenta i lokalne varijable.

```
>>> def novavrijednost(v):
...     vrijednost=v
...     print(vrijednost)
...     print(v)
...
>>> vrijednost
10
>>> novavrijednost(20)
```

```
20
20
>>> vrijednost
10
>>>
```

Nakon što ispišemo vrijednost varijable globalno, pa pokrenemo funkciju i ispišemo lokalne varijable, vidimo dvije stvari: Python je unutar funkcije potpuno ignorirao činjenicu da postoji globalna varijabla, te je ispisao njezinu lokalnu vrijednost. Isto tako, kada smo izišli iz funkcije, vrijednost se globalne varijable nije promijenila.

Spomenuvši globalne varijable, moramo uzeti u obzir još jednu stvar: čak i ako su globalne, varijable svoju vrijednost zadržavaju samo unutar jednog modula. Ako trebamo zatražiti vrijednost neke varijable iz nekog drugog modula, za to služi naredba **import**.

Zatrebamo li unutar funkcije mijenjati vrijednost globalne varijable, to je ipak moguće korištenjem ključne riječi *global*. Primjerice:

```
>>> def novavrijednost(v):
...     global vrijednost
...     vrijednost=v
...     print(vrijednost)
...     print(v)
...
```

Nakon poziva funkcije napisane na ovaj način, promijenit će se globalna vrijednost varijable *vrijednost*. No ovo nemojte činiti. Iako je tako na prvi pogled lakše riješiti neke probleme, ovo iznimno komplicira traženje mogućih pogrešaka – funkcije po definiciji pišemo zato da bismo ih pozivali s više mjesta. Napravimo li ovakvu funkciju, ona će svaki put izmijeniti globalne vrijednosti varijabli, te će vrijednost ovisiti, ne samo o funkciji, nego i o kontekstu u kojem je pozvana, pa, ako ne znamo kontekst, više ne možemo pratiti ni stanje globalnih varijabli.

5.3.2 Kreiranje vlastite funkcije – primjer

Vježba:

Zadajmo si jednostavan zadatak (preuzet iz predavanja).

Potrebno je napisati vlastitu funkciju koja će računati n-tu potenciju broja. Ulazni argument je broj, dok funkcija treba pitati na koju potenciju želimo „podići“ taj broj.

Prije nego se uopće počnemo baviti ovim problemom, moramo naglasiti da ta funkcija očekivano već postoji i zove se pow(a,b), i podiže broj a na potenciju b. Kako mi ovo radimo za vježbu, a i kreiramo funkciju sa samo jednim argumentom, funkciju pow() ćemo iskoristiti u našoj funkciji.

Riješimo prvo glavni dio programa, dakle onaj dio koji poziva funkciju.

Korisnika moramo pitati koji broj želi potencirati, te pozvati funkciju i ispisati rezultat.

```
n = int(input('Unesite broj: '))
print('Rezultat je', potencija(n))
```

Alternativno, mogli smo rezultat dodijeliti nekoj varijabli, pa je onda upotrijebiti pri ispisu, no ovako je preglednije.

Idući je korak osmisliti funkciju. Očito je da funkcija mora primiti neku varijablu, te pitati korisnika za neku vrijednost. To je ujedno i rješenje.

```
def potencija(broj):
    eksponent = int(input('Unesite eksponent: '))
    rezultat = pow(broj,eksponent)
    return rezultat
```

Krajnje rješenje je dakle:

```
#definicija funkcije
def potencija(broj):
    eksponent = int(input('Unesite eksponent: '))
    rezultat = pow(broj,eksponent)
    return rezultat

#glavni program
n = int(input('Unesite broj: '))
print('Rezultat je', potencija(n))
```

Pokušajte funkciju modificirati tako da prima dva argumenta!

6. poglavlje: **Kornjačina grafika**

U ovom poglavlju naučit ćete:

- ☑ što je to kornjačina grafika
- ☑ zašto uopće kornjačina grafika
- ☑ kako je koristiti.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

6.1 Što je kornjačina grafika?

Cijela radionica, ovako kako je zamišljena, realizirana je u tekstualnom okruženju, uz izričito korištenje samo jednostavnih tekstualnih primjera. Kornjačina grafika, posebni skup funkcija koje smatramo dovoljno zanimljivima da smo im posvetili cijelo poglavlje, vrlo je specifičan alat koji olakšava učenje programiranja, a istovremeno omogućuje i privlačenje mlađe generacije koja puno više stvari shvaća vizualno.

Kornjačina grafika je skup funkcija koje u svojem središtu imaju koncept kornjače, objekta koji se nalazi na ravnoj dvodimenzionalnoj površini i koji ima ograničeni skup funkcionalnosti. Ovaj nas konkretni alat stavlja u ulogu onoga koji upravlja kretanjem kornjače te olovkom koju kornjača nosi. Dok se kreće prostorom, kornjača može svojom olovkom ostavljati ili ne ostavljati trag, te se može kretati i okretati oko svoje osi. Krajnji je rezultat dvodimenzionalni crtež.

6.2 Što je potrebno za kornjačinu grafiku

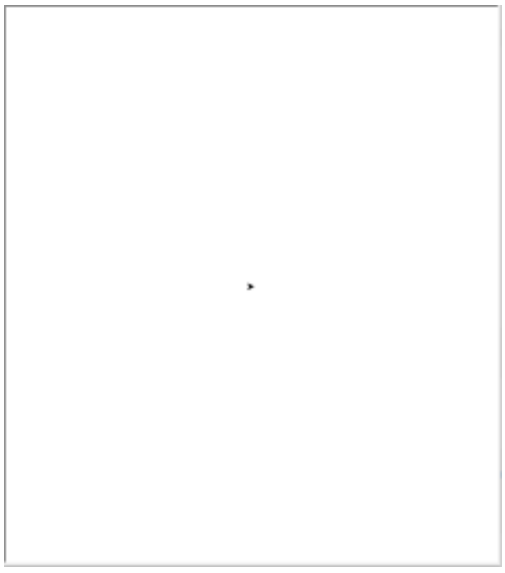
Ako imate iskustava s drugim programskim jezicima, iznenadit ćete se koliko je malo potrebno za prikaz kornjačine grafike. Postoji posebni modul imena „turtle“ koji je neophodno uvesti da bismo mogli koristiti sve funkcije koje kornjača „zna“. Kako je ovaj modul namijenjen u prvome redu početnicima, što se grafike tiče oko svega se brine Python, tako da je na korisniku samo da ispravno kombinira naredbe te nauči programirati. Sva crtanja i ostalo riješit će Pythonov interpreter.

6.3 Kako je koristiti?

Kornjačinu je grafiku prvo potrebno pokrenuti, za što su dovoljna dva koraka:

```
import turtle
turtle.home()
```

Python će onda inicijalizirati cijeli sustav za crtanje i prikazati nam „kornjaču“ i prostor u kojemu se ona može kretati.



Slika 7.: Izgled inicijaliziranog prozora kornjačine grafike

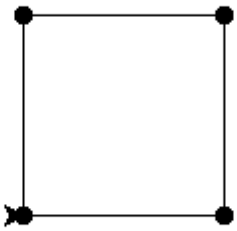
Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Tablica 2.: Najvažnije naredbe za pomicanje kornjače

Funkcija	Skraćeno	Opis
forward(d)	fd(d)	Pomiče kornjaču za d piksela naprijed
backward(d)	bk(d)	Pomiče kornjaču za d piksela nazad
right(kut)	rt(kut)	Zakreće kornjaču kut stupnjeva desno
left(kut)	lt(kut)	Zakreće kornjaču kut stupnjeva lijevo
penup()	pu() ili up()	Podiže se kornjača (ne ostavlja trag)
pendown()	pd() ili down()	Spušta se kornjača (ostavlja trag)

Nacrtajmo kvadrat.

```
from turtle import *
home()
dot(10); fd(100); lt(90)
dot(10); fd(100); lt(90)
dot(10); fd(100); lt(90)
dot(10); fd(100); lt(90)
```



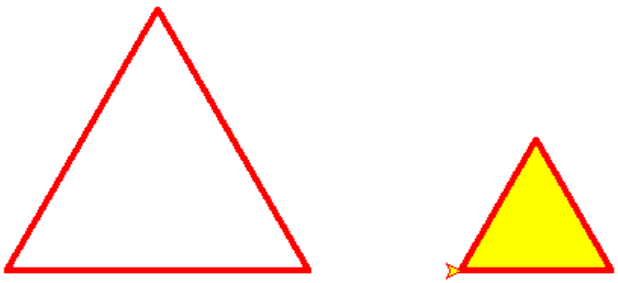
Slika 8.: Rezultat iscrtavanja kvadrata

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Što smo napravili? Počeli smo od prve koordinate, koja nam nije bila bitna, definirali debljinu crte, te krenuli 100 jedinica naprijed. Nakon toga smo okrenuli kornjaču ulijevo 90 stupnjeva te cijelu tu proceduru ponovili još tri puta i time se vratili na početnu točku. Primjer nam vrlo jasno pokazuje nekoliko stvari. Osim što odmah možemo vidjeti da je crtanje kornjačom iznimno jednostavno, jasno je i da se koraci pri crtanju ponavljaju te da bismo inteligentnijim programiranjem mogli ovaj kod učiniti znatno čitljivijim i fleksibilnijim. Zamislimo da, primjerice, želimo crtati kvadrate različitih dimenzija ili da želimo krenuti s različitih polaznih točaka.

No prvo nacrtajmo još dva kvadrata.

```
from turtle import *
home()
pensize(4)
pencolor('red')
fd(200); rt(240)
fd(200); rt(240)
fd(200); rt(240)
pu()
fd(300)
pd()
fillcolor('yellow')
begin_fill()
fd(100); lt(120)
fd(100); lt(120)
fd(100); lt(120)
end_fill()
```



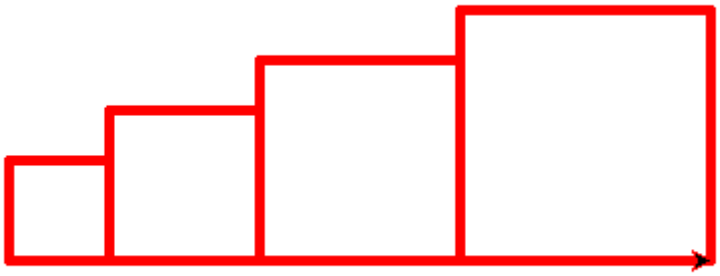
Slika 9.: Rezultat iscrtavanja trokuta

Ovaj primjer demonstrira kako se objekti mogu crtati drukčijim bojama, te kako ih ispuniti nekom bojom.

Kornjačinu smo grafiku spomenuli u kontekstu demonstracije načina na koji rade petlje, primjerice **for** petlja. Demonstrirajmo na jednome primjeru na što smo mislili. Potrebno je napisati program koji će korištenjem petlji nacrtati četiri kvadrata u nizu. Ulazni parametri su duljina stranice kvadrata te za koliko će svaki idući kvadrat imati veću stranicu.

```
from turtle import *
home()
pensize(5)
pencolor('red')
a = int(input('Duljina stranice kvadrata: '))
b = int(input('Povećanje stranice kvadrata: '))

for i in range(4):
    for j in range(4):
        fd(a); lt(90)
    pu(); fd(a);pd()
    a = a + b
```



Slika 10.: Rezultat iscrtavanja uzastopnih kvadrata

7. poglavlje: **Moduli**

U ovom poglavlju naučit ćete:

- ☑ što su moduli
- ☑ kako se koriste moduli
- ☑ koji se moduli često koriste.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

7.1 Što je modul?

Modul je u svojoj najgrubljoj formi zapravo samo još jedan Pythonov program koji nije moguće samostalno izvršiti, nego se u njemu nalaze atributi i funkcije koje možemo pozvati i koristiti u našem „glavnom“ programu. Module možemo zamisliti kao način izdvajanja stvari koje često koristimo u zasebnu datoteku. Postoje tri razloga zašto bismo se uopće bavili modulima.

Prvi je preglednost koda. Ako razumljive cjeline koda izdvojimo u odvojene module, naš će glavni program biti kraći i čitljiviji, a, ako je kasnije potrebno napraviti neke izmjene funkcionalnosti, lako je to napraviti unutar nekog od modula.

Drugi veliki razlog je ponovno korištenje koda. Razvijamo li, primjerice, nekoliko različitih malih programa koji se bave statistikom možda oni dijele zajedničku funkciju za brojenje riječi u tekst. Jedna je opcija da tu funkciju kopiramo u svaki program čime unutar pojedinačnog programa jasno vidimo kod koji nas zanima. No, ako istovremeno uočimo da je taj kod potrebno promijeniti, moramo promjene kopirati u svaki pojedinačni program i paziti da su kopije uvijek usklađene.

S druge strane, izdvojimo li kod u odvojenu datoteku i napravimo zasebni modul te ga pozovemo iz svakog programa, kod je na jednome mjestu i svaka promjena ili nadogradnja radi se samo na tom jednom mjestu. Ovo se posebno odnosi na funkcije koje stvarno često koristimo, primjerice, manipulacije *stringovima* ili imenima, ili funkcije za pristup nekim korisnički definiranim objektima.

I, na kraju, treći veliki razlog zbog kojega se koriste moduli je sučelje prema korisniku. Ovu ste situaciju sasvim sigurno vidjeli, aplikaciju vam autori isporuče u formi nekoliko datoteka, od kojih je jedna ili nekoliko njih označeno kao „konfiguracijske“. U njima se obično nalazi samo popis varijabli koje definiraju ponašanje i rad osnovne aplikacije. Upravo je to uloga modula jer, ako datoteku u kojoj se varijable nalaze ispravno formatiramo, te varijable možemo koristiti u našem dijelu programa, a da korisnika ne opterećujemo nepotrebnim informacijama i da ga ne prisiljavamo da naš program podešava tražeći relevantne varijable u našem kodu.

Najvažnija stvar koju moramo naglasiti je da je modul zapravo tekstualna datoteka te je njezin sadržaj moguće pročitati i pogledati kako je zapravo realizirano neko programsko rješenje, no istovremeno je ta datoteka za nas „crna kutija“ od koje očekujemo da napravi svoj posao i da nam ne komplicira razumijevanje našeg dijela programa.

U cijelome ovom priručniku bazirani smo na softveru i modulima koji su potpuno besplatni, odnosno objavljeni su pod nekom od licenci koje nam dozvoljavaju korištenje bez plaćanja, uz određene uvjete.

7.2 Kako koristiti module?

Kao primjer ćemo iskoristiti modul *turtle*, koji smo već spominjali kada smo pokazivali kako funkcionira kornjačina grafika. Unutar modula su, dakle, sve potrebne funkcije i metode koje nam omogućavaju pokretanje kornjačine grafike u njezinu grafičkom dijelu, te da upravljamo kornjačom i kompletnim ispisom. Ovaj primjer koristimo jer smo u poglavlju u kojem se bavimo ovom temom naveli i sve najvažnije funkcije koje se koriste u svakodnevnom radu.

Europska unija Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Prva stvar koju morate znati je da svaki modul koji želimo koristiti mora prije korištenja biti instaliran. U ovoj se radionici bavimo samo modulima koji su već dio instalacije Pythona, tako da ih je dovoljno samo pozvati, no uopćeno je pravilo da, module koje želimo koristiti, moramo instalirati.

Postoje dva načina na koje je moguće pozvati neki modul, ovisno o tome na koji način želimo pristupati njegovim funkcijama. Jedan je da pri pozivu u naš program uključimo cijeli modul, dakle sve njegove funkcije, metode i atribute. Način na koji ovo možemo napraviti je:

```
import turtle
```

Nakon toga su sve funkcije dostupne tako da navedemo ime modula, u ovome slučaju **turtle**, te, nakon njega, ime željene funkcije, razdvojene točkom. Dakle, želimo li inicijalizirati kornjačinu grafiku, to radimo ovako:

```
turtle.home()
```

Drugi način na koji je moguće pozvati neki modul je tako da pozovemo samo neke ili sve funkcije iz modula. Struktura je takvog poziva:

```
From turtle import *
```

Nakon ovako pozvanog modula funkcije su nam dostupne izravno i ne moramo se pozivati na ime modula, nego pozivamo samo funkcije, primjerice:

```
home()
```

Svaki od ovih dvaju načina ima svoje prednosti i mane. Pozovemo li cijeli modul, jasno je već pri samom pozivu neke funkcije otkud on dolazi i čemu služi. Pozovemo li samo neke funkcije iz modula, to može ubrzati rad našeg programa jer se memorija ne zauzima nepotrebnim dijelovima funkcija, no istovremeno može naš programski kod učiniti manje čitljivim.

7.3 Često korišteni moduli

Python je poznat po tome da ga prati tisuće, ako ne i desetci tisuća različitih manje ili više korisnih modula. Ovo je jedna od najvećih snaga jezika, jer je korištenjem modula moguće napraviti nevjerojatno složene stvari, a da istovremeno unutar našeg dijela programa ne moramo raditi ništa osim na ispravan način pozvati odgovarajući modul. Zamislite module kao cigle od kojih gradite kuću. Vi kao graditelj ne morate biti u stanju napraviti ni jednu jedinu ciglu, samo ih spojiti u stabilne zidove, te onda te zidove povezati i od toga stvoriti kuću. Moduli omogućavaju da čak i ljudi s minimalnim programerskim iskustvom mogu izvršiti kompleksne zadatke vrlo jednostavno. U nastavku ćemo ponuditi naš izbor najčešće korištenih modula, uz kratki opis njihove funkcionalnosti. Kao i sa svakim drugim izborom, i ovaj je potpuno subjektivan, iako je temeljen na informacijama o tome što drugi programeri koriste u svojem svakodnevnom radu. Postoji mali problem s modulima općenito jer ih je na tržištu toliko da je ponekad teško pronaći neki koji nam u potpunosti odgovara, a

česte su situacije da za neki uobičajeni zadatak imamo i više od jednog raspoloživog modula na tržištu.

Iako su po načinu funkcioniranja identični, postoje dvije grupe modula. Jedna su oni koji su već isporučeni u vašoj instalaciji Pythona, no koje morate pozvati ako ih želite koristiti. Razloga je nekoliko, no najvažniji je brzina izvršavanja, jer je uglavnom riječ o modulima koji omogućavaju napredne funkcionalnosti u Pythonu koje nisu uvijek potrebne, primjerice napredni rad s velikim blokovima teksta, statističke i napredne matematičke funkcije, moduli koji definiraju često korištene varijable i tako dalje.

Mi ćemo spomenuti neke od njih, i najvažnije atribute i funkcije koje iz njih možemo iskoristiti.

Drugu grupu čine moduli koje možemo preuzeti s interneta, i koje nećemo opisivati u ovoj radionici, jer ih jednostavno ima previše.

Krenimo od najčešće korištenog modula **math**. Nakon što ga pozovemo koristeći naredbu:

```
import math
```

na raspolaganju nam je niz vrlo korisnih naprednih matematičkih funkcija kao što su faktorijel, funkcije zaokruživanja, logaritamske i eksponencijalne funkcije, trigonometrijske i kutne funkcije, te važne matematičke konstante. Jedan od primjera je konstanta pi. Umjesto ručnog upisivanja dovoljno je napisati math.pi i Python će upotrijebiti najpreciznu vrijednost ove konstante koja odgovara u danom slučaju.

Drugi često korišteni modul je **numbers**, koji se bavi nekim specifičnim klasama brojeva. Govorimo o kompleksnim, realnim i racionalnim brojevima i svim operacijama koje je nad njima moguće napraviti.

Često je potrebno napraviti neku elementarnu statističku analizu nad nekim skupom podataka, te i za to postoji napredni modul, nazvan **statistics**, u kojem se uglavnom nalaze sve često korištene statističke funkcije, od računanja aritmetičke srednje vrijednosti, statistics.mean(), pa do funkcija koje se bave distribucijama, primjerice variance().

Matematika nije jedina disciplina koju pokrivaju ovakvi izdvojeni moduli. Česti rad s tekstom znači i nekoliko modula koje ćete često koristiti u slučajevima kada vam nije dovoljno ono što nudi osnovni jezik. Primjerice, modul **strings** nudi nekoliko konstanti koje će vam pomoći. Navest ćemo ih u primjerima.

```
>>> import string
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#
$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
>>> string.digits
'0123456789'
```

Riječ je o konstantama koje sadrže sve znakove koji spadaju u neku klasu, primjerice, sva slova koja je moguće ispisati, što najviše koristi onima koji rade na obradi i analizi velikih količina teksta.

Napomena: Nećemo nabrajati više modula. Ovo su samo oni koje često koristimo. Važno je da razumijemo način na koji ih je moguće upotrijebiti jer je procedura svaki put ista. Koristeći naredbu **import** uvezemo modul te onda izravno koristimo njegove funkcije. Iskoristit ćemo i ovu priliku da ukažemo na referencu za cijeli Python, njegovu fantastičnu dokumentaciju dostupnu na internetu na adresi: https://docs.python.org/3/library/index.html.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

8. poglavlje: **Datoteke**

U ovom poglavlju naučit ćete:

- ☑ što su datoteke
- ☑ za što ih koristimo
- ☑ kako ih koristiti.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

8.1 Što su datoteke?

Datoteka ili *file* je glavni način na koji će skripte komunicirati s okolinom. Datoteke kod Pythona mogu biti bilo što, od tekstualnog *loga*, preko slike ili zvuka do podatkovnih struktura kao što su Excel ili Word datoteke. Rad s datotekama u Pythonu malo je neobičan jer funkcioniraju kao objekti pa je njihovo otvaranje i kreiranje realizirano putem metoda i dodjele vrijednosti.

8.1.1 Otvaranje datoteke

Primjerice, da bismo otvorili datoteku i u nju nešto zapisali, sve to u trenutnom direktoriju treba napisati:

```
>>> f=open('test.txt','w')
>>> f.write('Testna poruka u datoteci')
24
>>> f.close()
>>>
```

Brojka koju je Python ispisao označava broj znakova koji su snimljeni.

Prođimo primjerom korak po korak.

Prva je naredba zapravo ona koja definira koju datoteku otvaramo i kako joj pristupamo. Za to koristimo funkciju open(), i dodjeljujemo njezinu vrijednost varijabli. Ovo je nešto što nismo nigdje susreli u cijeloj radionici, no nemoguće je pristup datoteci ostvariti na drugačiji način. Radi se o tome da funkcija pretvara varijablu u objekt, koji onda ima svoje metode – načine pristupa datoteci.

Funkcija open() prihvaća dva argumenta, jedan je ime datoteke, koju može i ne mora sadržavati i lokaciju na kojoj se datoteka nalazi. Ako lokacija nije navedena, podrazumijeva se da govorimo o trenutnoj radnoj mapi u kojoj je pokrenut program. Drugi argument je način na koji pristupamo datoteci, a u našem prvom primjeru riječ je o pisanju. Postoji nekoliko načina na koje možete pristupiti datoteci te ih pregledno navodimo u tablici ispod teksta.

Iduća naredba zapisuje poruku, odnosno niz znakova, u datoteku. Funkcija vraća rezultat koji govori koliko je podataka zapisano.

Na kraju datoteku moramo zatvoriti kako bismo operacijskom sustavu javili da pohrani podatke i da više ne mislimo pisati u datoteku. Ovime je datoteka zatvorena, i njoj više ne možemo pristupiti. Ako želimo još nešto zapisati, potrebno je ponoviti cijelu proceduru.

Slično tomu, da bismo pročitali datoteku:

```
>>> f=open('test.txt')
>>> f.read()
'Testna poruka u datoteci'
>>> f.close()
>>>
```

Primijetimo da u pozivu funkcije open() nismo definirali način rada kao u prvome primjeru (parametar *w* za pisanje u datoteku). Ako ne navedemo taj parametar, Python pretpostavlja da datoteku namjeravamo samo čitati, a ne i pisati u nju.

Druga naredba čita iz datoteke, te moramo napomenuti još nekoliko stvari. Python iz datoteke čita red po red. Kada datoteku otvaramo, čitanje započinje od početka datoteke, slijedno do njezina kraja. Jedina je iznimka korištenje načina pristupa „a“ koji će nas pozicionirati na kraj datoteke, pri čemu onda vrijede druga pravila te je neophodno pomicati se unutar datoteke ako želimo doći do nekog podatka, no to nadilazi potrebno znanje za ovu radionicu.

Tablica 3.: Načini pristupa datotekama

Oznaka načina pristupa	Značenje
r	Otvara datoteku za čitanje. Ovo je zadani način rada.
r+	Otvara datoteku za čitanje i pisanje.
w	Otvara datoteku samo za pisanje. Ako datoteka ne postoji, stvara se nova, ako postoji njezin sadržaj, briše se.
w+	Otvara datoteku za čitanje i pisanje. Ako datoteka ne postoji, stvara se nova, ako postoji, sadržaj se briše.
a	Datoteka se otvara za pisanje, ali u načinu dodavanja sadržaja. Sve zapisano u datoteku dopisuje se na njezin kraj.
a+	Datoteka se otvara za čitanje i pisanje. Ako postoji, pisanje će sadržaj pohraniti na kraj datoteke. Ako datoteka ne postoji, bit će stvorena.

8.1.2 Što još napraviti s datotekom?

Nad datotekama možemo raditi niz operacija, ne samo čitanje i pisanje. Jedna su od glavnih operacija ponavljanja. Spomenuli smo da se datoteka čita red po red, pa dakle možemo izravno korištenjem naredbe **for** tretirati datoteku kao skup redaka te čitati redak po redak i raditi neku radnju nad njime. Python iznimno dobro barata velikim datotekama, posebno ako ih čitamo ovako slijedno, tako da je u stanju otvoriti i najveće datoteke jednako brzo kao i one male. Ovakav pristup posebno dolazi do izražaja ako moramo na brzinu napraviti neku jednostavnu radnju nad nekim velikim tekstom, primjerice, prebrojiti broj pojavljivanja nekog pojma u tekstu, ili prebrojiti naslove.

Popis svih naredbi koje je moguće izvršiti nad datotekom pogledajte naredbom **dir(imeotvorenedatoteke)**.

```
>>> f=open('test.txt')
>>> dir(f)
['_CHUNK_SIZE', '__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__lt__', '__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__',
```

```
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_checkClosed',
'_checkReadable', '_checkSeekable', '_checkWritable', '_finalizing', 'buffer', 'close',
'closed', 'detach', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'line_buffering', 'mode',
'name', 'newlines', 'read', 'readable', 'readline', 'readlines', 'seek', 'seekable', 'tell',
'truncate', 'writable', 'write', 'writelines']
>>>
```

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Zaključak

Programiranje je vrlo složen, apstraktan i izrazito kreativan posao, i pronaći svoj ulaz u tu disciplinu na prvi se pogled čini teškim. Najveći je problem obično otkuda početi, i zapravo smo se upravo zato odlučili na ovu kombinaciju Pythona i osnova programiranja. Python je po svojoj strukturi gotovo identičan onome što nazivamo pseudokodom, tako da je lako čitljiv i razumljiv, a to ujedno znači i da možemo jednostavno pojasniti one najvažnije algoritme i načine na koje je neke probleme moguće riješiti algoritamskim razmišljanjem.

Ova je radionica prekratka da bismo u njoj mogli pokriti sve, pa čak i većinu tema koje se tiču programiranja, tako da smo se pokušali ograničiti na onaj minimalni skup znanja koji je dovoljan da pokažemo što je moguće, te da vas zainteresiramo za stvaranje nekog vlastitog programa. Programiranje se uči vježbanjem i primjerima. Stoga je ono najbolje, što mi možemo pružiti, naoružati vas osnovnim pojmovima i da zagrebete površinu programiranja, programskih jezika i algoritama. Na vama je da nastavite taj rad i pronađete nove izazove.

Europska unija
Zajedno do fondova EU
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI
Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.
Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Popis literature

Budin, L. i dr. 2017. *Računalno razmišljanje i programiranje u Pythonu.* Zagreb: Element.

Budin, L. – Povjerenstvo za uvođenje Informatike kao obaveznog predmeta u osnovnoškolski odgoj i obrazovanje. *Računalno razmišljanje i programiranje.* Preuzeto: 22. svibnja 2018. Dostupno na: https://mzo.hr/sites/default/files/dokumenti/2018/OBRAZOVANJE/Nacionalni-kurikulumi/Edukacija/prezentacija_webinara_racunalno_razmisljanje_i_programiranje_u_visim_razredima_osnovne_skole.pdf.

Deljac S., Dimovsk, Z. 2016. *MojPortal 3.0. – Priručnik za programiranje od 5. do 8. razreda osnovne škole.* Zagreb: Školska knjiga.

Kniewald, I. 2015. *Python – Osnove programiranja u programskom jeziku Python.* Zagreb: SysPrint.

Kalafatić, Z. i dr. 2017. *Python za znatiželjne.* Zagreb: Element.

Knuth, D. 1968. *The art of computer programming.* Addison-Wesley.

Python Software Foundation (Download Python). Preuzeto: 24. svibnja 2018. Dostupno na: https://www.python.org/downloads/.

Python Software Foundation (*Index of Python Enhancements Proposals)*. Preuzeto: 24. svibnja 2018. Dostupno na: https://www.python.org/dev/peps/.

Python Software Foundation (*Python 3.6.5 documentation)*. Preuzeto: 24. svibnja 2018. Dostupno na: https://docs.python.org/3/.

Reference

Budin, Leo – Povjerenstvo za uvođenje Informatike kao obaveznog predmeta u osnovnoškolski odgoj i obrazovanje. *Računalno razmišljanje i programiranje.* Preuzeto: 22. svibnja 2018. Dostupno na: https://mzo.hr/sites/default/files/dokumenti/2018/OBRAZOVANJE/Nacionalni-kurikulumi/Edukacija/prezentacija_webinara_racunalno_razmisljanje_i_programiranje_u_visim_razredima_osnovne_skole.pdf.

Knuth, D. 1968. *The art of computer programming.* Addison-Wesley.

Europska unija
Zajedno do fondova EU

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Više informacija o EU fondovima možete pronaći na: www.strukturnifondovi.hr

Impressum

Nakladnik: Hrvatska akademska i istraživačka mreža – CARNET

Projekt: „e-Škole: Uspostava sustava razvoja digitalno zrelih škola (pilot-projekt)“

Urednica: Alina Čabraja, prof.

Autori: dr. sc. Lada Maleš, Jasmin Redžepagić, bacc. ing. Comp, Daniel Rakijašić, prof.

Lektorica: Jasna Bićanić

Recenzent: dr. sc. Saša Mladenović

Priprema, prijelom: Algebra

Zagreb, srpanj 2018.

Kontakt

Hrvatska akademska i istraživačka mreža – CARNET

Josipa Marohnića 5, 10000 Zagreb

tel.: +385 1 6661 555

www.carnet.hr

Više informacija o EU fondovima možete pronaći na mrežnim stranicama Ministarstva regionalnoga razvoja i fondova Europske unije: www.strukturnifondovi.hr

Ovaj priručnik izrađen je s ciljem podizanja digitalne kompetencije korisnika u sklopu projekta e-Škole: Uspostava sustava razvoja digitalno zrelih škola (pilot-projekt), koji sufinancira Europska unija iz europskih strukturnih i investicijskih fondova. Nositelj projekta je Hrvatska akademska i istraživačka mreža – CARNET.