

Python 3 - Uvod u programski jezik Python

autor: Milenko Letic - <https://programiranje.ba>

e-mail: milenko.letic@programiranje.ba

godina: 2020

izmjene: verzija: 0.5

- izvorni kod za igre vjesala i potapanje brodova
- rad sa datotekama
- slike prilikom instalacije python i pycharm
- dopune teksta
- page break
- duzina linije

verzija: 0.4

- zadaci za vježbanje (lecije 1 i 2)

O kursu

Kurs **Python 3 - Uvod u programski jezik Python** je dizajniran za sto jednostavnije učenje. Namijenjen je početnicima i kao takav pokusava da zadrži vasu pažnju, postepenim dodavanjem manjih detalja kako bi se kreirala jasna slika šta je to Python programski jezik, gdje ga sve možemo koristiti, kako nam može biti od pomoći na dnevnoj bazi ali ono što je najvažnije kako vam može obezbijediti budućnost na polju informacionih tehnologija.

Nije tajna da je programer danas, kao i u prošlosti, veoma cijenjeno zanimanje a da je potražnja na tržištu rada veoma velika za ovim kadrovima što će i ostati u budućem periodu.

Ono što ćete postići na kraju ovog kursa, i što bi trebala biti mjera da li ste uspješno usvojili znanje, jeste da ćete biti u mogućnosti samostalno da kreirate osnovne programe, koji vam mogu biti od velike koristi prilikom automatizacije, ali ono najvažnije razumjećete principe programiranja i moći ćete u potpunosti da se oslonite na svoje steceno znanje. Takođe, lakše ćete moći da naučite druge programske jezike i da se upustite u ozbiljnije programiranje.

Citavo vrijeme ćete biti vodjeni kroz kurs na vama je samo, da u par sati koliko kurs traje, pomno pratite sve što instruktor radi, odradjujete vježbe nakon svake lekcije (rješenja su uvijek data na početku sledeće lekcije ili na kraju knjige koja dolazi sa kursom, takođe besplatno).

Primijetice da nazivi Python fajlova imaju malo čudnu konvenciju, ali na način kako su fajlovi nazvani autor je olaksao organizaciju izvornog koda kao i referisanje studenata na određeni kod.

Prilikom kreiranja kursa i primjera mahom su korišteni karakteri iz poznate anime serije **Zmajeve Kugle (Dragon Ball)**. Ponekad su dijelovi teksta uzeti iz pjesama **Miladina Sobica, Tome Zdravkovića, Dubioze Kolektiv i Ramba Amadeusa**. Takođe, postoje dijelovi teksta iz pjesmica za djecu, sve u cilju da se koncepti programiranja usvoje što je lakše i prirodnije moguće ali i da predavanje drži pažnju te da bude zanimljivo tokom cijelog kursa.

tbd. Igrice vjesala i potapanje brodova. tbd. python možete koristiti na svim poljima, automatizaciju, obradu ogromne količine podataka, web, igrice ...

Sva pitanja vezana za kurs možete postaviti preko e-mail adrese

pitanja-python@programiranje.ba

ili na YouTube kanalu

<https://www.youtube.com/channel/UCSYrkPyht9PAXMhAbkGTbsQ> (https://youtube.com/c/channel_name kada bude spreman tbd.)

Par rijeci o Python programskom jeziku

Python je jedan od rijetkih programskih jezika, koji je u isto vrijeme jednostavan i mocan. Nemojte se iznenaditi ako vam učenje Python programskog jezika ide veoma lako i brzo savladavate lekcije, jer to i jeste cilj Python kao programskog jezika, da vam omoguci usmjeravanje paznje na rješavanje konkretnog problema i pronalaznje rjesenja, umjesto da morate voditi racuna o sintaksi i strukturi programskog jezika kao sto je slucaj kod vecine ostalih programskih jezika.

Python je programski jezik opste namjene, sa elegantnom sintaksom i dinamikom, zajedno sa svojom prirodom interpretera, ali i mogucnoscu objektno orijentisanog nacina programiranja, pored sto je moze pisati skripte nudi mogucnost brzog razvoja aplikacija na razlicitim operativnim sistemima.

Jezik Python je kreiran u kasnim 80-tim od strane holandskog programera koji se zove **Guido van Rossum**, a naziv Python je dobio po BBC emisiji "Leteci Cirkus Monti Pajtona" (Monty Python's Flying Circus), cime je kreator htio na saljiv nacin da predstavi i programiranje kao takvo, jednostavno i zabavno.

Dizajn filozofija Python-a se svodi na jednostavnu citljivost, dakle u prvom planu ima za cilj sto lakse citanje i pisanje koda. Ovo se postize koristenjem white-space za odvajanje blokova koda umjesto vec dobro poznatog i ustaljenog nacina koristenja uglastih zagrada `{ }` i tacke zareza `;`.

Kako pokrenuti Python

Generalno sav python kod se pokrece koristenjem interpretera. Najpopularniji i originalni interpreter je CPython, zato sto je implementiran u C programskom jeziku. Takodje, postoji i par drugih interpretera, a mnogi od njih su implementirani u razlicitim jezicima od C-a, kao sto su Java ili C# (C sharp).

Najcesce koristen interpreter CPython, koristi automatski garbage kolektor (sakupljac smeca 😊), kako bi obezbijedio nesmetano i efikasno upravljenje memorijom kompjutera. Python je siroko poznat po usvajanju ne tradicionalne, minimalne sintakse, bazirane na white space, i dizajnu koju tezi cistom i citljivom kodu.

Verzije Python-a

Prije samo par mjeseci (pisano Mar. 2020), ako biste htjeli instalirati Python na vasem racunaru, dosli biste u konfuznu situaciju, jer Python, za razliku od mnogih drugih programskih jezika, ima dvije glavne (major), ne kompatibilne verzije koje su podjednako u širokoj upotrebi

Python verzije 2.7.3, released u 2012, je zadnja verzija popularnog Python-a 2 koji je released. Ova verzija je uglavnom u potpunosti kompatibilna unazad sa svim prethodnim verzijama.

Godine 2008, kreator, Guido van Rossum odlucio je da ocisti Python bazu (codebase) i rekonstruisi dosta drugih stvari u Python 2 koje mu se nisu svidjale, s toga je kreirao Python 3.

Python 3 je prihvatian ali veoma oprezno i polako na pocetku, najvise iz razloga sto nije kompatibilan unazad sa prethodnom verzijom Python 2, i zato je postojao ogroman eko-sistem biblioteka napisanih za Python 2 koje nece raditi sa novom verzijom Python-a 3.

Ovih dana Python 3 eco-sistem je uveliko pohvatao i izjednacio se sa prethodnom verzijom, sto nas dovodi do zakljucka da je Python 3 logican izbor za sve nove developere koji planiraju uciti ovaj programski jezik. Python 3 je verzija koju cemo ujedno obraditi u ovom kursu.

Karakteristike Python-a

Jednostavnost

Jednostavnost citanja i pisanja koda u Python-u, ponekad se granici sa citanjem nekog teksta napisanog na standardnom jeziku koji koristimo u svakodnevnoj komunikaciji. Ovo donosi ogromnu prednost Python programskom jeziku u odnosu na druge.

Lak za učenje

Kroz samo para dana i vec mozete kreirati programe, a u nekoliko sati rada sa Python-om vec cete se osjecati kao da upravljate avionom.

Besplatan i otvorenog koda (Free and Open Source)

Kada je u pitanju razvoj programskog jezika Python, od velikog je znacaja ova osobina, sto cini njegov kod u potpunosti otvorenim za daljnje distribuiranje, kao i specificne dijelove koda, ali omogucava programerima iz cijelog svijeta da rade na daljnjem razvoju ovog programskog jezika.

Jezik visokog nivoa (High-level)

Prilikom pisanje koda u Python-u mozete se u potpunosti posvetiti rjesavanju problema, bez da vam paznju odvlace stvari kao sto su upravljanje memorijom, striktno definisanje tipova promjenjivih ...

Prenosivost / Portabilnost

Python kod je portovan (prilagodjen da bi radio na drugim platformama), sto mu omogucava izvršavanje na bilo kojoj drugoj platformi, bez dodatnih intervencija nad izvornim kodom. Ovo znaci da ste u potpunosti ne zavisni od operativnog sistema ili tipa arhitekture na kojoj radite. Python kod ce se izvršavati isto kako na Windows-u, tako i na Linux, FreeBSD, Macintosh-u, Solaris-u, OS/2, Amigi ali i PlayStation-u, PalmOS-u, cak i na PocketPC-u! Postoje platforme poput Kivy-ja, koje vam omogucavaju kreiranje igrice za iOS ili Android.

Interpreter

Mnogi drugi programski jezici, poput C, C++ ili Java zahtijevaju kompajliranje koda, sto znaci prevodjenje/konvertovanje koda sa **Izvornog koda** (Source Code) u binarni jezik (jezik nula i jedinica), razumljiv jedino kompjuteru. Prilikom pokretanja kompajlera, linker software-e kopira program sa vasesg diska u radnu memoriju kompjutera i tada ga pokrace. Python-u nije potreban kompajler/prevodilac iz izvornog koda u binarni, on se u potpunosti oslanja na interpreter, sto znaci da se program pokrece direktno iz **Izvornog koda**, naravno u pozadini Python pretvara izvorni kode u neki medju kod, koji se jos nazive bajtkode (bytecode), a zatim prevodi bajtkode u jezik koji kompjuter razumije. Iako ovo zvuci duze i komplikovanije od kompajliranja sve se desava u realnom vremenu, tako da nema potrebe za vas kao programera da brinete da li je program kompajliran i uredno povezan (linkovan) sa potrebnim bibliotekama, kao ni da li su sve biblioteke učitane ili ne.

Objektno orijentisan

Pored proceduralno-orijentisanog (niz procedura ili funkcija koje se mogu upotrebljavati vise puta za rjesenje problema), Python u potpunosti podrzava objektno-orijentisani nacin programiranja. Prilikom objektno-orijentisanog nacina programiranja, koristeci objekte i funkcije, Python uveliko uproscava i pojednostavljuje pristup objektno-orijentisanom nacinu programiranja, u poredjenju sa velikanima kao sto su C++ ili Java.

Prosiriv

U slucaju da trebate sakriti neki dio koda (kompajlirati da sprijecite mogucnost citanja Izvornog koda) ili jednostavno zelite da se brze izvrsava (pisanje u C ili C++), mozete jednostavno taj dio kodirati u drugom programskom jeziku i pozvati ga direktno iz Python-a.

Veoma vazno je naglasiti prosirive biblioteke Python-a. Standardne biblioteke Python-a su ogromne. Mogucnosti su nevjerovatne, koriscenjem samo standardnih biblioteka mozete upravljati bazama podataka, serverima, raznim vrstama datoteka, grafickim okruzenjem ... Pored standardnih postoje druge biblioteke koje mozete koristiti unutar vasesg koda a mozete ih pronaci na oficijelnoj stranici <https://pypi.org>.

Nakon nekog vremena programiranja u Python-u shvaticete da je vecinu stvari neko vec napisao i sto je najbolje vi nemate potrebu da izmisljate toplu vodu mozete ove vec napisane dijelove koda jednostavno integrisati u svoj program i koristiti.

Ugradiv u druge programe

Python moze biti ugradjen u jezike poput C i C++ kako bi obezbijedili mogucnost skriptovanja.

Priprema radnog okruzenja

Izbor editora teksta i **Integrisanog razvojnog okruzenja** IDE (Integrated Development Environment)

Izbor tekst editora

Za pocetnike, se preporucuje koristenje nekog jednostavnog tekst editora kao Notpad++, Sublime, VisualStudio Code ...

Izbor Interisanog razvojnog okruzenja

Vecina programera odabere pisanje Python koda, koristenjem specijalnog integrisanog razvojnog okruzenja. Trenutno tri najistaknutija za Python su Eclipse, PyCharm i Netbeans. Za potrebe kursa, koristiceemo **PyCharm**.

Instalacija Windows

Korak 1.

Nakon sto otvoritmo stranicu <https://python.org> potrebno je da skinemo instalacijski paket za trenutnu aktuelnu verziju Python-a.

Za vrijeme pisanje kursa verzija Python-a je bila 3.8.2, sto se moze razlikovati u vasem slucaju ali princip instalacije je isti ili slican.

Iz padajuceg menija "Downloads" potrebno je izabrati opciju "Windows"

**Korak 2.**

Na sledecoj stranici je potrebno kliknuti na "**Latest Python 3 Release - Python 3.8.2**"



Korak 3.

Sada je potrebno skrolati do dna stranice



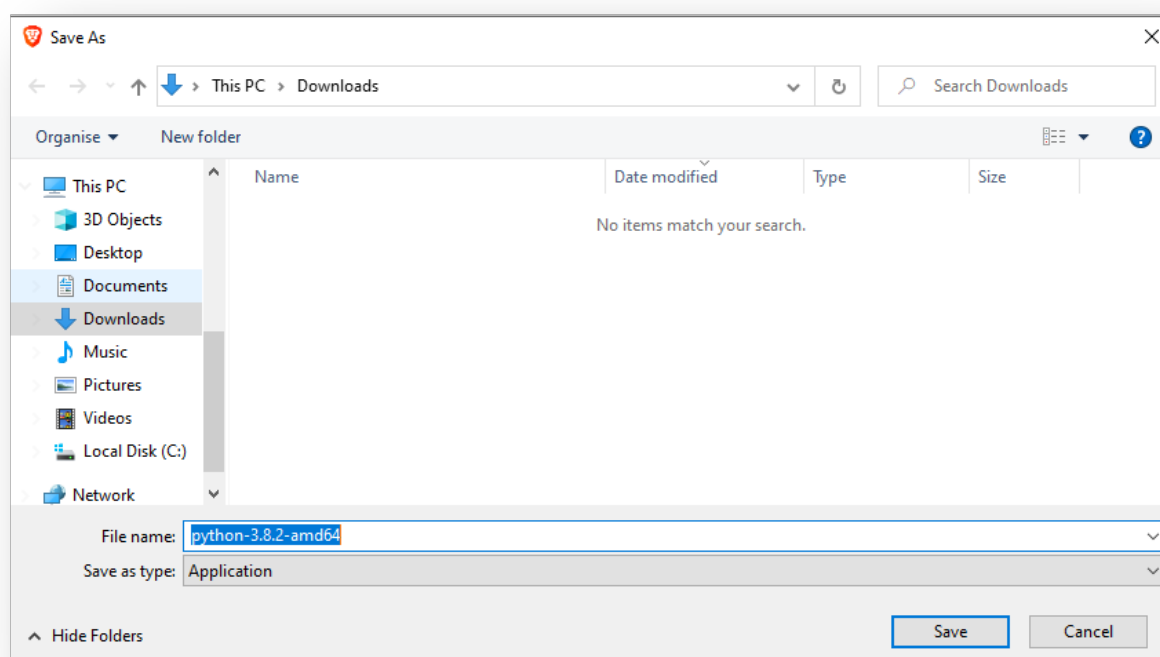
sve dok ne dodjemo do liste fajlova gdje mozemo skinuti instalacioni fajl za nas sistem, koji je u nasem slucaju **"Windows x86-64 executable installer"**

Files

Version	Operating System	Description	MD5 Sum
Gzipped source tarball	Source release		f9f3768f757e34b342dbc06b41cbc844
XZ compressed source tarball	Source release		e9d6ebc92183a177b8e8a58cad5b8d67
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	f12203128b5c639dc08e5a43a2812cc7
Windows help file	Windows		7506675dcbb9a1569b54e600ae66c9fb
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	1a98565285491c0ea65450e78afe6f8d
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	b5df1cbb2bc152cd70c3da9151cb510b
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	2586cdad1a363d1a8abb5fc102b2d418
Windows x86 embeddable zip file	Windows		1b1f0fc5ee8601f160cfad5b560e3a7
Windows x86 executable installer	Windows		6f0ba59c7dbeba7bb0ee21682fe39748
Windows x86 web-based installer	Windows		04d97979534f4bd33752c183fc4ce680

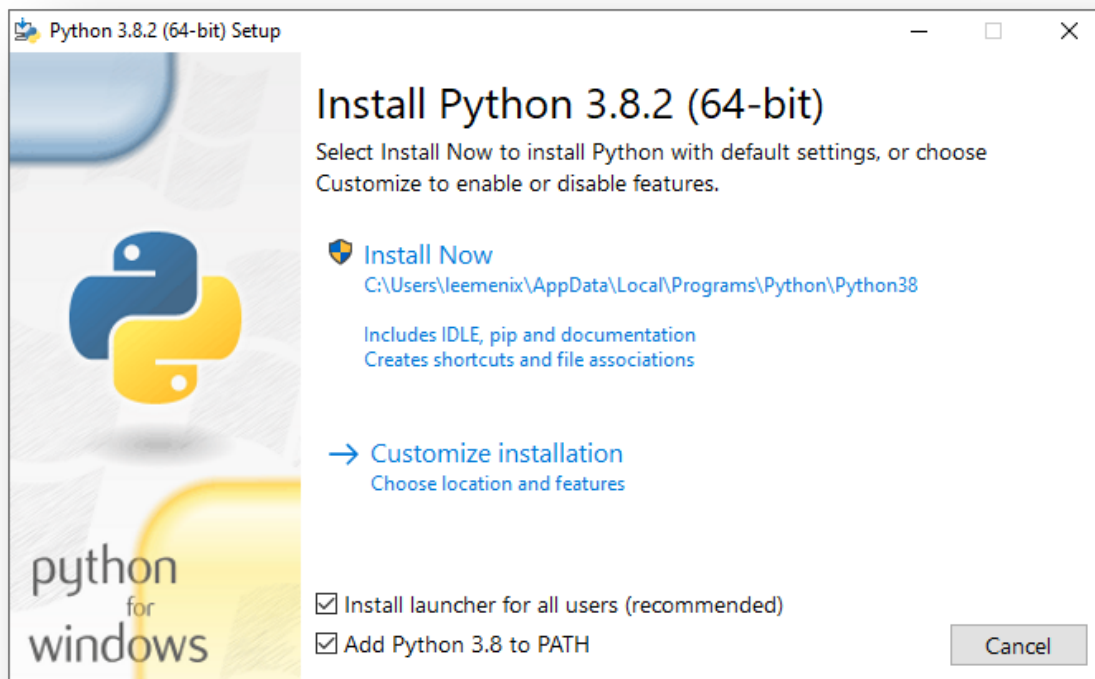
Korak 4.

Odaberemo lokaciju gdje zelimo sacuvati instalacioni fajl i sacuvamo ga sa klikom na dugme **"Save"**

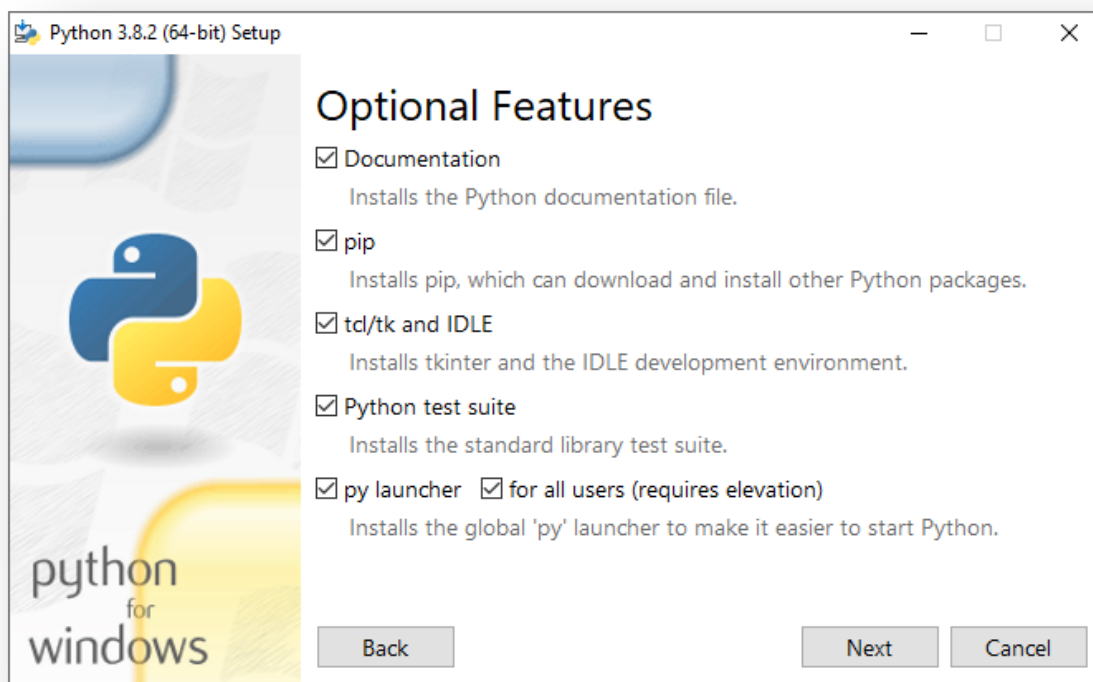


Korak 5.

Pokrenemo instalaciju Python-a, klikom na skinuti fajl. Veoma je vazno da u ovom koraku **prvo** cekiramo opciju **"Add Python 3.8 to PATH"** a nakon toga kao tip instalacije izaberemo **"Customize installation"**.

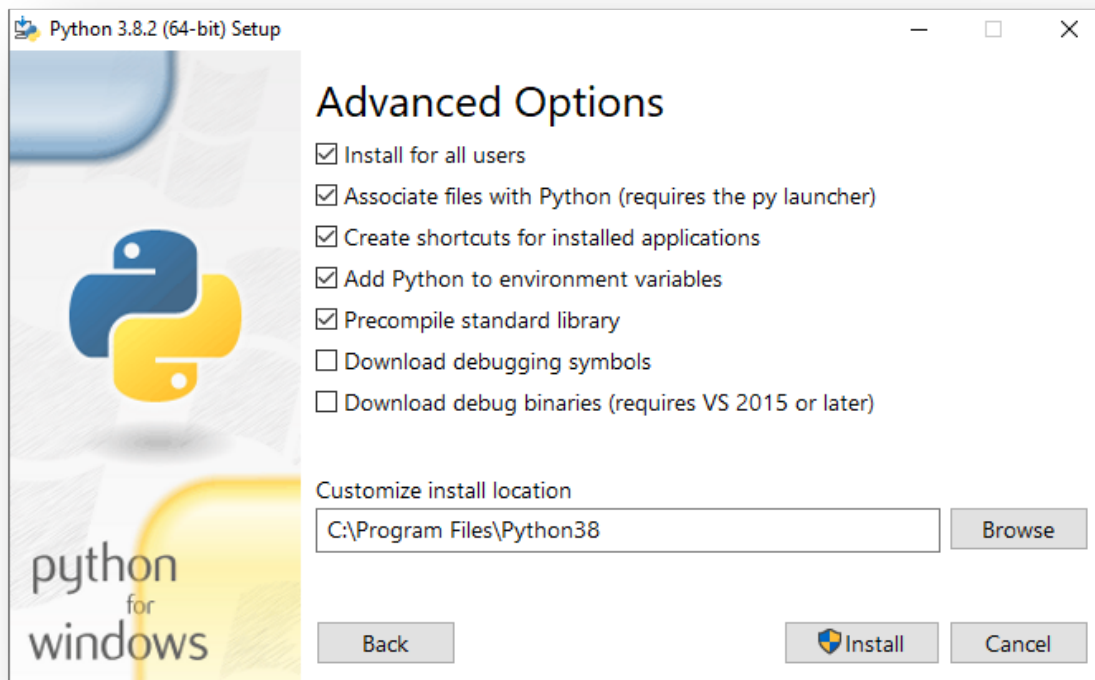
**Korak 6.**

Pod **"Optional Features"**, cekirajte sve opcije i kliknite na dugme **"Next"**

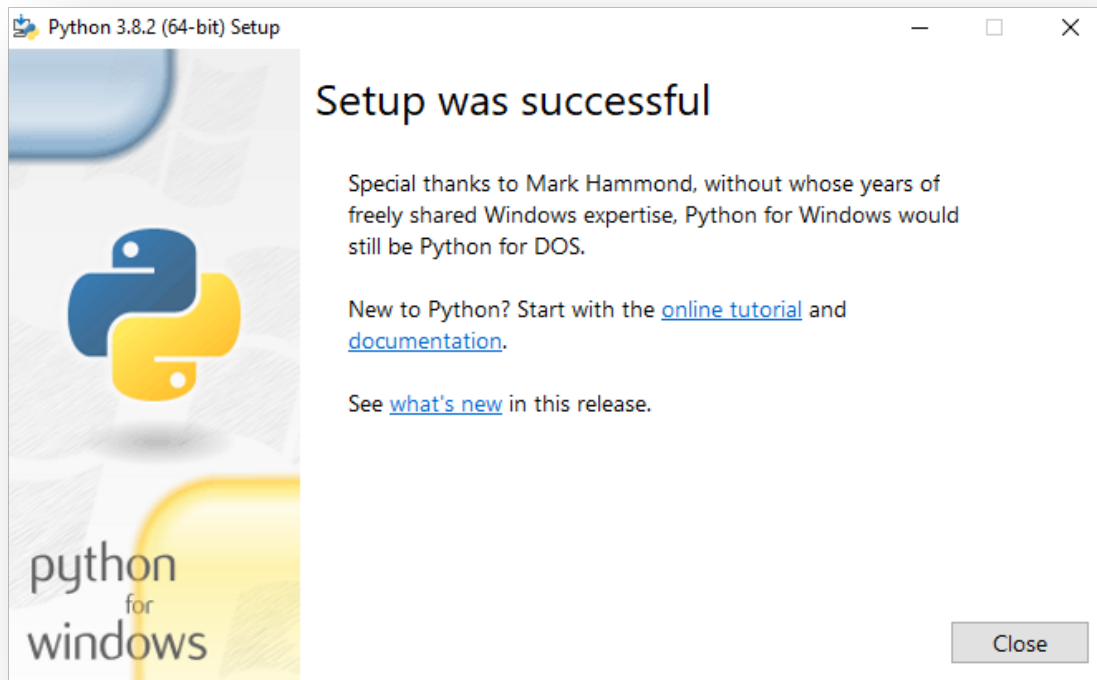


Korak 7.

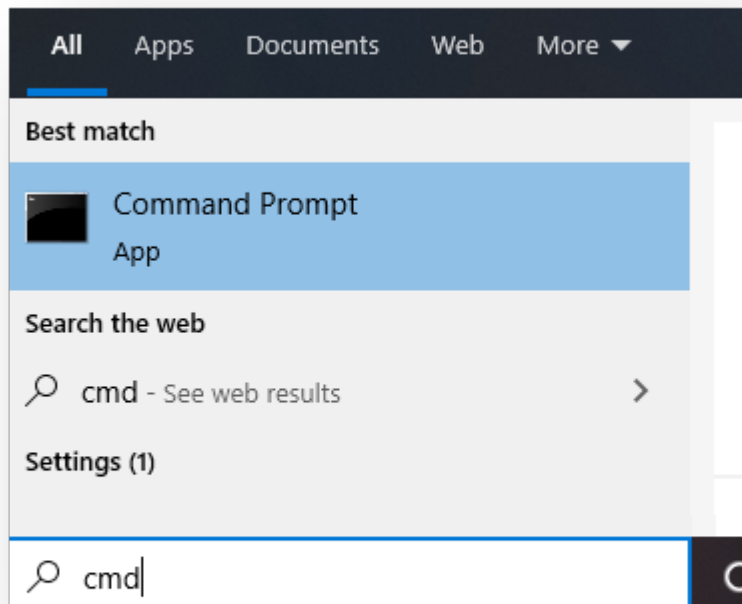
Pod **"Advanced Options"**, veoma je vazno da cekriate opciju **"Install for all users"** i kliknite na dugme **"Install"**. Primijeticete da se putanja do instalacije **"Customize install location"** automatski mijenja kada cekiramo navedenu opciju.

**Korak 8.**

Sada je vazno sacekati da se instalacija zavrshi i mozete kliknuti na **"Close"**

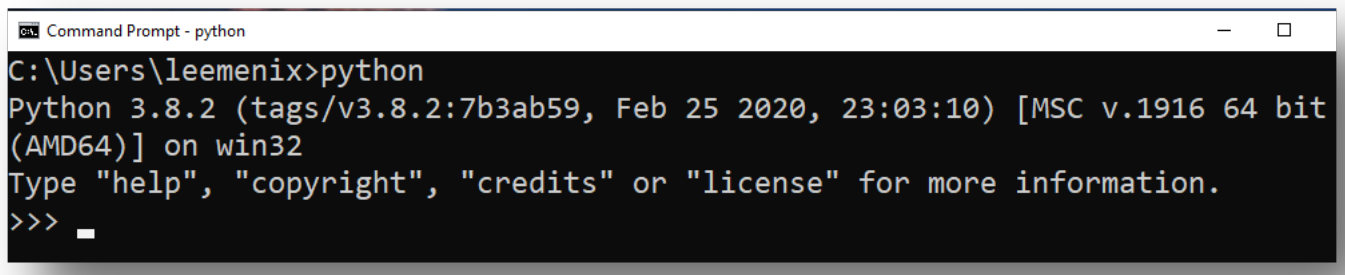
**Korak 9.**

Kako bi smo bili sigurni da je instalacija kompletna i uspjesna, pokrenucemo **"cmd"** komandni prompt, tako sto cemo u Windows Search unijeti cmd i kliknuti na **""Command Prompt"**.



Korak 10.

Ostaje jos da ukucamo komandu **"python"** i udarimo **"Enter"**, trebali bi dobiti Python shell, kao na slici



PyCharm installation

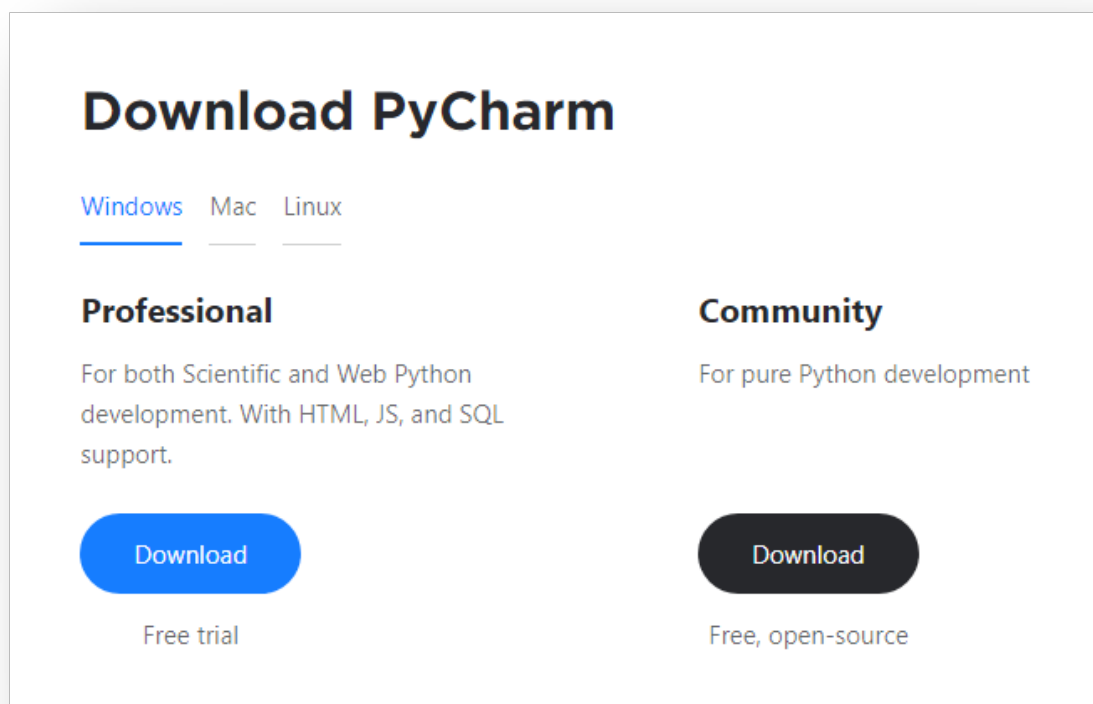
Korak 1.

Nakon sto otvoritmo stranicu <https://www.jetbrains.com/pycharm/> potrebno je da skinemo instalacijski paket za trenutnu aktuelnu verziju PyCharm-a, jednostavnim klikom na dugme **"DOWNLOAD"**.

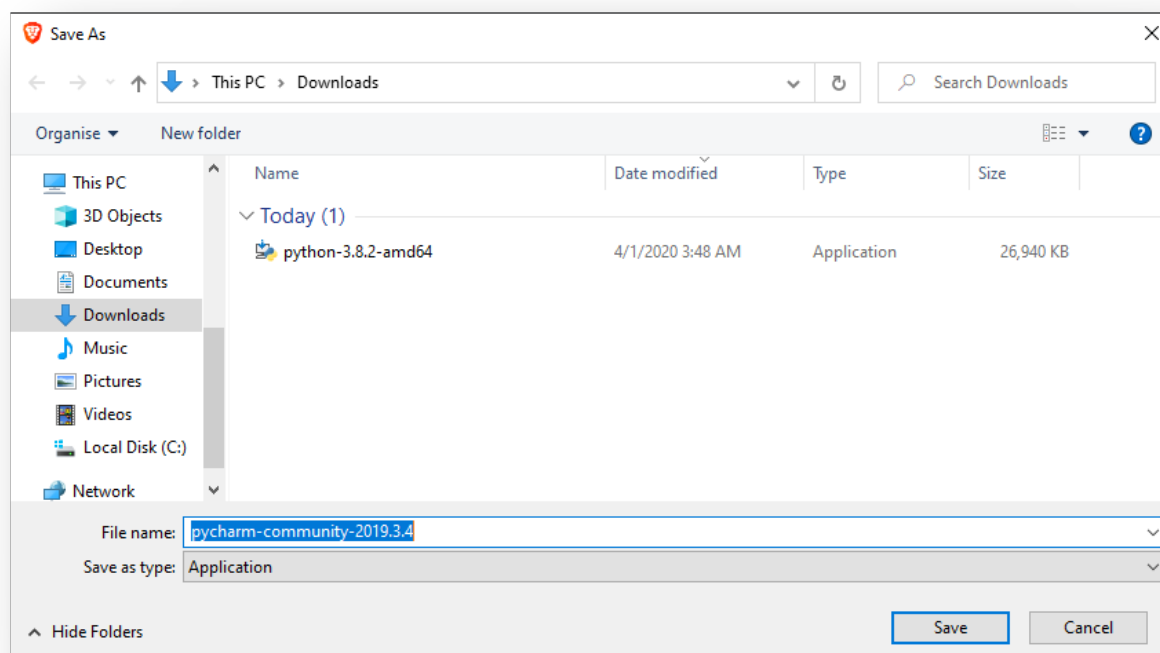


Korak 2.

Bitno je da izaberemo verziju "**Community**", i nakon sto kliknemo na "**Download**" dobicemo opciju da sacuvamo instalacioni fajl.

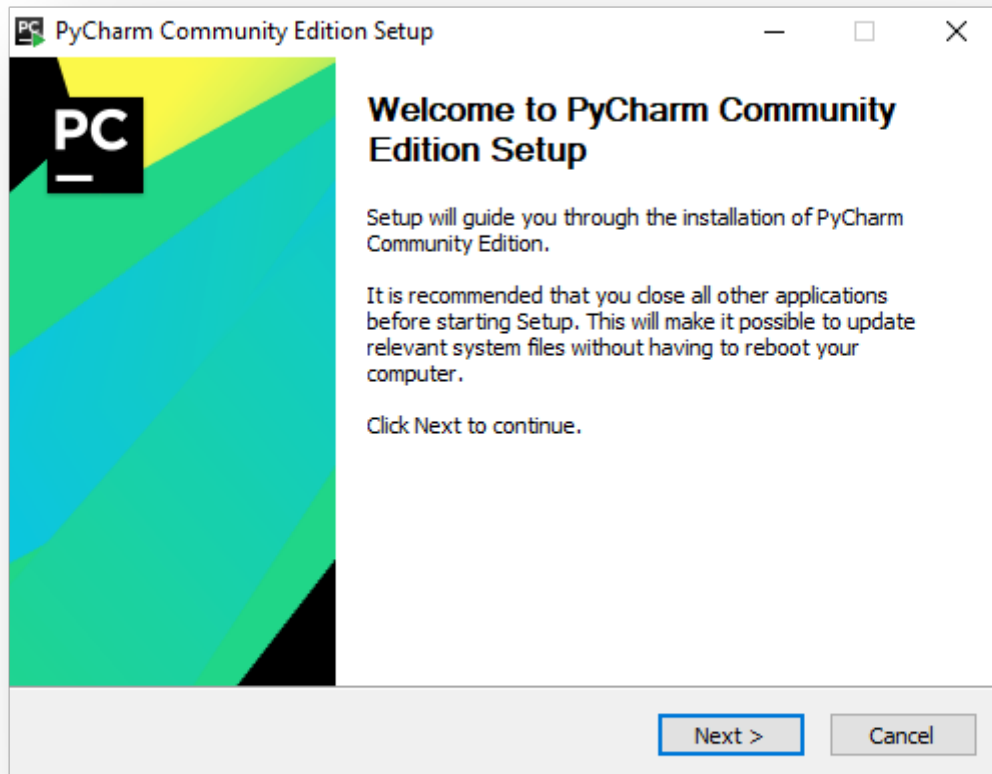
**Korak 3.**

Odaberemo lokaciju gdje zelimo sacuvati instalacioni fajl i sacuvamo ga sa klikom na dugme **"Save"**

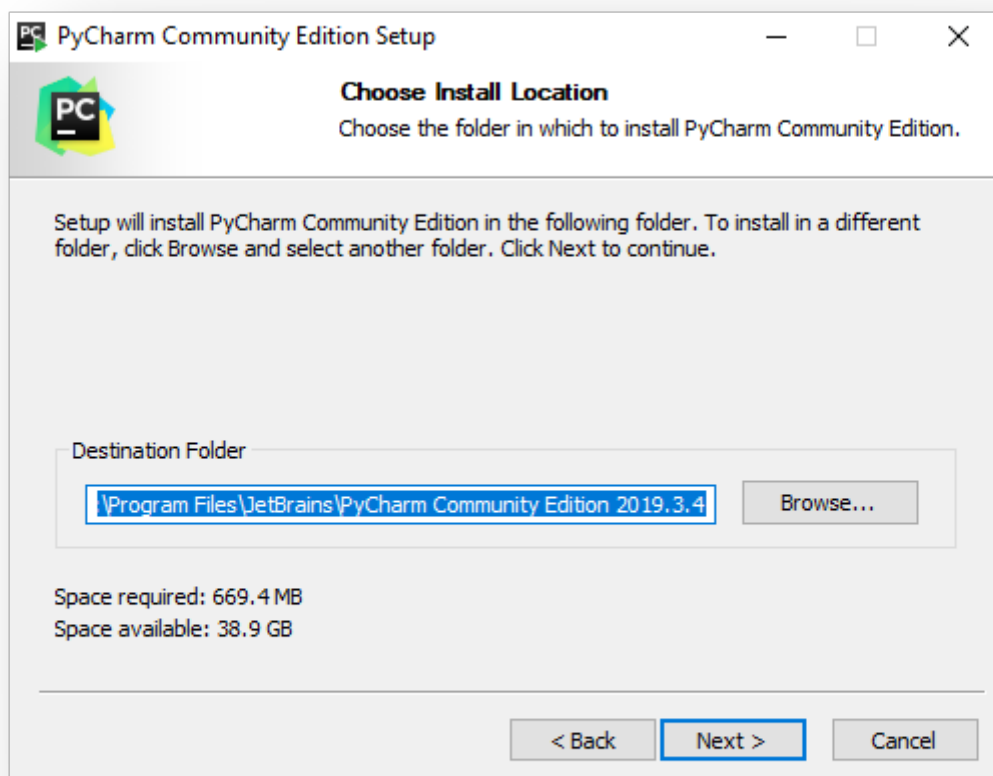


Korak 4.

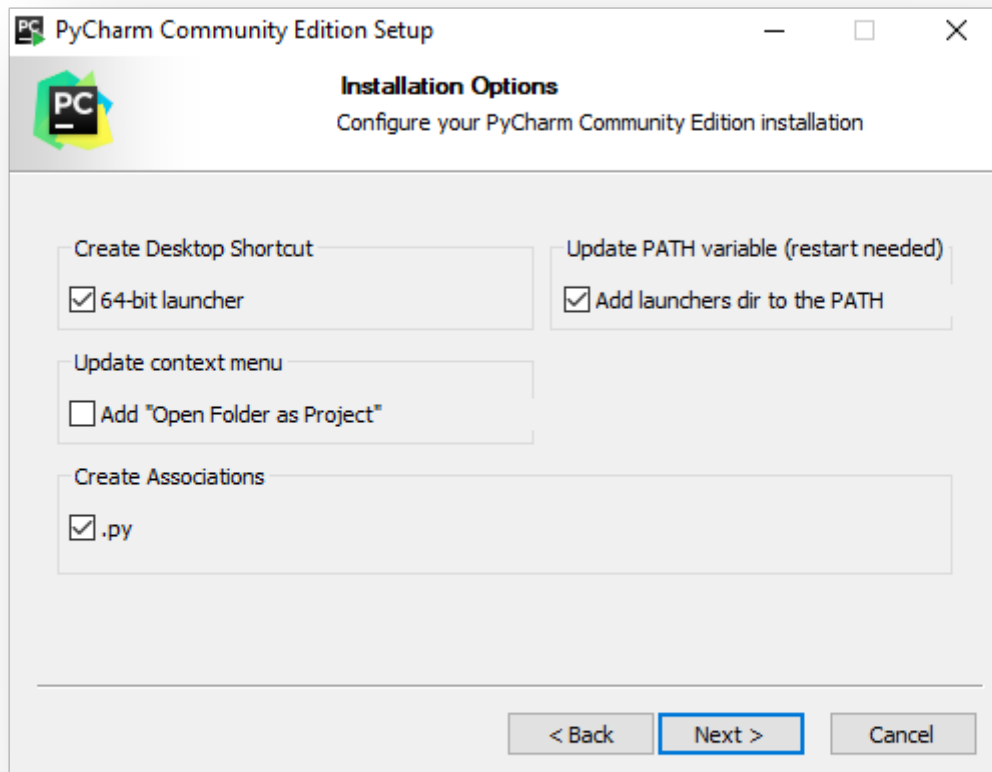
Pokrenemo instalaciju PyCharm-a, klikom na skinuti fajl. Sada je potrebno kliknuti na dugme **"Next"**.

**Korak 5.**

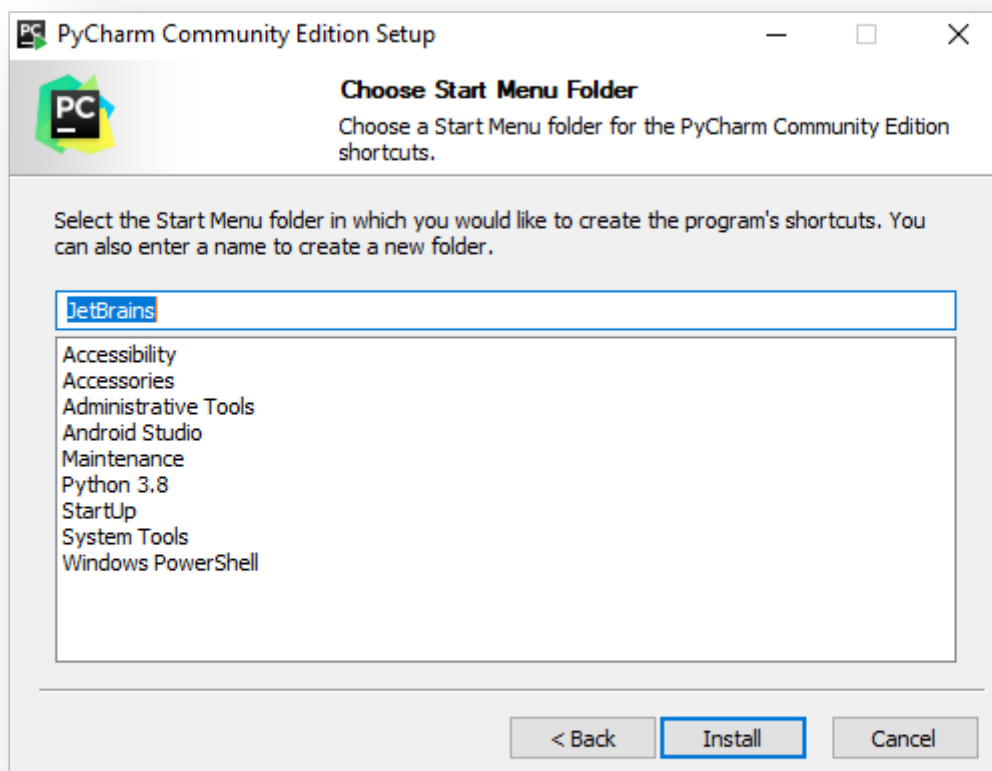
Ponovo izaberite dugme **"Next"**.

**Korak 6.**

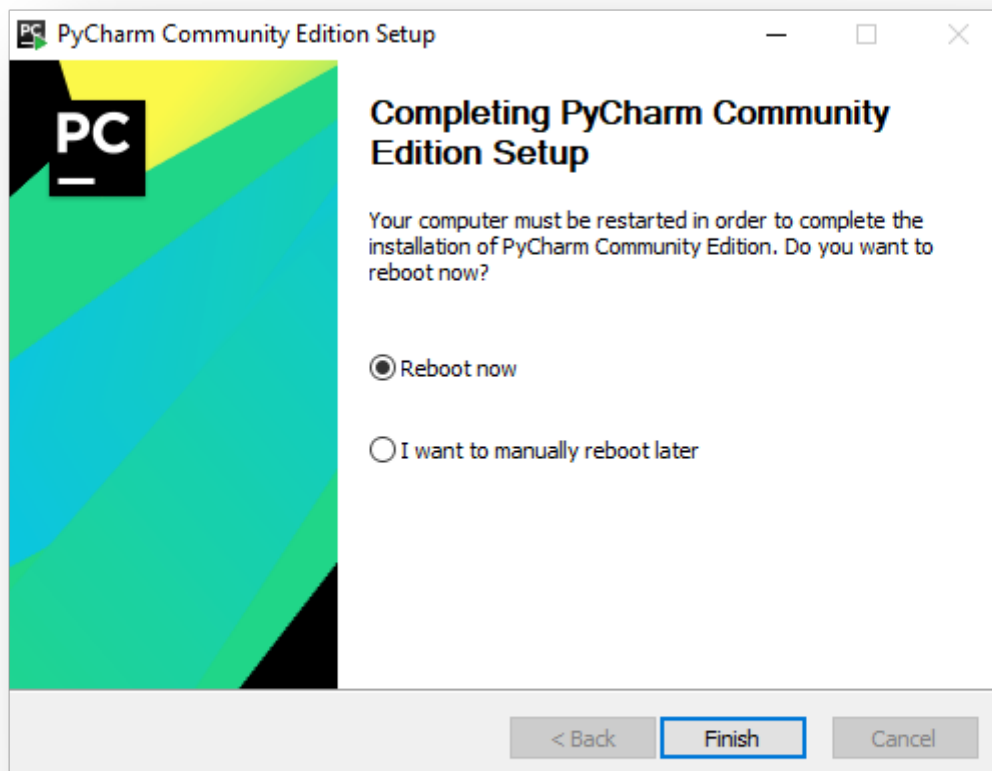
U prozoru **"Installation Options"**, bitno je da cekirate sledece opcije **"64-bit launcher"**, **"Add launchers dir to the PATH"**, **".py"**.

**Korak 7.**

U ovom koraku je bitno kliknuti na dugme **"Install"**.

**Korak 8.**

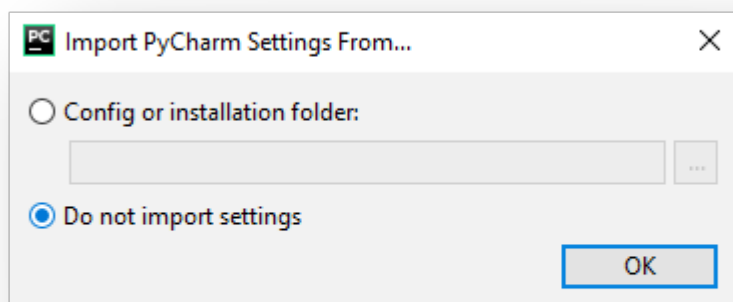
Mozete kliknuti na dugme **"Finish"**. Nakon sto se sistem restartuje mozemo krenuti sa osnovnim podesavanjem PyCharm radnog okruzenja.



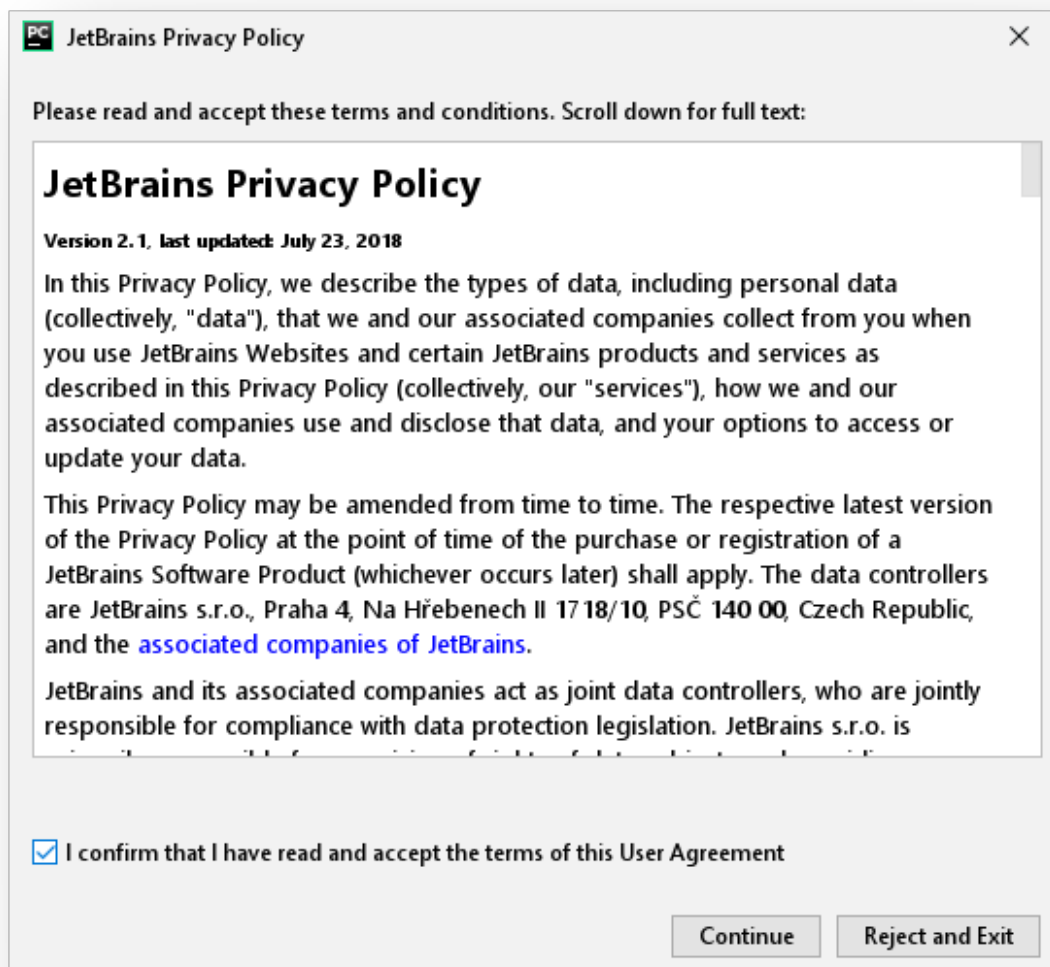
Podesavanje PyCharm-a i nas prvi program "Zdravo Svijete"

Korak 1.

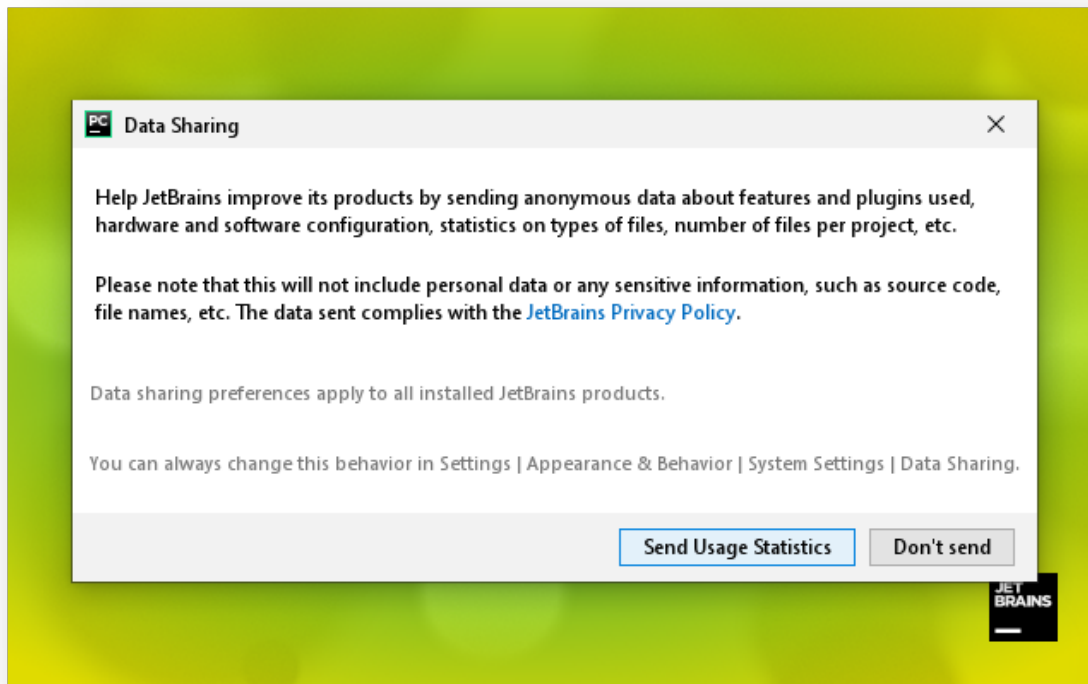
Prilikom prvog ili inicijalnog pokretanja PyCharm programa potrebno je da izvršimo osnovna podesavanja, kako bi prilagodili program za potrebe kursa i lakše pracenje. Na ovom koraku mozete ostaviti cekiranu opciju **"Do not import settings"** i nastaviti dalje sa klikom na dugme **"OK"**.

**Korak 2.**

U ovom koraku je potrebno cekirati polje **"I confirm that I have read and accept the terms of this User Agreement"**, cime se slazemo sa polisom koristenja ovog PyChar programa i naravno za dalje kliknuti dugme **"Continue"**.

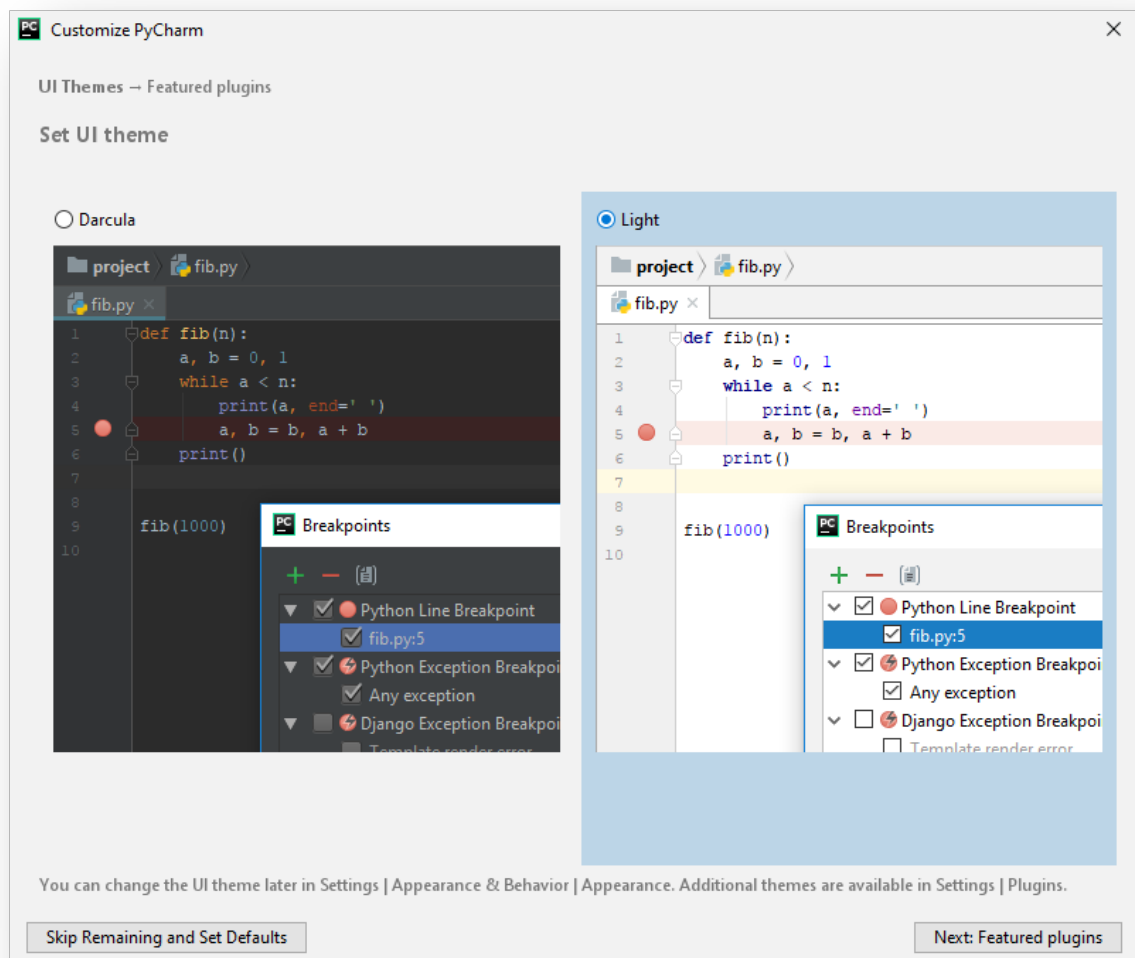
**Korak 3.**

Obzirom da ne zelimo slati nikakvu statistiku sa naseg kompjutera, ovde cemo izabrati opciju **"Don't send"**.



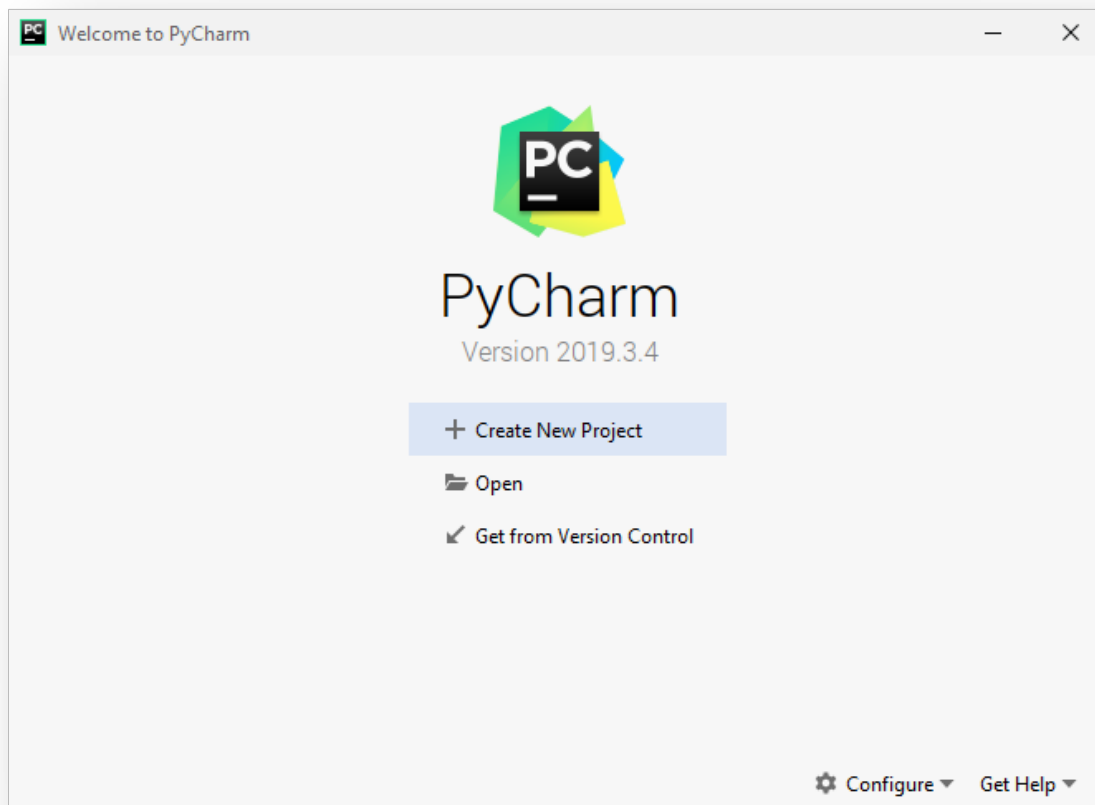
Korak 4.

Ovaj korak se odnosi na izbor teme, za potrebe kursa koristiceemo **"Light"** temu, naravno vi mozete izabrati koja vam vise odgovara. Konacno mozemo kliknuti na dugme **"Skip Remaining and Set Defaults"**.

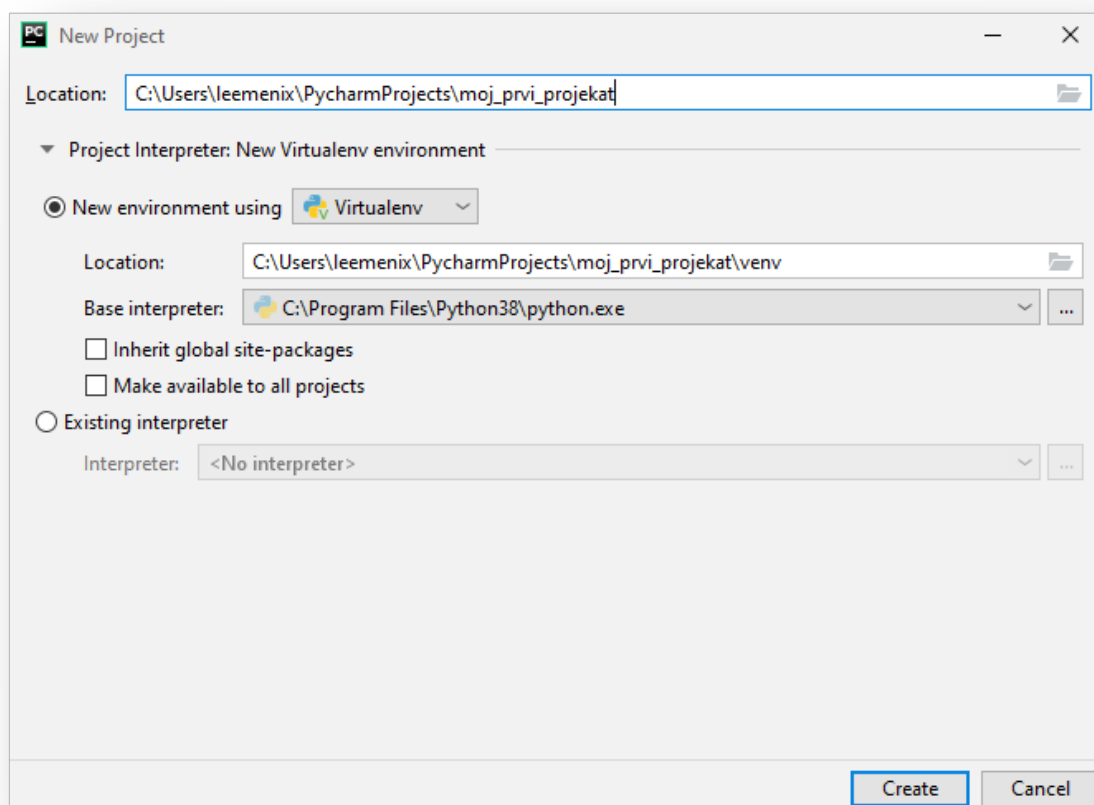


Korak 5.

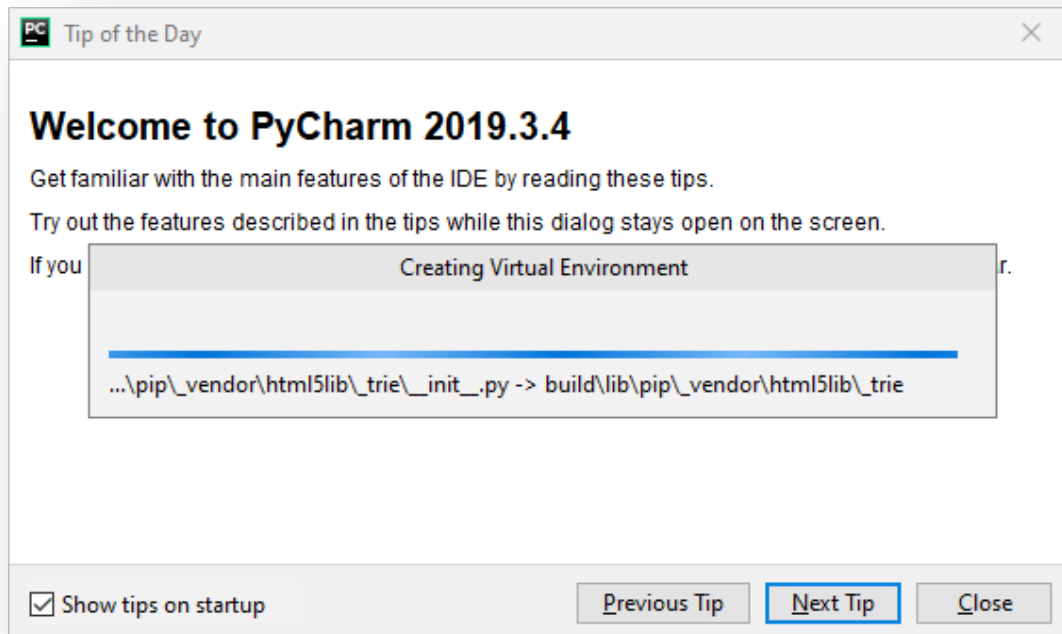
U sledecem prozoru mozemo odabrati opciju **"Crate New Project"**, nakon cega moramo podesiti radno okruzenje za novi projekt.

**Korak 6.**

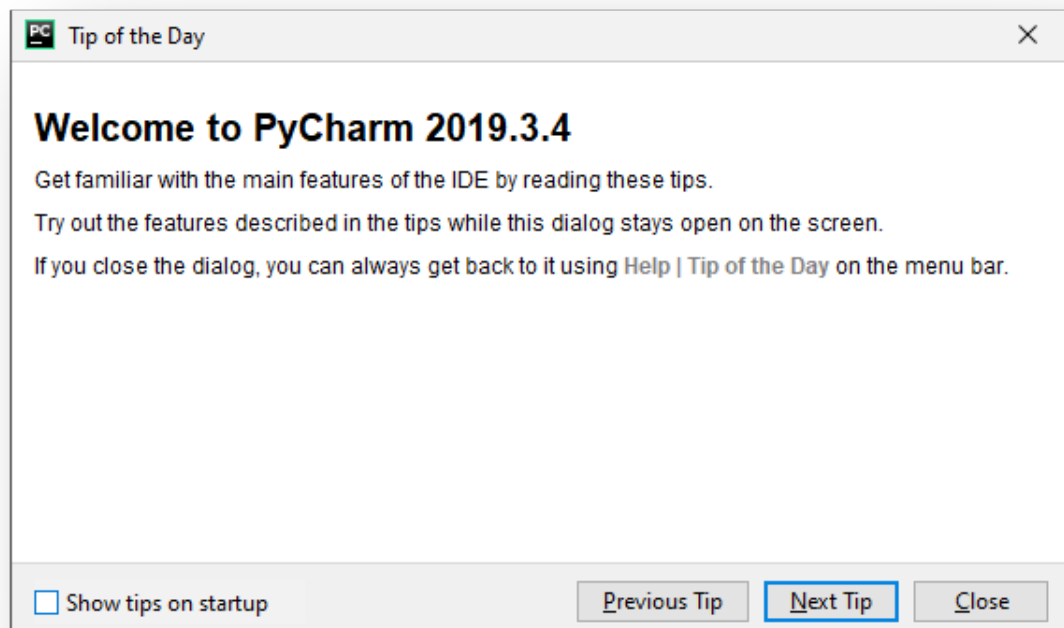
Nazovimo projekt **"moj_novi_projekt"** i kliknimo na dugme **"Create"**.

**Korak 7.**

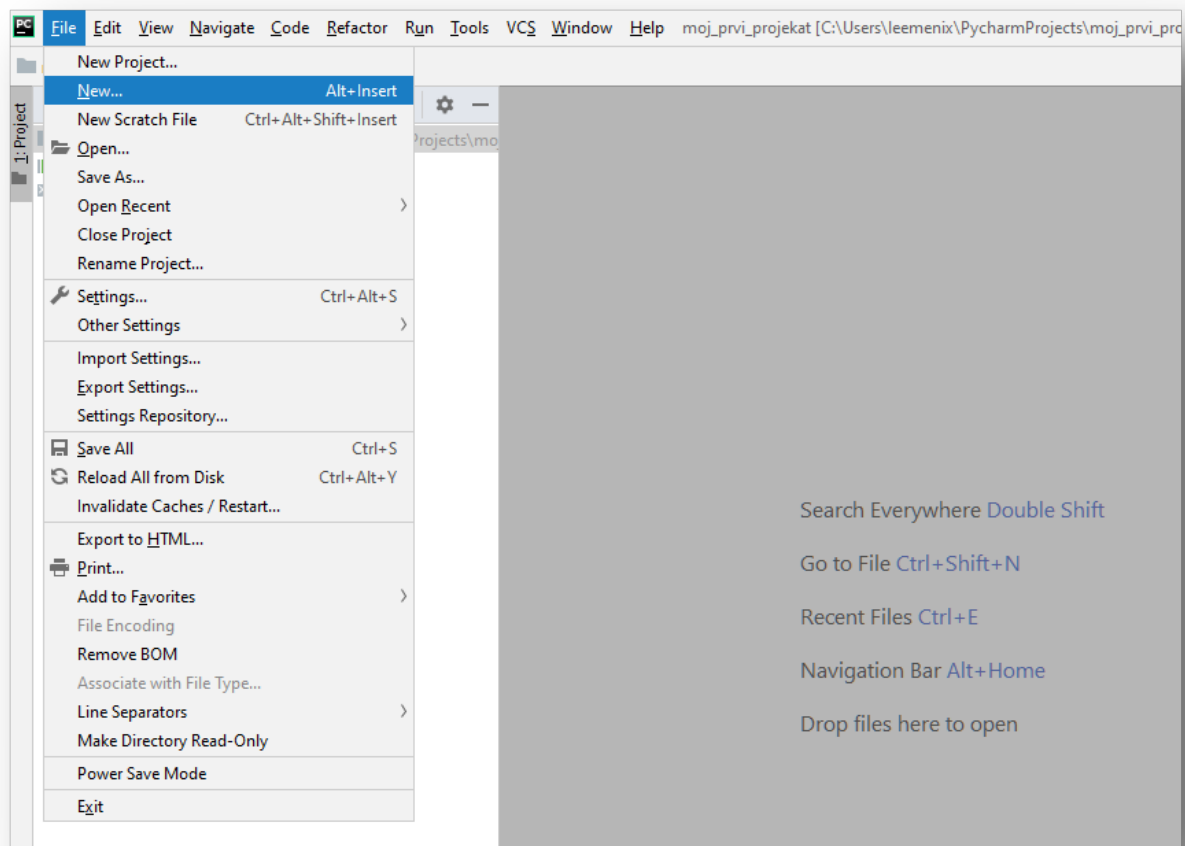
Sada je potrebno sačekati kako bi se kreiralo virtuelno okruzenje za nas novi projekat.

**Korak 8.**

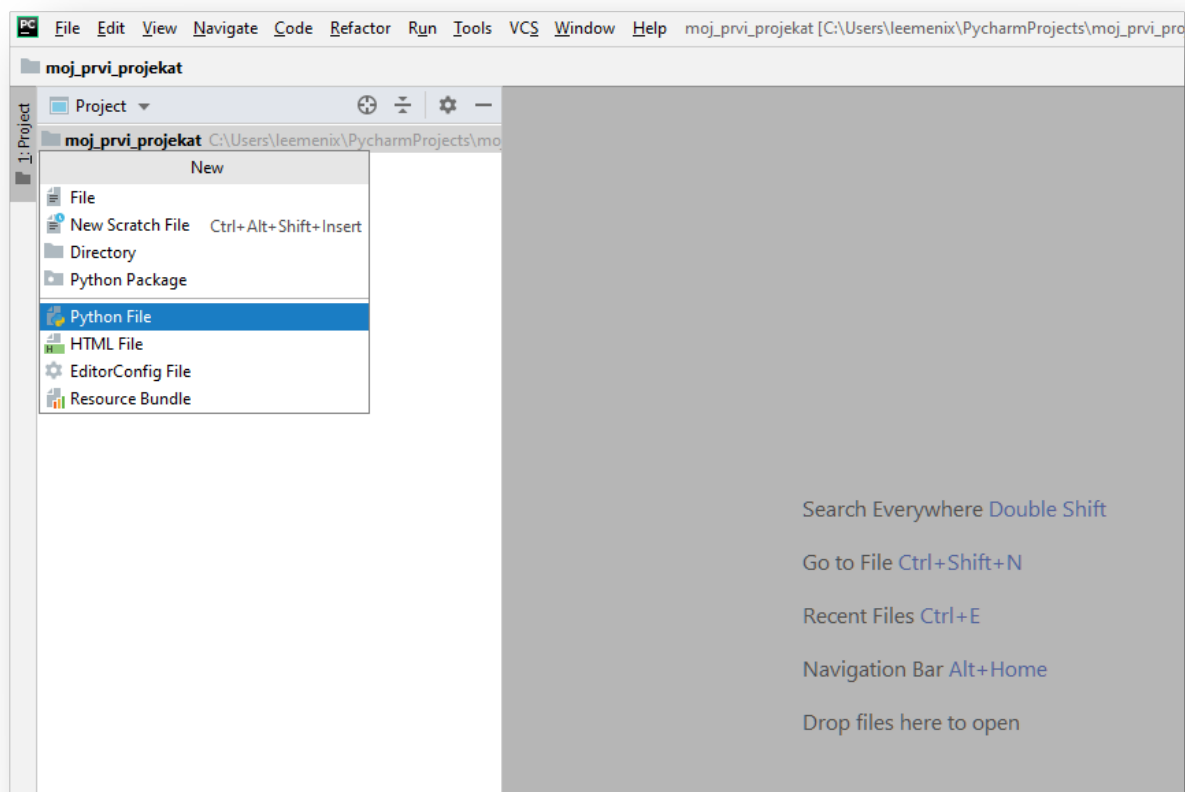
Nakon što se okruženje kreiralo, opciono možemo onemogućiti opciju **"Show tips on startup"** i kliknuti na dugme **"Close"**

**Korak 9.**

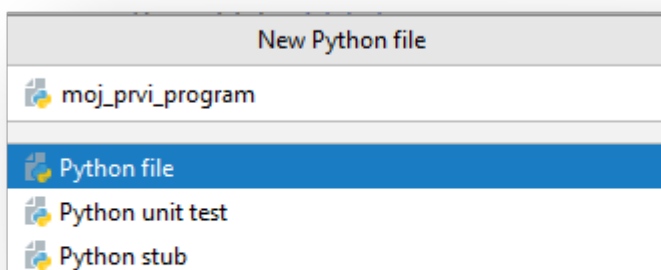
Iz glavnog menija izaberite opciju **"File"**

**Korak 10.**

zatim odaberite **"Python file"**

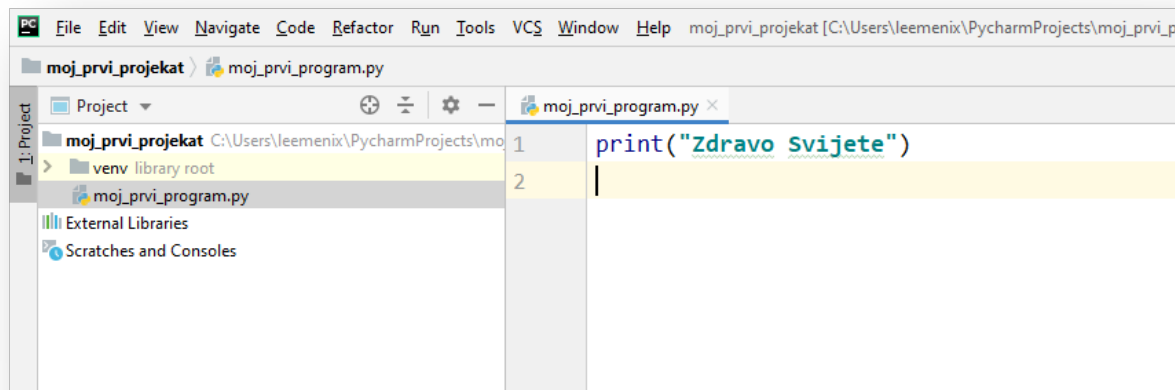
**Korak 11.**

a nakon toga dajte ime fajlu **"moj_prvi_program"** i kliknite **"Enter"**. Ovime smo kreirali novi fajl `moj_prvi_program.py` (ekstenziju `.py` ce dodijeliti sam PyCharm, na nama je samo da damo ime programu).

**Korak 12.**

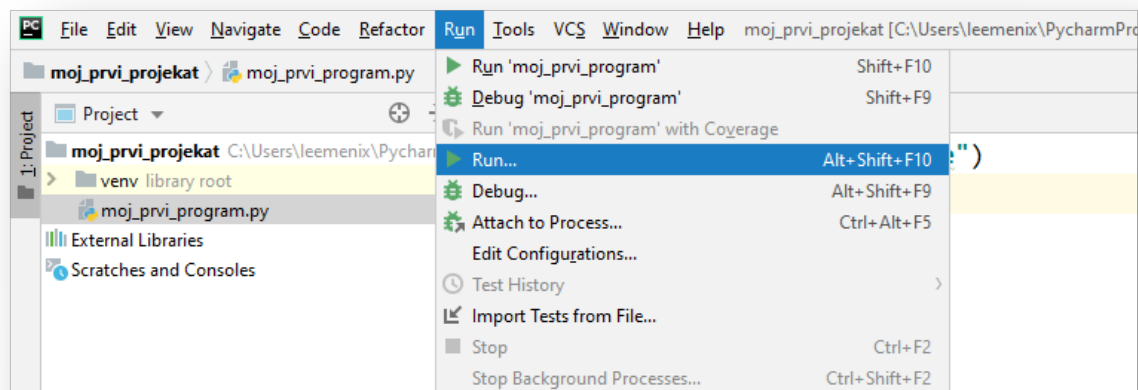
Kako bi smo testirali da li PyCharm u potpunosti radi sa nasim okruzenjem, napisacemo najosnovniji program u Python-u

```
print("Zdravo Svijete")
```



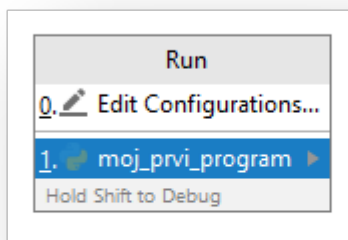
Korak 13.

a zatim u narednom koraku iz glavnog menija, izaberemo opciju **"Run"** pa u padajucem meniju opciju **"Run ..."**



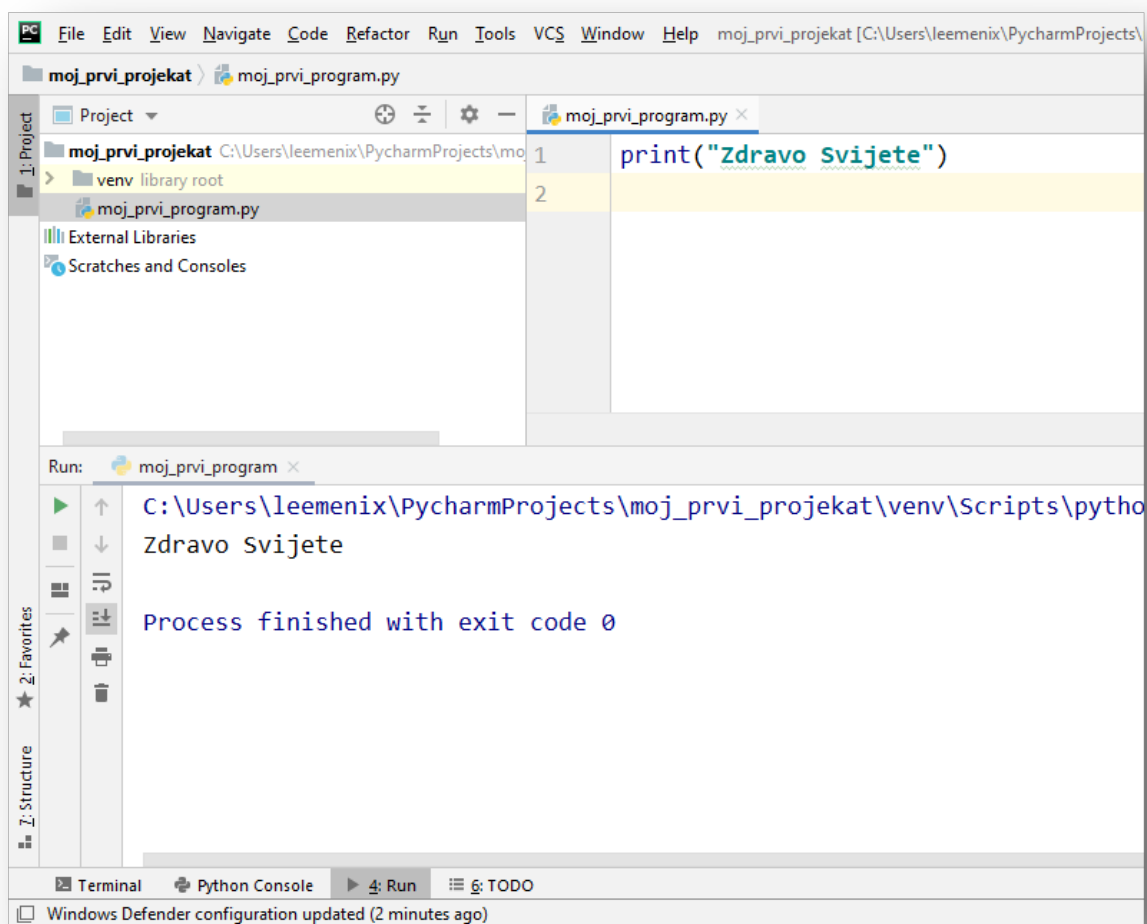
Korak 14.

PyCharm ce nam ponuditi ime programa koji zelimo da izvorsimo, na nama je samo da kliknemo **"Enter"**



Korak 15.

Nakon uspješnog pokretanja u donjem dijelu PyCharm-a vidjecete ispis teksta **"Zdravo Svijete"**. Cestitamo uspješno ste podesili vase radno okruzenje, igra moze da pocne 😊



Instalacija Mac OS

Otvorite vas terminal i jednostavno uz pomoc Homebrew-a:

```
brew install python3
```

Napomena: U vrijeme pisanja prirucnika, instruktor nije imao pristup kompjuteru sa Mac OS sistemom, tako da sva detaljnija podrška za Mac OS korisnike ce izostati do daljnjeg.

Instalacija GNU/Linux

Za instalaciju Python-a na GNU/Linux, korisnici mogu koristiti ugradjenje paket menadzere, u zavisnosti od distribucije koje koriste.

Za **Debian** familiju instalaciju mozete izvorsiti **apt** paket menadzerom:

```
sudo apt update && sudo apt install python3
```

Za **RedHat** familiju, instalaciju mozete izvorsiti pomocu **yum/dnf** paket menadzera:

```
yum update -y && yum install -y python3
```

ili

```
dnf update -y && dnf install -y python3
```

Naravno, za one malo iskusnije tu je mogucnost instalacije pomocu kompajliranja iz izvornog koda:

```
./configure  
make && make install
```

tbd.{

- Sublime installation, notpad ++
- mi cemo koristiti PyCharm - IDE (Integrated Development Environment)
- python 2 (legacy), python 3 (future)
 - razlika u sintaksi }

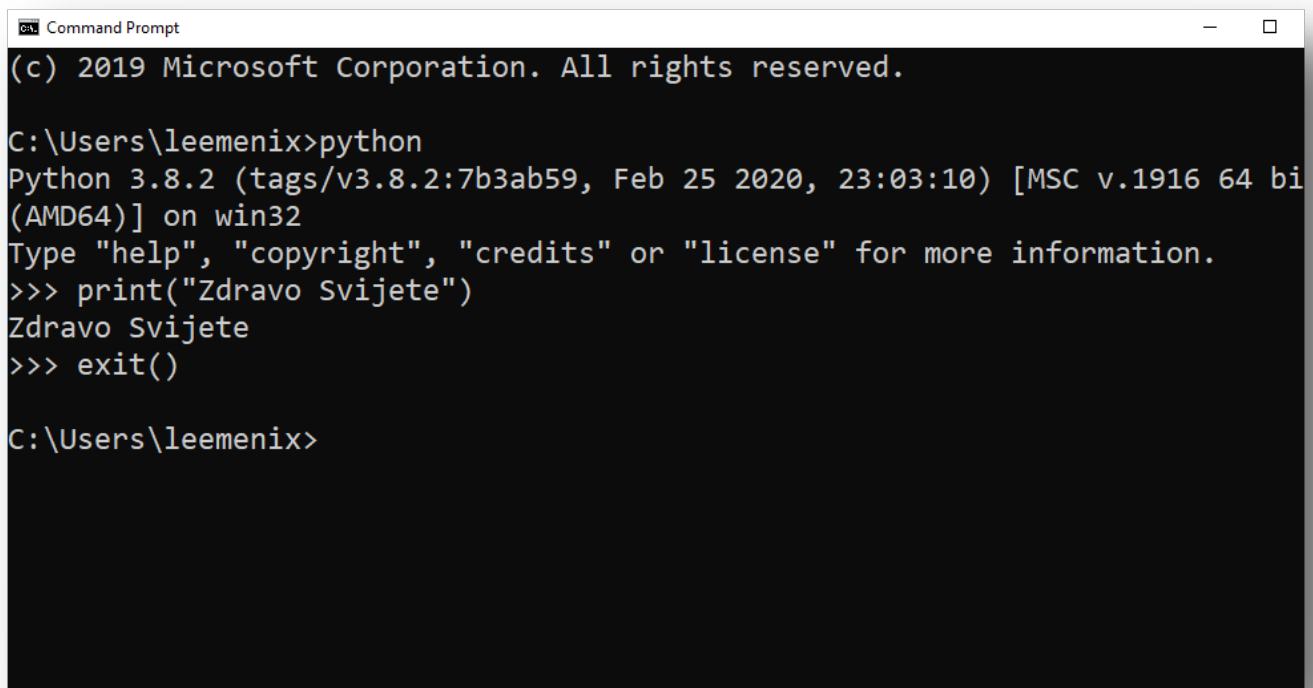
Zdravo Svijete

Izvorni kod: kod-10_zdravo-svijete.py

```
print("Zdravo Svijete!")
```

Imamo dva nacina za pokretanje Python programa:

1. **Python interaktivni prompt** U slucaju da nesto zelimo brzo provjeriti, recimo ispis jedne linije koda, sabiranje dva broja, pridruzivanje vrijednosti promjenljivoj koristimo Python interaktivni prompt. Iz komandnog prompta (command prompt) ili cmd (na Windows operativnom sistemu) ili terminal (na Linux, FreeBSD ili MacOS operativnim sistemima) pokrenimo Python interaktivni prompt komandom **python**:

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The text inside shows the command prompt at "C:\Users\leemenix>" where the command "python" has been entered. The output shows the Python 3.8.2 version information and the execution of the script "Zdravo Svijete.py", which prints "Zdravo Svijete". The prompt then returns to "C:\Users\leemenix>".

```
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\leemenix>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bi
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Zdravo Svijete")
Zdravo Svijete
>>> exit()

C:\Users\leemenix>
```

2. **Pokretanje izvornog koda** U slucaju da nas program zahtijeva vise linija koda, kreiranje odredjenje logicke strukture, izvorsavanje uslova i petlji, kreiranje funkcija ili klasa potrebno je prvo sav kod zapisati kao datoteku, sto u programiranju zovemo **Izvorni kod** i odatle ga izvršiti pomocu python komande ili direktno iz IDE-a. **python ime_datoteke.py**

```
Command Prompt

C:\Users\leemenix>cd PycharmProjects

C:\Users\leemenix\PycharmProjects>cd moj_prvi_projekat

C:\Users\leemenix\PycharmProjects\moj_prvi_projekat>dir
Volume in drive C has no label.
Volume Serial Number is 7273-184C

Directory of C:\Users\leemenix\PycharmProjects\moj_prvi_projekat

04/01/2020  04:15 AM    <DIR>          .
04/01/2020  04:15 AM    <DIR>          ..
04/01/2020  04:17 AM    <DIR>          .idea
04/01/2020  04:15 AM                25 moj_prvi_program.py
04/01/2020  04:10 AM    <DIR>          venv
                1 File(s)                25 bytes
                4 Dir(s)  40,832,086,016 bytes free

C:\Users\leemenix\PycharmProjects\moj_prvi_projekat>python moj_prvi_program.py
Zdravo Svijete

C:\Users\leemenix\PycharmProjects\moj_prvi_projekat>
```

Programiranje u najjednostavnijoj svojoj reprezentaciji predstavlja davanje instrukcija kompjuteru (kroz programski jezik) i na osnovu ovih instrukcija kompjuter donosi odluke. Učenje novog programski jezik se jednostavno može uporediti i sa učenjem bilo kog drugog stranog jezika, stim da je dosta jednostavnije, jer kad jednom shvatimo logiku programskog jezika ostalo je samo nadograđivanje i nasa masta.

Izvorni kod: kod-11_crtanje-oblika.py

```
print("")
print("***")
print("****")
print("*****")
print("*****")
print("*****")
```

U prethodnom primjeru Python ide liniju po liniju i izvršava kod. Sta se desava u slučaju da zamijenimo prvu i zadnju liniju?

Komentarisanje koda

Komentare koristimo kada želimo da zapisemo neki podsjetnik unutar koda, komentarisemo kod, objasnimo drugima i sebi sta određena linija koda radi. Praksa i preporuka je da se koristi simbol taraba (hash tag) **#**. Komentari se po definiciji ignorisu u Python-u, preciznije ignorisani od strane Python

interpretera, pa tako kad Python prilikom citanja koda naidje na znak **#**, ignorise sve u toj liniji sto se nalazi iza znaka **#**, odnosno na njegovoj desnoj strani.

Izvorni kod: kod-12_demonstracija-komentara.py

```
'''  
Viselinijski komentar  
'''  
''''  
Viselinijski komentar  
''''  
print("Komentari su korisni")  
# print("Ova linija koda nece biti ispisana")
```

Par pravila o pisanju koda u Python-u

Indentacija

Kako smo već naveli Python je fleksibilan ali i osjetljiv na način kako pisemo kod. Dakle prilikom pisanja izvornog koda potrebno je voditi računa o uvlacenju linija i razmacima na početku linije, što se naziva indentacija. Razmaci i tabovi koje koristimo na početku linije prilikom pisanja koda, određuju nivo indentacije (uvlacenja), što je bitno postovati tokom ostatka pisanja koda unutar datoteke, jer Python na ovaj način određuje blokove koda i na taj način ih grupise u logičke cjeline.

Dakle vodite računa da komande koje pisete, a treba da idu zajedno, imaju isti nivo uvlacenja, u suprotnom doći će do greške prilikom izvršavanja.

Naravno citav ovaj koncept ćete shvatiti do kraj ove sekcije ali svakako nije loše napomenuti ovo pravilo.

```
print("Zdravo Svijete!") # ispravno

print("Zdravo Svijete!") # greška! razmak na početku reda
```

Prilikom izvršenja prethodnog koda Python će se zaliti na indentaciju i dobićete sintaksnu grešku. Bez panike, samo obrišite razmak na početku reda i sve ok.

Pravilo je da za uvlacenje koda koristite samo razmake, a u slučaju da koristite tabove budite sigurni da je veličina taba podesena na 4 razmaka. Ovo je automatski regulisano u skoro svakom savremenom IDE-u.

Eksplisitno spajanje linija

U slučaju da prilikom pisanja koda želimo veoma dugacku liniju napisati u više redova i pridržite je promjenljivoj, koristimo obrnutu kosu crtu (backslash) \.

```
tekst_pjesme = "9 depresivaca \
9 depresivaca gajili su bostan \
puko lastik od bandzija, ostalo ih 8 \
8 depresivaca, k'o u dlan ih gledam \
u krivini hladnjaca, ostalo ih 7 \
... \
3 depresivca, svaki od njih vrijedan \
dvojici crk'o facebook, ostao je 1 \
1 depresivac, oprezan je bio\
onda se ozenio"
```

Zadaci za samostalni rad!

1. Napisati program koji ispisuje vase ime i prezime

Ime i Prezime

2. Napisati program koji crta pravougaonik oblika



3. Napisati program koji crta trougao oblika



Promjenjive i tipovi podataka

Promjenjive ili varijable su osnovni objekti podataka kojima se manipulise u programu. Recimo da zelimo imati promjenjivu **ime_korisnika** koju mozemo koristiti kroz citav program i koja sadrzi vrijednost **Goku**. To bi mogli napisati kao:

```
ime_karaktera = "Goku"
```

Ovo citamo: Deklarisana je promjenjiva, ciji je naziv (identifikator) **ime_karaktera** a cija je inicijalna (pocetna) vrijednost **Goku**.

Takodje iz navedenog primjera mozemo zakljuciti da je promjenjiva, ciji je naziv **ime_karaktera**, tekstualnog tipa, niz karaktera ["G","o","k","u"], ili na engleskom, tipa **string**.

Promjenjive, varijable (variables)

Programiranje se uglavnom svodi na obradu podataka, stim u vezi je potrebno pohraniti podatke i organizovati ih na najbolji moguci nacin. Varijabla ili promjenjiva predstavlja lokaciju u memoriji vaseg kompjutera i sluzi da pokaze na odredjenu vrijednost koju ta memorijska lokacija predstavlja.

Tri glavna faktora koji cine promjenjivu/varijablu jesu:

1. naziv varijable,
2. operator i
3. vrijednost varijable

naziv	operator	vrijednost
	pridruzivanja	
ime	=	"Goku"
godine	=	23

Gradjenje varijable

Prilikom kreiranja varijable ili promjenjive postoji par pravila kojih se trebamo pridrzavati:

1. Naziv varijable ne smije poceti sa brojem
2. Naziv varijable moze poceti, malim, velikim slovima ili donjom crticom (_) nakon cega moze ici broj
3. Mala i velika slova se razlikuju (a != A)

Izvorni kod: kod-155_naziv-varijabli.py

```
ime = "Goku"  
godine = 16
```



```

_nove_godine = 18
25_godine = 25 # ovde cemo dobiti gresku
a = 3
A = 4
print (ime)
print (godine)
print (_nove_godine)
print (25_godine) # ovde cemo dobiti gresku
print (a)
print (A)

```

Izvorni kod: kod-156_prakticna-primjena-varijabli.py

```

karakter_1 = "Son Goku"
karakter_2 = "Krillin"
godine = "17"

print("U dalekoj prošlosti živio je dječak po imenu Goku.")
print("Goku je imao 15 godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu Krillin!")
print (" ")
print("U dalekoj prošlosti živio je dječak po imenu " + karakter_1)
print(karakter_1 + " je imao " + godine + " godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu " + karakter_2)
# Veoma korisna funkcija type() služi nam da odredimo kog tipa je neka
promjenjiva tokom razvoja programa.
print(type(karakter_ime))
#godine = 17
print(type(godine))

```

Tipovi podataka

Imamo tri osnovna tipa podatka u Python-u, tekstualne, brojne i logičke.

tipovi podataka	python sintaksa	objasnjenje
Tekstualni (string - niz znakova)	string()	operacije nad znakovnim tipovima podataka
Brojevi cijeli, realni (integer, float)	int()	pretvara u cijeli broj (npr. 1,10,33)
	float()	pretvara u realni broj (npr. 1.0, 3.14, 33.333)

Logicki tacno, netacno (boolean True/False) (Bulovo da/ne)	bool()	operacije nad logickim tipovima podataka (True i False)
---	--------	---

Mijenjanje tipova promjenjive (kastovanje)

Promjenjiva moze mijenjati tip kroz izvorsavanje programa, sto se jos naziva i kastovanje (casting).

Izvorni kod: kod-157_tipovi-podataka.py

```

karakter_ime = "Goku"
karakter_godine = 15
karakter_visina = 168.5
karakter_osobina_dobar = True
karakter_osobina_los = False

print(karakter_ime + " ima " + karakter_godine)
print(karakter_ime + " ce za godinu dana imati " + karakter_godine + 1)
print(karakter_ime + " je visok " + karakter_visina + "cm.")
print(karakter_ime + " ce za godinu dana biti visok " + (karakter_visina +
5) + " cm.")
print(karakter_ime + " je dobar karakter" + karakter_osobina_dobar)
print(karakter_ime + " je los karakter" + karakter_osobina_los)

karakter_ime = "Goku"
karakter_godine = 15
karakter_visina = 164.5
karakter_osobina_dobar = True
karakter_osobina_los = False

print(karakter_ime + " ima " + str(karakter_godine))
print(karakter_ime + " ce za godinu dana imati " + str(karakter_godine +
1))
print(karakter_ime + " je visok " + str(karakter_visina) + " cm.")
print(karakter_ime + " ce za godinu dana biti visok " + str(karakter_visina
+ 5) + " cm.")
print(karakter_ime + " je dobar karakter " + str(karakter_osobina_dobar))
print(karakter_ime + " je los karakter " + str(karakter_osobina_los))

```

Rad sa stringovima

Stringovi predstavljaju niz karaktera (velika/mala slova, brojevi, znakovi interpunkcije, specijalni znakovi, ...), koji služi za skladištenje i rad sa tekstualni podacima. Mozemo reci da su stringovi uredjeni i smisleni niz karaktera.

Definisanje stringova, ili kako prepoznati odredjeni string u kodu, je lako.

String mozemo prepoznati ako se nalazi izmedju navodnika:

```
"Ja sam string"
```

ali i izmedju apostrofa:

```
'Ja sam takodje string'
```

Jednostavan primjer kreiranja stringa

Izvorni kod: [kod-165_rad_sa_stringovima.py](#)

```
print("programiranje.ba besplatni online kursevi")

# escape karakter \
print("programiranje.ba \n besplatni online kursevi")

sajt_naziv = "programiranje.ba"
sajt_slogan = " besplatni online kursevi"

# jos neki primjeri kreiranja string-a (kao sto smo imali slucaj sa
viselinijskim komentarom)
sajt_naziv = 'https://programiranje.ba'
sajt_slogan = """besplatni online kursevi
                za sve """
```

Funkcije nad stringovima

Izvorni kod: [kod-166_rad_sa_stringovima.py](#)

```
print("programiranje.ba besplatni online kursevi")

# escape karakter \
print("programiranje.ba \n besplatni online kursevi")
# takodje se koristi kada trebamo ispisati specijalne karaktere
#print("\n")
print("\\n")
```

```
print("a" + "b")
print('a' + 'b')

sajt_naziv = "programiranje.ba"
sajt_slogan = " besplatni online kursevi"
print(sajt_naziv)
print(sajt_slogan)
print(sajt_naziv + sajt_slogan)
print(sajt_naziv.upper() + sajt_slogan.upper())
print(sajt_naziv.isupper())
print(len(sajt_naziv))
print(sajt_naziv * 3)

# index stringa pocinje na poziciji 0
print(sajt_naziv[4])
print(sajt_naziv[-4])
print(sajt_naziv[1:4])
# index funkcija i prosljedjivanje parametara
print(sajt_naziv.index('g'))
print(sajt_naziv.index('mira'))
#print(sajt_naziv.index('h'))
# find funkcija i prosljedjivanje parametara
print(sajt_naziv.find('g'))
print(sajt_naziv.find('mira'))
# razlika izmedju find i index, u slucaju da ne postoji trazeni patern
# index() vraca gresku, dok find vraca -1

# replace funkcija
print(sajt_slogan.replace("kursevi", "tutoriali").upper())

print("{1}, {0}".format(sajt_naziv, sajt_slogan)) # metoda formatiranja
print(f"{sajt_naziv} {sajt_naziv}") # metoda formatiranja 2

# jos neki primjeri funkcija nad stringovima
sajt_naziv = "programiranje.ba"
sajt_godina = "2020"
sajt_kratki_slogan = "kursevi"

print(sajt_naziv.isalpha())
print(sajt_godina.isdigit())
print(sajt_kratki_slogan.isalpha())
### funkcije chr() i ord() kasnije potrebni za cezarovu sifru
# funkcija ord() daje cjelobrojnu vrijednost karaktera prema ASCII tabli
print(ord('A'))
print(ord('a'))
print(ord('b'))
print(ord('z'))

# funkcija chr() konvertuje cjelobrojnu vrijednost u odgovarajuci karakter
print(chr(64))
print(chr(33))
print(chr(97))
print(chr(100))
```

```
# kombinacija chr() i ord()
print(ord('b') + 3)
print(chr(ord('b') + 3))
```

Rad sa brojevima

Imamo dvije osnovne vrste brojeva u Python-u:

1. Cijeli brojevi (integer)
2. Decimalni ili brojevi sa pokretnim zarezom (float)

Osnovne aritmetičke operacije koje možemo vršiti u Python-u su

Operacija	Naziv operacije	Primjer	Rezultat
+	Sabiranje	<code>print(1 + 1)</code>	2
-	Oduzimanje	<code>print(6 - 5)</code>	1
*	Množenje	<code>print(4 * 3)</code>	12
/	Dijeljenje	<code>print(4 / 2)</code>	2
%	Ostatak pri dijeljenju (modulo)	<code>print(5 % 2)</code>	1
**	Potenciranje	<code>print(2 ** 3)</code>	8

Izvorni kod: `kod-185_rad-sa-brojevima.py`

```
print(3)
print(-4)
print(3.333)
print(7+3)
print(7+3.333)
print(8/4)
print(2*4)
print (2*(4+5))
# modulo , ostatak 9 mod 4
print(9%4)

moj_broj = 13
print(moj_broj)
print("Moj omiljeni broj" + str(moj_broj))
```

Funkcije nad brojevima

Izvorni kod: `kod-186_rad-sa-brojevima.py`

```
# math funkcije, funkcije su vec pripremljen kod koji odradjuje posao za
nas
broj = -2
print(abs(broj))
print(pow(4,2))
print(max(5,10))
print(min(3,6))
print(round(3.3333))
print(round(3.6))
print(round(3.5))

# import math modul (import math funkcije vise kada budemo pricali o
modulima)
from math import *
print(floor (3.6))
print(ceil(3.6))
print(sqrt(9))
```

Ulaz/upis podataka, prihvatanje podataka od korisnika ili interakcija sa programom

Slozicete se da je programiranje dosadno, ako nemamo interakciju, nekakav vid komunikacije sa nasim programom. Kako bi omogucili interakciju sa programom, Python nam na raspolaganje nudi funkciju `input()`.

Izvorni kod: kod-190_interakcija-sa-korisnikom.py

```
input()
# hej ti, cekam da uneses neku informaciju podatak, naravno korisnik nije
# siguran sta se desava

input("Unesite vase ime: ")
# aha ovo sad vec ima smisla

# <naziv varijable> <tip podatka>
korisnik_ime = input("Unesite vase ime: ")
# naravno, posto nam je korisnicki unos vazan mi zelimo sacuvati isti taj
# unos u neku varijablu
# kako bi smo kasnije mogli koristiti
print("Zdravo, " + korisnik_ime + " dobrodosli.")

korisnik_ime = input("Unesite vase ime: ")
korisnik_godine = input("Unesite vase godine: ")
print("Zdravo, " + korisnik_ime + ". Vi imate " + korisnik_godine + "
godina.")

# vjezba ispisati preghodni program koristeći funkciju format
```

Izvorni kod: kod-191_interaktivni_karakter_program.py

```
print("U dalekoj proslosti zivio je djecak po imenu Goku.")
print("Goku je imao 15 godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu Krilin!")

karakter_ime = input("Unesite ime karaktera: ")
karakter_godine = input("Unesite godine karaktera: ")
karakter_prijatelj = input("Unesite ime najboljeg prijatelja: ")

print(f"U dalekoj proslosti zivio je djecak po imenu {karakter_ime}.")
print(f"{karakter_ime} je imao {karakter_godine} godina.")
print(f"Volio je upoznavati nove karaktere ")
print(f"i imao je najboljeg druga po imenu {karakter_prijatelj}!")
```

Izvorni kod: kod-192_osnovni_kalkulator.py


```
broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = broj1 + broj2

print(rezultat)

broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = float(broj_1) + float(broj_2)

print(rezultat)
```

Zadaci za samostalni rad!

1. Napisati program koji traži da upisete vase ime, a on automatski ispisuje vase ime pet puta, sa razmakom izmedju imena.

```
Unesite vase ime: Goku
Goku Goku Goku Goku Goku
```

2. Napisati program koji traži da unesete dvije rijeci, a zatim ih ispise u istom redu sa razmakom od tri space karaktera izmedju izmedju.

```
Unesite prvu rijec: Dobar
Unesite drugu rijec: Dan
Dobar   Dan
```

3. Napisati program koji racuna aritmeticku sredinu za tri unesena broja. Pomoc: Aritmeticka sredina za a,b,c se izracunava po formuli $(a + b + c)/3$

```
Unesite prvi broj: 1
Unesite drugi broj: 2
Unesite treci broj: 3
Aritmeticka sredina je: 2.0
```

4. Napisati program koji za uneseni karakter sa tastaure ispisuje vrijednost iz ASCII tabele

```
Unesite znak sa tastature: }
Vrijednost znaka '}' u ASCII tabeli je 125
```

5. Napisati program koji ispisuje.

```
Omiljena pjesma od 'Miladina Sobica' - "Od druga, do druga"!
```

Liste - []

Liste predstavljaju niz objekata, gdje svaki član liste ima svoj indeks. Ovi članovi se nazivaju elementima lista. Slične su stringovima, s tim da svaki element liste može biti različitog tipa. Elementi liste su smješteni u uglaste zagrade [] i razdvojeni zarezom ,.

```
lista    |  [ 1, 4, 6.33, 10, "Goku" ]  
indeks   |      0  1      2    3    4
```

Rad sa listama, nam pomaže pri organizaciji i boljem praćenju toka podataka.

Izvorni kod: kod-210_rad-sa-listama.py

```
prazna_lista = []  
print(prazna_lista)  
print(type(prazna_lista))  
  
karakter_i = ["Goku", "Krillin", "Bulma", "Chi-Chi", "Yamcha", "Picollo"]  
print(karakter_i)  
  
##### mozete smjestiti stringove , brojeve, boolean u liste  
##### referenciranje po indexu, ako zelimo pristupiti elementu unutar liste  
print(karakter_i[0])  
print(karakter_i[4])  
  
print(karakter_i[-1])  
print(karakter_i[-2])  
  
print(karakter_i[1:])  
print(karakter_i[2:4])  
print(karakter_i[2:-2])  
  
##### izmjena elemenata u listi  
karakter_i[4] = "Master Roshi"  
print(karakter_i[4])  
print(karakter_i)
```

Funkcije nad listama

Izvorni kod: kod-211_rad-sa-listama.py

```
karakter_i = ["Goku", "Krillin", "Bulma", "Chi-Chi", "Yamcha", "Picollo"]  
print(karakter_i)  
loto_brojevi = [4, 7, 3, 10, 32]  
print(loto_brojevi)  
  
karakter_i.extend(loto_brojevi)
```

```
print(karakter_i)

karakter_i.append("Majin Buu")

print(karakter_i)

karakter_i.insert(2, "Vegeta")
print(karakter_i)

karakter_i.remove("Majin Buu")
print(karakter_i)

del(karakter_i[3]) # funkcija del
print(karakter_i)

izbrisan_karakter = karakter_i.pop(4) # metoda pop, u slucaju da hocemo da
sacuvamo izbrisani element
print(karakter_i)

karakter_i.clear("")
print(karakter_i)

##### provjeri da li je odredjeni element u listi

karakter_i = ["Goku", "Krillin", "Bulma", "Chi-Chi", "Yamcha", "Piccolo",
"Bulma"]
print(karakter_i)

print(len(karakter_i)) # funkcija len

print(karakter_i.index("Bulma")) # metoda index

print(karakter_i.count("Bulma")) # metoda count, prebrojavanje koliko se
trazeni element pojavljuje u listi

karakter_i.sort() # metoda sort
print(karakter_i)

karakter_i.reverse() # metoda reverse
print(karakter_i)

karakter_i_novi = karakter_i.copy() # metoda copy

print(karakter_i_novi)

karakter_i_novi.sort()
print(karakter_i_novi)

print list(string_karakter)
# funkcije min i max, vracaju najmanji ili najveći element liste
respektivno
karakter_i = ["Goku", "Krillin", "Bulma", "Chi-Chi", "Yamcha", "Piccolo",
"Bulma"]
```

```
print(karakter_i)
print min(karakter_i) # funkcija min
print max(karakter_i) # funkcija max

# konverzija stringa u listu
string_karakter = "Goku"
type(string_karakter)
print(string_karakter)
```

Tuples - torke (tip podaktivne strukture, veoma slican listama) - ()

Tuples predstavlja niz nepromjenjivih clanova. Clanovi unutar tuple-a mogu biti istih ili razlicitih tipova. Tuple definisemo nabranjanjem objekata odvojenih zarezom, cak i ako je u pitanju jedan jedini clan moramo imati zaraz, u suprotnom se gubi osobina tuple-a.

Razlikuju se od liste po tome sto su **nepromjenjive (immutable -ne mogu se mijenjati)**.

Tuple mozemo prevosti kao torke, a izraz dolazi iz matematike od pojma **n-torka** (eng.tuple) koji predstavlja konacni niz (poznat kao uredjena lista) od n objekata, od kojih je svaki specifincnog tipa.

Clanovi torke su smjesteni u obicne zagrade **()** i razdvojeni zarezom **,**. Clanovi torke mogu biti i same torke.

Izvorni kod: kod-220_rad-sa-torkama.py

```
karakteri = (1,2,3,4,"a","d","-")
print(karakter_i)
type(karakter_i)

karakteri = (1,)
print(karakter_i)
type(karakter_i)

karakteri = (1)
print(karakter_i)
type(karakter_i)

### list of tuples
koordinata = [(4,5), (6,3), (7.4)]
print(type(koordinata))
print(len(koordinata))

## tuple su nepromjenjive strukture
koordinata = (4, 5)
print(type(koordinata)) # funkcija type()
print(len(koordinata)) # funkcija len()
print(koordinata.index(5)) # metod index
print(koordinata[0])
print(koordinata[1])

koordinata[1] = 10 # dobicemo gresku

### pristupanje elementima torki slicno je kao i kad pristupamo listama
karakteri = (1,2,3,4,"a","d","-") # pristupanje elementima tuple-a
print(karakter_i[1])
print(karakter_i[2:4])
print(karakter_i[-2:])
print(karakter_i[::2]) # pristupanje svakom drugom elementu

# konverzija drugih tipova u tuple
karakter_ime = "Goku"
```

```
type(karakter_ime)
karakter_godine = 15
type(karakter_godine)

karakter_ime = tuple(karakter_ime)
type(karakter_ime)
print(karakter_ime)
karakter_godine = tuple(karakter_godine)
karakter_godine = tuple(str(karakter_godine))
type(karakter_godine)
print(karakter_godine)

karakter = karakter_ime + karakter_godine
type(karakter)
print(karakter)

# brisanje tuple-a
del(karakter[3]) # brisanje elemenata tuple-a nije moguće
del(karakter)
```

Rijecnici - Dictionaries - { }

Rijecnici su tipovi podataka, opet slicni listama, ali za razliku od listi indeksiranje se obavlja kljucevima.

Za lakse razumijevanje ih mozemo uporediti sa klasicnim rijecnikom za prevodjenje rijeci sa jednog jezika na drugi, gdje imamo strukturu strana rijec na lijevoj strani i detaljno objasnjenje rijeci na desnoj strani. Ako navedenu analogiju primijenimo rijecnicima, kao tipovima podataka u Python-u, onda rijec predstavlja kljuc (key), dok detaljno objasnjenje predstavlja vrijednost (value).

Elementi rijecnika su smjesteni u viticaste zagrade { } a parovi elemenata su razdvojeni zarezom ,.

```
{kljuc:vrijednost} ({key:value})
```

Bitno je napomeni da kljuc (key), mora biti jedinstven, sto znaci da ne mozemo imati dva ista kljuca.

```
"kljuc_1:vrijednost_1", "kljuc_2:vrijednost_2", "kljuc_3:vrijednost_1" -  
ispravno  
"kljuc_1:vrijednost_1", "kljuc_1:vrijednost_2", "kljuc_3:vrijednost_1" -  
nije ispravno
```

Izvorni kod: kod-230_rad-sa-rijecnicima

```
karakteri={} # prazan rijecnik  
print(karakter_i)  
  
karakteri_osobine={"Goku":"Vegeta", "Picolo":"Namek", "Krillin":"Zemlja"}  
print(karakter_i_planete)  
print(type(karakter_i_planete))  
  
karakteri_planete["Bulma"]="Zemlja"  
print(karakter_i_planete)  
karakteri_planete["Goku"]="Namek" # prepisace trenutnu vrijednost ako  
postoji  
  
del(karakter_i_planete["Goku"]) # brisanje elementa  
  
print(len(karakter_i_planete)) # primijetimo da se broje parovi  
  
print(karakter_i_planete.keys()) # metoda keys() nad rijecnicima, ispisuje  
sve kljuceve (keys), nema argumente  
print(karakter_i_planete.values()) # metoda values() ispisuje vrijednosti  
elementa, nema argumente  
print(karakter_i_planete.items()) # metoda items() ispisuje kljuc:  
vrijednost elementa, nema argumente  
  
# metode get i.setdefault  
karakteri_planete={'Goku':'Vegeta', 'Krillin':'Zemlja', 'Bulma':''}
```



```

print(karakter_i_planete.get('Goku','Karakter ne postoji u bazi'))
print(karakter_i_planete.get('Pikolo','Karakter ne postoji u bazi')) #
metoda get() nad rijecnikom vrši pretragu po zadatom kljucu, u slučaju da
kljuc ne postoji vraća default-nu vrijednost, vrijednost koja je
prosljeđena kao drugi parametar

print(karakter_i_planete.setdefault('Pikolo','Karakter nema definisanu
planetu')) # kljuc ce biti kreiran u slučaju da ne postoji, a vrijednost ce
biti podesena na vrijednost drugog prosljeđenog parametra
print(karakter_i_planete.setdefault('Bulma','Zemlja')) # obzirom da kljuc
postoji, neće doći do promjena

# metode pop i update
karakter_i_planete={'Goku':'Vegeta', 'Krillin':'Zemlja', 'Bulma':'Zemlja'}
obrisan_karakter=(karakter_i_planete.pop('Goku')) # pop() metoda prilikom
brisanja key:value, zadržava vrijednost (value)
print (karakter_i_planete)
print (obrisan_karakter)

# spajanje rijecnika mozemo izvesti upotrebom metode update()
karakter_i_1_planete={'Goku':'Vegeta', 'Krillin':'Zemlja', 'Bulma':'Zemlja'}
karakter_i_2_planete={'Chi-Chi':'Zemlja', 'Vegeta':'Vegeta',
'Bulma':'Namek'}
karakter_i_1_planete.update(karakter_i_2_planete) # update() metod nad
rijecnicima proširuje prvi rijecnik vrijednostima iz drugog, u slučaju da
imamo dva ista kljuc, kljuc iz prvog rijecnika bice zamijenjena kljucem iz
drugog rijecnika
karakter_i_1_planete={'Goku':'Vegeta', 'Krillin':'Zemlja', 'Bulma':'Zemlja'}
karakter_i_2_planete={'Chi-Chi':'Zemlja', 'Vegeta':'Vegeta',
'Bulma':'Namek'}
karakter_i_2_planete.update(karakter_i_1_planete)

```

Izvorni kod: kod-231_konverzija_mjeseci.py

```

# recimo da zelimo konvertovati kratke nazive mjeseca u standardne
# Jan -> Januar
# Mar -> Mart

konverzijaMjeseci = {
    "Jan": "Januar",
    "Feb": "Februar",
    "Mar": "Mart",
    "Apr": "April",
    "Maj": "Maj",
    "Jun": "Juni",
    "Jul": "Juli",
    "Avg": "Avgust",
    "Sep": "Septembar",
    "Okt": "Oktobar",
    "Nov": "Novembar",
    "Dec": "Decembar"
}

```

```
print(konverzijaMjeseci["Jan"])
print(konverzijaMjeseci["Feb"])
print(konverzijaMjeseci["Mar"])

print(konverzijaMjeseci.get("Jan"))
print(konverzijaMjeseci.get("Dec"))
print(konverzijaMjeseci.get("Dese", "Nije validan kljuc")) # ako koristimo
get necemo dobit gresku vec empty
```

Funkcije

Skup naredbi koje po pozivu izvršavaju određene zadatke. Funkcije služe također za bolju organizaciju koda.

Funkcije se definišu pomoću ključne riječi **def**, kad god se pojavi **def** na početku linije, python zna da korisnik želi kreirati funkciju i stići u vezi se i ponasa.

Sintaksa funkcije:

```
def naziv_funkcije(parametri): # parametri su opcioni, ali ako postoje
    moraju biti definisani/proslijedjeni
    blok naredbi # argumenti (poznat jos kao tijelo funkcije)

naziv_funkcije(argumenti) # poziv funkcije
```

Iz sintakse možemo zaključiti da je `naziv_funkcije` identifikator kojim pozivamo funkciju, dok parametri služe da bi se definisale vrijednosti koje se mogu proslijediti kao parametri naredbama unutar tijela funkcije.

Izvorni kod: kod-255_rad-sa-funkcijama.py

```
def zdravo_svijete(): # funkcija bez parametara
    print("Zdravo Svijete.")

# moramo pozvati funkciju ako želimo da je izvršimo
zdravo_svijete()

def zdravo_svijete():
    print("Zdravo Svijete.")

print("Prije funkcije")
zdravo_svijete()
print("Nakon funkcije")

# prosledjivanje parametara funkciji

def zdravo_svijete(ime):
    print("Zdravo " + ime)

# moramo pozvati funkciju ako želimo da je izvršimo
zdravo_svijete("Goku")
zdravo_svijete("Krillin")

def zdravo_svijete(ime, godine):
    print("Zdravo " + ime + " vi imate " + str(godine))

# moramo pozvati funkciju ako želimo da je izvršimo
zdravo_svijete("Goku", "15")
zdravo_svijete("Krillin", "16")
```

```
# primjer funkcije sa korisnickim unosom
korisnik_ime = input("Unesite ime : ")
def pozdrav(ime):
    print ("Zdravo " + ime)

pozdrav(korisnik_ime)
```

Naredba return (return direktiva)

Kada zelimo dobiti povratnu informaciju iz funkcije koristimo naredbu **return**. Sa ovom informacijom mozemo nastaviti manipulaciju kroz daljni dio koda. Naredba return se moze pojaviti samo unutar tijela funkcije. Takodje kada zelimo da funkcije mogu medjusobno komunicirati, razmjenjivati informacije koristimo return naredbu.

Izvorni kod: kod-260_kub-broja.py

```
def kub(broj):
    broj * broj * broj

cub(3)

print(kub(3))

####
def kub(broj):
    return broj * broj * broj # aha zelim vratiti informaciju ko god da je
    pozvao funkciju

print(kub(3))

####
def kub(broj):
    return broj * broj * broj

rezultat = kub(3) # sacuvaj vrijednost koju si dobio od funkcije, ne i samu
funkciju
print(rezultat)

####
def kub(broj):
    print("stampaj prije return direktive")
    return broj * broj * broj
    print("stampaj nakon return direktive")

print(cub(3))
```

Parametri funkcije

Vrijednosti (argumenti) koje prosljedjujemo funkciji se nazivaju parametrima funkcije. Sama funkcija bez parametara je prilicno staticna, dok se uvođenjem parametara postize dinamicnost i fleksibilnost, kao i sira upotreba funkcije.

Parametre funkcije definisemo prilikom inicijalnog kreiranja funkcije, stavljamo ih u zagrade i odvajamo zarezom, u slucaju da imamo vise od jednog parametra.

Prilikom pozivanja funkcije, definisemo vrijednost koje proslijedjujemo funkciji, takodje unutar zagrada, a proslijedjenje parametre jednom rijeci nazivamo **argumentima**.

```
def brojac(broj): <- (broj) predstavlja parametar funkcije
    broj = broj + 1
    return broj

brojac(3)          <- (3) predstavlja argument, vrijednost koju
                    proslijedjujemo
                    parametru (broj)
```

Lokalne i globalne promjenjive.

Promjenjiva koja se nalazi unutar bloka funkcije, ima odredjen opseg djelovanja ili *scope* i nikako nije povezana sa promjenjivim van tijela funkcije. Ovaj tip promjenjivih koje se nalaze unutar bloka funkcije se nazivaju jos i **lokalne promjenjive**.

U slucaju da ipak zelimo definisati vrijednost promjenjive koja djeluje izvan tijela funkcije, uvodimo komandu **global** i na taj nacin nasa promjenjiva djeluje globalno, cak iako se nalazi unutar tijela funkcije.

Ako lokalna i globalna promjenjiva unutar funkcije nose isti naziv, Pyhon ce koristiti lokalnu promjenjivu, ali ovo se ne preporucuje zbog zbunjivanja i konfuzije prilikom citanja koda.

Izvorni kod: kod-265_parametri-funkcijama.py

```
# funkcija ne mijenja sadrzaj promjenjive
broj_1 = 3
broj_2 = 10
def brojac(broj_1):
    broj_2 = 30
    print("Broj 2 iz funkcije ima vrijednost: " + str(broj_2))
    broj_1 = broj_1 + 1
    return broj_1

# vrijednost koju promjenjiva pokazuje, ali ne i samu promjenjivu, sto
# obezbjedjuje da funkcija ne moze mijenjati promjenjivu, samo kopiju
# vrijednosti koja je proslijedjena
brojac(broj_1)
print(broj_1)

brojac(broj_2)
print(broj_2)

# funkcije unutar sebe mogu imati lokalne (local) i globalne (global)
# promjenjive
# lokalne promjenjive su definisane po default-u ako se ne navede drugacije
broj_3 = 3

def brojac():
```

```
global broj_3
broj_3 = broj_3 + 1
return broj_3

brojac()
print(broj)
```

Nepoznati broj argumenta (VarArgs)

U slučaju da nismo sigurni koji je tačno broj argumenata koji želimo proslijediti funkciji, tada parametru funkcije dodamo * i time postizemo n-broj elemenata koji možemo proslijediti funkciji.

Izvorni kod: kod-266_srecni-brojevi.py

```
def srecni_brojevi(*brojevi):
    print("Loto brojevi za ovu sedmicu: " + str(brojevi))

srecni_brojevi(8,13,22,12,54)

# poziv funkcije iz druge funkcije
def kub(broj):
    return broj * broj * broj

def ispis_kubnog_broja(broj):
    kubni_broj = kub(broj)
    print("Kub broja " + str(broj) + " je " + str(kubni_broj))

ispis_kubnog_broja(2)

# docstring - dokumentacijski string, predstavlja dokumentaciju same
# funkcije koja može da se pozove funkcijom
# help()
def kub(broj):
    '''Funkcije izracunava kubunesenog broja, po formuli broj * broj * broj.
        Primjer koristenja funkcije: print(kub(2))
                                   daje vrijednost 8
    '''
    return broj * broj * broj

print(kub(2))
help(kub)

# neke od standardnih, ugradjenih, funkcija koje dolaze sa Python-om
print(abs(-1))
print(len("Goku"))
print(max(2,3))
print(min(2,3))
print(str(2))
print(type(kub))
```

```
print(type(print))
print(type(type))
```

Izvorni kod: kod-267_dva-i-vise-parametara.py

```
def zbir(*brojevi_1,**brojevi_2):
    broj = 0
    for broj_1 in brojevi_1:
        broj += broj_1
        print(broj_1)
    for broj_2 in brojevi_2:
        broj += broj_2
        print(broj_2)

    return broj

print(zbir(1,2,3,30,3))
```

Podrazumijevana vrijednost argumenta

U slučaju da želimo koristiti podrazumijevanu vrijednost argumenta ili jednostavno znamo da je ta vrijednost konstantna tokom izvršavanja koda, onda unutar definisanja parametara funkcije pomocu operatora dodjele = definisemo podrazumijevanu vrijednost.

Izvorni kod: kod-268_podrazumijevana-vrijednost-argumenta.py

```
def obim_kruga(poluprecnik, pi=3.14):
    obim = 2 * (poluprecnik * pi)
    return obim

print(obim_kruga(2))
```

Argumenti definisani pomocu kljucnih rijeci

U slučaju da nismo sigurni koji je tacno broj argumenata koji zelimo proslijediti funkciji, tada parametru funkcije dodamo * i time postizemo n-broj elemenata koji mozemo proslijediti funkciji.

Izvorni kod: kod-269_argumenti-definisani-pomocu-kljucnih-rijeci.py

```
####
def pozdrav_svijete(rijec="Pozdrav Svijete \n", broj=2):
    print((rijec) * broj)

print(pozdrav_svijete(rijec="Zdravo \n", broj=10))
```


Ako zelimo definisati parametre dostupne samo preko kljucnih rijeci bez obzira na poziciju na kojoj se nalaze, to mozemo uraditi definisanjem parametara posle prvog parametra sa *.

Izvorni kod: kod-270_vrijednost-parametra-bez-obzira-na-poziciju.py

```
def zbir(*brojevi_1,broj_2):  
    broj = 0  
    for broj_1 in brojevi_1:  
        broj += broj_1  
        print(broj_1)  
    broj += broj_2  
  
    return broj  
  
print(zbir(1,2,3,30,3,broj_2=2222))
```

Naredbe za kontrolu toka (if, while, for)

Do sad smo sve sve programe pisali u jednom smijeru, mozemo ovo uporediti sa jednosmjernom ulicom, gdje automobil nikako ne smijemo voziti u suprotnom smijeru. Sta ako bismo u nekom trenutku, pisanja koda, odlucili da promijenimo tok izvršavanja koda? Pa koristili bi naredbe za kontrolu toka:

1. if
2. while
3. for

Uslov if (if, elif, else)

Ako zelimo da donosenje odluke prepustimo nasem programu, na osnovu uslova koji se moraju ispuniti, a samim tim krairamo nas program pametnijim, uvescemo novi uslov if (naredbu if), koja se jos zove i naredba kontrole toka. Ukoliko je uslov ispunjen (Tacan - True), izvršava se naredba ili blok naredbi pod tim uslovom, u suprotnom izvršava se drugi blok naredbi ili se nastavlja ispitivanje.

Kako bi smo priblizili naredbu za kontrolu toka, navescemo primjer iz realnog zivota:

```

Probudio sam se i oprao zube
ako sam gladan
    trebam doruckovati

Trebam ici u vani
ako je oblacno
    ponijecu kisobran
u suprotnom
    ponijecu suncane naocare

U restoranu
ako zelim meso
    narucicu stejk
ako zelim pastu
    narucicu spagete
u suprotnom
    narucicu salatu
  
```

Relacijski operatori, operatori poredjenja (>, <, >=, <=, ==, !=)

Operacija	Naziv operacije	Primjer ako su : a=3 b=2 i c=3	Rezultat
>	Vece	print(a > b)	Tacno (True)
<	Manje	print(a < b)	Nije Tacno (False)
>=	Vece ili jednako	print(a >= b)	Tacno (True)

		<code>print(a >= c)</code>	Tacno (True)
<code><=</code>	Manje ili jednako	<code>print(a <= b)</code> <code>print(a <= c)</code>	Nije Tacno (False) Tacno (True)
<code>==</code>	Jednako	<code>print(a == b)</code> <code>print(a == c)</code>	Nije Tacno (False) Tacno (True)
<code>!=</code>	Nije jednako Razlicito	<code>print(a != b)</code> <code>print(a != c)</code>	Tacno (True) Nije Tacno (False)

Logicki operatori (and, or i not) ili bitski operatori

Operacija	Naziv operacije	Primjer ako su : a=3 b=2 i dobar=True	Rezultat
<code>and</code>	i	<code>if(a > 4 and b < 3):</code> <code>print("Zdravo")</code> <code>else:</code> <code>print("Dovidjenja")</code>	Dovidjenja
<code>or</code>	ili	<code>if(a > 4 or b < 3):</code> <code>print("Zdravo")</code> <code>else:</code> <code>print("Dovidjenja")</code>	Zdravo
<code>not</code>	ne, nije	<code>if(not(dobar)):</code> <code>print("Nije Dobro")</code> <code>else:</code> <code>print("Dobro je")</code>	Dobro je

Izvorni kod: kod-305_rad-sa-naredbom-if.py

```

dobar = True

if dobar:
    print("Goku je dobar karakter")

####
dobar = True
zabavan = False

if dobar or smijesan:
    print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
    print("Freza nije je zabavan")
else:

```

```
print("Freza je los karakter")

###
dobar = True
zabavan = True

if dobar and smijesan:
    print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
    print("Krilin je zabavan")
else:
    print("Freza je los karakter")

####
dobar = True
zabavan = False

if dobar or smijesan:
    print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
elseif dobar and not(zabavan):
    print("Freza nije je zabavan")
else:
    print("Freza je los karakter")
```

Izvorni kod: [kod-306_maksimalan_broj.py](#)

```
def maksimalan_broj(broj_1, broj_2, broj_3):
    if broj_1 >= broj_2 and broj_1 >= broj_3:
        return broj_1
    elif broj_2 >= broj_1 and broj_2 >= broj_3:
        return broj_2
    else:
        return broj_3

print(maksimalan_broj(7, 8, 9))
```

Kalkulator nadogradjena verzija

Referenca na [Izvorni kod: kod-192_osnovni_kalkulator.py](#)

```
broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = broj1 + broj2

print(rezultat)

broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = float(broj_1) + float(broj_2)
```

```
print(rezultat)
```

Izvorni kod: kod-306_kalkulator-nadogradjena-verzija.py

```
broj_1 = float(input("Unesite prvi broj: "))
broj_2 = float(input("Unesite drugi broj: "))
operator = input("Unesite operator: [+ , - , / , *] ")

if operator == "+":
    print(broj_1 + broj_2)
elif operator == "-":
    print(broj_1 - broj_2)
elif operator == "/":
    print(broj_1 / broj_2)
elif operator == "*":
    print(broj_1 * broj_2)
else:
    print("unijeli ste pogresan operator")
```

While petlja - (Izvršavaj blok koda sve dok je ispunjen uslov ...)

While petlja predstavlja strukturu u Python-u koja nam omogućava da prolazimo kroz isti blok koda više puta, onoliko puta koliko smo to zadali inicijalnim uslovom, odnosno sve dok uslov ima vrijednost `Tacno` (`True`) ili dok nasilno ne prekinemo uslov naredbom prekida (**`break`**).

Dakle svakom iteracijom (prolaskom) kroz blok koda, while petlja će da izvrši sve što se nalazi u tijelu petlje. Naravno, uz while petlju možemo kombinovati i uslove čime dobijamo na brzini koda i većoj efikasnosti.

Ono što je bitno napomeniti kod while petlje, ona se koristi uglavnom kada unaprijed **nemamo definisan broj iteracija**.

Izvorni kod: `kod-310_while_brojac.py`

```
i = 1
while i <= 10:
    print("Vrijednost i je : " + i)
    #i = i + 1
    i += 1

print("Kraj brojaca")
```

Izvorni kod: `kod-311_igra_pogadjanja.py`

```
# primjenimo do sad nauceno
tajna_rijec = "python"

pokusaj = ""

while guess != tajna_rijec:
    pokusaj = input("Pokusajte pogoditi tajnu rijec: ")

print("Cestitamo, pogodili ste")

# limitiraj broj pogresnih pokusaja
tajna_rijec = "python"
pokusaj = ""
pokusaj_broj = 0
pokusaj_limit = 4
kraj_igre=False

while pokusaj != tajna_rijec and not(kraj_igre):
    if pokusaj_broj < pokusaj_limit:
        pokusaj = input("Pokusajte pogoditi tajnu rijec: ")
        pokusaj_broj += 1
    else:
        kraj_igre = True

if kraj_igre:
```

```
    print("Iskoristili ste sve pokusaje. Kraj igre")
else:
    print("Cestitamo, pogodili ste")
```

Naredba prekida (**break**)

U slucaju da nasilno zelimo prekinuti petlju, koristimo naredbu prekida **break**. Na ovaj nacin se zaustavlja izvršenje petlje, cak i u slucaju da zadani uslov nije postao **False**. Takodje, u slucaju da unutar **while** petlje imate blok koda **else**, blok se **nece** izvršiti.

Izvorni kod: kod-312_demonstracija-naredbe-break.py

```
karakter_opis = {}

brojac=0
limit=10

while brojac <= limit:
    karakter_ime = input("Unesi ime karaktera: ")
    karakter_godine = input("Unesi godine karaktera :")

    if int(karakter_godine) <= 0:
        print("Godine ne mogu biti manje od 1!")
        break
    else:
        karakter_opis[(karakter_ime)] = (karakter_godine)
        brojac+=1

print(karakter_opis)
```

else kod While petlje

Kao sto smo vidjeli sa uslovom **if**, takodje mozemo koristiti granu **else** prilikom konstrukcije **while** petlje, ali trebamo imati na umu da se **else** izvršava samo jednom, ako i samo ako je glavni uslov **while** petlje netacan (**False**). Naravno ukoliko unutar **while** petlje imamo naredbu **break** koja je izvršena, **else** naredba ce biti preskocena.

Izvorni kod: kod-313_demonstracija-grane-else.py

```
karakter_opis = {}

brojac=0
limit=2

while brojac <= limit:
    karakter_ime = input("Unesi ime karaktera: ")
    karakter_godine = input("Unesi godine karaktera :")
```

```
if int(karakter_godine) <= 0:
    print("Godine ne mogu biti manje od 1!")
    break
else:
    karakter_opis[(karakter_ime)] = (karakter_godine)
    brojac+=1
else:
    # sadrzaj rijecnika ce biti ispisan samo ako se
    # kompletan program izvrši bez okidanja/trigerovanja naredbe break
    print(karakter_opis)
```

Naredba nastavljanja (**continue**)

U slučaju da nasilno želimo prekinuti **dio petlje** ili blaze receno preskociti izvršavanje tog bloka ali nastaviti sa izvršavanjem sledećeg bloka unutar iste petlje kositicemo naredbu nastavljanja **continue**.

Izvorni kod: kod-312_demonstracija-naredbe-break.py

```
karakter_opis = {}

brojac=0
limit=10

while brojac <= limit:
    karakter_ime = input("Unesi ime karaktera: ")
    karakter_godine = input("Unesi godine karaktera :")

    if int(karakter_godine) <= 0:
        print("Godine ne mogu biti manje od 1!")
        break
    elif int(karakter_godine) >= 100:
        print("Treba li biste dobiti Nobelovu nagradu za godine")
        continue
    else:
        karakter_opis[(karakter_ime)] = (karakter_godine)
        brojac+=1

print(karakter_opis)
```


For petlja

For petlju (ili **for..in**) mozemo nazvati specijalni tip petlje u Python-u, a za razliku od while petlje, for petlju koristimo kada zelimo da vrsimo iteraciju kroz tijelo petlje ako unaprijed **znamo broj potrebnih iteracija**.

Vrijednosti se uglavnom zadaju kao predefinisane ali mozemo koristiti izvore poput lista, stringova, rijecnika.

Izvorni kod: kod-320_rad-sa-for-petljom.py

```
# operator in, za iteraciju nad listama, torkama, rijecnicima mozemo
koristit kljucnu rijec in
for slovo in "programiranje.ba":
    print(slovo)

for karakter in karakteri:
    print(karakter)

karakter_i = ["Goku", "Kirlin", "Yamcha"]
for indeks in range(len(karakter_i)):
    print (karakter_i[indeks])

loto_brojevi = [1, 33, 13, 43, 56]
for broj in loto_brojevi:
    print (broj)

for broj in range(20):
    print(broj)

for broj in range(14, 20):
    print (broj)

for indeks in range(len(karakter_i)):
    print (karakter[indeks])

for broj in range(5):
    if broj == 0:
        print("prvi pokusaj")
    else:
        print("ostali")

for i in range(10):
    print (i)
    i+=1

# break naredba unutar for petlje
karakter_i = ["Goku", "Kirlin", "Yamcha", "Goku", "Bulma"]
for i in range(len(karakter_i)):
    if karakter_i[i] == "Yamcha":
        print("Prekini izvršenje for petlje")
        break
```

```

    print (karakter[i])

# enumerate() funkcija - enumeracija
# ukoliko zelimo da zajedno sa vrijednostima iz liste, stringa ili
# rijecnika ispisujemo i njihove
# indekse
# koristicemo funkciju enumerate()
karakter = ["Goku", "Kirlin", "Yamcha", "Goku", "Bulma"]
for i, ime in enumerate(karakter):
    print(str(i) + " " + ime)

### eksponencijaln funkcija - kada ne znamo koliki je eksponent
#print(2**3)

def eksponent_broja(baza, eksponent):
    rezultat = 1
    for i in range(eksponent):
        rezultat = rezultat * baza
    return rezultat

print(eksponent_broja(2,3))

```

Primjer algoritma sortiranje mjehuricama (bubble sorting)

Prije nego napisemo kod potrebno je kratko objasnjenje algoritma. Algoritam sortiranje mjehuricama, ima za cilj da nad zadatim nizom nasumicnih/slucajnih brojeva izvrši sortiranje od najmanjeg ka najvećem. Ovakvi tipovi zadataka predstavljaju osnovne koncepte teorije algoritma, a možemo ih naći, kao zadatke, na intervjuima u velikim firmama poput Google-a, Amazon-a, Facebook-a, Microsoft-a ...

Predpostavimo da imamo niz brojeva:

```
[4, 2, 1, 5, 3]
```

Primjenom algoritma sortiranja mjehuricama, svakom novom iteracijom, svaki element niza će se upoređivati sledećim, u slučaju da je prvi element veći od sledećeg, zamenjuje mesta, u suprotnom prvi element ostaje na svom mestu. Ovaj proces se nastavlja sve dok se svi elementi konačno ne sortiraju od najmanjeg ka najvećem. Dakle, procedura sortiranja će se obaviti sledećim redoslijedom:

Pocetno stanje	[4, 2, 1, 5, 3]	Objasnjenje
Nakon 1. iteracije	[(2, 4), 1, 5, 3]	(4 > 2) 4 i 2 mijenjaju mesta
	[2, (1, 4), 5, 3]	(4 > 1) 4 i 1 mijenjaju mesta
	[2, 1, (4, 5), 3]	(4 < 5) 4 i 5 ostaju na svom mestu
	[2, 1, 4, (3, 5)]	(5 > 3) 5 i 3 mijenjaju mesta
Nakon 2. iteracije	[(1, 2), 4, 3, 5]	(2 > 1) 2 i 1 mijenjaju mesta
	[1, (2, 4), 3, 5]	(1 < 4) 2 i 4 ostaju na svom mestu
	[1, 2, (3, 4), 5]	(4 > 3) 4 i 3 mijenjaju mesta

| [1,2,3,(4,5)] | (4 < 5) 4 i 5 ostaju na svom mjestu

Izvorni kod: kod-321_sortiranje-mjehuricama.py

```
niz_brojeva = [4,2,1,5,3]

# inicijalno stanje varijable, koja nam govori da li je bilo
# zamjene brojeva prilikom iteracije kroz niz
zamjena_izvrsena = True

zadnji_element_niza = (len(niz_brojeva) - 1)

while zamjena_izvrsena:
    # pretpostavimo da je niz sortiran
    zamjena_izvrsena = False

    # isijecamo poslednji element niza, jer unutar petlje provjeravamo
    # naredni preko indeks + 1
    for indeks, broj in enumerate(niz_brojeva[0:zadnji_element_niza]):
        if niz_brojeva[indeks] > niz_brojeva[indeks+1]:
            # mijenjamo mjesta elemenata niza
            niz_brojeva[indeks], niz_brojeva[indeks+1] = niz_brojeva[indeks+1], niz_brojeva[indeks]
            zamjena_izvrsena = True
    else:
        print(niz_brojeva)
```

Izvorni kod: kod-322_sortiranje-mjehurica-slucajni-izbor.py

```
import random
niz_brojeva=[]
for i in range (10000):
    #print (random.randint(1,100))
    niz_brojeva.append(random.randint(1,10000))

print (niz_brojeva)

# niz_brojeva = [4,2,1,5,3]

zamjena_izvrsena = True

zadnji_element_niza = (len(niz_brojeva) - 1)

while zamjena_izvrsena:
    zamjena_izvrsena = False
```

```
for indeks, broj in enumerate(niz_brojeva[0:zadnji_element_niza]):  
    #print(str(indeks) + " -> " + str(broj))  
  
    if niz_brojeva[indeks] > niz_brojeva[indeks+1]:  
  
niz_brojeva[indeks], niz_brojeva[indeks+1]=niz_brojeva[indeks+1], niz_brojeva  
[indeks]  
    zamjena_izvrsena = True  
  
else:  
    print(niz_brojeva)
```

Dvodimenzionalne liste i ugniježdene petlje (nested)

Izvorni kod: kod-355_rad-sa-dvodimenzionalnim-listama.py

```
 mreza = [  
     [1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9],  
     [0]  
 ]  
  
 print(mreza[0][2])  
 print(mreza[2][1])  
 print(mreza[3][0])  
  
 # nested for loop  
  
 # mreza = [  
 #     [1, 2, 3],  
 #     [4, 5, 6],  
 #     [7, 8, 9],  
 #     [0]  
 # ]  
  
 for row in mreza:  
     for col in row:  
         print(col)  
  
 for col in mreza:  
     for row in col:  
         print(row)  
  
 # cezarova sifra u python  
 def enkripcija(tekst, pomak):  
     rezultat = ""  
  
     for broj in range(len(tekst)):  
         slovo = tekst[broj]  
  
         if(slovo.isupper()):  
             rezultat += chr((ord(slovo) + pomak - 65) % 26 + 65)  
         else:  
             rezultat += chr((ord(slovo) + pomak - 97) % 26 + 65)  
     return rezultat  
  
 tekst = input("Unesite tekst: ")  
 pomak = 2  
  
 print("Unijeli ste: " + tekst)  
 print("Pomak: " + str(pomak))  
 print("Sifrovan tekst: " + enkripcija(tekst, pomak))
```


Hvatanje greski - try / except (catch)

Kada imamo situaciju da bez obzira na gresku u nasem programu, ipak zelimo da nastavimo sa izvršenjem programa i damo smislenu povratnu informaciju korisniku, tada koristimo **try/except** za hvatanje greske.

Izvorni kod: kod-405_hvatanje-greski.py

```
try:
    broj = int(input("Unesite broj: "))
    print(broj)
except:
    print("Pogresan unos")

###

try:
    vrijednost = 10 / 0
    broj = int(input("Unesite broj: "))
    print(broj)
except ZeroDivisionError:
    print("Dijeljenje sa nulom")
except ValueError:
    print("Pogresan unos")

###

try:
    vrijednost = 10 / 0
    broj = int(input("Unesite broj: "))
    print(broj)
except ZeroDivisionError as err:
    print(err)
except ValueError:
    print("Pogresan unos")
```

Klase i objekti

Znamo od pocetka da je Python objektno orijentisani programski jezik, sto znaci da su svi podaci predstavljeni kao objekti. Prihvatanjem objektno orijentisanog nacina programiranja, nasi programi, ali sam kod, postaju ekstremno korisni, organizovaniji i mocniji.

Kada programiramo susrecemo se sa razlicitim tipovima podataka, takodje susrecemo se sa razlicitim strukturama podataka, ali sta u slucaju kad ne mozemo predstaviti neku pojavu iz prirode sa vec postojećim tipovima ili strukturama podataka?

Upravo su nam za to korisne klase i objekti. U Python-u mozemo krairati klase (definise vas licni tip podatka, ponasa se kao template, patern kako nesto treba da izgleda). Objekat je podatak u memoriji, pravi podatak kreiran iz klase, sto se jos naziva instanciranje objekta.

Metode su funkcije unutar klase, koje se koriste da izvrse neku akciju nad odredjenim tipom objekta.

Sintaksa klase:

```
class naziv_klase():                # kljucna rijec class
    def __init__(self, param_1, param_2): # konstruktor/funkcija __init__
        self.param_1 = param_1
        self.param_2 = param_2

klasa = naziv_klase()                # instanciranje objekta
```

Iz sintakse vidimo da u slucaju pojave kljucne rijeci **class** Python automatski prepoznaje da je rijec o novoj klasi. Takodje, primijetimo da je za instanciranje objekta, potrebno da imamo funkciju **** init **** zaduzenu za instanciranje, koja se jos naziva konstruktorom i uvijek prihvata minimalno jedan element **self**. Taj prvi element, koji prima svaka funkcija unutar klase, se uzima kao licni, prisvojni, sebi svojstven, element i sve sto se radi nad samim objektom se referencira pomocu ovog elementa.

Prazan objekat:

```
class prazna_klasa():
    pass

klasa = prazna_klasa()
```

Primijetimo da prethodni objekat nema funkciju inicijalizacije, postoji podrazumijevana funkcija, koju ne vidimo, ali moramo znati da je ovaj objekat prazan objekat.

Posto ne postoji student tip podaka, kreiracemo klasu **Student**.

Izvorni kod: kod-555_student.py


```
class Student:
    # inicijalizacija klase (inicijalna funkcija)
    def __init__(self, ime, smjer, ocjena, brucos):
        self.ime = ime
        self.smjer = smjer
        self.ocjena = ocjena
        self.brucos = brucos
```

Izvorni kod: kod-556-main.py

```
from student import Student

# kreiranje instance Student student_prvi objekta
student_prvi = Student("Goku", "Programiranje", 8, False)
print(student_prvi)
print(student_prvi.ime) # pristup objektu ime unutar objekta student_prvi
print(student_prvi.ocjena)
# student_prvi.ocjena znaci da imamo pristup ocjeni instance student_prvi

student_drugi = Student("Krillin", "Ekonomija", 8.3, True)
print(student_drugi.brucos) # pristup objektu brucos unutar objekta
student_drugi
```

Student		<= naziv klase
ime	"Goku"	<= objekat ime unutar objekta Student
smjer	"Programiranje"	<= objekat smjer unutar objekta Student
ocjena	8	<= objekat ocjena unutar objekta Student
brucos	False	<= objekat brucos unutar objekta Student

metode - funkcije unutar klase (funkcije objekta)

Izvorni kod: kod-557_funkcija-unutar-klase.py

```
class Student:
    naziv_fakulteta = "Elektrotehnicki" # podrazumijevana/defaultna
    vrijednost
    def __init__(self, ime, smjer, ocjena, brucos):
        self.ime = ime
        self.smjer = smjer
        self.ocjena = ocjena
        self.brucos = brucos
```

```
def dobar(self):
    if self.ocjena > 8:
        return True
    else:
        return False
```

Izvorni kod: kod-558_main.py

```
from student import Student

student_prvi = Student("Goku", "programiranje", 7.9, False)
print(student_prvi.naziv_fakulteta)
#student_prvi.ime = "Goku"
#student_prvi.ocjena = 7.9

print(student_prvi.ime)
print(student_prvi.dobar())
student_prvi.naziv_fakulteta = "Prirodno Matematički"
student_prvi.smjer = "Fizika"
student_prvi.ocjena = 9.0

print(student_prvi.ime)
print(student_prvi.naziv_fakulteta)
print(student_prvi.dobar())
```

U prethodnom primjeru smo vidjeli jednu od metoda **dobar()**, ali ajde da pogledamo malo detaljnije o čemu se radi.

Recimo da želimo funkciju u nasoj klasi **Student** koja nam ispisuje kompletan sadržaj instanciranog objekta **Student**, ali i omiljenog pisca tog studenta. Za ovo su nam potrebne dvije metode unutar klase **Student** koje ćemo nazvati **student_opis** i **student_pisac**.

Izvorni kod: kod-559-metode.py

```
class Student:
    naziv_fakulteta = "Elektrotehnički"
    pisac = "nije definisan"
    def __init__(self, ime, smjer, ocjena, brucos):
        self.ime = ime
        self.smjer = smjer
        self.ocjena = ocjena
        self.brucos = brucos

    def dobar(self):
        if self.ocjena > 8:
            return True
        else:
            return False
```

```
def student_opis(self):
    print(f"Ime studenta: {self.ime}, Naziv fakulteta:
{self.naziv_fakulteta}, Smjer: {self.smjer}, Ocjena: {self.ocjena}, Brucos:
{self.brucos}, Pisac: {self.pisac}")

def student_pisac(self):
    print(self.pisac)
```

Izvorni kod: kod-560_main.py

```
from student import Student

novi_student = Student("Goku", "programiranje", 7.9, False)
print(novi_student.naziv_fakulteta)
#novi_student.ime = "Goku"
#novi_student.ocjena = 7.9

print(novi_student.ime)
print(novi_student.dobar())
novi_student.naziv_fakulteta = "Prirodno Matematicki"
novi_student.smjer = "Fizika"
novi_student.ocjena = 9.0

print(novi_student.ime)
print(novi_student.naziv_fakulteta)
print(novi_student.dobar())
print(novi_student.student_opis())
print(novi_student.student_pisac())
```

Naslijedjivanje - podklase

U slucaju da planiramo kreirati novu klasu, a vec unaprijed znamo da imamo ili da vec postoji klasa koja sadrzi vecinu metoda koje trebamo koristiti, medjutim mi trebamo definisati jos par novih objekata ili novih metoda, u tom slucaju koristimo naslijedjivanje ili podklase, tako sto kreiramo novu klasu koja naslijedjuje staru klasu.

Nova klasa ce imati sve osobine stare klase sa novim opcijama.

Izvorni kod: kod-565_kuvar.py

```
class Kuvar:
    def priprema_mesa(self):
        print ("Priprema pileceg mesa.")

    def priprema_salate(self):
        print("Priprema Cezar salata")

    def priprema_specijalnog_jela(self):
        print("Priprema rebarcadi")
```

Izvorni kod: kod-566_kineski_kuvar.py

```
from kuvar import Kuvar

class KineskiKuvar(Kuvar):
    def priprema_rize(self):
        print("Priprema rize na kineski nacin")

    def priprema_specijalnog_jela(self):
        print ("Pekinska patka")
```

Dakle, sve metode koje su definisane u klasi **Kuvar** , koja se jos naziva nadklasa, mogu se koristiti u novoj klasi **KineskiKuvar** , koja se jos naziva podklasa. Mozemo primijetiti redefinisane metode **priprema_specijalnog_jela** u novoj klasi **KineskiKuvar**. U slucaju redefinisanja metode, prilikom pozivanja metode **priprema_specijalnog_jela**, koristi se nova redefinisana metoda.

Izvorni kod: kod-567_main.py

```
from kuvar import Kuvar
from kineski_kuvar import KineskiKuvar

novi_kuvar = Kuvar()

novi_kuvar.priprema_mesa()
novi_kuvar.priprema_specijalnog_jela()
```

```
novi_kineski_kuvar = KineskiKuvar()  
  
novi_kineski_kuvar.priprema_rize()  
novi_kineski_kuvar.priprema_mesa()  
novi_kineski_kuvar.priprema_specijalnog_jela()
```

Rad sa datotekama (fajlovima)

Fajl ili datoteka, predstavlja kontejner, mjesto gdje skladištimo, pohranjujemo podatke. Na ovaj način podaci su trajno sacuvani na disku, za razliku od radne memorije i može im se naknadno pristupiti iz drugih programa, ali se može vršiti i razmjena između različitih sistema.

Citanje iz eksternog fajla

- dosta puta imamo potrebu za citanjem sadržaja iz drugih fajlova
- parsiranje teksta ...
- apsolutni, relativna lokacija

access_mode	rezim rada nad fajlom nakon otvaranja
r - read (citanje)	otvara datoteku samo za citanje, ovo je default rezim, ako nije navedeno drugacije, uzima se ovaj rezim
r+	otvara datoteku za citanje i pisanje
w - write (pisanje)	otvara datoteku samo za pisanje, ako datoteka već postoji snima se nova datoteka preko nje, ako ne postoji kreira novu datoteku.
w+	otvara datoteku za pisanje i citanje, ako datoteka već postoji snima se nova datoteka preko nje, ako ne postoji kreira novu datoteku
a - append (dodavanje)	otvara datoteku za dodavanje i citanje, dodaje liniju na kraju datoteke, u slučaju da datoteka ne postoji kreira se nova
a+	otvara datoteku za dodavanje i citanje, dodaje liniju na kraju datoteke, u slučaju da datoteka ne postoji kreira se nova

Sadržaj fajla: fajl-605_karakter_i_porijeklo.txt

```
Goku - Vegeta
Krillin - Zemlja
Piccolo - Namek
Frieza - Universe 7
```

Izvorni kod: kod-606_rad-sa-fajlovima.py

```
# otvoren fajl
```

```
# funkcija open()
karakter_i_fajl = open("fajl-605_karakter_i_porijeklo.txt", "r")

# provjeri da li je fajl citljiv
print(karakter_i_fajl.readable())

# citanje informacija iz fajla
print(karakter_i_fajl.read())

# citanje linije u fajlu
print(karakter_i_fajl.readline())
print(karakter_i_fajl.readline())
print(karakter_i_fajl.readline())
print(karakter_i_fajl.readline())

# citanje linije po liniju, citaj svaku liniju i pohrani u niz
print(karakter_i_fajl.readlines())

# koristenjem for petlje
for karakter in karakter_i_fajl.readlines():
    print(karakter)

# obzirom da se tokom rada sa fajlovima koristi pomocna memorija (buffer),
# nakon rada sa fajlom potrebno je da se pozove funkcija close(), kako bi
se
# podaci upisali u fajl
# cak i ako koristimo funkciju write() ali na kraju ne pozovemo close()
# podaci ce biti izgubljeni
# moguće je zadati velicinu pomocne memorije kao treci parametar u funkciji
# open() reda bajta.
karakter_i_fajl.close()
```

Upisivanje u eksterni fajl

Izvorni kod: kod-607_rad-sa-fajlovima.py

```
# dodavanje na vec postojeći fajl
karakter_i_fajl = open("fajl-605_karakter_i_porijeklo.txt", "a")

karakter_i_fajl.write("Bulma - Zemlja")
karakter_i_fajl.write("\n Chi-Chi - Zemlja")

karakter_i_fajl.close()

# upisivanje u novi fajl
karakter_i_fajl = open("fajl-608_karakter_i_porijeklo.txt", "w")

karakter_i_fajl.write("Bulma - Zemlja")
karakter_i_fajl.write("\n Chi-Chi - Zemlja")

karakter_i_fajl.close()
```


Moduli i pip alat

Moduli nisu nista drugo, nego odredjen broj funkcija koje mozete importovati, pozvati u vas kod. Takodje, mozete napraviti vas modul jednostavnim kreiranjem python fajla koji sadrzi vase funkcije i kasnije ga pozvati u vas kod naredbom **import**.

U ovoj lekciji cete nauciti kako kako importovati standardne module, instalirati eksterne module sa **pip** komandom, kako ih importovati nakon instalacije, i konacno kako kreirati i importovati svoj modul.

Importovanje standardnih Python modula **import**

Recimo da nam je potrebna informacija, prilikom pisanja naseg koda, koja nam govori u kom se trenutno direktoriju nalazimo. Za ovo bi nam trebalo dosta prethodnog iskusta, vremena i poznavanja operativnog sistema, a da ne pricamo da su strukture direktorija razlicite na razlicitim operativnim sistemima. U tu svrhu jednostavno mozemo da importujemo standardni Python modul **os**, bez obzira na kom operativnom sistemu radimo nacin koriscenja je isti.

```
import os

print(help(os))

print("Trenutni radni direktorij: ")
print(os.getcwd())
```

Kada Python uspjesno importuje modul **os**, on u principu pronadje na vasem sistemu fajl po imenu **os.py**, u prethodno definisanom direktorijumu, specijalno namijenjenom za skladištenje standardnih modula.

Djelimicno importovanje **from .. import**

tbd.

```
{## how to install with pip
## how to import and use modules
}

tbd.
```text
{## how to install with pip
how to import and use modules
}
```

Izvorni kod: [kod-655\\_korisni-alati.py](#)

```
#
import random

def srecan_broj(broj):
 return random.randint(1, broj)

def pozdrav(tekst):
 return ("Pozdrav " + tekst)

##
import korisni_alati

print(korisni_alati.srecan_broj(3))
print(korisni_alati.pozdrav("Goku"))

##
from korisni_alati import *

print(srecan_broj(3))
print(pozdrav("Goku"))

##
import korisni_alati as ka

print(ka.srecan_broj(3))
print(ka.pozdrav("Goku"))
```

#### devet\_depresivaca.py - (Rambo Amadeus - Devet depresivaca)

```
print("9 depresivaca gajili su bostan")
print("Puko lastik od bandzija, ostalo ih 8")

print("8 depresivaca, k'o u dlan ih gledam")
print("u krivini hladnjaca, ostalo ih 7")

print("7 depresivaca, opet losa vijest")
print("Neuzemljen bojler, ostalo ih 6")

print("6 depresivaca, turbulentan let")
print("Dnevno kilo vinjaka, ostalo ih 5")

print("5 depresivaca bez mrlje na jetri")
print("Moca od pecenja, ostalo ih 4")

print("4 depresivca, veseli su svi")
print("Droga jeftinija od viskija, ostalo ih 3")

print("3 depresivca, svaki od njih vrijedan")
print("Dvojici crko facebook, ostao je 1")
```

```
print("1 depresivac, oprezan je bio")
print("Onda se ozenio")
```

```
Vjesala

import random

fajl = "rijecnik.txt"

def ucitaj_rijeci():
 """
 Vraca listu validnih rijeci. Rijeci su tipa string, napisane malim
 slovima

 U zavisnosti od duzine liste rijeci, ova funkcija moze potrajati.
 """
 print("Ucitavanje rijeci iz fajla 'rijecnik.txt'...")
 # otvori_fajl: fajl
 otvori_fajl = open(fajl, 'r')
 # linija: string (procitaj liniju u fajlu - citav fajl je napisan kao
 jedna linija)
 linija = otvori_fajl.readline()
 # lista_rijeci: lista rijeci (rijec po rijec)
 lista_rijeci = linija.split()
 print(" ", len(lista_rijeci), "rijeci ucitano.")
 return lista_rijeci

listarijeci_pomoc = ucitaj_rijeci.__doc__
print(listarijeci_pomoc)
print(listarijeci)
lista_rijeci = ucitaj_rijeci()
print(lista_rijeci)

def izbor_rijeci(lista_rijeci):
 """
 fajl (lista): lista rijeci (string)

 Funkcija vraca slucajnu rijec iz liste lista_rijeci
 """
 return random.choice(lista_rijeci)

izabrana_rijec = izbor_rijeci(lista_rijeci)
print(izabrana_rijec)

kraj pomocnog koda

Ucitaj listu rijeci u varijablu lista_rijeci
kako bi smo mogli pristupiti listi bilo gdje iz programa
lista_rijeci = ucitaj_rijeci()
```

```
def da_li_je_rijec_pogodjena(tajna_rijec, pogodjena_slova):
 '''
 tajna_rijec: string, rijec koju igrac pogadja
 pogodjena_slova: lista, koja slova su pogodjena
 Funkcija vraca: boolean, Tacno (True) ako su sva koja se nalaze u
 tajna_rijec rijeci takodje
 u pogodjena_slova, u suprotnom Netacno (False)
 '''
 brojac=0
 for slovo in tajna_rijec:
 if slovo in pogodjena_slova:
 brojac+=1
 if brojac==len(tajna_rijec):
 return True
 else:
 return False

def dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova):
 '''
 tajna_rijec: string, rijec koju igrac pogadja
 pogodjena_slova: lista, koja slova su pogodjena
 Funkcija vraca: string, kombinaciju slova i donjih crtica koji
 predstavljaju pogodjena slova i
 slova koja jos nisu pogodjena, respektivno.
 '''
 lista=[]
 rijec=""
 for key in tajna_rijec:
 if key in pogodjena_slova:
 rijec+=key
 else:
 rijec+="_ "
 return rijec

tajna_rijec="asa"
pogodjena_slova=["a"]
rijecica=dohvati_pogodjenu_rijec(tajna_rijec,pogodjena_slova)
print(rijecica)

def dohvati_raspoloziva_slova(pogodjena_slova):
 '''
 pogodjena_slova: lista, koja slova su pogodjena
 Funkcija vraca: string, listu slova koji sacinjavaju slova koja jos
 trebaju biti pogodjena.
 '''
 rijec=""
 brojac=0
 slova='abcdefghijklmnopqrstuvwxyz'
 for slovo in slova:
 if slovo in pogodjena_slova:
 brojac+=1
 else:
 rijec+=slovo
```

```
 return rijec

pogodjena_slova=["a"]
rijecica=dohvati_rasploziva_slova(pogodjena_slova)
print(rijecica)

def vjesala(tajna_rijec):
 '''
 tajna_rijec: string, rijec koju igrac pogadja
 pogodjena_slova: lista, koja slova su pogodjena
 Funkcija vraca: string, kombinaciju slova i donjih crtica koji
 predstavljaju pogodjena slova i
 slova koja jos nisu pogodjena, respektivno.

 Startuje interaktivnu igricu vjesala.

 * Na pocetku igre, daje informaciju igracu koliko
 slova ima tajnoj rijeci.

 * Pitaj igraca da proslijedi jedno slovo po pokusaju.

 * Igrac bi trebao dobiti informaciju odmah nakon pokusaja
 gdje se u rijeci nalazi slovo u slucaju da je pogodio.

 * Nakon svakog pokusaja, igrac bi trebao vidjeti djelimicno
 pogodjenu rijec, ali i slova koja nedostaju u rijeci.

 '''
 # main funkcija
 duzina_tajne_rijeci=len(tajna_rijec)
 print("Dobrodošli u igricu, Vjesala!")
 print("Razmisliam o rijeci duzine " + str(duzina_tajne_rijeci) + "
karaktera.")
 dozvoljen_broj_pokusaja=2*len(tajna_rijec)
 i=0
 pogodjena_slova=[]
 while (dozvoljen_broj_pokusaja != 0):
 print("-----")
 if tajna_rijec != dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova):
 print("Imate jos " + str(dozvoljen_broj_pokusaja) + "
pokusaja.")
 print("Rasploziva slova:
",dohvati_rasploziva_slova(pogodjena_slova))
 pokusaj=input("Molimo pokusajte slovo: ")
 pokusaj_mala_slova = pokusaj.lower()

 if pokusaj_mala_slova in pogodjena_slova:
 print("Ups! Vec ste pokusali to slovo:
",dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova))

 elif pokusaj_mala_slova not in tajna_rijec:
 print("Ups! Slovo se ne nalazi u rijeci koju sam
```

```

zamislio:",dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova))
 dozvoljen_broj_pokusaja-=1
 else:
 pogodjena_slova.append(pokusaj_mala_slova)
 print("Pogodili ste: ",dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova))
 #chances+=1
 pogodjena_slova.append(pokusaj_mala_slova)
 elif tajna_rijec==dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova):
 print("Cestitamo!")
 break
 else:
 print("-----")
 print("Nazalost, nemate vise pokusaja. Tajna rijec je bila "+
tajna_rijec +".")

```

```

Kada zavrсите funkciju vjesala, mozete testirati vas kod definisanjem
tajne rijeci tajna_rijec="tajna"

```

```

#tajna_rijec= izbor_rijeci(lista_rijeci).lower()
#tajna_rijec="tajna"
tajna_rijec=izbor_rijeci(lista_rijeci)

```

```

vjesala(tajna_rijec)

```

```

``````python

```

```

from random import randint
mreza=[]

```

```

for i in range(0,7):
    mreza.append(["#"] * 7)
def stampaj_mrezu (board):
    for red in mreza:
        print (" ".join(red))

```

```

print ('Igra Potapanje brodova moze da pocne!')
stampaj_mrezu(mreza)

```

```

#Slučajnim odabirom u mrežu ubacujemo oba broda

```

```

def nasumicni_red(mreza):
    return randint(0, len(mreza)-1)

```

```

def nasumicna_kolona(mreza):
    return randint(0, len(mreza[0])-1)

```

```

#brod 1

```

```

red_1 = nasumicni_red(mreza)
kol_1 = nasumicna_kolona(mreza)

```

```

#brod 2

```

```

red_2 = nasumicni_red(mreza)
kol_2 = nasumicna_kolona(mreza)

```

```
#Da se brodovi ne bi preklapali, potrebno je da nemaju zajednička polja
#To se obezbeđuje funkcijom razliciti()
def razliciti(r,c):
    while r == red_1 and c == kol_1:
        r = nasumicni_red(mreza)
        c = nasumicna_kolona(mreza)
        red_2 = r
        kol_2 = c
    razliciti(red_2,kol_2)
#Kada izaberete jedno polje, preostala dva mogu biti horizontalno(levo ili
desno) ili vertikalno(gore ili dole)
#Zato se definišu sledeći pravci
def nasumicni_pravac():
    n = randint(1,4)
    if n == 1:
        return "gore"
    elif n == 2:
        return "desno"
    elif n == 3:
        return "dole"
    elif n == 4:
        return "levo"
#Nasumično se odredi pravac, i na osnovu njega sledeća dva polja
while True:
    d = nasumicni_pravac()
    if d == "gore":
        if red_1 >= 2:

            red_1_2 = red_1 - 1
            kol_1_2 = kol_1
            red_1_3 = red_1 - 2
            kol_1_3 = kol_1
            break
    if d == "desno":
        if kol_1 <= len(mreza[0])-3:

            red_1_2 = red_1
            kol_1_2 = kol_1 + 1
            red_1_3 = red_1
            kol_1_3 = kol_1 + 2
            break
    if d == "dole":
        if red_1 <= len(mreza)-3:

            red_1_2 = red_1 + 1
            kol_1_2 = kol_1
            red_1_3 = red_1 + 2
            kol_1_3 = kol_1
            break
    if d == "levo":
        if kol_1 >= 2:

            red_1_2 = red_1
            kol_1_2 = kol_1 - 1
```

```
        red_1_3 = red_1
        kol_1_3 = kol_1 - 2
        break
brod_1 = [(red_1 ,kol_1 ),(red_1_2 ,kol_1_2 ),(red_1_3 ,kol_1_3 )]

#drugi brod:
while True:
    #Nasumično se odredi pravac, i na osnovu njega sledeća dva polja
    #Uslov je da se ne preklapaju sa poljima prvog broda
    d = nasumicni_pravac()
    if d == "gore":
        if red_2 >= 2:
            if (red_2 - 1,kol_2) not in brod_1 and (red_2 - 2,kol_2) not in
brod_1:

                red_2_2 = red_2 - 1
                kol_2_2 = kol_2
                red_2_3 = red_2 - 2
                kol_2_3 = kol_2
                break
    if d == "desno":
        if kol_2 <= len(mreza[0])-3:
            if (red_2 ,kol_2 + 1) not in brod_1 and (red_2,kol_2 + 2) not
in brod_1:

                red_2_2 = red_2
                kol_2_2 = kol_2 + 1
                red_2_3 = red_2
                kol_2_3 = kol_2 + 2
                break
    if d == "dole":
        if red_2 <= len(mreza)-3:
            if (red_2 + 1 ,kol_2) not in brod_1 and (red_2 + 2,kol_2) not
in brod_1:

                red_2_2 = red_2 + 1
                kol_2_2 = kol_2
                red_2_3 = red_2 + 2
                kol_2_3 = kol_2
                break
    if d == "levo":
        if kol_2 >= 2:
            if (red_2 ,kol_2 - 1) not in brod_1 and (red_2,kol_2 - 2) not
in brod_1:

                red_2_2 = red_2
                kol_2_2 = kol_2 - 1
                red_2_3 = red_2
                kol_2_3 = kol_2 - 2
                break
```



```
tacan = 0 #U ovoj promenljivoj smešta se ukupan broj pogođenih polja oba broda
prvi_brod = 0 #U ovoj promenljivoj smešta se broj pogođenih polja prvog broda
drugi_brod = 0 #U ovoj promenljivoj smešta se broj pogođenih polja drugog broda
#Na početku nemamo nijedno pogođeno polje, pa sve promenljive postavljamo na 0
#U ovoj promenljivoj smešta se ukupan broj pogođenih polja oba broda
tacan = 0
#U ovoj promenljivoj smešta se broj pogođenih polja prvog broda
prvi_brod = 0
#U ovoj promenljivoj smešta se broj pogođenih polja drugog broda
drugi_brod = 0

#Dozvoljeno je 15 pokušaja da se potope oba broda
for pokusaj in range(1,16):
    print (str(pokusaj ) + '. pokusaj:')

    nagadjanje_reda = int(input('Pogodite red:'))
    nagadjanje_kolone = int(input('Pogodite kolonu:'))
#Ispituje se da li je korisnik pogodio neko polje prvog broda
#Ako jeste, broj pogođenih polja se povećava za jedan
    if ((nagadjanje_reda -1 == red_1 ) and (nagadjanje_kolone -1 == kol_1)) or ((nagadjanje_reda -1 == red_1_2 ) and (nagadjanje_kolone -1 == kol_1_2)) or((nagadjanje_reda -1 == red_1_3 ) and (nagadjanje_kolone -1 == kol_1_3)) and mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] != 'X' :

        tacan = tacan+1
        prvi_brod = prvi_brod + 1

        if (tacan != 6) and (prvi_brod != 3) :

            print ('Bravo, pogodak!')
            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'X'
#Ako je pogođeno polje treće polje prvog broda, korisnik se obaveštava da je potopio ceo brod
            elif (tacan != 6) and (prvi_brod == 3):

                mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'X'

                print ('Bravo, potopili ste ceo brod! Ostao vam je jos jedan!')
#Ako je reč o šestom pogođenom polju, korisnik se obaveštava da je potopio oba broda
                if (tacan == 6):
                    mreza[nagadjanje_reda -1][nagadjanje_kolone -1] = 'X'
                    print ('Svaka cast, potopili ste oba broda!')

                    break

#Ispituje se da li je korisnik pogodio neko polje drugog broda
#Ako jeste, broj pogođenih polja se povećava za jedan
```

```
elif ((nagadjanje_reda -1 == red_2 ) and (nagadjanje_kolone -1 == kol_2)) or ((nagadjanje_reda -1 == red_2_2 ) and (nagadjanje_kolone -1 == kol_2_2)) or ((nagadjanje_reda -1 == red_2_3 ) and (nagadjanje_kolone -1 == kol_2_3)) and mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] != 'Y' :
    tacan = tacan+1
    drugi_brod = drugi_brod + 1
    if (tacan != 6) and (drugi_brod != 3):

        print ('Bravo, pogodak!')
        mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'Y'
    elif (tacan != 6) and (drugi_brod ==3):

        mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'Y'
#Ako je pogodeno polje, treće polje prvog broda, korisnik se obaveštava da je potopio ceo brod

    print ('Bravo, potopili ste ceo brod! Ostao vam je jos jedan!')
#Ako je reč o šestom pogodenom polju, korisnik se obaveštava da je potopio oba broda

    if (tacan == 6):
        mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'Y'
        print ('Svaka cast, potopili ste oba broda!')
        break
    else:
        if (nagadjanje_reda < 1 or nagadjanje_reda > 7) or (nagadjanje_kolone < 1 or nagadjanje_kolone > 7):
            print ('Ups, izvan opsega ste!')
        elif (mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1]=='X'):
            print ('Vec ste pronasli ovaj deo broda!')
        elif (mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1]=='0'):
            print ('Vec ste pogadjali isto polje!')
        else:
            print ('Promasili ste!')
            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = '0'

stampaj_mrezu(mreza)
if (pokusaj == 15):
    print ('Igra je završena!')

if (mreza[red_1 ][kol_1 ] != "X" or mreza[red_1_2 ][kol_1_2 ] != "X" or mreza[red_1_3 ][kol_1_3 ] != "X") or (mreza[red_2 ][kol_2 ] != "Y" or mreza[red_2_2 ][kol_2_2 ] != "Y" or mreza[red_2_3 ][kol_2_3 ] != "Y"):
    print ('Brodovi su se nalazili na ovim pozicijama! Više sreće drugi put!')
    mreza[red_1 ][kol_1 ] = "X"
    mreza[red_1_2 ][kol_1_2 ] = "X"
    mreza[red_1_3 ][kol_1_3 ] = "X"
    mreza[red_2 ][kol_2 ] = "Y"
    mreza[red_2_2 ][kol_2_2 ] = "Y"
```

```
mreza[red_2_3][kol_2_3 ] = "Y"  
  
stampaj_mrezu(mreza)
```

Rjesenja zadataka

```
ime = input("Unesite vase ime: ")
pet_puta=(ime+" ")*5
print(pet_puta)
```

```
a=input("Unesite znak sa tastature: ")
b=ord(a)

print("Vrijednost znaka '" + a + "' u ASCII tabeli je " + str(b))
```

```
a=input("Unesite prvi broj: ")
b=input("Unesite drugi broj: ")
c=input("Unesite treci broj: ")
asredina=(int(a)+int(b)+int(c))/3
print("Aritmeticka sredina je: " + str(asredina))
```

```
prva_rijec = input("Unesite prvu rijec: ")
druga_rijec = input("Unesite drugu rijec: ")

print(prva_rijec + ' ' + druga_rijec)
```

dodati ascii tablu dodati binarne brojeve