



# Osnove programiranja u Pythonu

# Teme

- Uvod u programski jezik Python
- Varijable i tipovi podataka
- Upravljanje greškama u programskom kôdu
- Kontrola toka izvršavanja programskog kôda
- Funkcije u Pythonu



# Uvod u programske jezike Python

# Sadržaj

- Povijest programskog jezika Python
- Razvojno okruženje
- Početak programiranja u Pythonu
  - Tradicionalni „Hello World” program
  - Ključne riječi u Pythonu

# Povijest Pythona

- Razvijen krajem 80-ih prošlog stoljeća
  - Autor nizozemski programer Guido von Rossum
- Prvo put objavljen 1991. (verzija 0.9.0)
  - Tek 1994. objavljena verzija Python 1.0
- 2000. godine objavljena verzija Python 2.0
  - 2020. godine prestaje razvoj, unapređenje, popravak grešaka ... na verziji Python 2.0
  - I dalje dostupna i funkcionalna, ali preporuka je preći na verziju 3
- 2008. godine objavljena verzija Python 3.0

*Monty Python's*



Slika preuzeta s: <https://www.imdb.com/title/tt0063929/>

# Zen of Python

- 1999 Tim Peters na Python mailing listu za unapređenje Python programskog jezika, poslao skup od 19 načela kako napisati dobar softver.
- Python Enhancement Proposals (PEP ili PEP lista):
  - *PEP 20 – The Zen of Python*  
Petersova načela pod nazivom „Zen of Python” su, 2004. godine, uključena u službenu PEP listu po oznakom PEP 20  
(<https://www.python.org/dev/peps/pep-0020/>)
  - **PEP 8 – Style Guide for Python Code**  
(<https://www.python.org/dev/peps/pep-0008/>)

# Zen of Python

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one, — and preferably only one — obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than right now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea — let's do more of those!

# Tko sve koristi Python?

- Google tražilica, YouTube, Dropbox, Yahoo!, Walt Disney, Pixar, NASA, Red Hat, Nokia, IBM, Netflix, Yelp, Intel, Cisco, HP ...
- Igrice: Battlefield 2, Civilization IV, QuArK ...
- Python koriste za: sistemsko programiranje, web aplikacije, GUI aplikacije, igre, robotika, data science ...

# Distribucije Pythona

## Anaconda – Data Science

The screenshot shows the Anaconda Individual Edition website. At the top, there's a navigation bar with links for Products, Pricing, Solutions, Resources, Blog, and Company. A prominent "Get Started" button is located in the top right. Below the navigation, there's a section titled "Individual Edition" featuring a green icon of a person with a Q. The text "Your data science toolkit" is displayed, followed by a paragraph about the tool's popularity and features. A "Download" button is at the bottom.

## Samostalna distribucija

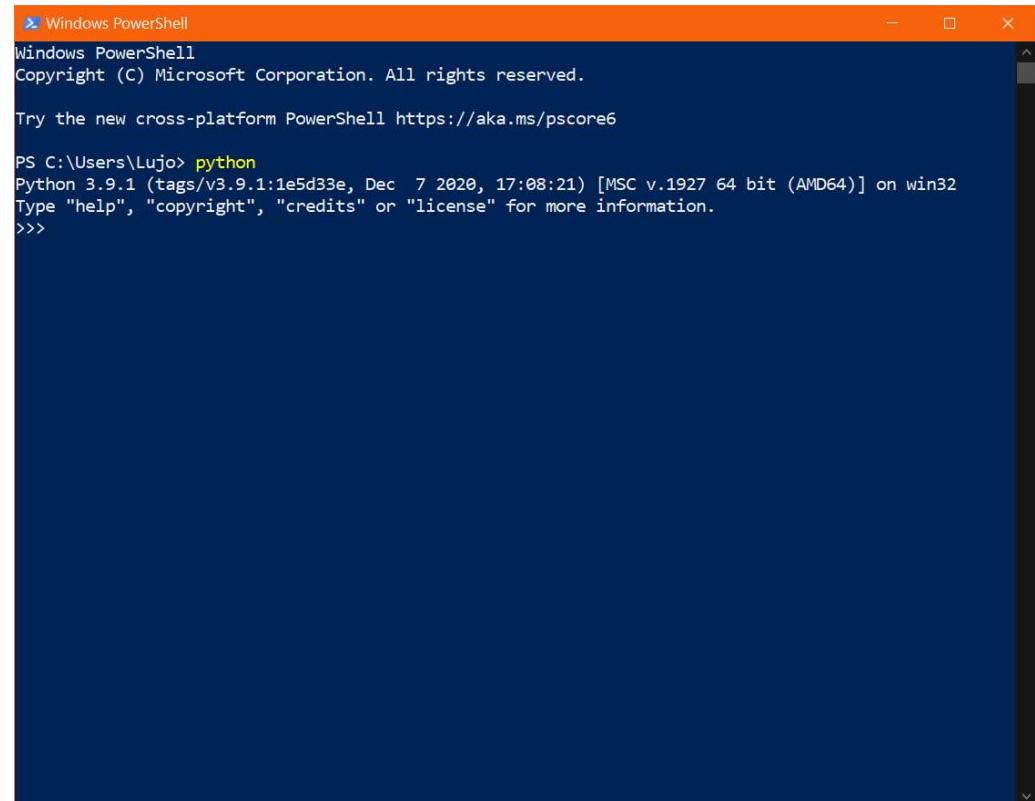
The screenshot shows the Python.org download page. At the top, there's a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. The Python logo is prominently displayed. Below the navigation, there's a section for "Download the latest version for Windows" with a "Download Python 3.9.1" button. There are also links for other operating systems and developer resources. To the right, there's a cartoon illustration of two boxes with yellow and white striped parachutes. Below this, there's a section titled "Active Python Releases" with a table showing the following data:

Python version	Maintenance status	First released	End of support	Release schedule
3.9	bugfix	2020-10-05	2025-10	PEP 596
3.8	bugfix	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537
3.6	security	2016-12-23	2021-12-23	PEP 494
2.7	end-of-life	2010-07-03	2020-01-01	PEP 373

At the bottom, there's a section for finding specific releases and a "Click for more" link.

# Python – Unutar PowerShell

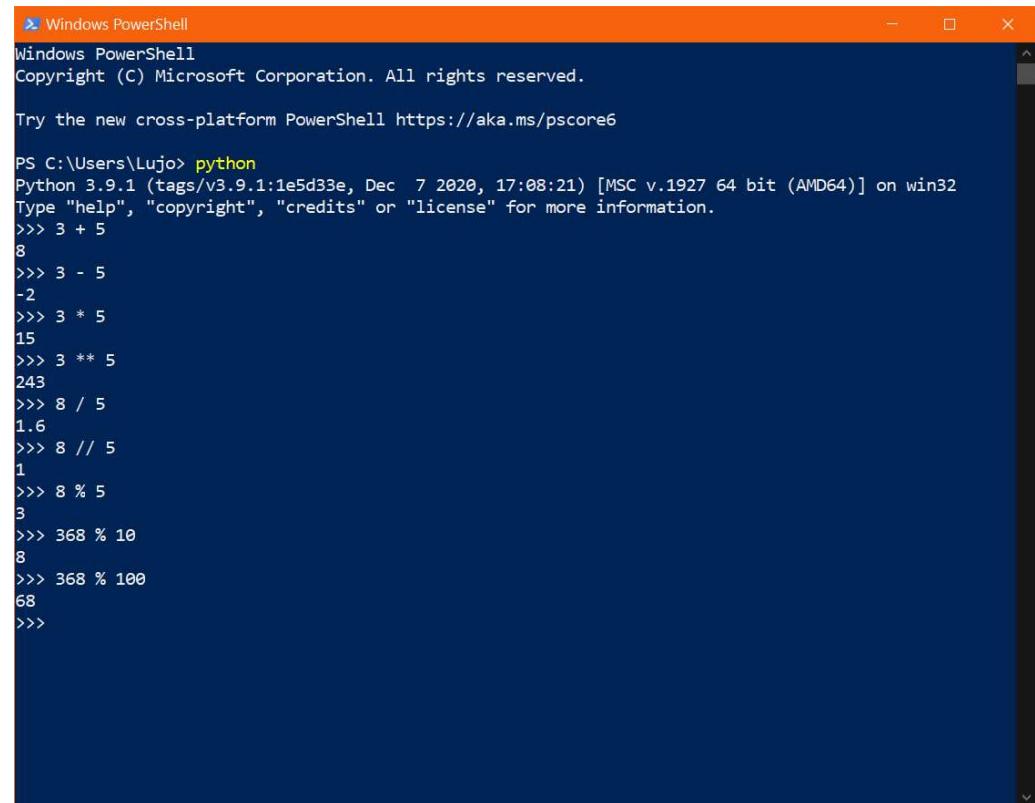
- Pokrenuti PowerShell
- Upisati python i pritisnuti ENTER
- Dobit ćete tri znaka >>> na početku svakog reda. Tako znate da ste u Python načinu rada.
- Pokušajte upisati:  
import this i pritisnuti enter
- Upišite help i pritisnite enter pa onda help(print)



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the Python interactive shell. The output includes the standard Python copyright notice, the Python version (3.9.1), and the command "PS C:\Users\Lujo> python". Below that, the Python prompt ">>>" is visible. The rest of the window is mostly blank, indicating that the user has not yet typed anything into the shell.

# Python – Unutar PowerShell

- Zbrajanje –  $3 + 5 = 8$
- Oduzimanje –  $3 - 5 = -2$
- Množenje –  $3 * 5 = 15$
- Potenciranje –  $3 ** 5 = 243$
- Dijeljenje –  $8 / 5 = 1.6$
- Cjelobrojno dijeljenje  
 $8 // 5 = 1$ .
- *Modulo* (ostatak dijeljenja)  
 $8 \% 5 = 3$



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lujo> python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> 3 + 5
8
>>> 3 - 5
-2
>>> 3 * 5
15
>>> 3 ** 5
243
>>> 8 / 5
1.6
>>> 8 // 5
1
>>> 8 % 5
3
>>> 368 % 10
8
>>> 368 % 100
68
>>>
```

# Modulo – ostatak od dijeljenja

- **$8 \% 5 = 3$**  zato jer u 8 imamo samo jednu peticu i ostalo nam je 3 od one druge petice.
- **$368 \% 10 = 8$**  zato jer u 368 imamo 36 desetica i ostalo je 8 od zadnje desetice.  
Idealno za dobiti zadnju znamenku nekog broja.
- **$368 \% 100 = 68$**  zato jer u 368 imamo 3 stotice i ostalo je 68 od one četvrte stotice.  
Idealno za dobiti zadnje dvije znamenke nekog broja

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lujo> python
```

in32

3	6	8	%	1	0	=	3	6
-	3	0						
		6	8					
		-	6	0				
					8			

# Aritmetički operatori – sažetak

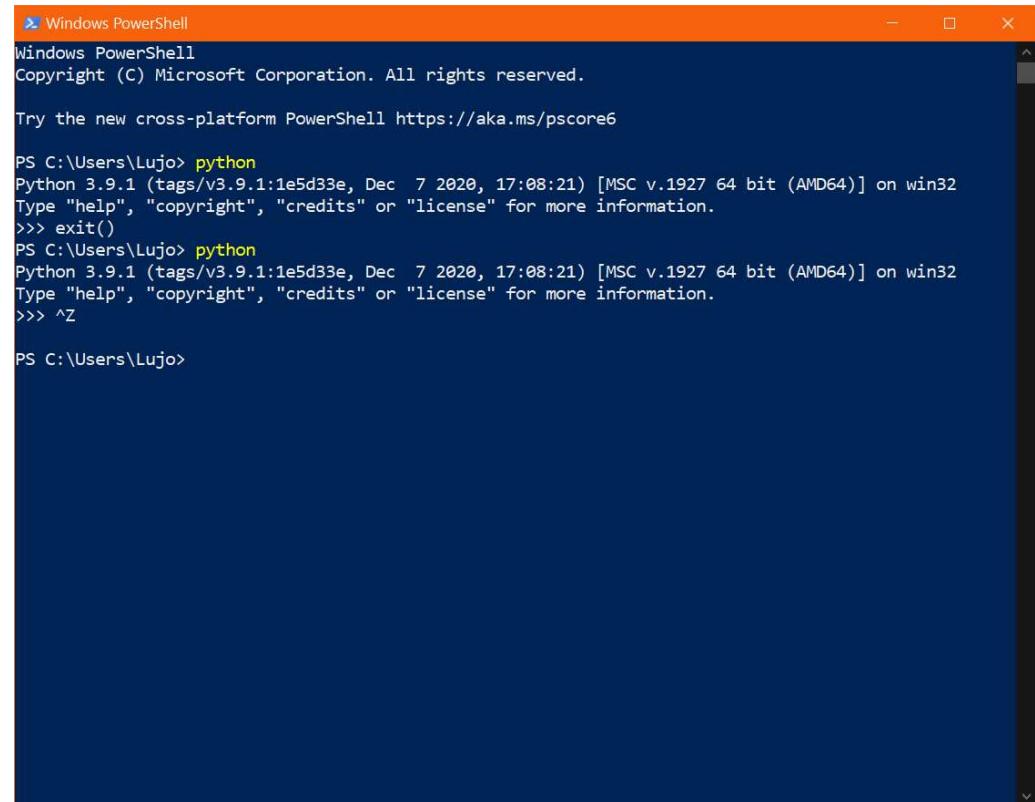
Operator	Namjena
+	Zbrajanje
-	Oduzimanje
*	Množenje
**	Potenciranje
/	Dijeljenje
//	Cjelobrojno dijeljenje
%	Modulo ostatak dijeljenja

# Aritmetički operatori – skraćeni oblik

Operator	Primjer	Osnovni oblik
<code>+=</code>	<code>a += b</code>	$a = a + b$
<code>-=</code>	<code>a -= b</code>	$a = a - b$
<code>*=</code>	<code>a *= b</code>	$a = a * b$
<code>**=</code>	<code>a **= b</code>	$a = a ** b$
<code>/=</code>	<code>a /= b</code>	$a = a / b$
<code>//=</code>	<code>a //= b</code>	$a = a // b$
<code>%=</code>	<code>a %= b</code>	$a = a \% b$

# Python – Izlaz iz PowerShell

- Izlazak iz Python command line načina rada:
  - `exit()`
  - `Ctrl + Z`
- Ili jednostavno zatvorite prozor PowerShella



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window shows the following text:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

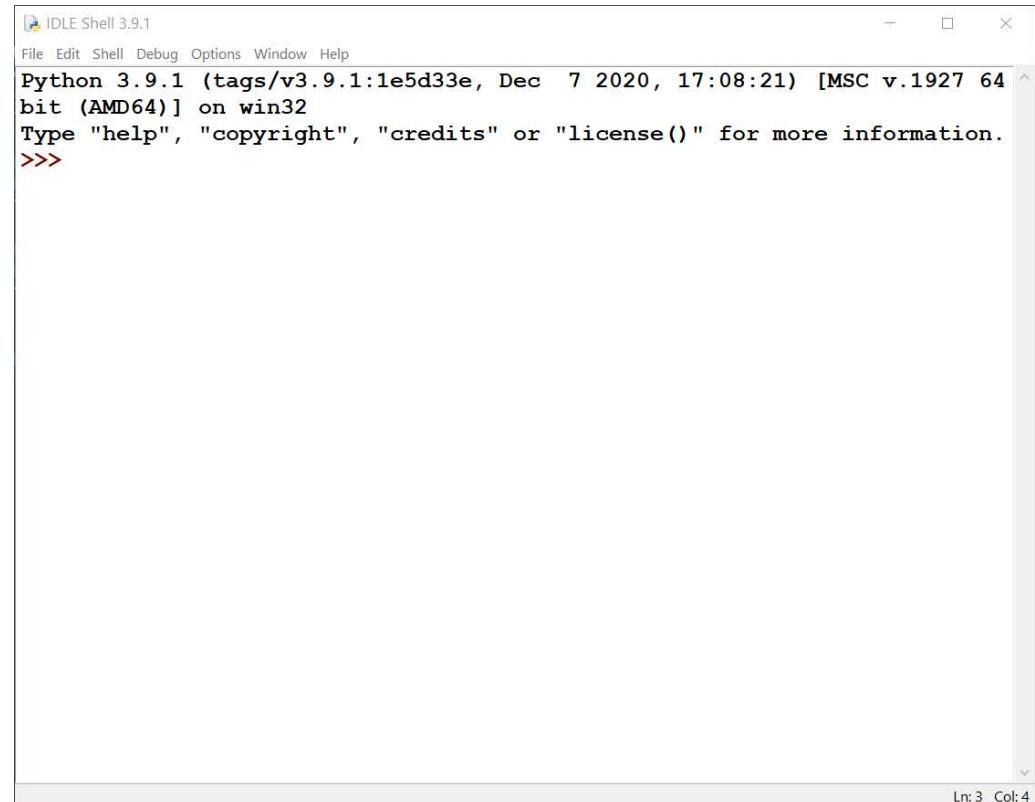
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lujo> python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
PS C:\Users\Lujo> python
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ^Z

PS C:\Users\Lujo>
```

# IDLE Python

- IDE – Integrated Development Environment
- IDLE – Integrated Development Environment for Python
- Start Menu – IDLE
- Nešto naprednije sučelje za pisanje programskog kôda od PowerShell
- Dokumentacija:  
<https://docs.python.org/3/library/idle.html>



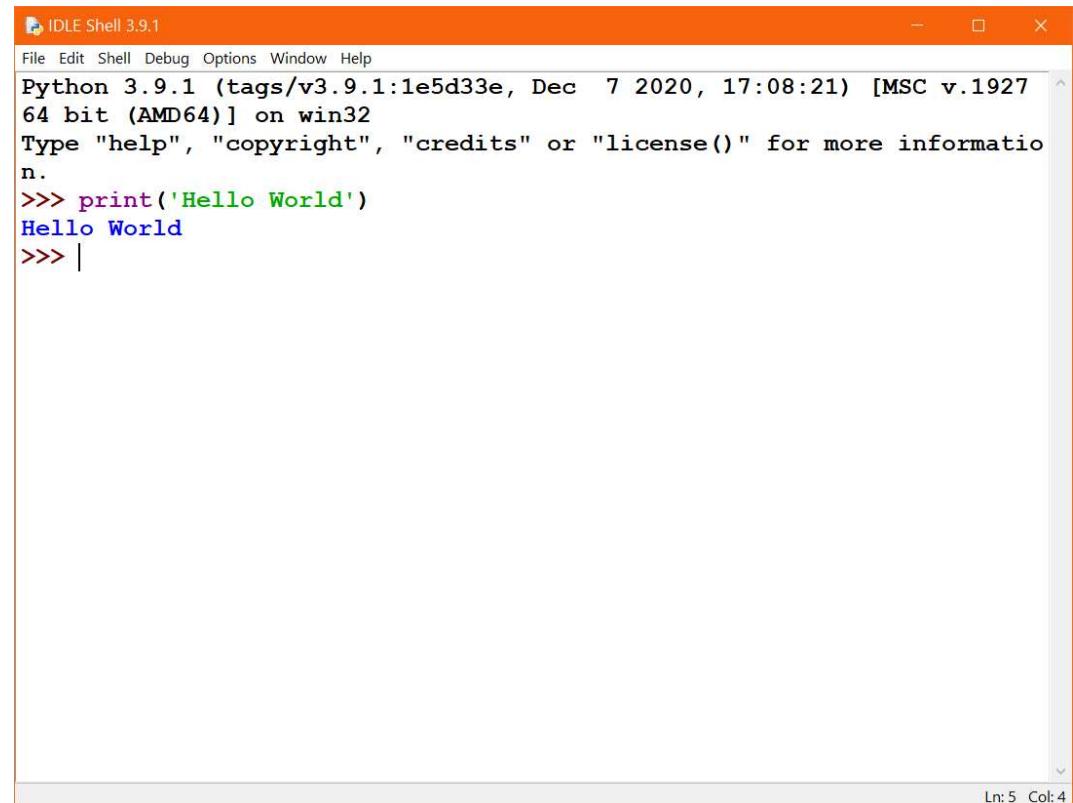
The screenshot shows the IDLE Shell 3.9.1 window. The title bar reads "IDLE Shell 3.9.1". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python 3.9.1 welcome message:

```
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

In the bottom right corner of the window, there is a status bar with "Ln: 3 Col: 4".

# “Hello World”

- Tradicionalno, prilikom učenja novog programskog jezika, prvi program ispisuje poruku „Hello World” na ekran.



The screenshot shows the IDLE Shell 3.9.1 interface. The title bar reads "IDLE Shell 3.9.1". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. Below the menu, it says "Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32". It also says "Type "help", "copyright", "credits" or "license()" for more information." A command line input shows ">>> print('Hello World')", followed by the output "Hello World", and then another prompt ">>> |". In the bottom right corner, there is a status bar with "Ln: 5 Col: 4".

# „Hello World” – primjeri

## C++

```
#include <iostream>
using namespace std;

int main(){
    cout << "Hello World" << endl;
    return 0;
}
```

## Java

```
public class HelloWorld{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

# Integrated Development Environment

- Integrated Development Environment – IDE
- Integrirani set alata nužnih za programiranje u nekom programskom jeziku.
  - Text editor za pisanje kôda
  - Kontrola verzija kôda
  - Debugging – pronalazak i otklanjanje grešaka
  - Compiler – prevođenje izvornog kôda u izvršni kôd (izrada .exe datoteka)
  - Ovisno o namjeni aplikacije koja se razvija, IDE omogućava pokretanje aplikacije za potrebe testiranja funkcionalnosti
  - ...

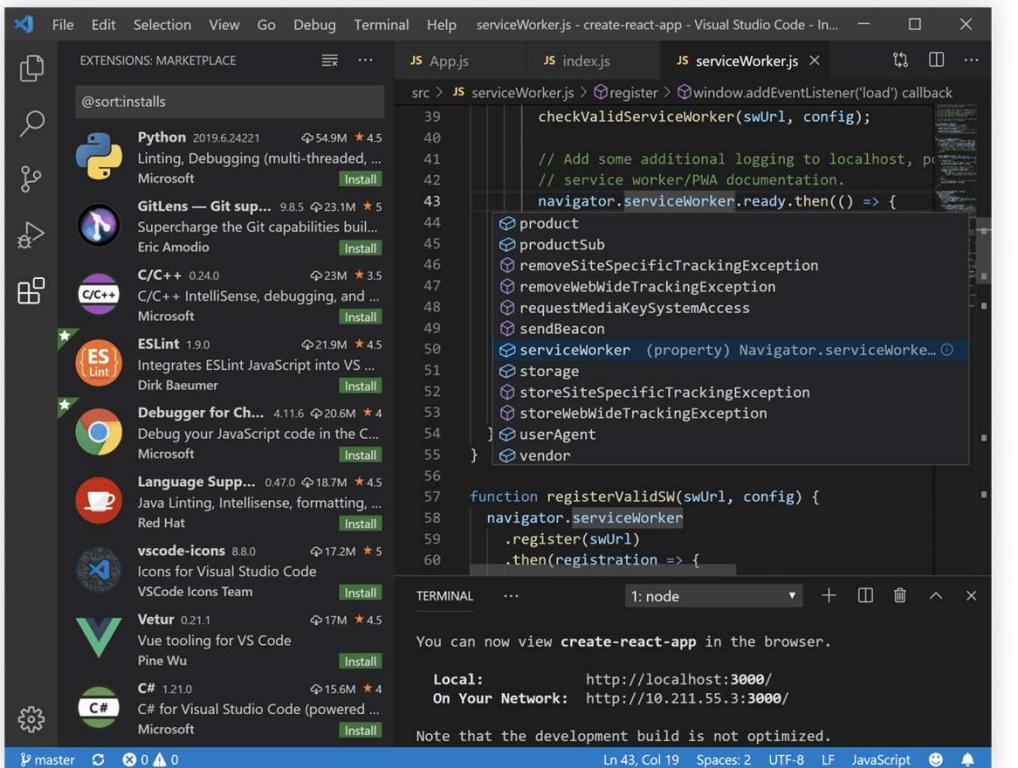
# Besplatni IDE za Python

- Atom – (<https://atom.io/>)
- PyCharm Community – (<https://www.jetbrains.com/pycharm/>)
- Sublime Text 3 – (<https://www.sublimetext.com/>)
- Visual Studio Code – (<https://code.visualstudio.com/>)
  - Mi ćemo koristiti Visual Studio Code – VS Code
  - Nije isto kao i Microsoft Visual Studio

IDE alati su poredani abecednim redom

# Visual Studio Code – VS Code

- Microsoft Visual Studio Code
- Dostupan na:  
<https://code.visualstudio.com/>
- Dostupna verzija za sve verzije platformi
- Zahvaljujući ekstenzijama lako ga je prenadjenu za gotovo bilo koji programski jezik
- Jako popularan za Python i JavaScript programske jezike



The screenshot shows the Visual Studio Code interface. On the left, the Extensions Marketplace is open, displaying a list of extensions such as Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, Vetur, and C#. On the right, the main code editor window shows a file named serviceWorker.js with code related to service workers. The status bar at the bottom indicates the file is named 'create-react-app' and provides local and network URLs for previewing the application.

```
src > JS serviceWorker.js > register > window.addEventListener('load') callback
    checkValidServiceWorker(swUrl, config);
    // Add some additional logging to localhost, p...
    // service worker/PWA documentation.
    navigator.serviceWorker.ready.then(() => {
        product
        productSub
        removeSiteSpecificTrackingException
        removeWebWideTrackingException
        requestMediaKeySystemAccess
        sendBeacon
        serviceWorker (property) Navigator.serviceWor...
        storage
        storeSiteSpecificTrackingException
        storeWebWideTrackingException
    } userAgent
    vendor

function registerValidSW(swUrl, config) {
    navigator.serviceWorker
        .register(swUrl)
        .then(registration => {
            You can now view create-react-app in the browser.
            Local: http://localhost:3000/
            On Your Network: http://10.211.55.3:3000/
            Note that the development build is not optimized.
        })
    Ln 43, Col 19  Spaces: 2  UTF-8  LF  JavaScript
```

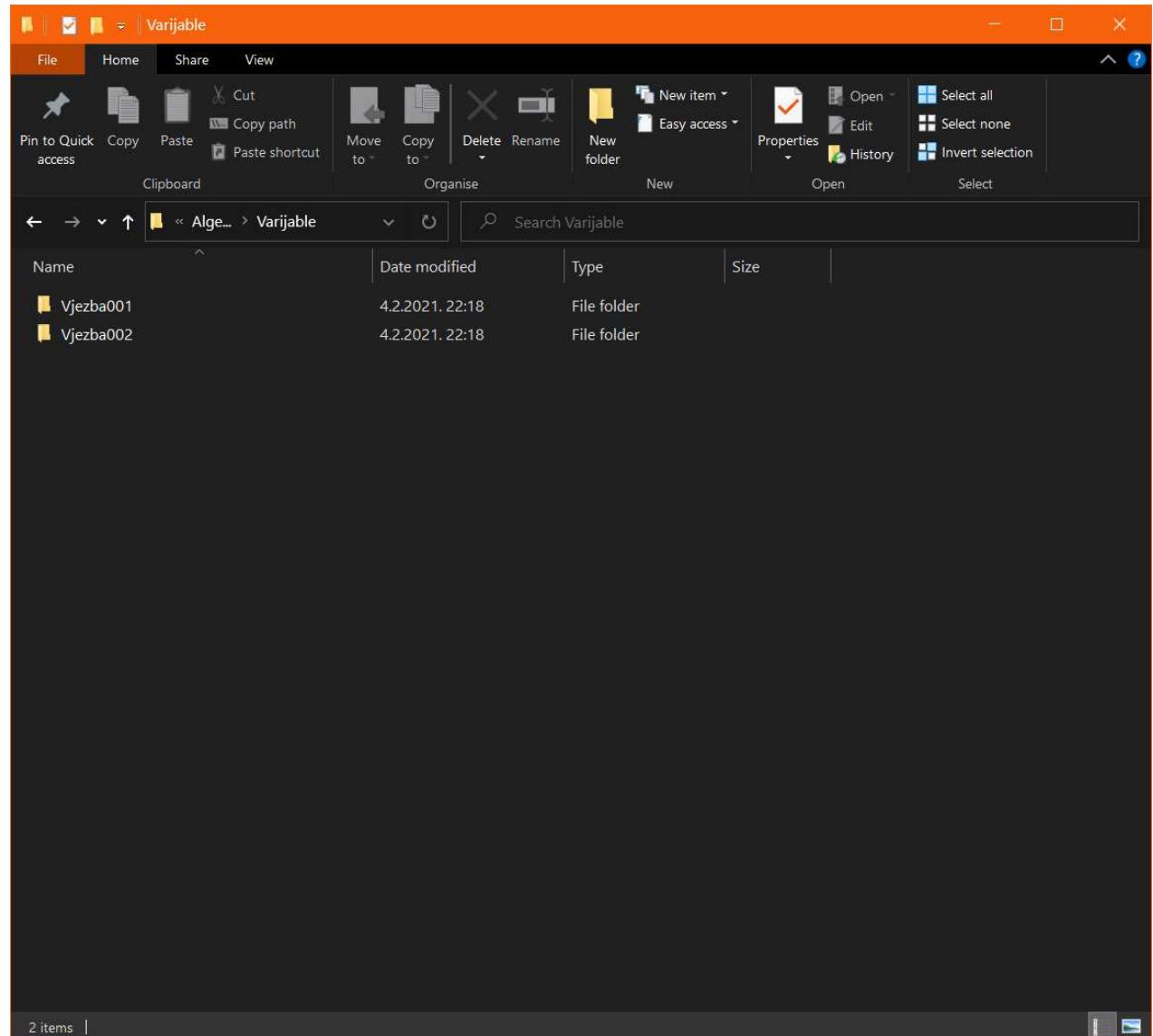
Visual Studio Code - Code Editing. Redefined

# Kreiranje projekta u VS Code

- Projekt u VS Code predstavlja vršna mapa u kojoj su pohranjene sve datoteke bitne za pokretanje programa (uključujući i fotografije i sl.)

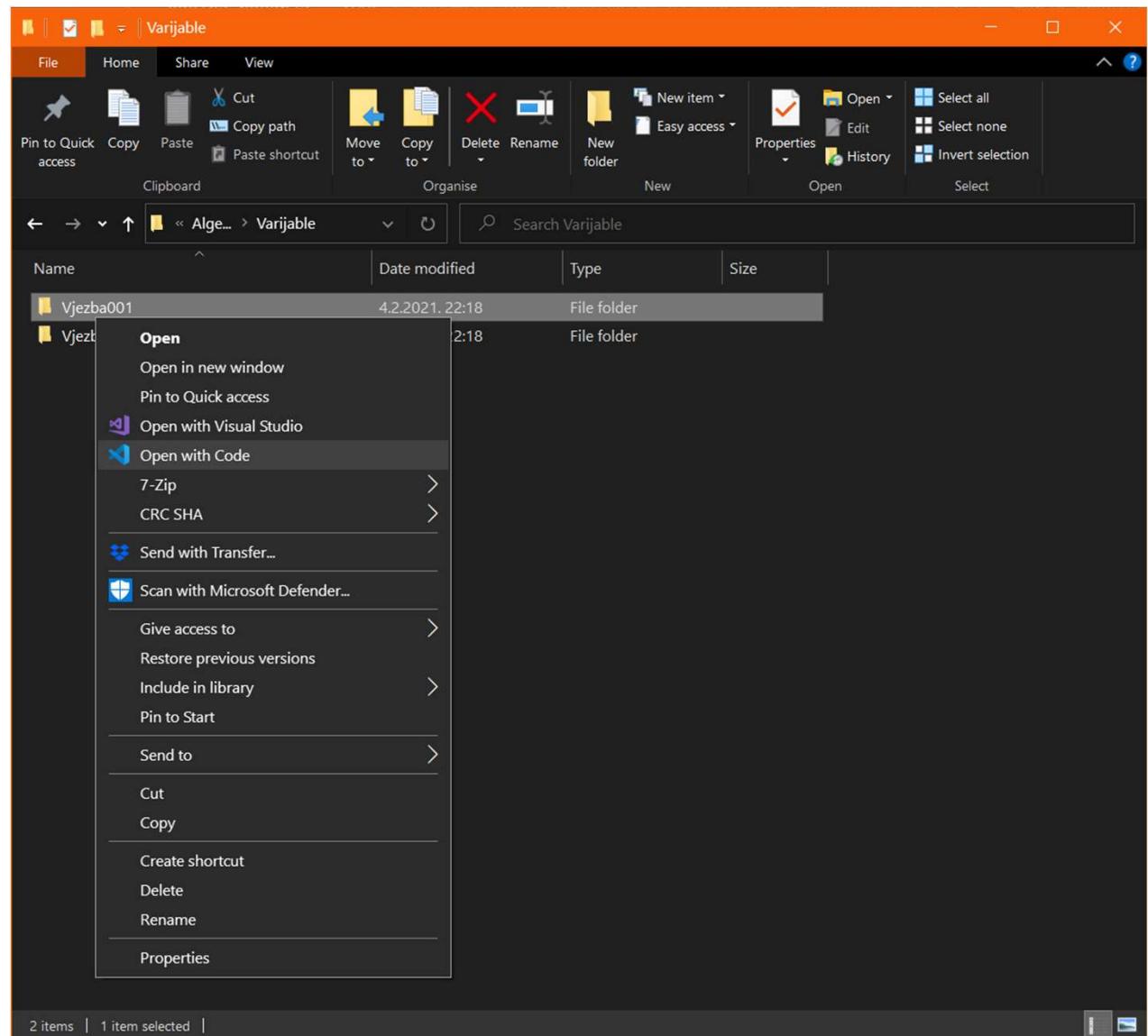
# Kreiranje projekta u VS Code

- Kreirajte vršnu mapu u kojoj će imati sve projekte vezane uz program.  
Primjer: D:\Algebra
- Unutar mape "Algebra" kreirajte jednu mapu koja će objedinjavati sve vježbe nekog poglavlja.  
Primjer: D:\Algebra\Variable
- I na kraju unutar mape poglavlja kreirajte mapu za svaku vježbu, zadatak ili primjer.  
Primjer:  
D:\Algebra\Variable\Vjezba001



# Kreiranje projekta u VS Code

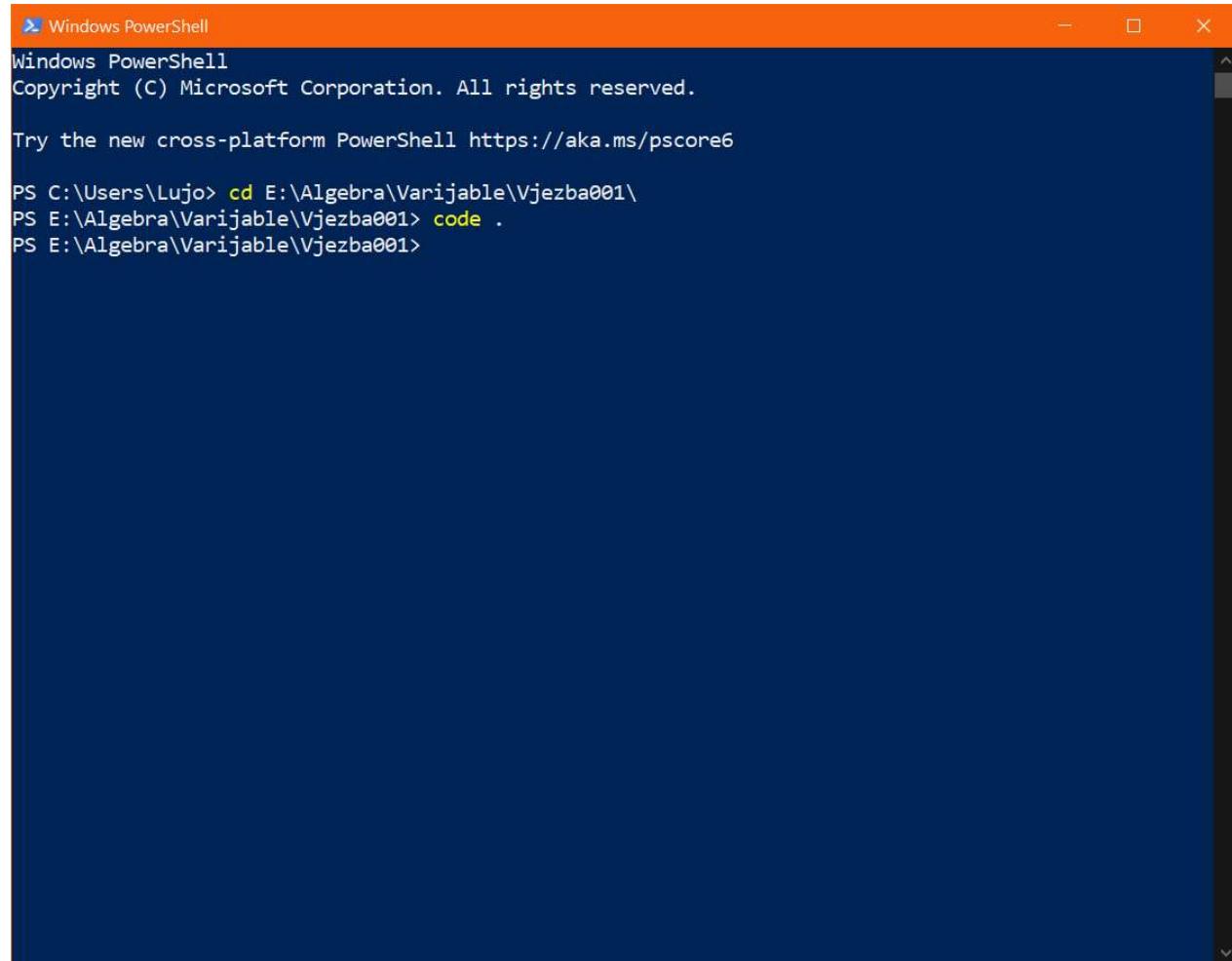
- Slijedeći korak je pokrenuti VS Code i dodijelitei
- VS Code možete pokrenuti tako da na mapu koju ste kreirali za čuvanje Python projekta, kliknute desnim gumbom i odaberete "*Open with Code*". Otvorit će se VS Code s već selektiranim folderom.



# Kreiranje projekta u VS Code

- VS Code možete pokrenuti tako da se unutar PowerShell konzole pozicionirate na mapu koju ste kreirali za čuvanje Python projekta te upišete "code ." (code razmak točka).

Otvorit će se VS Code na pozicioniranom folderu.



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window has an orange header bar with the title and standard window controls. The main area of the window is dark blue. It displays the following text:

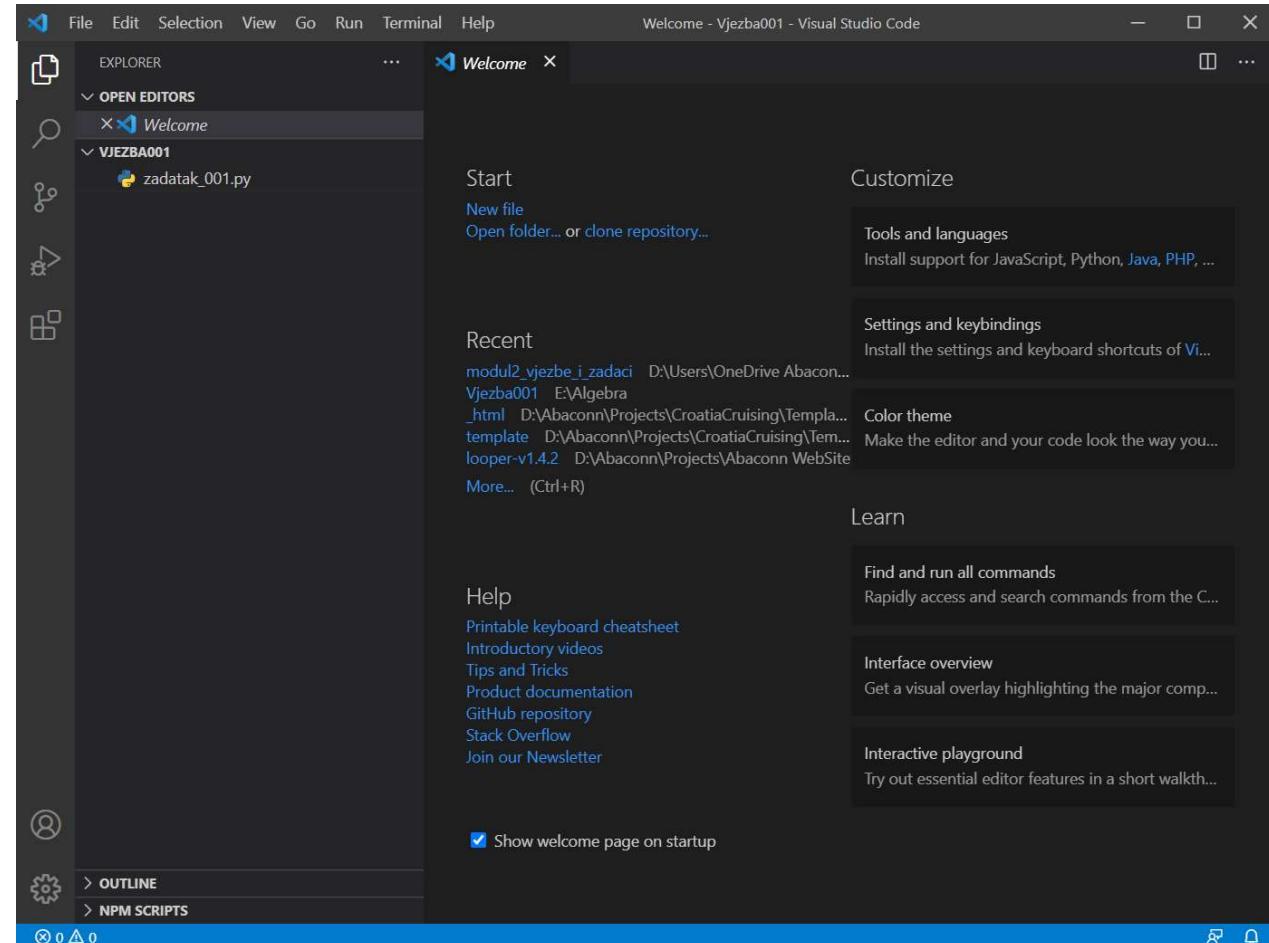
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lujo> cd E:\Algebra\Varijable\Vjezba001\
PS E:\Algebra\Varijable\Vjezba001> code .
PS E:\Algebra\Varijable\Vjezba001>
```

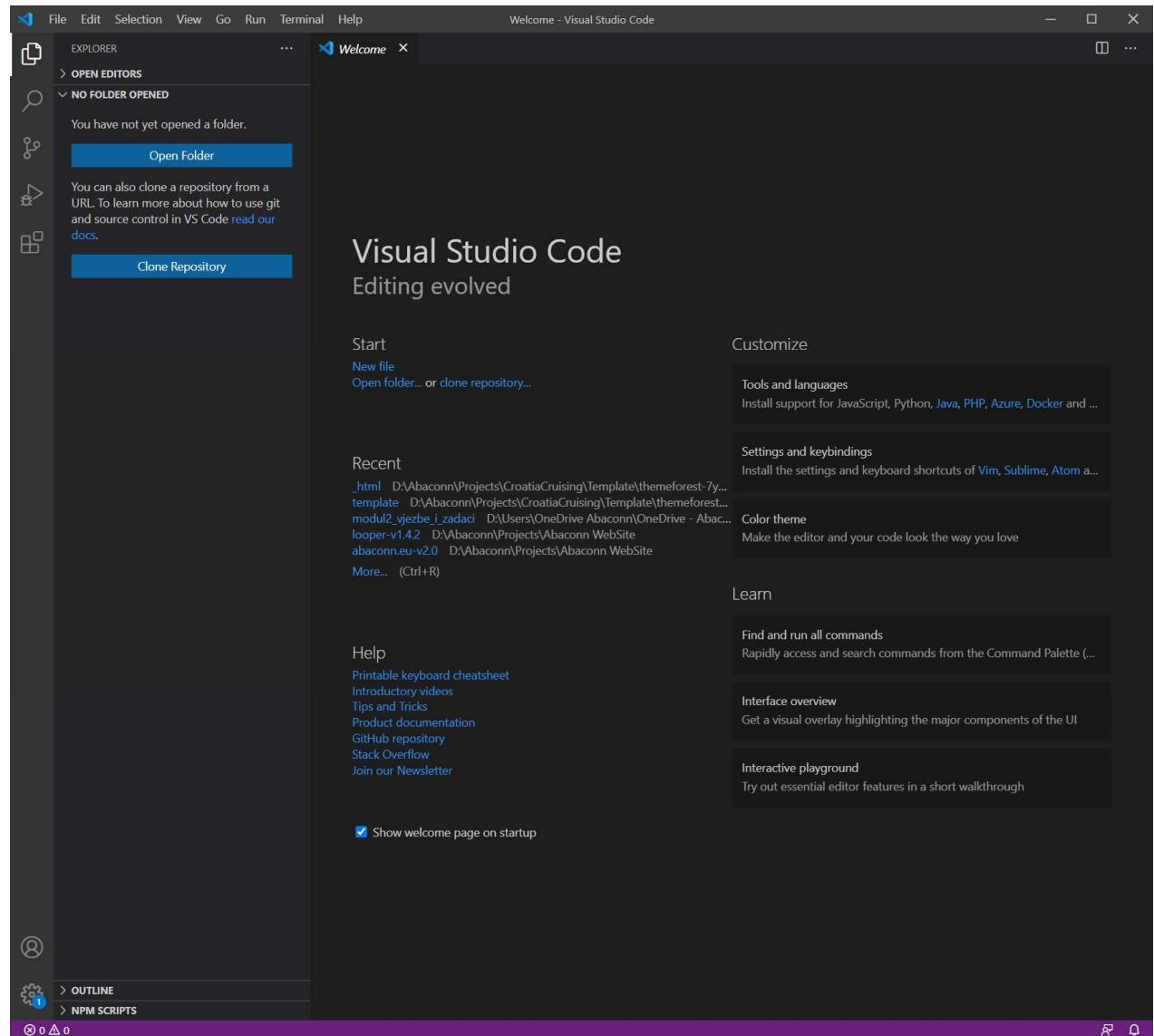
# Kreiranje projekta u VS Code

- Lijevi dio VS Code prozora bit će otvoren na alatu za rad s folderima i datotekama. Ako već imate datoteke unutar foldera one će biti prikazane.
- VS Code prepoznaje ekstenzije datoteka pa će za .py datoteke pridružiti ikonu Pythona.
- Desni dio VS Code prozora prikazuje Welcome ekran koji možete zatvoriti klikom na X na vrhu kartice tog prozora ili ga možete zanemariti.
- Ako već imate datoteke unutar foldera, dvostruki klik na datoteku otvorit će datoteku u novoj kartici na desnoj strani prozora.



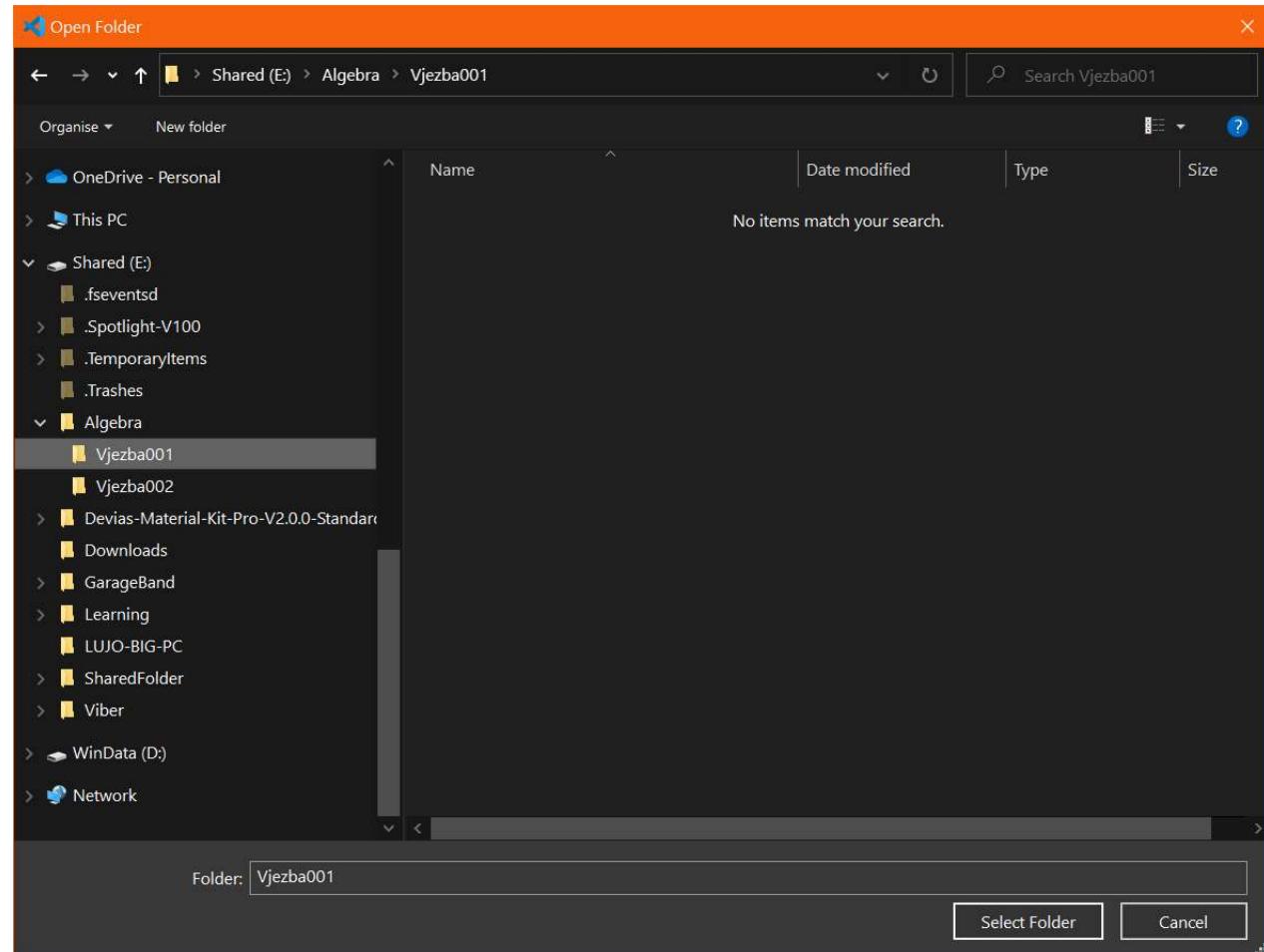
# Kreiranje projekta u VS Code

- Projekt unutar VS Code možete kreirati i tako da pokrenete VS Code te odete u alat za rad s folderima i datotekama (prvi stupac lijevo, prva ikona).
- Unutar drugog stupca lijevo, kliknuti na plavi gumb "Open Folder"



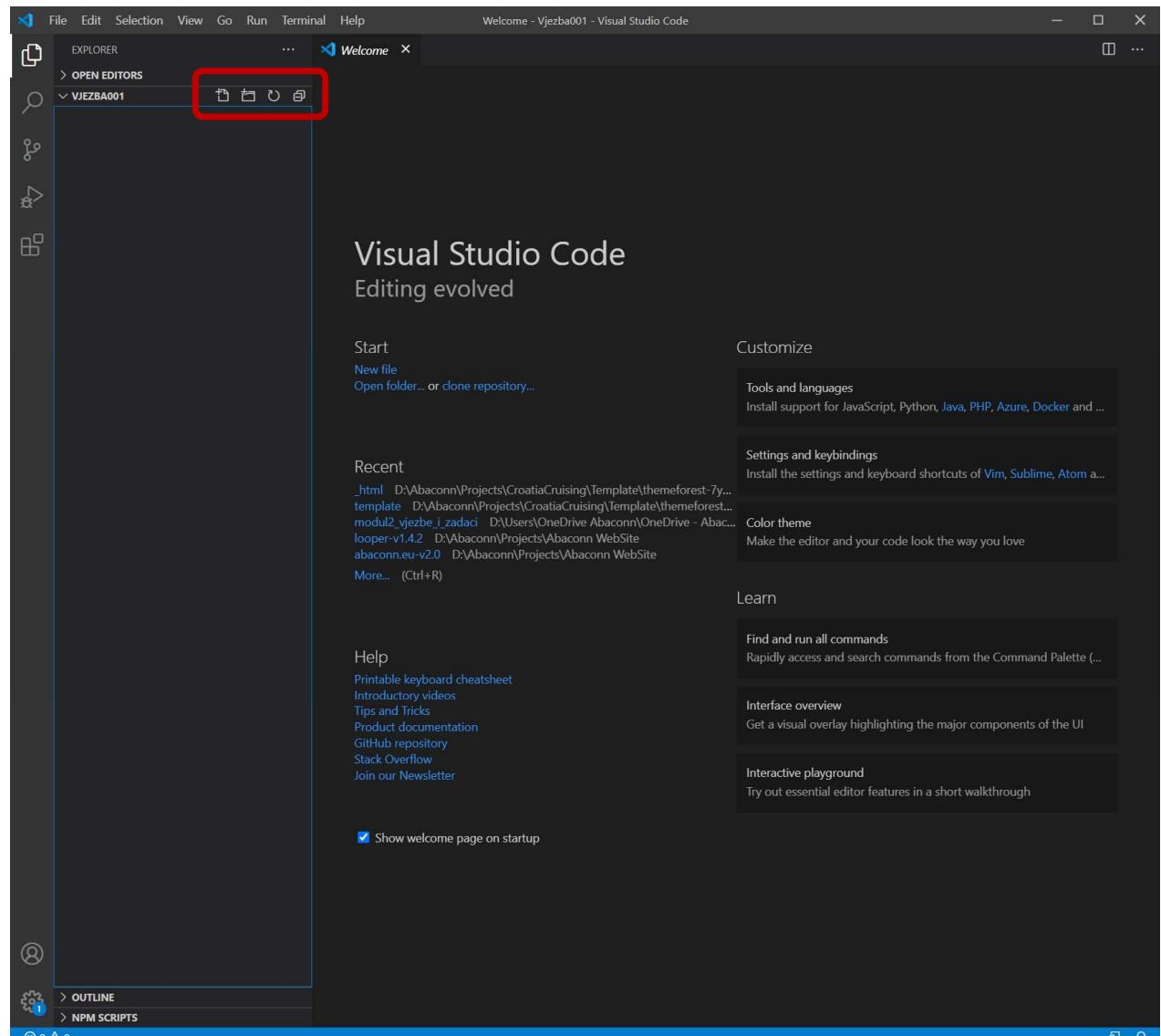
# Kreiranje projekta u VS Code

- Odabratи folder koji ћelimo da bude vršni folder našeg projekta.
- Kliknuti na gumb "Select Folder"
- Rezultat ћe biti isti kao i kod prethodna dva načina kreiranja projekta unutar VS Code IDE programa.



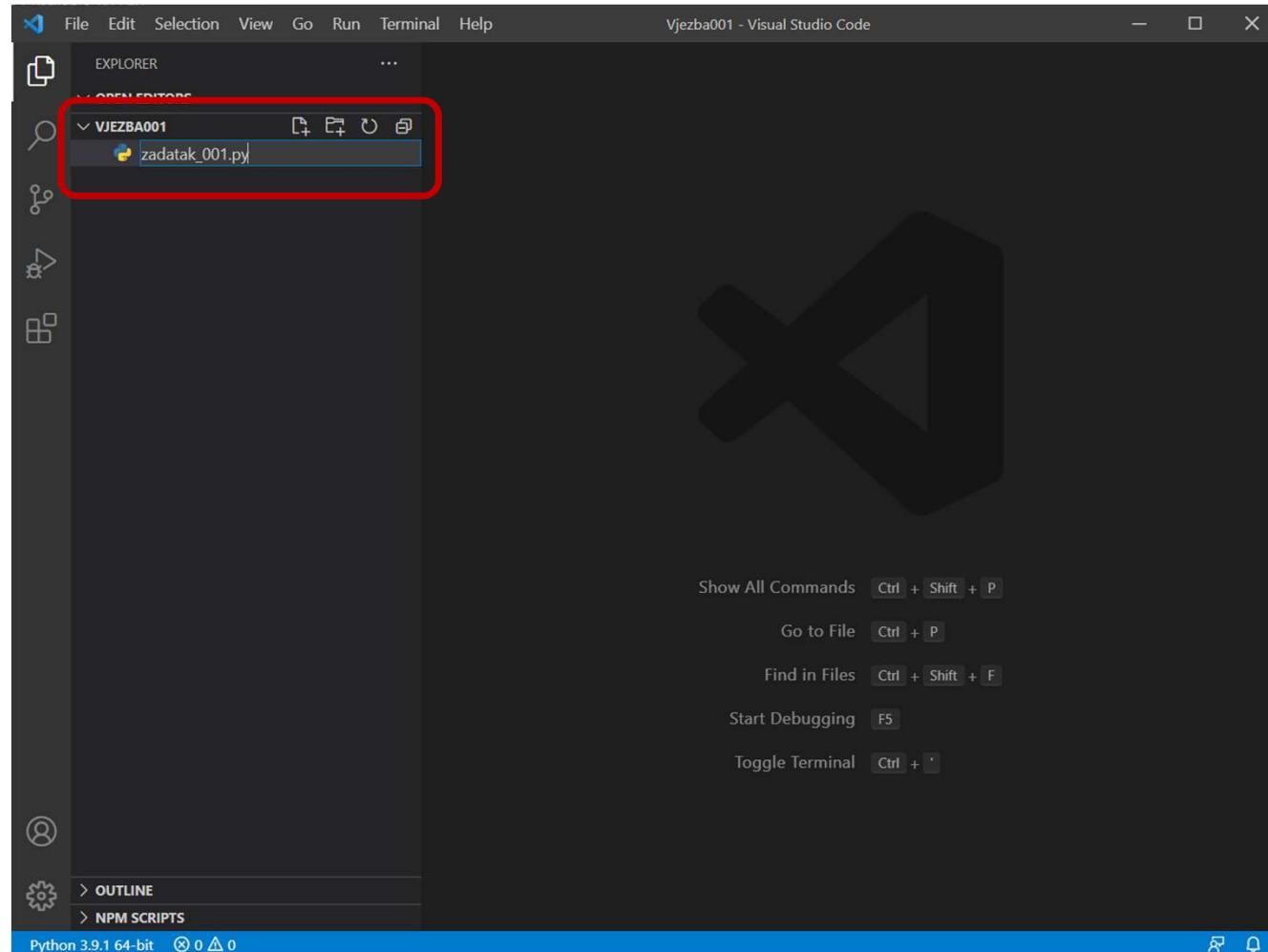
# Kreiranje datoteke

- Za pisanje programskog kôda potrebno je kreirati odgovarajuću datoteku koja će sadržavati naš programski kôd.
- Unutar crvenog pravokutnika prikazane su ikone koje NISU uvijek vidljive. Ako pomaknete strelicu miša unutar ovog drugog lijevo stupca te ikone će se pojaviti.
  - Prva ikona služi za kreiranje datoteke
  - Druga za kreiranje pod-folder-a (kasnije)
  - Treća služi za osvježavanje prikaza. Ako ne vidimo neku datoteku koja se nalazi u folderu pa želimo osvježiti prikaz



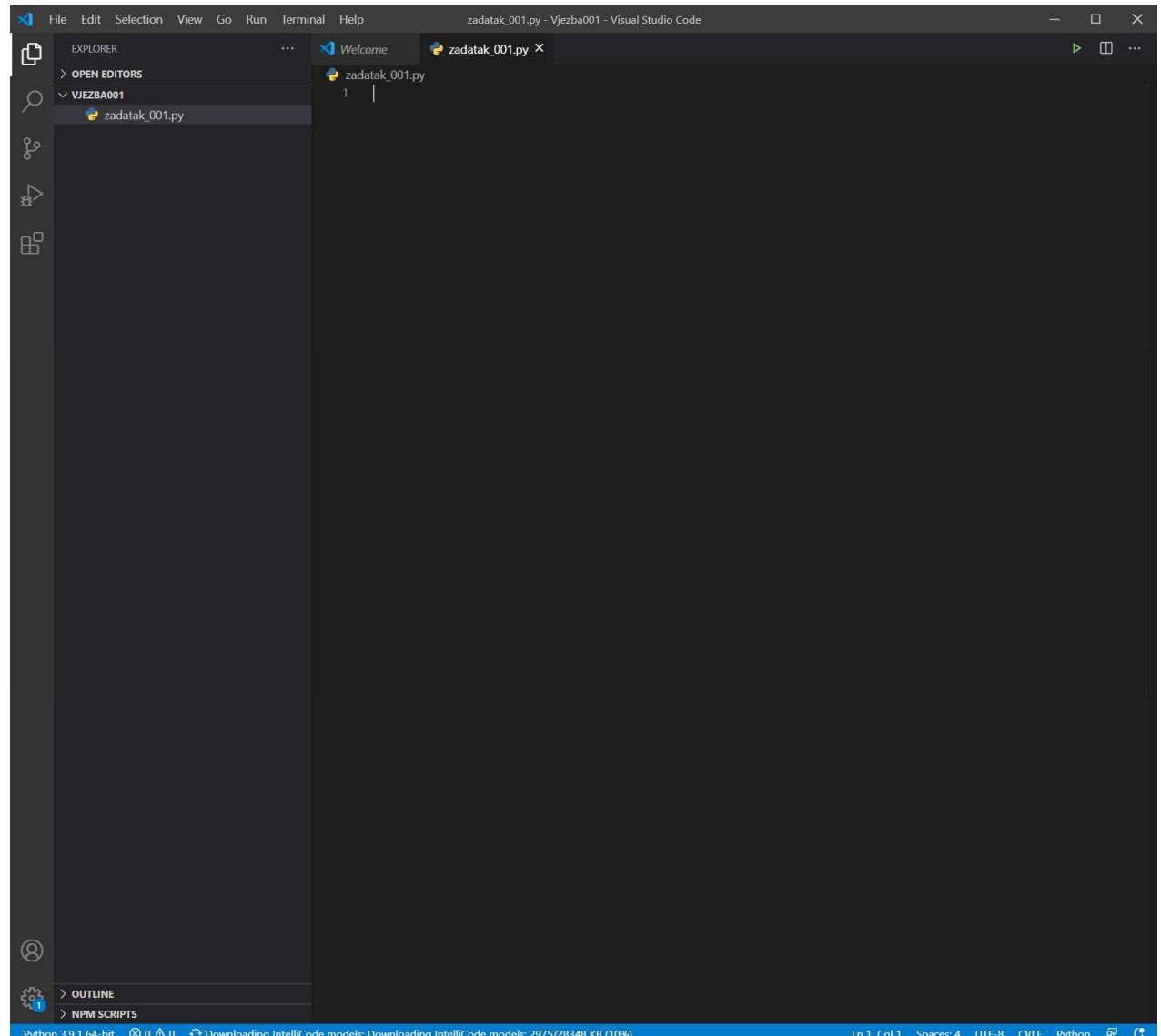
# Kreiranje datoteke

- Klikom na prvu ikonu, pojavit će se okvir za unos naziva datoteke.
- Upišite naziv datoteke i NE ZABORAVITE dodati ekstenziju **.py**
- Ako zaboravite dodati ekstenziju VS Code će datoteku tretirati kao običnu tekstualnu datoteku i program se neće moći pokretati.
- Ako ste pogriješili prilikom unosa, jednostavno desnim gumbom kliknite na naziv datoteke i odaberite "Rename" te ponovite unos.  
Preimenovanje datoteke možete napraviti i ako selektirate datoteku i pritisnete tipku F2 na vašoj tikovnici



# Kreiranje datoteke

- Nakon što ste unijeli naziv i ekstenziju datoteke te unos potvrdili ENTER tipkom, otvorit će se novi prozor, odnosno kartica sa sadržajem datoteke.
- Naša datoteka je prazna te je spremna za unos programskog kôda.
- Ovo možete ponoviti za svaku novu datoteku koju želite dodati unutar projekta. Jedan zadatak, neka bude jedna datoteka.



# Pokretanje programa

- U prozor datoteke upišite: `print('Hello World from Visual Studio Code!')`
- Program pokrenite kombinacijom tipki **CTRL+F5**
- Ili klikom na izbornik **Run -> Start Without Debugging**
- Rezultat je prikazan u donjem dijelu prozora unutar crvenog pravokutnika

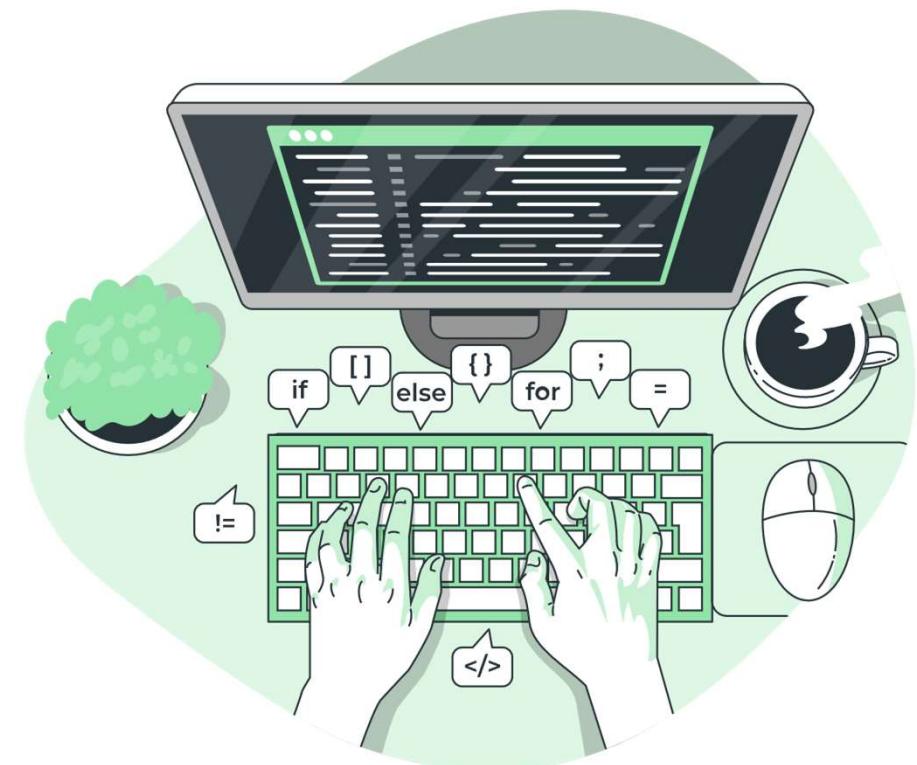
The screenshot shows the Visual Studio Code interface. In the top right, there's a tab for 'zadatak\_001.py - Vjezba001 - Visual Studio Code'. The left sidebar has sections for 'EXPLORER' (with 'OPEN EDITORS' showing 'zadatak\_001.py' and 'VJEZBA001' showing 'zadatak\_001.py'), 'OUTLINE', and 'NPM SCRIPTS'. The main area shows the code 'zadatak\_001.py' with the line `print('Hello World from Visual Studio Code!')`. Below the code editor is a terminal window titled '2: Python Debug Consc' with a red border around its content. The terminal output is:  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
Try the new cross-platform PowerShell <https://aka.ms/pscore6>  
PS E:\Algebra\Varijable\Vjezba001> & 'python' 'c:\Users\Lujo\.vscode\extensions\ms-python.python-2021.1.502429796\pythonFiles\lib\python\debugpy\launcher' '59718' '--' 'E:\Algebra\Varijable\Vjezba001\zadatak\_001.py'  
Hello World from Visual Studio Code!  
PS E:\Algebra\Varijable\Vjezba001>

# Varijable i tipovi podataka



# Dobro je ponoviti

- **Programiranje** (engl. Computer Programming) je pisanje instrukcija koje računalo „razumije“ kako bi slijedom tih instrukcija izvršilo željene aktivnosti.
- **Instrukcija** (engl. Instruction) predstavlja skup naredbi poredanih točno definiranim redoslijedom koje računalo „slijepo“ slijedi. Ako je programer pogriješio u redoslijedu, računalo nije inteligentno da to prepozna i prilagodi redoslijed. Kao što je rečenica skup riječi poredanih određenim redoslijedom.
- **Naredba** (engl. Statement) je najmanji dio programskog jezika koja predstavlja određenu radnju koju treba izvršiti.
  - Primjer: `print(); input(); if ... else; while ...`



# Variable u programiranju

- Svaki program treba imati mogućnost „zapamtiti“ jedan ili više podataka koji se koriste tijekom izvršavanja programa.
- Za pohranu tih podataka koriste se variable.
- Varijabla predstavlja NAZIV lokacije unutar memorije računala u koju je pohranjen određeni podatak.
- Svaku varijablu čine minimalno:
  - Naziv varijable
  - Tip podatka pohranjenog u varijabli
  - Vrijednost pohranjena u varijabli
- Variable mogu imati i konstantnu vrijednost (PI, g, c ...). Njih nazivamo konstantama.

Naziv varijable	Vrijednost varijable	Memorijska adresa
broj	15	0x0041F3
		0x0041F4
ime	Petar	0x0041F5
		0x0041F6
cijena	89.99	0x0041F7
		0x0041F8
valuta	kn	0x0041F9
		...

# Varijable u Pythonu – naziv varijable

- Svaka varijabla mora imati naziv
- Pravila imenovanja varijabli ovise o programskom jeziku
- Neka pravila vrijede za sve programske jezike.
- Naziv varijable:
  - NE smije biti isti kao neka od ključnih riječi
  - NE smije početi brojkom
  - NE smije imati razmak
  - NE smije imati posebne znakove: !, ?, @, #, \$, %, ...

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

# Varijable u Pythonu – naziv varijable

- Preporuke za imenovanje varijabli.
- Naziv varijable bi trebao:
  - Opisivati podatak koji je pohranjen u varijabli
  - Imati znak „\_“ (podvlaka ili underscore) umjesto razmaka.  
\*Opcionalno bez razmaka tako da svaka riječ počinje velikim slovom *camelCase*
  - Imati sve znakove napisane malim slovima
  - Bez (hrvatskih) dijakritičkih znakova
  - Zadržati dosljednost pa kako je varijabla dio programskog kôda, bolje je za naziv varijable koristiti engleski jezik, ali nikako NE miješati malo engleski, malo hrvatski ili neki drugi jezik.

Naziv varijable	Podatak koji čuva
\$x	'Patar Perić' – NE
ime_prezime	'Petar Perić' – DA
A	23,2° C – NE
unutarnja_temperatura	28,9° C – DA
vanska_temperatura	23,2° C – DA
polumjer kružnice	24.45 – NE
polumjerKruznice*	52 – DA

# Varijable u Pythonu – naziv varijable

- Nazivi varijabli u Pythonu su Case Sensitive. Python razlikuje velika i mala slova u nazivu varijable
- Ime ≠ ime ≠ iMe
- PEP 8 – Style Guide for Python Code  
(<https://www.python.org/dev/peps/pep-0008/#naming-conventions>)
- Dodatno: Google preporuke za Python pravila imenovanja:  
<https://google.github.io/styleguide/pyguide.html#316-naming>

# Varijable u Pythonu – Primjeri

## TOČNO

```
ime = 'Petar'  
 prezime = "Peric"  
 godina_rodenja = 1995  
 zaposlen = True  
 tezina = 86.7  
 spol = 'M'
```

## POGREŠNO

```
1broj = 6  
ime i prezime = 'Petar Pe  
rić'  
break = 'Gablec'  
naziv_racuna_$ = 'Devizna  
štедња u dolarima'
```

# Vježba – varijable

- Kreirajte varijable (imenujte ih i dodijelite im odgovarajuću vrijednost) za:
  - Ime, prezime, godinu rođenja, državu rođenja, status radnog odnosa, težinu te spol
  - Stranice a i b, četverokuta te za površinu tog četverokuta.

# Komentari u kôdu

- Dio programskog kôda koji se NE izvršava
- Namjena za kratke opise unutar kôda
- Komentari u jednoj liniji počinju znakom "#"
- Komentari koji se protežu na više linija počinju i završavaju trostrukim navodnicima  
""" Komentar ... """"

```
# Komentar unutar jedne linije koda  
# Komentar u drugoj liniji koda
```

....

Ovo je komentar koji može biti u jednoj liniji koda, ali i nastaviti se u drugoj ili trećoj liniji koda

....

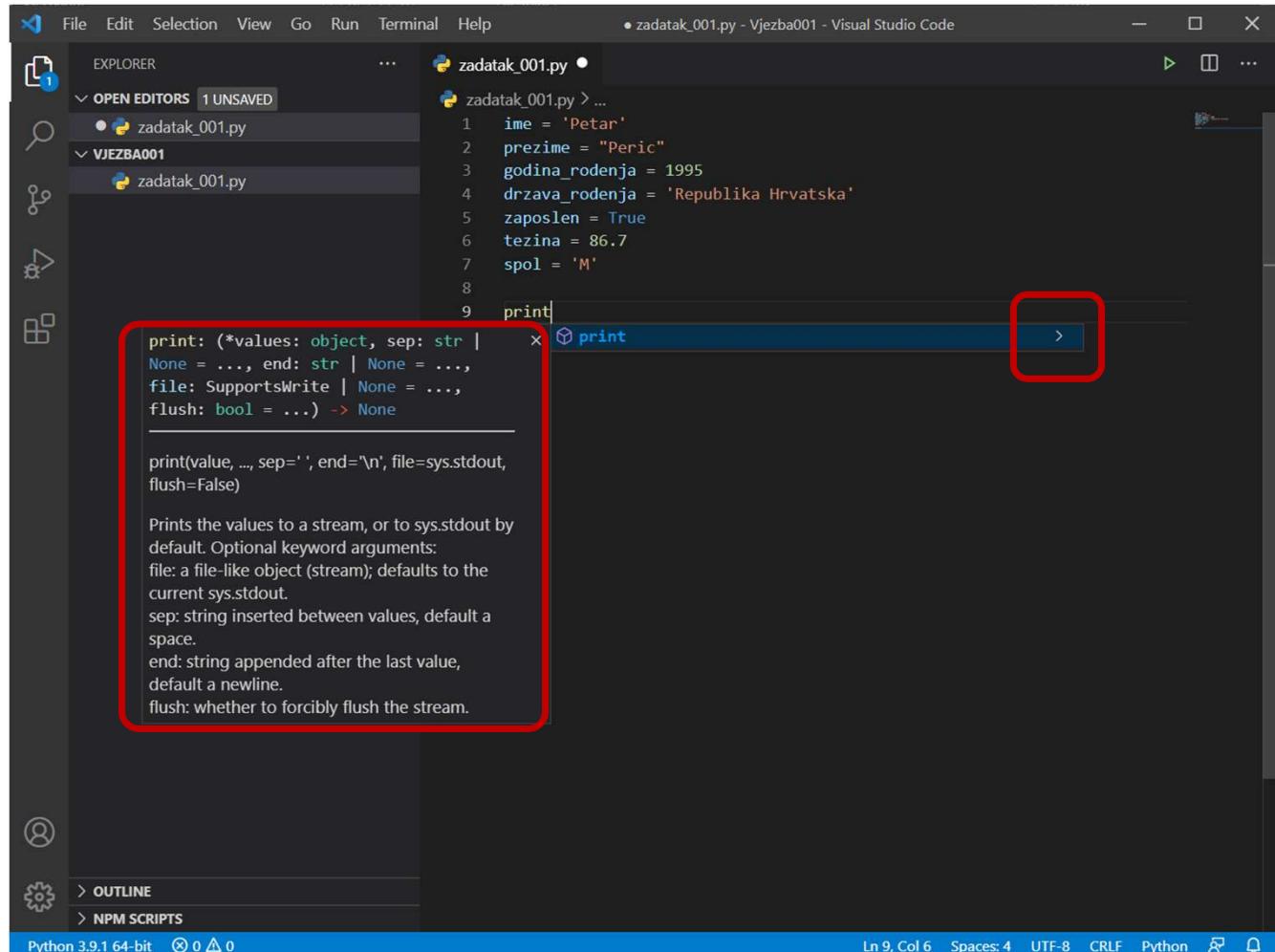
# Komentari u kôdu – preporuka

- NE pretjerivati s komentarima
  - Ako je nešto očito u kôdu, nema potrebe za dodavanjem komentara
- Pišite *razumljiv* kôd tako da nema potrebe za komentarima.
- Lakše je napisati malo duži naziv varijable nego dodati komentar koji će opisati koja je namjena varijable.
- Koristiti ih za izradu mjesta u kôdu koje treba još završiti – TODO komentari
- *PEP 8 -- Style Guide for Python Code* (<https://www.python.org/dev/peps/pep-0008/#comments>)
- "Docstring" ili "Documentation Strings" su komentari koji se koriste za izradu dokumentacije kôda. To su komentari koji počinju i završavaju s tri navodnika. *PEP 257 -- Docstring Conventions* (<https://www.python.org/dev/peps/pep-0257/>)
- Odlična knjiga o programiranju općenito: Robert C. "Uncle Bob" Martin. 2008 *Clean Code: A Handbook of Agile Software Craftsmanship*, Pearson

# Docstring help() *Intellisense*

Znak ">" kada kliknete otvorit će okvir s više detalja.

Ovaj okvir možete zatvoriti klikom na X u gornjem desnom dijelu okvira.



The screenshot shows a Visual Studio Code interface with a dark theme. In the center-right, there is an open Python file named 'zadatak\_001.py' containing code. A cursor is positioned at the end of the word 'print'. A tooltip window has appeared, containing the Python documentation for the 'print' function. The tooltip is highlighted with a red border. The documentation text is as follows:

```
print: (*values: object, sep: str | None = ..., end: str | None = ..., file: SupportsWrite | None = ..., flush: bool = ...) -> None

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

The status bar at the bottom of the code editor shows: Python 3.9.1 64-bit, 0 errors, 0 warnings, Line 9, Column 6, Spaces: 4, UTF-8, CRLF, Python. There is also a small logo for 'ALGEBRA' in the bottom left corner.

# Tipovi podataka u programiranju

- Već smo spomenuli da varijable trebaju sadržavati tri informacije:
  - Naziv varijable ✓
  - Tip podatka pohranjenih u varijabli
  - Vrijednost
- Tipovi podataka
  - Osnovni ili primitivni tipovi podataka
    - Brojevi – cijeli i decimalni brojevi
    - Znakovi (čak i znakovi za brojke)
    - Boolean – točno/netočno ili True/False ili 1/0
  - Kompleksni
    - Tekst ili String – kolekcija znakova. Ponekad se i *string* ubraja u primitivne tipove podatka
    - Kolekcije podataka – liste, rječnici ... Ovdje može biti kolekcija znakova, ali i stringova i bilo kojih drugih tipova podataka
    - Korisnički definirani tipovi podataka – klase
- Postoji još i podjela na ugrađene i korisnički definirane tipove podatka

# Ugrađeni tipovi podataka

- Primitivni tipovi podataka
  - Brojevi
    - Integer – Cijeli brojevi
    - Float – Decimalni brojevi
  - Znakovi (čak i brojke kao znak)
  - Boolean – točno ili netočno (True ili False)
- Složeni tipovi podataka
  - Tekst ili string – kolekcija znakova
  - Kolekcije podataka
    - lista, rječnik, n-terac ili tuple, set ...

Pixar filmovi			
Naslov (string)	Ocjena (float)	Broj pregleda (integer)	Favorit (Boolean)
Toy Story	9.78	159432	False
A Bug's Life	9.87	75365	False
Cars	9.79	45645	True
WALL-E	9.97	789654	True

# Ispis vrijednosti varijable – print()

- Vrijednost varijabli često trebamo prikazati korisniku kako bi korisnik imao koristi od našeg programa
- Za prikaz vrijednosti varijable na ekran koristi se naredba **print()**

```
print('Hello World!')
```

```
>>> help(print)
```

Help on built-in function print in module builtins:

```
print(...)
```

```
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

# Vježba – varijable i ispis na ekran

- Kreirajte varijable (imenujte ih i dodijelite im odgovarajuću vrijednost) te ispišite na ekran odgovarajuće vrijednosti, za:
  - Ime, prezime, godinu rođenja, državu rođenja, status radnog odnosa, težinu te spol
  - Stranice a i b, četverokuta te za površinu tog četverokuta.
  - Izračun mjesecne potrošnje el. struje te cijene el. struje koju potroši mikrovalna pećnica snage 1,3 kW ako se koristi 2 sata dnevno?
  - Stranice trokuta, površinu trokuta ( $P = \frac{a * v_a}{2}$ ,  $v_a$  je visina na stranicu a) te opseg trokuta.

# Vježba

- Kreirajte varijable (imenujte ih i dodijelite im odgovarajuću vrijednost) te ispišite na ekran odgovarajuće vrijednosti, za:
  - Ako automobil troši 5.3 litara na 100 km i ako je cijena goriva 9.56 kn po litri (nije važno kojeg goriva), izračunajte koliko košta 1 km vožnje automobilom. Prikažite mjesecni trošak (30 dana) odlaska na posao automobilom koji je udaljen 20 km u jednom smjeru.
  - Imate 10000 kn i možete zaboraviti na njih na 15 godina. Ako Vam banka nudi 2.5% godišnju kamatu za taj iznos, koliko ćete zaraditi nakon 15 godina. Jednostavni kamatni račun  $k = C * p * t$ 
    - $k$  = iznos kamata odnosno prinos
    - $C$  = iznos glavnice
    - $p$  = godišnja kamatna stopa – NAPOMENA:  $5\% = 5 / 100 = 0.05$
    - $t$  = vrijeme u godinama

# Unos vrijednosti od korisnika – `input()`

- Vrijednost varijabli često trebamo dobiti od korisnika našeg programa
- Za unos vrijednosti varijable od korisnika koristi se naredba **input()**

```
ime = input('Upišite ime')
```

```
>>> help(input)
```

Help on built-in function `input` in module `builtins`:

```
input(prompt=None, /)
```

Read a string from standard input. The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a trailing newline before reading input.

If the user hits EOF (\*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise `EOFError`.

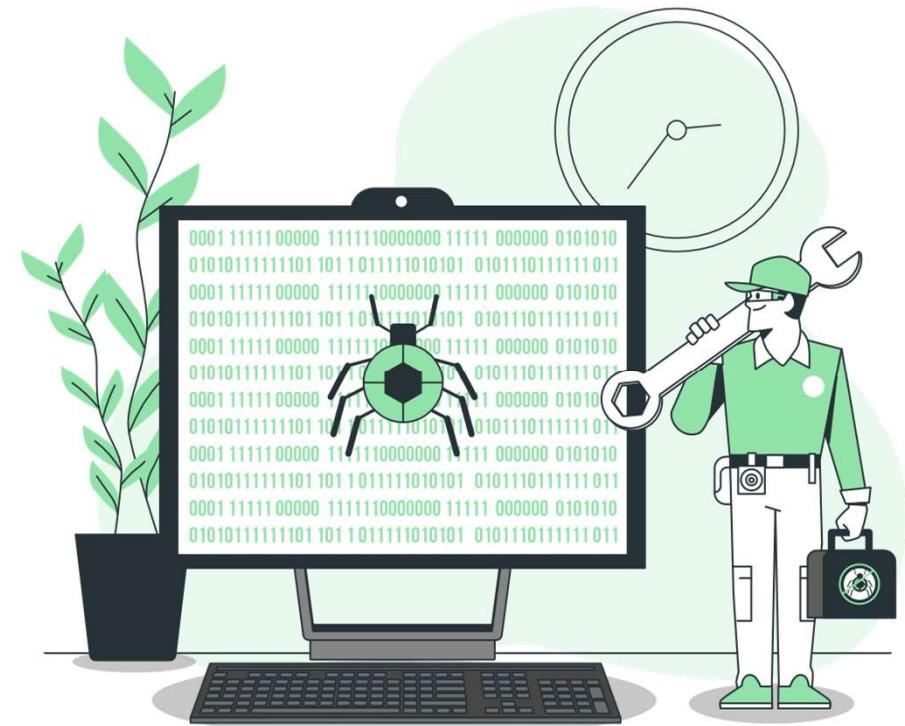
On \*nix systems, readline is used if available.

# Vježba – varijable, print() i input()

- Zatražite od korisnika unos dva broja.
  - Nakon unosa brojeva, ispišite: zbroj, razliku, umnožak, količnik (rezultat djeljna), potenciranje te modulo unesenih brojeva
  - Svaka operacija treba biti ispisana u novom redu, a ispis treba imati uključene brojeve, znak računske operacije te rezultat.
  - PRIMJER ISPISA:
  - $5 + 8 = 13$
  - $5 - 8 = -3$
  - NAPOMENA Za sada kod unosa neka kod prvog unosa drugi broj NE bude 0 (nula), jer nije dopušteno dijeliti s nulom. To svakako pokušajte napraviti, ali NE u prvom pokušaju.
- **PROBLEM!!! Dogodila se greška!**

# Greške u programskom kôdu

- Ljudi nisu savršeni.
- Kôd koji ljudi napišu nije savršen, ima grešaka.
- Kôd koji napišu roboti, koje su napravili ljudi, također ima grešaka



# BUG

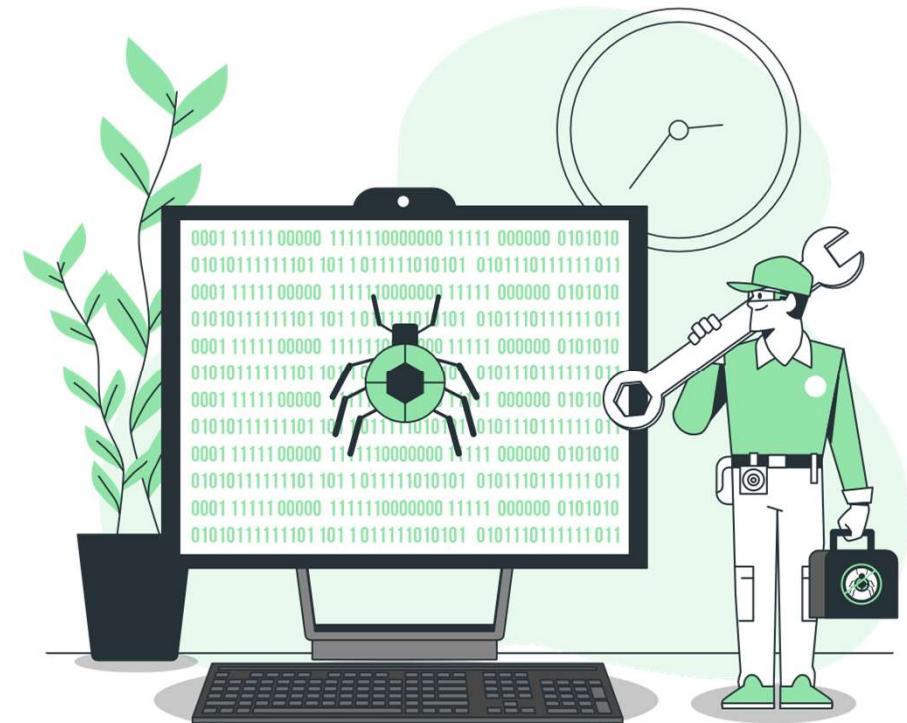
Znate li priču o bubi u računalu zbog koje je računalo prestalo raditi i zbog koje se kolokvijalno greške u kôdu nazivaju BUG.

9/9	
0.800	Antam started
1.000	" stopped - antam ✓
13:00 (033) MP-MC	<del>1.130476415 (-3)</del> 4.615925059(-2) 033 PRO 2 convct 2.130476415
	Relays 6-2 in 033 failed special sped test in relay " 11.000 test .
11:00	Started Cosine Tape (Sine check)
15:25	Started Multi Adder Test.
15:45	 Relay #70 Panel F (moth) in relay.
16:30	First actual case of bug being found.
17:00	Closed down.

[First Computer Bug, 1945 - Software bug - Wikipedia](#)

# Greške u kôdu – tipovi grešaka

- **Syntax Error** – greške u sintaksi. Krivo napisane naredbe, nazivi funkcija ... Najčešće program NE možete pokrenuti s ovom greškom, ali IDE točno ukazuje gdje se nalazi ovakva greška.
- **Runtime Error ili Exception Error** – greške koje nastaju prilikom izvođenja programa. Ponekad ih ne uočite odmah (primjer pokretanje zadnjeg zadatka). Malo ih je teže otkriti.
- **Bugs** – greške koje je najteže otkriti. To su greške koje predstavljaju krivo funkcioniranje programa. Primjer: uz cijenu se ne obračunava PDV, nego su sve cijene iste. Dakle, program funkcioniра, kôd je ispravno napisan, ali program ne radi ono što bi trebao.



# Konverzija tipova podataka

- Ponekad je nužno napraviti konverziju iz jednog tipa podatka u drugi tip
- Najčešće se konverzija radi iz teksta (string) u broj (integer ili float) ili obrnuto (Primjer kada trebamo vrijednost varijable zapisati u datoteku).
- Postoje i naredbe za konverziju brojeva između brojevnih sustava – binarni, oktalni, dekadski, heksadekadski

# Konverzije tipova podatka

- Integer – **int('string\_koji\_konvertiramo')** – cijeli broj
- Float – **float('string\_koji\_konvertiramo')** – decimalni broj,
- String – **str(broj\_ili\_objekt\_koji\_konvertiramo)** – pretvara u tekst,
- Boolean – **bool('True')** – string u boolean  
OPREZ!!! Ova funkcija će pretvoriti bilo koji string koji ima znakove u True, osim jedne iznimke. To je 'False'.
- Character – **chr(broj\_koji\_konvertiramo)** – pretvara u Unicode znak
- Unicode Character – **ord('string\_koji\_konvertiramo')** – pretvara u cijeli broj koji predstavlja Unicode znak

# Konverzija između brojevnih sustava

- Dodatne funkcije za konverziju između brojevnih sustava (NE tipova podataka)
- Iz dekadskog u binarni – **bin(cijeli\_broj\_koji\_konvertiramo)**
- Iz dekadskog u oktalni – **oct(cijeli\_broj\_koji\_konvertiramo)**
- Iz dekadskog u heksadekadski –  
**hex(cijeli\_broj\_koji\_konvertiramo)**
- Iz ... natrag u dekadski – **int('broj\_koji\_konvertiramo', baza)**
  - Broj koji konvertiramo treba biti tipa string pa zato imamo navodnike
  - Baza predstavlja bazu sustava iz kojeg konvertiramo. Binarni je 2, oktalni je 8, heksadekadski je 16

# Vježba – variable, print(), input(), konverzija

- Prepravite prethodne vježbe i zadatke tako da vrijednosti varijabli tražite od korisnika te napravite potrebnu konverziju.
  - Otvorite .py datoteke u kojima imate riješene zadatke te ih prepravite tako da koristite i naredbu `input()`. Zatim ih možete pohraniti pod novim imenom ili ostaviti isti naziv.
- IP adresa je adresa svakog računala na mreži koja se sastoji od četiri broja između 0 i 256. Primjer IP adrese: 192.168.0.184
  - IP adresu iz primjera ispišite u binarnom, oktalno i heksadekadskom obliku.
  - SAVJET: Za sada koristite zasebnu varijablu za svaki od četiri broja, odnosno dijela (okteta) IP adrese, ali ispišite ih u istom obliku kako je navedeno u primjeru (192.168.0.184).
  - Ispis treba napraviti za sve oblike brojevnih sustava.
- Na stranici <https://www.color-hex.com/color-palette/33532> imate boje Google logotipa. Pomoću odgovarajućih naredbi za konverziju pokušajte pretvoriti RGB zapise u HEX boja i obratno.
  - Primjer:
  - Naziv boje HEX zapis RGB (Red Green Blue)
  - CRVENA #EA4335 (234, 67, 53)
  - Za HEX zapis EA-43-35 trebate dobiti RGB zapis 234-67-53
  - NAPOMENA Zanemarite početni # znak u HEX zapisu sa stranice.
  - NAPOMENA HEX zapis čine tri grupe po dva znaka EA-43-35, svaka dva znaka čine jednu boju RGB

# Vježba – prepravite ove zadatke...

- ... tako da za unos vrijednosti pitate korisnika:
  - Ako automobil troši 5.3 litara na 100 km i ako je cijena goriva 9.56 kn po litri (nije važno kojeg goriva), izračunajte koliko košta 1 km vožnje automobilom. Prikažite mjesечni trošak (30 dana) odlaska na posao automobilom koji je udaljen 20 km u jednom smjeru.
  - Imate 10000 kn i možete zaboraviti na njih na 15 godina. Ako Vam banka nudi 2.5% godišnju kamatu za taj iznos, koliko ćete zaraditi nakon 15 godina. Jednostavni kamatni račun  $k = C * p * t$ 
    - $k$  = iznos kamata odnosno prinos
    - $C$  = iznos glavnice
    - $p$  = godišnja kamatna stopa – NAPOMENA:  $5\% = 5 / 100 = 0.05$
    - $t$  = vrijeme u godinama

# Tip podataka string ili TEKST

- STRING je tekstualni tip podataka
  - Ime i prezime, ulica, grad, država, poruka ...
- Vrijednost string varijable deklariramo pomoću navodnika
  - Svejedno je koje navodnike koristimo, ali uvijek moramo koristiti isti tip navodnika na početku i završetku teksta
- Za unos vrijednosti string podatka od korisnika koristimo input()
- NE možemo mijenjati string tip podatka. String je nepromjenjiv IMMUTABLE – Testirajte.
- Konverzija u drugi tip podataka i obrnuto (int(), float(), str() ...)

# Format stringa – format()

- Cilj pisati slično načinu pisanja rečenice
1. Napišimo rečenicu koju želimo imati kao tekst  
'Puno ime i prezime je: Petar Perić'
  2. Nakon zadnjeg navodnika stavimo točku i upišemo format()  
'Puno ime i prezime je: Petar Perić'.format()
  3. Na mjesto vrijednosti upisujemo {} koliko god nam treba, a unutar format() dodamo nazive varijabli  
'Puno ime i prezime je: {} {}'.format(ime, prezime)
  4. Sve to pohranimo u neku varijablu  
`puno_ime = 'Puno ime i prezime je: {} {}'.format(ime, prezime)`  
`print(puno_ime)`

# Format stringa – kraći format

- Kraći i potencijalno jasniji način formatiranja teksta
1. Umjesto format() na kraju navodnika, koristimo slovo f prije prvog navodnika  
`f'Puno ime i prezime je: Petar Perić'`
  2. Isto kao i kod prvog načina, gdje smo pisali { } zagrade pišemo ih i dalje, samo što sada zagrade ne ostavljamo prazne nego u njih upisujemo nazive varijabli  
`f'Puno ime i prezime je: {ime} {prezime}'`
  3. I na kraju sve to opet pohranimo u neku varijablu  
`puno_ime = f'Puno ime i prezime je: {ime} {prezime}'  
print(puno_ime)`

# Format stringa – "\" Escape Character

- "\" ili Backslesh ili Escape Character
- Omogućava dodavanje nekih posebnih znakova unutar teksta
- Nakon \ znaka se, bez razmaka dodaje znak koji želimo dodati
- Desno u tabeli NISU sve kombinacije već najčešće korištene
- Backslash neki prevode kao obrnuta kosa crta ... ili obrnuti kroz ( obrnuti "/"). Mi ćemo koristiti backslash

Kombinacija	Naziv	Namjena
\\"\\	Backslesh	Ispisuje JEDAN znak \ Prvi služi kao signal da dolazi poseban znak
\\'	Jednostruki navodnik	Ispisuje jedan jednostruku navodnik
\\""	Dvostruki navodnik	Ispisuje jedan dvostruki navodnik
\n	Novi red	Ispis teksta nastavlja u novom redu. Efekt isti kao i tipka ENTER
\t	Tab	Pomjeranje teksta u desno. Efekt isti kao tipka TAB

# Vježba – rad sa string-ovima

- Pomoću nekog od načina formatiranja teksta ispišite podatke o osobi i filmu iz prethodnih primjera.
- Ako string ne možemo mijenjati, kako onda možemo raditi zbrajanje, odnosno kombiniranje dva i više stringova u jedan?

# Kolekcije podataka

- Adresar
- Lista propuštenih poziva
- To-Do lista
- Diskografija albuma nekog glazbenika/glazbenice
- Popis stvari koje želim napraviti prije 60. rođendana
- TEMP folder na računalu
- ...



# Python kolekcije podataka

- Lista – lista[1, 2, 3, ‘Petar Perić’, True]
  - Lista koristi [ ] zagrade
  - Svaki element liste ima svoj jedinstveni indeks ili redni broj
  - Indeks brojevi počinju od 0 (nula)
- Dictionary / Rječnik – dict{1 : ‘Ban Josip Jelačić’, }
  - Rječnik koristi { } zagrade
  - Elementi su par ključ : vrijednost
  - Svaki element nema indeks, ali ima ključ. Zato ključ MORA biti jedinstven. Nema duplikata.
- Tuple / N-terac – tuple(1, 2, 3)
  - Tuple koristi ( ) obične zagrade
  - Tuple je NEPROMJENJIV. Kada kreirate Tuple, ne možete kasnije dodati ili maknuti element.
  - Svaki elementi N-terca ima svoj jedinstveni indeks ili redni broj
  - Indeks brojevi također počinju od 0 (nula)

# Python kolekcije podataka – lista

- Lista koristi [ ] zagrade
- Svaki elementi liste ima svoj jedinstveni indeks ili redni broj
- Indeks brojevi počinju od 0 (nula)
  - `lista = [154, 'tekst', 'jos jedan tekst', 3.14, True, 'Pa opet tekst']`
  - `prazna_lista = []`
- Pristup pojedinačnom elementu liste
  - `prvi_element = zadaci[0]`
  - `drugi_element = zadaci[1]`
  - `treci_element = zadaci[2]`
- String – lista znakova

# Grupiranje instrukcija u blokove

- Python za grupiranje kôda, umjesto { } zagrada, koristi uvlaku (tab) ili 4 razmaka (space).
- O ovome ne treba voditi brigu ukoliko koristite program za pisanje kôda jer se nakon : znaka za početak bloka, automatski novi red uvuče za potrebnii broj razmaka
- Ovo je jedan od načina kako Python želi povećati čitljivost programskog kôda. Kao paragraf.
- Lako se vizualno izdvajaju cjeline, odnosno blokovi kôda.



Primjer kôda:

```
for element in kolekcija:  
    instrukcije nad element
```

# FOR petlja

- Petlja koja neku instrukciju ponavlja određeni broj puta.
- Ključna riječ for
- FOR petlja se može čitati kao da računalu zadajemo zadatak:  
**„Za svako ime unutar imenika:**  
ispisi ime na ekran.,

**for element in kolekcija:**  
aktivnost nad *element*

*element* predstavlja varijablu koja se koristi samo dok se izvršava FOR petlja, nakon toga ta varijabla postaje nedostupna.

Naziv varijable *element* je proizvoljan.

Naziv kolekcije *kolekcija* je predefiniran, odnosno to je naziv varijable u koju smo prethodno pohranili neku kolekciju podataka (*elemenata*)

# Raspon brojeva – range()

- Ugrađena funkcija koja vraća listu brojeva unutar zadatog raspona
- `range(n)` – vraća brojeve u rasponu od 0 (nula) do n uz korak 1
- `range(n, m)` – vraća brojeve u rasponu od n (uključen) do m (m nije uključen) uz korak 1
- `range(n, m, k)` – vraća brojeve u rasponu od n (uključen) do m (m nije uključen) uz korak k

`range(5)` – 0, 1, 2, 3, 4, 5  
`range(5, 10)` – 5, 6, 7, 8, 9  
`range(5, 10, 2)` – 5, 7, 9

# Lista – osnovne naredbe

- *naziv\_liste.append(novi\_element)* – naredba za dodavanje novog elementa na kraj liste
- *naziv\_liste[indeks] = nova\_vrijednost* – naredba za izmjenu vrijednosti pohranjene u element liste na poziciji *indeks*
- *broj\_elmenata\_liste = len(naziv\_liste)* – len() naredba dohvaća broj elemenata pohranjenih u listi
- *Provjera je li tekst zaista lista znakova?*

# Slice – lista[START : STOP : STEP]

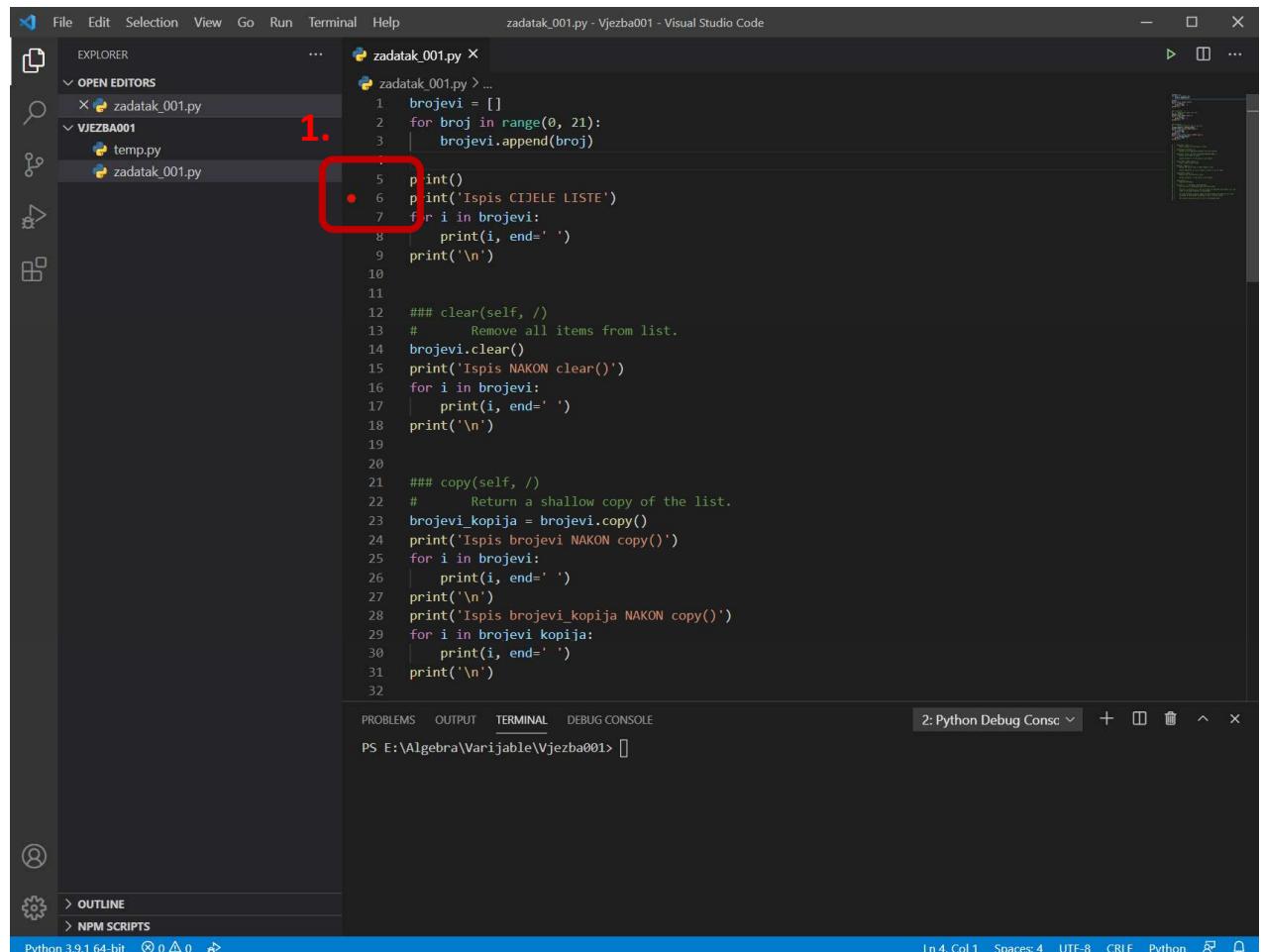
- Način kreiranja nove liste na osnovu manipulacije elementima prethodno kreirane liste. Primjer:
  - izdvojiti zadnja dva elementa liste
  - izdvojiti svaki treći element liste
- Po sintaksi jako slično range() naredbi.
- Primjeri na listi brojeva od 1 do 100.

# Lista – ostale naredbe

- *naziv\_liste.clear()* – naredba za brisanje svih elemenata liste
- *nova\_lista = naziv\_liste.copy()* – naredba za kopiranje liste
- **PROBLEM!!! .copy() NE kopira listu!**

# Debugging

1. Kliknite na stupac lijevo od linije kôda.  
U našem primjeru to je linija 6. Kada kliknete na taj stupac ispred te linije će se pojaviti točka.  
Ta točka se zove Break Point i to je točka u kojoj će se ZAUSTAVITI izvršavanje programa.
2. Pokrenite program u DEBUGGING načinu rada jednostavno tipkom F5 na tipkovnici ili izbornika Run -> Start Debugging



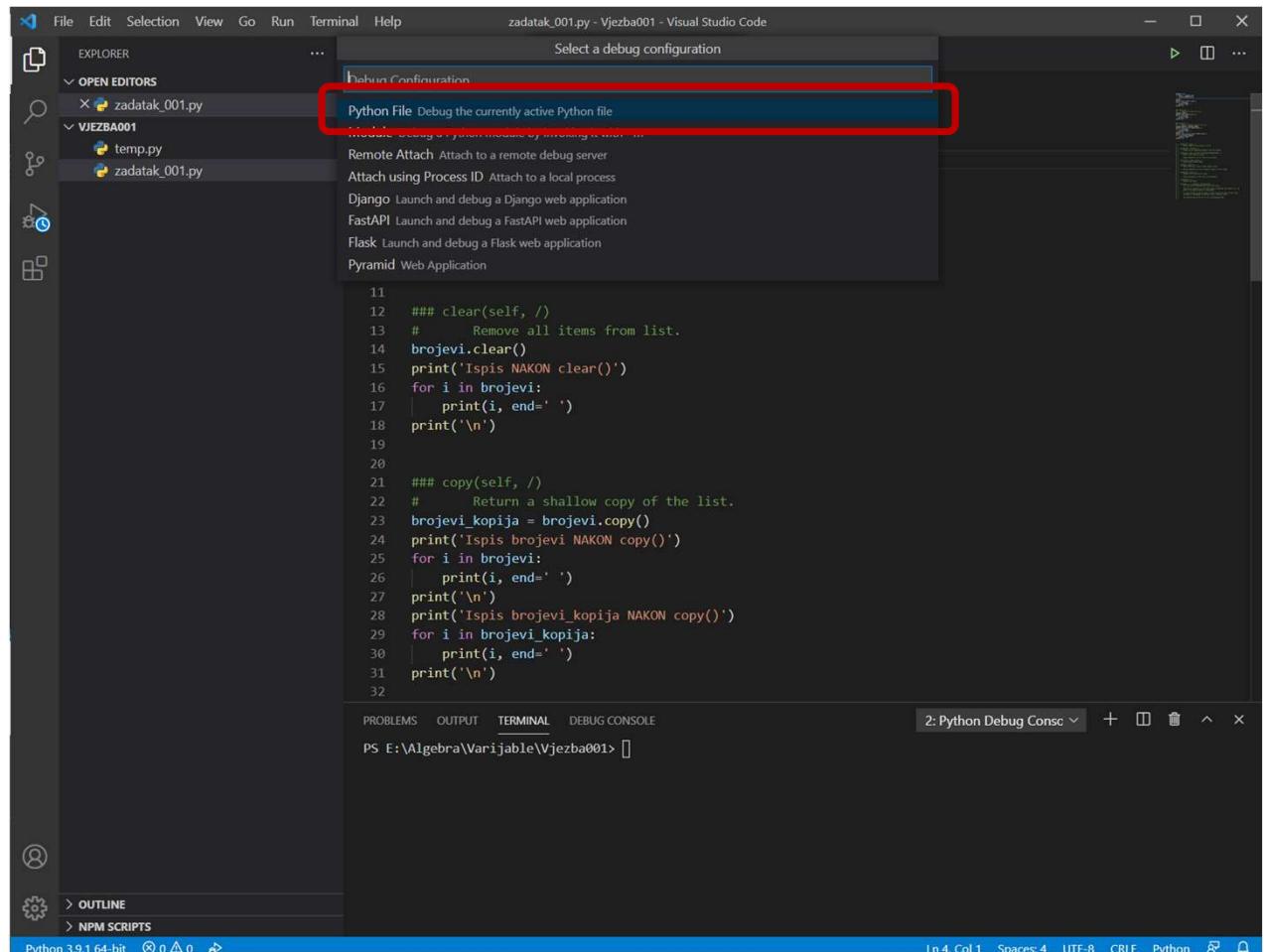
The screenshot shows the Visual Studio Code interface with a Python file named 'zadatak\_001.py' open. The code contains several print statements and a clear method for a list. A red box highlights the number '1.' above the sixth line of code, which is a print statement. This indicates where a break point was set. The code is as follows:

```
1 brojevi = []
2 for broj in range(0, 21):
3     brojevi.append(broj)
4
5 print()
6 print('Ispis CIJELE LISTE')
7 for i in brojevi:
8     print(i, end=' ')
9 print('\n')
10
11
12 ### clear(self, /)
13 #     Remove all items from list.
14 brojevi.clear()
15 print('Ispis NAKON clear()')
16 for i in brojevi:
17     print(i, end=' ')
18 print('\n')
19
20
21 ### copy(self, /)
22 #     Return a shallow copy of the list.
23 brojevi_kopija = brojevi.copy()
24 print('Ispis brojevi NAKON copy()')
25 for i in brojevi:
26     print(i, end=' ')
27 print('\n')
28 print('Ispis brojevi_kopija NAKON copy()')
29 for i in brojevi_kopija:
30     print(i, end=' ')
31 print('\n')
32
```

The status bar at the bottom shows 'Python 3.9.1 64-bit' and other terminal details. The bottom right corner shows the Python logo.

# Debugging

1. Nakon klika na F5, VS Code će vam ponuditi što želite debugirati?
2. Odaberite "**Python File Debug the currently active Python file**"



```
11     ### clear(self, /)
12     #     Remove all items from list.
13     brojevi.clear()
14     print('Ispis NAKON clear()')
15     for i in brojevi:
16         print(i, end=' ')
17     print('\n')
18
19
20     ### copy(self, /)
21     #     Return a shallow copy of the list.
22     brojevi_kopija = brojevi.copy()
23     print('Ispis brojevi NAKON copy()')
24     for i in brojevi:
25         print(i, end=' ')
26     print('\n')
27     print('Ispis brojevi_kopija NAKON copy()')
28     for i in brojevi_kopija:
29         print(i, end=' ')
30     print('\n')
31
32
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE  
PS E:\Algebra\Varijable\Vjezba001> [ ]

File Edit Selection View Go Run Terminal Help zadatak\_001.py - Vjezba001 - Visual Studio Code

EXPLORER OPEN EDITORS zadatak\_001.py VJEZBA001 temp.py zadatak\_001.py

... Select a debug configuration

Python Configuration

Python File Debug the currently active Python file

Module Debug a Python module by invoking it with ...

Remote Attach Attach to a remote debug server

Attach using Process ID Attach to a local process

Django Launch and debug a Django web application

FastAPI Launch and debug a FastAPI web application

Flask Launch and debug a Flask web application

Pyramid Web Application

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Ln 4, Col 1 Spaces: 4 UFT-8 CRLF Python

# Debugging

1. Prvo kliknite na ikonu koja prikazuje aktivne programe.

Ostali crveni okviri prikazuju:

- Ikone za upravljanje pokretanjem programa. Opis na slijedećem slideu.
- Liniju u kojoj je program zaustavljen, a kada se pomoću upravljačkih ikona izvršavanje programa pokrene na slijedeću liniju, onda će ta slijedeća linija biti označena.
  - VAŽNO: Kod u označenoj liniji NIJE se izvršio. To znači da je izvršavanje programa ZAUSTAVLJENO na POČETKU te linije.

1.

```
zadatak_001.py > ...
1 brojevi = []
2 for broj in range(0, 21):
3     brojevi.append(broj)
4
5 print('Ispis CIJELE LISTE')
6 print('Ispis CIJELE LISTE')
7 for i in brojevi:
8     print(i, end=' ')
9 print('\n')
10
11
12 ### clear(self, /)
13 # Remove all items from list.
14 brojevi.clear()
15 print('Ispis NAKON clear()')
16 for i in brojevi:
17     print(i, end=' ')
18 print('\n')
19
20
21 ### copy(self, /)
22 # Return a shallow copy of the list.
23 brojevi_kopija = brojevi.copy()
24 print('Ispis brojevi NAKON copy()')
25 for i in brojevi:
26     print(i, end=' ')
27 print('\n')
28 print('Ispis brojevi_kopija NAKON copy()')
29 for i in brojevi_kopija:
30     print(i, end=' ')
31 print('\n')
32
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS E:\Algebra\Varijable\Vjezba001> e:; cd 'e:\Algebra\Varijable\Vjezba001'; & 'C:\Program Files\Python39\python.exe' 'c:\Users\Lujo\.vscode\extensions\ms-python.python-2021.1.502429796\pythonFiles\lib\python\debugpy\launcher' '59002' '--' 'e:\Algebra\Varijable\Vjezba001\zadatak\_001.py'

Ln 6 Col 1 Spaces:4 UTF-8 CRLF Python

# Debugging

- Ikone za upravljanje pokretanjem programa. Opis s lijeva na desno:
  - Ikona s točkicama je za pozicioniranje ovog okvira - zanemarite
  - Continue (F5) – nastavi izvršavanje programa do kraja.
  - **Step Over (F10)** – ovo je ikona koju ćete najčešće koristiti. Pokreće izvršavanje programa red po red.
  - Step Into (F11)
  - Step Out (Shift + F11)
  - Restart (Ctrl + Shift + F5)
  - Stop (Shift + F5)

Stupac VARIABLES prikazuje trenutnu vrijednost lokalnih i globalnih varijabli. Ako nemate prikaz kao na slici, raširite prikaz klikom na > znak.

```
zadatak_001.py
zadatak_001.py > ...
1 brojevi = []
2 for broj in range(0, 21):
3     brojevi.append(broj)
4
5     print('Ispis CIJELE LISTE')
6     for i in brojevi:
7         print(i, end=' ')
8     print('\n')
9
10
11     ### clear(self, /)
12     # Remove all items from list.
13     brojevi.clear()
14     print('Ispis NAKON clear()')
15     for i in brojevi:
16         print(i, end=' ')
17     print('\n')
18
19
20     ### copy(self, /)
21     # Return a shallow copy of the list.
22     brojevi_kopija = brojevi.copy()
23     print('Ispis brojevi NAKON copy()')
24     for i in brojevi:
25         print(i, end=' ')
26     print('\n')
27     print('Ispis brojevi_kopija NAKON copy()')
28     for i in brojevi_kopija:
29         print(i, end=' ')
30     print('\n')
31
32
33
34
35
36
37
38
39
39
```

VARIABLES

- Locals
  - > special variables
  - broj: 20
  - > brojevi: [0, 1, 2, 3, 4, 5, 6, 7, ..., 20]
- Globals
  - > special variables
  - broj: 20
  - > brojevi: [0, 1, 2, 3, 4, 5, 6, 7, ..., 20]

WATCH

CALL STACK

PAUSED ON BREAKPOINT

BREAKPOINTS

- Raised Exceptions
- Uncaught Exceptions
- zadatak\_001.py

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Ispis brojevi NAKON copy()

Ispis brojevi\_kopija NAKON copy()

PS E:\Algebra\Varijable\Vjezba001> e;; cd 'e:\Algebra\Varijable\Vjezba001'; & 'C:\Program Files\Python39\python.exe' 'c:\Users\Lujo\.vscode\extensions\ms-python.python-2021.1.502429796\pythonFiles\lib\python\debugpy\launcher' '50990' '--> 'e:\Algebra\Varijable\Vjezba001\zadatak\_001.py'

Ln 6, Col 1 Spaces: 4 UTF-8 CRLF Python

# Lista – ostale naredbe

- *naziv\_liste.clear()* – naredba za brisanje svih elemenata liste
- *nova\_lista = naziv\_liste.copy()* – naredba za kopiranje liste
- *broj\_ponavljanja\_u\_listi = naziv\_liste.count(element)* – naredba koja prebrojava koliko se puta element pojavljuje u listi
- *naziv\_liste.extend(nova\_lista)* – naredba koja proširuje postojeću listu novom listom
- *indeks\_elementa = naziv\_liste.index(element)* – naredba koja dohvaća indeks pozicije na kojoj se nalazi element
- *naziv\_liste.insert(indeks, element)* – naredba za umetanje elementa na poziciju točno ispred pozicije označene navedenim indeksom
- *naziv\_liste.sort()* – naredba za sortiranje elemenata liste
- *naziv\_liste.reverse()* – naredba za sortiranje elemenata liste obrnutim redoslijedom

# Zadatak – lista

- Napišite program koji kreira akorde na osnovu početnog tona, odnosno note.
  - POJAŠNJENJE
  - Akord se sastoji od tri tona koji se mogu ponavljati.
  - Durski akord čine: početni ton, 4. ton te 7. ton. Označava se samo velikim slovom početnog tona ili velikim slovom početnog tona uz dodatak dur
  - Molski akord čine: početni ton, 3. ton te 7. ton. Označava se samo malim slovom početnog tona ili malim slovom početnog tona uz dodatak mol
  - Glazbena abeceda počinje od C: C, C#, D, D#, E, F, F#, G, G#, A, A#, H
  - Engleska oznaka za H ton je B tako da oni imaju A B C D E F G tonove
  - Postoji pojašnjenje u teoriji glazbe zašto je prvi ton C, ali to sada nije važno.



# Kolekcije podataka – Rječnik

- Dictionary ili Rječnik je kolekcija parova podataka.
- Rječnik koristi { } zagrade
- Svaki element Rječnika ima dva dijela:
  - Key ili Ključ
  - Value ili Vrijednost
- Key ili Ključ mora biti jedinstven u Rječniku. Zato jer Ključ mora biti jedinstven dopušten tipovi podataka za Ključ su:
  - String; Brojevi i N-terac
- Pomoću Ključa pristupamo drugom dijelu para, odnosno podacima. Ključ je isto kao i Indeks u listi.
- Value ili Vrijednost predstavlja sadržaj koji želimo pohraniti u kolekciju, odnosno Rječnik. Value može biti bilo koji tip podatka, a često je druga kolekcija kao lista ili drugi rječnik

# Rad s Rječnikom / Dictionary

- Manipulacija podacima unutar Rječnika:
  - *naziv\_rjecnika[key]*
  - *naziv\_rjecnika.items()*
  - *naziv\_rjecnika.keys()*
  - *naziv\_rjecnika.values()*

# Vježba – rječnik

- Kreirajte bazu s vozilima firme. ID svakog retka je cijeli broj, a podaci koji se čuvaju o svakom vozilu su: tip, proizvođač, registrarska oznaka, godina prve registracije te cijena u eurima. Ispišite cijelu tablicu tako da ID odvojite od ostatka retka jednim TABom, a druge informacije formatirajte tako da prvi red tablice predstavlja naslovni red, a ostali redovi tablice predstavljaju podatke iz baze.



# Vježba – rječnik

ID	Tip	Proizvođač	Registarska oznaka	Godina prve registracije	Cijena u EUR
1	Kamion	Iveco	OS 001 ZZ	2015	45.000,00 €
2	Kamion	Iveco	OS 002 ZZ	2015	47.000,00 €
3	Tegljač	MAN	RI 001 ZZ	2018	78.000,00 €
4	Tegljač	MAN	RI 002 ZZ	2020	97.000,00 €
5	Kombi	Mercedes Benz	ST 001 ZZ	2013	12.000,00 €
6	Kombi	Volkswagen	ST 002 ZZ	2021	35.000,00 €
7	Dostavno vozilo	Volkswagen	ZG 001 ZZ	2010	9.000,00 €
8	Dostavno vozilo	Volkswagen	ZG 002 ZZ	2010	9.300,00 €

# Rad s Rječnikom / Dictionary

- Manipulacija podacima unutar Rječnika nastavak:
  - *naziv\_rjecnika.clear()*
  - *naziv\_rjecnika.pop(key, default)*
  - *naziv\_rjecnika.popitem()*

# Kolekcije podataka – N-terac (tuple)

- Tuple koristi ( ) zagrade
- N-terac je Nepromjenjiv.
- Primjena
  - TUPLE se često koristi kao tip pohrane povezanih podataka unutar neke kolekcije. Zato jer je nepromjenjiv koristi se kao ključ u Rječnicima
  - Recimo podaci o nekoj osobi su pohranjeni u jednu TUPLE kolekciju, a onda te TUPLE kolekcije su pohranjene u neku listu
- Provjerite što vraćaju naredbe za Dictionary:
  - *naziv\_rjecnika.items()*
  - *naziv\_rjecnika.keys()*
  - *naziv\_rjecnika.values()*



Kontrola toka  
izvršavanja  
programskog  
kôda

# Tok izvršavanja kôda

- Često koristimo uvjete kako bismo kontrolirati tijek događaja. Recimo, kada dođete u banku i želite podići neki iznos novca. Ako imate dovoljno na računu i odobren minus, onda ćete dobiti novac, a ako nemate, dobit ćete ispriku da nemate dovoljno novaca na računu i da vam ne mogu isplatiti trženi iznos.
- Uvjete koristimo i za kontrolu toka izvršavanja kôda. Za to koristimo petlje.
- Ponavljanje kôda i predefiniranom broju iteracija
  - FOR petlja – sve dok ima elemenata i kolekciji ...
- Uvjetno izvršavanje
  - IF ... ELSE; IF ... ELIF ... ELSE petlja – ako je uvjet ispunjen izvrši ...
- Uvjetno ponavljanje kôda
  - WHILE petlja – sve dok je uvjet ispunjen izvršavaj ...
- Kombinacija

# Logički operatori

Operator	Opis	Primjer
>	Veće od – a je veće od b	$a > b$
<	Manje od – a je manje od b	$a < b$
==	Identično – a je identično b	$a == b$
!=	NIJE identično – a nije identično b	$a != b$
>=	Veće i jednako od – a je veće i jednako od b	$a >= b$
<=	Manje i jednako od – a je manje i jednako od b	$a <= b$

# Tabela logičkih izraza

A	B	A and B	A or B	not B
✓ True	✓ True	✓ True	✓ True	✗ False
✓ True	✗ False	✗ False	✓ True	✓ True
✗ False	✓ True	✗ False	✓ True	✗ False
✗ False	✗ False	✗ False	✗ False	✓ True

# IF ELSE UVJETNA NAREDBA

- Analogno primjeru iz banke u vezi podizanja novca s računa, u programskim jezicima imamo naredbe koje ovisno o uvjetu određuju tijek izvršavanja programa. To je IF ELSE naredba i u svim programskim jezicima ima istu logiku.
- AKO (IF) je uvjet ispunjen (njegova vrijednost je True), tada će se izvršiti blok instrukcija INAČE (ELSE) će se izvršiti drugi blok instrukcija.

# IF ELSE UVJETNA NAREDBA

`if prvi_uvjet:`

izvrši instrukcije SAMO AKO je prvi\_uvjet točan ili ima vrijednost True

`elif drugi_uvjet:`

Ako prvi uvjet NIJE zadovoljen, znači da je prvi\_uvjet NE točan ili ima vrijednost False

Tada je izvršavanje programa došlo do ove linije pa OPET slijedi provjera AKO je drugi\_uvjet točan ili ima vrijednost True izvrši instrukcije u ovom bloku

`elif treci_uvjet:`

isto kao i za drugi uvjet i za četvrti i peti i ... nema ograničenja u ELIF provjerama

`else:`

AKO niti jedan od uvjeta NIJE ispunjen, odnosno svi su FALSE

TADA BEZ obzira na UVJETE I VRIJEDNOSTI IZVRŠI aktivnosti iz ovog bloka

# IF ELSE UVJETNA NAREDBA - Zadaci

- Kreirajte listu od 1 do broja 30. Ispišite sve brojeve koji su djeljivi s 3, 6 i 9
  - Provjera je li broj djeljiv s nekim drugim radimo pomoću % (modulo) operanda.
  - $15 \% 3$  NEMA ostatka, odnosno to je 0 pa je 15 djeljiv s 3.
  - $16 \% 3$  je 1, odnosno NIJE jednak 0 pa 16 NIJE djeljiv s 3.
- Napišite program koji provjerava pripada li unesena riječ vrsti riječi *palindrom*.
  - Palindrom je riječ koja se jednako piše (i čita) s lijeva na desno i s desna na lijevo

# Vježba – rječnik ISPRAVITE ISPIS

- Kreirajte bazu s vozilima firme. ID svakog retka je cijeli broj, a podaci koji se čuvaju o svakom vozilu su: tip, proizvođač, registrarska oznaka, godina prve registracije te cijena u eur. Ispišite cijelu tablicu tako da ID odvojite od ostatka retka jednim TABom, a druge informacije formatirajte tako da prvi red tablice predstavlja naslovni red, a ostali redovi tablice predstavljaju podatke iz baze.



# IF ELSE NAREDBA – Zadaci tekst

- U generičkom tekstu 'Lorem ipsum ...' (<https://www.lipsum.com/>) pronađite koliko se puta pojavljuje neka riječ.
  - Probajte s Lorem.

# WHILE petlja

- IF petlja je imala uvjet koji ako je ispunjen izvršava se blok programskog kôda te petlje SAMO JEDNOM.
- Nakon završetka bloka programskog kôda IF petlje, program se nastavlja izvršavati dalje, izvan IF petlje.
- WHILE petlja je slična. Isto tako ima uvjet koji ako je ispunjen, osigurava pokretanje bloka programskog kôda unutar WHILE petlje, ali nakon završetka tog bloka programskog kôda, WHILE petlja će ponoviti provjeru uvjeta.
- Ako je uvjet ispunjen, OPET će se pokrenuti isti blok programskog kôda.
- I to će se ponavljati SVE DOK (while) je uvjet ispunjen.
- WHILE petlju možemo usporediti s FOR petljom, samo što je uvjet kod WHILE petlje jasno iskazan, naveden, dok kod FOR petlje je uvjet skriven i vezan je uz postojanje elemenata u kolekciji.

# **WHILE petlja**

**while uvjet:**

sve dok je uvjet točan izvrši instrukcije u ovom bloku.

Kada završiš s izvršavanjem zadnje instrukcije ponovi provjeru uvjeta te ako je ispunjen ponovi izvršavanje bloka instrukcija. To ponavljam sve dok uvjet nije ispunjen.

- Prepravite zadatak „vozni park” tako da umjesto FOR, radi s WHILE petljom

# **break, continue, pass**

- Kada želimo izvršavanje petlje završiti prije kraja - "nasilno izaći iz petlje" koristimo ključnu riječ BREAK
- Kada NE želimo izaći iz petlje, ali želimo da se jedan ili više ciklusa petlje NE izvrši do kraja, nego da se jedan dio instrukcija PRESKOČI te ako su ispunjeni uvjeti započne novi ciklus, koristimo naredbu CONTINUE
- Komentari se u kôdu ignoriraju, preskaču, ne postoje. Međutim, kada trebamo izvršiti naredbu koje na radi ništa koristimo naredbu PASS
  - PASS možemo smatrati kao komentar koji se izvršava. Program smatra da ima naredbu za izvršiti, samo što ta naredba ne radi ništa.

# Zadaci

- Preraditi vježbe vezane uz FOR petlju tako da umjesto FOR petlje koriste WHILE petlju:
  - Je li riječ palindrom
  - Baza vozila firme
  - Generator akorda

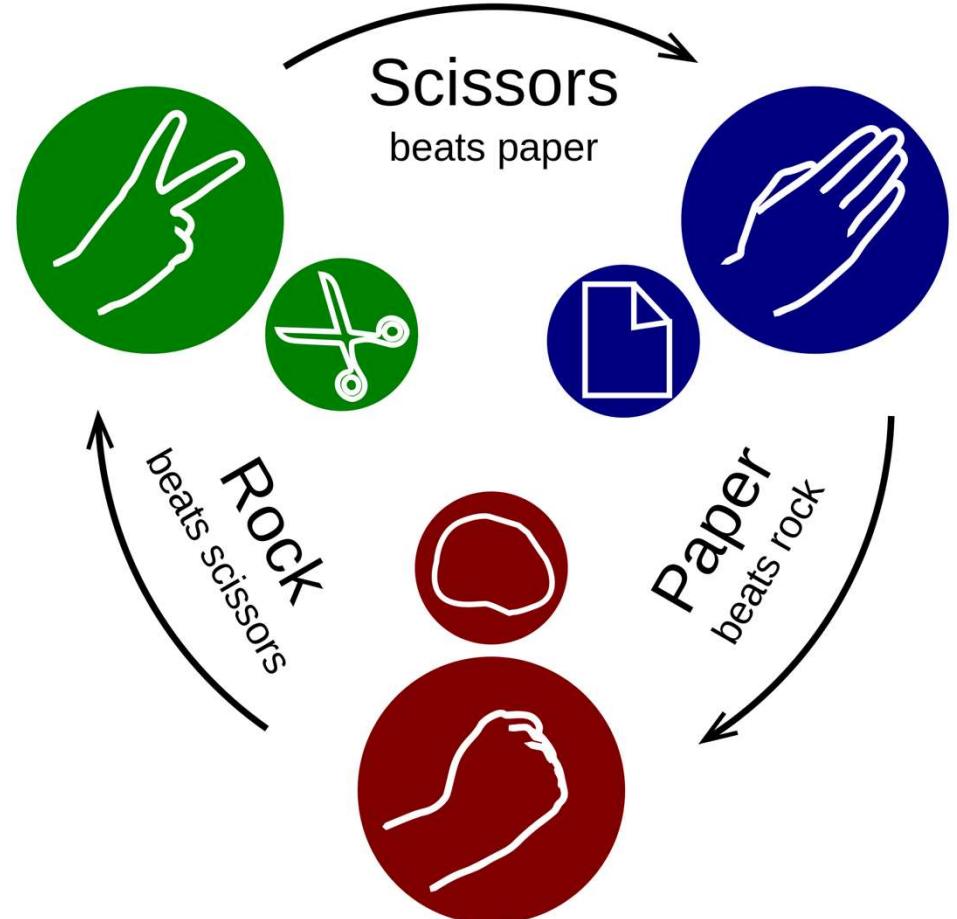
# Zadaci

- Napravite aplikaciju za prikaz tablice množenja. Korisnik treba moći unijeti do kojeg broja će se prikazati tablica.
- Sjećate se Računalnog razmišljanja i pograđanja broja između 1 i 100. Sada imate dovoljno znanja da napišete program koji će Vam omogućiti igranje igre pograđanje broja.
- Napravite aplikaciju za konverziju (u oba smjera):
  - km u milju – (1 km = 0.6214 milje)
  - °C u °F – ( $0^{\circ}\text{C} = 32^{\circ}\text{F}$ ) obrnuto  $T_{(^{\circ}\text{F})} = T_{(^{\circ}\text{C})} * (9/5) + 32$
  - kg u funtu (pounds) –  $1 \text{ kg} = 2.2046 \text{ pounds}$
  - Litra u US galon –  $1l = 0.2642 \text{ US gal}$
  - kW (kilowatt) u ks (horsepower ili konjska snaga) –  $1 \text{ kW} = 1.3596$
- Rezultate u svim zadacima je potrebno formatirano ispisati na ekran.
- U svim zadacima, nakon pokretanja programa, korisnik treba imati mogućnost izbora želi li nastaviti koristiti program ili želi završiti, odnosno izaći iz programa.

# Kamen-Škare-Papir

## Pogodi broj

- Napravite program koji će vam omogućiti igranje igre protiv računala - kamen, škare, papir



[Rock-paper-scissors - Rock paper scissors - Wikipedia](#)

# Funkcije u Python-u



# Funkcije

- Funkcija kao crna kutija:
  - Na jednoj strani ima ulaz u koji ubacimo ono što želimo „obraditi“. Mi ne znamo kako funkcija obrađuje ono što smo joj predali (čak nam nije bitno)
  - Na drugoj strani ima izlaz gdje se pojavi rezultat obrade. Kada funkcija završi obradu onoga što smo joj predali na obradu, funkcija nam vrati obrađeni rezultat.
- Ubacite novac u aparat i dobijt ćete kavu, čaj, slatkiš ...



# Funkcije

- Skup instrukcija koje se izdvojene u zasebnu cjelinu.
  - Instrukcije u funkciji izdvojene su kao i u FOR ili WHILE petlji
  - Samo za pokretanje tih instrukcija vrijede drugačija pravila.
- Funkcije pokrećemo pomoću „poziva“ funkcije
- Namjena funkcija je izdvojiti kôd koji se ponavlja u zasebnu cjelinu. Tako dobivamo:
  - Jednom napisani kôd možemo koristiti koliko god nam puta treba
  - Održavanje kôda je jednostavnije
  - Debugging je jednostavniji jer nema redundancije (ponavljanja)
- Funkcija bi trebala raditi samo jednu aktivnost. Funkcija koja pohranjuje podatke u bazu, ne treba računati s tim podacima, ne treba pitati korisnika da unese podatke ... funkcija iz primjera ima samo jednu funkciju, a to je pohrana podataka u bazu
- Metode – pojednostavljeno, metode su funkcije unutar klase.

# Funkcije imaju

- Naziv
  - Koristi se pokretanje funkcije (ovo još nazivamo "poziv" funkcije)
  - Vrijede ista pravila kao i za varijable
- Argumente ili parametre
  - To su podaci koje će funkcija prihvati na ulazu i obratiti
  - Funkcije mogu, ali i ne moraju imati parametre. To znači da ima funkcija koje ne zahtijevaju ništa na ulazu da bi obavile posao (funkcija koja ispisuje glavni izbornik). Dovoljno je samo pozvati takve funkcije
- Tijelo ili blok instrukcija
  - Predstavlja skup instrukcija koje će izvršiti potrebnu transformaciju ulaznih podataka ili će samo izvršiti određenu akciju.
  - Ovo predstavlja način kako funkcija obavlja svoju namjenu.
- Rezultat ili return parametar
  - Funkcije nakon obrade argumenata „vrate“ nekakav rezultat. Neke funkcije ne vrate „ništa“, nego samo ispišu poruku o uspjehu ili neuspjehu aktivnosti.

# Funkcije – osnovna podjela

## Ugrađene funkcije

- Svaki programski jezik ima ugrađene funkcije
  - help()
  - print()
  - input()
  - int(), float(), str()
  - len()
  - ...

## Korisnički definirane funkcije

- Funkcije koje kreira programer ovisno o potrebama programa

# Ugrađene funkcije

- Do sada smo upoznali neke ugrađene funkcije
- Uz Python se često dodaje „Batteries included“
  - Dakle, većinu toga što trebate dolazi u paketu
- Neke opcije ipak treba uključiti – moduli
- Moduli predstavljaju skup funkcionalnosti koje su izdvojene u zasebnu cjelinu.
  - Moduli imaju svoje specifične funkcije
- Neki su uključeni odmah, a neke module treba uključiti po potrebi
- Dio rezultata `help(print)`:  
**Help on built-in function print in module builtins:**

# Moduli

- Uključivanje modula se radi na početku datoteke pomoću ključne riječi **import** iza koje dolazi naziv modula
- Kasnije možemo koristiti ugrađene funkcije iz tog modula
- Neki od ugrađenih modula:
  - **Random** – funkcije za generiranje nasumičnih brojeva
  - **Math** – matematičke funkcije kao potenciranje, kvadriranje, trigonometrijske funkcije i sl.
  - **Datetime** – funkcije za upravljanje tipom varijabli za pohranu vremena (datuma i vremena kao sati, minuta, sekundi)
  - **Os** – funkcije za rad s nekim elementima operativnog sustava na kojem se pokreće Python program. Na primjer datoteke, mape, konzola i sl.
- Ako postoje „ugrađeni“ moduli, znači da postoje i „vanjski“ moduli. To su moduli koji ne dolaze s instalacijom Pythona i potrebno ih je naknadno instalirati.
  - Takve module ćemo najviše koristiti kod obrade velike količine podataka, a neke i ranije

# Korisnički definirane funkcije

- Funkcije koje ovisno o potrebama programa radimo sami
- Kreiranje funkcije u Pythonu:

```
def naziv_funkcije(argumenti):
    """ Docstring """
    instrukcije
```

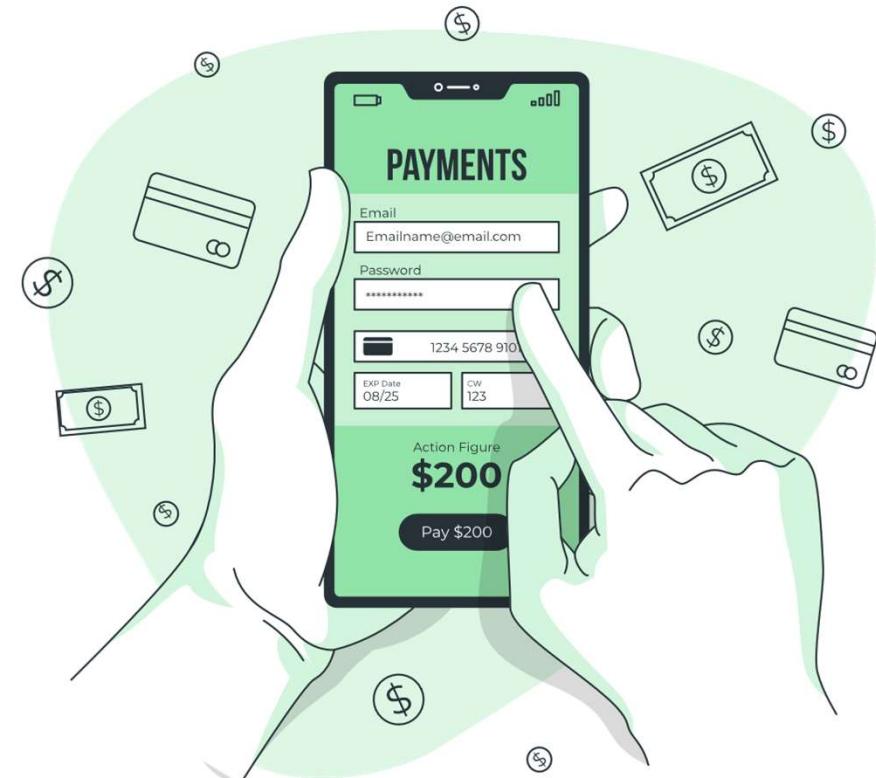
- Kreiranje funkcije počinje ključnom riječi **def**
- Nakon def dolazi **naziv funkcije**, po pravilima koja vrijede za varijable
- Iza naziva funkcije slijedi **lista argumenata**, međusobno odvojenih zarezom te grupiranih unutar zagrada
- Na kraju prve linije obavezna :
- Odmah ispod deklaracije funkcije, **opcionalno** se dodaje **Docstring**. Opis što radi funkcija te koje argumente prima.
- I nakon toga piše se kôd koji radi ono zbog čega smo i kreirali funkciju.

# Zadaci

- Preraditi prethodne vježbe tako da sada koriste funkcije:
  - Je li riječ palindrom
  - Baza vozila firme
  - Kamen-Škare-Papir
  - Izradite aplikaciju za konverziju mjernih jedinica tako da korisnik bira koju jedinicu će konvertirati u koju
  - Izradite aplikaciju za preračunavanje potrošnje goriva automobila u kune uz mogućnost izbora izračuna koliko maksimalno auto smije trošiti na 100 km, ako je ciljana mjesecna potrošnja X kuna?

# Korisnički definirane funkcije – vježbe

- Kupovina u online dućanima je postala svakodnevica. Za to nam je potrebna kreditna kartica. Ali Internet nije sigurno mjesto.  
Napišite program koji će uneseni broj kartice korisnika zaštititi tako da sve znakove osim zadnja četiri maskira pomoću # znakova. Primjer:
  - Broj:  
521478523691234, treba biti:  
#####/#1234
  - Dodatak:**
  - Dodajte mogućnost da ako je korisnik unio broj kartice s "-" znakovima da se onda ti znakovi ne zaštićuju.  
3698-521-47852, treba biti: #####-###-#7852
  - Dodatno neka korisnik bira kojim znakom će zaštititi broj kartice.



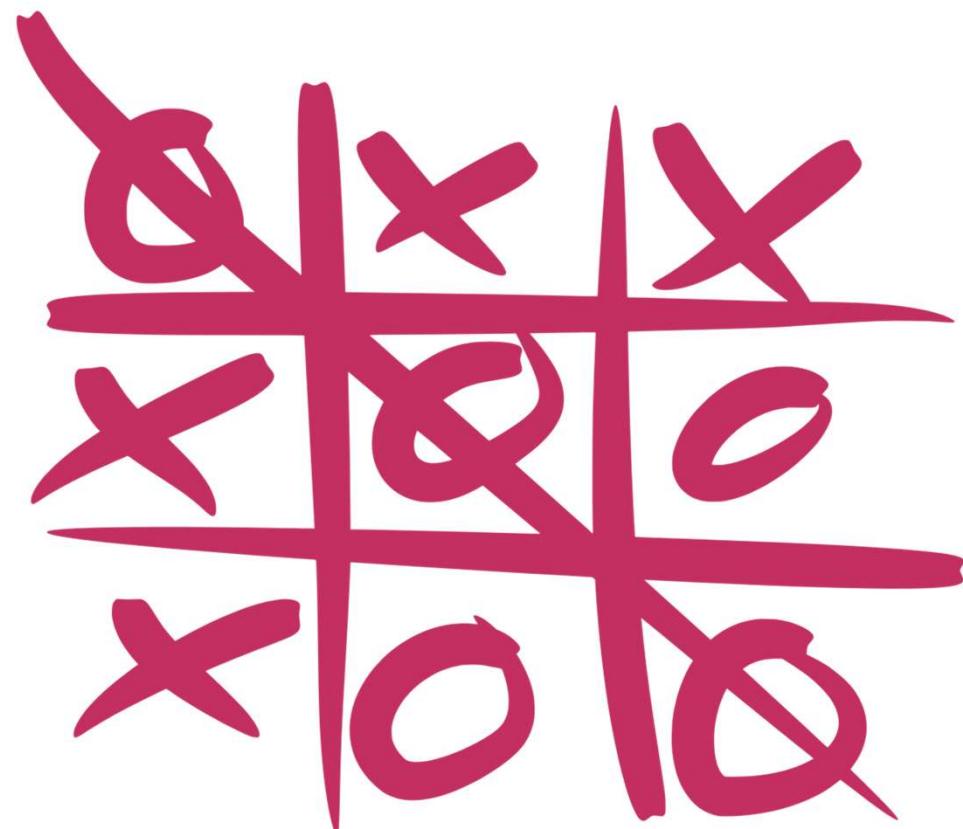
# Korisnički definirane funkcije – vježbe

- Prepravite zadatak koji generira akorde na osnovu početnog tako da se generiranje akorda izvršava u funkciji



# Vježba

- Izradite igricu Križić kružić
- Za sada se igrica pokreće u konzoli.



# Identity Management

Funkcionalnosti:

- Predefinirani Administrator sustava koji može dodati nove korisnike u sustav, ažurirati i brisati postojeće.
- Svaki korisnik ima: ime, prezime, korisničko ime (UserName) i Zaporku (Password)
- Zaporka (Password) mora imati minimalno 10 znakova
- Korisničko ime (UserName) mora biti jedinstveno

Uspješna prijava na ekranu ispisuje poruku:

*Dobro došli, {ime} {prezime}*

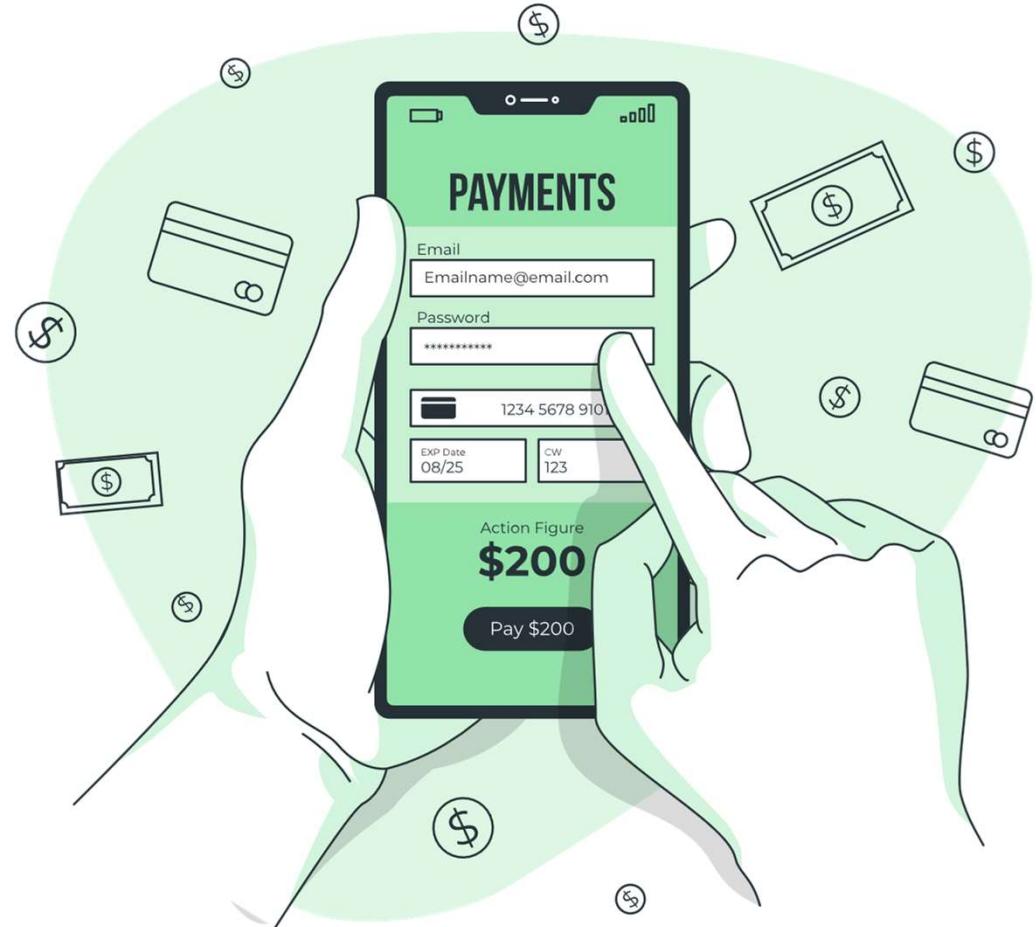


# Napravite program za rad na pultu u banci

Program za sada izvršava u konzoli

Funkcionalnosti:

- Izbornik
- Otvaranje računa tvrtke
- Prikaz stanja računa
- Prikaz prometa po računu
- Polog novca na račun
- Podizanje novca s računa
- Izlaz iz programa (program se nakon svake akcije vrati na početni izbornik u kojem postoji opcija Izlaz)



Hvala na  
pažnji!

