



Co-funded by  
the European Union

University of Bihac  
Technical faculty

# BASICS OF PROGRAMMING PYTHON

Una Drakulić MA, Melisa Haurdić MA

UNBI



UNIVERSITY OF LJUBLJANA  
Faculty of Electrical Engineering



University of Pristina  
Kosovska Mitrovica



# 1. Introduction to Python

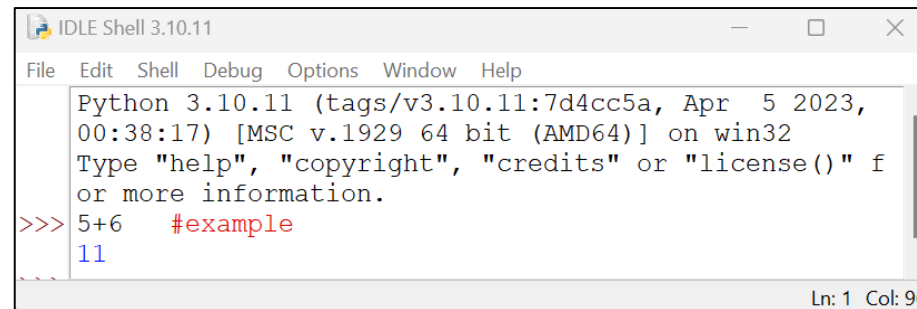
---

- Variables in Python don't have a type designation, although the concept of data types exists. This means that we can assign a value of any type to a variable.
- In Python, memory is automatically freed. In other words, there is no command to explicitly remove or deallocate an object from memory. This is made possible by the garbage collector system, an automatic memory management system that is part of Python's runtime system.
- A program written in Python is not compiled into the native code of the specific computer, but is executed using another program called an interpreter.



# 1. Introduction to Python

- Python comes with a standard development environment called IDLE (Integrated Development and Learning Environment). After launching IDLE, you get a command line where you can enter your code. After entering the code, pressing the Enter or Return key below the line gives you the result of the entered code.



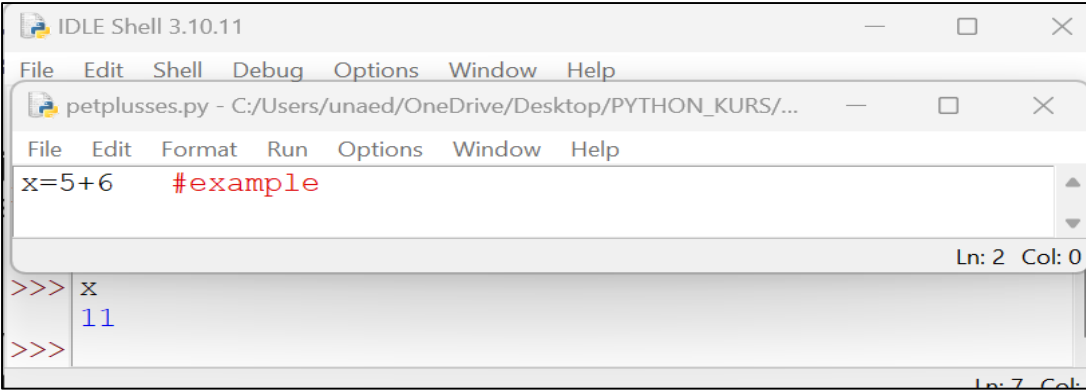
```
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5+6 #example
11
```

Figure 1. Example of adding two numbers using the IDLE Shell

- Text after the symbol # is considered a comment and is not executed when the code runs.

## 2. Basic concepts

- An expression is any construct in a programming language whose execution returns a value as a result ( $5 + 6$  is an expression because it gives a result, i.e., the value 11;  $5 < 6$  is a logical expression, whose result in this case would be 1 or True).
- Statements do not have a defined result ( $x = 5 + 6$  represents an assignment statement). When we assign a certain value to a variable, it remains assigned to that variable until we assign a different value to it.



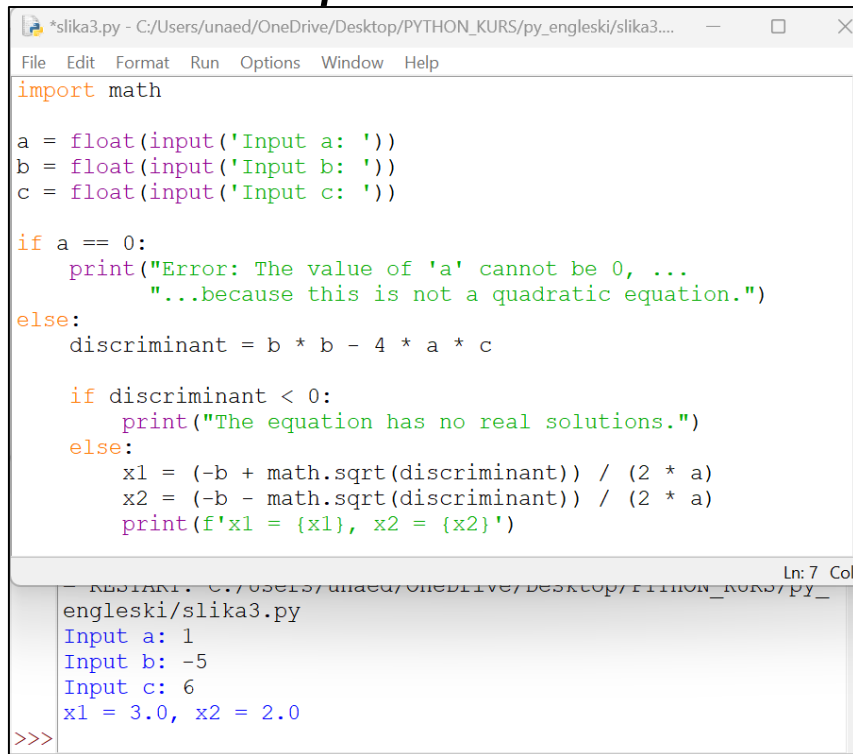
The screenshot shows two windows from the IDLE 3.10.11 environment. The top window, titled 'petplusses.py', contains a single line of code: `x=5+6` followed by a red comment `#example`. The bottom window is the IDLE Shell, which shows the execution of the script. It displays the prompt `>>>`, the variable `x` being assigned, and the resulting value `11` printed below it. The shell prompt `>>>` appears again on the next line.

Figure 2. Calling the value of  $x$  from a module `petplusses.py`



## 2. Basic concepts

- The function *print* is used to output the result of any expressions..



```
*slika3.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika3...
File Edit Format Run Options Window Help
import math

a = float(input('Input a: '))
b = float(input('Input b: '))
c = float(input('Input c: '))

if a == 0:
    print("Error: The value of 'a' cannot be 0, ...
        "...because this is not a quadratic equation.")
else:
    discriminant = b * b - 4 * a * c

    if discriminant < 0:
        print("The equation has no real solutions.")
    else:
        x1 = (-b + math.sqrt(discriminant)) / (2 * a)
        x2 = (-b - math.sqrt(discriminant)) / (2 * a)
        print(f'x1 = {x1}, x2 = {x2}')
```

Ln: 7 Col: 1

```
Restart: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_
engleski/slika3.py
Input a: 1
Input b: -5
Input c: 6
x1 = 3.0, x2 = 2.0
>>>
```

Figure 3. Uses of the *print* command for different data types

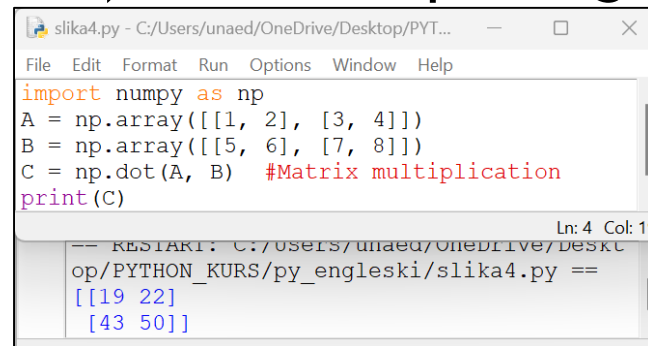
*Math* is a built-in Python module that provides mathematical functions and constants. The most commonly used ones are:

- `math.sqrt(x)`;
- `math.pow(x,y)`;
- `math.sin(x)`;
- `math.cos(x)`;
- `math.pi`.



## 2. Basic concepts

- *NumPy* (Numerical Python) is a basic package for numerical calculations.

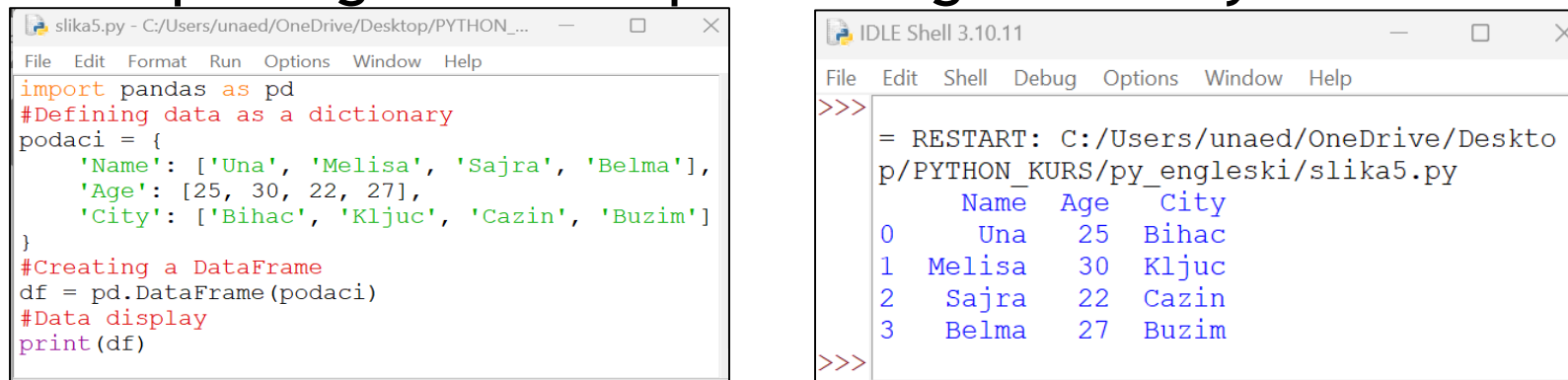


```
File Edit Format Run Options Window Help
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B) #Matrix multiplication
print(C)

Ln: 4 Col: 19
-- RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika4.py ==
[[19 22]
 [43 50]]
```

Figure 4. Example of using the NumPy package

- *Pandas* is a package for data processing and analysis.



```
File Edit Format Run Options Window Help
import pandas as pd
#Defining data as a dictionary
podaci = {
    'Name': ['Una', 'Melisa', 'Sajra', 'Belma'],
    'Age': [25, 30, 22, 27],
    'City': ['Bihac', 'Kljuc', 'Cazin', 'Buzim']
}
#Creating a DataFrame
df = pd.DataFrame(podaci)
#Data display
print(df)
```

```
IDLE Shell 3.10.11
File Edit Shell Debug Options Window Help
>>>
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika5.py
   Name  Age  City
0   Una   25 Bihac
1 Melisa  30 Kljuc
2  Sajra  22 Cazin
3  Belma  27 Buzim
>>>
```

Figure 5. Example of using the Pandas package



## 2. Basic concepts

- *Matplotlib* is the standard package for plotting graphs.



Figure 6. Example of using the matplotlib package

- *Scikit-learn* is a package for classic machine learning algorithms.

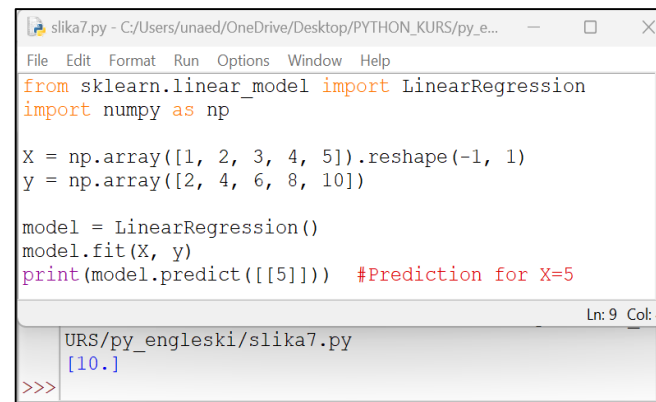


Figure 7. Example of using the scikit-learn package



## 2. Basic concepts

- *TensorFlow/PyTorch* package is a deep learning package. The way to install the package is to use the command: `pip install tensorflow` or `pip install torch`, and an example of use is shown in Figure 8.

```
*slika8.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika8.py (3.10.11)*
File Edit Format Run Options Window Help
import tensorflow as tf
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
#Loading Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
#Division into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
#Preprocessing the target variable (label)
encoder = LabelEncoder()
y_train = encoder.fit_transform(y_train)
y_test = encoder.transform(y_test)
#Model building
model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu')) #Input layer with 4 inputs and 10 neurons
model.add(Dense(3, activation='softmax')) #Output layer with 3 classes (iris types)
#Model compilation
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#Model training
model.fit(X_train, y_train, epochs=50, batch_size=10)
#Model evaluation
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy}")
```

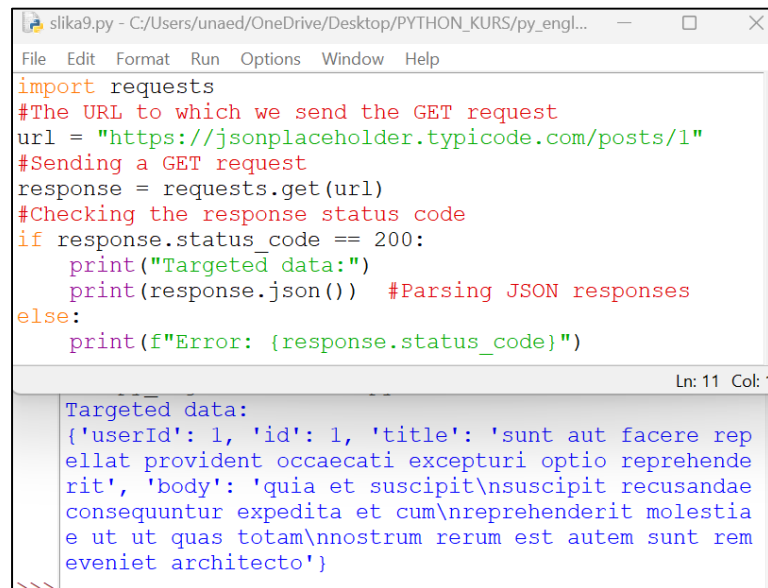
```
IDLE Shell 3.10.11
File Edit Shell Debug Options Window Help
.7500 - loss: 0.5169 [Progress bar]
[Progress bar] [1m11/12] [0m] [32m]
[Progress bar] [0m] [37m] [0m] [1m0s] [0m] 14ms/st
ep - accuracy: 0.8723 - loss: 0.4836 [Progress bar]
[Progress bar] [1m12/12]
[0m] [32m] [Progress bar] [0m] [37m] [0m] [
1m0s] [0m] 18ms/step - accuracy: 0.8753 - loss: 0.4815
[1m1/1] [0m] [32m] [Progress bar] [0m] [3
7m] [0m] [1m0s] [0m] 230ms/step - accuracy: 0.9333 - loss: 0.47
43 [Progress bar]
[Progress bar] [1m1/1] [0m] [32m]
[Progress bar] [0m] [37m] [0m] [1m0s] [0m] 279ms/step - accuracy: 0.933
3 - loss: 0.4743
```

Figure 8. Example of using the Tensorflow package



## 2. Basic concepts

- *Requests* - HTTP package for requests, which enables sending GET and POST requests, authentication and working with APIs, downloading data in JSON format and others. The way to install the package is to use the command: `pip install requests`, and an example of use is shown in Figures 9 and 10.

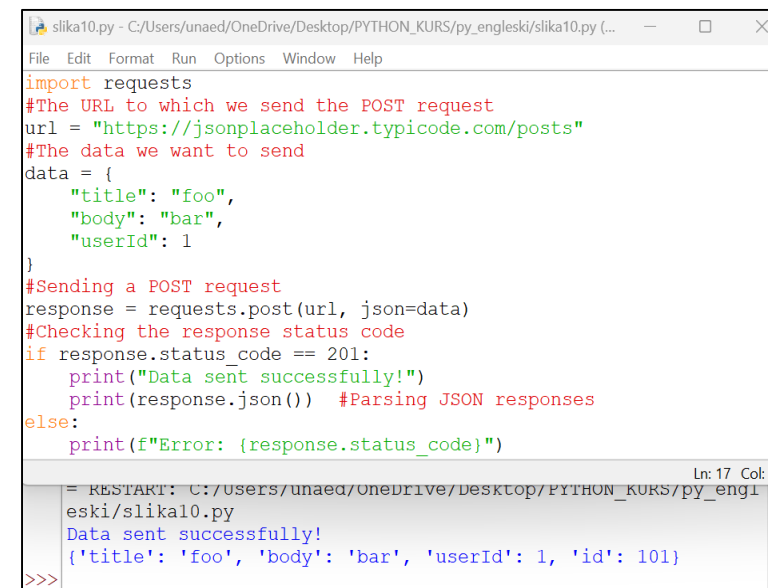


```
File Edit Format Run Options Window Help
import requests
#The URL to which we send the GET request
url = "https://jsonplaceholder.typicode.com/posts/1"
#Sending a GET request
response = requests.get(url)
#Checking the response status code
if response.status_code == 200:
    print("Targeted data:")
    print(response.json()) #Parsing JSON responses
else:
    print(f"Error: {response.status_code}")

Ln: 11 Col: 1

Targeted data:
{'userId': 1, 'id': 1, 'title': 'sunt aut facere rep
ellat provident occaecati excepturi optio reprehende
rit', 'body': 'quia et suscipit\nsuscipit recusandae
consequatur expedita et cum\nreprehenderit molestia
e ut ut quas totam\nnostrum rerum est autem sunt rem
eveniet architecto'}
```

Figure 9. Example of using the request package - GET request



```
File Edit Format Run Options Window Help
import requests
#The URL to which we send the POST request
url = "https://jsonplaceholder.typicode.com/posts"
#The data we want to send
data = {
    "title": "foo",
    "body": "bar",
    "userId": 1
}
#Sending a POST request
response = requests.post(url, json=data)
#Checking the response status code
if response.status_code == 201:
    print("Data sent successfully!")
    print(response.json()) #Parsing JSON responses
else:
    print(f"Error: {response.status_code}")

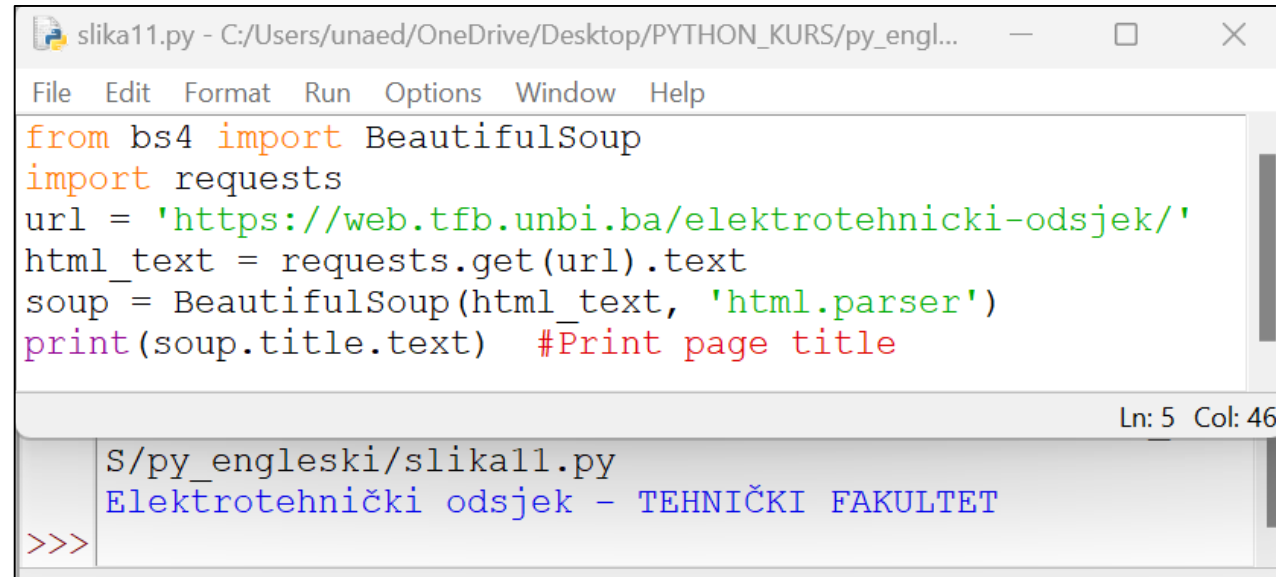
Ln: 17 Col: 1

= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engl
eski/slika10.py
Data sent successfully!
{'title': 'foo', 'body': 'bar', 'userId': 1, 'id': 101}
```

Figure 10. Example of using the request package - POST request

## 2. Basic concepts

- *BeautifulSoup* is a package that enables parsing and data extraction from HTML and XML documents. The way to install the package is to use the command: `pip install beautifulsoup4`, and an example of use is shown in Figure 11.

A screenshot of a Python IDE window titled 'slika11.py - C:/Users/unaed/OneDrive/Desktop/PYTHON\_KURS/py\_engl...'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains the following Python code:

```
from bs4 import BeautifulSoup
import requests
url = 'https://web.tfb.unbi.ba/elektrotehnicki-odsjek/'
html_text = requests.get(url).text
soup = BeautifulSoup(html_text, 'html.parser')
print(soup.title.text) #Print page title
```

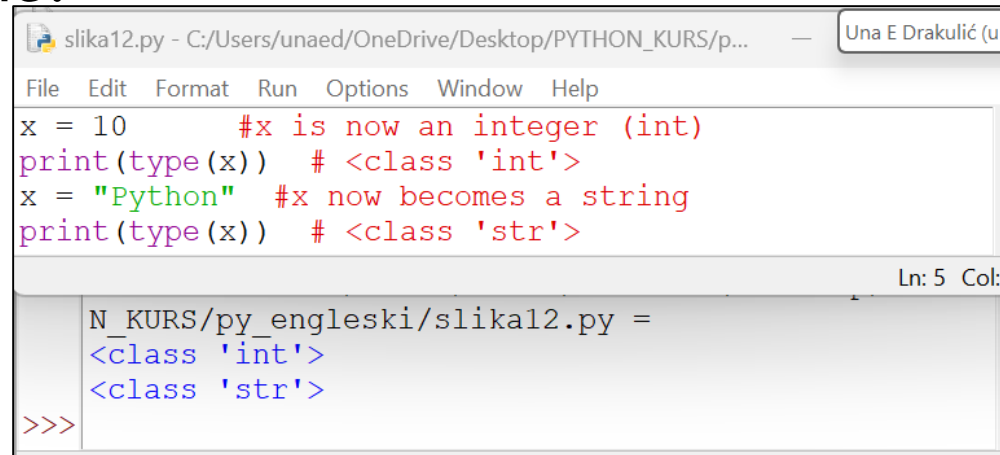
The status bar at the bottom right shows 'Ln: 5 Col: 46'. Below the code area, the output of the script is displayed: 'S/py\_engleski/slika11.py' and 'Elektrotehnički odsjek - TEHNIČKI FAKULTET'. The prompt '(>>>)' is visible at the bottom left of the output area.

Figure 11. Example of using the *beautifulsoup4* package



### 3. Data types

- Python belongs to the group of dynamically typed programming languages, which means that the type of a variable is not fixed at the time of declaration.
- In Python, a variable can contain a value of any type at any time, and the interpreter itself automatically determines the type of the variable based on the assigned value.

A screenshot of a Python IDE window titled 'slika12.py'. The code defines a variable 'x' as an integer (10) and then as a string ('Python'), with corresponding type checks using 'print(type(x))'. The output at the bottom shows the variable's type changing from '<class 'int'>' to '<class 'str'>'.

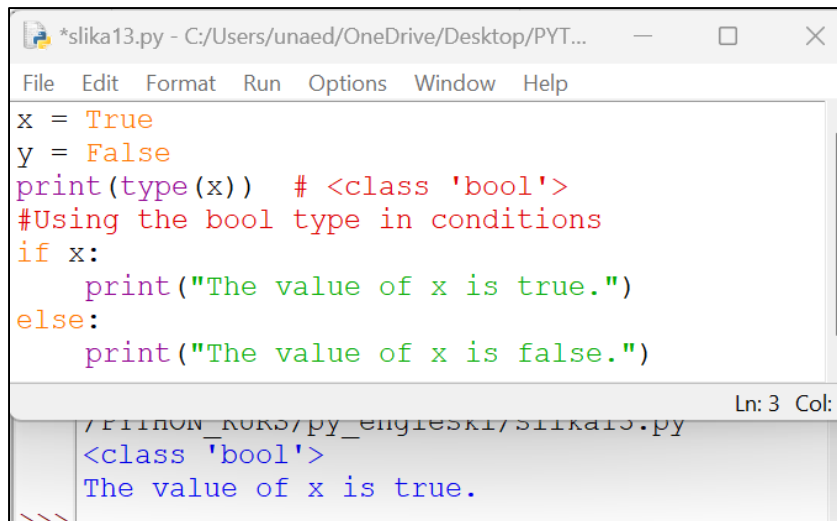
```
slika12.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/p... — Una E Drakulić (un)
File Edit Format Run Options Window Help
x = 10      #x is now an integer (int)
print(type(x)) # <class 'int'>
x = "Python" #x now becomes a string
print(type(x)) # <class 'str'>
Ln: 5 Col: 0
N_KURS/py_engleski/slika12.py =
<class 'int'>
<class 'str'>
>>>
```

Figure 12. Data type conversion example

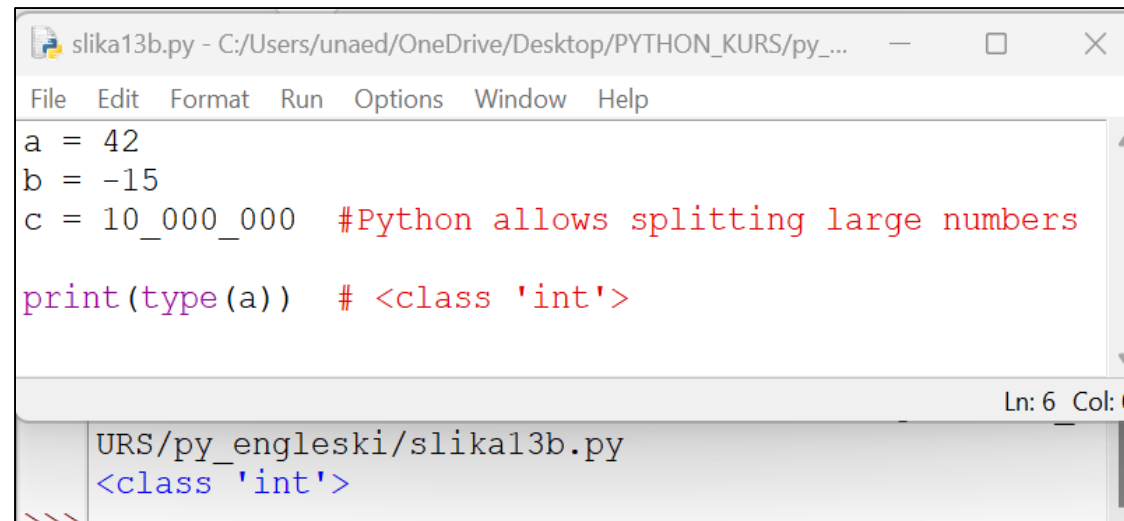


## 3. Data types

- The logical type (*bool*) is used to represent truth values, which can be True or False.
- The integer type (*int*) is used to store whole numbers.



```
*slika13.py - C:/Users/unaed/OneDrive/Desktop/PYT...
File Edit Format Run Options Window Help
x = True
y = False
print(type(x)) # <class 'bool'>
#Using the bool type in conditions
if x:
    print("The value of x is true.")
else:
    print("The value of x is false.")
Ln: 3 Col: 0
PYTHON_KURS/py_engleski/slika13.py
<class 'bool'>
The value of x is true.
///
```



```
slika13b.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_...
File Edit Format Run Options Window Help
a = 42
b = -15
c = 10_000_000 #Python allows splitting large numbers
print(type(a)) # <class 'int'>
Ln: 6 Col: 0
URS/py_engleski/slika13b.py
<class 'int'>
///
```

Figure 13. Example of logical and integer data types

### 3. Data types

- The *float* type is used to store real numbers with a decimal part, including exponential notation. Python provides automatic conversion between int and float when needed.
- A *complex* type of data, consisting of a real and an imaginary part.

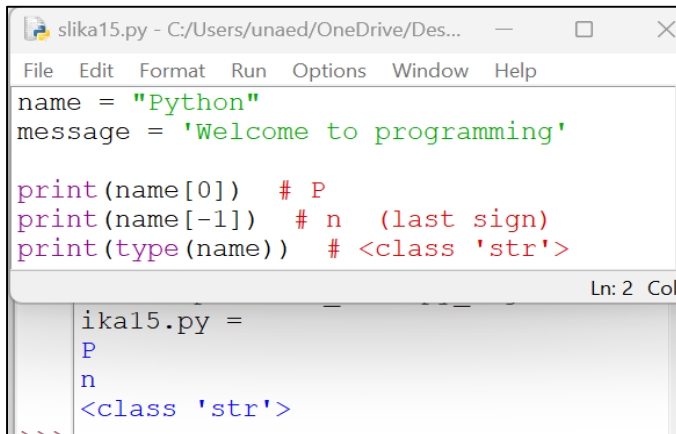
```
slika14.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika14.py (3....
File Edit Format Run Options Window Help
pi = 3.14159
e = 2.71
large_number = 1.2e3 #Exponential notation:: 1.2 * 10^3 = 1200.0
print(type(pi)) # <class 'float'>
Ln: 3 Col: 0
eski/slika14.py
<class 'float'>
>>>
```

```
slika14b.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engle...
File Edit Format Run Options Window Help
z = 3 + 4j #3 is the real part, 4j is the imaginary part
w = complex(2, -3) #Creating a complex number
print(z.real) # 3.0
print(z.imag) # 4.0
print(type(z)) # <class 'complex'>
Ln: 7 Col: 0
/py_engleski/slika14b.py
3.0
4.0
<class 'complex'>
>>>
```

Figure 14. Example of a real and complex data type

## 3. Data types

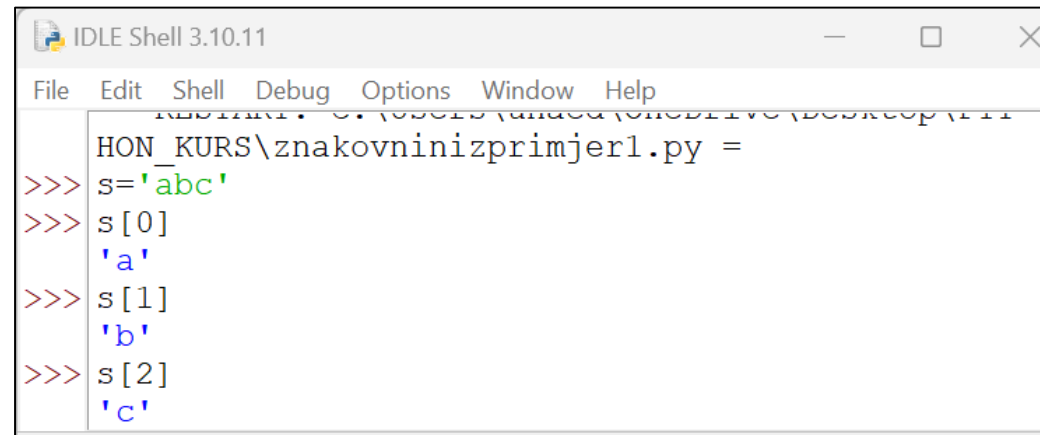
- The *str* type is used to store text data (strings of characters). It can be defined with single ('...') or double ("...") quotes.
- Given that a character string is an ordered sequence of characters, its characters can be accessed via an index, starting from zero. Strings are immutable, which means that they cannot be changed after declaration, but a new string must be created.



```
slika15.py - C:/Users/unaed/OneDrive/Des...
File Edit Format Run Options Window Help
name = "Python"
message = 'Welcome to programming'

print(name[0]) # P
print(name[-1]) # n (last sign)
print(type(name)) # <class 'str'>
```

Figure 15. Example of a character data type



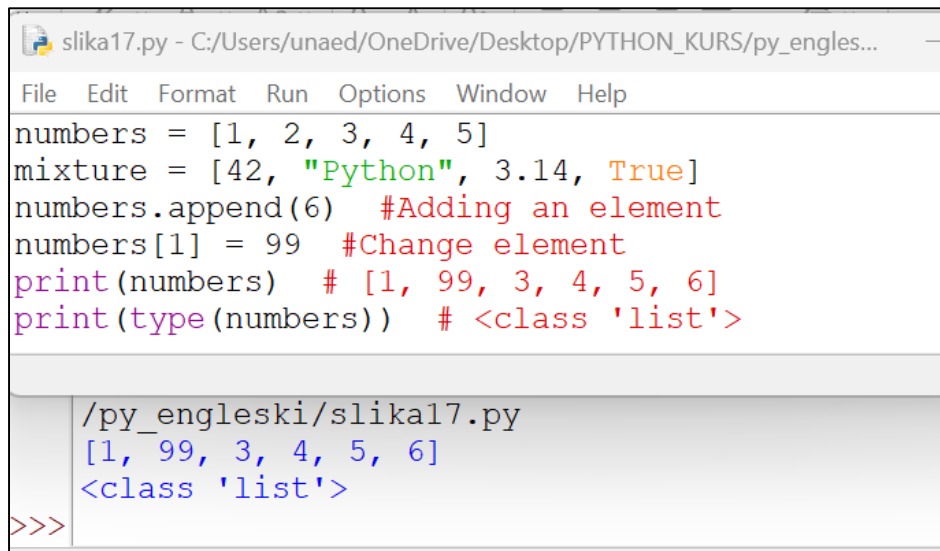
```
IDLE Shell 3.10.11
File Edit Shell Debug Options Window Help
HON_KURS\znakovninizprimjer1.py =
>>> s='abc'
>>> s[0]
'a'
>>> s[1]
'b'
>>> s[2]
'c'
```

Figure 16. Example of accessing characters via index



## 3. Data types

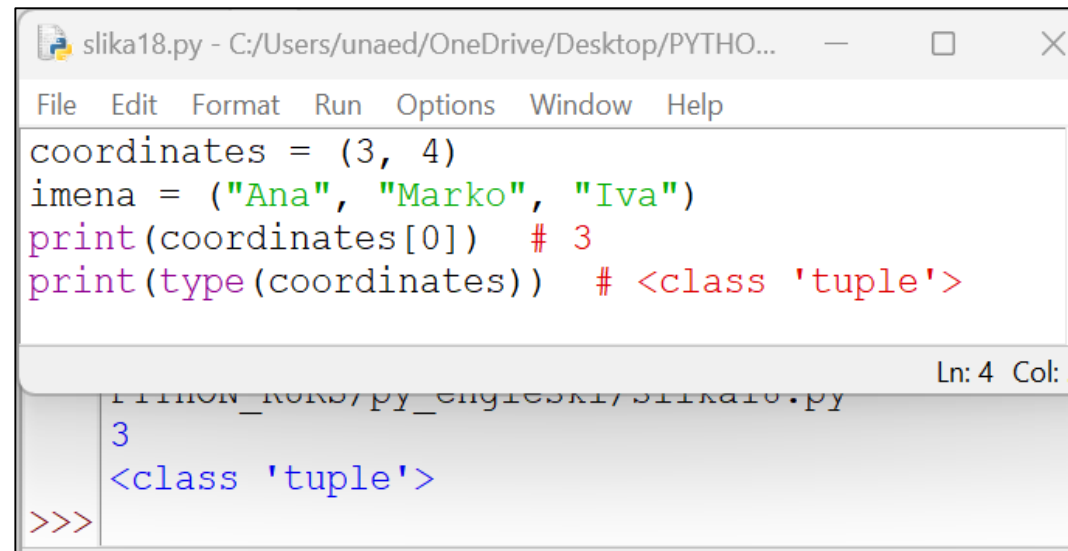
- A *list* is a mutable collection of elements, which can contain different types of data.
- *Tuples* are similar to lists, but they are immutable, which means that they cannot be changed after they are created.



```
slika17.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engles...
File Edit Format Run Options Window Help
numbers = [1, 2, 3, 4, 5]
mixture = [42, "Python", 3.14, True]
numbers.append(6) #Adding an element
numbers[1] = 99 #Change element
print(numbers) # [1, 99, 3, 4, 5, 6]
print(type(numbers)) # <class 'list'>
```

```
/py_engleski/slika17.py
[1, 99, 3, 4, 5, 6]
<class 'list'>
>>>
```

Figure 17. Example of the list data type



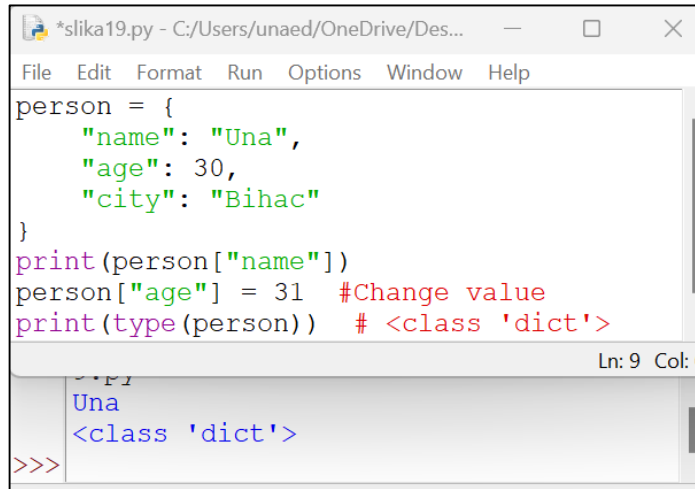
```
slika18.py - C:/Users/unaed/OneDrive/Desktop/PYTHO...
File Edit Format Run Options Window Help
coordinates = (3, 4)
imena = ("Ana", "Marko", "Iva")
print(coordinates[0]) # 3
print(type(coordinates)) # <class 'tuple'>
```

```
Python_KURS/py_engleski/slika18.py
3
<class 'tuple'>
>>>
```

Figure 18. Example of the tuple data type

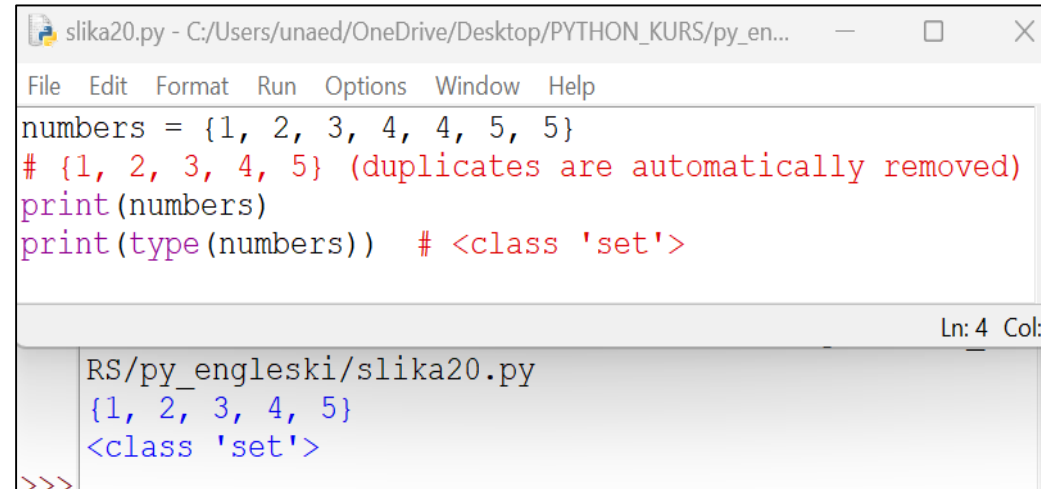
## 3. Data types

- A *dictionary* is a collection of keys and values, where each key must be unique.
- A *set* is a collection of unique elements, which means that it cannot contain duplicates.



```
*slika19.py - C:/Users/unaed/OneDrive/Des...
File Edit Format Run Options Window Help
person = {
    "name": "Una",
    "age": 30,
    "city": "Bihac"
}
print(person["name"])
person["age"] = 31 #Change value
print(type(person)) # <class 'dict'>
Ln: 9 Col: 1
>>>
Una
<class 'dict'>
```

Figure 19. Example of dic data type



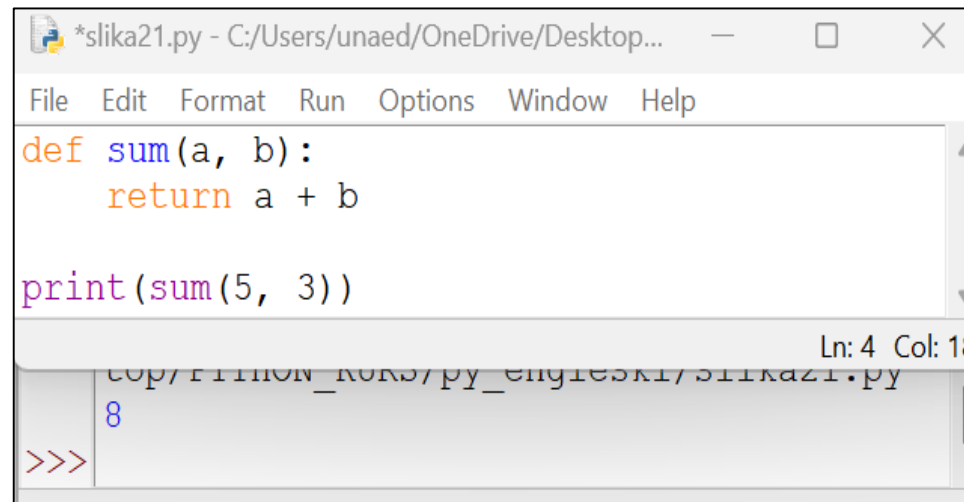
```
slika20.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_en...
File Edit Format Run Options Window Help
numbers = {1, 2, 3, 4, 4, 5, 5}
# {1, 2, 3, 4, 5} (duplicates are automatically removed)
print(numbers)
print(type(numbers)) # <class 'set'>
Ln: 4 Col: 1
RS/py_engleski/slika20.py
{1, 2, 3, 4, 5}
<class 'set'>
>>>
```

Figure 20. Example of set data type



## 4. Functions

- *Standard (defined) functions* - we define these functions using the keyword `def`.
- *Functions without parameters* - a function that does not receive parameters can simply be called by name.



```
File Edit Format Run Options Window Help
def sum(a, b):
    return a + b

print(sum(5, 3))
```

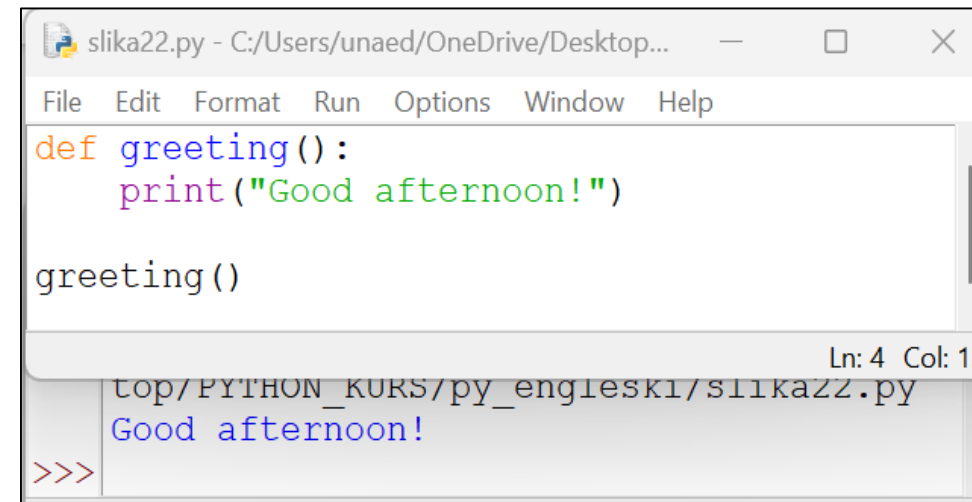
Ln: 4 Col: 18

Cop/PYTHON\_KURS/py\_engleski/slika21.py

8

>>>

Figure 21. Example of a standard function with defined



```
File Edit Format Run Options Window Help
def greeting():
    print("Good afternoon!")

greeting()
```

Ln: 4 Col: 1

Cop/PYTHON\_KURS/py\_engleski/slika22.py

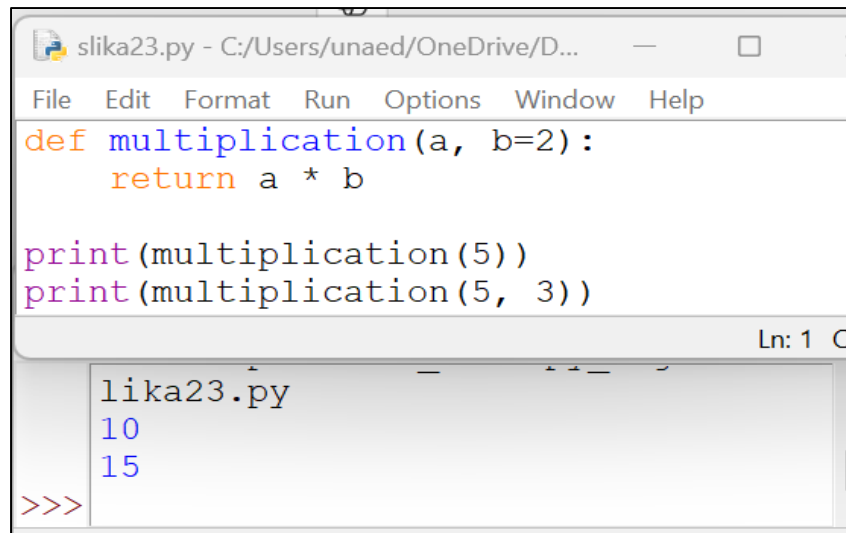
Good afternoon!

>>>

Figure 22. Example of a functions without  
parameters

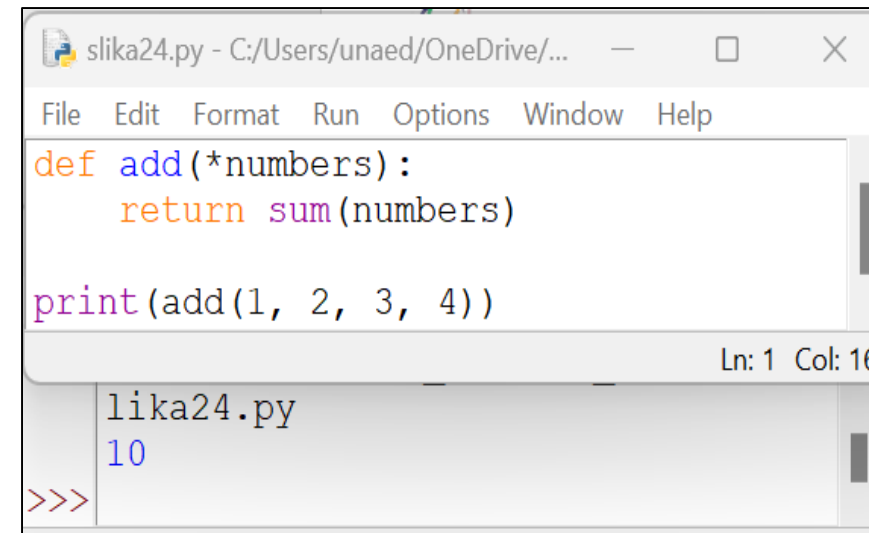
## 4. Functions

- *Functions with default parameter values* - if no argument is passed, the default value is used.
- *Functions with arbitrary number of arguments (\*args)* - allows passing multiple arguments that are treated as tuples.



```
def multiplication(a, b=2):  
    return a * b  
  
print(multiplication(5))  
print(multiplication(5, 3))  
  
lik23.py  
10  
15  
>>>
```

Figure 23. Functions with default parameter values

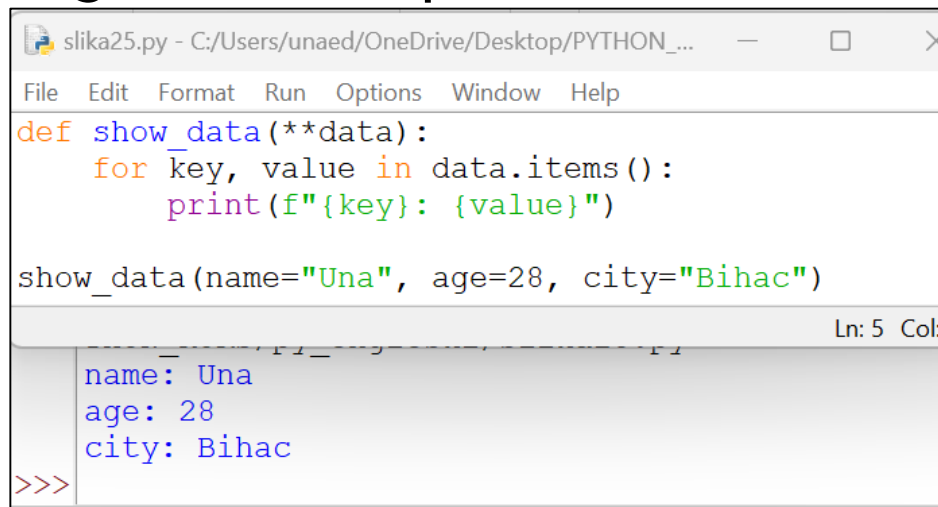


```
def add(*numbers):  
    return sum(numbers)  
  
print(add(1, 2, 3, 4))  
  
lik24.py  
10  
>>>
```

Figure 24. Functions with arbitrary number of arguments(\*args)

## 4. Functions

- *Functions with arbitrary named arguments (\*\*kwargs)* - allows passing named arguments as a dictionary (dict).
- *Lambda (anonymous) functions* - compact functions that are defined using lambda expressions.

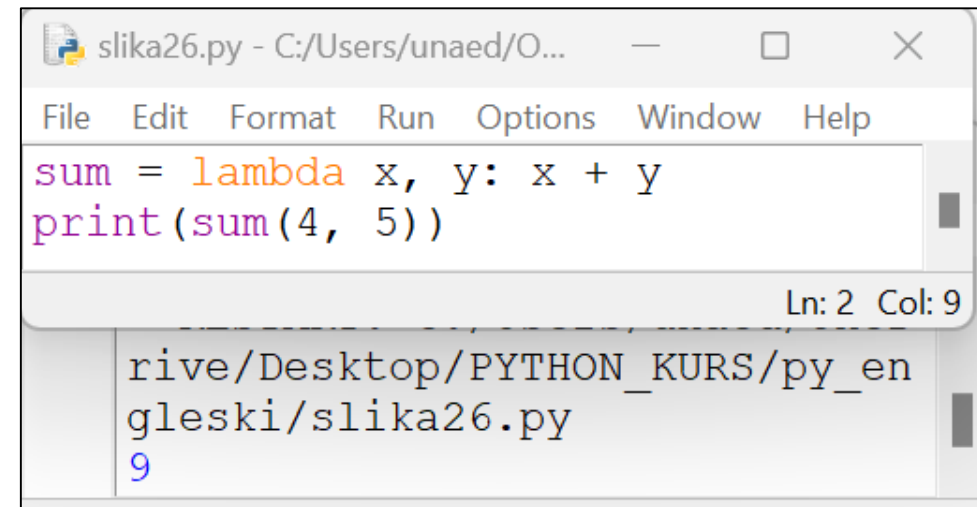


```
slika25.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_...
File Edit Format Run Options Window Help
def show_data(**data):
    for key, value in data.items():
        print(f"{key}: {value}")

show_data(name="Una", age=28, city="Bihac")

Ln: 5 Col:
name: Una
age: 28
city: Bihac
>>>
```

Figure 25. Functions with arbitrary named arguments(\*\*kwargs)



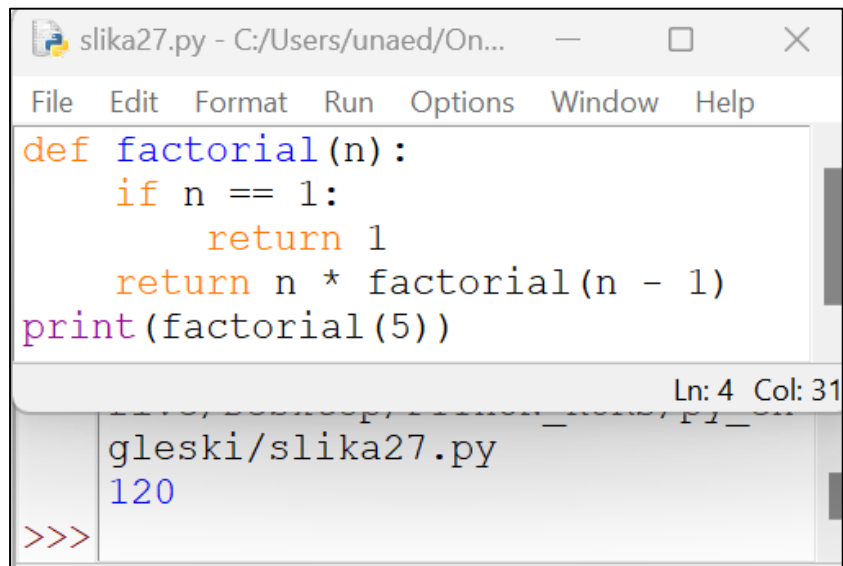
```
slika26.py - C:/Users/unaed/O...
File Edit Format Run Options Window Help
sum = lambda x, y: x + y
print(sum(4, 5))

Ln: 2 Col: 9
rive/Desktop/PYTHON_KURS/py_en
gleski/slika26.py
9
```

Figure 26. Lambda (anonymous) functions

## 4. Functions

- *Recursive functions* - functions that call themselves
- *Functions within functions (nested functions)* - a function defined inside another function.



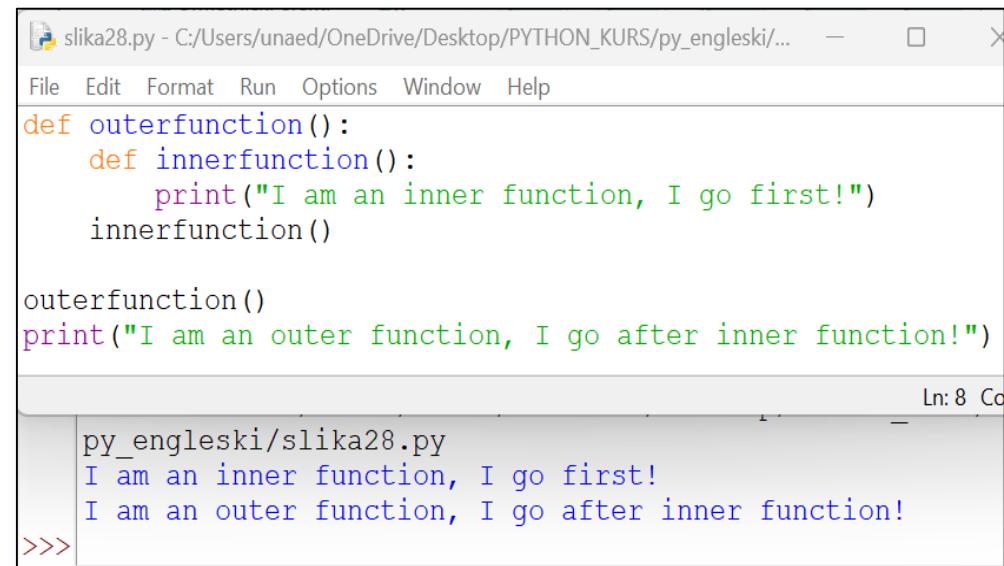
The screenshot shows a Python IDE window titled 'slika27.py'. The code defines a recursive function 'factorial' that takes an argument 'n'. If 'n' is 1, it returns 1. Otherwise, it returns 'n' multiplied by 'factorial(n - 1)'. The function is called with 'factorial(5)'. The output in the console is '120'.

```
def factorial(n):  
    if n == 1:  
        return 1  
    return n * factorial(n - 1)  
print(factorial(5))
```

Ln: 4 Col: 31

gleski/slika27.py  
120  
>>>

Figure 27. Recursive functions



The screenshot shows a Python IDE window titled 'slika28.py'. The code defines an 'outerfunction' which contains an 'innerfunction'. The 'innerfunction' prints a message and calls itself. The 'outerfunction' is called, which then calls 'innerfunction'. The output in the console shows the inner function's message first, followed by the outer function's message.

```
def outerfunction():  
    def innerfunction():  
        print("I am an inner function, I go first!")  
        innerfunction()  
  
outerfunction()  
print("I am an outer function, I go after inner function!")
```

Ln: 8 Co

py\_engleski/slika28.py  
I am an inner function, I go first!  
I am an outer function, I go after inner function!  
>>>

Figure 28. Nested functions

# 4. Functions



The screenshot displays a Python IDE with two windows. The main window, titled '\*slika29.py - C:\Users\unaed\OneDrive\Desktop\PYTHON\_KURS\py\_engleski\slika29.py (3)', contains a Python script. The script defines a 'calculator' function that takes an 'operation' string as an argument. Inside 'calculator', there are four nested functions: 'add', 'subtract', 'multiply', and 'divide'. Each nested function performs its respective arithmetic operation and returns the result. The 'calculator' function uses a series of 'if' and 'elif' statements to call the appropriate nested function based on the input 'operation'. If the operation is not recognized, it returns 'None'. Below the function definitions, the script demonstrates the use of these functions: it calls 'calculator("add")' with arguments 10 and 5, prints the result (15), and then calls 'calculator("divide")' with arguments 20 and 4, printing the result (5.0). The second window, titled 'IDLE Shell 3.10.11', shows the execution of the script. It displays the Python version and architecture, followed by the prompt 'Type "help", "copyright", "credits" or "license()" for more'. The user has entered '== RESTART: C:\Users\unaed\OneDrive\Desktop\PYTHON\_KURS\py\_e', and the shell has printed the results of the function calls: 'result: 15' and 'result: 5.0'.

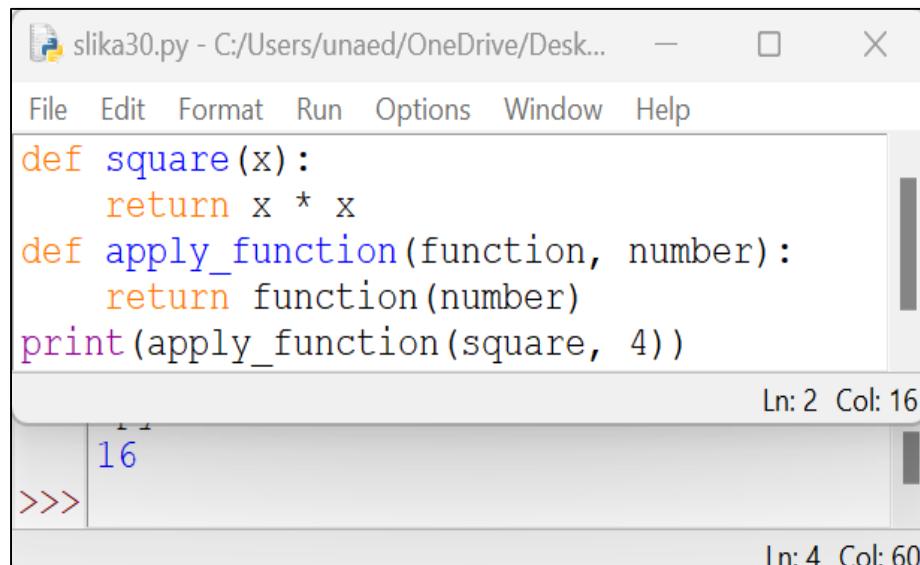
```
*slika29.py - C:\Users\unaed\OneDrive\Desktop\PYTHON_KURS\py_engleski\slika29.py (3)
File Edit Format Run Options Window Help
def calculator(operation):
    """The main function"""
    def add(a, b):
        return a + b
    def subtract(a, b):
        return a - b
    def multiply(a, b):
        return a * b
    def divide(a, b):
        if b == 0:
            return "Error: Division by zero is not allowed."
        return a / b
    if operation == "add":
        return add
    elif operation == "subtract":
        return subtract
    elif operation == "multiply":
        return multiply
    elif operation == "divide":
        return divide
    else:
        return None #Invalid operation
#Selects an operation
operation = calculator("add")
if operation:
    result = operation(10, 5)
    print("result:", result)
operation = calculator("divide")
if operation:
    result = operation(20, 4)
    print("result:", result)
```

IDLE Shell 3.10.11
File Edit Shell Debug Options Window Help
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17 (AMD64)) on win32
Type "help", "copyright", "credits" or "license()" for more
>>> == RESTART: C:\Users\unaed\OneDrive\Desktop\PYTHON\_KURS\py\_e
result: 15
result: 5.0
>>>

Figure 29. Nested functions

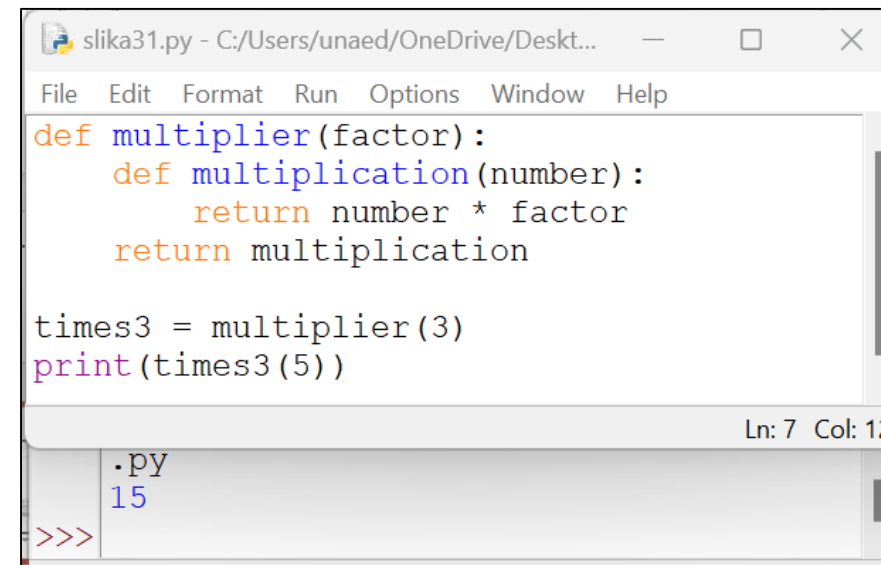
## 4. Functions

- *Functions as arguments to other functions* - functions that can be passed as arguments to other functions.
- *Functions that return other functions.*



```
slika30.py - C:/Users/unaed/OneDrive/Desk...
File Edit Format Run Options Window Help
def square(x):
    return x * x
def apply_function(function, number):
    return function(number)
print(apply_function(square, 4))
Ln: 2 Col: 16
16
>>>
Ln: 4 Col: 60
```

Figure 30. *Functions as arguments to other functions*

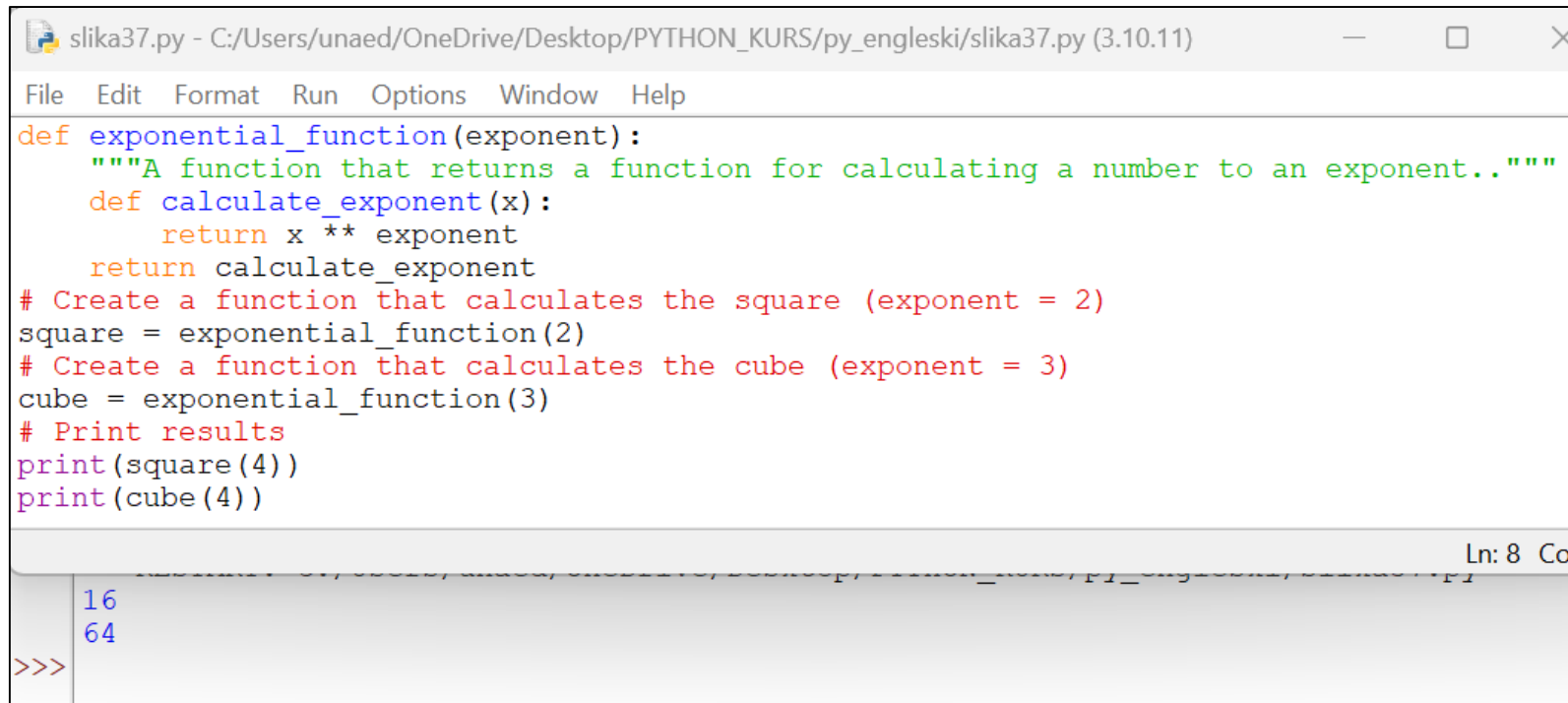


```
slika31.py - C:/Users/unaed/OneDrive/Desk...
File Edit Format Run Options Window Help
def multiplier(factor):
    def multiplication(number):
        return number * factor
    return multiplication

times3 = multiplier(3)
print(times3(5))
Ln: 7 Col: 12
.py
15
>>>
```

Figure 31. *Functions that return other functions*

# 4. Functions



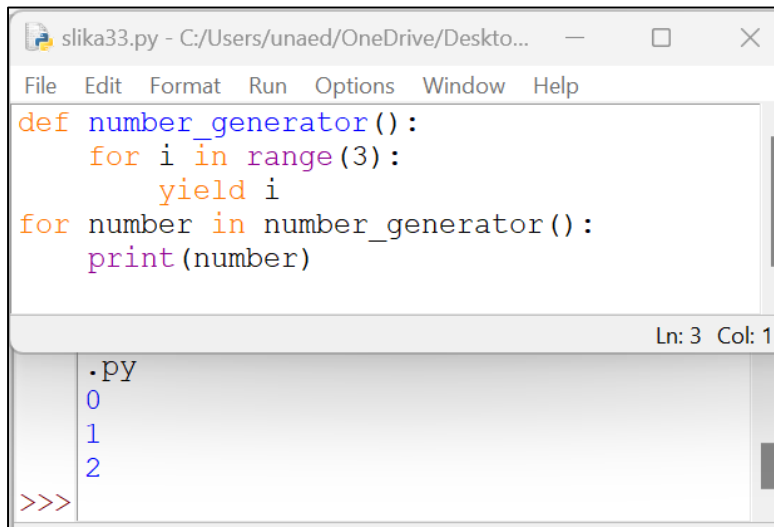
```
slika37.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika37.py (3.10.11)
File Edit Format Run Options Window Help
def exponential_function(exponent):
    """A function that returns a function for calculating a number to an exponent.."""
    def calculate_exponent(x):
        return x ** exponent
    return calculate_exponent
# Create a function that calculates the square (exponent = 2)
square = exponential_function(2)
# Create a function that calculates the cube (exponent = 3)
cube = exponential_function(3)
# Print results
print(square(4))
print(cube(4))
Ln: 8 Co
16
64
>>>
```

Figure 32. *Functions that return other functions*



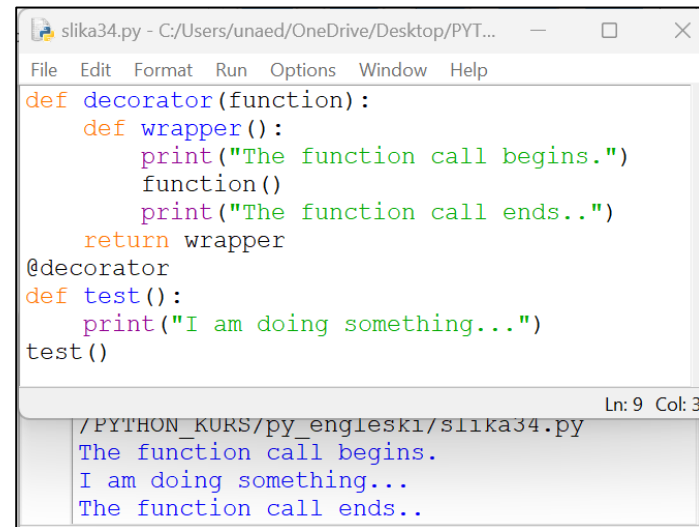
## 4. Functions

- *Function generator (yield)* - instead of return, they use yield to create an iterator.
- *Decorator functions* - functions that change the behavior of other functions.



```
slika33.py - C:/Users/unaed/OneDrive/Desktop...
File Edit Format Run Options Window Help
def number_generator():
    for i in range(3):
        yield i
for number in number_generator():
    print(number)
Ln: 3 Col: 15
.py
0
1
2
>>>
```

Figure 33. *Function generator*



```
slika34.py - C:/Users/unaed/OneDrive/Desktop/PYT...
File Edit Format Run Options Window Help
def decorator(function):
    def wrapper():
        print("The function call begins.")
        function()
        print("The function call ends..")
    return wrapper
@decorator
def test():
    print("I am doing something...")
test()
Ln: 9 Col: 31
/PYTHON_KURS/py_engleski/slika34.py
The function call begins.
I am doing something...
The function call ends..
```

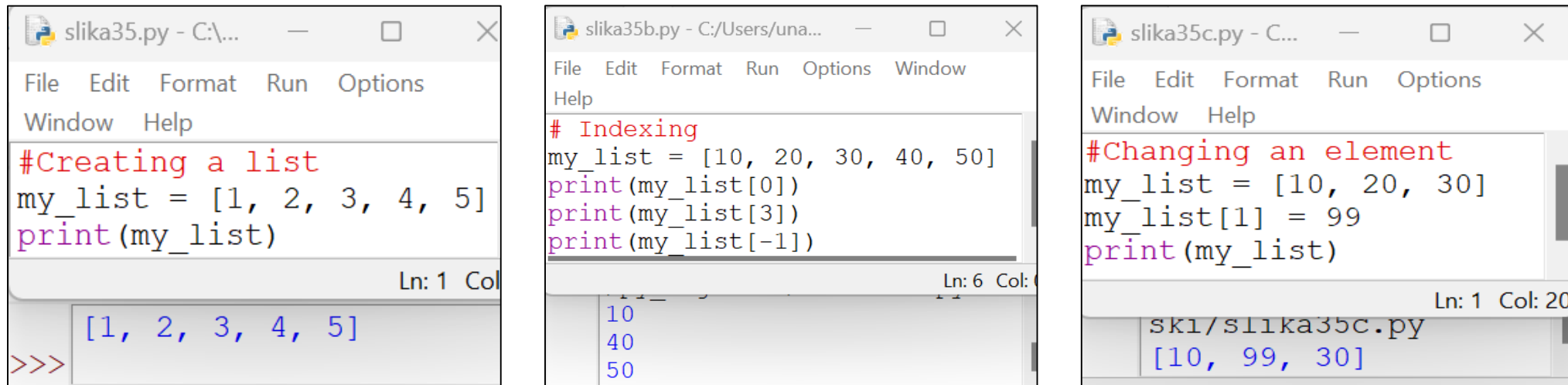
Figure 34. *Decorator functions*





## 5. Lists

- A list can be created using square brackets [ ], and elements within the list are separated by commas.
- Indexing of elements within the list starts from 0.
- Elements within the list can be changed.



The figure consists of three side-by-side screenshots of a Python IDE window, each showing a different list operation. The first screenshot, titled 'slika35.py', shows the creation of a list 'my\_list' with the values [1, 2, 3, 4, 5] and its printing. The second screenshot, titled 'slika35b.py', shows the indexing of the list 'my\_list' with values [10, 20, 30, 40, 50], printing the first element, the element at index 3, and the last element. The third screenshot, titled 'slika35c.py', shows the modification of the list 'my\_list' from [10, 20, 30] to [10, 99, 30] by changing the element at index 1, and then printing the updated list.

```
#Creating a list
my_list = [1, 2, 3, 4, 5]
print(my_list)

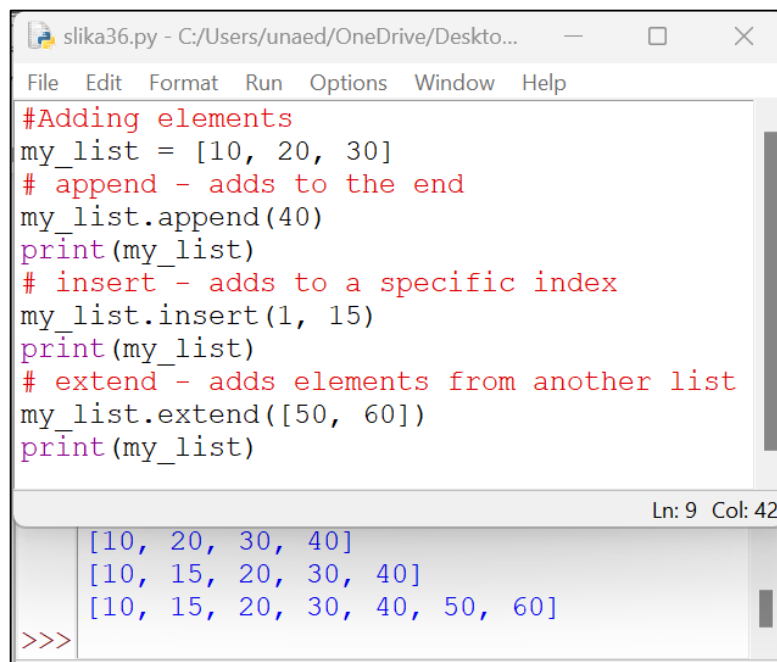
# Indexing
my_list = [10, 20, 30, 40, 50]
print(my_list[0])
print(my_list[3])
print(my_list[-1])

#Changing an element
my_list = [10, 20, 30]
my_list[1] = 99
print(my_list)
```

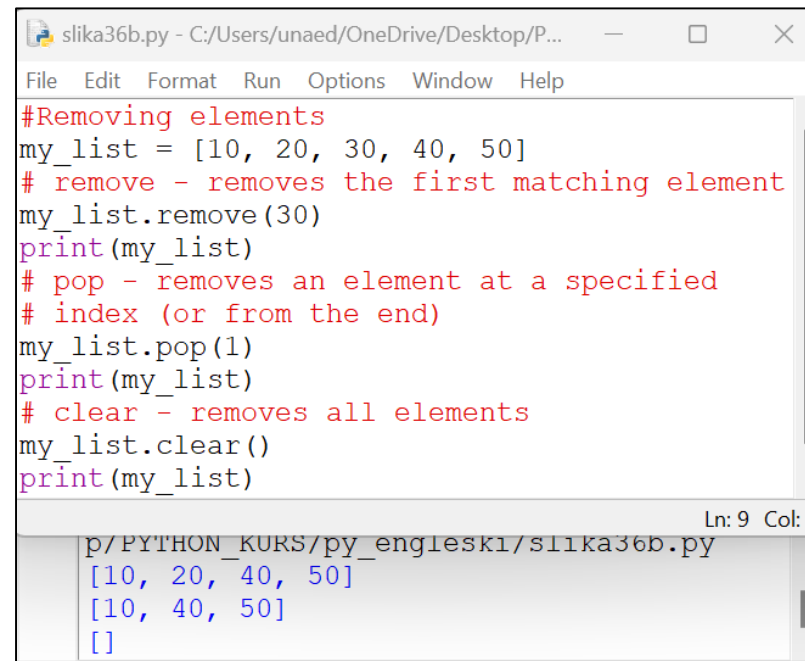
Figure 35. List creation; List indexing; Changing an element in a list

## 5. Lists

- We can add new elements to the list, and we can remove, search and count existing elements.
- We can sort, copy and merge lists.



```
slika36.py - C:/Users/unaed/OneDrive/Desktop/...
File Edit Format Run Options Window Help
#Adding elements
my_list = [10, 20, 30]
# append - adds to the end
my_list.append(40)
print(my_list)
# insert - adds to a specific index
my_list.insert(1, 15)
print(my_list)
# extend - adds elements from another list
my_list.extend([50, 60])
print(my_list)
Ln: 9 Col: 42
[10, 20, 30, 40]
[10, 15, 20, 30, 40]
[10, 15, 20, 30, 40, 50, 60]
>>>
```

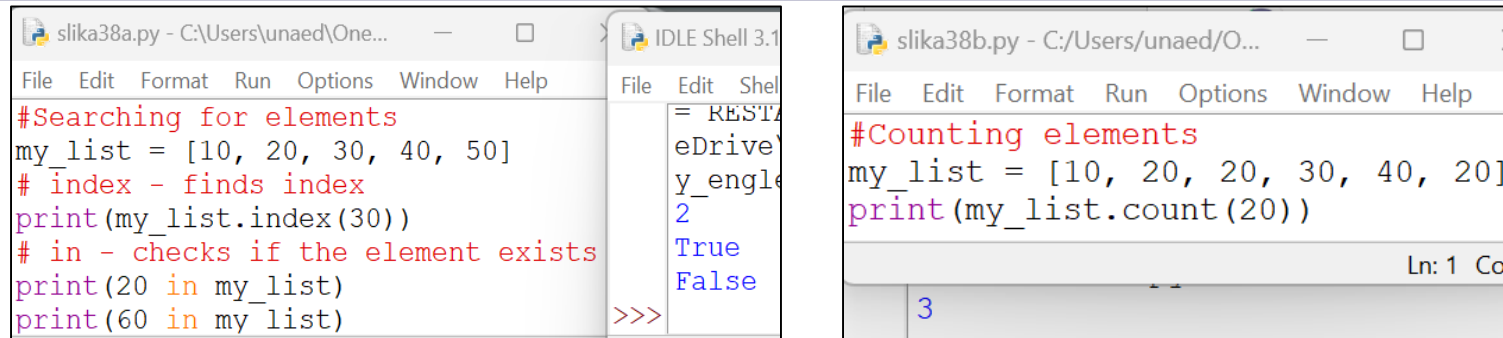


```
slika36b.py - C:/Users/unaed/OneDrive/Desktop/P...
File Edit Format Run Options Window Help
#Removing elements
my_list = [10, 20, 30, 40, 50]
# remove - removes the first matching element
my_list.remove(30)
print(my_list)
# pop - removes an element at a specified
# index (or from the end)
my_list.pop(1)
print(my_list)
# clear - removes all elements
my_list.clear()
print(my_list)
Ln: 9 Col: 42
p/PYTHON_KURS/py_engleski/slika36b.py
[10, 20, 40, 50]
[10, 40, 50]
[]
```

Figure 36. Adding and removing elements in lists



# 5. Lists



The first screenshot shows a Python IDLE Shell window with the following code and output:

```
#Searching for elements
my_list = [10, 20, 30, 40, 50]
# index - finds index
print(my_list.index(30))
# in - checks if the element exists
print(20 in my_list)
print(60 in my_list)
```

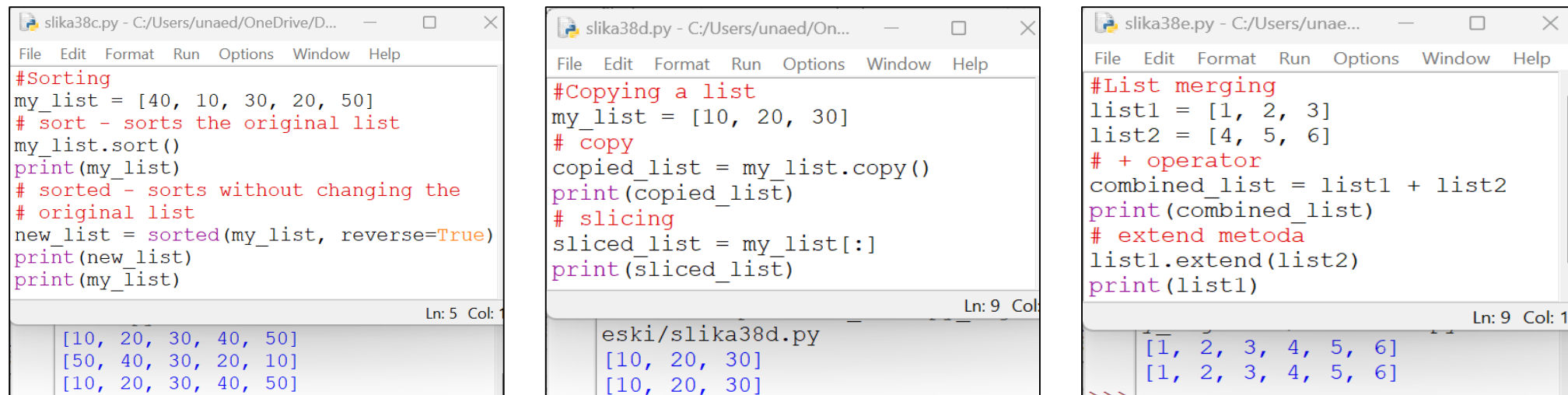
The output shows the index of 30 is 2, 20 is in the list (True), and 60 is not in the list (False).

The second screenshot shows a Python IDLE Shell window with the following code and output:

```
#Counting elements
my_list = [10, 20, 20, 30, 40, 20]
print(my_list.count(20))
```

The output shows the count of 20 is 3.

Figure 37. Searching and counting elements in lists



The first screenshot shows a Python IDLE Shell window with the following code and output:

```
#Sorting
my_list = [40, 10, 30, 20, 50]
# sort - sorts the original list
my_list.sort()
print(my_list)
# sorted - sorts without changing the
# original list
new_list = sorted(my_list, reverse=True)
print(new_list)
print(my_list)
```

The output shows the sorted list [10, 20, 30, 40, 50], the reverse sorted list [50, 40, 30, 20, 10], and the original list [10, 20, 30, 40, 50].

The second screenshot shows a Python IDLE Shell window with the following code and output:

```
#Copying a list
my_list = [10, 20, 30]
# copy
copied_list = my_list.copy()
print(copied_list)
# slicing
sliced_list = my_list[:]
print(sliced_list)
```

The output shows the copied list [10, 20, 30] and the sliced list [10, 20, 30].

The third screenshot shows a Python IDLE Shell window with the following code and output:

```
#List merging
list1 = [1, 2, 3]
list2 = [4, 5, 6]
# + operator
combined_list = list1 + list2
print(combined_list)
# extend metoda
list1.extend(list2)
print(list1)
```

The output shows the combined list [1, 2, 3, 4, 5, 6] and the extended list [1, 2, 3, 4, 5, 6].

Figure 38. Sorting, copying and merging lists

## 5. Lists

- *List Comprehension (or list generation)* is a way to create new lists using a shorter and more efficient syntax compared to classic for loops.
- It is used for: generating new lists using a for loop, filtering data using an if condition, transforming elements using an expression within a list.

```
File Edit Format Run Options Window Help
#Classic for loop
numbers = []
for x in range(5):
    numbers.append(x**2)
print(numbers)
# List Comprehension
square_numbers = [x**2 for x in range(5)]
print(square_numbers)

Ln: 8 Col: 2

[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
>>>
```

Figure 39. Example of generating a list of numbers

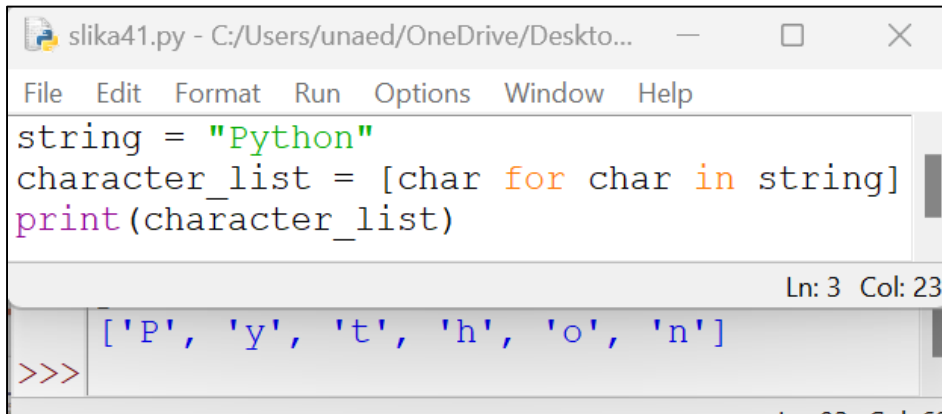
```
File Edit Format Run Options Window Help
#Classic for loop
even_numbers = []
for x in range(10):
    if x % 2 == 0:
        even_numbers.append(x)
print(even_numbers)
# List Comprehension
even_numbers = [x for x in range(10) if x % 2 == 0]
print(even_numbers)

Ln: 7 Col:

[0, 2, 4, 6, 8]
[0, 2, 4, 6, 8]
```

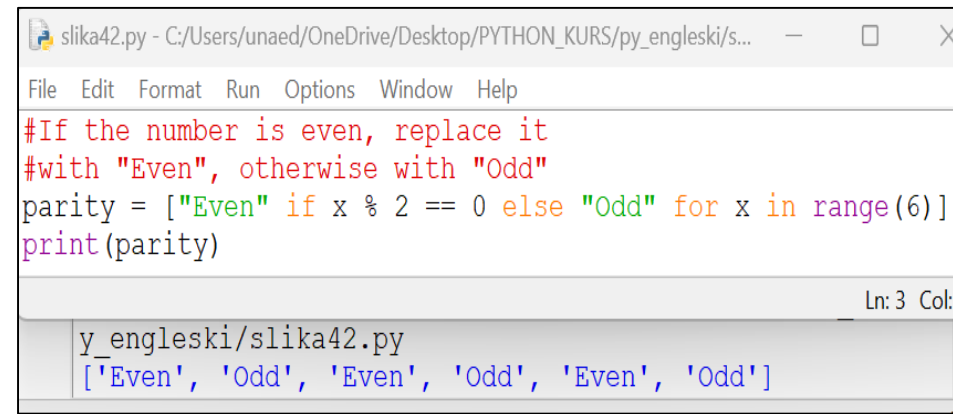
Figure 40. Example of filtering a list of numbers

# 5. Lists



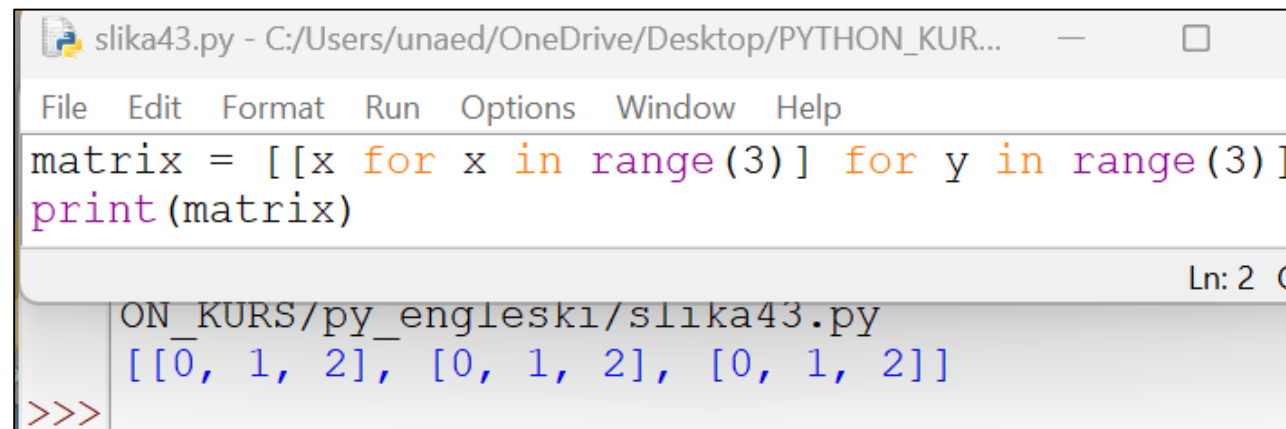
```
slika41.py - C:/Users/unaed/OneDrive/Deskto...
File Edit Format Run Options Window Help
string = "Python"
character_list = [char for char in string]
print(character_list)
Ln: 3 Col: 23
['P', 'y', 't', 'h', 'o', 'n']
>>>
```

Figure 41. *Example of string conversion in the character sheet*



```
slika42.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/s...
File Edit Format Run Options Window Help
#If the number is even, replace it
#with "Even", otherwise with "Odd"
parity = ["Even" if x % 2 == 0 else "Odd" for x in range(6)]
print(parity)
Ln: 3 Col:
y_engleski/slika42.py
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

Figure 42. *Example of replacing if – else in Sheet Comprehension*



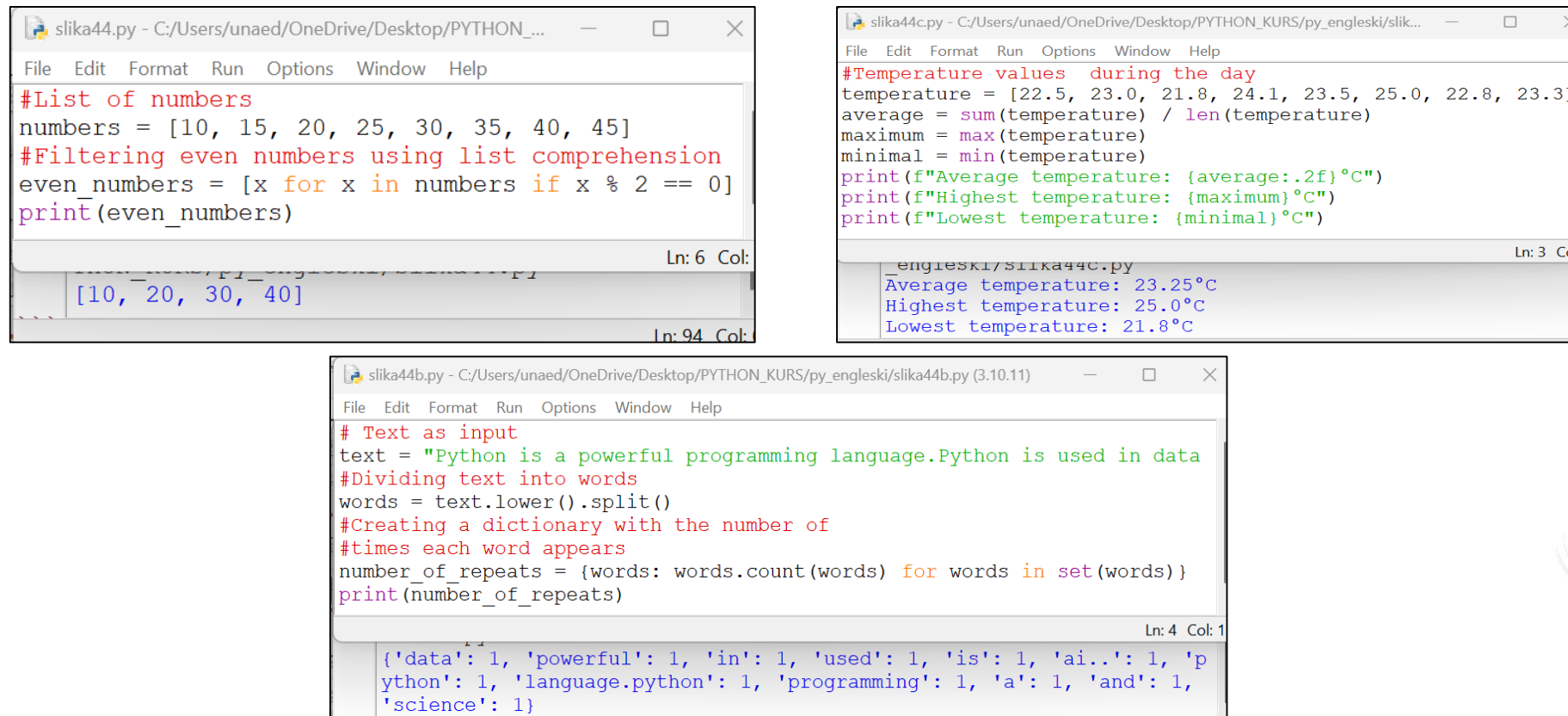
```
slika43.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KUR...
File Edit Format Run Options Window Help
matrix = [[x for x in range(3)] for y in range(3)]
print(matrix)
Ln: 2 Col:
ON_KURS/py_engleski/slika43.py
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
>>>
```

Figure 43. *Example for Nested List Comprehension*



# 5. Lists

- Lists are very useful in Python and are used in a variety of situations, from data processing to working with and analyzing databases.



The figure consists of three screenshots of a Python IDE, likely PyCharm, showing different list operations. The first screenshot shows a list of numbers being filtered for even values using list comprehension. The second screenshot shows a list of temperature values being analyzed for average, maximum, and minimum values. The third screenshot shows a string of text being split into words and counted to create a dictionary of word frequencies.

```
slika44.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_...  
File Edit Format Run Options Window Help  
#List of numbers  
numbers = [10, 15, 20, 25, 30, 35, 40, 45]  
#Filtering even numbers using list comprehension  
even_numbers = [x for x in numbers if x % 2 == 0]  
print(even_numbers)  
Ln: 6 Col: 1  
[10, 20, 30, 40]  
Ln: 94 Col: 1
```

```
slika44c.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slik...  
File Edit Format Run Options Window Help  
#Temperature values during the day  
temperature = [22.5, 23.0, 21.8, 24.1, 23.5, 25.0, 22.8, 23.3]  
average = sum(temperature) / len(temperature)  
maximum = max(temperature)  
minimal = min(temperature)  
print(f"Average temperature: {average:.2f}°C")  
print(f"Highest temperature: {maximum}°C")  
print(f"Lowest temperature: {minimal}°C")  
Ln: 3 Col: 1  
engleski/slika44c.py  
Average temperature: 23.25°C  
Highest temperature: 25.0°C  
Lowest temperature: 21.8°C
```

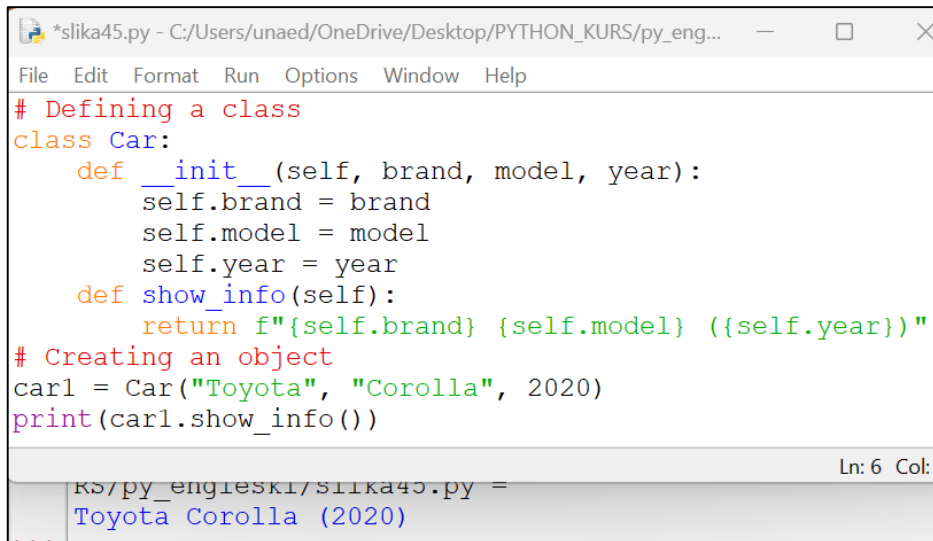
```
slika44b.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika44b.py (3.10.11)  
File Edit Format Run Options Window Help  
# Text as input  
text = "Python is a powerful programming language. Python is used in data  
#Dividing text into words  
words = text.lower().split()  
#Creating a dictionary with the number of  
#times each word appears  
number_of_repeats = {words: words.count(words) for words in set(words)}  
print(number_of_repeats)  
Ln: 4 Col: 1  
{'data': 1, 'powerful': 1, 'in': 1, 'used': 1, 'is': 1, 'ai..': 1, 'p  
ython': 1, 'language.python': 1, 'programming': 1, 'a': 1, 'and': 1,  
'science': 1}
```

Figure 44. Filtering numbers; Working with text in a list, Data analysis



## 6. Classes

- Classes are the foundation of object-oriented programming (OOP) in Python. They enable the organization of code into objects that contain data (attributes) and methods (functions that work with that data).
- A class is defined using the class keyword, and objects are created from it.

A screenshot of a Python IDE window titled '\*slika45.py - C:/Users/unaed/OneDrive/Desktop/PYTHON\_KURS/py\_eng...'. The window contains the following Python code:

```
# Defining a class
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year
    def show_info(self):
        return f"{self.brand} {self.model} ({self.year})"
# Creating an object
car1 = Car("Toyota", "Corolla", 2020)
print(car1.show_info())
```

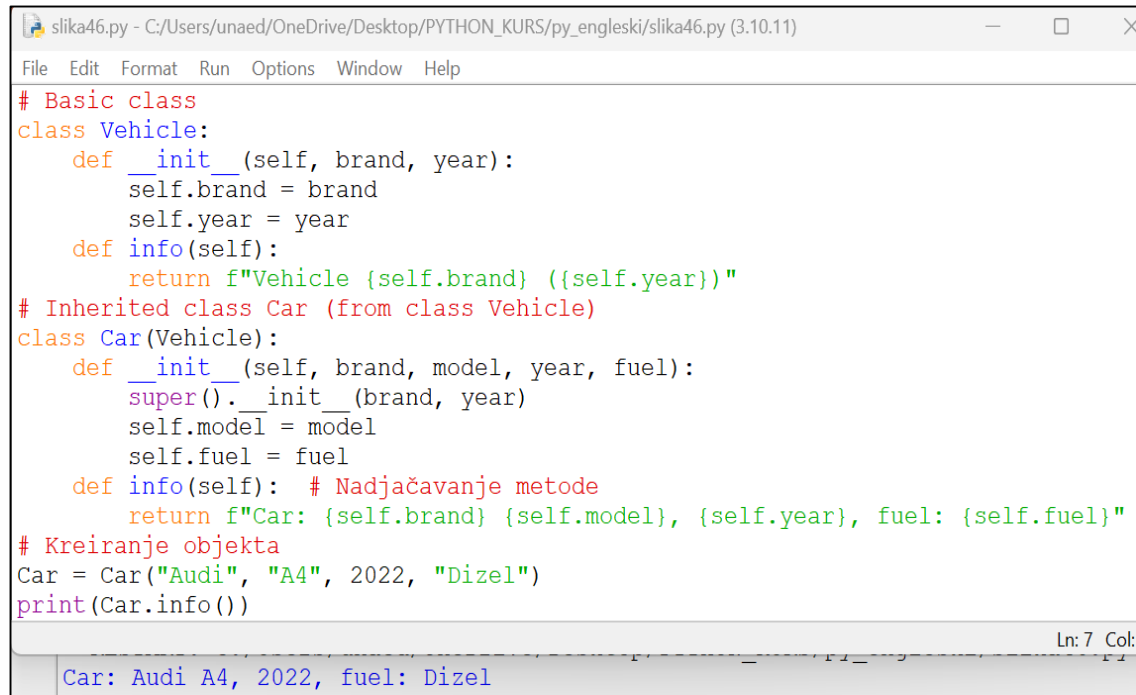
The status bar at the bottom indicates 'Ln: 6 Col:'. Below the code editor, the output of the program is displayed: 'Toyota Corolla (2020)'.

Figure 45. Example of object creation

- `__init__` is a constructor that initializes attributes when creating an object;
- `self` indicates a reference to a class instance;
- `show_info()` returns a string with information about the car.

## 6. Classes

- Inheritance allows the creation of a new class that takes over the properties of an existing class, thus avoiding code duplication.



```
slika46.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika46.py (3.10.11)
File Edit Format Run Options Window Help
# Basic class
class Vehicle:
    def __init__(self, brand, year):
        self.brand = brand
        self.year = year
    def info(self):
        return f"Vehicle {self.brand} ({self.year})"
# Inherited class Car (from class Vehicle)
class Car(Vehicle):
    def __init__(self, brand, model, year, fuel):
        super().__init__(brand, year)
        self.model = model
        self.fuel = fuel
    def info(self): # Nadjačavanje metode
        return f"Car: {self.brand} {self.model}, {self.year}, fuel: {self.fuel}"
# Kreiranje objekta
Car = Car("Audi", "A4", 2022, "Dizel")
print(Car.info())
Ln: 7 Col:
Car: Audi A4, 2022, fuel: Dizel
```

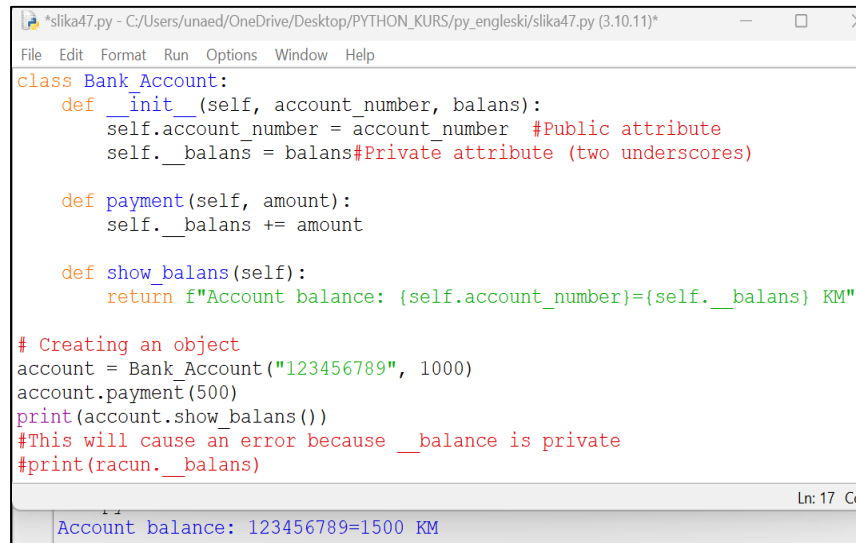
- Car class inherits the Vehicle class, which means that it inherits its methods and attributes;
- `super().__init__(brand, year)` calls the base class constructor;
- `info()` method has been overridden and now gives more specific output.

Figure 46. Example of inheritance



## 6. Classes

- Encapsulation (Private and Protected attributes) restricts access to data within the class, protecting it from external modification.
- Polymorphism (methods with the same name in different classes), enables the use of the same methods in different classes with different behavior.



```
*slika47.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika47.py (3.10.11)*
File Edit Format Run Options Window Help
class Bank Account:
    def __init__(self, account_number, balans):
        self.account_number = account_number #Public attribute
        self.__balans = balans#Private attribute (two underscores)

    def payment(self, amount):
        self.__balans += amount

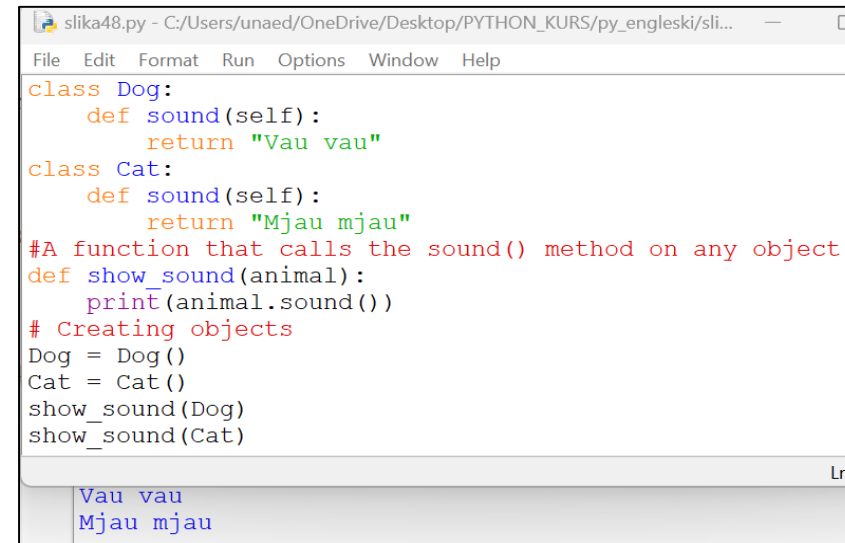
    def show_balans(self):
        return f"Account balance: {self.account_number}={self.__balans} KM"

# Creating an object
account = Bank_Account("123456789", 1000)
account.payment(500)
print(account.show_balans())
#This will cause an error because __balance is private
#print(racun.__balans)
```

Ln: 17 Co

Account balance: 123456789=1500 KM

Figure 47. Example of encapsulation



```
slika48.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/sli...
File Edit Format Run Options Window Help
class Dog:
    def sound(self):
        return "Vau vau"
class Cat:
    def sound(self):
        return "Mjau mjau"
#A function that calls the sound() method on any object
def show_sound(animal):
    print(animal.sound())
# Creating objects
Dog = Dog()
Cat = Cat()
show_sound(Dog)
show_sound(Cat)
```

Ln:

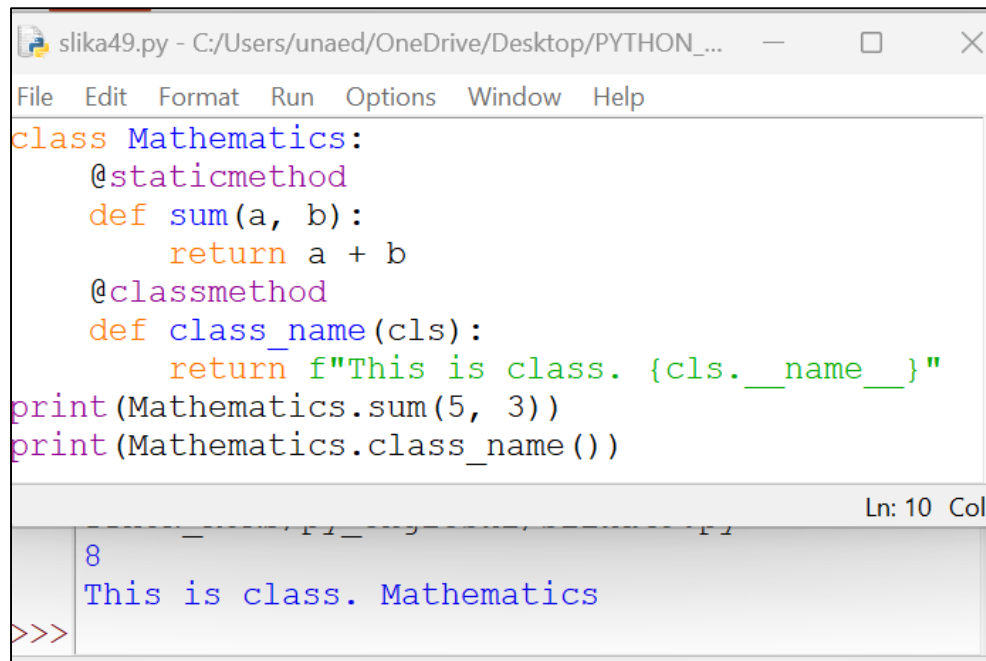
Vau vau  
Mjau mjau

Figure 48. Example of polymorphism



# 6 Classes

- Static methods do not use self and do not depend on an instance of the class.
- Class methods use cls and work with all instances of the class.



```
slika49.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_...
File Edit Format Run Options Window Help
class Mathematics:
    @staticmethod
    def sum(a, b):
        return a + b
    @classmethod
    def class_name(cls):
        return f"This is class. {cls.__name__}"
print(Mathematics.sum(5, 3))
print(Mathematics.class_name())
Ln: 10 Col:
8
This is class. Mathematics
>>>
```

- @staticmethod indicates a method that does not depend on the instance (self);
- @classmethod uses cls, which allows working with a class, not just an instance.



Figure 49. Example of static method and class method

## 6. Classes

- Classes are used in a variety of applications, from web programming to artificial intelligence and data processing.

```
slika50.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika5...
File Edit Format Run Options Window Help
from django.db import models
class Korisnik(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    registration_date = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"{self.name} - {self.email}"
```

Figure 50. Example of defining Django

```
*slika51.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika51.py (...
File Edit Format Run Options Window Help
import tensorflow as tf
from tensorflow import keras
class Network(keras.Model):
    def __init__(self):
        super(Network, self).__init__()
        self.layer1 = keras.layers.Dense(64, activation='relu')
        self.layer2 = keras.layers.Dense(10, activation='softmax')
    def call(self, ulaz):
        x = self.layer1(input)
        return self.layer2(x)
# Model creation
model = Network()
```

Figure 51. Example of defining a neural network

```
*slika52.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_...
File Edit Format Run Options Window Help
import pandas as pd
class Data_Analysis:
    def __init__(self, file):
        self.data = pd.read_csv(file)
    def show_info(self):
        return self.data.info()
    def show_first_n(self, n=5):
        return self.data.head(n)
# Creating object
analiza = Data_Analysis("data.csv")
print(analiza.show_info())
print(analiza.show_first_n(3))
```

Figure 52. Example of data processing

```
import requests
from bs4 import BeautifulSoup
class WebScraper:
    def __init__(self, url):
        self.url = url
        self.content = self.download_page()
    def download_page(self):
        answer = requests.get(self.url)
        return BeautifulSoup(answer.text, 'html.parser')
    def find_titles(self):
        title = self.content.find_all('h2')
        return [title.text for title in title]
# Kreiranje objekta
scraper = WebScraper("https://web.tfb.unbi.ba/elektrotehnicki-odsjek/")
print(scraper.find_titles())
Ln: 3 Col:
['Elektrotehnicki odsjek', 'Vijesti', 'O odsjeku', 'Voditelj odsje
ka', 'Nastavno osoblje', 'Studijski programi', 'Galerija']
```

Figure 53. Example of downloading data



## 6. Classes

- Classes are used to build video games, simulations, and reinforcement learning systems. The most commonly used packages are Pygame, OpenAI Gym.

```
import pygame
# Initialization Pygame-a
pygame.init()
# Setting window dimensions
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Pong")
# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
# Blade and ball dimensions
PADDLE_WIDTH, PADDLE_HEIGHT = 10, 100
BALL_SIZE = 10
# Starting positions of the blades
paddle1 = pygame.Rect(50, HEIGHT//2 - PADDLE_HEIGHT//2, PADDLE_WIDTH, PADDLE_HEIGHT)
paddle2 = pygame.Rect(WIDTH - 50 - PADDLE_WIDTH, HEIGHT//2 - PADDLE_HEIGHT//2, PADDLE_WIDTH, PADDLE_HEIGHT)
# Initial ball position and speed
ball = pygame.Rect(WIDTH//2 - BALL_SIZE//2, HEIGHT//2 - BALL_SIZE//2, BALL_SIZE, BALL_SIZE)
ball_speed_x = 4
ball_speed_y = 4
# Blade speed
paddle_speed = 5
# Main game loop
running = True
while running:
    pygame.time.delay(10) # Refreshing the game
    # event handling
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Player controls
    keys = pygame.key.get_pressed()
    if keys[pygame.K_w] and paddle1.top > 0:
        paddle1.y -= paddle_speed
    if keys[pygame.K_s] and paddle1.bottom < HEIGHT:
        paddle1.y += paddle_speed
    if keys[pygame.K_UP] and paddle2.top > 0:
        paddle2.y -= paddle_speed
    if keys[pygame.K_DOWN] and paddle2.bottom < HEIGHT:
        paddle2.y += paddle_speed

    # Moving the ball
    ball.x += ball_speed_x
    ball.y += ball_speed_y
    # Ball collision with walls (up and down)
    if ball.top <= 0 or ball.bottom >= HEIGHT:
        ball_speed_y = -ball_speed_y
    # Ball collision with paddles
    if ball.colliderect(paddle1) or ball.colliderect(paddle2):
        ball_speed_x = -ball_speed_x
    # Reset the ball if it goes off the screen (point for the opponent)
    if ball.left <= 0 or ball.right >= WIDTH:
        ball.x = WIDTH//2 - BALL_SIZE//2
        ball.y = HEIGHT//2 - BALL_SIZE//2
        # Change of direction after each goal
        ball_speed_x = -ball_speed_x
    # Drawing game elements
    screen.fill(BLACK) # Screen refresh
    pygame.draw.rect(screen, WHITE, paddle1)
    pygame.draw.rect(screen, WHITE, paddle2)
    pygame.draw.ellipse(screen, WHITE, ball)
    # Line in the middle of the field
    pygame.draw.aaline(screen, WHITE, (WIDTH//2, 0), (WIDTH//2, HEIGHT))
    # Screen update
    pygame.display.flip()

pygame.quit()
```

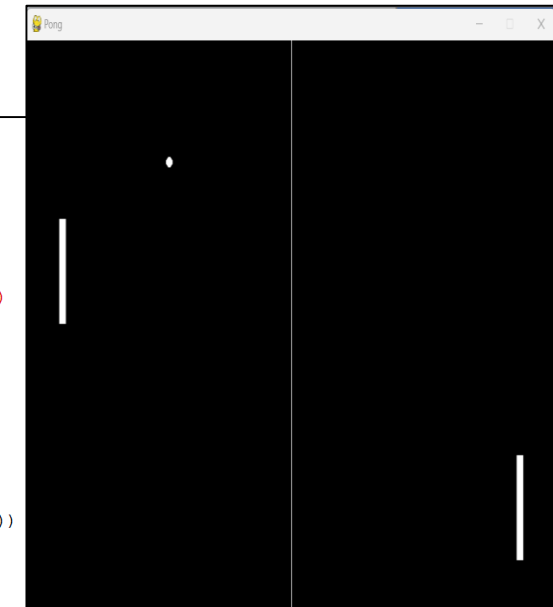


Figure 54. Example of creating a Pong game

## 6. Classes

```
slika55.py - C:\Users\unaed\OneDrive\Desktop\PYTHON_KURS\py_engleski\slika55.py (3.10.11)
File Edit Format Run Options Window Help
import random
def guess_number():
    number = random.randint(1, 5)
    attempts1 = 0

    while True:
        attempts = int(input("Guess the number (1-5): "))
        attempts1 += 1
        if attempts < number:
            print("Too low! Try again.")
        elif attempts > number:
            print("Too high! Try again.")
        else:
            print(f"Congratulations! You guessed the number {number} in {attempts1} attempts.")
            break
guess_number()

Ln: 17 Col: 1
```

Guess the number (1-5): 3  
Too low! Try again.  
Guess the number (1-5): 4  
Too low! Try again.  
Guess the number (1-5): 5  
Congratulations! You guessed the number 5 in 3 attempts.

Figure 55. Example of creating a Number Guessing game

```
slika56.py - C:\Users\unaed\OneDrive\Desktop\PYTHON_KURS\py_eng...
File Edit Format Run Options Window Help
import random
def game():
    options = ["stone", "paper", "scissors"]
    computer = random.choice(options)
    player = input("Izaberi: stone, paper ili scissors? ").lower()
    if player not in options:
        print("Invalid entry!")
        return
    print(f"The computer chose: {computer}")
    if player == computer:
        print("Undecided!")
    elif (player == "stone" and computer == "scissors") or \
        (player == "paper" and computer == "stone") or \
        (player == "scissors" and computer == "paper"):
        print("You won!")
    else:
        print("You lost!")
game()

Ln: 19 Col: 1
```

Izaberi: stone, paper ili scissors? paper  
The computer chose: stone  
You won!

Figure 56. Example of creating a Rock, Paper, Scissors game

# 7. Control commands

- Control commands are a key element of any programming language. Through statements such as if, elif, else, for, while, break, continue, and pass, Python enables conditional and loop control.

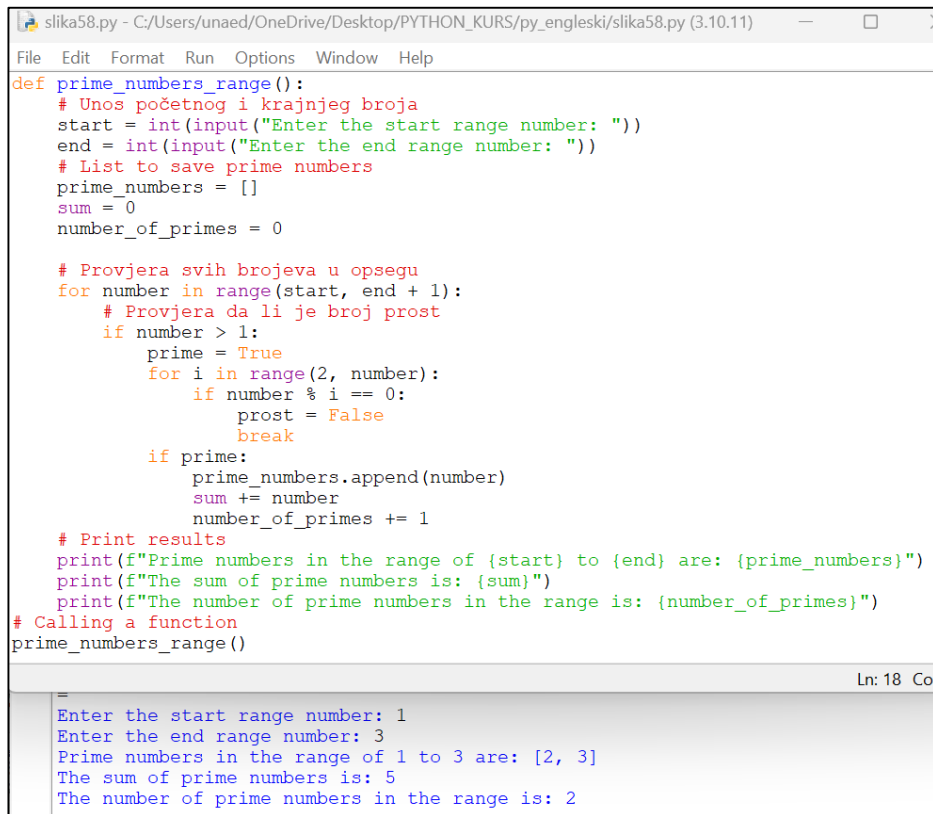
```
slika57a.py - C:/Users/unaed/OneDrive/Desktop/PYT...
File Edit Format Run Options Window Help
x = 10
if x > 0:
    print("Positive number")
elif x < 0:
    print("Negative number")
else:
    print("The number is zero")
Ln: 7 Co
KURS/py_engleski/slika57a.py =
Positive number
>>>
```

```
slika57b.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py...
File Edit Format Run Options Window Help
def student_grade():
    # Enter a grade
    grade = float(input("Input your grade (1-10): "))
    # Checking if the grade is within a valid range
    if grade < 1 or grade > 10:
        print("Invalid grade! Enter a grade between 1 and 10.")
    elif grade < 5:
        print("You failed. Try harder next time!")
    elif 5 <= grade < 7:
        print("You passed, but you have room for improvement.")
    elif 7 <= grade < 9:
        print("Good grade, well done!")
    else:
        print("Extraordinary! Excellent!")
# Calling a function
student_grade()
Ln: 15 Col:
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_e
ngleski/slika57b.py
Input your grade (1-10): 6
You passed, but you have room for improvement.
>>>
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_e
ngleski/slika57b.py
Input your grade (1-10): 1
You failed. Try harder next time.!
>>>
```

Figure 57. Example of the application of branch commands

# 7. Control commands

- Loops (for, while) make it possible to repeat a part of the code several times.

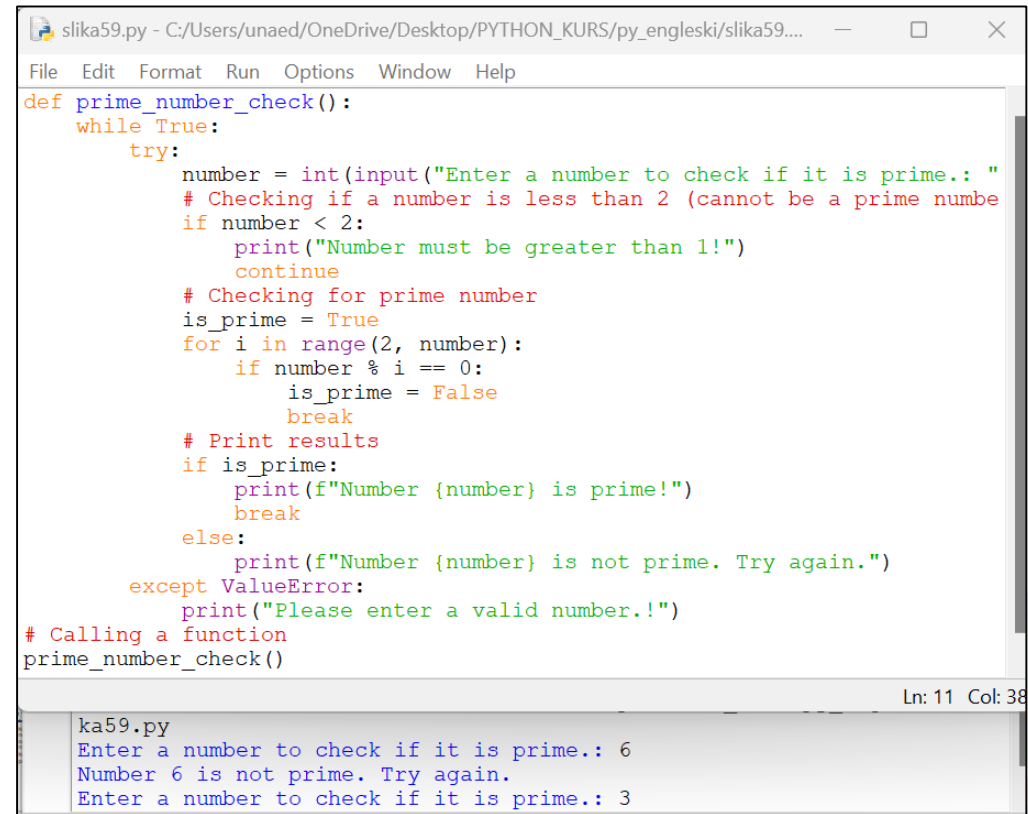
A screenshot of a Python IDE window titled 'slika58.py'. The code defines a function 'prime\_numbers\_range()' that takes no arguments. It prompts the user for a start and end range, then iterates through that range using a 'for' loop. Inside the loop, it checks if each number is prime by testing divisibility from 2 to the number itself. If prime, it adds it to a list and increments a counter. Finally, it prints the list of primes, their sum, and the count. The console output shows the user entering 1 and 3, resulting in the primes [2, 3], a sum of 5, and a count of 2.

```
def prime_numbers_range():  
    # Unos početnog i krajnjeg broja  
    start = int(input("Enter the start range number: "))  
    end = int(input("Enter the end range number: "))  
    # List to save prime numbers  
    prime_numbers = []  
    sum = 0  
    number_of_primes = 0  
  
    # Provjera svih brojeva u opsegu  
    for number in range(start, end + 1):  
        # Provjera da li je broj prost  
        if number > 1:  
            prime = True  
            for i in range(2, number):  
                if number % i == 0:  
                    prime = False  
                    break  
            if prime:  
                prime_numbers.append(number)  
                sum += number  
                number_of_primes += 1  
  
    # Print results  
    print(f"Prime numbers in the range of {start} to {end} are: {prime_numbers}")  
    print(f"The sum of prime numbers is: {sum}")  
    print(f"The number of prime numbers in the range is: {number_of_primes}")  
# Calling a function  
prime_numbers_range()
```

Ln: 18 Col: 1

Enter the start range number: 1  
Enter the end range number: 3  
Prime numbers in the range of 1 to 3 are: [2, 3]  
The sum of prime numbers is: 5  
The number of prime numbers in the range is: 2

Figure 58. Example of using a for loop

A screenshot of a Python IDE window titled 'slika59.py'. The code defines a function 'prime\_number\_check()' that uses a 'while True' loop to repeatedly prompt the user for a number to check if it is prime. It includes a try-except block to handle 'ValueError' exceptions from the 'int()' conversion. If the number is less than 2, it prints a message and continues the loop. If the number is 2 or greater, it checks for primality by testing divisibility from 2 to the number. If not prime, it prints a message and continues the loop. If prime, it prints the result and breaks the loop. The console output shows the user entering 6 (which is not prime) and then 3 (which is prime).

```
def prime_number_check():  
    while True:  
        try:  
            number = int(input("Enter a number to check if it is prime.: "))  
            # Checking if a number is less than 2 (cannot be a prime number)  
            if number < 2:  
                print("Number must be greater than 1!")  
                continue  
            # Checking for prime number  
            is_prime = True  
            for i in range(2, number):  
                if number % i == 0:  
                    is_prime = False  
                    break  
            # Print results  
            if is_prime:  
                print(f"Number {number} is prime!")  
                break  
            else:  
                print(f"Number {number} is not prime. Try again.")  
        except ValueError:  
            print("Please enter a valid number.!!")  
# Calling a function  
prime_number_check()
```

Ln: 11 Col: 38

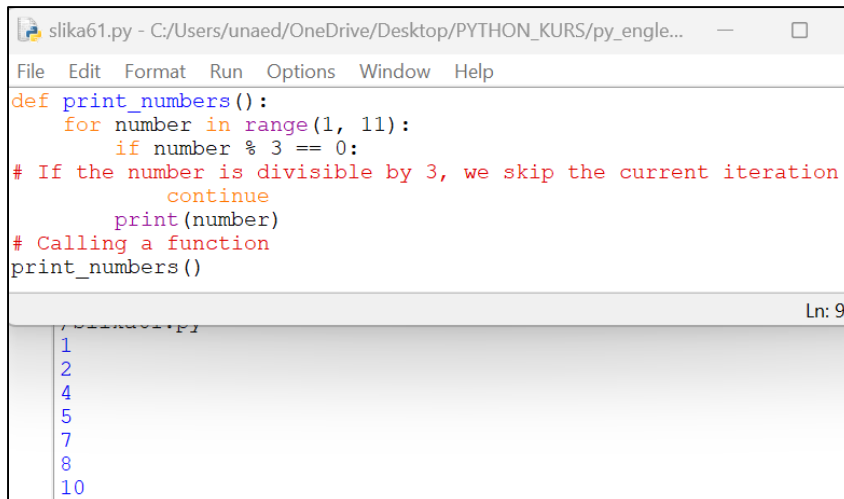
ka59.py  
Enter a number to check if it is prime.: 6  
Number 6 is not prime. Try again.  
Enter a number to check if it is prime.: 3

Figure 59. Example of using a while loop



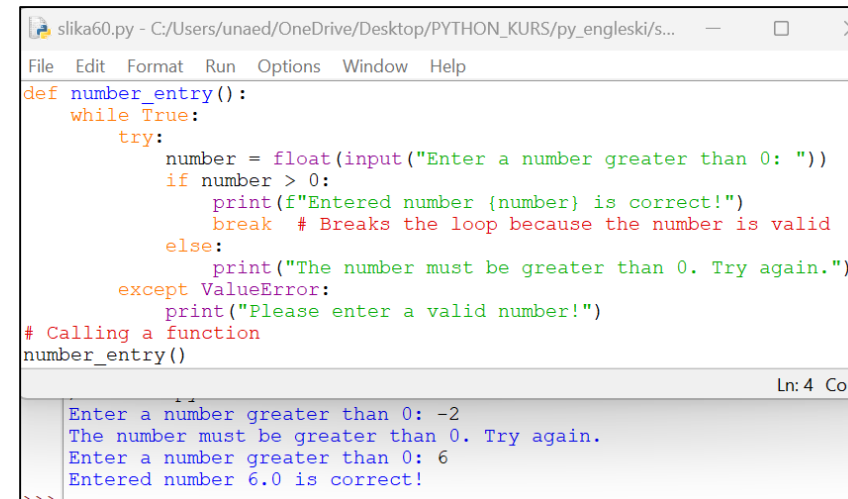
# 7. Control commands

- The break statement is used to terminate a loop (either for or while) before it has reached its normal end. When break is executed, the loop immediately stops working, and the program moves to the command that follows the loop.
- The continue statement is used to skip the current loop iteration and move to the next one.



```
slika61.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engle...
File Edit Format Run Options Window Help
def print_numbers():
    for number in range(1, 11):
        if number % 3 == 0:
            # If the number is divisible by 3, we skip the current iteration
            continue
        print(number)
    # Calling a function
    print_numbers()
Ln: 9
1
2
4
5
7
8
10
```

Figure 60. Example of using the continue command



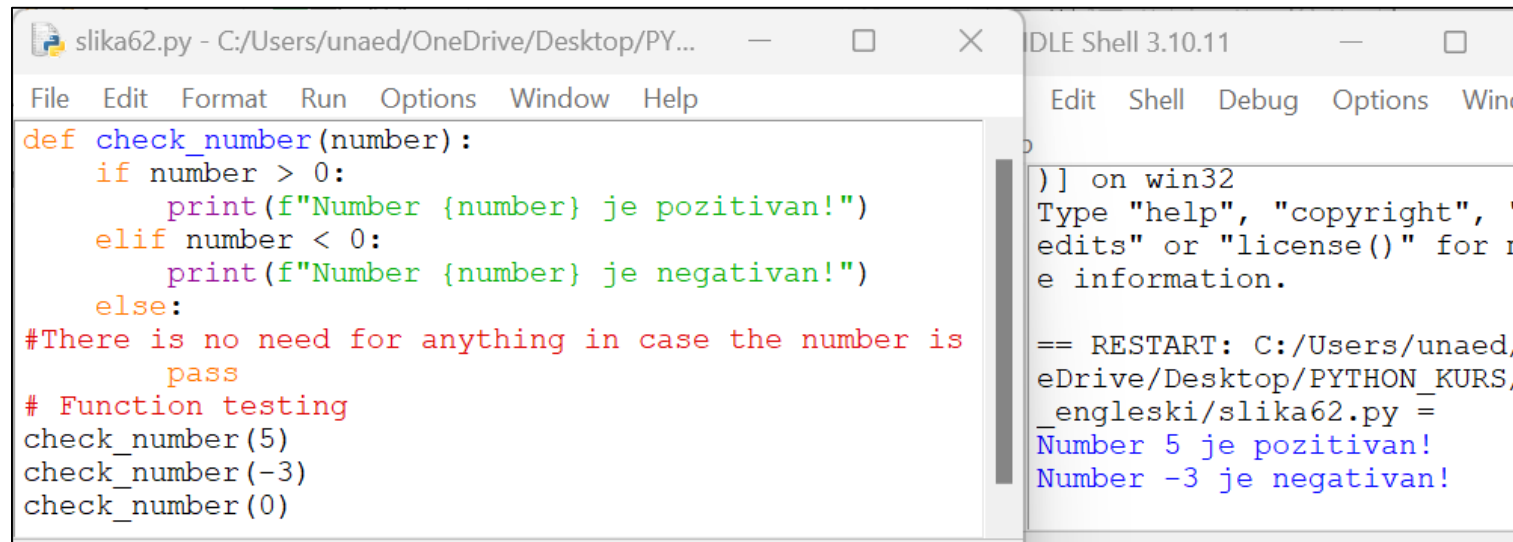
```
slika60.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/s...
File Edit Format Run Options Window Help
def number_entry():
    while True:
        try:
            number = float(input("Enter a number greater than 0: "))
            if number > 0:
                print(f"Entered number {number} is correct!")
                break # Breaks the loop because the number is valid
            else:
                print("The number must be greater than 0. Try again.")
        except ValueError:
            print("Please enter a valid number!")
    # Calling a function
    number_entry()
Ln: 4 Col:
Enter a number greater than 0: -2
The number must be greater than 0. Try again.
Enter a number greater than 0: 6
Entered number 6.0 is correct!
>>>
```

Figure 61. Example of using the break command



## 7. Control commands

- The pass statement is an empty statement that is used as a placeholder in situations where syntactically Python expects some code, but nothing is wanted to be executed. It is often used in function definitions, loops or conditional branches that have not yet been implemented.



The screenshot shows a Python IDE window titled 'slika62.py' with the following code:

```
def check_number(number):  
    if number > 0:  
        print(f"Number {number} je pozitivan!")  
    elif number < 0:  
        print(f"Number {number} je negativan!")  
    else:  
        #There is no need for anything in case the number is  
        pass  
    # Function testing  
check_number(5)  
check_number(-3)  
check_number(0)
```

The output window on the right shows the execution results:

```
)] on win32  
Type "help", "copyright", "  
edits" or "license()" for m  
e information.  
  
== RESTART: C:/Users/unaed/  
eDrive/Desktop/PYTHON_KURS/  
_engleski/slika62.py =  
Number 5 je pozitivan!  
Number -3 je negativan!
```

Figure 62. Example of using the pass command

## 8. Functions for graphical representation

---

- In Python, the Matplotlib package is most often used for data visualization in 2D and 3D space. Matplotlib provides a simple and flexible way to generate a wide range of graphs and charts, including 2D and 3D data representations. In addition, for more advanced graphical representation and interactive 3D graphics, we can use `mpl_toolkits.mplot3d`, which is an extension of Matplotlib that enables working with 3D data.
- 2D graphics are the most common and are used to visualize data in two-dimensional space. Matplotlib provides a rich set of functionalities for drawing line graphs, scatter graphs, histogram graphs, bar graphs and others.



## 8. Functions for graphical representation

```
import matplotlib.pyplot as plt
# Defining data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]
# Drawing a line chart
plt.plot(x, y)
# Label for x and y axis
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
# Chart title
plt.title('Line Chart')
# Show chart
plt.show()
```

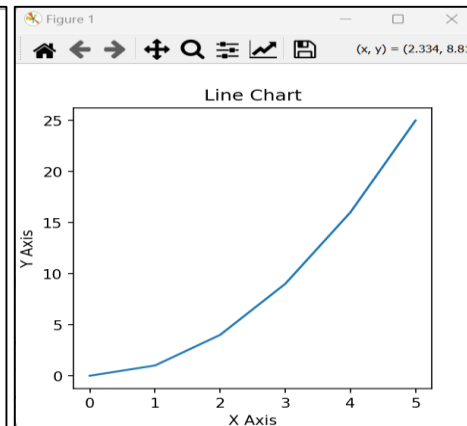


Figure 63. Example of a line graph

```
import matplotlib.pyplot as plt
# Data for scatter chart
x = [1, 2, 3, 4, 5]
y = [5, 4, 6, 8, 7]
# Drawing a scatter chart
plt.scatter(x, y, color='red')
# Labels
plt.xlabel('X Axes')
plt.ylabel('Y Axes')
# Title
plt.title('Scatter Chart')
# Show chart
plt.show()
```

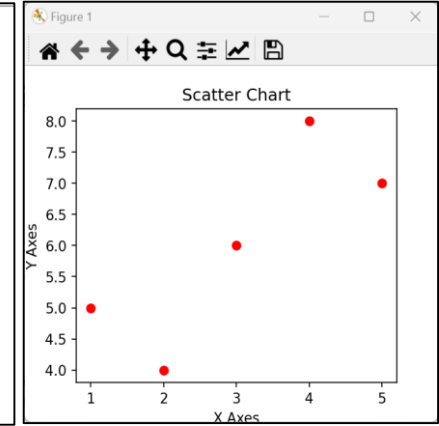


Figure 64. Example of a scatter graph

```
import matplotlib.pyplot as plt
# Histogram data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]
# Plotting a histogram
plt.hist(data, bins=5, color='blue', edgecolor='black')
# Labels
plt.xlabel('Values')
plt.ylabel('Frequency')
# Title
plt.title('Histogram')
# Show charts
plt.show()
```

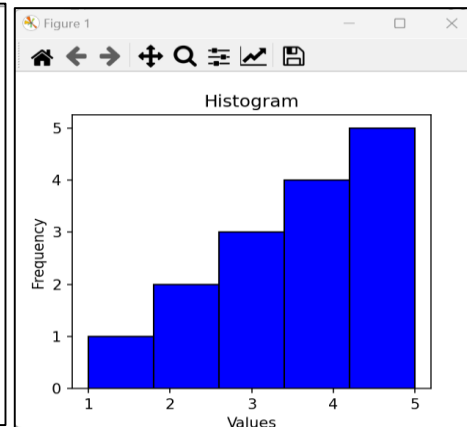


Figure 65. Example of a histogram

```
import matplotlib.pyplot as plt
# Categories and values
categories = ['A', 'B', 'C', 'D', 'E']
values = [3, 7, 5, 8, 6]
# Drawing a bar chart
plt.bar(categories, values)
# Labels
plt.xlabel('Categories')
plt.ylabel('Values')
# Title
plt.title('Bar Chart')
# Show chart
plt.show()
```

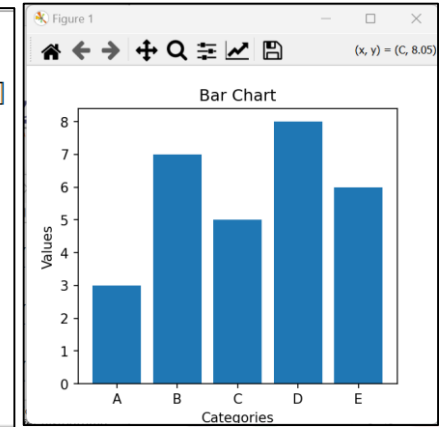


Figure 66. Example of a bar graph

## 8. Functions for graphical representation

- To work with 3D graphs in Python, the `mpl_toolkits.mplot3d` extension is used, which allows drawing three-dimensional objects such as 3D line graphs, scatter graphs, surface graphs, etc.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Data definition
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]
z = [0, 2, 4, 6, 8, 10]
# Creating a figure and 3D axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Drawing a 3D line graph
ax.plot(x, y, z)
# Labels and title
ax.set_xlabel('X axes')
ax.set_ylabel('Y axes')
ax.set_zlabel('Z axes')
ax.set_title('3D Line Chart')
# Show Chart
plt.show()
```

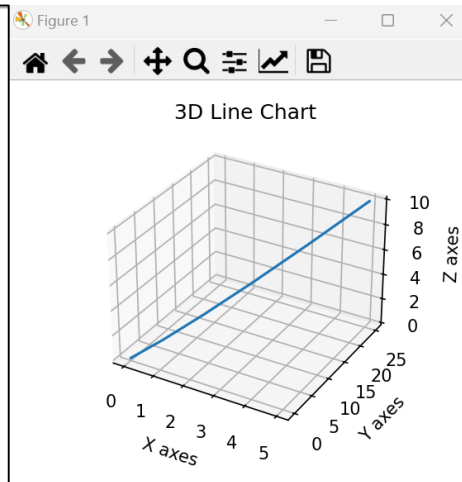


Figure 67. Example of a 3D line graph

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Data definition
x = [1, 2, 3, 4, 5]
y = [5, 4, 6, 8, 7]
z = [2, 3, 4, 6, 7]
# Creating a figure and 3D axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Drawing a 3D scatter chart
ax.scatter(x, y, z, color='red')
# Labels and title
ax.set_xlabel('X axes')
ax.set_ylabel('Y axes')
ax.set_zlabel('Z axes')
ax.set_title('3D Scatter Chart')
# Show Chart
plt.show()
```

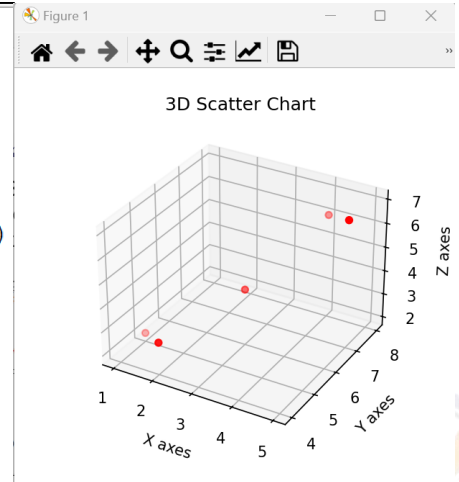


Figure 68. Example of a 3D scatter graph

# 8. Functions for graphical representation

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
# Defining data
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))
# Creating a figure and 3D axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Drawing a 3D surface chart
ax.plot_surface(x, y, z, cmap='viridis')
# Labels and title
ax.set_xlabel('X axes')
ax.set_ylabel('Y axes')
ax.set_zlabel('Z axes')
ax.set_title('3D Surface Chart')
# Show Chart
plt.show()
```

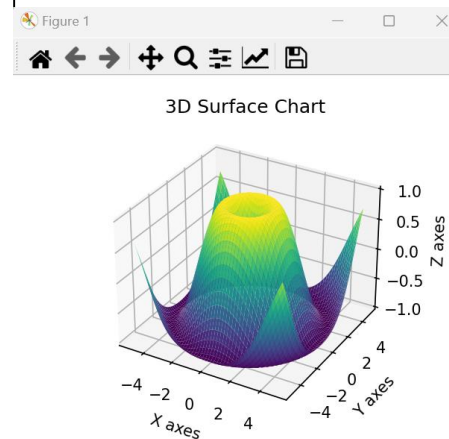


Figure 69. Example of a 3D surface graph

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generating data
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.cos(np.sqrt(x**2 + y**2))
# Creating a figure and 3D axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Drawing a 3D wireframe
ax.plot_wireframe(x, y, z, color='blue')
# Labels and title
ax.set_xlabel('X label')
ax.set_ylabel('Y label')
ax.set_zlabel('Z label')
ax.set_title('3D Wireframe')
# Chart rotation
# Changed rotation angles: elevation 45, azimuth 45
ax.view_init(elev=45, azim=45)
# Show chart
plt.show()
```

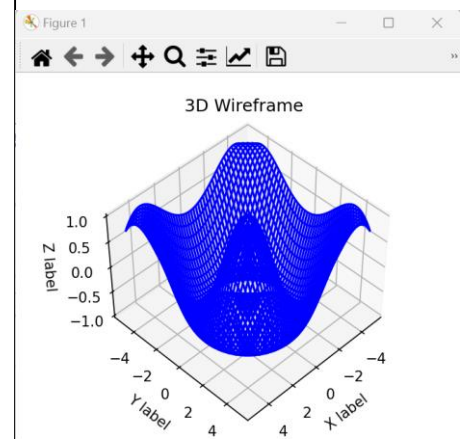


Figure 70. Example of rotating a 3D graph



## 9. Graphical interface

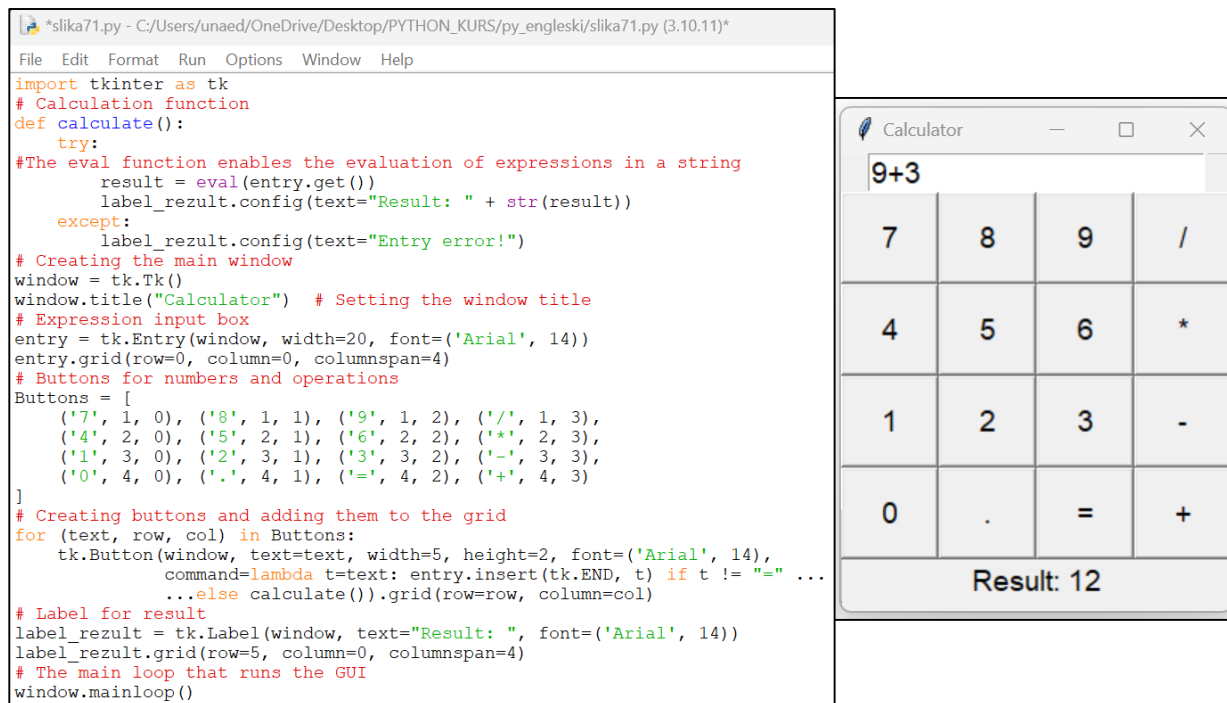
---

- The graphical user interface (GUI) in Python enables the development of applications with interactive windows, buttons, inputs and other graphical components. In Python, there are various packages and libraries that enable the creation of GUI applications, and the most famous and most used are:
  1. Tkinter - Basic GUI library in Python
  2. PyQt - A library based on the Qt framework
  3. wxPython - GUI framework, based on the wxWidgets library
  4. Kivy - A library for creating GUI applications with an emphasis on mobile devices and touch-based applications



## 9. Graphical interface

- Tkinter is a standard Python library for creating desktop applications. It provides an easy way to create windows, buttons, text fields, and other GUI elements.



Basic elements of Tkinter:

1. Tk(): Main application window
2. Label(): Displays the text
3. Button(): Creates a button
4. Entry(): Text input box
5. Canvas(): Drawing surface
6. MessageBox(): A dialog box for displaying messages

Figure 71. Example of using the Tkinter library

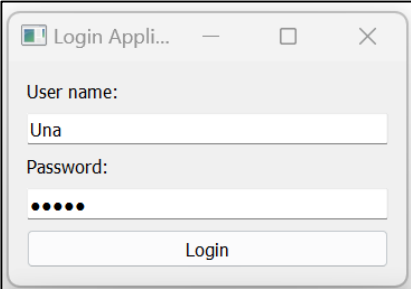
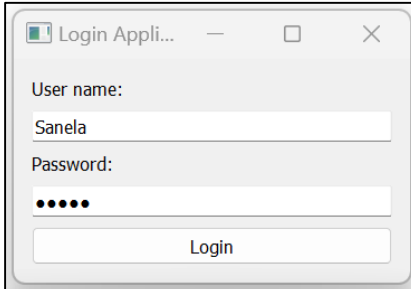


## 9. Graphical interface

- PyQt5 is a library based on the Qt framework, which provides greater capabilities and flexibility.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, ...
...QHBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox
class LoginApp(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()
    def init_ui(self):
        # Setting the basic layout
        self.setWindowTitle("Login Application")
        self.setGeometry(100, 100, 400, 200)
        # Creating a vertical layout for the main components
        vbox = QVBoxLayout()
        # Creating a vertical layout for input
        self.input_layout = QVBoxLayout()
        # Username and password entry fields
        self.username_input = QLineEdit(self)
        self.password_input = QLineEdit(self)
        # Hiding password entry
        self.password_input.setEchoMode(QLineEdit.Password)
        self.input_layout.addWidget(QLabel("User name:"))
        self.input_layout.addWidget(self.username_input)
        self.input_layout.addWidget(QLabel("Password:"))
        self.input_layout.addWidget(self.password_input)
        vbox.addLayout(self.input_layout)
        # Login button
        self.login_button = QPushButton("Login", self)
        self.login_button.clicked.connect(self.login)
        vbox.addWidget(self.login_button)
        # Setting the layout on the main window
        self.setLayout(vbox)
    def login(self):
        # Predefined user and password
        correct_username = "Una"
        correct_password = "12345una"
        # Retrieving entered data
        username = self.username_input.text()
        password = self.password_input.text()
```

```
# Checking entered data
if username == correct_username and password == correct_password:
    # If the data is correct
    self.show_message("You have successfully logged in!", "Welcome!")
else:
    # If the data is not correct
    self.show_message("Error", "The username or password is incorrect!")
def show_message(self, title, message):
    # Displaying a message with title and text
    msg = QMessageBox(self)
    msg.setWindowTitle(title)
    msg.setText(message)
    msg.exec_()
if __name__ == '__main__':
    app = QApplication(sys.argv)
    login_app = LoginApp()
    login_app.show()
    sys.exit(app.exec_())
```

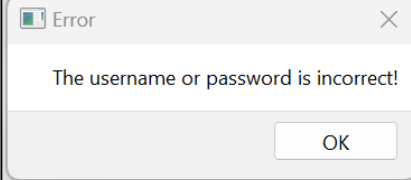
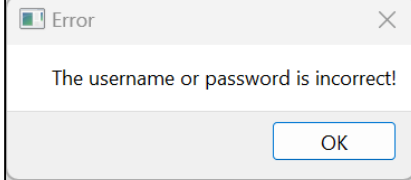



Figure 72. Example of using the PyQt5 library (Application for user login)





## 9. Graphical interface

- wxPython is a library for developing a graphical user interface (GUI) in Python. It is a Python wrapper for wxWidgets.

```
slika74.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika74.py (3.10.11)
File Edit Format Run Options Window Help
import wx
class ItemListFrame(wx.Frame):
    def __init__(self, parent, title):
        super().__init__(parent, title=title, size=(400, 300))
        self.panel = wx.Panel(self)
        # Creating an entry for a new item
        self.item_label = wx.StaticText(self.panel, label="Enter an item:", pos=(20, 20))
        self.item_input = wx.TextCtrl(self.panel, pos=(150, 20), size=(200, -1))
        # Creating a button to add an item
        self.add_button = wx.Button(self.panel, label="Add item", pos=(150, 60))
        self.add_button.Bind(wx.EVT_BUTTON, self.add_item) # Link to the item addition event
        # Creating a list to display items
        self.item_listbox = wx.ListBox(self.panel, pos=(20, 100), size=(350, 120))
        # Displaying windows
        self.Centre()
        self.Show()
    def add_item(self, event):
        # Retrieving entered data
        item = self.item_input.GetValue()
        # If the entry is empty, we display a message
        if not item:
            self.show_message("Error", "Please enter an item!")
        else:
            # Adding an item to a list
            self.item_listbox.Append(item)
            self.item_input.Clear() # Clearing entries after adding an item
    def show_message(self, title, message):
        # Creating a message dialog
        dialog = wx.MessageDialog(self.panel, message, title, wx.OK | wx.ICON_INFORMATION)
        dialog.ShowModal() # displaying dialogue
        dialog.Destroy() # Deleting a dialog after it is closed
# Main function for launching the application
def main():
    app = wx.App(False)
    ItemListFrame(None, "List of Items")
    app.MainLoop()
if __name__ == "__main__":
    main()
```

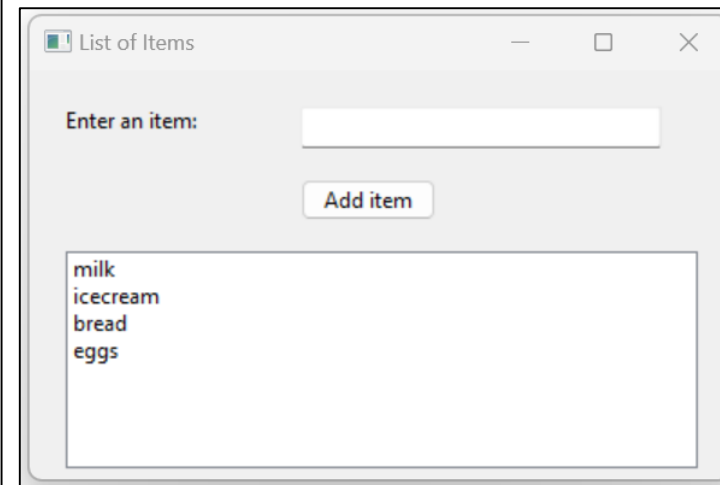


Figure 74. Example of using the wxPython library

## 9. Graphical interface

- Kivy is a library for creating GUI applications with an emphasis on mobile devices and touch-based applications.

```
slika75.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika75.py (3.10.11)
File Edit Format Run Options Window Help
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label
class PersonalInfoApp(App):
    def build(self):
        # Main Layout - vertical layout
        main_layout = BoxLayout(orientation="vertical", padding=10, spacing=10)
        # Creating a name input field
        self.name_input = TextInput(hint_text="Enter your name: ", size_hint=(1, None), height=50)
        # Creating a field for entering last name
        self.surname_input = TextInput(hint_text="Enter your surname: ", size_hint=(1, None), height=50)
        # Creating an age input field
        self.age_input = TextInput(hint_text="Enter your age: ", size_hint=(1, None), height=50)
        # Button for displaying the entered data
        display_button = Button(text="Data display", size_hint=(1, None), height=50)
        display_button.bind(on_press=self.display_data)
        # Label for displaying results
        self.result_label = Label(text="The entered data", size_hint=(1, None), height=50)
        # Adding components to the main layout
        main_layout.add_widget(self.name_input)
        main_layout.add_widget(self.surname_input)
        main_layout.add_widget(self.age_input)
        main_layout.add_widget(display_button)
        main_layout.add_widget(self.result_label)
        return main_layout
    def display_data(self, instance):
        # Getting data from input fields
        name = self.name_input.text
        surname = self.surname_input.text
        age = self.age_input.text
        # Creating a display format
        if name and surname and age:
            self.result_label.text = f"Your name: {name} {surname}, Your age: {age} age"
        else:
            self.result_label.text = "Please enter all information.!"
if __name__ == "__main__":
    PersonalInfoApp().run()
```

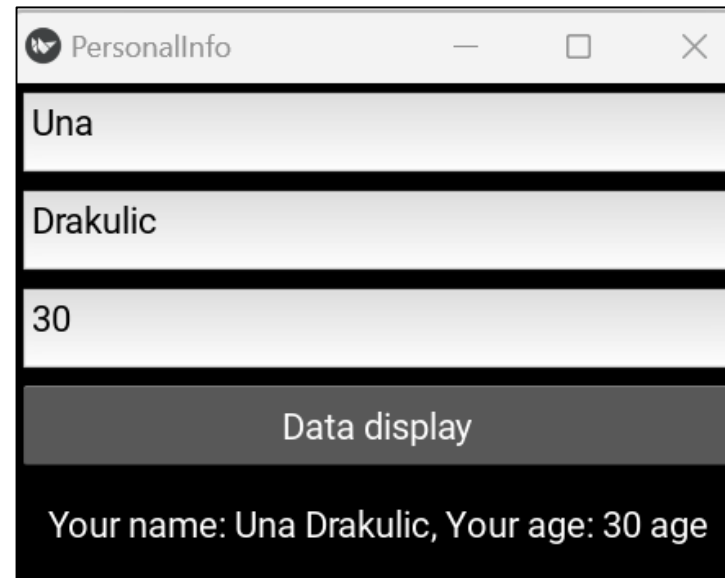


Figure 75. Example of using the Kivy library

## 9. Graphical interface

---

- An example of creating a GUI application that allows users to:
  - a) choose the shape they want to draw (circle or rectangle),
  - b) clicking on the canvas to draw the selected shape,
  - c) changes in the color of the shape,
  - d) empty the canvas if they want to start over,is given in figure 76.



# 9. Graphical interface

```
import tkinter as tk
from tkinter import colorchooser
# Function for drawing a rectangle
def draw_rectangle(event):
    if shape_var.get() == "Rectangle":
        canvas.create_rectangle(event.x - 50, event.y - 25, event.x + 50, event.y + 25, fill=color_var.get())
# Function for drawing a circle
def draw_circle(event):
    if shape_var.get() == "Circle":
        canvas.create_oval(event.x - 25, event.y - 25, event.x + 25, event.y + 25, fill=color_var.get())
# Function to change the shape color
def change_color():
    color_code = colorchooser.askcolor()[1]
    color_var.set(color_code)
# Canvas erase function
def clear_canvas():
    canvas.delete("all")
# Creating the main window
root = tk.Tk()
root.title("Drawing Shapes")
# Shape and color variables
shape_var = tk.StringVar(value="Circle") # The default shape is a circle.
color_var = tk.StringVar(value="black") # The default color is black.
# Creating widgets
canvas = tk.Canvas(root, width=500, height=500, bg="white")
canvas.pack()
# Shape selector buttons
circle_button = tk.Radiobutton(root, text="Circle", variable=shape_var, value="Circle")
circle_button.pack(side=tk.LEFT, padx=10)
rectangle_button = tk.Radiobutton(root, text="Rectangle", variable=shape_var, value="Rectangle")
rectangle_button.pack(side=tk.LEFT, padx=10)
# Color change button
color_button = tk.Button(root, text="Choose a color", command=change_color)
color_button.pack(side=tk.LEFT, padx=10)
# Canvas erase button
clear_button = tk.Button(root, text="Clear canvas", command=clear_canvas)
clear_button.pack(side=tk.LEFT, padx=10)
# The connection between the canvas and the drawing functions
canvas.bind("<Button-1>", lambda event: draw_circle(event) if shape_var.get() == "Circle" else draw_rectangle(event))
# Starting the main loop
root.mainloop()
```

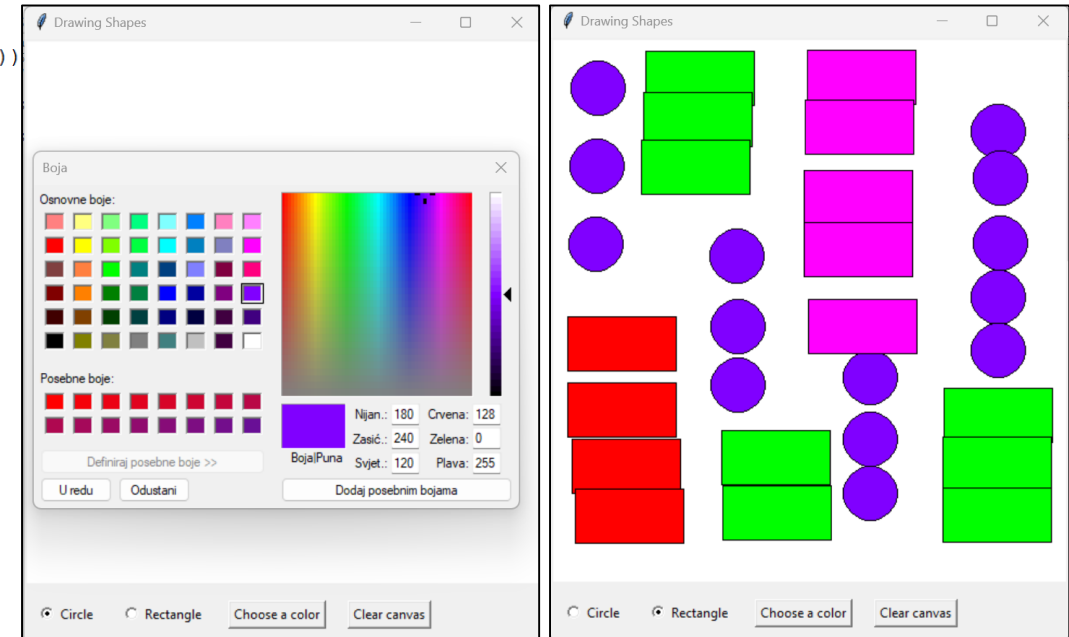


Figure 76. Example of a GUI application

# 10. Working with files

---

- Working with files in Python is very important for many applications, from simple programs that read and write data to complex data analysis applications. Python provides simple functions to open, read, write and close files.
- The `open()` function is used to work with files in Python. It opens the file and enables working with it.
- The syntax is: `file = open('file_name', 'mode')`, where 'file\_name' represents the name of the file to be opened (it can be an absolute or relative path), and 'mode' specifies how the file will be opened.
- The most common modes are: 'r'-reading (read); 'w'-writing (write); 'a'-adding (append); 'rb'-reading in binary format; 'wb'-writing in binary format.



# 10. Working with files

---

- The syntax for reading the content is

```
file=open('file_name.txt', 'r')  
content=file.read()  
print(content)  
file.close()
```

- The readline() method reads a single line from a file

```
file=open('file_name.txt', 'r')  
line=file.readline()  
while line:  
    print(line, end="") #end="" is for avoid empty spaces  
    lline=file.readline()  
file.close()
```



## 10. Working with files

---

- To write to the file, we use mode 'w' or 'a'. The 'w' mode deletes the contents of the file if the file already exists, while the 'a' mode appends new data to the end of the file.

```
file=open('file_name.txt', 'r')  
file.write('this is text! \n')  
file.write('this is new line of text! \n')  
file.close()
```

- Adding content to a file:

```
file=open('file_name', 'r')  
file.write('adding new line of text in file! \n')  
file.close()
```



# 10. Working with files

---

- If working with binary files, 'rb' and 'wb' modes can be used to read and write in binary format.

```
file=open('file_name.txt', 'rb')  
binary_data=file.read()  
print(binary_data) #print binary data  
file.close()
```

- Writing to a binary file:

```
file=open('new_image.jpg', 'wb')  
file.write(binary_data)  
file.close()
```





## 10. Working with files

---

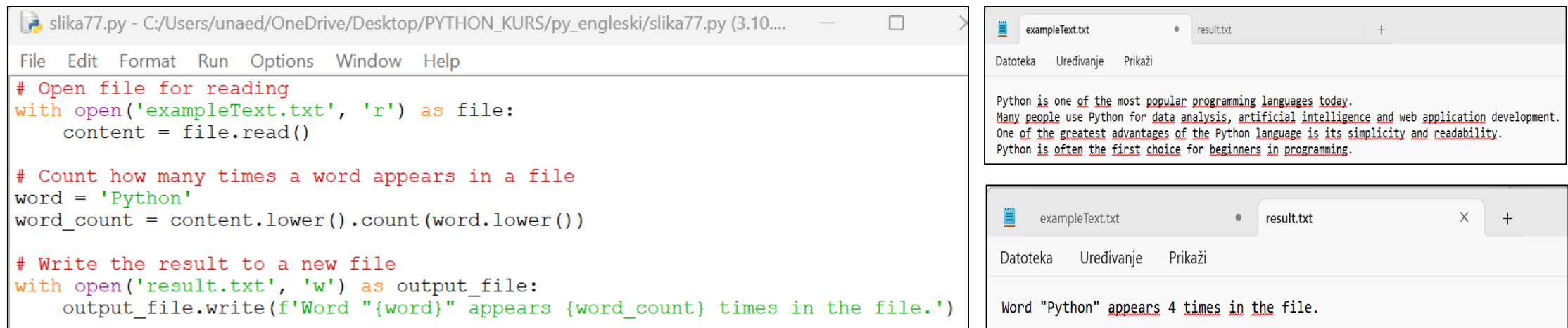
- Instead of calling `file.close()` explicitly, we can use a `with` statement that automatically closes the file when it's done working with it. This is the better way because it ensures that the file will be closed, even if an error occurs during processing.

```
with open('file_name.txt', 'r') as file:  
    content=file.read()  
    print(content)
```



## 10. Working with files

- An example of working with files is given in Figure 77. A program was created that reads the contents of a text file, counts how many times a certain word appears in the file, and then writes the number into a new file.



The figure shows two screenshots. The left screenshot displays a Python script named 'slika77.py' in a text editor. The script reads a file 'exampleText.txt', counts the occurrences of the word 'Python' (case-insensitive), and writes the result to 'result.txt'. The right screenshot shows the content of 'exampleText.txt' and the output in 'result.txt'.

```
slika77.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika77.py (3.10....
File Edit Format Run Options Window Help
# Open file for reading
with open('exampleText.txt', 'r') as file:
    content = file.read()

# Count how many times a word appears in a file
word = 'Python'
word_count = content.lower().count(word.lower())

# Write the result to a new file
with open('result.txt', 'w') as output_file:
    output_file.write(f'Word "{word}" appears {word_count} times in the file.')
```

exampleText.txt result.txt +

Datoteka Uređivanje Prikaži

Python is one of the most popular programming languages today.  
Many people use Python for data analysis, artificial intelligence and web application development.  
One of the greatest advantages of the Python language is its simplicity and readability.  
Python is often the first choice for beginners in programming.

exampleText.txt result.txt X +

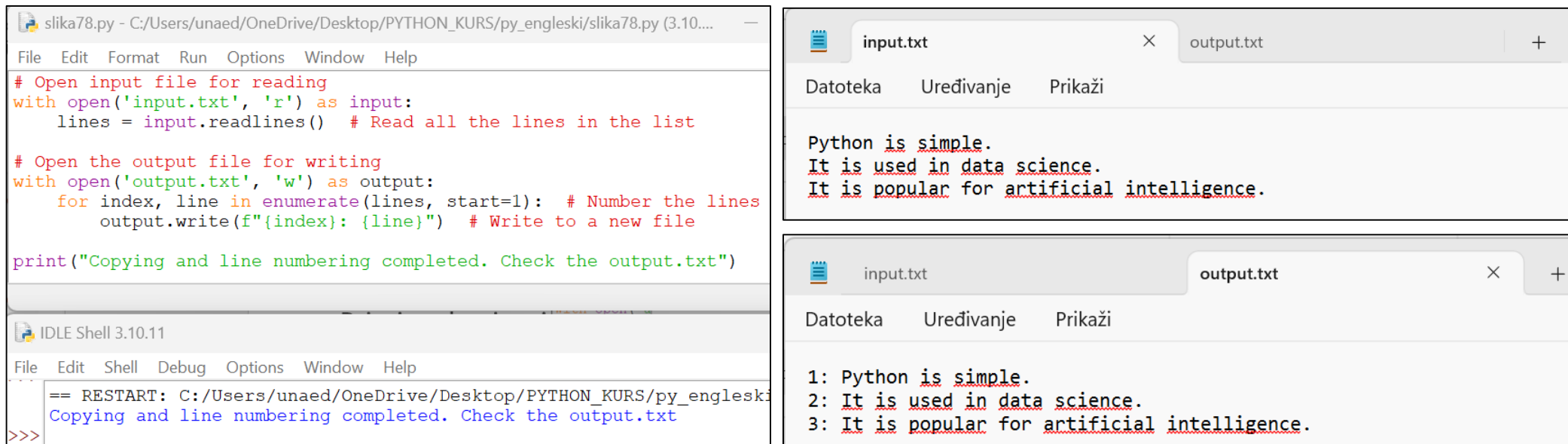
Datoteka Uređivanje Prikaži

Word "Python" appears 4 times in the file.

Figure 77. Example of working with a file

# 10. Working with files

- An example, copying content from one file to another and numbering lines, is shown in Figure 78.



```
slika78.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski/slika78.py (3.10....  
File Edit Format Run Options Window Help  
# Open input file for reading  
with open('input.txt', 'r') as input:  
    lines = input.readlines() # Read all the lines in the list  
  
# Open the output file for writing  
with open('output.txt', 'w') as output:  
    for index, line in enumerate(lines, start=1): # Number the lines  
        output.write(f"{index}: {line}") # Write to a new file  
  
print("Copying and line numbering completed. Check the output.txt")  
  
IDLE Shell 3.10.11  
File Edit Shell Debug Options Window Help  
== RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/py_engleski  
Copying and line numbering completed. Check the output.txt  
>>>
```

input.txt output.txt

Datoteka Uređivanje Prikaži

Python is simple.  
It is used in data science.  
It is popular for artificial intelligence.

input.txt output.txt

Datoteka Uređivanje Prikaži

1: Python is simple.  
2: It is used in data science.  
3: It is popular for artificial intelligence.

Figure 78. Example of working with a file

# Exercise tasks

---

- Write a program that represents a simple library management system, enabling the user to:
  - 1) Adds books - Enters the title and author of the book into the database;
  - 2) Book overview - Displays all books in the database using a table;
  - 3) Searches for books - Allows the user to search for books by title or author;
  - 4) Updates book information - Allows the user to change the title or author of the selected book;
  - 5) Delete books - Allows the user to delete a book from the database according to its ID;
  - 6) Exits the program - Closes the application;



# Exercise tasks

---

- Create an application that allows adding, viewing, updating and deleting students and their grades. Enable calculation of the average grade and sorting of students by average. The functionalities of the application are:
  - 1) Adding a student (name, surname);
  - 2) Adding grades to the student;
  - 3) Display of all students with averages;
  - 4) Updating grades;
  - 5) Deleting a student;
  - 6) Sorting by average;



# Exercise tasks

---

- Write a program that allows the user to calculate the area and volume for different geometric shapes. The program uses Python and mathematical functions from the math library to calculate areas and perimeters of shapes such as: circle, rectangle, triangle, trapezoid and parallelogram.
- Write a program that allows the user to calculate the area and volume of basic geometric shapes through a GUI graphic application.
- Write a program that allows users to draw free lines on the graphics canvas using the mouse. When the user clicks on the canvas (left mouse click), the initial coordinates (where he clicked) are remembered. When the user drags the mouse while holding down the left click, a line is drawn from the point where they previously clicked to the current mouse position. Each drawn line segment should connect the previous and current coordinates, creating a continuous line.



Co-funded by  
the European Union

# QUESTIONS & ANSWERS

*"Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them."*

Network of centers for regional short study programs in the countries of the Western Balkans

Call: ERASMUS-EDU-2023-CBHE

Project number: 101128813



UNIVERSITY OF LJUBLJANA  
Faculty of Electrical Engineering



University of Pristina  
Kosovska Mitrovica

