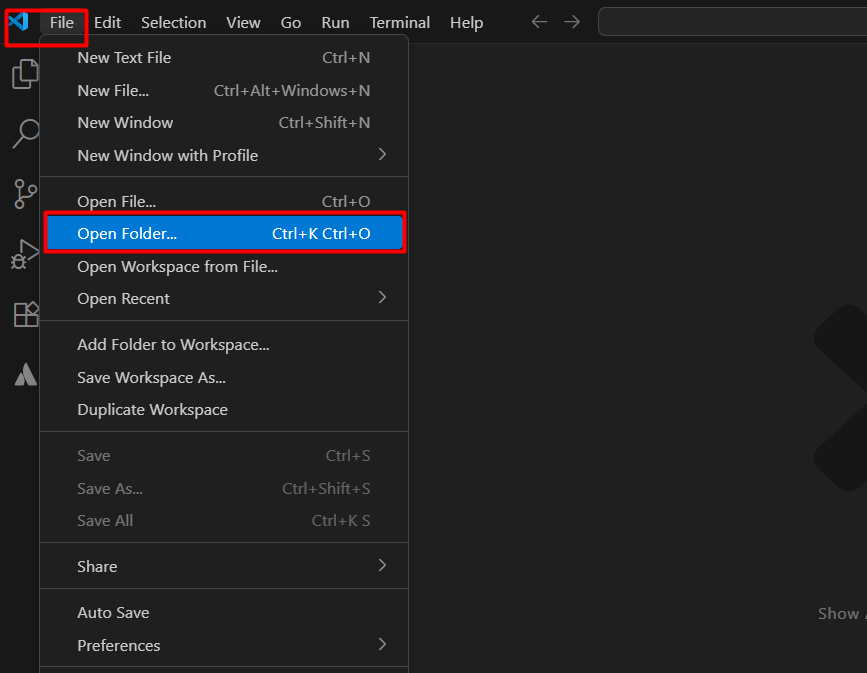
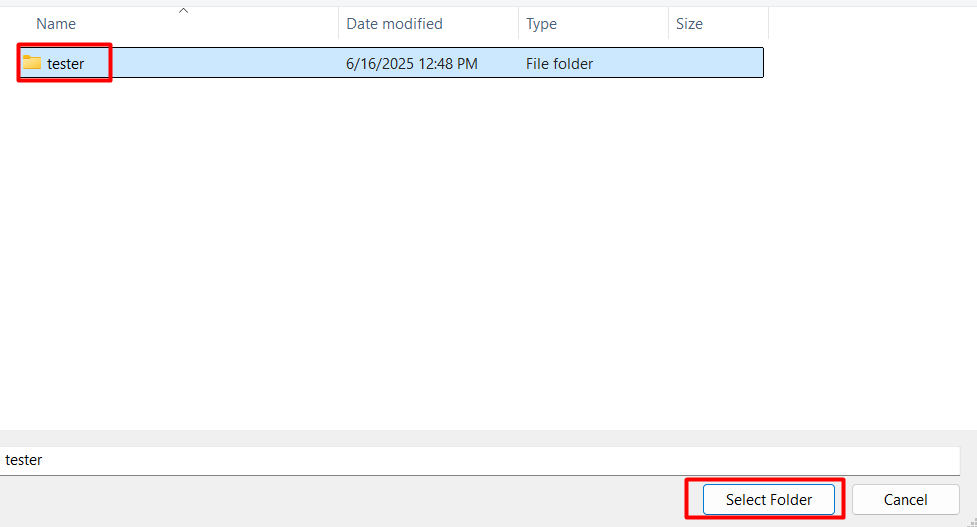
**LECTURE-09 (Saturday 14-June-2025)**

* Create a Python project in VS Code:

1. Create folder (e.g.: tester)
2. Open folder in VS Code





OR

Open folder path (e.g.: “D:\PY\tester”) in “File Explorer”, write “cmd” and press “Enter” to open command prompt in “tester” folder

Then write “code .” in command prompt inside “tester” folder and press “Enter”

1. Add new file “test.py” in project



1. Open terminal

by shortcut key “Ctrl + `”

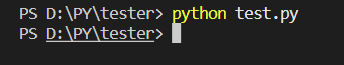
OR

“View” then “Terminal”

1. Run below command in terminal

Python test.py

1. Command successfully executed



* “main” method: Other programming languages like C#, have “main” method in their console application. Similarly, below “main” method code is a common Python pattern that ensures certain code only runs when the file executed directly means we run command “python filename.py”.

Code:

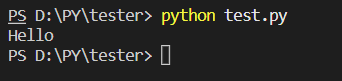
def main():

    print("Hello")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:



Here,

if \_\_name\_\_ == "\_\_main\_\_":

main()

Above code checks if the script is being run directly (i.e., not imported).

“\_\_name\_\_” is a special built-in variable.

When a Python file run directly, “\_\_name\_\_” is set to value "\_\_main\_\_".

However, above “main” method code will not executed when it imported as a module in another file.

When it imported in another file, “\_\_name\_\_” becomes the name of the module “test”.

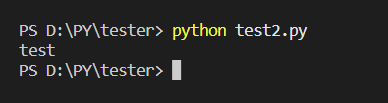
Therefore, we created another file “test2.py” in the same project then we imported “test” file in “test2” file.

Code of “test2.py”:

import test

print(test.\_\_name\_\_)

Output:

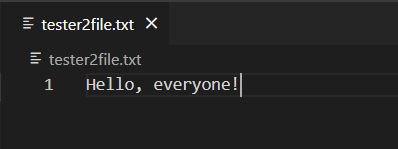


Purpose of “main” method is to keep your script both:

1. Reusable (as a module)
2. Runnable (as a standalone program)

* Write data in file:

1. Create new project “tester2” in VS Code
2. Add file “tester2file.txt”
3. Add text in file “Hello, everyone!”



1. Add file “tester2.py” and add below code in this file:

def create\_and\_write\_file(filename, text):

    with open(filename, "a") as file:

        file.write(text)

def main():

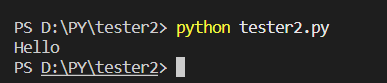
    print("Hello")

    create\_and\_write\_file("tester2file.txt", "\nThis is a testing text to check if we are able to write data in file")

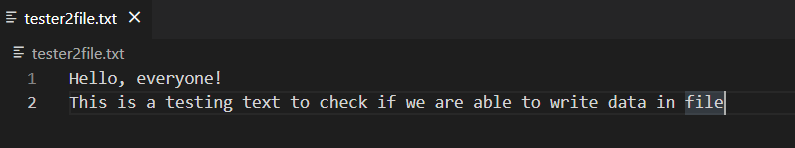
if \_\_name\_\_ == "\_\_main\_\_":

    main()

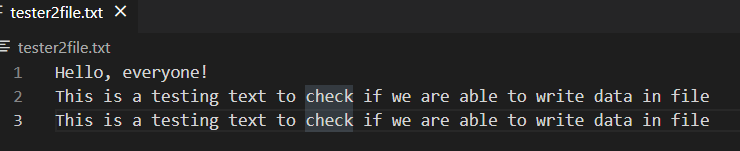
Output:



Each time, we execute code, text “This is a testing text to check if we are able to write data in file” added in file “tester2file.txt”



If we again execute code present, in “tester2.py” file then



In above code:

“open” method is used to open the file.

“a” stands for append mode (if the file doesn't exist, it will be created; if it does exist, text will be added to the end).

“with” ensures that the file is properly closed after writing (even if an error occurs).

“write” method add the text to the file.

* Read data from file:

1. Create new project “tester3” in VS Code
2. Add file “tester3file.txt”

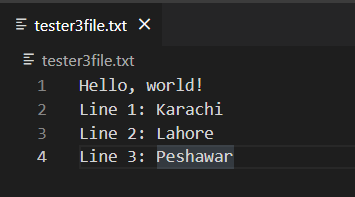
Add below text in file:

Hello, world!

Line 1: Karachi

Line 2: Lahore

Line 3: Peshawar



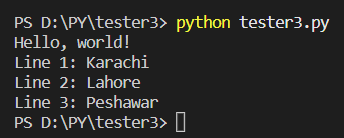
1. Add file “tester3.py” and add below code in this file:

file = open("tester3file.txt", "r")

content = file.read()

print(content)

Output:



Here,

file = open("tester3file.txt", "r")

This line opens the file named “tester3file.txt” in read mode “r”. If the file does not exist, Python will raise an error FileNotFoundError. “file” variable is now a file object that can be used to read data.

content = file.read()

This line reads the complete data of file “tester3file.txt” and save data into variable “content” as a string. If the file is large then variable “content” could use a lot of memory.

In above code, file “tester3file.txt” not closed after reading. It's a good practice to close file after performing operation. Therefore, updated code in file “tester3.py”:

file = open("tester3file.txt", "r")

content = file.read()

file.close()

print(content)

In addition, if you want to read data of file based on lines present inside file we will use code as:

file = open("tester3file.txt", "r")

content = file.readline()

print(content)

content = file.readline()

print(content)

content = file.readline()

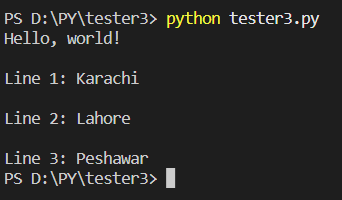
print(content)

content = file.readline()

print(content)

file.close()

Output:



* <https://github.com/panaversity/learn-modern-ai-python/tree/main/00_python_colab/10_file_handling>
* <https://colab.research.google.com/drive/12l0swaVyLwOvsMfoPNzUmpvfs2_VZ5bW?usp=sharing>
* We use “open” function to open a file and in arguments of “open” function, we specify filename with extension and the mode (read, write, append, etc.).
* Modes in which a file can be opened (tester4 project):

1. “r” (read): used to open file for reading data present inside file. It is default mode

Example:

# "r" (read) mode

file = open("tester4.txt","r")

content = file.read()

print(content)

1. “w” (write): used to open file so that we can add data in file. It creates the file if it does not exist and overwrite data means if data is already present in file then remove existing file data and new data in file.

Example:

# "w" (write) mode

file = open("tester4.txt","w")

file.write("abc\nxyz")

file = open("tester4.txt","r")

content = file.read()

print(content)

1. “a” (append): used to open file so that we can add data in file. It creates the file if it does not exist but add data means if data is already present in file then it keeps existing file data and add new data in file. .”\n” will not work with “write” function so we have to use “writelines” function.

Example:

# "w" (write) mode

file = open("tester4.txt","a")

file.write("jklvvw")

file.writelines("jkl\nvvw")

file = open("tester4.txt","r")

content = file.read()

print(content)

1. x: Exclusive creation (fails if file exists)
2. b: Binary mode (Used with the other modes (e.g., "rb", "wb") for working with binary files.)
3. +: Update mode (Can be combined with other modes (e.g., "r+", "w+") to allow both reading and writing.)

* File Pointer: In Python, a file pointer (also called a file cursor or file position indicator) is an internal marker that keeps track of the current position in a file where the next read or write operation will occur. When you open a file, the file pointer is initialized to a specific position (usually 0, the beginning of the file), and it moves automatically as you read from or write to the file. Key Concepts:

1. Starting Position:
2. “r” (read mode): pointer starts at 0 (beginning of the file).
3. “a” (append mode): pointer starts at the end of the file.
4. “w” (write mode): The file data is truncated (removed), and the pointer starts at 0.
5. How It Moves: when you read/write data, the pointer moves forward by the number of bytes/characters processed means reading 10 characters moves the pointer 10 positions forward.
6. Manual Control:
7. Use “seek” (offset) to move the pointer to a specific byte position.
8. Use “tell” function” to check the current pointer position.

Example (tester5 project):

File “tester5file.txt” having below text

Learning something new every day helps your brain stay sharp and your mind open to fresh ideas.

“seek” function code:

file = open("tester5file.txt", "r")

file.seek(10)

content = file.read()

print(content)

Output:



“tell” function code:

file = open("tester5file.txt", "r")

file.seek(18 + 7)

current\_position = file.tell()

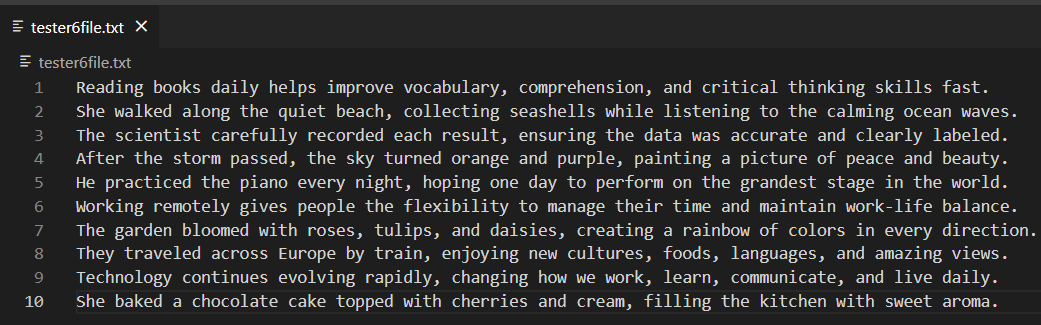
print(current\_position)

Output:



* “tester6” project

“tester6file.txt” contains text:



1. “readline”: This function read only one line and add new line in the end. This function returns string value.

Code:

file = open("tester6file.txt", "r")

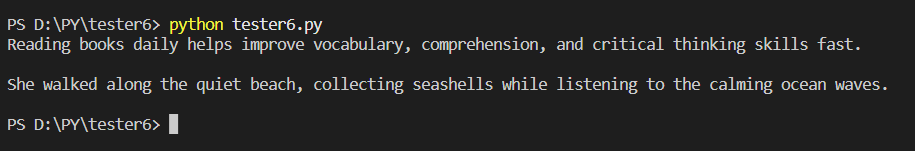
content = file.readline()

print(content)

content = file.readline()

print(content)

Output:



1. “readlines”: This function read all lines present inside file. This function returns list datatype having string data.

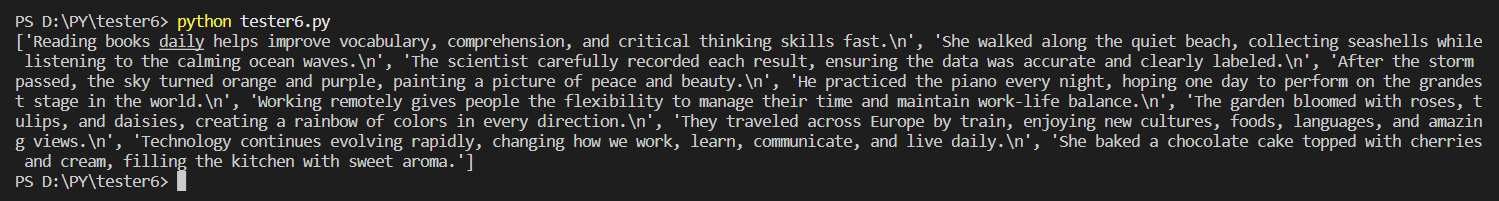
Code:

file = open("tester6file.txt", "r")

lines = file.readlines()

print(lines)

Output:



Code to read line one by one:

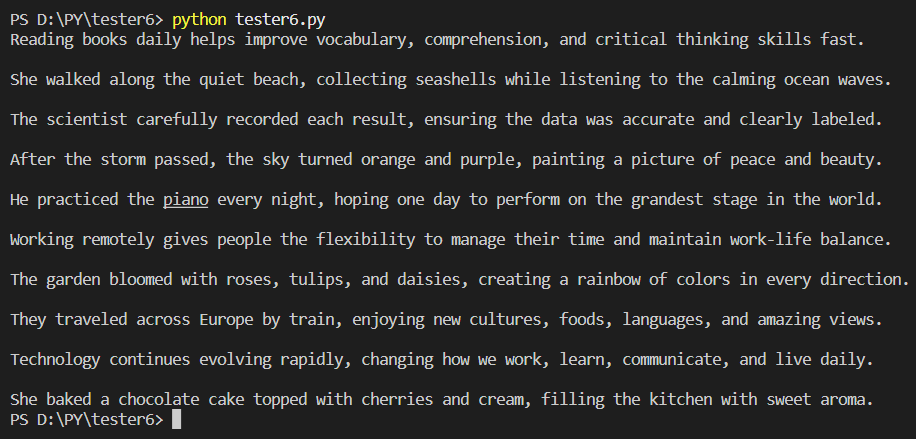
file = open("tester6file.txt", "r")

lines = file.readlines()

for line in lines:

    print(line)

Output:



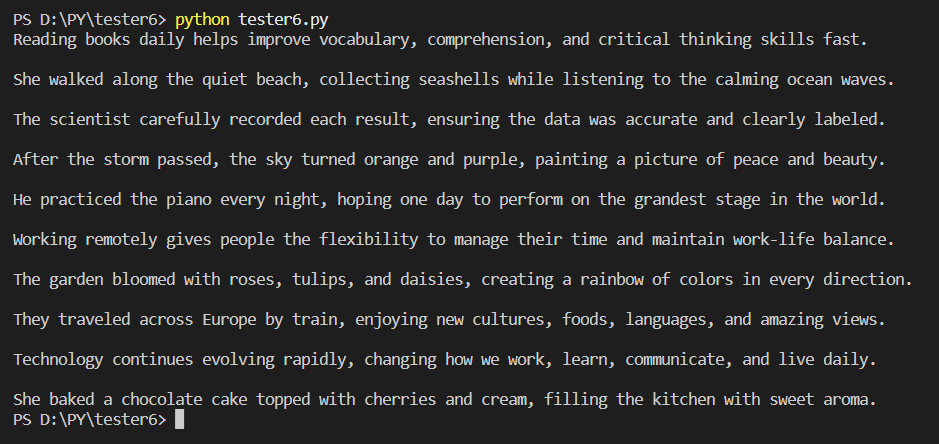
Code to read data directly from the file line by line:

file = open("tester6file.txt", "r")

for line in file:

    print(line)

Output:



* “strip”: This function is used to removes leading/trailing whitespace means it remove spaces present before start of text and end of text but not in between text.

Example:

abc : str = "    pqr  def   "

print(abc)

print(abc.strip())

Output:

pqr def

pqr def

* “close”: This function is used to close files. It is crucial to close files after we are finished working with files as file closing will releases system resources and ensures that data properly written to the disk.
* It is best practice to use “with” block for automatic cleanup and “try-except” block for handling exceptions while working with file operations.

Example:

try:

  with open('unique.txt', 'x') as file:

    file.write('Created exclusively!')

    print("File created successfully!")

except FileExistsError:

  print("File already exists.")

When code first time executed then output and “unique.txt” file will be created and text “Created exclusively!” added inside file.

File created successfully!



When we again executed same code then below output because file already exist so “except” block executed:

File already exists.

* “file.seek(offset, whence)”: This code moves the file pointer to a new location in the file.

1. “offset”: how many bytes to move
2. “whence”: from where to start moving (defaults to 0 if not provided)
3. “0” (start of file): move offset bytes from the beginning
4. “1” (current position”: move offset bytes forward/backward from where you are now
5. “2” (end of file): move offset bytes from the very end (use negative offsets to move backwards)

Example: “tester7” project

Data in “tester7file.txt” file:

Reading books daily helps improve vocabulary, comprehension, and critical thinking skills fast.

Code:

file = open("tester7file.txt", "rb")

file.seek(5, 0)

print(file.read(5))

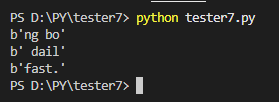
file.seek(3, 1)

print(file.read(5))

file.seek(-5, 2)

print(file.read())

Output:



Here,

file.seek(5, 0)

This code will move cursor “5” (first argument) position forward from start (“0” in second argument) then

print(file.read(5))

This code will read next “5” (argument) characters from current cursor position (“5” in above step) means will read “5” characters from 6thposition. Therefore, output of this code is “ng bo”

file.seek(3, 1)

This code will move cursor “3” (first argument) position forward from current position (“1” in second argument) then

print(file.read(5))

This code will read next “5” (argument) characters from current cursor position (“13” in above step) means will read “5” characters from 14th position. Therefore, output of this code is “ dail”

file.seek(-5, 2)

This code will move cursor “5” (first argument) position backward (-5) from last position (“2” in second argument) then

print(file.read())

Now, ”file” object contains only last five characters of file because of above step. Therefore, output of this code is “fast.”

* Copy a File:

Code: “tester8” project

def copy\_file(source\_path, destination\_path):

    try:

        with open(source\_path, "r") as source\_file, open(destination\_path, "w") as dest\_file:

            for line in source\_file:

                dest\_file.write(line)

        print(f"File '{source\_path}' copied to '{destination\_path}' successfully.")

    except FileNotFoundError:

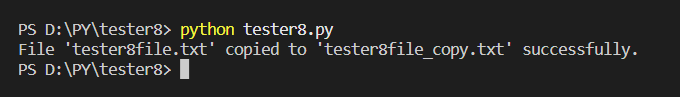
        print(f"Error: Source file '{source\_path}' not found.")

    except Exception as e:

        print(f"An error occurred: {e}")

copy\_file("tester8file.txt", "tester8file\_copy.txt")

Output:

Here,

File present at “source\_path” opened to read while new file created and opened at “destination\_path” then one by one lines from “source\_path” file is read and read data written in file of “destination\_path”

* Binary Mode:

Example: “tester9” project

Code:

def copy\_image(source\_path, destination\_path):

    try:

        with open(source\_path, 'rb') as source\_file:

            with open(destination\_path, 'wb') as destination\_file:

                while True:

                    chunk = source\_file.read(1024)

                    if not chunk:

                        break

                    destination\_file.write(chunk)

        print(f"Image copied successfully from '{source\_path}' to '{destination\_path}'")

    except FileNotFoundError:

        print(f"Error: Source file '{source\_path}' not found.")

    except Exception as e:

        print(f"An error occurred: {e}")

copy\_image("testimage.jpg", "copy\_testimage.jpg")

Output:

Data of image file “testimage.jpg” is copied to image file “copy\_testimage.jpg” file

Here,

“rb”: binary read mode

“wb”: binary write mode

“chunk = source\_file.read(1024)”: it reads 1024 bytes (1 KB) and store into “chunk”

“while” loop continues until there is no data left in “source\_path” file to read means chunk is empty.

“destination\_file.write(chunk)” : it writes each chunk to the destination file.

* The difference between "rb" and "r" is that “r” is used to read text (string, characters) e.g.: text files (.txt, .csv) while “rb” is used to read bytes (binary) data e.g.: images, audio, videos, PDFs.
* The difference between "wb" and "w" is that “w” is used to write text (string, characters) e.g.: text files (.txt, .csv) while “wb” is used to write bytes (binary) data e.g.: images, audio, videos, PDFs.
* OOP (Object Oriented Programming): It is a programming approach that organizes software design around objects rather than functions or logic.
* Key Concepts of OOP:

1. Object: An instance of a class that contains data (attributes) and behavior (methods).

Example: A Car object can have attributes like color and speed, and methods like drive() or brake().

1. Class: A blueprint for creating objects. It defines what attributes and methods the objects will have.

Example:

class Person:

  species = "Human"

  def \_\_init\_\_(self, name):

    self.name = name

    self.state = "Idle"

  def running(self):

    self.state = "Running"

    print(f"{self.name} is now running.")

  def walking(self):

    self.state = "walking"

    print(f"{self.name} is now walking.")

  def sleeping(self):

    self.state = "sleeping"

    print(f"{self.name} is now sleeping.")

  def show\_state(self):

    print(f"{self.name} is currently in {self.state}")

Here,

“Person” is a class, which models a person with a name and a state (such as Idle, Running, Walking, or Sleeping).

“species” is class variable/attribute.

“\_\_init\_\_” is a constructor method and it is a special method called when a new object of class “Person” created.

In Python, “self” refers to the current instance of the class. It is used inside class methods to access or modify the attributes and other methods of the same object. When you create an object from a class, each object has its own data. To access that data inside the class, Python uses “self”.

“self.name”, “name” is instance variable/attribute.

“self.state”, “state” is also instance variable/attribute.

The method “running” changes the person's state to "Running" and prints a message showing the person's current action which is similar to methods “walking” and “sleeping”.

The method “show\_state” displays the current state of the person.

In below code, we have used class “Person” and its methods and attributes:

p1 = Person("ABC")

p1.walking()

p1.show\_state()

p1.sleeping()

p1.show\_state()

print(p1.species)

p2 = Person("DEF")

p2.species = "Man"

p2.show\_state()

p2.sleeping()

p2.show\_state()

print(p2.species)

Output:

ABC is now walking.

ABC is currently in walking

ABC is now sleeping.

ABC is currently in sleeping

Human

DEF is currently in Idle

DEF is now sleeping.

DEF is currently in sleeping

Man

Here,

p1 = Person("ABC"): It creates a “Person” class’s object named "p1".

p1.walking(): It calls “walking” method of class “Person”.

print(p1.species): It calls class “Person” attribute ‘s “species”.

p2 = Person("DEF"): : It creates another “Person” class’s object named "p2".

p2.species = "Man": It overrides the “species” only for “p2” object. Now, “p2.species” is "Man" but “p1.species” remains "Human".

* Key Principles/Pillars of OOP:

1. Encapsulation: It is hiding internal details and only exposing what is necessary. It protects data from unwanted changes.

Example: using private variables and getter/setter methods.

1. Inheritance: It is inheriting properties and methods to one class (child) from another class (parent).

Example: A “SportsCar” class can inherit from the “Car” class and add more features.

1. Polymorphism: It is the ability of different objects to respond in their own way to the same method call.

Example: Dog.make\_sound() and Cat.make\_sound() can both exist, but behave differently.

1. Abstraction: It is hiding complex implementation and showing only essential features.

Example: You can use the “drive” method without knowing how the engine works internally.