**1. Python installation and configuration with windows and Linux.**

**Ans :-**

**Python Installation and Configuration on Windows:**

Visit the official Python website at https://www.python.org/downloads/ and go to the Downloads section.

Choose the latest stable version of Python for Windows and select the installer appropriate for your system (either 32-bit or 64-bit).

Run the downloaded installer, and on the installation wizard, select the "Add Python to PATH" option. This ensures that Python can be easily accessed from the command prompt.

Proceed with the installation by following the on-screen instructions.

After the installation is complete, open the command prompt and verify that Python is installed by typing python --version. You should see the installed Python version.

**Python Installation and Configuration on Linux (Ubuntu) :-**

Open a terminal window.

Update the package lists by running the command: sudo apt update.

Install Python by running the command: sudo apt install python3.

Verify that Python is installed by typing python3 --version. You should see the installed Python version.

By default, Ubuntu comes with Python 3 installed. If you want to use Python 2.x alongside Python 3, you can install it by running the command: sudo apt install python.

**Configuration Steps (Applies to both Windows and Linux) :-**

Installing Python packages: You can use the package manager pip to install Python packages. For example, to install a package named requests, you would run pip install requests in the command prompt or terminal.

Virtual environments (optional): It's recommended to use virtual environments to isolate Python environments for different projects. You can create a virtual environment by running python -m venv myenv (replace myenv with your preferred name). Activate the virtual environment using the command source myenv/bin/activate (Linux) or

myenv\Scripts\activate (Windows). To deactivate the virtual environment, simply type deactivate.

**2. Programs for understanding the data types, control flow statements, blocks and loops.**

**2(i). Program to find the area of a square.**

**2(ii). Program to find the area of rectangle.**

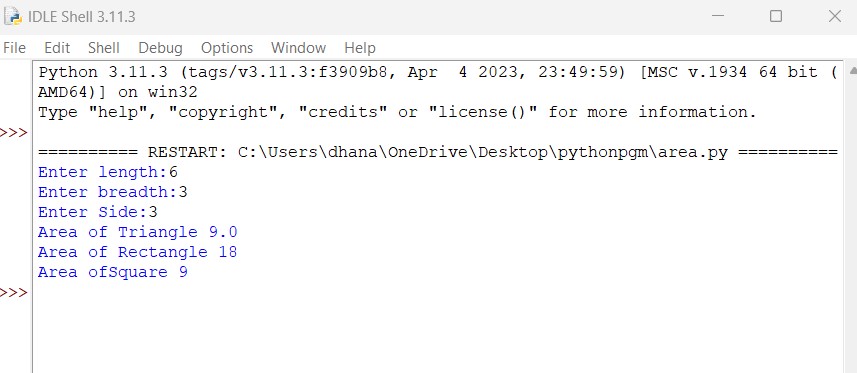
l=int(input("Enter length:")) b=int(input("Enter breadth:")) a=int(input("Enter Side:")) area=0.5\*l\*b

print("Area of Triangle",area) area=l\*b

print("Area of Rectangle",area) square=a\*a

print("Area of Square",square)

**Output :-**



**2(iii). Program to Check is a Number is Odd or Even.**

def check\_odd\_even(number): if number % 2 == 0:

print(number, "is even.")

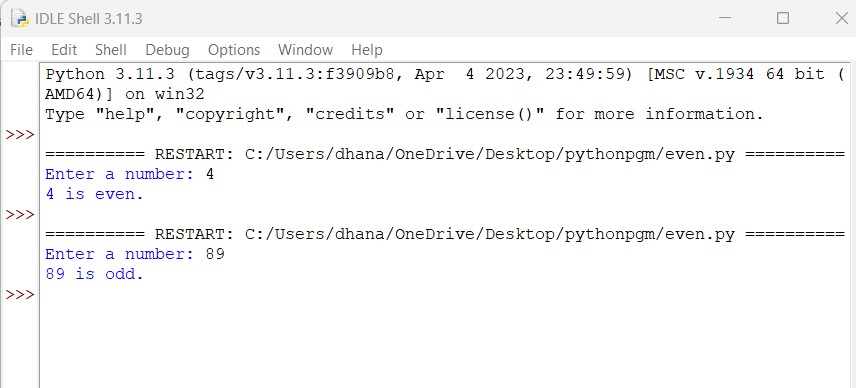
else:

print(number, "is odd.")

# Test the function

num = int(input("Enter a number: ")) check\_odd\_even(num)

**Output :-**



**2(iv). Program to Check is a Number is Positive, Negative or Zero.**

def check\_positive\_negative\_zero(number):

if number > 0:

print(number, "is positive.")

elif number < 0:

print(number, "is negative.")

else:

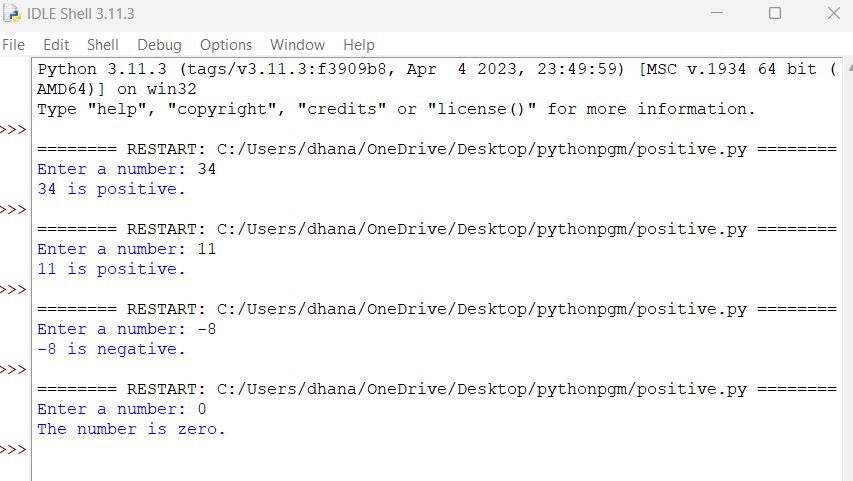
print("The number is zero.")

# Test the function

num = int(input("Enter a number: "))

check\_positive\_negative\_zero(num)

**Output :-**



**2(v). Python Program to Check is a Number is Prime or not Prime.**

def is\_prime(number):

if number <= 1:

return False

for i in range(2, int(number \*\* 0.5) + 1): if number % i == 0:

return False return True

# Test the function

num = int(input("Enter a number: "))

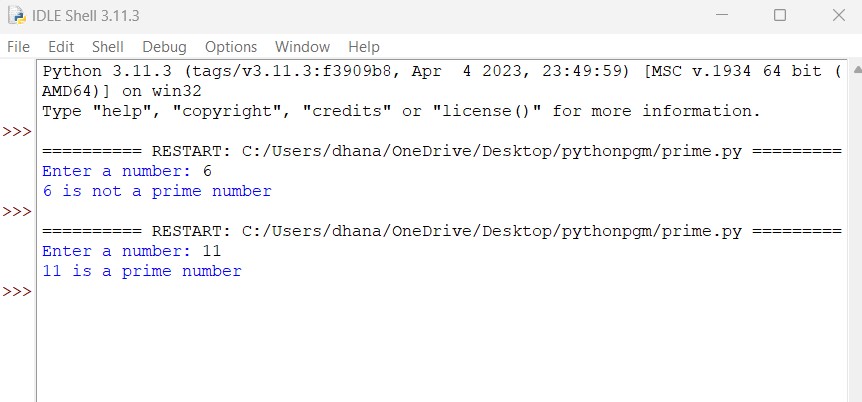
if is\_prime(num):

print(num, "is a prime number")

else:

print(num, "is not a prime number")

**Output :-**



**2(vi). Python Program to Check Year is Leap or Non-Leap.**

def is\_leap\_year(year):

if year % 4 == 0:

if year % 100 == 0:

if year % 400 == 0:

return True # Divisible by 400, so it's a leap year

else:

return False # Divisible by 100 but not by 400, so it's not a leap year

else:

return True # Divisible by 4 but not by 100, so it's a leap year

else:

return False # Not divisible by 4, so it's not a leap year

# Test the function

year = int(input("Enter a year: "))

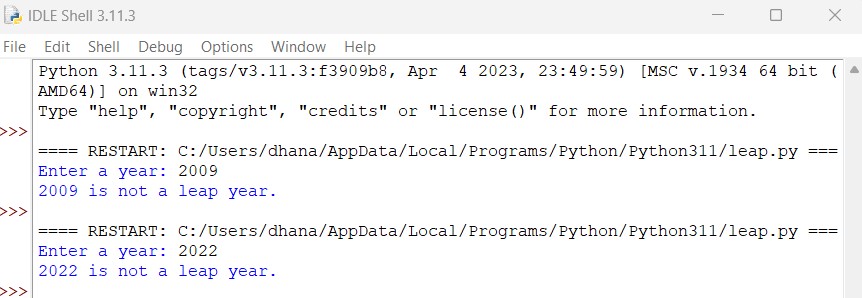
if is\_leap\_year(year):

print(year, "is a leap year.")

else:

print(year, "is not a leap year.")

**Output :-**



**2(vii). Python Program to Find the Factorial of a Number. Make the Function for it.**

def is\_armstrong\_number(num):

# Convert the number to a string to count the digits num\_str = str(num)

num\_digits = len(num\_str)

# Calculate the sum of digits raised to the power of num\_digits

armstrong\_sum = sum(int(digit) \*\* num\_digits for digit in num\_str)

# Check if the sum is equal to the original number

return armstrong\_sum == num

# Test the function

number = int(input("Enter a number: "))

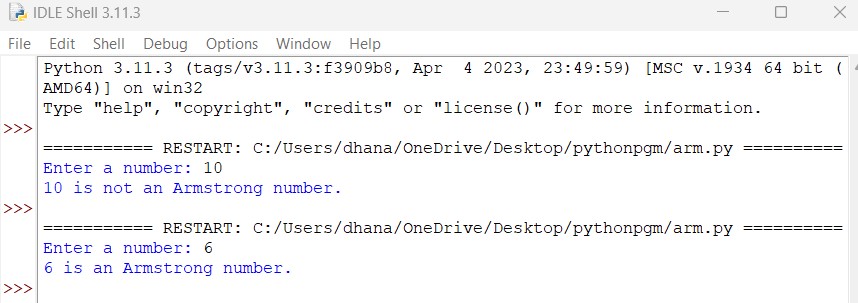
if is\_armstrong\_number(number):

print(number, "is an Armstrong number.")

else:

print(number, "is not an Armstrong number.")

**Output :-**



**2(viii). Python Program to Check Armstrong Number.**

num = int(input("Enter a number: "))

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* 3

temp //= 10

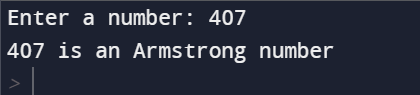
if num == sum:

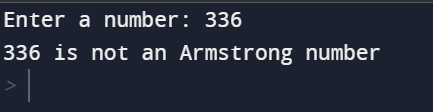
print(num,"is an Armstrong number")

else:

print(num,"is not an Armstrong number")

**Output :-**

****

****

**3.Programs for understanding functions, use of built in functions, user defined functions.**

**3(i). Python Program to Find the Addition, Subtraction, Multiplication and Division of two numbers, make the Function for it.**

def perform\_operations(num1, num2):

addition = num1 + num2

subtraction = num1 - num2

multiplication = num1 \* num2

division = num1 / num2

return addition, subtraction, multiplication, division

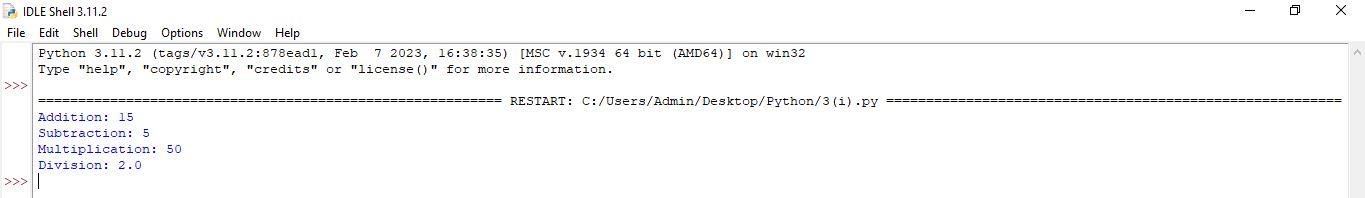
# Example usage: number1 = 10 number2 = 5

result = perform\_operations(number1, number2) print(f"Addition: {result[0]}")

print(f"Subtraction: {result[1]}") print(f"Multiplication: {result[2]}")

print(f"Division: {result[3]}")

**Output :-**



**3(ii).** **Python Program to Find the Factorial of a Number. Make the Function for it.**

def factorial(n): if n == 0: return 1

else:

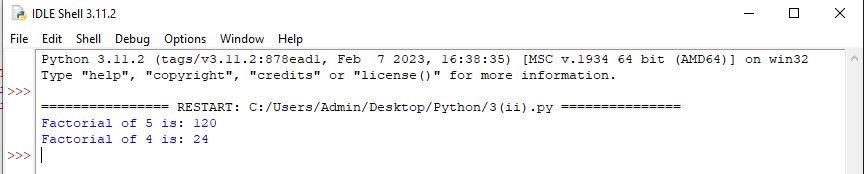
return n \* factorial(n - 1)

# Test the factorial function

print("Factorial of 5 is:",factorial(5))

print("Factorial of 4 is:",factorial(4))

**Output :-**



**3(iii). Python Program to Display the multiplication Table. Make the Function for it.**

def multiplication\_

table(n):

for i in range(1, 11):

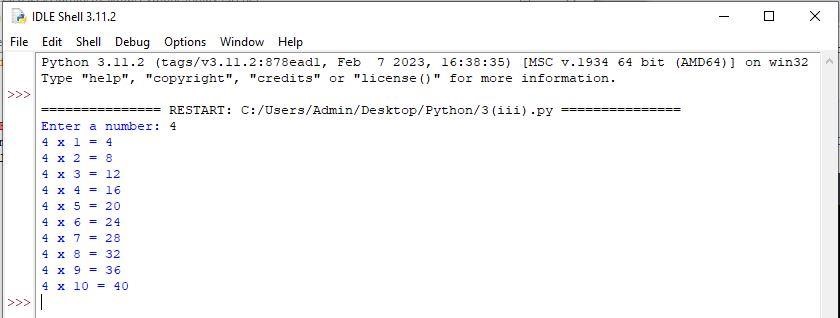
print(f"{n} x {i} = {n \* i}")

# Example usage

number = int(input("Enter a number: "))

multiplication\_table(number)

**Output :-**



**4. Programs to use existing modules, packages and creating modules, packages**

**4(i).Python program shows use of built in module and packages**.

# Importing the built-in 'random' module import random

# Using the random module to generate a random number between 1 and 10

random\_number = random.randint(1, 10)

print("Random number between 1 and 10: ", random\_number)

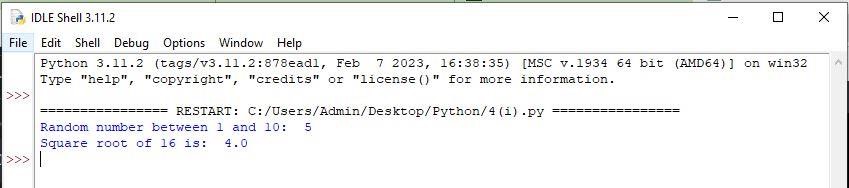
# Importing the 'math' package from built-in modules import math

# Using the math package to calculate the square root of a number number = 16

square\_root = math.sqrt(number)

print("Square root of", number, "is: ", square\_root)

**Output :-**



**4(ii). Python program to create module of Arithmetic operation**.

Arithmetic.py 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 a / b else:

print("Error: Division by zero!") return None

Operation.py

import arithmetic

# Perform addition

result = arithmetic.add(5, 3) print("Addition:", result)

# Perform subtraction

result = arithmetic.subtract(8, 4) print("Subtraction:", result)

# Perform multiplication

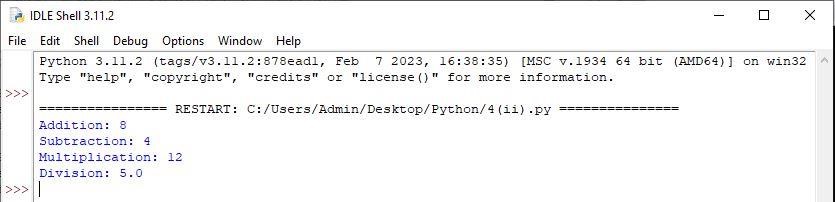
result = arithmetic.multiply(2, 6) print("Multiplication:", result)

# Perform division

result = arithmetic.divide(10, 2)

print("Division:", result)

**Output :-**



**4(iii). Python program to create package and use package.**

**mypack.py** def subtract(a, b): return a - b

def add(a, b): return a + b

**demo.py**

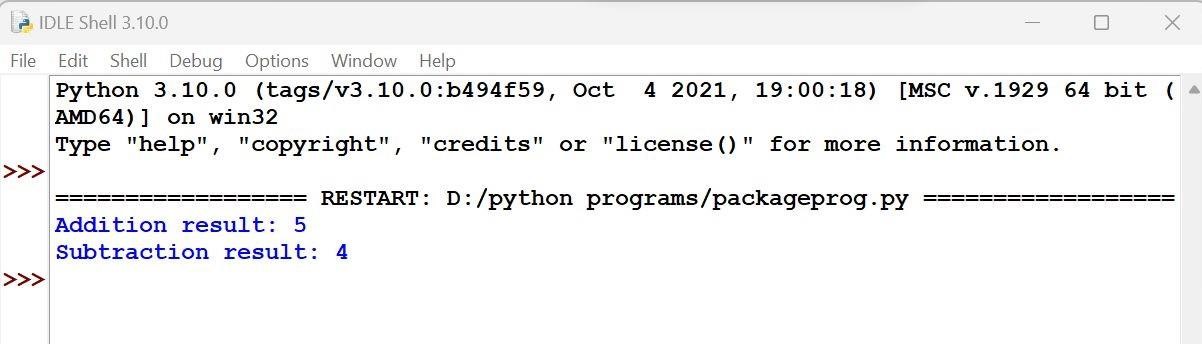
from mypack

import add as a

result1 = a.add(2, 3)

result2 = a.subtract(5, 1) print("Addition result:", result1) print("Subtraction result:", result2)

**Output :-**



**5. Programs for implementations of all object-oriented concepts like class, method, inheritance, polymorphism etc. (Real life examples must be covered for the implementation of object-oriented concepts).**

**5(i). Python program for class which has at least two methods: getString: to get a string from console input, printString: to print the string in uppercase.**

class StringProcessor: def \_\_init\_\_(self): self.string = ""

def getString(self):

self.string = input("Enter a string: ")

def printString(self):

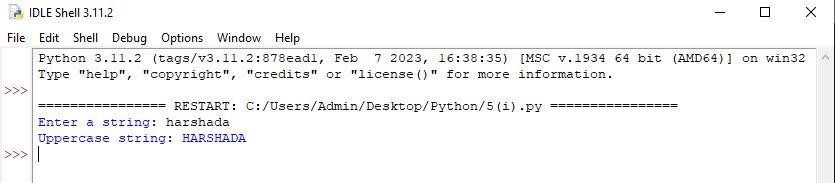
print("Uppercase string:", self.string.upper())

# Create an instance of the StringProcessor class processor = StringProcessor()

# Get a string from the user processor.getString()

# Print the string in uppercase processor.printString()

**Output :-**



**5(ii). Python program for Bank demo.**

class Bank:

def \_\_init\_\_(self, name):

self.name = name

self.accounts = {}

def create\_account(self, account\_number, initial\_balance): if account\_number in self.accounts:

print("Account already exists.")

else:

if initial\_balance < 0:

print("Initial balance cannot be negative.")

else:

self.accounts[account\_number] = initial\_balance print("Account created successfully.")

def deposit(self, account\_number, amount):

if account\_number in self.accounts:

if amount < 0:

print("Deposit amount cannot be negative.")

else:

self.accounts[account\_number] += amount print("Deposit successful.")

else:

print("Account not found.")

def withdraw(self, account\_number, amount): if account\_number in self.accounts:

if amount < 0:

print("Withdrawal amount cannot be negative.") elif amount > self.accounts[account\_number]: print("Insufficient balance.")

else:

self.accounts[account\_number] -= amount print("Withdrawal successful.")

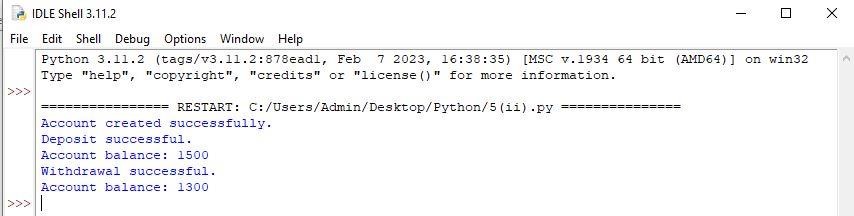
else:

print("Account not found.")

def check\_balance(self, account\_number): if account\_number in self.accounts:

print(f"Account balance: {self.accounts[account\_number]}")

**Output :-**



**5(iii). Python program for Inheritance demo covering all type of inheritance.**

# Single Inheritance class Vehicle:

def \_\_init\_\_(self, name):

self.name = name def display\_info(self):

print(f"Name: {self.name}")

class Car(Vehicle):

def \_\_init\_\_(self, name, model): super().\_\_init\_\_(name) self.model = model

def

display\_info(self): super().display\_info() print(f"Model:{self.model}")

# Multiple Inheritance class Animal: def\_\_init\_\_(self,species): self.species=species

def display\_info(self):

print(f"Species: {self.species}") class Pet:

def \_\_init\_\_(self, name):

self.name = name

def display\_info(self):

print(f"Name: {self.name}")

class Dog(Animal, Pet):

def \_\_init\_\_(self, species, name, breed):

Animal.\_\_init\_\_(self, species) Pet.\_\_init\_\_(self, name) self.breed = breed

def display\_info(self):

Animal.display\_info(self) Pet.display\_info(self) print(f"Breed: {self.breed}")

# Multilevel Inheritance class Shape:

def \_\_init\_\_(self, color): self.color = color

def display\_info(self):

print(f"Color: {self.color}")

class Square(Shape):

def \_\_init\_\_(self, color, side\_length): super().\_\_init\_\_(color)

self.side\_length = side\_length

def display\_info(self):

super().display\_info()

print(f"Side Length: {self.side\_length}")

class Cube(Square):

def \_\_init\_\_(self, color, side\_length, depth): super().\_\_init\_\_(color, side\_length) self.depth = depth

def display\_info(self):

super().display\_info()

print(f"Depth: {self.depth}")

# Hierarchical Inheritance

class Parent:

def \_\_init\_\_(self, name):

self.name = name

def display\_info(self):

print(f"Parent Name: {self.name}") class Child1(Parent):

def \_\_init\_\_(self, name, age): super().\_\_init\_\_(name)

self.age = age

def display\_info(self): super().display\_info()

print(f"Child1 Age: {self.age}")

class Child2(Parent):

def \_\_init\_\_(self, name, grade): super().\_\_init\_\_(name)

self.grade = grade

def display\_info(self): super().display\_info()

print(f"Child2 Grade: {self.grade}")

# Single Inheritance

print("Single Inheritance Example:")

car = Car("BMW", "X5")

car.display\_info()

print()

# Multiple Inheritance

print("Multiple Inheritance Example:")

dog = Dog("Canine", "Max", "Labrador") dog.display\_info()

print()

# Multilevel Inheritance

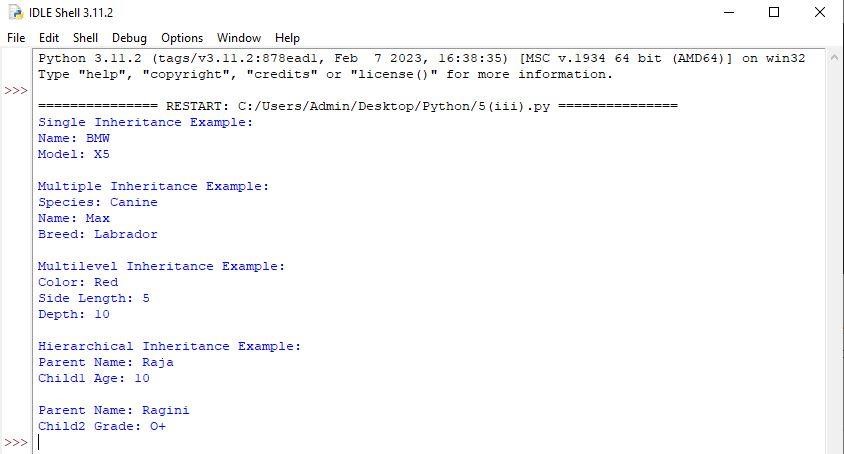
print("Multilevel Inheritance Example:") cube = Cube("Red", 5, 10) cube.display\_info()

print()

# Hierarchical Inheritance print("Hierarchical Inheritance Example:") child1 = Child1("Raja", 10) child1.display\_info()

print() child2 = Child2("Ragini", "O+") child2.display\_info()

**Output :-**



**5(iv). Python program for Polymorphism Demo(Operator overloading)**

class Rectangle:

def \_\_init\_\_(self, length, width):

self.length = length self.width = width

def area(self):

return self.length \* self.width def \_\_add\_\_(self, other):

if isinstance(other, Rectangle):

return Rectangle(self.length + other.length, self.width + other.width)

elif isinstance(other, (int, float)):

return Rectangle(self.length + other, self.width + other)

else:

raise TypeError("Unsupported operand type for +")

def \_\_str\_\_(self):

return f"Rectangle with length {self.length} and width {self.width}"

class Square:

def \_\_init\_\_(self, side):

self.side = side

def area(self):

return self.side \*\* 2

def \_\_add\_\_(self, other):

if isinstance(other, Square):

return Square(self.side + other.side) elif isinstance(other, (int, float)):

Return Square(self.side + other)

else:

raise TypeError("Unsupported operand type for +")

def \_\_str\_\_(self):

return f"Square with side {self.side}"

# Create a Rectangle object rectangle = Rectangle(4, 6)

print(rectangle) # Output: Rectangle with length 4 and width 6

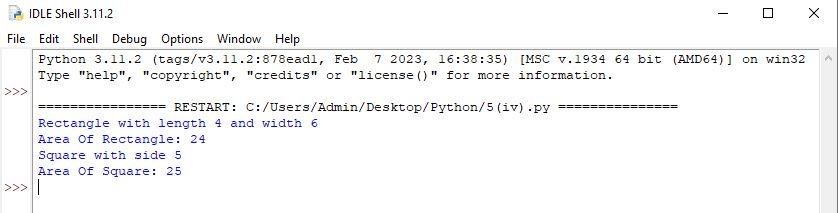
print("Area Of Rectangle:", rectangle.area())

# Create a Square object square = Square(5) print(square)

# Output: Square with side 5

print("Area Of Square:", square.area())

**Output :-**



**6. Programs for parsing of data, validations like Password, email, URL, etc.**

**Password Validation**

import re def validate\_password(password):

if re.match(r"^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*#?&])[A-Za-z\d@$!%\*#?&]{8,}$", password):

return True else:

return False

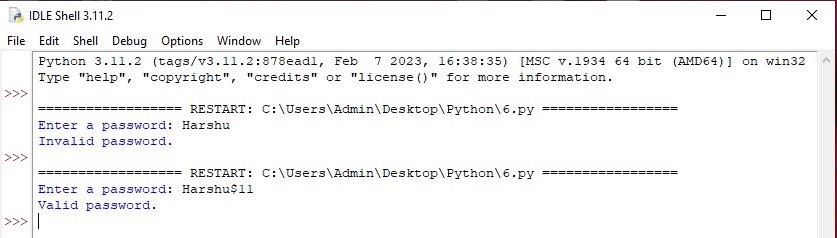
password = input("Enter a password: ")

if validate\_password(password): print("Valid password.")

else:

print("Invalid password.")

**Output :-**



**7. Programs for Pattern finding should be covered**.

def find\_pattern(string, pattern):

pattern\_length = len(pattern)

string\_length = len(string)

# Iterate through the string

for i in range(string\_length - pattern\_length + 1):

# Check if the substring matches the pattern

if string[i:i+pattern\_length] == pattern:

return i

# Return the starting index of the pattern

return -1

# Pattern not found in the string

# Test the function

text = input("Enter a string: ")

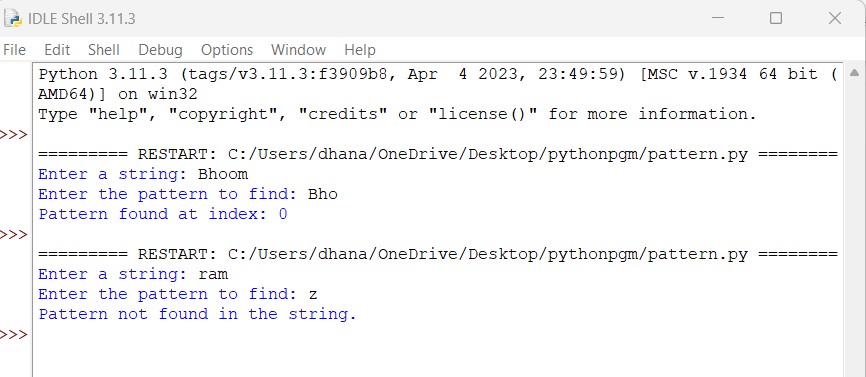
pattern = input("Enter the pattern to find: ") index = find\_pattern(text, pattern)

if index != -1:

print("Pattern found at index:", index) else:

print("Pattern not found in the string.")

**Output :-**



**8. Programs covering all the aspects of Exception handling, user defined exception, Multithreading should be covered.**

**8(i). Python program to perform Exception handling (Divide by zero Exception)**

try:

dividend = int(input("Enter the dividend: "))

divisor = int(input("Enter the divisor: "))

result = dividend / divisor

print("Result:", result) except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

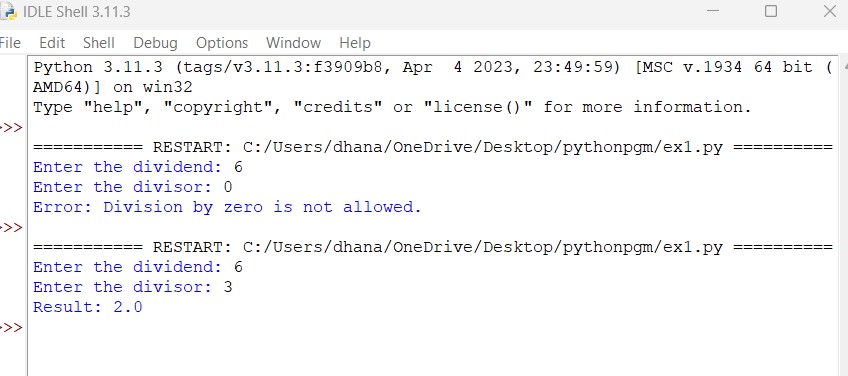
except ValueError:

print("Error: Invalid input. Please enter a valid number.")

except Exception as e:

print("An error occurred:", str(e))

**Output :-**



**8(ii). Python program for Integer Input Validation with Exception Handling (Example of ValueError Exception).**

while True:

try:

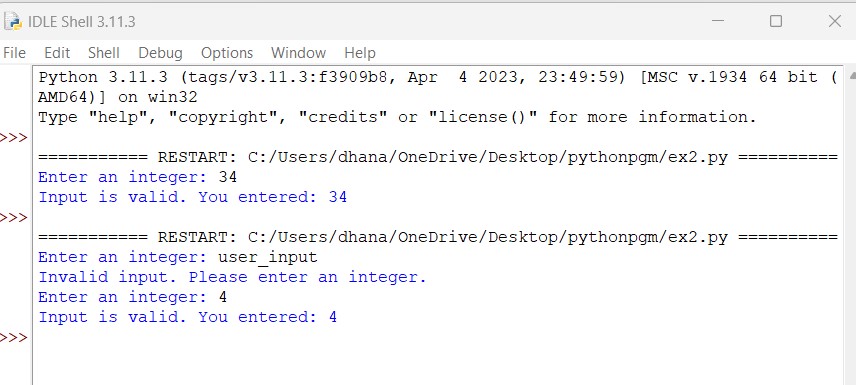
user\_input = int(input("Enter an integer: ")) break

except ValueError:

print("Invalid input. Please enter an integer.")

print("Input is valid. You entered:", user\_input)

**Output :-**



**8(iii). Program for IndexError Exception in Python with Example.**

try:

numbers = [1, 2, 3]

index = 4

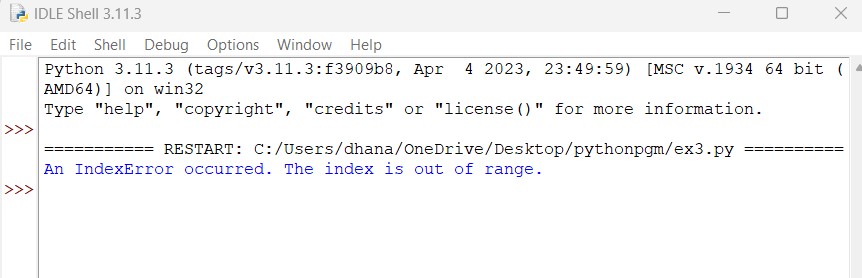
value = numbers[index]

print(f"The value at index {index} is {value}")

except IndexError:

print("An IndexError occurred. The index is out of range.")

**Output :-**



**8(iv). Python Program for Raising User Generated Exception**.

class MyException(Exception):

pass def divide\_numbers(a, b):

if b == 0:

raise MyException("Cannot divide by zero!") return a / b try:

num1 = int(input("Enter the numerator: "))

num2 = int(input("Enter the denominator: "))

result = divide\_numbers(num1, num2) print("Result:", result)

except MyException as e:

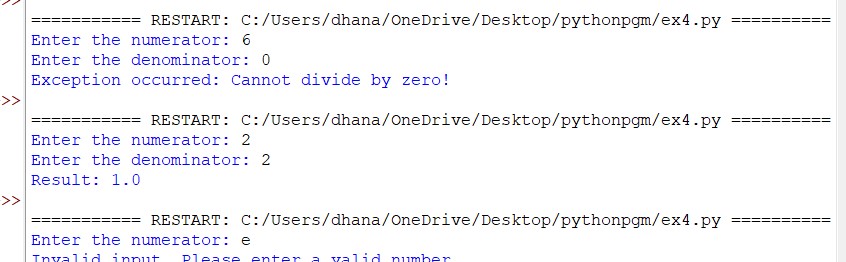
print("Exception occurred:", e)

except ValueError:

print("Invalid input. Please enter a valid number.") except Exception as e:

print("An error occurred:", e)

**Output :-**



**8(v). Python Program for Multithreading**

import threading

def print\_numbers():

for i in range(1, 11): print("Thread 1:", i)

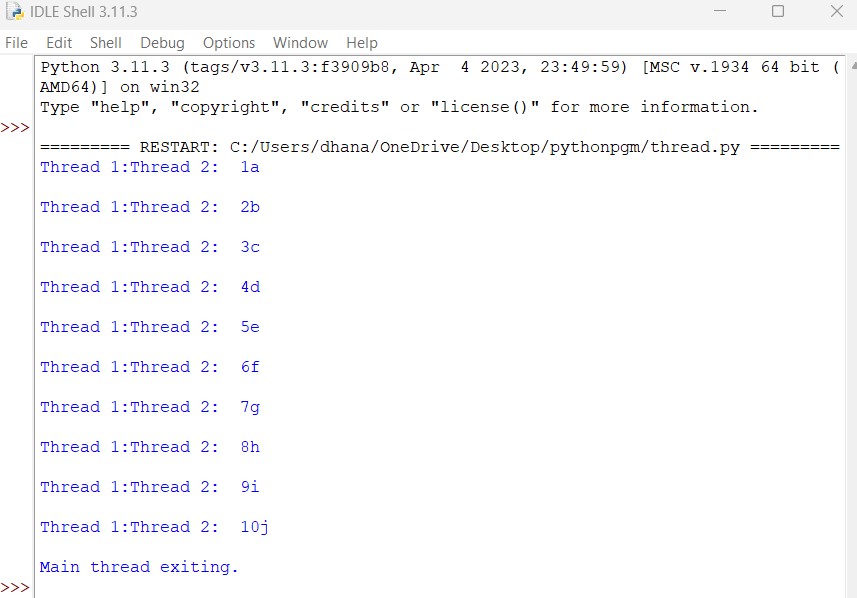
def print\_letters():

for letter in "abcdefghij": print("Thread 2:", letter)

thread1 = threading.Thread(target=print\_numbers) thread2 = threading.Thread(target=print\_letters) thread1.start() thread2.start() thread1.join() thread2.join()

print("Main thread exiting.")

**Output :-**



**9.Programs demonstrating the IO operations like reading from file, writing into file from different file types like data file, binary file, etc.**

**9(i). Python Program for writing log and change the log level.**

import logging

# Set up logging logging.basicConfig(filename='example.log',level=logging.DEBUG)

# Change the log level logging.getLogger().setLevel(logging.INFO)

# Create a console handler and set its log level console\_handler = logging.StreamHandler() console\_handler.setLevel(logging.DEBUG)

# Create a formatter and add it to the console handler

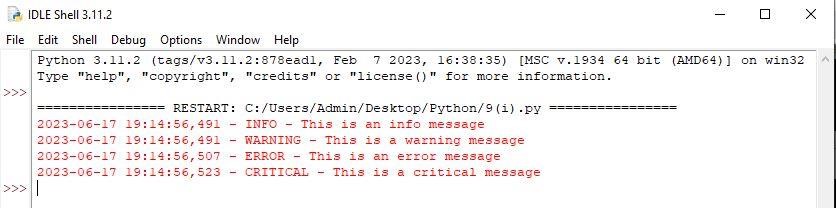
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s') console\_handler.setFormatter(formatter)

# Add the console handler to the root logger logging.getLogger().addHandler(console\_handler)

# Log messages

logging.debug('This is a debug message') logging.info('This is an info message') logging.warning('This is a warning message') logging.error('This is an error message') logging.critical('This is a critical message')

**Output :-**



**9(ii). Python Program for read functions and write functions such as read(), readline() and readlines() and write() and writelines().**

def read\_example():

# Open the file in read mode file = open('example.txt', 'r')

# Read the entire file content print("for read()")

content = file.read() print(content)

# Move the file pointer back to the beginning

file.seek(0)

# Read one line at a time print("for readline()")

line = file.readline()

while line:

print(line.strip()) # strip() removes the newline character

line = file.readline()

# Move the file pointer back to the beginning

file.seek(0)

# Read all lines at once and store them in a list print("for readlines()")

lines = file.readlines()

for line in lines:

print(line.strip())

# Close the file

file.close() def write\_example():

# Open the file in write mode

file = open('example.txt', 'w')

# Write a single line

print("for writeline()")

file.write("Hello, World!\n")

# Write multiple lines using a list of strings

print("for writelines()")

lines = ["This is line 1\n", "This is line 2\n", "This is line 3\n"] file.writelines(lines)

# Close the file

file.close()

# Read example print("Reading the file:") read\_example() # Write example print("Writing to the file:")

write\_example()

# Read again to verify the changes

print("Reading the modified file:")

read\_example()

**Output :-**



**9(iii). Python program on binary data file showing the file operations.**

# Creating a binary file def create\_binary\_file(file\_name):

try:

with open(file\_name, 'wb') as file:

file.write(b'\x48\x65\x6C\x6C\x6F\x20\x57\x6F\x72\x6C\x64')

# Writing binary data

print(f"Binary file '{file\_name}' created successfully.")

except IOError:

print(f"An error occurred while creating the binary file.")

# Reading binary file

def read\_binary\_

file(file\_name):

try:

with open(file\_name, 'rb') as file:

data = file.read()

# Reading binary data

print(f"Binary file '{file\_name}' content: {data}")

except IOError:

print(f"An error occurred while reading the binary file.")

# Updating binary file

def update\_binary\_file(file\_name):

try:

with open(file\_name, 'r+b') as file:

file.seek(6)

# Move the file pointer to the desired position file.write(b'\x57\x6F\x72\x6C\x64')

# Updating binary data

print(f"Binary file '{file\_name}' updated successfully.")

except IOError:

print(f"An error occurred while updating the binary file.")

# Deleting binary file

def delete\_binary\_file(file\_name):

try:

import os

os.remove(file\_name)

print(f"Binary file '{file\_name}' deleted successfully.")

except FileNotFoundError:

print(f"Binary file '{file\_name}' not found.")

except PermissionError:

print(f"You don't have permission to delete the binary file.")

# Main program if \_\_name\_\_ == "\_\_main\_\_":

file\_name = "binary\_data.bin"

create\_binary\_file(file\_name)

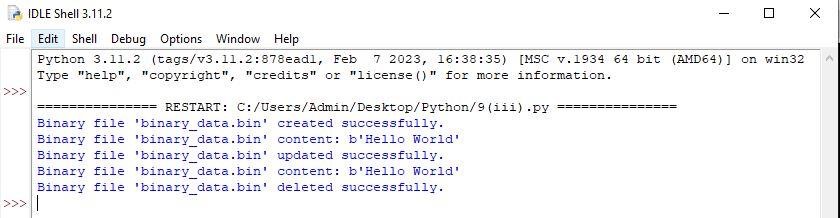
read\_binary\_file(file\_name)

update\_binary\_file(file\_name)

read\_binary\_file(file\_name)

delete\_binary\_file(file\_name)

**Output :-**



**10.Programs to perform searching, adding, updating the content from the file.**

**# Searching for a specific word** search\_word = "Banana"

with open("data.txt", "r") as file:

for line in file:

if search\_word in line:

print(f"Found '{search\_word}' in the file.") break

else:

print(f"'{search\_word}' not found in the file.")

**Output:**

Found 'Banana' in the file.

**# Adding new content**

new\_content = "Grapes"

with open("data.txt", "a") as file: file.write("\n" + new\_content) print(f"'{new\_content}' added to the file.")

**Output:**

'Grapes' added to the file.

**# Updating content**

old\_word = "Orange" new\_word = "Pineapple"

with open("data.txt", "r") as file:

lines = file.readlines()

with open("data.txt", "w") as file: f

or line in lines:

if old\_word in line:

line = line.replace(old\_word, new\_word)

file.write(line)

print(f"Updated '{old\_word}' to '{new\_word}' in the file.")

**Output:**

Updated 'Orange' to 'Pineapple' in the file.

**11**.**Program for performing CRUD operation with MongoDB and Python.**

from pymongo

import MongoClient

**# Connect to MongoDB**

client = MongoClient('mongodb://localhost:27017/') database = client['mydatabase']

collection = database['mycollection']

# Create operation new\_document = {

'name': 'John Doe',

'age': 30,

'email': 'johndoe@example.com'

}

inserted\_doc = collection.insert\_one(new\_document)

print(f"Created document with ID: {inserted\_doc.inserted\_id}")

**# Read operation**

result = collection.find\_one({'name': 'John Doe'}) print("Read document:")

print(result)

**# Update operation**

update\_query = {'name': 'John Doe'} new\_values = {'$set': {'age': 32}}

collection.update\_one(update\_query, new\_values)

print("Updated document:")

updated\_result = collection.find\_one({'name': 'John Doe'}) print(updated\_result)

**# Delete operation**

delete\_query = {'name': 'John Doe'} collection.delete\_one(delete\_query)

print("Deleted document:")

deleted\_result = collection.find\_one({'name': 'John Doe'}) print(deleted\_result)

**Output:**

Created document with ID: ObjectId('6093468eac9a3d4d5f5a16f2')

Read document:

{'\_id': ObjectId('6093468eac9a3d4d5f5a16f2'), 'name': 'John Doe', 'age': 30, 'email': 'johndoe@example.com'} Updated document:

{'\_id': ObjectId('6093468eac9a3d4d5f5a16f2'), 'name': 'John Doe', 'age': 32, 'email': 'johndoe@example.com'} Deleted document:

None

**12**.**Basic programs with NumPy as Array, Searching and Sorting, date & time and String handling.**

**Array**

import numpy as np

# Create an array

arr = np.array([1, 2, 3, 4, 5])

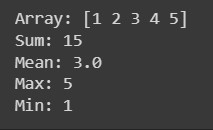
# Print the array

print("Array:", arr)

# Perform array operations sum\_result = np.sum(arr) mean\_result = np.mean(arr) max\_result = np.max(arr) min\_result = np.min(arr)

# Print the results print("Sum:", sum\_result) print("Mean:", mean\_result) print("Max:", max\_result) print("Min:", min\_result)

**Output :-**



**Searching and Sorting**

import numpy as np

# Create an array

arr = np.array([5, 2, 8, 1, 4])

# Sort the array sorted\_

arr = np.sort(arr)

# Search for a value search\_value = 4

index = np.where(sorted\_arr == search\_value)

# Print the sorted array and search result

print("Sorted Array:", sorted\_arr)

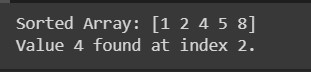
if len(index[0]) > 0:

print(f"Value {search\_value} found at index {index[0][0]}.")

else:

print(f"Value {search\_value} not found.")

**Output :-**



**date & time**

import datetime

# Get the current date and time

current\_datetime = datetime.datetime.now()

# Print the current date and time

print("Current Date and Time:", current\_datetime)

# Format the date and time

formatted\_datetime = current\_datetime.strftime("%Y-%m-%d %H:%M:%S")

# Print the formatted date and time

print("Formatted Date and Time:", formatted\_datetime)

**Output :-**



**String Handling**

import numpy as np

# Create a string array

arr = np.array(["apple", "banana", "cherry"])

# Concatenate strings

concatenated\_str = np.char.add(arr, " is a fruit.")

uppercased\_str = np.char.upper(arr)

# Print the results

print("Concatenated Strings:", concatenated\_str) print("Uppercased Strings:", uppercased\_str)

**Output :-**



**13.Programs for series and data frames should be covered.**

**Series**

import pandas as pd

# Create a series

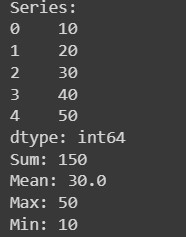
data = [10, 20, 30, 40, 50] series = pd.Series(data)

# Print the series print("Series:") print(series)

# Perform series operations sum\_result = series.sum() mean\_result = series.mean() max\_result = series.max() min\_result = series.min()

# Print the results print("Sum:", sum\_result) print("Mean:", mean\_result) print("Max:", max\_result) print("Min:", min\_result)

**Output :-**



**Data Frames**

import pandas as pd

# Create a data frame

data = {'Name': ['John', 'Emma', 'Ryan', 'Emily'],

'Age': [25, 30, 35, 28],

'City': ['New York', 'London', 'Sydney', 'Paris']}

df = pd.DataFrame(data)

# Print the data frame print("Data Frame:") print(df)

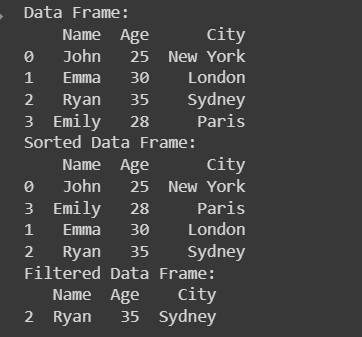
# Perform data frame operations df\_sorted = df.sort\_values('Age') df\_filtered = df[df['Age'] > 30]

# Print the results

print("Sorted Data Frame:") print(df\_sorted)

print("Filtered Data Frame:") print(df\_filtered)

**Output:-**



**14.Programs to demonstrate data pre-processing and data handling with data frame.**

**Data Pre-processing with DataFrame** import pandas as pd

# Creating a DataFrame

data = {'Name': ['John', 'Emma', 'Michael', 'Sophia', 'William'],

'Age': [32, 28, 45, 36, 39],

'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney'], 'Salary': [50000, 65000, 80000, 45000, 70000]}

df = pd.DataFrame(data)

# Displaying the original DataFrame print("Original DataFrame:")

print(df)

# Data Pre-processing

df['Age'] = df['Age'] + 1

# Incrementing age by 1

df['Salary'] = df['Salary'] \* 1.1

# Increasing salary by 10% df['City'] = df['City'].str.upper()

# Converting city names to uppercase

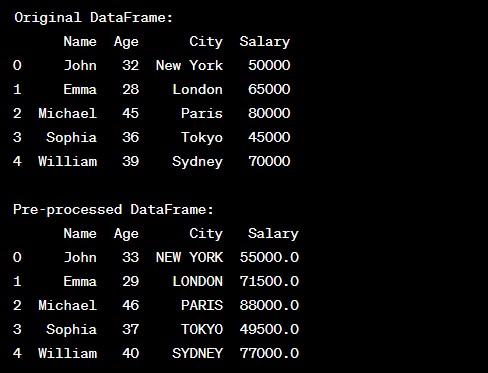
# Displaying the pre-processed

DataFrame

print("\nPre-processed DataFrame:")

print(df)

**Output :-**



**Data Handling with DataFrame** import pandas as pd

# Creating a DataFrame

data = {'Name': ['John', 'Emma', 'Michael', 'Sophia', 'William'],

'Age': [32, 28, 45, 36, 39],

'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney'], 'Salary': [50000, 65000, 80000, 45000, 70000]}

df = pd.DataFrame(data)

# Displaying the original DataFrame

print("Original DataFrame:") print(df)

# Data Handling

# Sorting the DataFrame by Age in descending order df\_sorted = df.sort\_values(by='Age', ascending=False)

# Selecting rows where Salary is greater than 60000 df\_filtered = df[df['Salary'] > 60000]

# Calculating the average salary

average\_salary = df['Salary'].mean()

# Displaying the sorted DataFrame, filtered DataFrame, and average salary print("\nSorted DataFrame (by Age):")

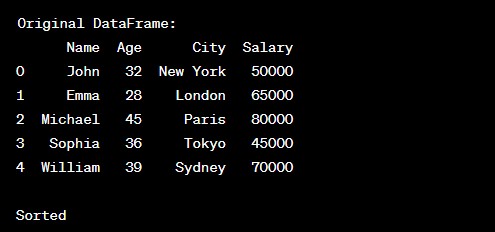
print(df\_sorted)

print("\nFiltered DataFrame (Salary > 60000):")

print(df\_filtered)

print(f"\nAverage Salary: {average\_salary}")

**Output :-**



**15. Program for data visualization should be covered.**

**15(i). Python program for simple graph(line) generation.**

import matplotlib.pyplot as plt

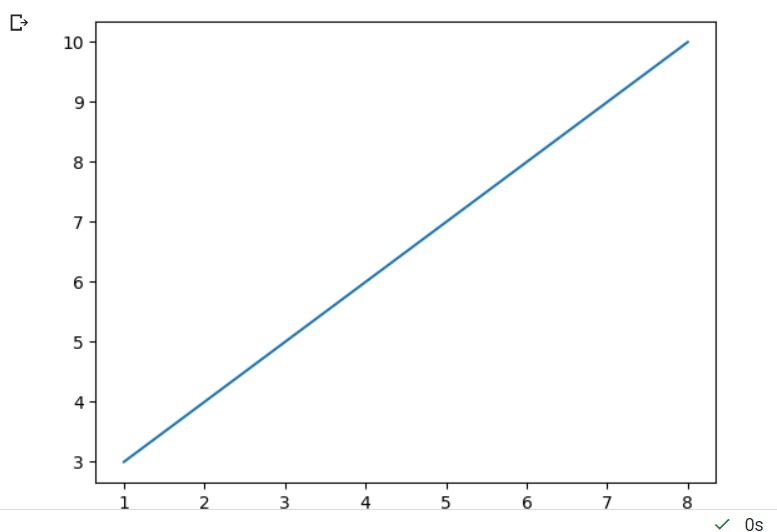
import numpy as np

xpoints = np.array([1, 8])

ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints) plt.show()

**Output :-**



**15(ii). Python program for Multiline graph.**

# importing package

import matplotlib.pyplot as plt import numpy as np

# create data

x = [1,2,3,4,5]

y = [3,3,3,3,3]

# plot lines

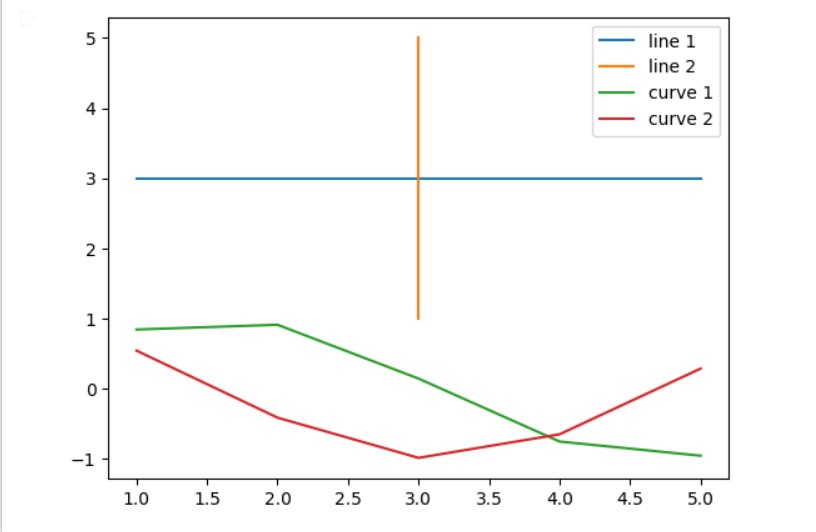
plt.plot(x, y, label = "line 1")

plt.plot(y, x, label = "line 2")

plt.plot(x, np.sin(x), label = "curve 1") plt.plot(x, np.cos(x), label = "curve 2") plt.legend()

plt.show()

**Output :-**



**15(iii). Python program for creating scatter, bar, histogram, pie chart.**

import matplotlib.pyplot as plt

import numpy as np

#day one, the age and speed of 13 cars:

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])

y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)

#day two, the age and speed of 15 cars:

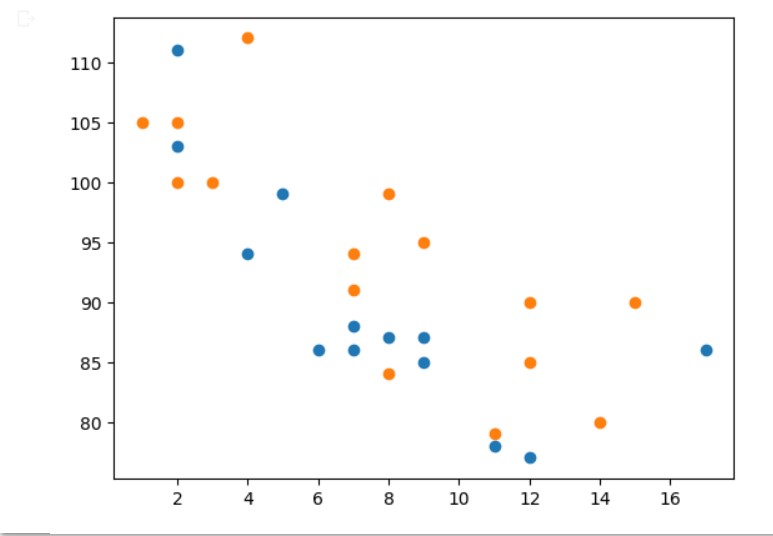
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])

y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])

plt.scatter(x, y)

plt.show()

**Output :-**



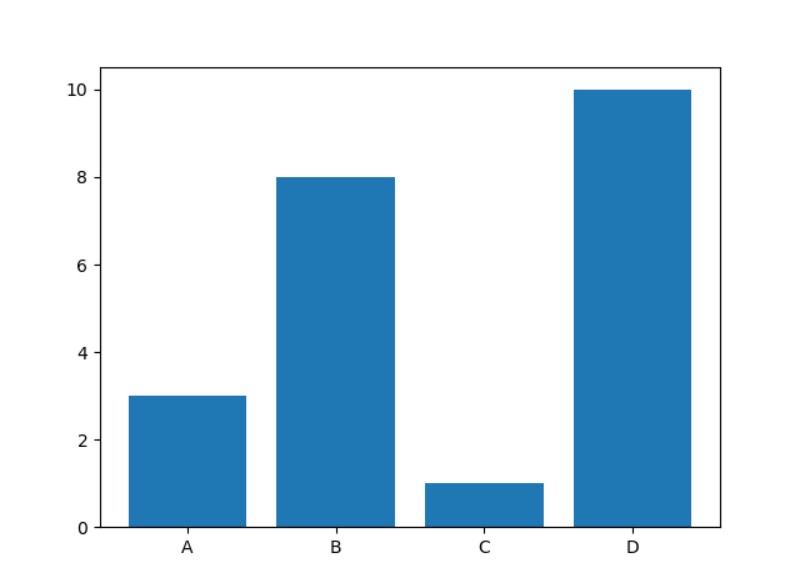
**Bar :**

import matplotlib.pyplot as plt import numpy as np

x = np.array(["A", "B", "C", "D"]) y = np.array([3, 8, 1, 10])

plt.bar(x,y) plt.show()

**Output :-**



**Histogram :**

import matplotlib.pyplot as plt

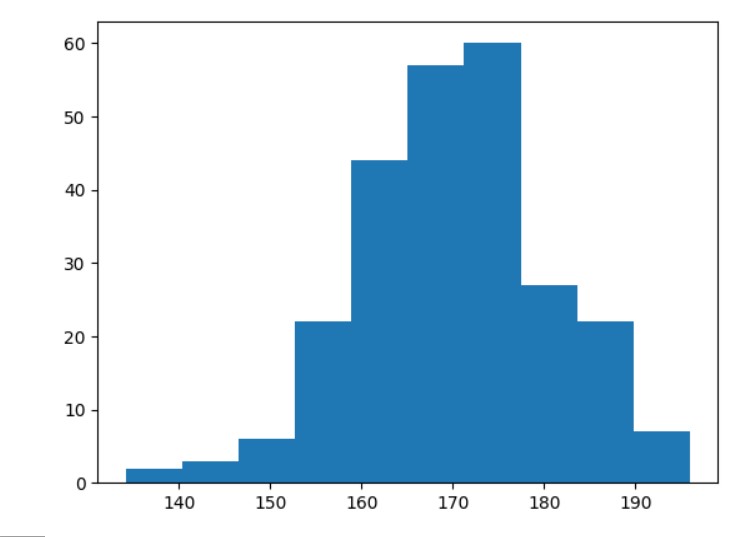
import numpy as np

1. = np.random.normal(170, 10, 250)

plt.hist(x)

plt.show()

**Output :-**



**Pie chart:**

import matplotlib.pyplot as plt

import numpy as np

1. = np.array([35, 25, 25, 15])

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)

plt.show()

**Output :-**

