DEPARTMENT OF COMPUTER APPLICATIONS

LAB MANUAL

Artificial Intelligence Using Python

YEAR: 2025-26

Name of the Student:

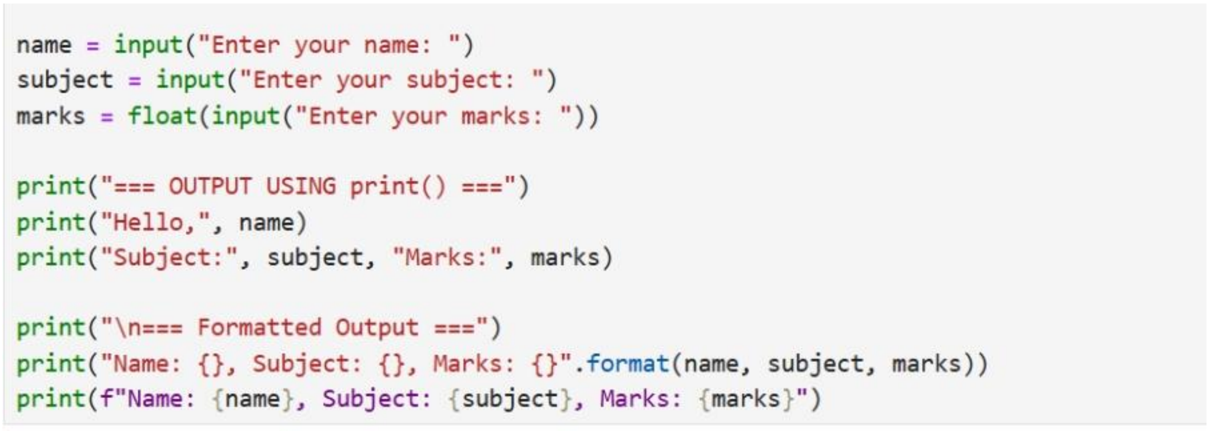
____

Register Number :

____

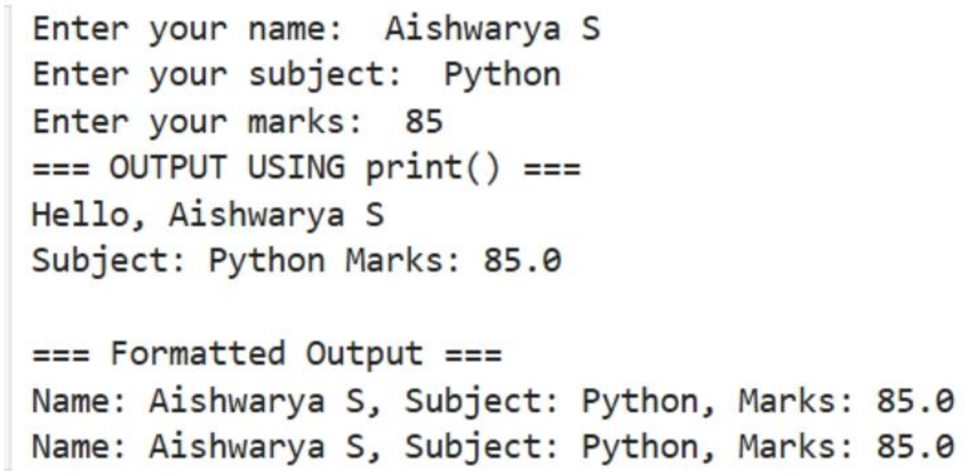
Signature of the Faculty Signature of the HoD

Program 1: Console Input and Output



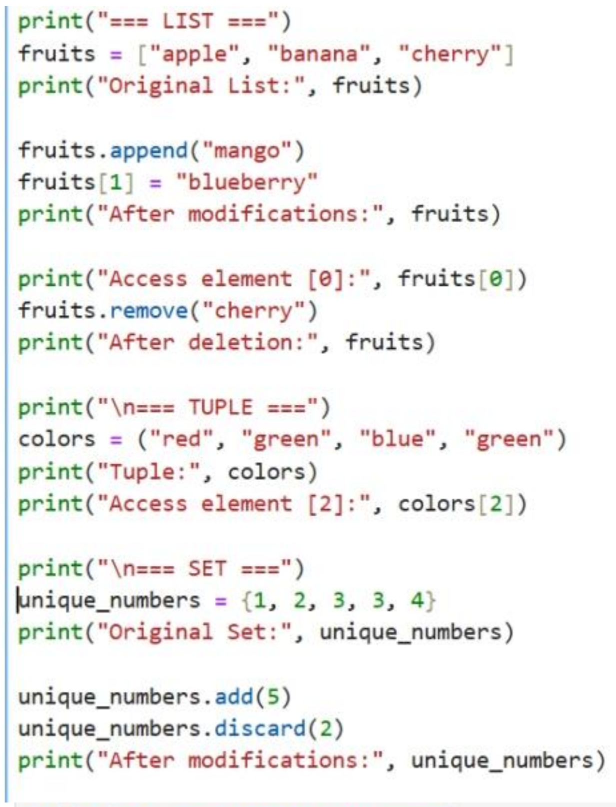
```python
name = input("Enter your name: ")
subject = input("Enter your subject: ")
marks = float(input("Enter your marks: "))

print("=== OUTPUT USING print() ===")
print("Hello,", name)
print("Subject:", subject, "Marks:", marks)

print("\n=== Formatted Output ===")
print("Name: {}, Subject: {}, Marks: {}".format(name, subject, marks))
print(f"Name: {name}, Subject: {subject}, Marks: {marks}")
```

Output:

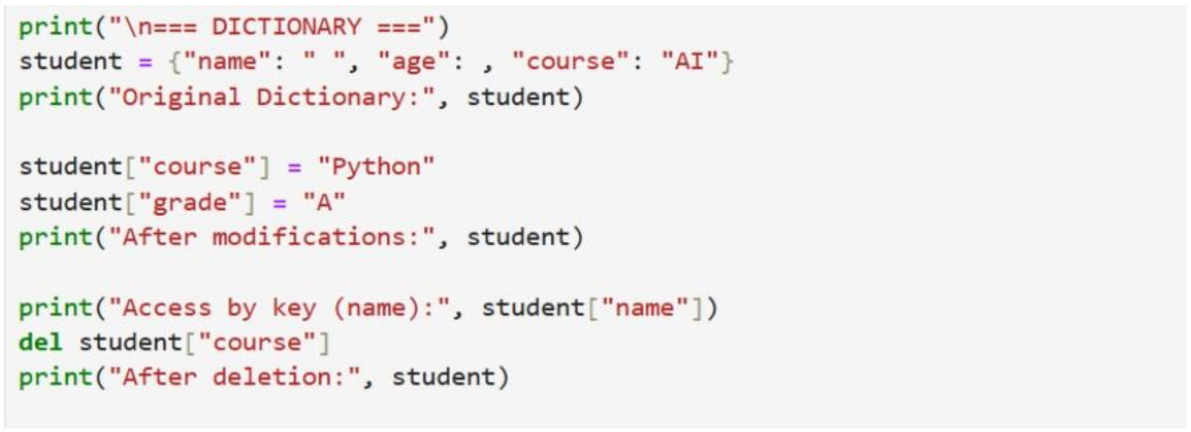


```
Enter your name:  Aishwarya S
Enter your subject:  Python
Enter your marks:  85
=== OUTPUT USING print() ===
Hello, Aishwarya S
Subject: Python Marks: 85.0

=== Formatted Output ===
Name: Aishwarya S, Subject: Python, Marks: 85.0
Name: Aishwarya S, Subject: Python, Marks: 85.0
```

Program 2: Implementation of Python Collections

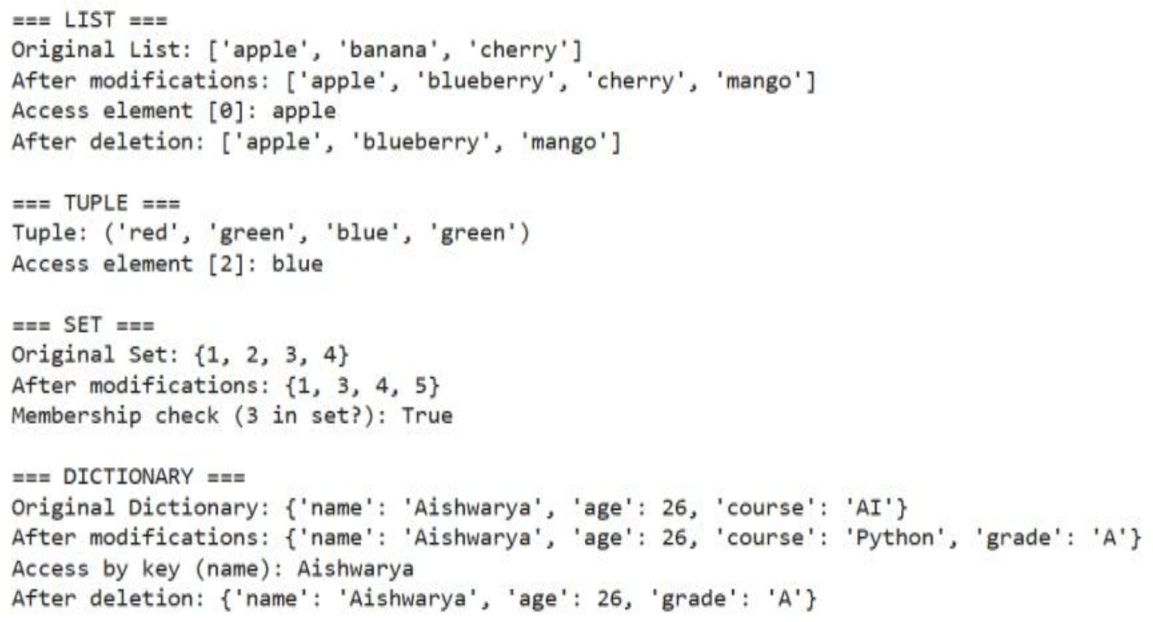


```python
print("=== LIST ===")
fruits = ["apple", "banana", "cherry"]
print("Original List:", fruits)

fruits.append("mango")
fruits[1] = "blueberry"
print("After modifications:", fruits)

print("Access element [0]:", fruits[0])
fruits.remove("cherry")
print("After deletion:", fruits)

print("\n=== TUPLE ===")
colors = ("red", "green", "blue", "green")
print("Tuple:", colors)
print("Access element [2]:", colors[2])

print("\n=== SET ===")
unique_numbers = {1, 2, 3, 3, 4}
print("Original Set:", unique_numbers)

unique_numbers.add(5)
unique_numbers.discard(2)
print("After modifications:", unique_numbers)
```



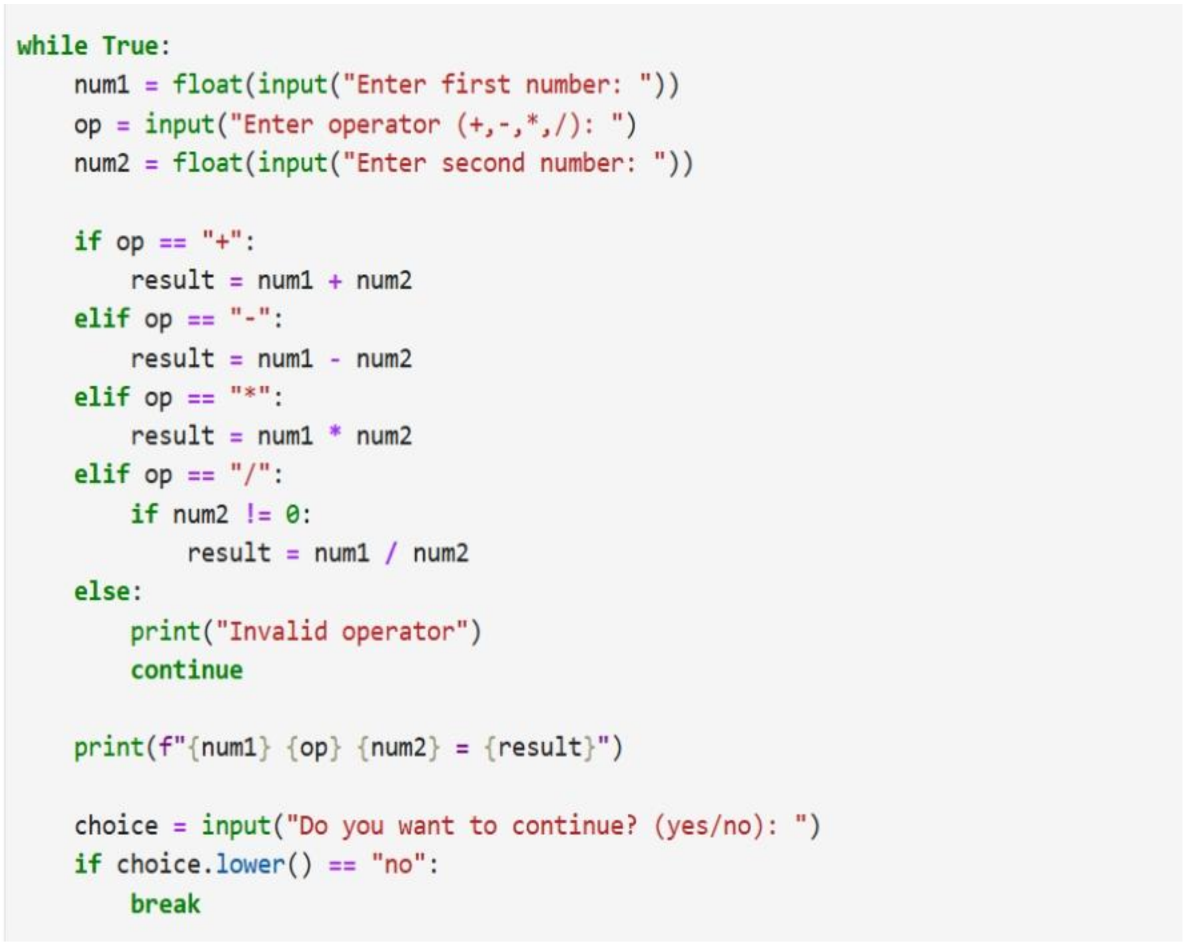
```python
print("\n=== DICTIONARY ===")
student = {"name": " ", "age": , "course": "AI"}
print("Original Dictionary:", student)

student["course"] = "Python"
student["grade"] = "A"
print("After modifications:", student)

print("Access by key (name):", student["name"])
del student["course"]
print("After deletion:", student)
```

Output:



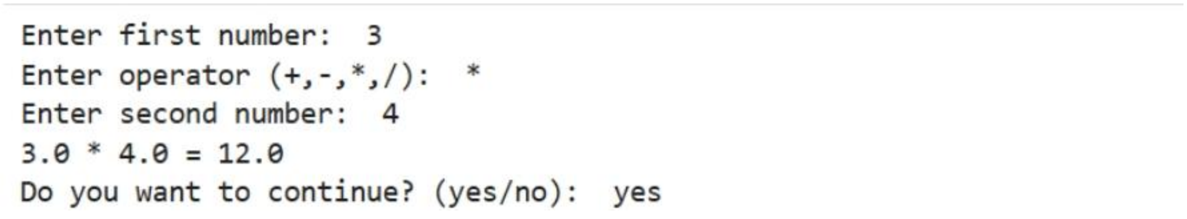
```
=== LIST ===
Original List: ['apple', 'banana', 'cherry']
After modifications: ['apple', 'blueberry', 'cherry', 'mango']
Access element [0]: apple
After deletion: ['apple', 'blueberry', 'mango']

=== TUPLE ===
Tuple: ('red', 'green', 'blue', 'green')
Access element [2]: blue

=== SET ===
Original Set: {1, 2, 3, 4}
After modifications: {1, 3, 4, 5}
Membership check (3 in set?): True

=== DICTIONARY ===
Original Dictionary: {'name': 'Aishwarya', 'age': 26, 'course': 'AI'}
After modifications: {'name': 'Aishwarya', 'age': 26, 'course': 'Python', 'grade': 'A'}
Access by key (name): Aishwarya
After deletion: {'name': 'Aishwarya', 'age': 26, 'grade': 'A'}
```

Program 3: Create a simple calculator



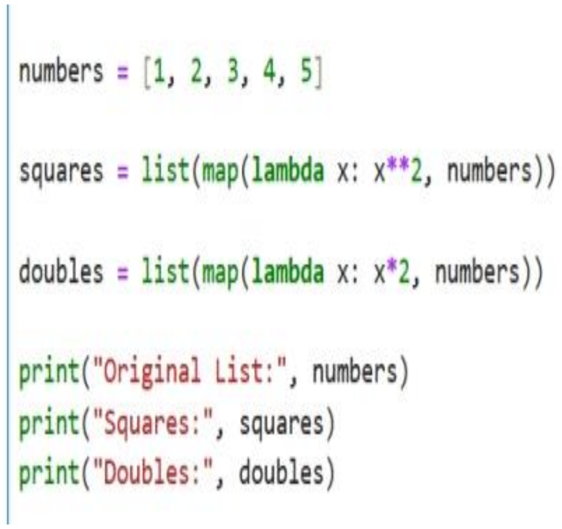
```python
while True:
    num1 = float(input("Enter first number: "))
    op = input("Enter operator (+,-,*,/): ")
    num2 = float(input("Enter second number: "))

    if op == "+":
        result = num1 + num2
    elif op == "-":
        result = num1 - num2
    elif op == "*":
        result = num1 * num2
    elif op == "/":
        if num2 != 0:
            result = num1 / num2
    else:
        print("Invalid operator")
        continue

    print(f"{num1} {op} {num2} = {result}")

    choice = input("Do you want to continue? (yes/no): ")
    if choice.lower() == "no":
        break
```

Output:



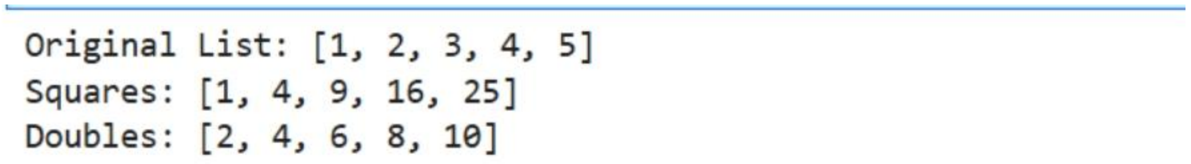
```
Enter first number:  3
Enter operator (+,-,*,/):  *
Enter second number:  4
3.0 * 4.0 = 12.0
Do you want to continue? (yes/no):  yes
```

Program 4: Demonstration of Lambda and Map Functions in Python



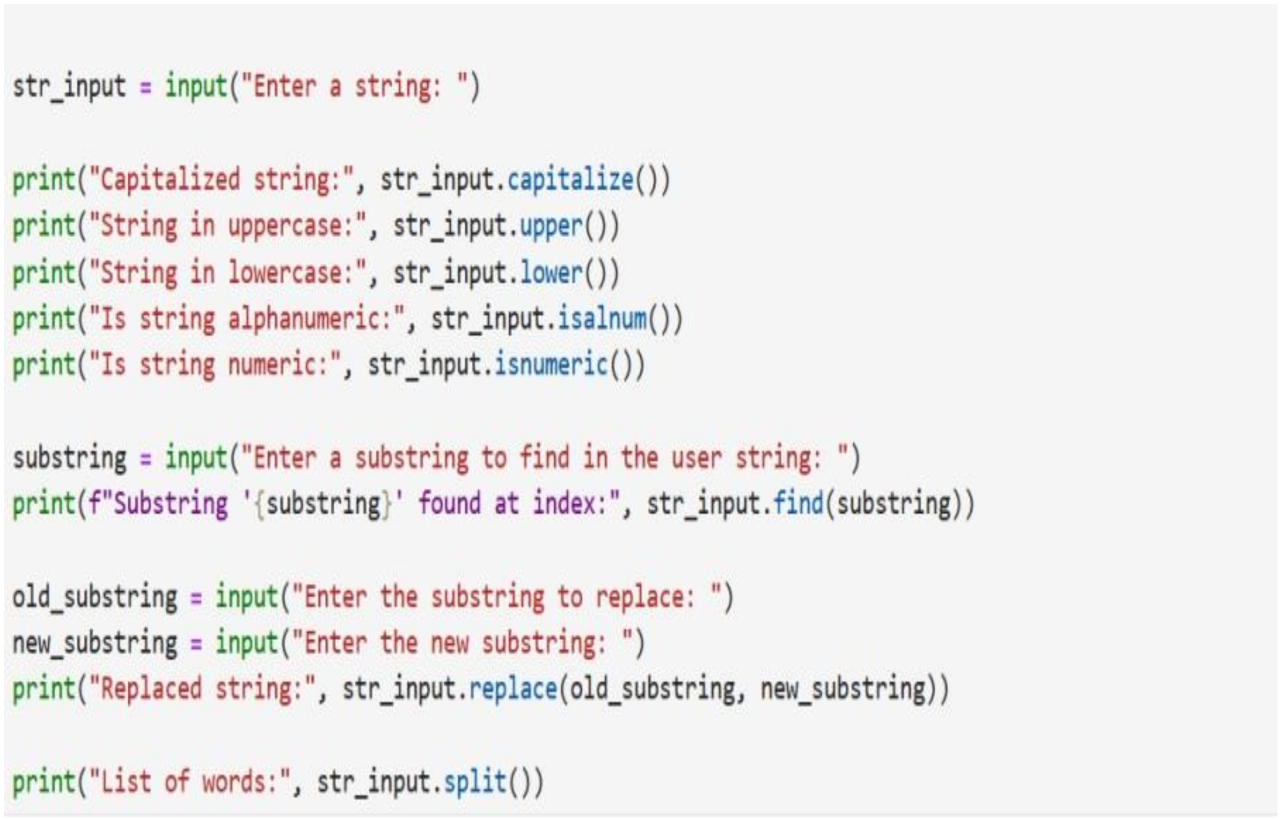
```python
numbers = [1, 2, 3, 4, 5]

squares = list(map(lambda x: x**2, numbers))

doubles = list(map(lambda x: x*2, numbers))

print("Original List:", numbers)
print("Squares:", squares)
print("Doubles:", doubles)
```

Output:



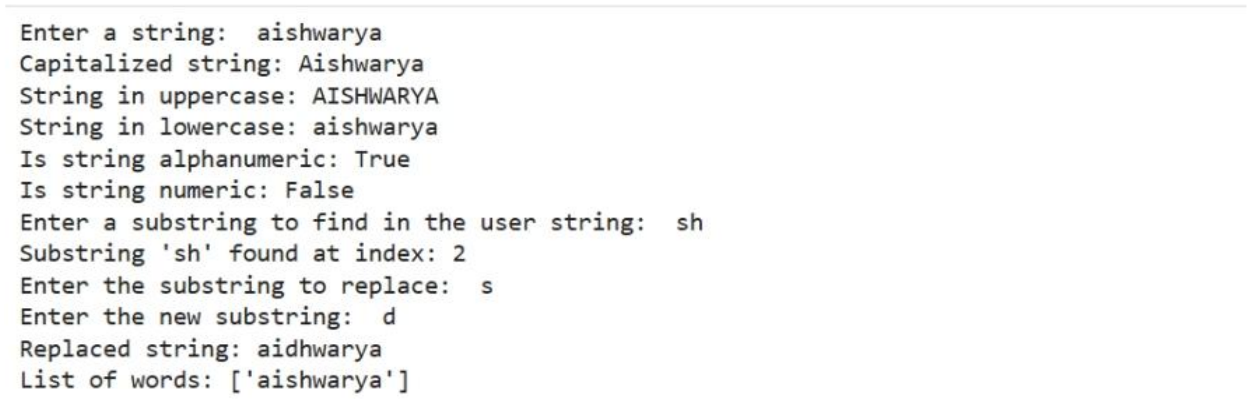
```
Original List: [1, 2, 3, 4, 5]
Squares: [1, 4, 9, 16, 25]
Doubles: [2, 4, 6, 8, 10]
```

Program 5: Demonstration of common String Functions in Python



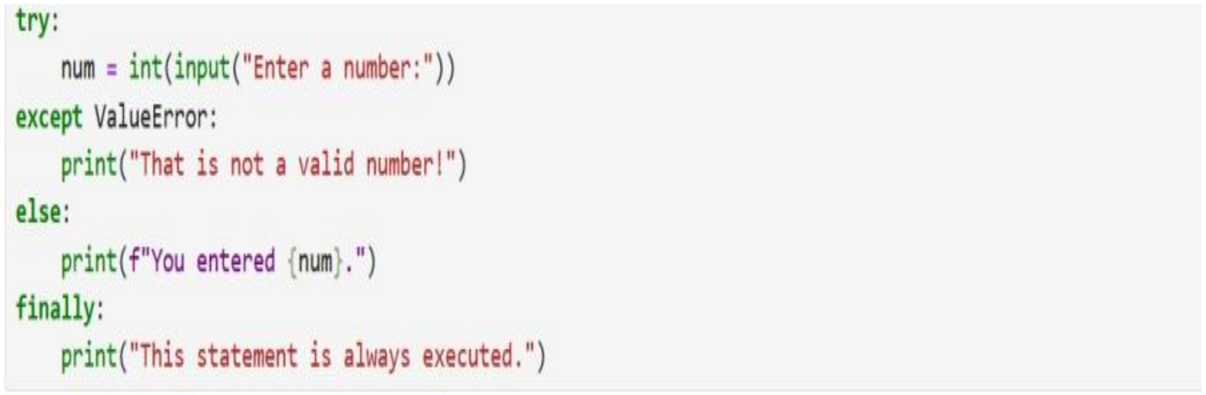
```
str_input = input("Enter a string: ")

print("Capitalized string:", str_input.capitalize())
print("String in uppercase:", str_input.upper())
print("String in lowercase:", str_input.lower())
print("Is string alphanumeric:", str_input.isalnum())
print("Is string numeric:", str_input.isnumeric())

substring = input("Enter a substring to find in the user string: ")
print(f"Substring '{substring}' found at index:", str_input.find(substring))

old_substring = input("Enter the substring to replace: ")
new_substring = input("Enter the new substring: ")
print("Replaced string:", str_input.replace(old_substring, new_substring))

print("List of words:", str_input.split())
```

Output:



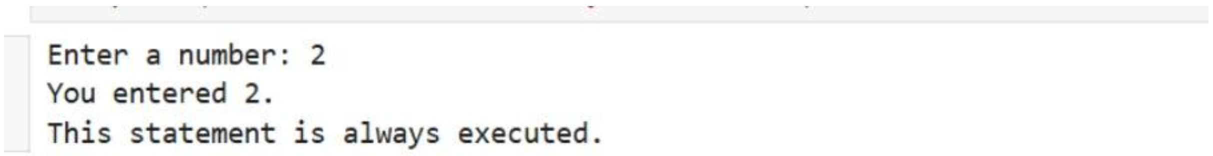
```
Enter a string:  aishwarya
Capitalized string: Aishwarya
String in uppercase: AISHWARYA
String in lowercase: aishwarya
Is string alphanumeric: True
Is string numeric: False
Enter a substring to find in the user string:  sh
Substring 'sh' found at index: 2
Enter the substring to replace:  s
Enter the new substring:  d
Replaced string: aidhwarya
List of words: ['aishwarya']
```

Program 6: Demonstration of Exception handling



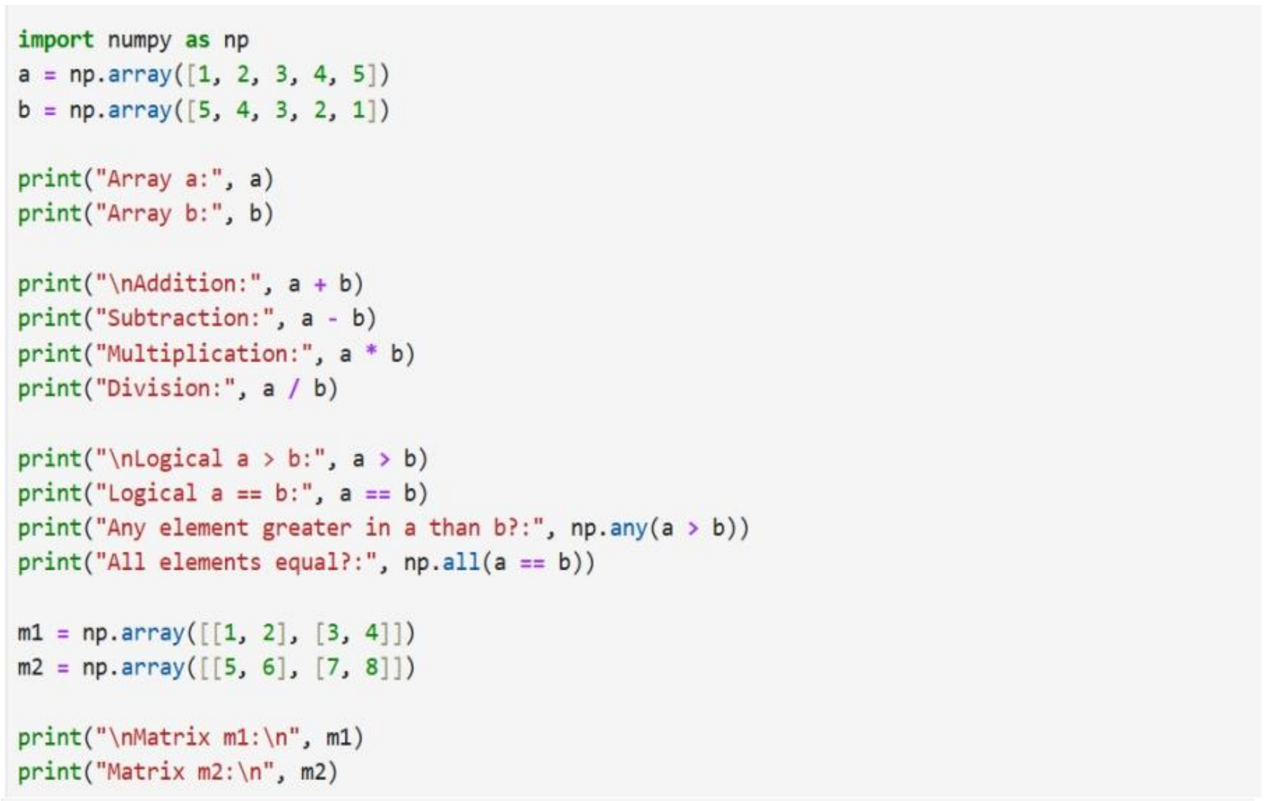
```
try:
    num = int(input("Enter a number:"))
except ValueError:
    print("That is not a valid number!")
else:
    print(f"You entered {num}.")
finally:
    print("This statement is always executed.")
```

Output:

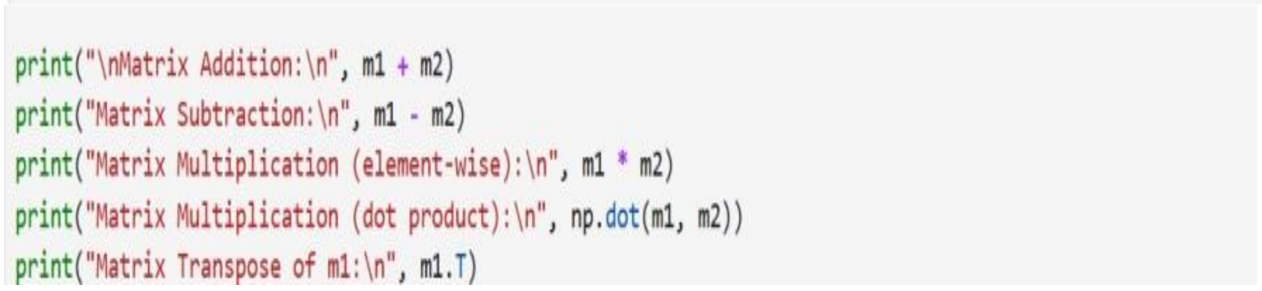


```
Enter a number: 2
You entered 2.
This statement is always executed.
```

Program 7 :Create arrays using numpy and perform array operations.

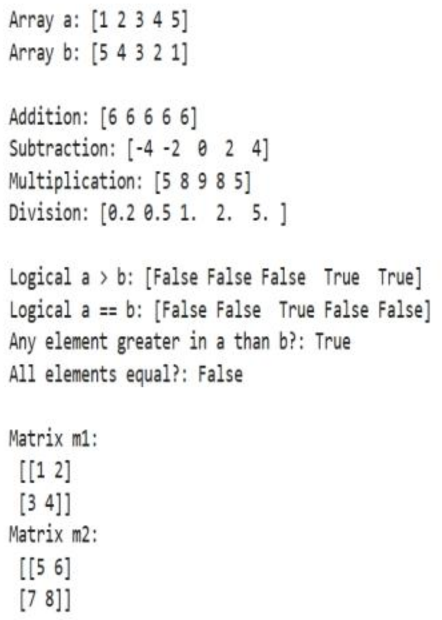


```python
import numpy as np
a = np.array([1, 2, 3, 4, 5])
b = np.array([5, 4, 3, 2, 1])

print("Array a:", a)
print("Array b:", b)

print("\nAddition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)

print("\nLogical a > b:", a > b)
print("Logical a == b:", a == b)
print("Any element greater in a than b?:", np.any(a > b))
print("All elements equal?:", np.all(a == b))

m1 = np.array([[1, 2], [3, 4]])
m2 = np.array([[5, 6], [7, 8]])

print("\nMatrix m1:\n", m1)
print("Matrix m2:\n", m2)
```

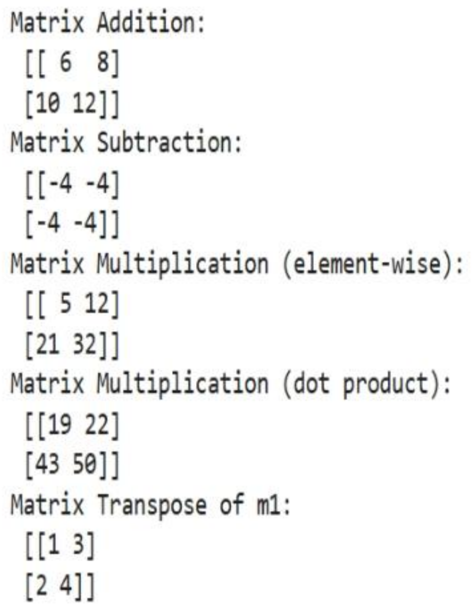


```python
print("\nMatrix Addition:\n", m1 + m2)
print("Matrix Subtraction:\n", m1 - m2)
print("Matrix Multiplication (element-wise):\n", m1 * m2)
print("Matrix Multiplication (dot product):\n", np.dot(m1, m2))
print("Matrix Transpose of m1:\n", m1.T)
```

Output:

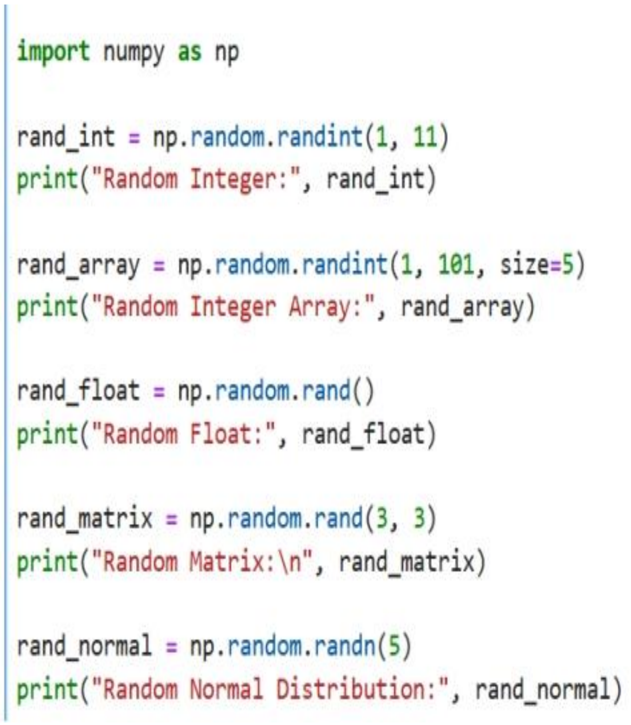


```
Array a: [1 2 3 4 5]
Array b: [5 4 3 2 1]

Addition: [6 6 6 6 6]
Subtraction: [-4 -2  0  2  4]
Multiplication: [5 8 9 8 5]
Division: [0.2 0.5 1.  2.  5. ]

Logical a > b: [False False False  True  True]
Logical a == b: [False False  True False False]
Any element greater in a than b?: True
All elements equal?: False

Matrix m1:
 [[1 2]
 [3 4]]
Matrix m2:
 [[5 6]
 [7 8]]
```



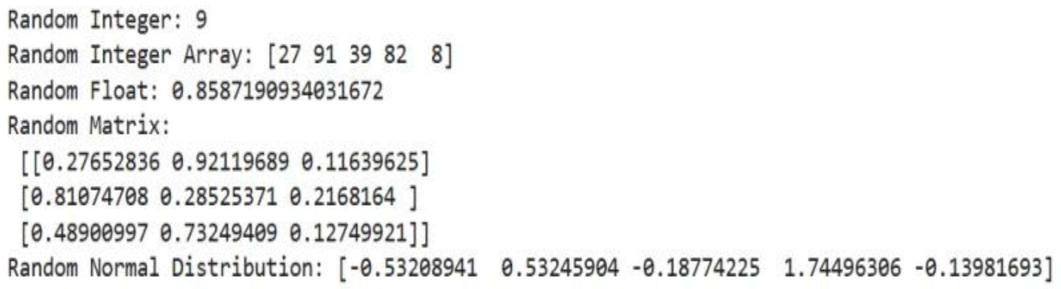
```
Matrix Addition:
 [[ 6  8]
 [10 12]]
Matrix Subtraction:
 [[-4 -4]
 [-4 -4]]
Matrix Multiplication (element-wise):
 [[ 5 12]
 [21 32]]
Matrix Multiplication (dot product):
 [[19 22]
 [43 50]]
Matrix Transpose of m1:
 [[1 3]
 [2 4]]
```

Program 8: Generate random numbers using numpy.



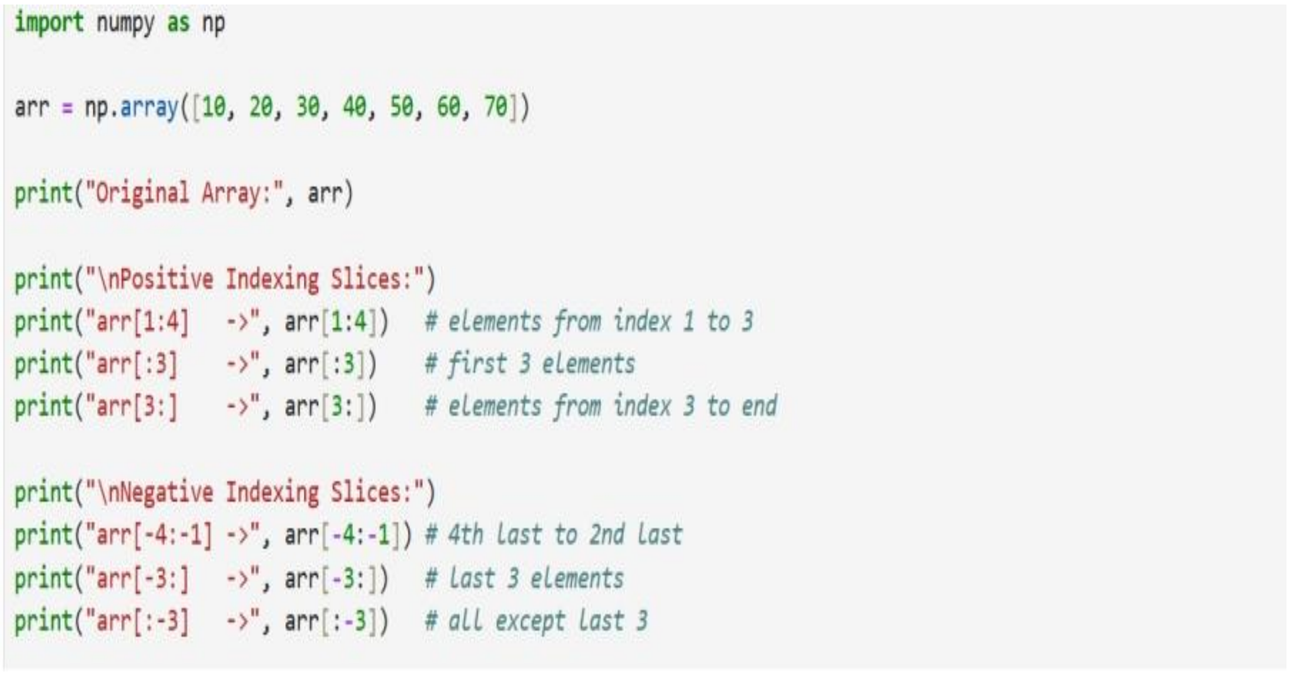
```python
import numpy as np

rand_int = np.random.randint(1, 11)
print("Random Integer:", rand_int)

rand_array = np.random.randint(1, 101, size=5)
print("Random Integer Array:", rand_array)

rand_float = np.random.rand()
print("Random Float:", rand_float)

rand_matrix = np.random.rand(3, 3)
print("Random Matrix:\n", rand_matrix)

rand_normal = np.random.randn(5)
print("Random Normal Distribution:", rand_normal)
```

Output:



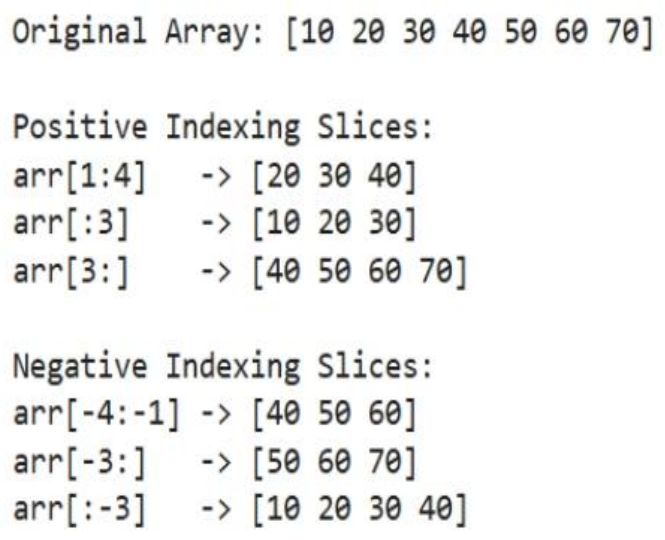
```
Random Integer: 9
Random Integer Array: [27 91 39 82  8]
Random Float: 0.8587190934031672
Random Matrix:
 [[0.27652836 0.92119689 0.11639625]
 [0.81074708 0.28525371 0.2168164 ]
 [0.48900997 0.73249409 0.12749921]]
Random Normal Distribution: [-0.53208941  0.53245904 -0.18774225  1.74496306 -0.13981693]
```

Program 9: Create a NumPy array and demonstrate slicing using both positive and negative indexing.



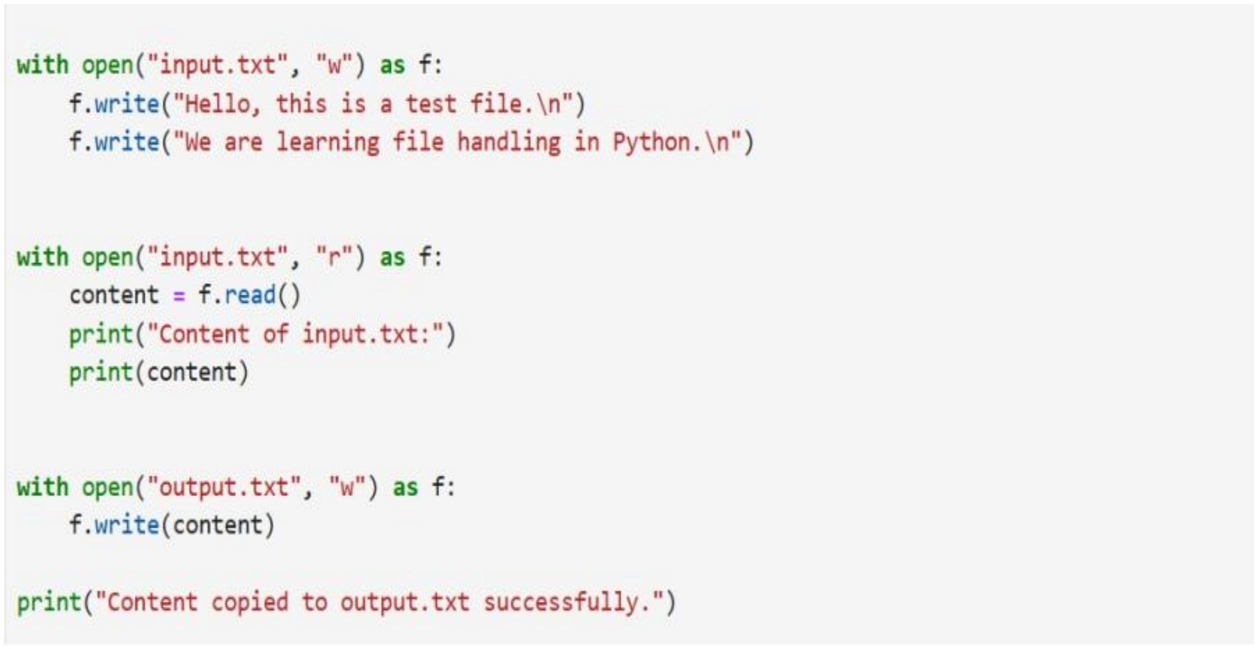
```python
import numpy as np

arr = np.array([10, 20, 30, 40, 50, 60, 70])

print("Original Array:", arr)

print("\nPositive Indexing Slices:")
print("arr[1:4]   ->", arr[1:4])   # elements from index 1 to 3
print("arr[:3]    ->", arr[:3])    # first 3 elements
print("arr[3:]    ->", arr[3:])    # elements from index 3 to end

print("\nNegative Indexing Slices:")
print("arr[-4:-1] ->", arr[-4:-1]) # 4th last to 2nd last
print("arr[-3:]   ->", arr[-3:])   # last 3 elements
print("arr[:-3]   ->", arr[:-3])   # all except last 3
```

Output:



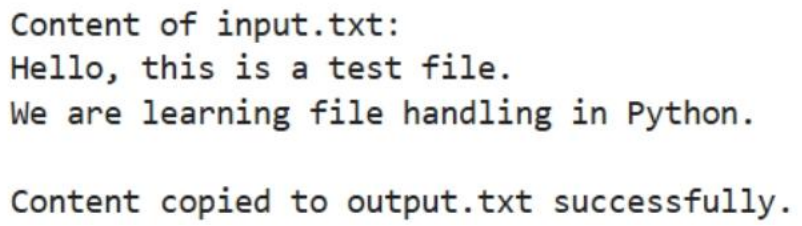
```
Original Array: [10 20 30 40 50 60 70]

Positive Indexing Slices:
arr[1:4]   -> [20 30 40]
arr[:3]    -> [10 20 30]
arr[3:]    -> [40 50 60 70]

Negative Indexing Slices:
arr[-4:-1] -> [40 50 60]
arr[-3:]   -> [50 60 70]
arr[:-3]   -> [10 20 30 40]
```

Program 10 : Read and write into file.



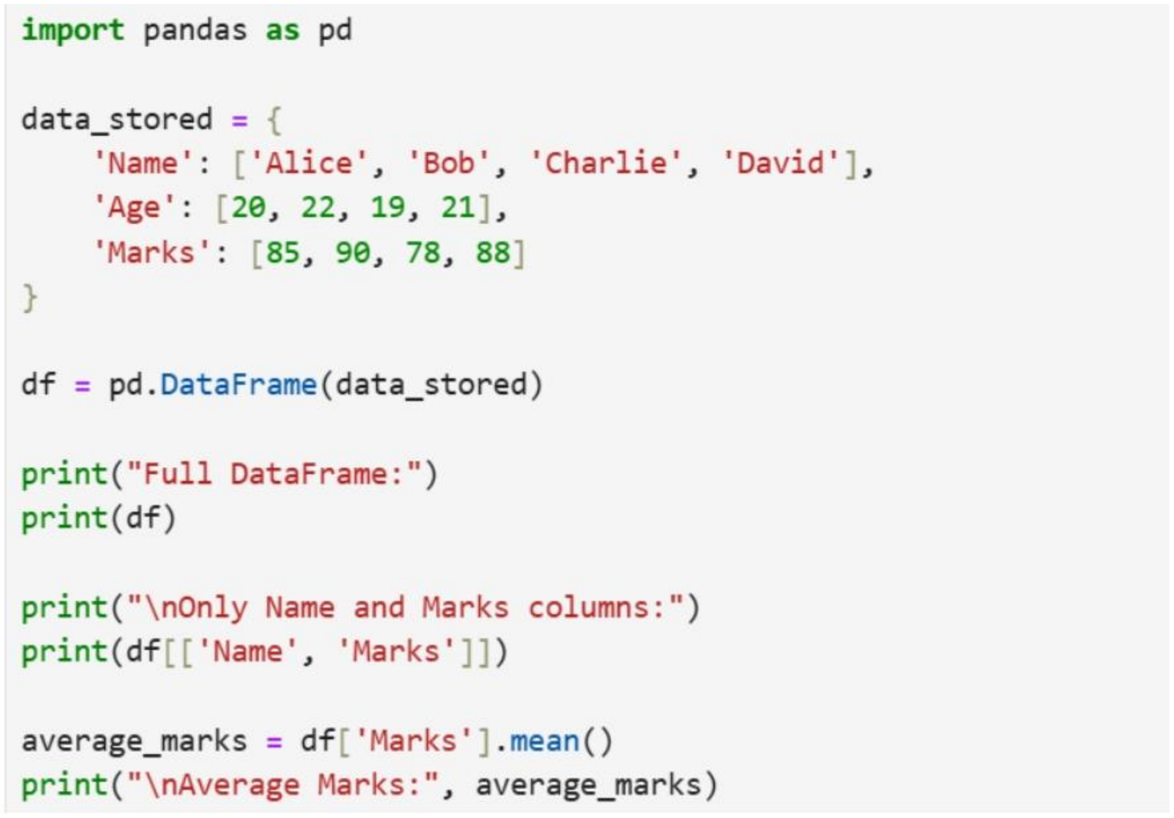
```python
with open("input.txt", "w") as f:
    f.write("Hello, this is a test file.\n")
    f.write("We are learning file handling in Python.\n")


with open("input.txt", "r") as f:
    content = f.read()
    print("Content of input.txt:")
    print(content)


with open("output.txt", "w") as f:
    f.write(content)

print("Content copied to output.txt successfully.")
```

Output:



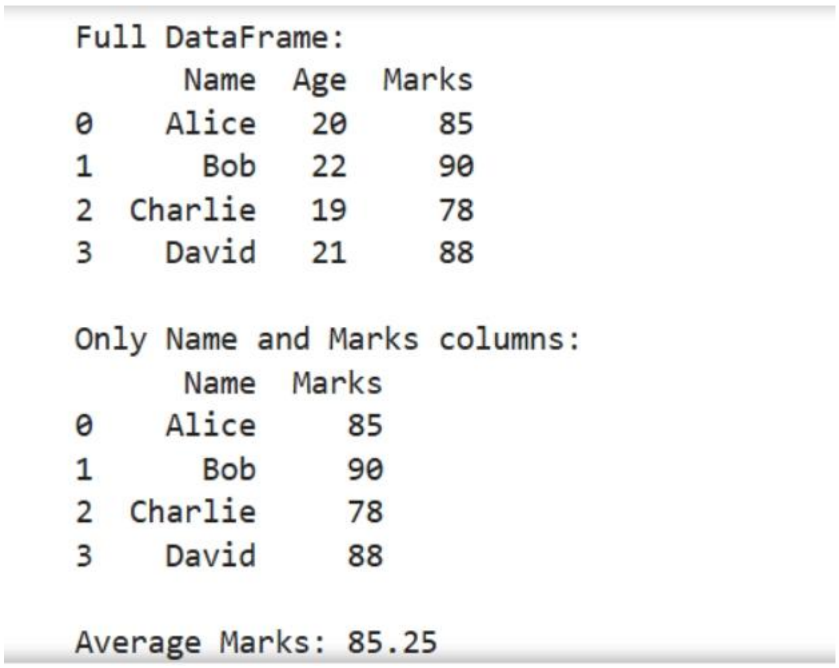
```
Content of input.txt:
Hello, this is a test file.
We are learning file handling in Python.

Content copied to output.txt successfully.
```

Program 11: Create a DataFrame using a Python dictionary and perform basic operations such as displaying specific columns and finding the average of a numeric column.



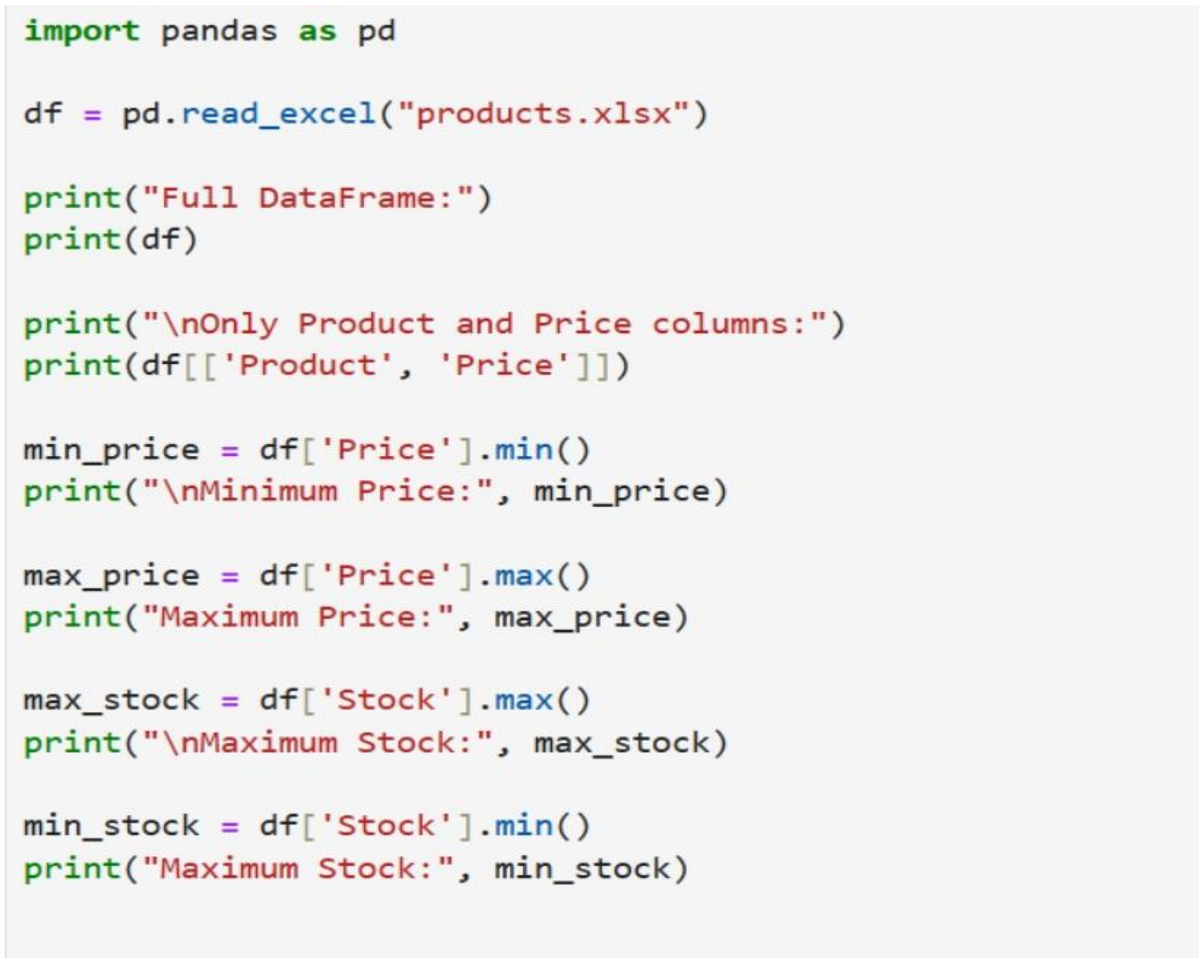
```python
import pandas as pd

data_stored = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [20, 22, 19, 21],
    'Marks': [85, 90, 78, 88]
}

df = pd.DataFrame(data_stored)

print("Full DataFrame:")
print(df)

print("\nOnly Name and Marks columns:")
print(df[['Name', 'Marks']])

average_marks = df['Marks'].mean()
print("\nAverage Marks:", average_marks)
```

Output:



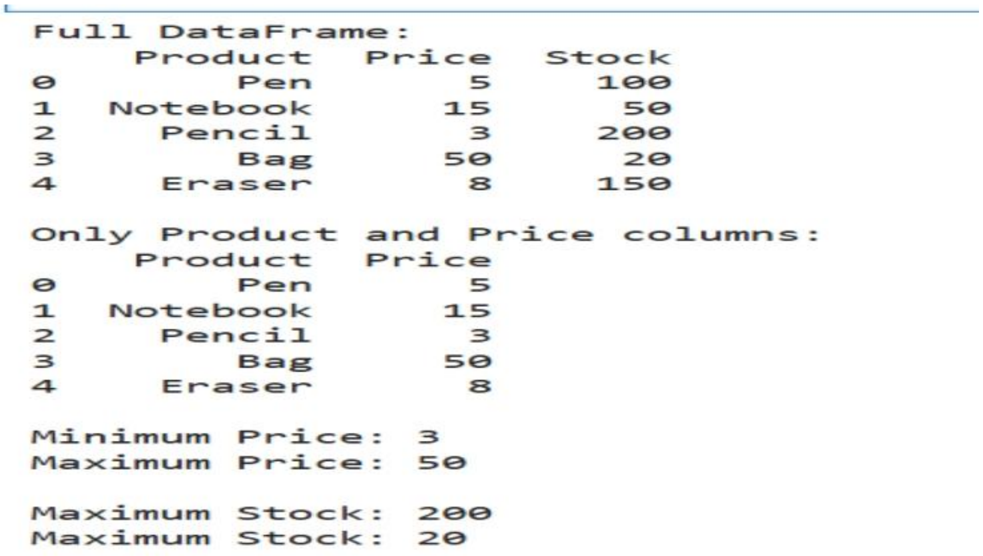
```
Full DataFrame:
      Name  Age  Marks
0    Alice   20     85
1      Bob   22     90
2  Charlie   19     78
3    David   21     88

Only Name and Marks columns:
      Name  Marks
0    Alice     85
1      Bob     90
2  Charlie     78
3    David     88

Average Marks: 85.25
```

Program 12 : Create DataFrames from an Excel sheet using Pandas and perform operations on DataFrames.

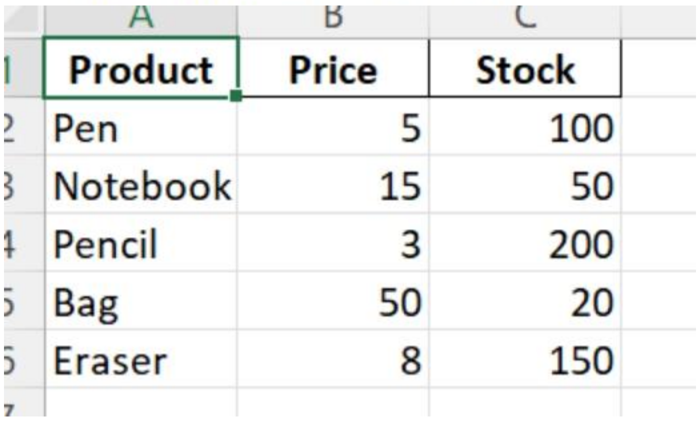


```python
import pandas as pd

df = pd.read_excel("products.xlsx")

print("Full DataFrame:")
print(df)

print("\nOnly Product and Price columns:")
print(df[['Product', 'Price']])

min_price = df['Price'].min()
print("\nMinimum Price:", min_price)

max_price = df['Price'].max()
print("Maximum Price:", max_price)

max_stock = df['Stock'].max()
print("\nMaximum Stock:", max_stock)

min_stock = df['Stock'].min()
print("Maximum Stock:", min_stock)
```

Output:

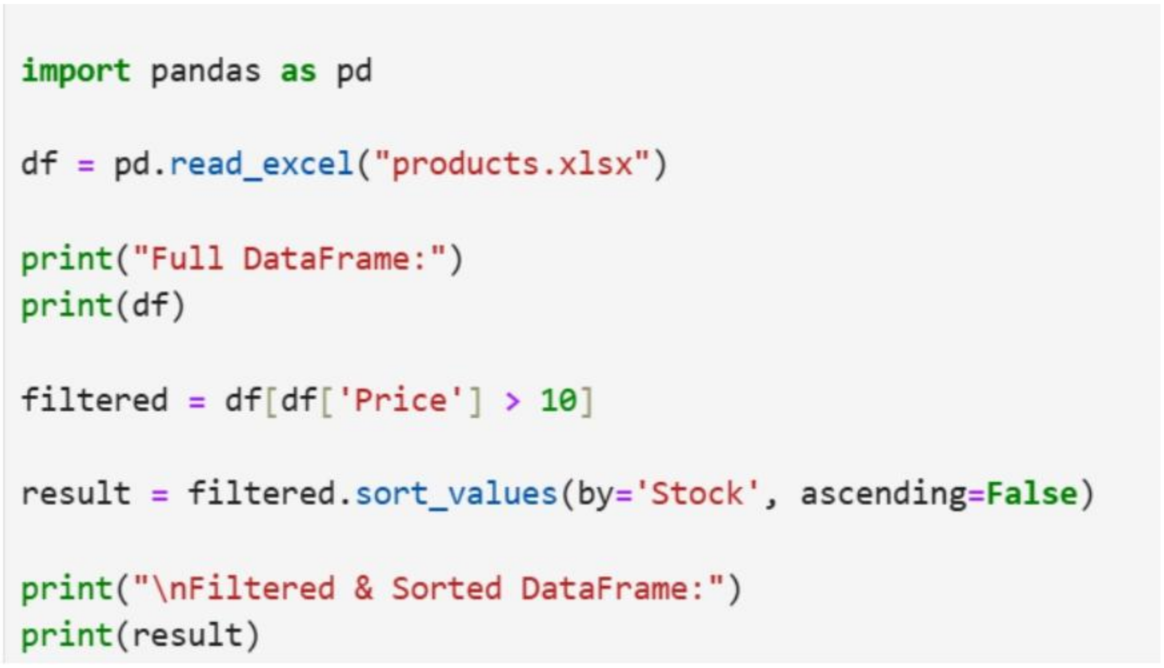


```
Full DataFrame:
    Product  Price  Stock
0       Pen      5    100
1  Notebook     15     50
2    Pencil      3    200
3       Bag     50     20
4    Eraser      8    150

Only Product and Price columns:
    Product  Price
0       Pen      5
1  Notebook     15
2    Pencil      3
3       Bag     50
4    Eraser      8

Minimum Price: 3
Maximum Price: 50

Maximum Stock: 200
Maximum Stock: 20
```



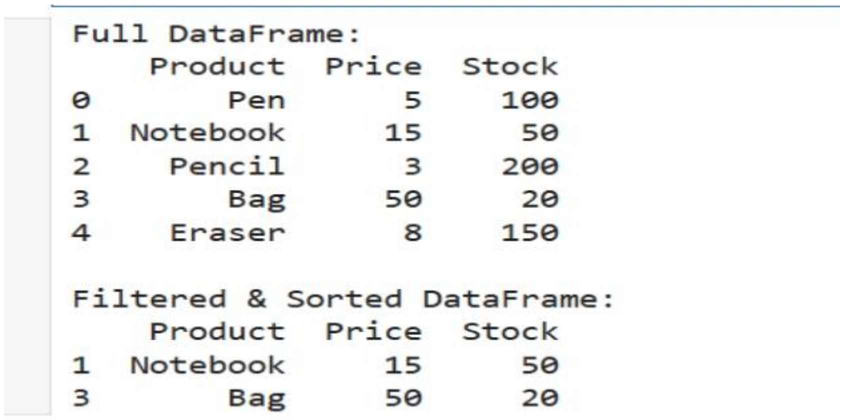
Product	Price	Stock
Pen	5	100
Notebook	15	50
Pencil	3	200
Bag	50	20
Eraser	8	150

Progran 13: Create a DataFrame with product details. Filter the products whose price is greater than 10 and sort the DataFrame by stock in descending order.

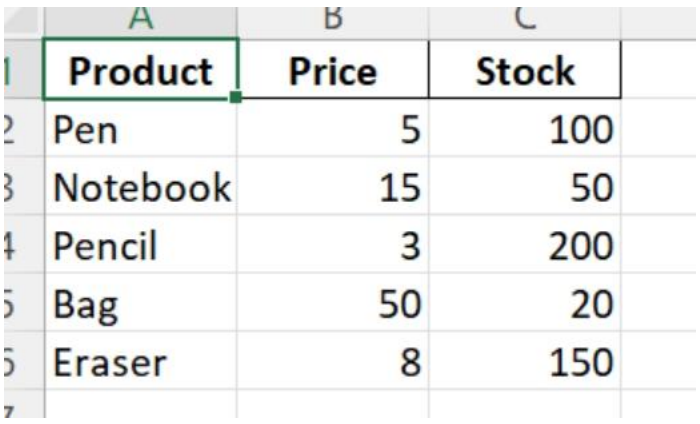


```python
import pandas as pd

df = pd.read_excel("products.xlsx")

print("Full DataFrame:")
print(df)

filtered = df[df['Price'] > 10]

result = filtered.sort_values(by='Stock', ascending=False)

print("\nFiltered & Sorted DataFrame:")
print(result)
```

Output:

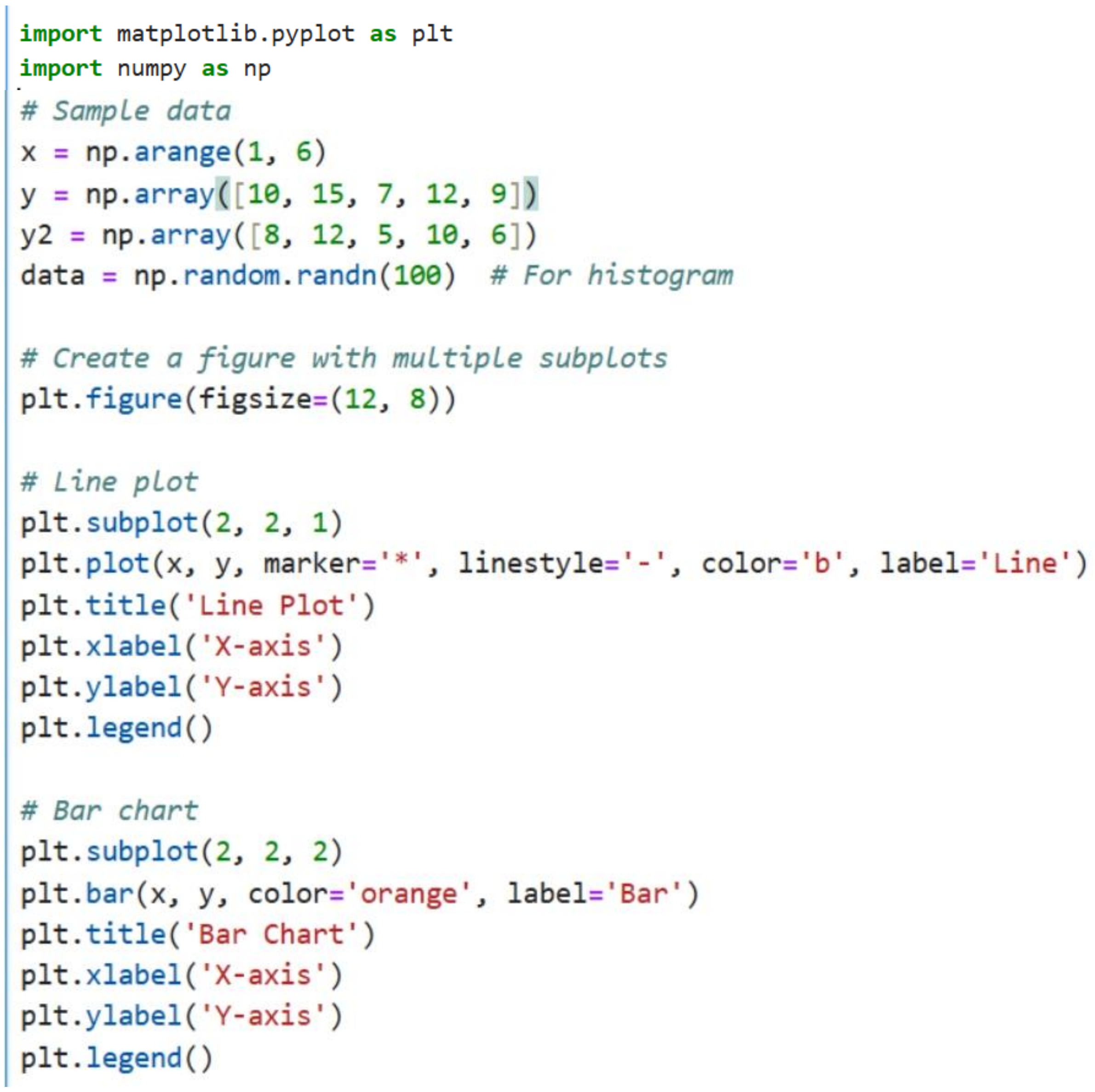


```
Full DataFrame:
    Product  Price  Stock
0       Pen      5    100
1  Notebook     15     50
2    Pencil      3    200
3       Bag     50     20
4    Eraser      8    150

Filtered & Sorted DataFrame:
    Product  Price  Stock
1  Notebook     15     50
3       Bag     50     20
```



Product	Price	Stock
Pen	5	100
Notebook	15	50
Pencil	3	200
Bag	50	20
Eraser	8	150

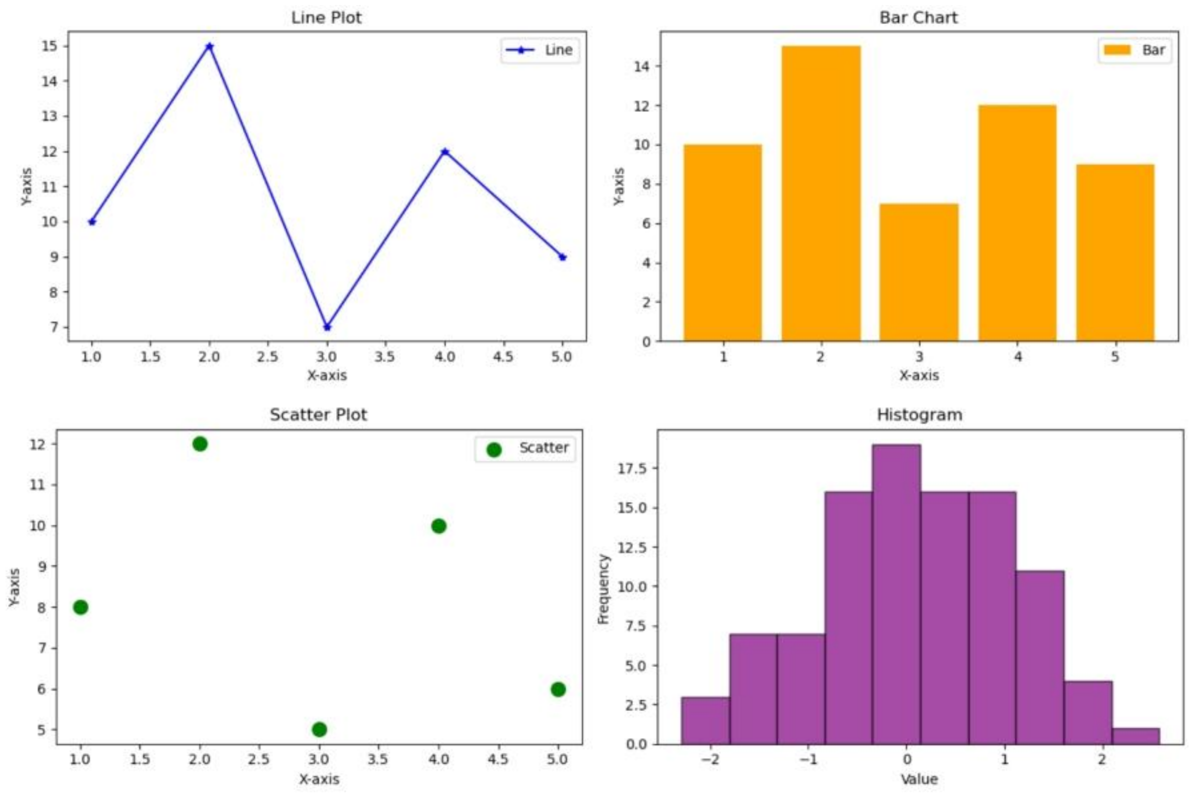
Program 14 : Matplotlib Visualization Suite



```python
import matplotlib.pyplot as plt
import numpy as np
# Sample data
x = np.arange(1, 6)
y = np.array([10, 15, 7, 12, 9])
y2 = np.array([8, 12, 5, 10, 6])
data = np.random.randn(100)  # For histogram

# Create a figure with multiple subplots
plt.figure(figsize=(12, 8))

# Line plot
plt.subplot(2, 2, 1)
plt.plot(x, y, marker='*', linestyle='-', color='b', label='Line')
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()

# Bar chart
plt.subplot(2, 2, 2)
plt.bar(x, y, color='orange', label='Bar')
plt.title('Bar Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
```

```python
# Scatter plot
plt.subplot(2, 2, 3)
plt.scatter(x, y2, color='green', label='Scatter', s=100)
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()

# Histogram
plt.subplot(2, 2, 4)
plt.hist(data, bins=10, color='purple', edgecolor='black', alpha=0.7)
plt.title('Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Adjust layout and show
plt.tight_layout()
plt.show()
```

Output:



PART B

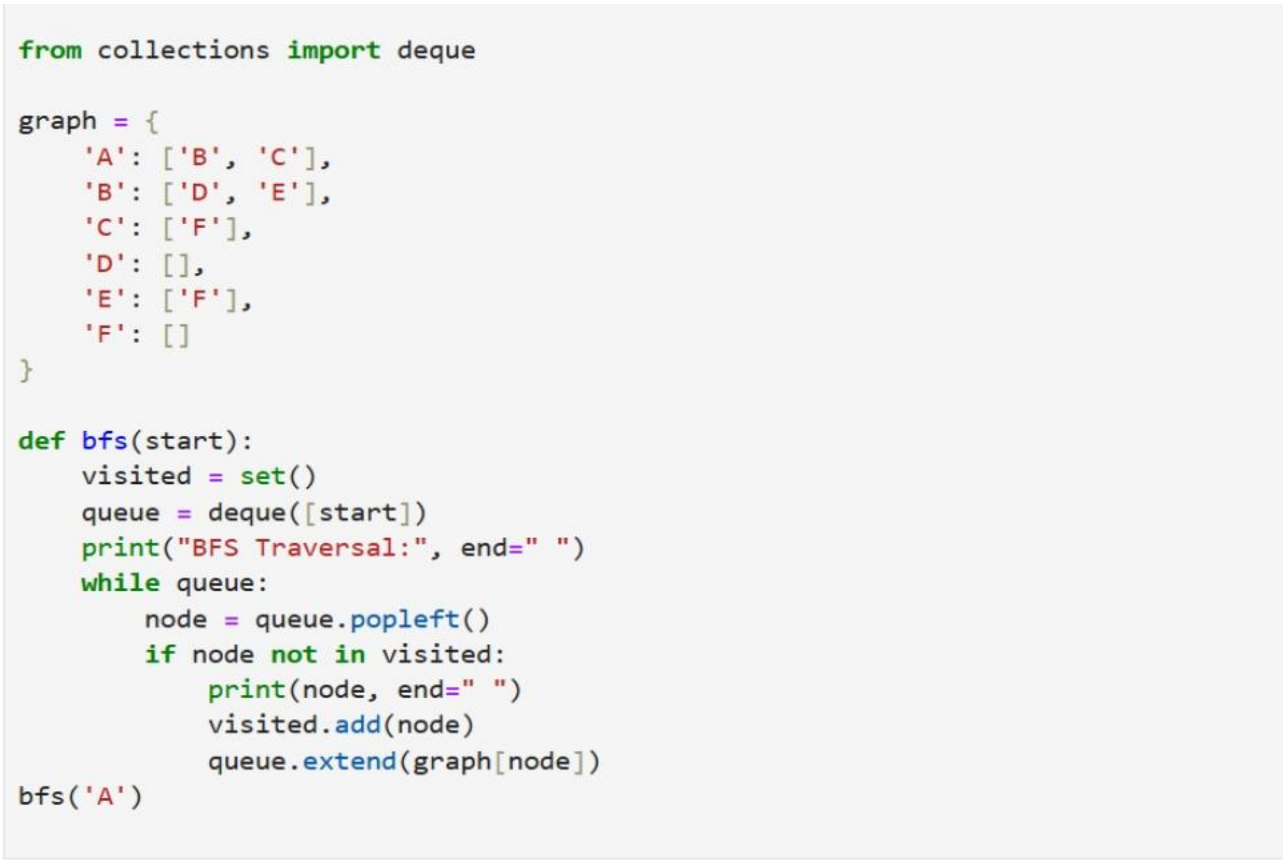
Program 1. Implement Depth-First Search (DFS) on a small graph.

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

visited = set()

def dfs(node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(neighbour)
dfs('A')
```

Output:

```
A B D E F C
```

Program 2. Implement Breadth-First Search (BFS) on a small graph.

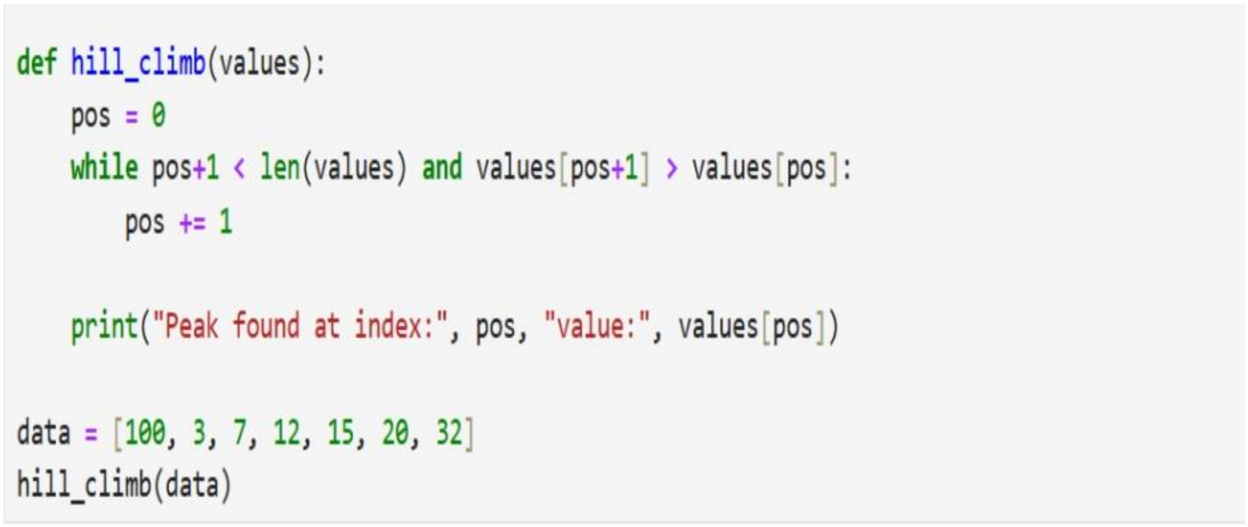


```python
from collections import deque

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

def bfs(start):
    visited = set()
    queue = deque([start])
    print("BFS Traversal:", end=" ")
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend(graph[node])
bfs('A')
```

Output:

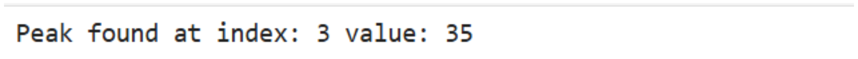
```
BFS Traversal: A B C D E F
```

Program 3. Implement a Hill Climbing search to find the peak in a numeric dataset.



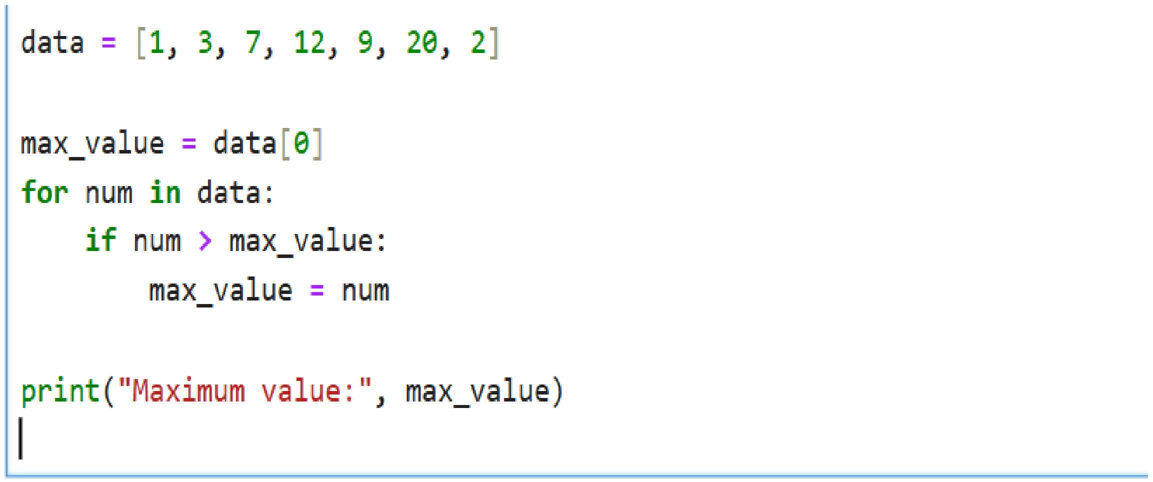
```python
def hill_climb(values):
    pos = 0
    while pos+1 < len(values) and values[pos+1] > values[pos]:
        pos += 1

    print("Peak found at index:", pos, "value:", values[pos])

data = [100, 3, 7, 12, 15, 20, 32]
hill_climb(data)
```

Output:



```
Peak found at index: 3 value: 35
```

Program 4. Apply optimization technique to find the maximum value in a list.

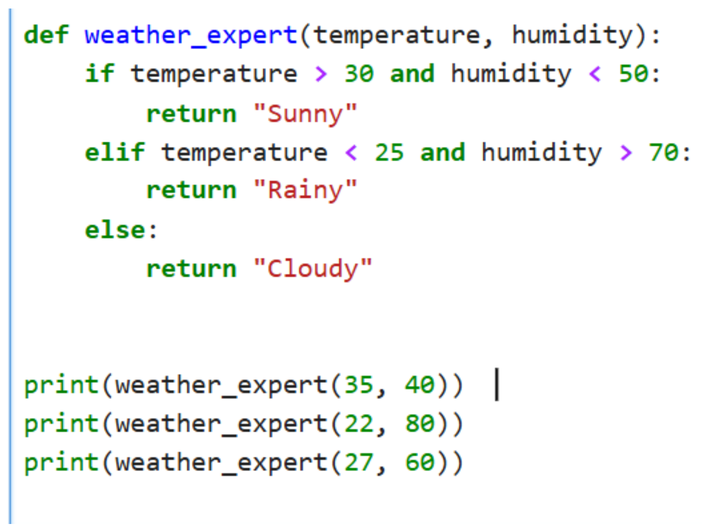


```python
data = [1, 3, 7, 12, 9, 20, 2]

max_value = data[0]
for num in data:
    if num > max_value:
        max_value = num

print("Maximum value:", max_value)
```

Output:

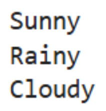
```
Maximum value: 20
```

Program 5. Implement a basic rule-based expert system for weather classification.



```python
def weather_expert(temperature, humidity):
    if temperature > 30 and humidity < 50:
        return "Sunny"
    elif temperature < 25 and humidity > 70:
        return "Rainy"
    else:
        return "Cloudy"


print(weather_expert(35, 40))
print(weather_expert(22, 80))
print(weather_expert(27, 60))
```

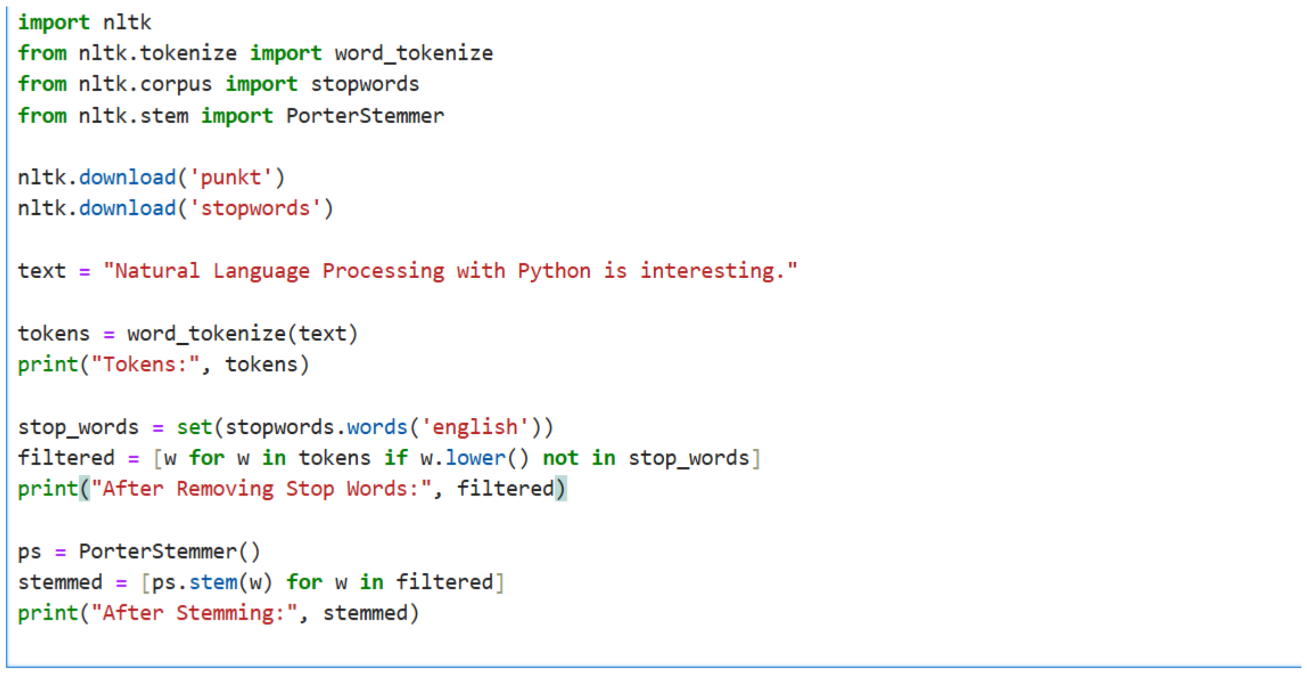
Output:



```
Sunny
Rainy
Cloudy
```

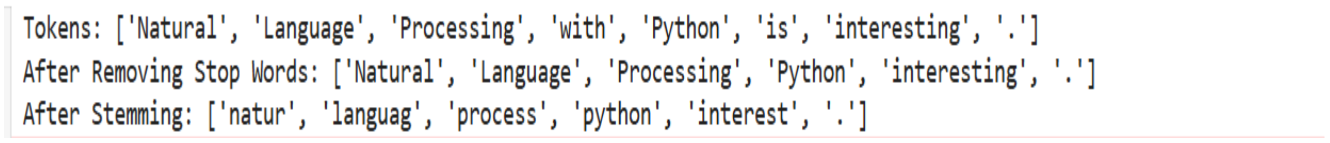
Program 6. Using Python NLTK, perform the following Natural Language Processing (NLP) tasks for text content:

a) Tokenizing
b) Filtering Stop Words
c) Stemming



```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

nltk.download('punkt')
nltk.download('stopwords')

text = "Natural Language Processing with Python is interesting."

tokens = word_tokenize(text)
print("Tokens:", tokens)

stop_words = set(stopwords.words('english'))
filtered = [w for w in tokens if w.lower() not in stop_words]
print("After Removing Stop Words:", filtered)

ps = PorterStemmer()
stemmed = [ps.stem(w) for w in filtered]
print("After Stemming:", stemmed)
```

Output:



```
Tokens: ['Natural', 'Language', 'Processing', 'with', 'Python', 'is', 'interesting', '.']
After Removing Stop Words: ['Natural', 'Language', 'Processing', 'Python', 'interesting', '.']
After Stemming: ['natur', 'languag', 'process', 'python', 'interest', '.']
```