**PRACTICAL-1**

**AIM:**

**a. Write a program to demonstrate all the basic data types in python.**

**Source Code:**

int\_var = 10

float\_var = 3.14

str\_var = "Hello, World!"

bool\_var = True

list\_var = [1, 2, 3]

tuple\_var = (4, 5, 6)

dict\_var = {"name": "Om", "age": 20}

set\_var = {7, 8, 9}

print(int\_var, "typeof(int\_var):", type(int\_var))

print(float\_var, "typeof(float\_var):", type(float\_var))

print(str\_var, "typeof(str\_var):", type(str\_var))

print(bool\_var, "typeof(bool\_var):", type(bool\_var))

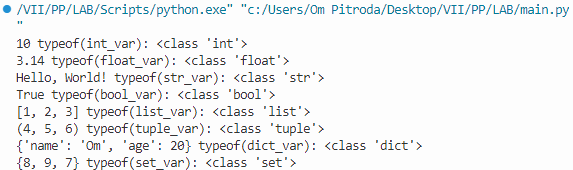
print(list\_var, "typeof(list\_var):", type(list\_var))

print(tuple\_var, "typeof(tuple\_var):", type(tuple\_var))

print(dict\_var, "typeof(dict\_var):", type(dict\_var))

print(set\_var, "typeof(set\_var):", type(set\_var))

**Output:**

****

**b. Write a program that takes two numbers as command line arguments and prints its summation.**

**Source Code:**

import sys

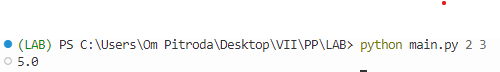
num1 = float(sys.argv[1])

num2 = float(sys.argv[2])

result = num1 + num2

print(result)

**Output:**



**c. Write a program to print the largest and smallest number of three numbers input from user with and without using library functions.**

**Source Code:**

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

num3 = float(input("Enter the third number: "))

if num1 >= num2 and num1 >= num3:

max\_num = num1

elif num2 >= num1 and num2 >= num3:

max\_num = num2

else:

max\_num = num3

if num1 <= num2 and num1 <= num3:

min\_num = num1

elif num2 <= num1 and num2 <= num3:

min\_num = num2

else:

min\_num = num3

print(f"Largest number: {max\_num}")

print(f"Smallest number: {min\_num}") num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

num3 = float(input("Enter the third number: "))

if num1 >= num2 and num1 >= num3:

max\_num = num1

elif num2 >= num1 and num2 >= num3:

max\_num = num2

else:

max\_num = num3

if num1 <= num2 and num1 <= num3:

min\_num = num1

elif num2 <= num1 and num2 <= num3:

min\_num = num2

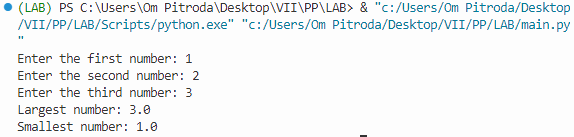
else:

min\_num = num3

print(f"Largest number: {max\_num}")

print(f"Smallest number: {min\_num}")

**Output:**

****

**d. Write a program to calculate GCD of two numbers.**

**Source Code:**

def find\_gcd(x, y):

while(y):

x, y = y, x % y

return x

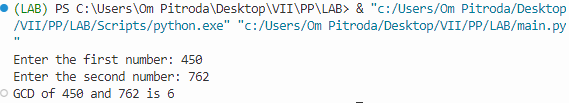
num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

gcd = find\_gcd(num1, num2)

print(f"GCD of {num1} and {num2} is {gcd}")

**Output:**

****

**PRACTICAL-2**

**AIM:**

**a.** Write a program to calculate the square root of a number by Newton's Method.

**Source Code:**

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

guess = number / 2 # Initial guess

while True:

new\_guess = (guess + number / guess) / 2

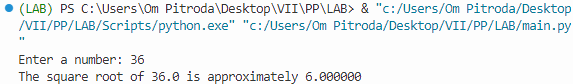
if abs(new\_guess - guess) < 1e-6: # Tolerance for convergence

break

guess = new\_guess

print(f"The square root of {number} is approximately {new\_guess:.6f}")

**Output:**



**b.** Write a program for checking whether the given number is an even number or not.

**Source Code:**

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

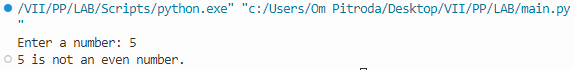
if number % 2 == 0:

print(f"{number} is an even number.")

else:

print(f"{number} is not an even number.")

**Output:**



**c.** Write a program using a while loop that asks the user for a number, and prints a countdown from that number to zero.

**Source Code:**

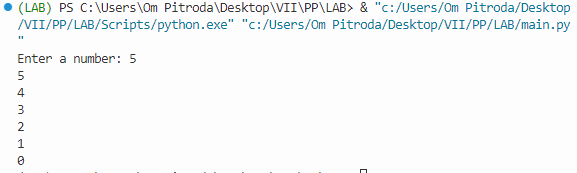
number = int(input("Enter a number: "))

while number >= 0:

print(number)

number -= 1

**Output:**



**d.** Write a program that uses for loop to print all the odd numbers in the range input by user

**Source Code:**

e start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

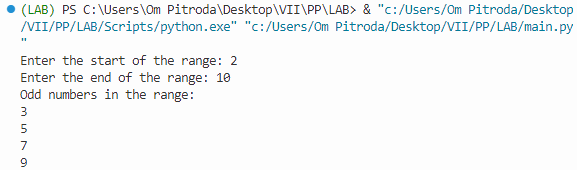
print("Odd numbers in the range:")

for num in range(start, end + 1):

if num % 2 != 0:

print(num)

**Output:**

****

**PRACTICAL-3**

**AIM:**

**a.** Write a program to check whether the given string is palindrome or not.

**Source Code:**

def is\_palindrome(input\_str):

input\_str = input\_str.lower() # Convert to lowercase to make it case-insensitive

input\_str = input\_str.replace(" ", "") # Remove spaces

return input\_str == input\_str[::-1]

user\_input = input("Enter a string: ")

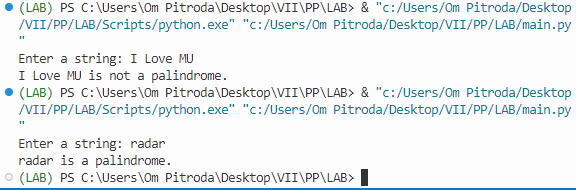
if is\_palindrome(user\_input):

print(f"{user\_input} is a palindrome.")

else:

print(f"{user\_input} is not a palindrome.")

**Output:**



**b.** Write a program that accepts a string from user and performs the following operations:

i) Print the string in reverse order

ii) Print all the odd indexed characters of the string

iii) Print the count of all the vowels in the string

iv) Print the count of the frequency of an input character in the string

**Source Code:**

user\_input = input("Enter a string: ")

reverse\_string = user\_input[::-1]

print(f"Reversed string: {reverse\_string}")

odd\_characters = user\_input[1::2]

print(f"Odd indexed characters: {odd\_characters}")

vowels = "aeiouAEIOU"

vowel\_count = sum(1 for char in user\_input if char in vowels)

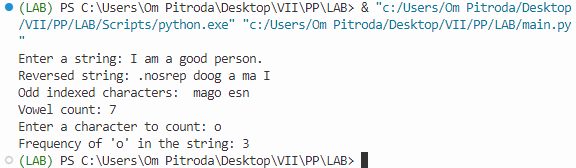
print(f"Vowel count: {vowel\_count}")

char\_to\_count = input("Enter a character to count: ")

char\_count = user\_input.count(char\_to\_count)

print(f"Frequency of '{char\_to\_count}' in the string: {char\_count}")

**Output:**



**PRACTICAL-4**

**AIM:**

**a.** Write a program to create an empty list. Demonstrate the use of the append function to add elements onto the list.

**Source Code:**

my\_list = []

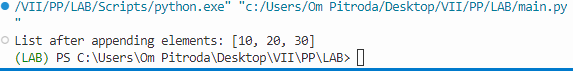
my\_list.append(10)

my\_list.append(20)

my\_list.append(30)

print("List after appending elements:", my\_list)

**Output:**



**b.** Demonstrate the use of the following functions of List Data Structure:

i) Operations on List: copy(), count(), extend(), index(), reverse(), sort()

ii) Manipulating List: append(), insert(), pop(), remove(), clear()

**Source Code:**

sample\_list = [10, 20, 30, 20, 40, 20]

copy\_of\_list = sample\_list.copy()

count\_of\_20 = sample\_list.count(20)

extension\_list = [50, 60]

sample\_list.extend(extension\_list)

index\_of\_30 = sample\_list.index(30)

sample\_list.reverse()

sample\_list.sort()

print("Copy of List:", copy\_of\_list)

print("Count of 20 in List:", count\_of\_20)

print("List after extending with [50, 60]:", sample\_list)

print("Index of 30 in List:", index\_of\_30)

print("List after reversing:", sample\_list)

print("List after sorting:", sample\_list)

sample\_list.append(70)

sample\_list.insert(2, 25)

popped\_element = sample\_list.pop()

sample\_list.remove(20)

sample\_list.clear()

print("List after appending 70:", sample\_list)

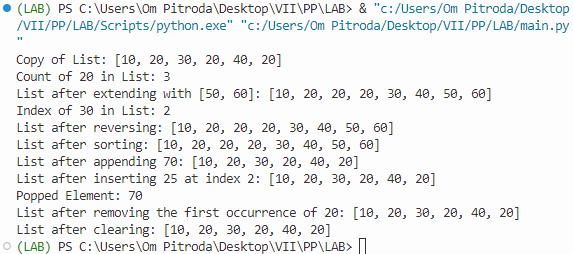
print("List after inserting 25 at index 2:", sample\_list)

print("Popped Element:", popped\_element)

print("List after removing the first occurrence of 20:", sample\_list)

print("List after clearing:", sample\_list)

**Output:**



**PRACTICAL-5**

**AIM:**

**a.** Write a program to create an empty set. Input the elements from user and write a for loop to add these elements onto the set.

**Source Code:**

my\_set = set()

num\_elements = int(input("Enter the number of elements to add to the set: "))

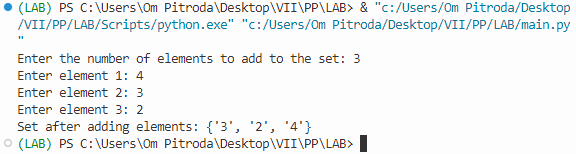
for i in range(num\_elements):

element = input(f"Enter element {i + 1}: ")

my\_set.add(element)

print("Set after adding elements:", my\_set)

**Output:**



**b.** Demonstrate the use of the following functions of Set Data Structure:

1. Operations on Set: difference(), difference\_update(), intersection(), intersection\_update(), symmetric\_difference(), symmetric\_difference\_update(), isdisjoint(), issuperset(), issubset()
2. Manipulating Set: discard(), add(), clear(), copy(), pop(), remove()

**­Source Code:**

set1 = {1, 2, 3, 4, 5}

set2 = {3, 4, 5, 6, 7}

diff\_set = set1.difference(set2)

set1.difference\_update(set2)

inter\_set = set1.intersection(set2)

set1.intersection\_update(set2)

sym\_diff\_set = set1.symmetric\_difference(set2)

set1.symmetric\_difference\_update(set2)

disjoint = set1.isdisjoint(set2)

superset = set1.issuperset(set2)

subset = set1.issubset(set2)

print("Difference of set1 and set2:", diff\_set)

print("Set1 after difference update:", set1)

print("Intersection of set1 and set2:", inter\_set)

print("Set1 after intersection update:", set1)

print("Symmetric difference of set1 and set2:", sym\_diff\_set)

print("Set1 after symmetric difference update:", set1)

print("Are set1 and set2 disjoint?", disjoint)

print("Is set1 a superset of set2?", superset)

print("Is set1 a subset of set2?", subset)

set1.discard(3)

set1.add(8)

set1.clear()

copy\_of\_set = set2.copy()

popped\_element = set2.pop()

set2.remove(5)

print("Set1 after discarding 3:", set1)

print("Set1 after adding 8:", set1)

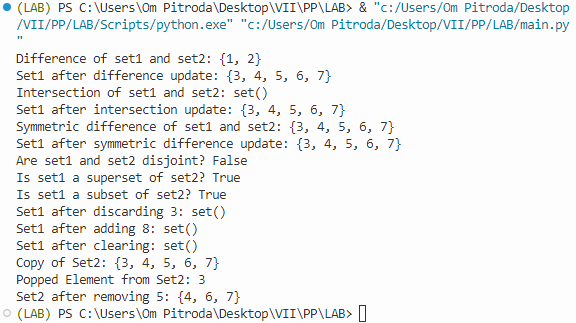
print("Set1 after clearing:", set1)

print("Copy of Set2:", copy\_of\_set)

print("Popped Element from Set2:", popped\_element)

print("Set2 after removing 5:", set2)

**Output:**



**PRACTICAL-6**

**AIM:**

**a.** Write a program to demonstrate the use of the following methods in a Tuple: i) count, ii) index

**Source Code:**

my\_tuple = (1, 2, 3, 4, 2, 5, 2)

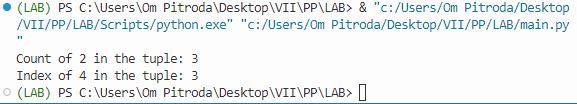
count\_of\_2 = my\_tuple.count(2)

index\_of\_4 = my\_tuple.index(4)

print("Count of 2 in the tuple:", count\_of\_2)

print("Index of 4 in the tuple:", index\_of\_4)

**Output:**



**b.** Create an empty dictionary and write a program to add single and multiple elements onto the dictionary.

**Source Code:**

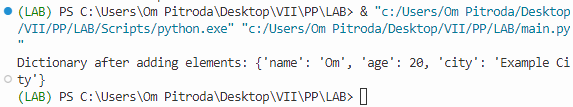
my\_dict = {}

my\_dict['name'] = 'Om'

my\_dict.update({'age': 20, 'city': 'Example City'})

print("Dictionary after adding elements:", my\_dict)

**Output:**

****

**c.** Write a program demonstrating the use of following functions of the Dictionary Data Structure:

i) Operations on Dictionary: copy(), fromkeys(), get(), items(), keys(), values()

ii) Manipulating Dictionary: update(), pop(), popitem(), clear()

**Source Code:**

e sample\_dict = {'name': 'John', 'age': 25, 'city': 'Sample City'}

copy\_dict = sample\_dict.copy()

keys\_list = ['name', 'age', 'city']

default\_value = 'Not found'

new\_dict = dict.fromkeys(keys\_list, default\_value)

age = sample\_dict.get('age', 'N/A')

gender = sample\_dict.get('gender', 'N/A')

dict\_items = sample\_dict.items()

dict\_keys = sample\_dict.keys()

dict\_values = sample\_dict.values()

print("Copy of Dictionary:", copy\_dict)

print("New Dictionary with default values:", new\_dict)

print("Age from Dictionary:", age)

print("Gender from Dictionary:", gender)

print("Items in Dictionary:", dict\_items)

print("Keys in Dictionary:", dict\_keys)

print("Values in Dictionary:", dict\_values)

sample\_dict.update({'city': 'New City', 'gender': 'Male'})

age = sample\_dict.pop('age')

item = sample\_dict.popitem()

sample\_dict.clear()

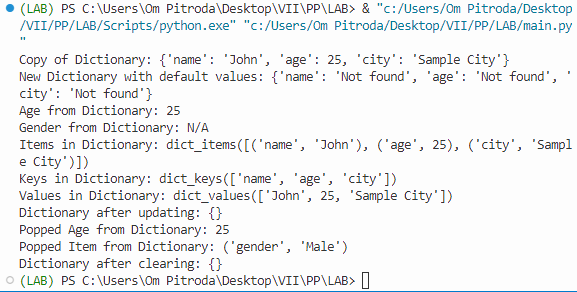
print("Dictionary after updating:", sample\_dict)

print("Popped Age from Dictionary:", age)

print("Popped Item from Dictionary:", item)

print("Dictionary after clearing:", sample\_dict)

**Output:**



**PRACTICAL-7**

**AIM:**

**a.** Create a python function to find all the unique elements in the list.

**Source Code:**

def find\_unique\_elements(input\_list):

unique\_elements = list(set(input\_list))

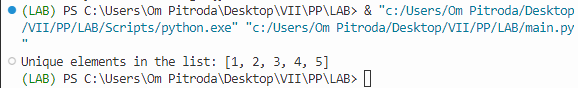
return unique\_elements

my\_list = [1, 2, 2, 3, 4, 4, 5]

unique\_elements = find\_unique\_elements(my\_list)

print("Unique elements in the list:", unique\_elements)

**Output:**

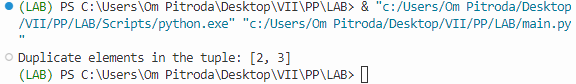


**b.** Create a python function to find all the duplicate elements in a tuple.

**Source Code:**

print("Duplicate elements in the tuple:", duplicate\_elements)

**Output:**



**c.** Write a program to create a function in python that compares two dictionaries and returns true or false accordingly.

**Source Code:**

def compare\_dictionaries(dict1, dict2):

return dict1 == dict2

dict1 = {'name': 'John', 'age': 30}

dict2 = {'name': 'John', 'age': 30}

dict3 = {'name': 'Alice', 'age': 25}

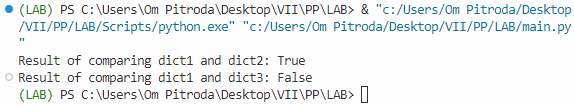
result1 = compare\_dictionaries(dict1, dict2)

result2 = compare\_dictionaries(dict1, dict3)

print("Result of comparing dict1 and dict2:", result1)

print("Result of comparing dict1 and dict3:", result2)

**Output:**



**PRACTICAL-7**

**AIM:**

**a.** Write a program to demonstrate recursion in Python.

**Source Code:**

def factorial(n):

if n == 0:

return 1

else:

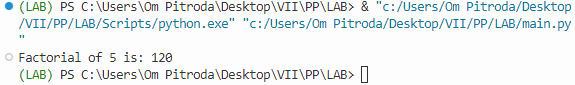
return n \* factorial(n - 1)

# Example usage:

result = factorial(5)

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

**Output:**



**b.** Create a function for Stack data structure in Python and implement necessary operations.

**Source Code:**

class Stack:

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

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

if not self.is\_empty():

return self.items.pop()

def is\_empty(self):

return len(self.items) == 0

def peek(self):

if not self.is\_empty():

return self.items[-1]

def size(self):

return len(self.items)

my\_stack = Stack()

my\_stack.push(1)

my\_stack.push(2)

my\_stack.push(3)

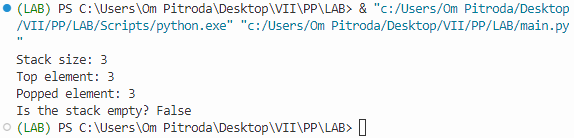
print("Stack size:", my\_stack.size())

print("Top element:", my\_stack.peek())

print("Popped element:", my\_stack.pop())

print("Is the stack empty?", my\_stack.is\_empty())

**Output:**



**c.** Create a function for Queue data structure in Python and implement necessary operations

**Source Code:**

from collections import deque

class Queue:

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

self.items = deque()

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.is\_empty():

return self.items.popleft()

def is\_empty(self):

return len(self.items) == 0

def size(self):

return len(self.items)

# Example usage:

my\_queue = Queue()

my\_queue.enqueue(1)

my\_queue.enqueue(2)

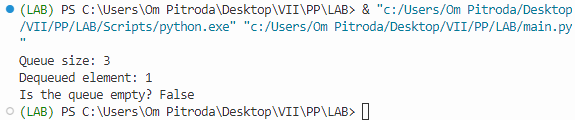
my\_queue.enqueue(3)

print("Queue size:", my\_queue.size())

print("Dequeued element:", my\_queue.dequeue())

print("Is the queue empty?", my\_queue.is\_empty())

**Output:**



**PRACTICAL-9**

**AIM:** Implement a Python Program to perform the following Sorting and Searching algorithms:

i) Linear search

ii) Binary search

iii) Selection sort

iv) Insertion sort

**Source Code:**

edef linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1

def selection\_sort(arr):

for i in range(len(arr)):

min\_idx = i

for j in range(i+1, len(arr)):

if arr[j] < arr[min\_idx]:

min\_idx = j

arr[i], arr[min\_idx] = arr[min\_idx], arr[i]

def insertion\_sort(arr):

for i in range(1, len(arr)):

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

arr = [64, 34, 25, 12, 22, 11, 90]

target = 25

linear\_result = linear\_search(arr, target)

print(f"Linear Search: {target} found at index {linear\_result}" if linear\_result != -1 else f"{target} not found")

arr.sort()

binary\_result = binary\_search(arr, target)

print(f"Binary Search: {target} found at index {binary\_result}" if binary\_result != -1 else f"{target} not found")

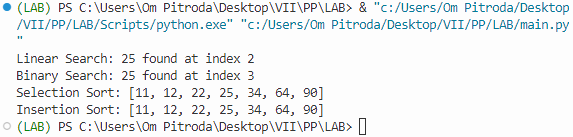
selection\_sort(arr.copy())

print("Selection Sort:", arr)

insertion\_sort(arr.copy())

print("Insertion Sort:", arr)

**Output:**



**PRACTICAL-10**

**AIM:**

**a.** Write a program to search content using regular expression library in python.

**Source Code:**

import re

text = "Hello, my email is example123@gmail.com, and my phone number is 123-456-7890."

email\_pattern = r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b'

phone\_pattern = r'\b\d{3}-\d{3}-\d{4}\b'

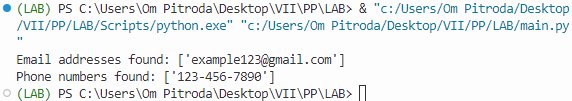
email\_matches = re.findall(email\_pattern, text)

phone\_matches = re.findall(phone\_pattern, text)

print("Email addresses found:", email\_matches)

print("Phone numbers found:", phone\_matches)

**Output:**



**b.** Write a program to implement all the functionalities of Numpy library in Python

**Source Code:**

import numpy as np

arr\_1d = np.array([1, 2, 3])

arr\_2d = np.array([[1, 2, 3], [4, 5, 6]])

arr\_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

arr\_arange = np.arange(0, 10, 2)

arr\_zeros = np.zeros((2, 3))

arr\_ones = np.ones((3, 2))

arr\_linspace = np.linspace(0, 1, 5)

arr\_eye = np.eye(3)

print(f"Dimensions of arr\_1d: {arr\_1d.ndim}, Shape: {arr\_1d.shape}, Size: {arr\_1d.size}, Data type: {arr\_1d.dtype}, Item size: {arr\_1d.itemsize}")

arr\_reshaped = arr\_2d.reshape(3, 2)

print("Reshaped 2D array:")

print(arr\_reshaped)

arr\_flattened = arr\_2d.flatten()

print("Flattened 2D array:")

print(arr\_flattened)

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr\_concatenated = np.concatenate((arr1, arr2))

print("Concatenated arrays:")

print(arr\_concatenated)

arr\_shuffled = arr\_3d.transpose(1, 0, 2)

print("Shuffled 3D array:")

print(arr\_shuffled)

arr\_split = np.split(arr\_1d, 3)

print("Split 1D array:")

for subarray in arr\_split:

print(subarray)

arr\_append = np.append(arr1, [7, 8])

arr\_insert = np.insert(arr1, 1, 9)

arr\_delete = np.delete(arr1, 1)

print("Array after append:")

print(arr\_append)

print("Array after insert:")

print(arr\_insert)

print("Array after delete:")

print(arr\_delete)

print(f"Element at index 2 in arr\_1d: {arr\_1d[2]}")

print(f"Element in row 1, column 2 in arr\_2d: {arr\_2d[1, 2]}")

print(f"Element in array 0, row 1, column 2 in arr\_3d: {arr\_3d[0, 1, 2]}")

print("1-D Array Iteration:")

for item in arr\_1d:

print(item)

print("2-D Array Iteration:")

for row in arr\_2d:

for item in row:

print(item)

print("3-D Array Iteration:")

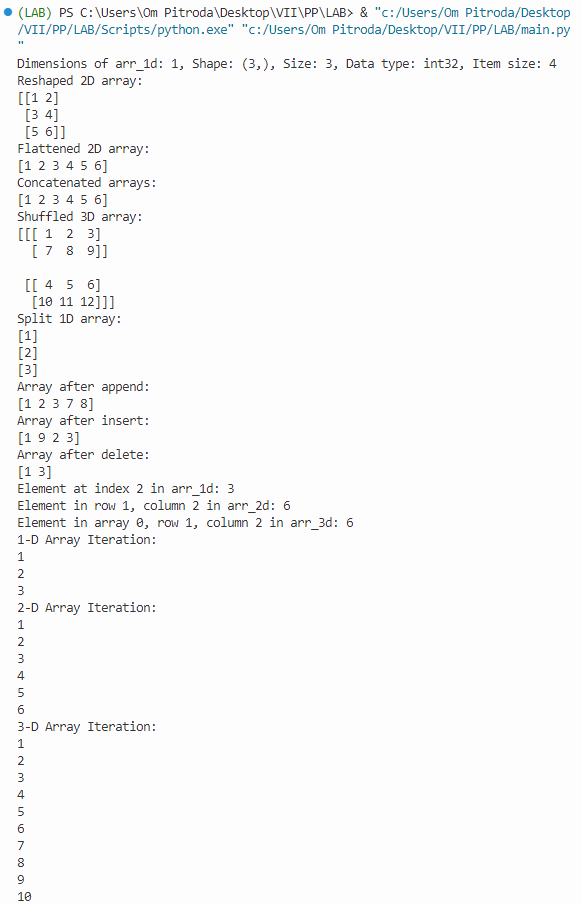
for array in arr\_3d:

for row in array:

for item in row:

print(item)

**Output:**



**PRACTICAL-11**

**AIM:** Write a program to implement all the functionalities of the Pandas library in Python

**Source Code:**

import pandas as pd

list\_data = [10, 20, 30, 40]

list\_series = pd.Series(list\_data)

print("Series from list:")

print(list\_series)

print()

dict\_data = {'A': 100, 'B': 200, 'C': 300, 'D': 400}

dict\_series = pd.Series(dict\_data)

print("Series from dictionary:")

print(dict\_series)

print()

import numpy as np

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

array\_series = pd.Series(array\_data)

print("Series from array:")

print(array\_series)

print()

csv\_series = pd.read\_csv('C:/Users/ompit/Desktop/VII/PP/Labs/sample.csv', header=None, names=['Value'])

print("Series from CSV file:")

print(csv\_series)

print()

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 28, 22],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}

df = pd.DataFrame(data)

print("DataFrame from dictionary:")

print(df)

print()

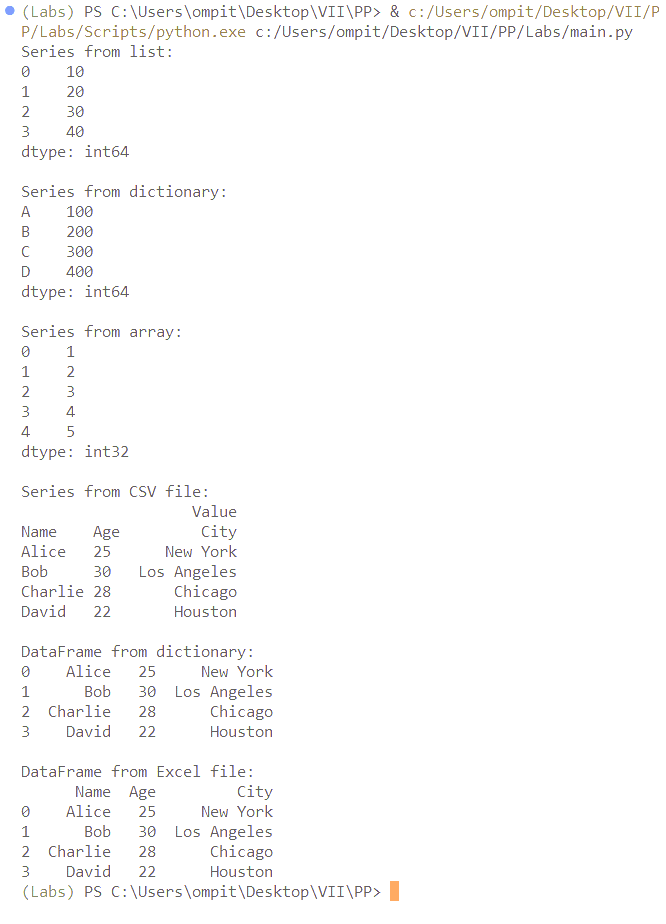
df.to\_excel('people.xlsx', index=False)

excel\_df = pd.read\_excel('people.xlsx')

print("DataFrame from Excel file:")

print(excel\_df)

**Output:**



**PRACTICAL-12**

**AIM:** Write a program to implement all the functionalities of the MatPlotLib, NetworkX library in Python.

**Source Code:**

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

G = nx.Graph()

def display\_graph(G):

pos = nx.spring\_layout(G)

nx.draw(G, pos, with\_labels=True, node\_size=500, node\_color='lightblue', font\_size=12)

plt.title("Network Graph")

plt.axis('off')

plt.show()

G.add\_node(1)

display\_graph(G) # Display the graph after adding a node

G.add\_nodes\_from([2, 3, 4, 7, 9])

G.add\_edges\_from([(1, 2), (3, 1), (2, 4), (4, 1), (9, 1), (1, 7), (2, 9)])

display\_graph(G) # Display the graph after adding nodes and edges

G.remove\_node(3)

display\_graph(G) # Display the graph after removing a node

G.remove\_edge(1, 2)

display\_graph(G) # Display the graph after removing an edge

n = G.number\_of\_nodes()

m = G.number\_of\_edges()

print("Number of nodes:", n)

print("Number of edges:", m)

d = G.degree(2)

print("Degree of node 2:", d)

neighbor\_list = list(G.neighbors(2))

print("Neighbors of node 2:", neighbor\_list)

G.clear()

data = [25, 30, 15, 10, 20]

labels = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']

plt.figure(1)

plt.pie(data, labels=labels, autopct='%1.1f%%')

plt.title('Pie Chart')

plt.figure(2)

plt.bar(labels, data)

plt.title('Bar Chart')

x = np.arange(0, 10, 0.1)

y = np.sin(x)

plt.figure(3)

plt.plot(x, y)

plt.title('Line Plot')

x = np.random.rand(50)

y = np.random.rand(50)

plt.figure(4)

plt.scatter(x, y)

plt.title('Scatter Plot')

data = np.random.normal(0, 1, 1000)

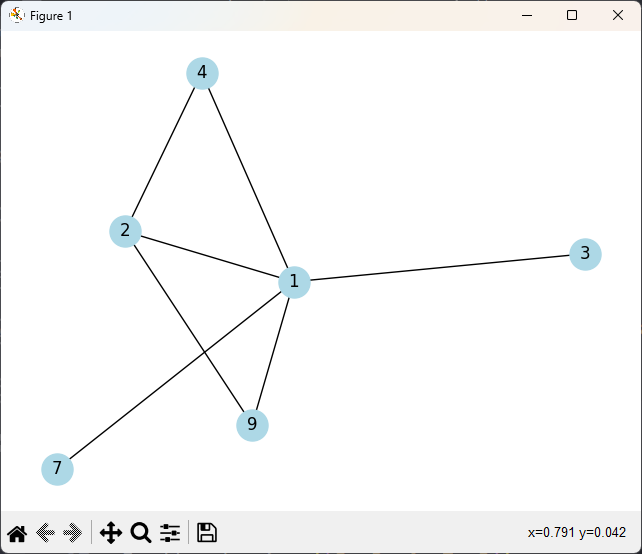
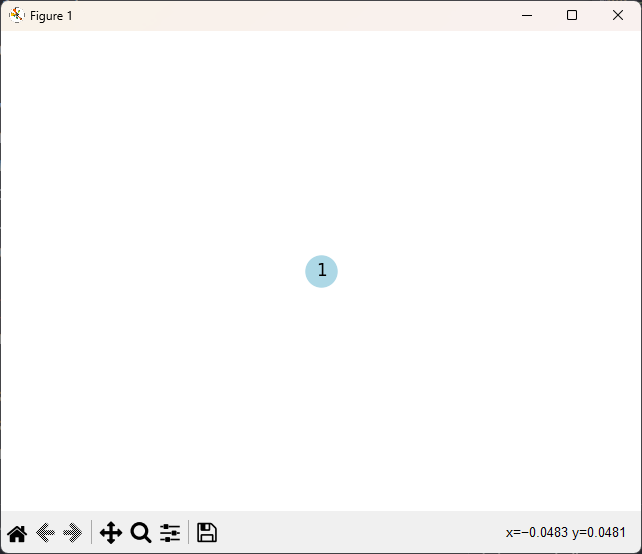
plt.figure(5)

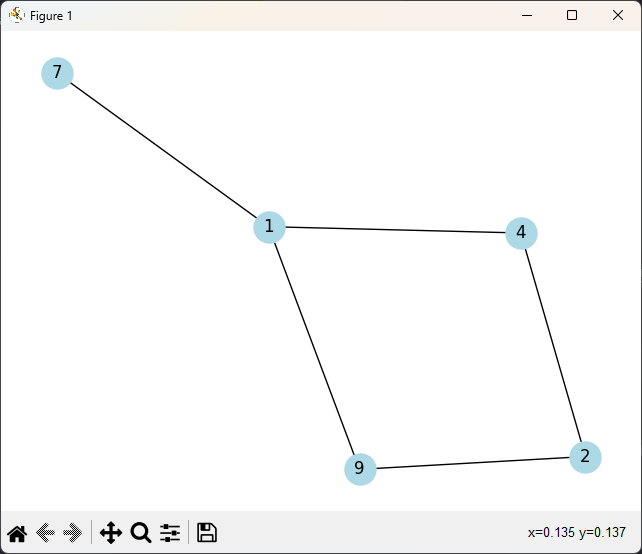
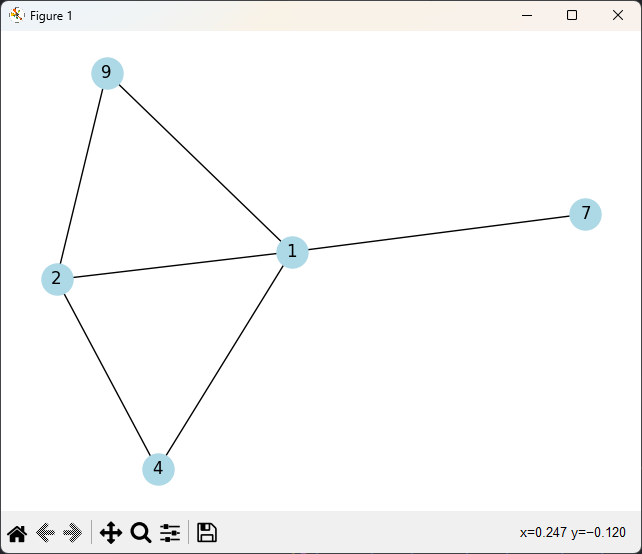
plt.hist(data, bins=30)

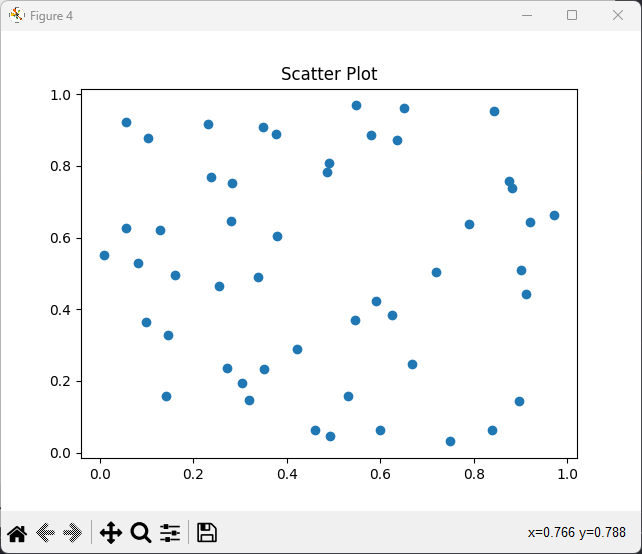
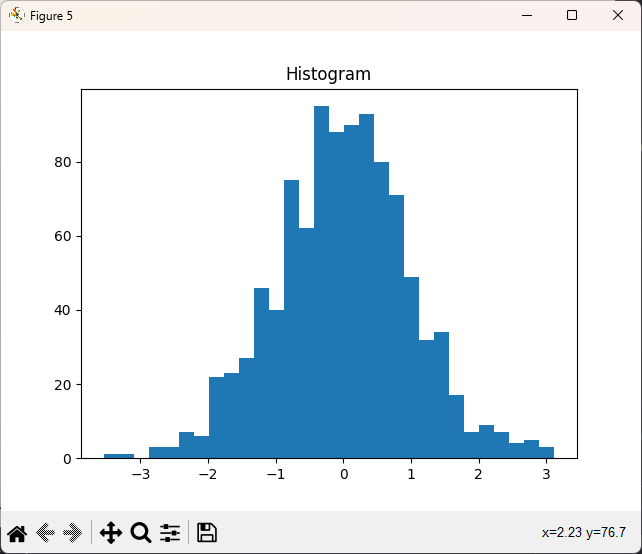
plt.title('Histogram')

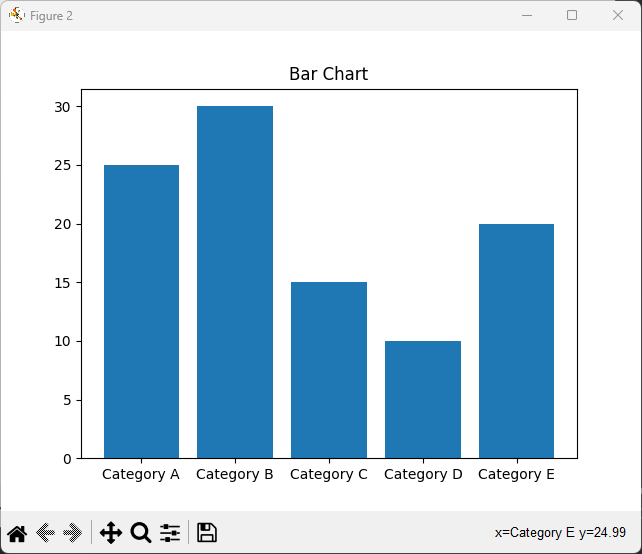
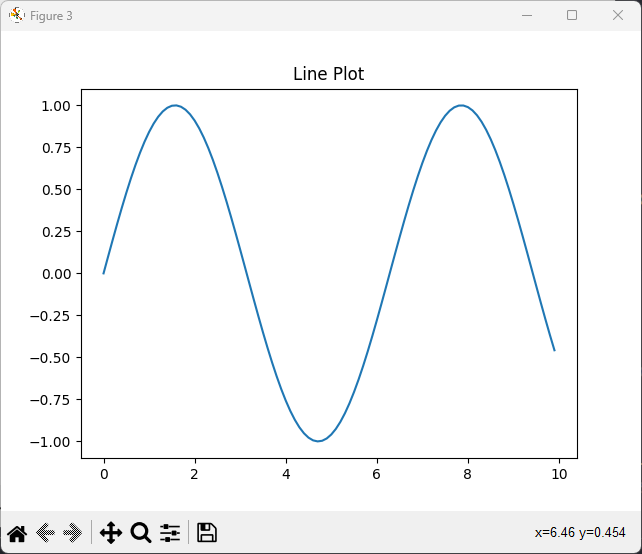
plt.show()

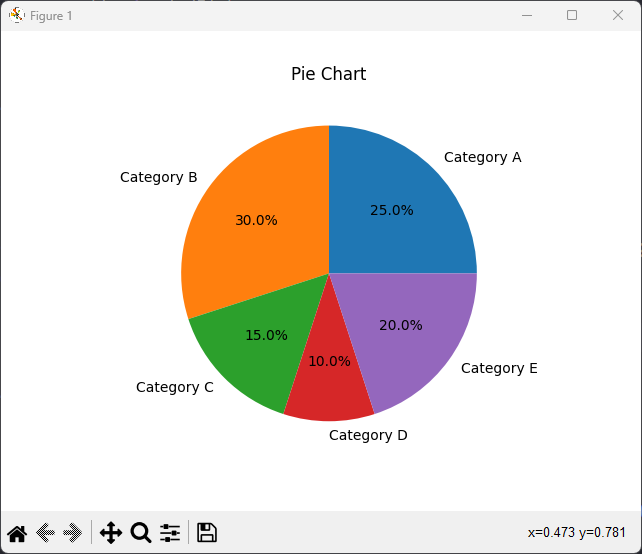
**Output:**









­­­­­­­­

**PRACTICAL-13**

**AIM:** Write a program to implement all the functionalities of the Scikit-learn library in Python

**Source Code:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.feature\_selection import SelectKBest, f\_classif

from sklearn.model\_selection import GridSearchCV

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train a Random Forest classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = clf.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

report = classification\_report(y\_test, y\_pred)

print("Classification Report:\n", report)

# K-Means clustering

kmeans = KMeans(n\_clusters=3)

kmeans.fit(X)

# Principal Component Analysis (PCA)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# SelectKBest for feature selection

selector = SelectKBest(score\_func=f\_classif, k=2)

X\_new = selector.fit\_transform(X, y)

# Grid Search for hyperparameter tuning

param\_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}

grid\_search = GridSearchCV(SVC(), param\_grid, cv=3)

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

print("Best Hyperparameters:", best\_params)

**Output:**

