”Program 01*

```
Import os
Print(“Operating System: “, os.name)
Print(“Working Directory: “, os.getcwd())
Files_and_dirs = os.listdir()
Files = []  Directories = []
For item in files_and_dirs:
 If os.path.isfile(item):
 Files. Append(item)
 Elif os.path.isdir(item):
 Directories. Append(item)
Print(“\nFiles:”)
For file in files:
 Print(file)
  Print(“\nDirectories:”)
For directory in directories:
 Print(directory)
```

PROGRAM NO. 2

```
import struct
def array_to_bytes(array):
      format_string = '{}{}'.format(len(array), 'B')
     packed_data = struct.pack(format_string, *array)
     return packed_data
def bytes_to_array(bytes_data):
      format_string = '{}{}'.format(len(bytes_data), 'B')
    unpacked_data = struct.unpack(format_string, bytes_data)
    return list(unpacked_data)
input_array = [10, 20, 30, 40, 50]
bytes_data = array_to_bytes(input_array)
```

```
print("Array as bytes:", bytes_data)
output_array = bytes_to_array(bytes_data)
print("Bytes as array:", output_array)
```

*****Program 03*****

```
import os
import time
Def get_file_info(file_path):
If not os.path.exists(file_path):
Print(“File does not exist.”)
Return
File_mode = os.stat(file_path).st_mode
Print(“File Mode:”, file_mode)
Local_time = os.path.getmtime(file_path)
Local_time_components = time.localtime(local_time)
Print(“Local Time:”)
Print(“Year:”, local_time_components.tm_year)
Print(“Month:”, local_time_components.tm_mon)
Print(“Day:”, local_time_components.tm_mday)
Print(“Hour:”, local_time_components.tm_hour)
Print(“Minute:”, local_time_components.tm_min)
Print(“Second:”, local_time_components.tm_sec)
Gmt_time = os.path.getmtime(file_path)
Gmt_time_components = time.gmtime(gmt_time)
Print(“\nGMT (UTC) Time:”) Print(“Year:”, gmt_time_components.tm_year)
Print(“Month:”, gmt_time_components.tm_mon) Print(“Day:”, gmt_time_components.tm_mday)
Print(“Hour:”, gmt_time_components.tm_hour) Print(“Minute:”, gmt_time_components.tm_min)
Print(“Second:”, gmt_time_components.tm_sec)
File_path = “path/to/your/file.txt”
Get_file_info(file_path)
```

*****Program 04*****

```
Import socket
Def connect_to_google():
Host = www.google.com
Port = 80
Try:
    Client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    Client_socket.connect((host, port))
    Print("Connected to Google successfully.")
   Client_socket.close()
Except socket.error as e:
   Print("Failed to connect to Google. Error:", e)
If __name__ == "__main__":
Connect_to_google()
```

*****Program 05*****

```
Import numpy as np
A = np.array([1, 2, 3, 4, 5])
B = np.array([6, 7, 8, 9, 10])
Print("Array a:", a)  Print("Array b:", b)
Print("Sum of arrays a and b:", np.add(a, b))   Print("Difference of arrays a and b:", np.subtract(a, b))
Print("Product of arrays a and b:", np.multiply(a, b))   Print("Division of arrays a and b:", np.divide(a, b))
Print("Square root of array a:", np.sqrt(a)) .  Print("Exponential of array a:", np.exp(a))
 Print("Minimum value of array a:", np.min(a))    Print("Maximum value of array b:", np.max(b))
Print("Mean of array a:", np.mean(a))     Print("Standard deviation of array b:", np.std(b))
Print("Sum of all elements in array a:", np.sum(a))
C = np.array([[1, 2], [3, 4], [5, 6]])
Print("Array c:")   Print(c)    Print("Reshaped array c (2 rows, 3 columns):")  Print(np.reshape(c, (2, 3)))
D = np.array([[1, 2, 3], [4, 5, 6]])
Print("Array d:") .  Print(d) .  Print("Transposed array d:").   Print(np.transpose(d)).
```

*****Program 06*****

```
Import pandas as pd
Data = {
      ‘Name’: [‘John’, ‘Emma’, ‘Sam’, ‘Lisa’, ‘Tom’],
       ‘Age’: [25, 30, 28, 32, 27],
      ‘Country’: [‘USA’, ‘Canada’, ‘Australia’, ‘UK’, 'Germany'],
       'Salary': [50000, 60000, 55000, 70000, 52000]
}
df = pd.DataFrame(data)
print("Original DataFrame:")   print(df)
name_age = df[['Name', 'Age']]
print("\nName and Age columns:")   print(name_age)
filtered_df = df[df['Country'] == 'USA']
print("\nFiltered DataFrame (Country = 'USA'):")    print(filtered_df)
sorted_df = df.sort_values('Salary', ascending=False)
print(“\nSorted DataFrame (by Salary in descending order):”)
print(sorted_df)
Average_salary = df[‘Salary’].mean()
Print(“\nAverage Salary:”, average_salary)
Df[‘Experience’] = [3, 6, 4, 8, 5]
Print(“\nDataFrame with added Experience column:”)   Print(df)
 Df.loc[df[‘Name’] == ‘Emma’, ‘Salary’] = 65000
Print(“\nDataFrame after updating Emma’s Salary:”)    Print(df)
Df = df.drop(‘Experience’, axis=1)
Print(“\nDataFrame after deleting Experience column:”)    Print(df)
```

*****Program 07*****

```
Import matplotlib.pyplot as plt
Import numpy as np
X = np.linspace(0, 10, 100)    Y = np.sin(x)
Plt.figure()   Plt.plot(x, y)    Plt.title("Line Chart")    Plt.xlabel("X-axis")    Plt.ylabel("Y-axis")
Categories = ['A', 'B', 'C', 'D']
Values = [20, 35, 30, 25]
Plt.figure()   Plt.bar(categories, values) . Plt.title("Bar Chart")  Plt.xlabel("Categories").Plt.ylabel("Values")
X = np.random.randn(100)
Y = np.random.randn(100)
Colors = np.random.rand(100)
Sizes = 100 * np.random.rand(100)
Plt.figure()   Plt.scatter(x, y, c=colors, s=sizes, alpha=0.5). Plt.title("Scatter Plot")   Plt.xlabel("X-axis")
Plt.ylabel("Y-axis")
Sizes = [30, 20, 25, 15, 10]
Labels = ['A', 'B', 'C', 'D', 'E']
Plt.figure()   Plt.pie(sizes, labels=labels, autopct='%1.1f%%')    Plt.title("Pie Chart") . Plt.show()
```

*****Program 08*****

```
Import pandas as pd
Df = pd.read_excel('data.xlsx')
Print("First few rows:")  Print(df.head())  Print("\nSummary statistics:")  Print(df.describe())
Filtered_data = df[df['Age'] > 30]
Print("\nFiltered data (Age > 30):")   Print(filtered_data)
Sorted_data = df.sort_values(by='Salary', ascending=False)
Print("\nSorted data (by Salary):")   Print(sorted_data)
Df['Bonus'] = df['Salary'] * 0.1
Print("\nData with new column (Bonus):") Print(df)
Df.to_excel('output.xlsx', index=False).
Print("\nData written to output.xlsx")
```

*****Program 09*****

```
Import nltk
From sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
From sklearn.model_selection import train_test_split
From sklearn.svm import LinearSVC
From sklearn.metrics import accuracy_score
Sentences = ['I love Python programming.',
    'I dislike writing code.',
    'Machine learning is fascinating.',
    'Natural language processing is challenging.']
Labels = ['positive', 'negative', 'positive', 'negative']
 Nltk.download('punkt')
Corpus = [nltk.word_tokenize(sentence) for sentence in sentences]
Vectorizer = CountVectorizer()
X = vectorizer.fit_transform([' '.join(sentence) for sentence in corpus])
 Transformer = TfidfTransformer()
X = transformer.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)
Classifier = LinearSVC()
Classifier.fit(X_train, y_train)
Y_pred = classifier.predict(X_test)
Accuracy = accuracy_score(y_test, y_pred)
Print('Accuracy:', accuracy)
```