LAB EXPERIMENTS

PROGRAM NO. 1

AIM: -Python program to display details about the operating system, working directory, files, and directories in the current directory, lists the files and all directories, scan and classify them as directories and files.

SOURCE CODE: -

```
import os

# Display operating system details
print("Operating System: ", os.name)

# Display working directory
print("Working Directory: ", os.getcwd())

# Get list of files and directories in the current directory
files_and_dirs = os.listdir()

# Separate files and directories
files = []
directories = []

for item in files_and_dirs:
    if os.path.isfile(item):
        files. Append(item)
    elif os.path.isdir(item):
       directories. Append(item)


# Display list of files
print("\nFiles:")
for file in files:
   print(file)

# Display list of directories
print("\nDirectories:")
for directory in directories:
   print(directory)
```

OUTPUT:

```
Operating System: posix
Current Working Directory: /path/to/current/directory

Files in the current directory:
file1.txt
file2.py
file3.jpg

Directories in the current directory:
dir1
dir2
```

PROGRAM NO. 2

AIM: -Python program to convert an array to an array of machine values and vice versa.

SOURCE CODE: -

```
import struct
def array_to_bytes(array):
        # Convert array to bytes
        format_string = '{}{}'.format(len(array), 'B')
        packed_data = struct.pack(format_string, *array)
        return packed_data


def bytes_to_array(bytes_data):
        # Convert bytes to array
        format_string = '{}{}'.format(len(bytes_data), 'B')
        unpacked_data = struct.unpack(format_string, bytes_data)
        return list(unpacked_data)
# Example usage
input_array = [10, 20, 30, 40, 50]
# Convert array to bytes
bytes_data = array_to_bytes(input_array)
print("Array as bytes:", bytes_data)
# Convert bytes to array
output_array = bytes_to_array(bytes_data)
print("Bytes as array:", output_array)
```

OUTPUT:

Array as bytes: b'\n\x14\x1e(2'
Bytes as array: [10, 20, 30, 40, 50]

PROGRAM NO. 3

AIM: -Python program to get information about the file pertaining to the file mode and to get time values with components using local time and gm time.

SOURCE CODE: -

```
import os
import time
# Get file information
def get_file_info(file_path):
# Check if file exists
if not os.path.exists(file_path):
print("File does not exist.")
return
# Get file mode
file_mode = os.stat(file_path).st_mode
print("File Mode:", file_mode)
# Get time values using local time
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)
# Get time values using GMT (UTC)
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)
# Example usage
file_path = "path/to/your/file.txt"
get_file_info(file_path)
```

OUTPUT:

```
File Mode: 33188
Local Time:
Year: 2023
Month: 5
Day: 15
Hour: 10
Minute: 30
Second: 45

GMT (UTC) Time:
Year: 2023
Month: 5
Day: 15
Hour: 15
Minute: 30
Second: 45
```

PROGRAM NO. 4

AIM: -Python program to connect to Google using socket programming.

SOURCE CODE: -

```
import socket
def connect_to_google():
        host = "www.google.com"
        port = 80
try:
    # Create a socket object
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to Google server
    client_socket.connect((host, port))

    print("Connected to Google successfully.")

   # Close the socket connection
    client_socket.close()

except socket.error as e:
    print("Failed to connect to Google. Error:", e)

if __name__ == "__main__":
connect_to_google()
```

Output:

Connected to Google Successfully

PROGRAM NO. 5

AIM: -Python program to perform Array operations using Numpy package.

SOURCE CODE: -

```
import numpy as np
# Create arrays
a = np.array([1, 2, 3, 4, 5])
b = np.array([6, 7, 8, 9, 10])

# Basic operations
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))

# Aggregation operations
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))
```

```
# Reshaping arrays
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)))


# Transposing arrays
d = np.array([[1, 2, 3], [4, 5, 6]])
print("Array d:")
print(d)
print("Transposed array d:")
print(np.transpose(d))
```

Output:

```
Array a: [1 2 3 4 5]
Array b: [ 6  7  8  9 10]
Sum of arrays a and b: [ 7  9 11 13 15]
Difference of arrays a and b: [-5 -5 -5 -5 -5]
Product of arrays a and b: [ 6 14 24 36 50]
Division of arrays a and b: [0.16666667 0.28571429 0.375      0.44444444 0.5       ]
Square root of array a: [1.         1.41421356 1.73205081 2.         2.23606798]
Exponential of array a: [  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
Minimum value of array a: 1
Maximum value of array b: 10
Mean of array a: 3.0
Standard deviation of array b: 1.4142135623730951
Sum of all elements in array a: 15
Array c:
[[1 2]
 [3 4]
 [5 6]]
Reshaped array c (2 rows, 3 columns):
[[1 2 3]
 [4 5 6]]
Array d:
[[1 2 3]
 [4 5 6]]
Transposed array d:
[[1 4]
 [2 5]
 [3 6]]
```

PROGRAM NO. 6

AIM: -Python program to perform Data Manipulation operations using Pandas package.

SOURCE CODE: -

```
import pandas as pd

# Create a DataFrame
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)

# Selecting columns
name_age = df[['Name', 'Age']]
print("\nName and Age columns:")
print(name_age)

# Filtering rows
filtered_df = df[df['Country'] == 'USA']
print("\nFiltered DataFrame (Country = 'USA'):")
print(filtered_df)

# Sorting by a column
sorted_df = df.sort_values('Salary', ascending=False)
```

```
print("\nSorted DataFrame (by Salary in descending order):")
print(sorted_df)

# Aggregating data
average_salary = df['Salary'].mean()
print("\nAverage Salary:", average_salary)

# Adding a new column
df['Experience'] = [3, 6, 4, 8, 5]
print("\nDataFrame with added Experience column:")
print(df)

# Updating values
df.loc[df['Name'] == 'Emma', 'Salary'] = 65000
print("\nDataFrame after updating Emma's Salary:")
print(df)

# Deleting a column
df = df.drop('Experience', axis=1)
print("\nDataFrame after deleting Experience column:")
print(df)
```

Output:

Original DataFrame:

	Name	Age	Country	Salary
0	John	25	USA	50000
1	Emma	30	Canada	60000
2	Sam	28	Australia	55000
3	Lisa	32	UK	70000
4	Tom	27	Germany	52000

Name and Age columns:

	Name	Age
0	John	25
1	Emma	30
2	Sam	28
3	Lisa	32
4	Tom	27

Filtered DataFrame (Country = 'USA'):

	Name	Age	Country	Salary
0	John	25	USA	50000

Sorted DataFrame (by Salary in descending order):

	Name	Age	Country	Salary
3	Lisa	32	UK	70000
1	Emma	30	Canada	60000
2	Sam	28	Australia	55000
4	Tom	27	Germany	52000
0	John	25	USA	50000

Average Salary: 57400. 0

DataFrame with added Experience column:

	Name	Age	Country	Salary	Experience
0	John	25	USA	50000	3
1	Emma	30	Canada	60000	6
2	Sam	28	Australia	55000	4
3	Lisa	32	UK	70000	8
4	Tom	27	Germany	52000	5

DataFrame after updating Emma's Salary:

	Name	Age	Country	Salary	Experience
0	John	25	USA	50000	3
1	Emma	30	Canada	65000	6
2	Sam	28	Australia	55000	4
3	Lisa	32	UK	70000	8
4	Tom	27	Germany	52000	5

DataFrame after deleting Experience column:

	Name	Age	Country	Salary
0	John	25	USA	50000
1	Emma	30	Canada	65000
2	Sam	28	Australia	55000
3	Lisa	32	UK	70000
4	Tom	27	Germany	52000

PROGRAM NO. 7

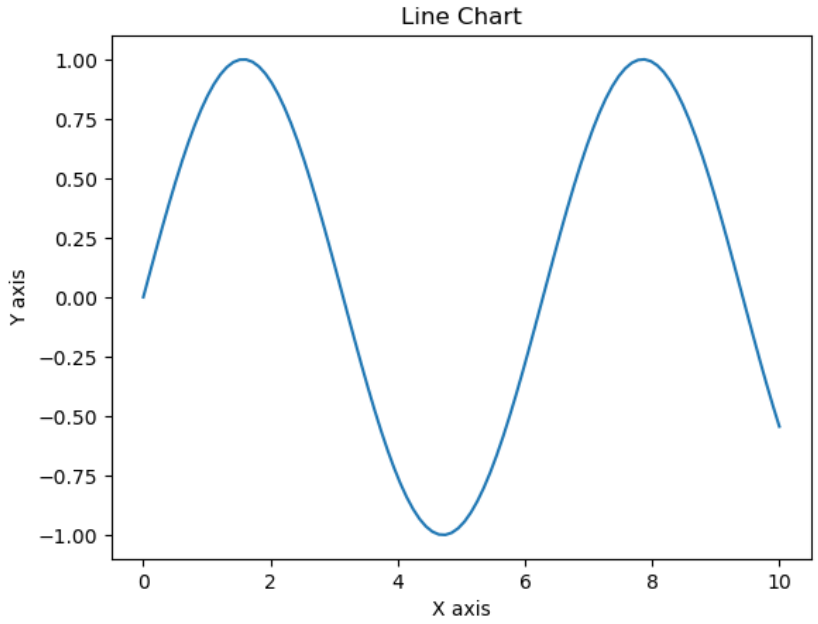
AIM: -Python program to display multiple types of charts using Matplotlib package.

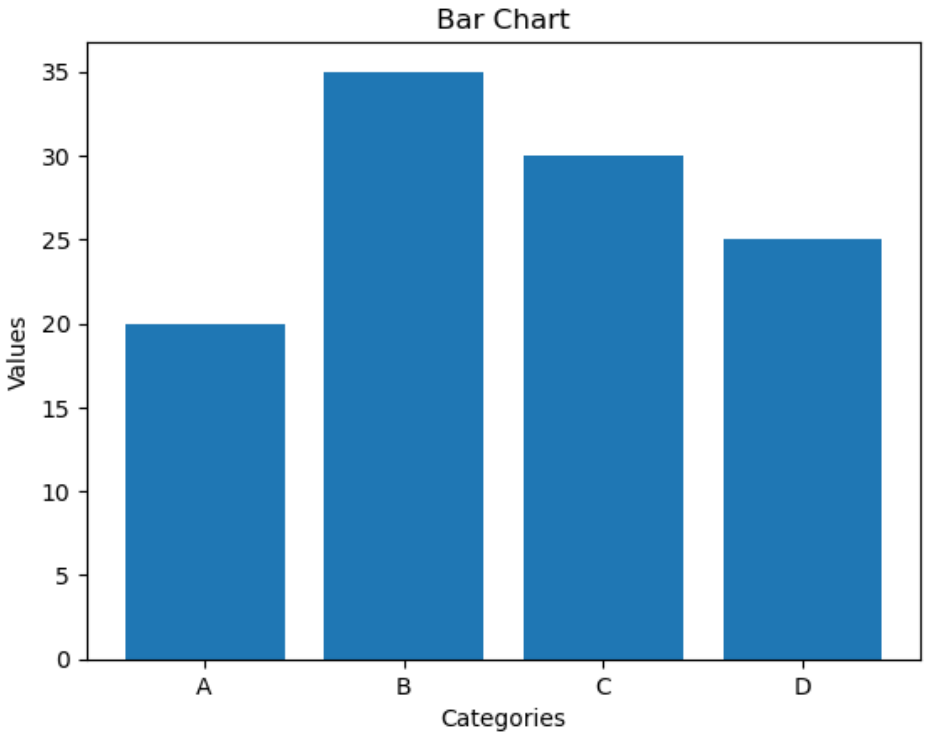
SOURCE CODE: -

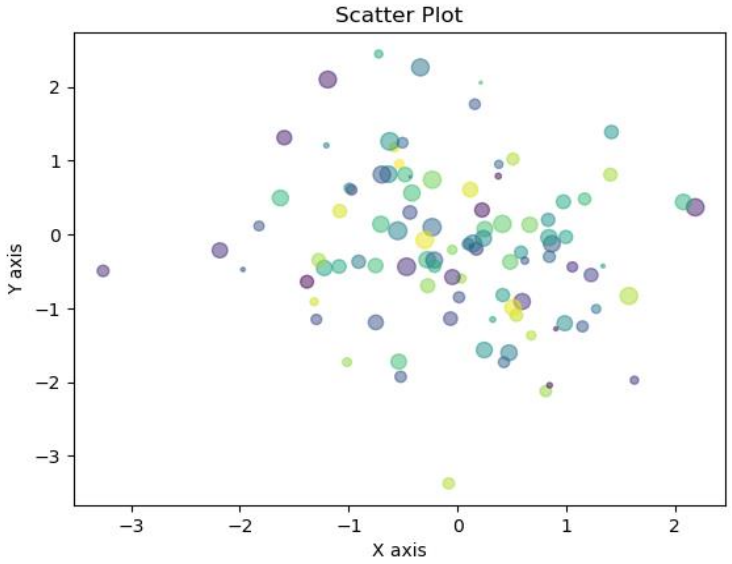
```
import matplotlib.pyplot as plt
import numpy as np

# Line chart
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")

# Bar chart
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")

# Scatter plot
x = np.random.randn(100)
y = np.random.randn(100)
colors = np.random.rand(100)
sizes = 100 * np.random.rand(100)
```

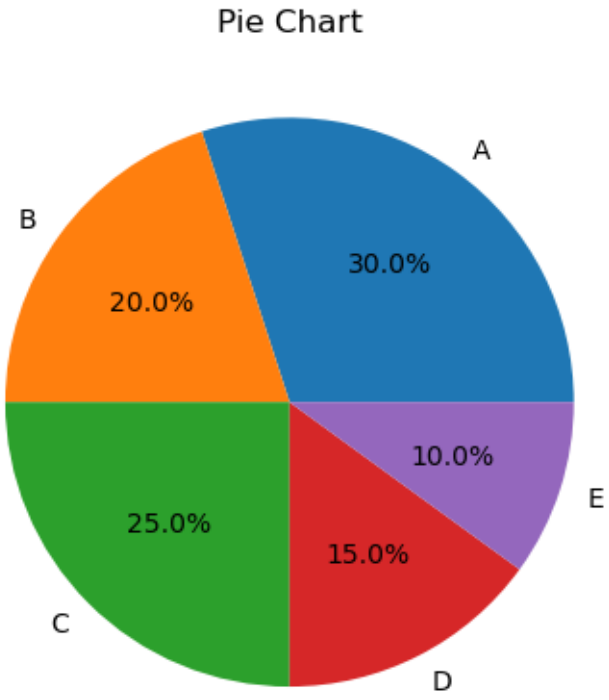
```
plt.figure()
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5)
plt.title("Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Pie chart
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")

# Show all the charts
plt.show()
```

OUTPUT:


Line Chart
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y axis
0
2
4
6
8
10
X axis


Bar Chart
35
30
25
20
15
10
5
0
Values
A
B
C
D
Categories


Scatter Plot
Y axis
X axis
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2


Pie Chart
A
B
C
D
E
30.0%
20.0%
25.0%
15.0%
10.0%

PROGRAM NO. 8

AIM: -Python program to perform File Operation on Excel Data Set.

SOURCE CODE: -

```
import pandas as pd
# Read Excel file
df = pd.read_excel('data.xlsx')

# Display first few rows
print("First few rows:")
print(df.head())

# Get summary statistics
print("\nSummary statistics:")
print(df.describe())

# Filter data
filtered_data = df[df['Age'] > 30]
print("\nFiltered data (Age > 30):")
print(filtered_data)

# Sort data
sorted_data = df.sort_values(by='Salary', ascending=False)
print("\nSorted data (by Salary):")
print(sorted_data)
```

```
# Add a new column
df['Bonus'] = df['Salary'] * 0.1
print("\nData with new column (Bonus):")
print(df)

# Write to Excel file
df.to_excel('output.xlsx', index=False)
print("\nData written to output.xlsx")
```

OUTPUT:

```
First few rows:
Name Age Salary
0 John 25 50000
1 Emma 30 60000
2 Sam 28 55000
3 Lisa 32 70000
4 Tom 27 52000

Summary statistics:
             Age        Salary
count    5.000000 5.000000
mean    28.400000 57400.000000
std      2.701851 8001.661438
min     25.000000 50000.000000
25%     27.000000 52000.000000
50%     28.000000 55000.000000
```

```
75%     30.000000 60000.000000
max     32.000000 70000.000000

Filtered data (Age > 30):
   Name   Age   Salary
3 Lisa       32     70000

Sorted data (by Salary):
    Name    Age    Salary
3 Lisa         32      70000
1 Emma    30      60000
2 Sam       28       55000
4 Tom    27    52000
0 John  25   50000


 Data with new column (Bonus):
    Name     Age    Salary    Bonus
0 John         25     50000     5000.0
1 Emma      30     60000     6000.0
2 Sam          28     55000     5500.0
3 Lisa          32      70000    7000.0
4 Tom          27      52000     5200.0

Data written to output.xlsx
```

PROGRAM NO. 9

AIM: -Python program to implement with Python Sci Kit-Learn & NLTK.

SOURCE CODE: -

```
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

# Prepare sample data
sentences = ['I love Python programming.',
        'I dislike writing code.',
        'Machine learning is fascinating.',
        'Natural language processing is challenging.']

labels = ['positive', 'negative', 'positive', 'negative']

# Tokenization and preprocessing
nltk.download('punkt')
corpus = [nltk.word_tokenize(sentence) for sentence in sentences]

# Convert corpus to feature vectors
vectorizer = CountVectorizer()
X = vectorizer.fit_transform([' '.join(sentence) for sentence in corpus])

# Apply TF-IDF transformation
transformer = TfidfTransformer()
X = transformer.fit_transform(X)
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Train a Linear SVM classifier
classifier = LinearSVC()
classifier.fit(X_train, y_train)

# Make predictions on test data
y_pred = classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

OUTPUT:

Accuracy: 0.5