# PYTHON PROGRAMMING
# DIGITAL ASSESSMENT – 1

**NAME:** ANANTHAN M

**REG NO:** 22MCA0017

1. **Elucidate the steps for installing Python and MYSQL (Server, Workbench, and Shell) in windows.**

   a) Installing Python:

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

      2. Download the latest version of Python for Windows by clicking on the "Download Python" button. Choose the appropriate installer based on your system architecture (32-bit or 64-bit).

      3. Run the installer file that you downloaded. Check the box that says "Add Python to PATH" during the installation process. This will make it easier to run Python from the command line.

      4. Follow the installation wizard instructions, selecting the desired installation location and customizing the installation options if needed.

      5. Once the installation is complete, open the command prompt and type "python" to check if Python is successfully installed. You should see the Python version information.

   b) Installing MySQL Server:

      1. Visit the official MySQL website at https://dev.mysql.com/downloads/installer/ and download the MySQL Installer for Windows.

      2. Run the MySQL Installer executable file that you downloaded.

      3. In the installer, choose the "Developer Default" setup type. This will install MySQL Server, MySQL Workbench, and other necessary components.

      4. Follow the installation wizard instructions, accepting the license agreement and selecting the installation directory.

      5. In the product selection screen, ensure that MySQL Server, MySQL Workbench, and MySQL Shell are selected for installation. You can also choose to install other optional components as per your requirements.

      6. Set a root password for MySQL Server during the installation process. Remember this password as you will need it later to access MySQL.

      7. Complete the installation by following the remaining steps of the installation wizard.

c) Installing MySQL Workbench:

1. Once the MySQL Server installation is complete, open the MySQL Workbench application.

2. Click on the "Local Instance MySQL" connection option in the Workbench home screen.

3. Enter the root password you set during the MySQL Server installation and click "Test Connection" to ensure that the connection is successful.

4. After the successful connection, you can start using MySQL Workbench for database management and administration.

d) Installing MySQL Shell:

1. Open the command prompt and navigate to the MySQL installation directory. By default, it is located at "C:\Program Files\MySQL\MySQL Server X.X\bin".

2. Run the MySQL Shell by typing "mysqlsh" in the command prompt. This will launch the MySQL Shell interactive mode.

3. You can now use the MySQL Shell to execute SQL queries, manage databases, and perform other tasks.

**Explain different MYSQL modules used to connect python to the server (MySQL Connector Python, PyMySQL, MySQLDB, MySqlClient, OurSQL). Discuss the advantages of each module. Create databases, tables, and perform the table operations.**

MySQL Connector Python:

MySQL Connector Python is an official MySQL driver provided by Oracle. It is a pure Python implementation that allows Python programs to communicate with MySQL databases.

Advantages:

- Developed and maintained by Oracle, ensuring reliability and compatibility with the latest MySQL server versions.
- Supports the Python Database API (Python DB API) specification, making it compatible with various Python database interfaces.
- Provides support for advanced MySQL features such as stored procedures, prepared statements, and server-side cursors.
- Works with both Python 2 and Python 3.

PyMySQL:

PyMySQL is a pure Python library that provides a client interface for connecting to MySQL databases. It is designed to be a drop-in replacement for MySQLdb, which is no longer actively maintained.

Advantages:

- Pure Python implementation, making it platform-independent.
- Compatible with both Python 2 and Python 3.

- Offers a lightweight and easy-to-use interface for connecting to MySQL databases.
- Supports advanced MySQL features such as SSL connections and server-side cursors.

MySQLDB (MySQL-python):

MySQLDB is one of the earliest and most widely used MySQL modules for Python. It is a C-based extension module that provides an interface to the MySQL client library.

Advantages:

- Mature and stable with a large user base.
- Offers a fast and efficient C-based interface for improved performance.
- Compatible with Python 2. However, it is not compatible with Python 3, and there is no official Python 3 support.

mysqlclient:

mysqlclient is a fork of MySQLdb that is compatible with both Python 2 and Python 3. It provides a C-based interface similar to MySQLDB for connecting to MySQL databases.

Advantages:

- Compatible with Python 2 and Python 3, making it suitable for cross-version Python development.
- Offers improved compatibility and bug fixes compared to MySQLdb.
- Supports advanced MySQL features and SSL connections.

OurSQL:

OurSQL is a pure Python MySQL library that aims to provide a simpler and more intuitive API compared to other modules.

Advantages:

- Pure Python implementation, making it platform-independent.
- Supports both Python 2 and Python 3.
- Provides additional features such as automatic character set handling and automatic type conversion.
- Offers a simplified API for easier usage.

1. Install the MySQL Connector Python module by running the command: `pip install mysql-connector-python

2. Import the module in your Python script:

```
import mysql.connector
```

3. Establish a connection to the MySQL server:

```
  mydb = mysql.connector.connect(
      host="localhost",
      user="yourusername",
      password="yourpassword"
  )
```

4. Create a database:

```
  mycursor = mydb.cursor()
  mycursor.execute("CREATE DATABASE mydatabase")
```

5. Use the created database:

```
  mycursor.execute("USE mydatabase")
```

6. Create a table:

```
  mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY
KEY, name VARCHAR(255), email VARCHAR(255))")
```

7. Insert data into the table:

```
  sql = "INSERT INTO customers (name, email) VALUES (%s, %s)"
  val = ("John Doe", "johndoe@example.com")
  mycursor.execute(sql, val)
  mydb.commit()
```

8. Retrieve data from the table:

```
  mycursor.execute("SELECT * FROM customers")
  result = mycursor.fetchall()
  for row in result:
      print(row)
```

9. Update data in the table:

```
  sql = "UPDATE customers SET email = 'newemail@example.com' WHERE id = 1"
  mycursor.execute(sql)
  mydb.commit()
```

10. Delete data from the table:

```
   sql = "DELETE FROM customers WHERE id = 1"
   mycursor.execute(sql)
```

```
    mydb.commit()
```

2. **What is numpy in python? How is it different from python lists? Define broadcasting in numpy with an example. Explain any five methods that can be applied on numpy arrays with an example.**

NumPy is a fundamental package in Python for scientific computing and data manipulation. It provides a powerful N-dimensional array object, along with functions for working with arrays. Here's an explanation of NumPy and how it differs from Python lists, followed by an explanation of broadcasting and five commonly used NumPy array methods:

1. NumPy vs. Python lists:

- NumPy arrays are homogeneous, meaning they can store elements of the same data type, whereas Python lists can contain elements of different types.

- NumPy arrays offer better performance and efficiency for numerical computations and mathematical operations compared to Python lists.

- NumPy arrays have a fixed size at creation, whereas Python lists can grow dynamically.

- NumPy arrays allow for vectorized operations, which enable element-wise computations on arrays without the need for explicit loops.

2. Broadcasting in NumPy:

- Broadcasting is a powerful feature in NumPy that allows arrays of different shapes to be used in arithmetic operations. It eliminates the need for explicit loops when performing element-wise computations on arrays.

- In broadcasting, the smaller array is "broadcasted" across the larger array to match their shapes. This allows element-wise operations to be performed even if the arrays have different dimensions.

- Broadcasting follows a set of rules to determine how arrays with different shapes can be combined. It automatically replicates the smaller array to match the shape of the larger array.

- Example:

```
import numpy as np


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

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

result = a + b

print(result)  # Output: [5, 7, 9]
```

Five commonly used methods in NumPy arrays:

1. numpy.reshape():

   - The `**reshape()**` method allows you to change the shape of an array without changing its data. It returns a new array with the modified shape.

   - Example:

```
import numpy as np

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

print(reshaped_arr)
# Output:
# [[1 2 3]
#  [4 5 6]]
```

2. numpy.mean():

   - The `**mean()**` method calculates the arithmetic mean of the elements in an array.

   - Example:

```
import numpy as np

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

print(mean_value)  # Output: 3.0
```

3. numpy.max():

   - The `**max()**` method returns the maximum value from an array.

- Example:

```
import numpy as np

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

print(max_value)  # Output: 5
```

4. numpy.min():
   - The **min()** method returns the minimum value from an array.
   - Example:

```
import numpy as np

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

print(min_value)  # Output: 1
```

5. numpy.argsort():
   - The **argsort()** method returns the indices that would sort an array in ascending order.
   - Example:

```
import numpy as np

arr = np.array([3, 1, 4, 2, 5])
sorted_indices = np.argsort(arr)
print(sorted_indices)  # Output: [1 2 3 4 5]
```

These are just a few examples of the many methods available in NumPy for array manipulation, mathematical operations, and statistical computations. NumPy's extensive functionality and efficient array processing make it a fundamental tool for scientific computing in Python.

3. **What is Pandas dataframe? List out and explain the methods that can be applied on pandas dataframe.**

A Pandas DataFrame is a two-dimensional tabular data structure in Python that is widely used for data manipulation and analysis. It provides labeled columns and rows, similar to a spreadsheet or a SQL table, allowing for efficient data organization and operations. Here are some commonly used methods that can be applied on Pandas DataFrame:

1. DataFrame.head():

- The `head()` method is used to retrieve the first few rows of a DataFrame. By default, it returns the first five rows.

- Example:

```
import pandas as pd


data = {'Name': ['John', 'Emma', 'Peter', 'Lisa', 'Michael'],
        'Age': [25, 28, 22, 30, 35],
        'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney']}


df = pd.DataFrame(data)
print(df.head())
# Output:
#     Name  Age      City
# 0   John   25  New York
# 1   Emma   28    London
# 2  Peter   22     Paris
# 3   Lisa   30     Tokyo
# 4 Michael   35    Sydney
```

2. DataFrame.describe():

- The `describe()` method provides summary statistics of numerical columns in a DataFrame, such as count, mean, standard deviation, minimum value, quartiles, and maximum value.

- Example:

```
import pandas as pd
```

```
data = {'Name': ['John', 'Emma', 'Peter', 'Lisa', 'Michael'],
        'Age': [25, 28, 22, 30, 35]}

df = pd.DataFrame(data)
print(df.describe())
# Output:
#             Age
# count   5.000000
# mean   28.000000
# std     5.744563
# min    22.000000
# 25%    25.000000
# 50%    28.000000
# 75%    30.000000
# max    35.000000
```

3. DataFrame.groupby():

 - The `groupby()` method allows you to group the DataFrame based on one or more columns and perform aggregation or transformation operations on the grouped data.

 - Example:

```
import pandas as pd

data = {'Name': ['John', 'Emma', 'Peter', 'Lisa', 'Michael'],
        'Age': [25, 28, 22, 30, 35],
        'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney']}

df = pd.DataFrame(data)
grouped_df = df.groupby('City')
print(grouped_df.mean())
# Output:
#           Age
# City
# London    28
# New York  25
# Paris     22
# Sydney    35
# Tokyo     30
```

4. DataFrame.sort_values():

   - The `sort_values()` method is used to sort the DataFrame by one or more columns in either ascending or descending order.

  - Example:

```
    import pandas as pd


    data = {'Name': ['John', 'Emma', 'Peter', 'Lisa', 'Michael'],
            'Age': [25, 28, 22, 30, 35],
            'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney']}


    df


 = pd.DataFrame(data)
    sorted_df = df.sort_values('Age', ascending=False)
    print(sorted_df)
    # Output:
    #       Name  Age       City
    # 4 Michael   35     Sydney
    # 3   Lisa   30      Tokyo
    # 1   Emma   28     London
    # 0   John   25   New York
    # 2  Peter   22      Paris
```

5. DataFrame.to_csv():

- The `to_csv()` method allows you to save the DataFrame as a CSV (Comma Separated Values) file.

- Example:

```python
import pandas as pd


data = {'Name': ['John', 'Emma', 'Peter', 'Lisa', 'Michael'],
        'Age': [25, 28, 22, 30, 35],
        'City': ['New York', 'London', 'Paris', 'Tokyo', 'Sydney']}


df = pd.DataFrame(data)
df.to_csv('data.csv', index=False)
# This will save the DataFrame as a CSV file named "data.csv" in the current directory.
```

These are just a few examples of the numerous methods available in Pandas for data manipulation, analysis, and visualization. The Pandas library offers a comprehensive set of tools for working with tabular data, making it a popular choice for data scientists and analysts.