UNIT-V

(Plotting Data using Matplotlib & Database Connectivity using MySQL)

Plotting Data using Matplotlib:-

Matplotlib is a popular data visualization library in python used to create different types of graphs and charts. it helps in representing data visually so that it becomes easier to understand patterns, trends and relationships in the data.

Matplotlib was developed by John D. Hunter in 2003. it is widely used in fields such as data science, machine learning, statistics, scientific research, and business analysis. the library provides many functions to generate plots like line graphs, bar charts, histograms, scatter plots, pie charts and more.

in Matplotlib the pyplot module is commonly used to create plots and charts. it provides functions similar to MATLAB for drawing graphs.

Installation of Matplotlib:-

pip install matplotlib

Features of Matplotlib:-

1. Simple and Easy to Use

Matplotlib is simple and easy to learn for beginners. It provides easy functions such as plot(), bar(), scatter(), and hist() to create graphs. With only a few lines of Python code, users can generate different types of charts. Because of its simplicity, it is widely used in educational and research environments.

2. Supports Multiple Types of Graphs

Matplotlib supports many types of graphs such as line graphs, bar charts, scatter plots, histograms, pie charts, and box plots. These graphs help visualize different types of data and make analysis easier. Users can choose the graph type depending on the nature of the data they want to represent.

3. Highly Customizable

Matplotlib allows users to customize almost every part of the graph. Users can change colors, line styles, markers, fonts, labels, titles, grid lines, and background colors. This helps in creating visually attractive and informative graphs. Customization makes the graphs suitable for presentations and reports.

4. Integration with NumPy and Pandas

Matplotlib works very well with NumPy and Pandas libraries. Data stored in NumPy arrays or Pandas DataFrames can be easily plotted using Matplotlib. This makes it very useful for data analysis and scientific computing.

5. Platform Independent

Matplotlib works on different operating systems such as Windows, Linux, and macOS. Since it is a Python library, it can run on any system that supports Python. This makes it flexible and widely usable in many environments.

6. Supports Interactive and Static Plots

Matplotlib can create both static images and interactive visualizations. Graphs can be displayed in Python environments like Jupyter Notebook, Spyder, and VS Code. Users can also save graphs as image files like PNG, JPG, PDF, and SVG.

Plotting Using Matplotlib:

Plotting using Matplotlib means creating graphs or charts to visualize data using the Matplotlib library in Python. Visualization helps us understand patterns, comparisons, and trends in data easily.

Matplotlib provides the pyplot module, which contains functions used to create different types of plots such as line plots, bar charts, histograms, scatter plots, and pie charts.

importing Matplotlib for Plotting:-

to create graphs, we usually import the pyplot module of Matplotlib.

```
import matplotlib.pyplot as plt
```

Explanation:

matplotlib → main library
pyplot → module used to create plots
plt → alias name for easy use

Ex:-

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,30,40]
plt.plot(x,y)
plt.show()
```

plot--> draws the graph
show() --> displays the graph on the screen.

Ex:-(Adding Title and Labels)

```
import matplotlib .pyplot as plt
x=[1,3,6,9]
y=[100, 250, 430, 350]
plt.plot(x,y)
plt.title("Sales Report")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.show()
```

Explanation:

title() → adds graph title
xlabel() → label for X-axis
ylabel() → label for Y-axis

Ex:-(Changing Line Style and Markers)

```
import matplotlib .pyplot as plt
x=[1,3,6,9]
y=[100, 250, 430, 350]
plt.plot(x,y, color='red', marker='o', linestyle='--' ,linewidth=5)
plt.title("Sales Report")
plt.xlabel("Months")
plt.ylabel("Sales")
plt.show()
```

Options used:

color → changes line color
marker → marks data points
linestyle → changes line style

Matplotlib Markers:-

makers in matplotlib are symbols used to highlight individual data points on a graph. when plotting data, makers help us clearly see where each data value is located. makers are used with the marker parameter inside the plt.plot() function.

commonly used markers in matplotlib:

Marker	Symbol Shape
'o'	Circle
's'	Square
'^'	Triangle Up
'v'	Triangle Down
'*'	Star

'+'	Plus
'x'	Cross
'D'	Diamond
'p'	Pentagon
'h'	Hexagon

Ex:-

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,15,30]
plt.plot(x,y, marker='o', markersize=8, markerfacecolor='red',
markeredgecolor='black')
plt.show()
```

Matplotlib Line(Line Styles):-

In Matplotlib, a line style defines the pattern or appearance of the line that connects data points in a graph. Line styles help make graphs more readable and allow us to distinguish between multiple lines plotted in the same graph.

Line styles are specified using the linestyle parameter in the plt.plot() function.

Line Style	Symbol	Description
Solid Line	'-'	Default continuous line
Dashed Line	'--'	Line with dashes
Dotted Line	':'	Line with dots
Dash-Dot Line	'-.'	Combination of dash and dot

Ex:-

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,15,30]
plt.plot(x,y,linestyle='--', linewidth=6)
plt.show()
```

Matplotlib Labels and Title:-

In Matplotlib, labels and titles are used to make graphs clear and understandable.

Title describes the overall purpose of the graph.
Labels describe the X-axis and Y-axis values.

title:-
the title() function is used to add a title to the graph.
Ex:- plt.title("Title Name")

x-axis label:-
the xlabel() function is used to add a label to the x-axis.
Ex:- plt.xlabel("X Axis Name")

y-axis label:-
the ylabel() function is used to add a label to the y-axis.

Ex:- plt.ylabel("Y Axis Name")

Ex:-
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[100,200,150,300]
plt.plot(x,y)
plt.title("Monthly Sales Report", fontsize=25, color='blue')
plt.xlabel("Months", fontsize=19)
plt.ylabel("sales amount", fontsize=19)
plt.show()
```

Grid Lines:-
Grid lines in Matplotlib are horizontal and vertical lines that appear in the background of a graph. They help users read the values of data points more easily by providing reference lines across the plot.
Grid lines improve the clarity and readability of graphs, especially when analyzing data trends.

In Matplotlib, grid lines are added using the grid() function.

Ex:-
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[12,25,18,30]
plt.plot(x,y,marker='o')
plt.title("student marks graph")
plt.xlabel('subjects')
plt.ylabel('marks')
plt.grid(color='blue', linestyle=':', linewidth=1)
plt.show()
```

Changing the Background Color:-

1. Changing the Whole Background Color

We use the figure() function with the facecolor parameter.

Ex:-

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,15,30]

plt.figure(facecolor='lightblue')
plt.plot(x,y)

plt.show()
```

Explanation:

plt.figure() creates the graph window.
facecolor='lightblue' changes the entire background color to light blue.
plt.plot() draws the graph.
plt.show() displays the graph.

Changing Only the Plot Area Background:-

Ex:-

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,15,30]
plt.plot(x,y)
plt.gca().set_facecolor('yellow')
plt.show()
```

Explanation:-

plt.plot(x,y) → draws the graph
plt.gca() → gets the current graph area
set_facecolor() → changes the background color of the plot
Example 2: Using Grid with gca()

Types of Graphs and Charts in Matplotlib

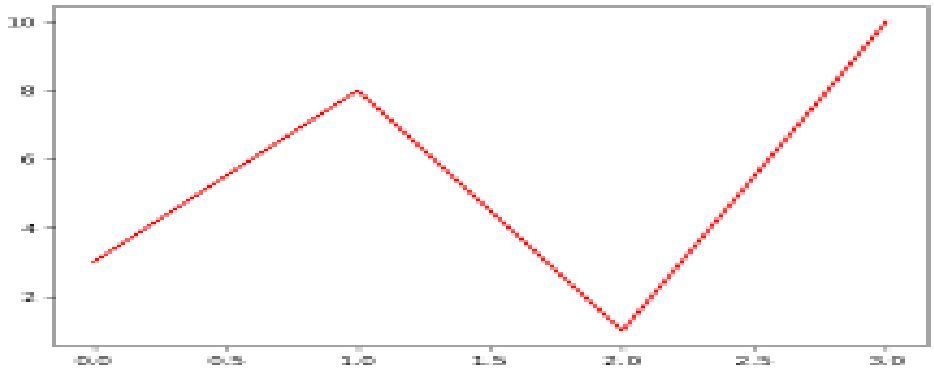
Matplotlib is a popular Python data visualization library used to create different types of graphs and charts. These graphs help represent data visually so that it becomes easier to understand patterns, comparisons, and trends. Matplotlib provides many plotting functions to display data in graphical form.

1. Line Graph
2. Bar Chart
3. Scatter Plot
4. Histogram
5. Pie Chart
6. Box Plot

1. Line Graph

A line Graph is used to show the relationship between two variables and display changes or trends over time. in this graph, data points are connected using straight lines. it is created using the plot() function in matplotlib.

Line Graphs are useful for showing continuous data such as temperature changes, sales growth, and stock market trends. they help in easily identifying increases and decreases in data values.



Ex:-

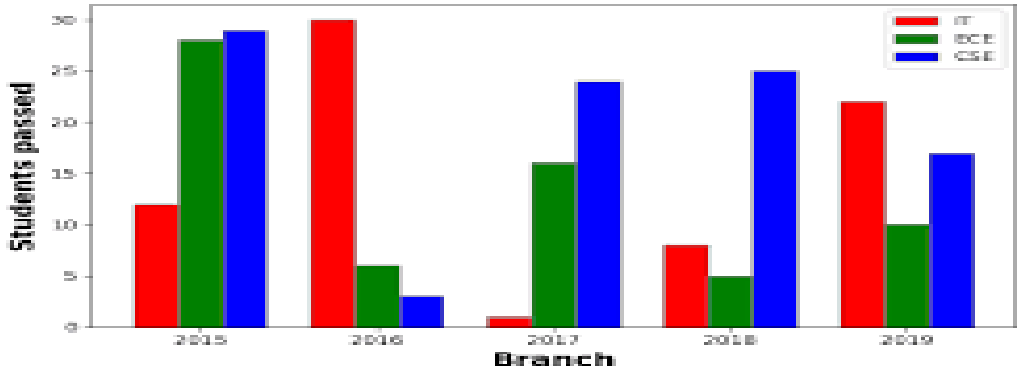
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,15,30]
plt.plot(x,y)
plt.show()
```

2.Bar Chart:-

A bar chart is used to compare different categories of data. It represents data using rectangular bars where the length or height of each bar corresponds to the value it represents. In Matplotlib, a bar chart is created using the bar() function. Bar charts can be displayed in vertical or horizontal form. They are useful for comparing values such as student marks, product sales, and population statistics.

Ex:-

```
import matplotlib.pyplot as plt
students=["A","B","C","D"]
marks=[80,90,75,85]
plt.bar(students,marks)
plt.show()
```



1.width parameter:- (values between 0 to 1)

The width parameter is used to control the width (thickness) of bars in a bar chart.

Ex:-

```
plt.bar(students,marks,width=0.4)
```

2. color parameter:-

The color parameter is used to change the color of bars.

Ex:-

```
plt.bar(students,marks,color=["red","blue","green","orange"])
```

3.edgecolor parameter:-

This parameter changes the border color of bars.

Ex:-

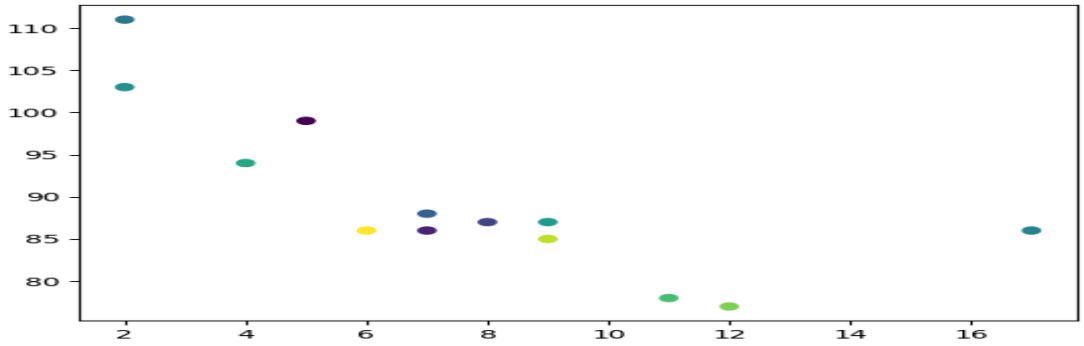
```
plt.bar(students,marks,color="yellow",edgecolor="black")
```

complete Example:-

```
import matplotlib.pyplot as plt

students=["A","B","C","D"]
marks=[80,90,75,85]

plt.bar(students,marks,width=0.5,color=["red","blue","green","orange"],edgecolor="
black",label="Marks")

plt.xlabel("Students")
plt.ylabel("Marks")
plt.title("Student Marks Bar Chart")

plt.legend()
plt.show()
```

3. Scatter Plot

A scatter plot is used to show the relationship between two numerical variables. It represents data points as dots on the graph. Each dot indicates the value of two variables on the X-axis and Y-axis. In Matplotlib, it is created using the scatter() function. Scatter plots are helpful in identifying correlations, patterns, and trends in data. They are widely used in data science, statistics, and machine learning for analyzing datasets.



Ex:-

```
import matplotlib.pyplot as plt
x=[5,7,8,7,2]
y=[99,86,87,88,100]

plt.scatter(x,y)
plt.show()
```

1. color Parameter

The color parameter is used to change the color of the points in the scatter plot.

Ex:-

```
plt.scatter(x, y, color="red")
```

2. s Parameter (Size of Points)

The s parameter is used to change the size of the points (dots) in the scatter plot.

Ex:-

```
plt.scatter(x, y, s=100)
```

3. marker Parameter

The marker parameter is used to change the shape of the points.

Example:

```
plt.scatter(x, y, marker="*")
```

Common markers:

o → circle

* → star

^ → triangle

s → square

4.edgecolors Parameter

The edgecolors parameter is used to change the border color of the points.

Ex:-

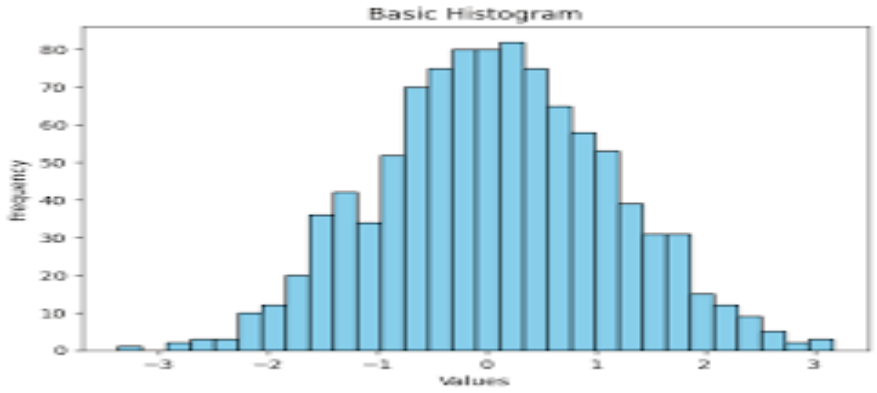
```
plt.scatter(x, y, color="yellow", edgecolors="black")
```

Complete Example:-

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2]
y = [99,86,87,88,100]

plt.scatter(x, y, color="blue", s=120, marker="*", alpha=0.7, edgecolors="black")

plt.title("Customized Scatter Plot")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.show()
```

4. Histogram

A histogram is used to represent the distribution of numerical data. It divides data into intervals called bins and displays the frequency of values in each bin. In Matplotlib, a histogram is created using the hist() function. Histograms help in understanding how data is spread across different ranges. They are commonly used for analyzing exam marks, age groups, and income distribution. Histograms are important tools in statistics and data analysis.



Ex:-

```
import matplotlib.pyplot as plt
marks = [40,45,50,55,60,65,70,75,80,85,90]
plt.hist(marks, bins=5, edgecolor='black')
plt.title("Histogram of Students Marks")
plt.xlabel("Marks Range")
plt.ylabel("Frequency")
plt.show()
```

Explanation of the Code:-

1.import matplotlib.pyplot as plt
Imports the matplotlib library for plotting graphs.
2.marks list
Contains the numerical data values.
3.plt.hist()
Function used to create a histogram.
4.bins=5
Divides the data into 5 intervals (ranges).
5.edgecolor='black'
Adds borders to bars for better visualization.
6.plt.title()
Displays the title of the graph.
7.plt.xlabel() and plt.ylabel()
Labels the X-axis and Y-axis.
8.plt.show()
Displays the histogram graph.

Important Customization Parameters:-

1. bins:-

used to divide the data into intervals or ranges.
more bins-more detailed distribution.
Ex:-
plt.hist(data, bins=5)

2. color:-

changes the color of histogram bars.
Ex:-
plt.hist(data, bins=5, color='blue')

3.edgecolor:-

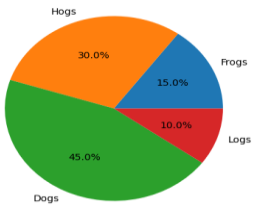
adds a border color to bars.
Ex:-
plt.hist(data, bins=5, edgecolor='black')

4.label:-

adds legend label to the histogram.
Ex:-
plt.hist(data, bins=5, label="Marks")
plt.legend()

5. Pie Chart

A pie chart is a circular chart used to represent data as parts of a whole. Each section of the pie represents a percentage of the total data. In Matplotlib, it is created using the pie() function. Pie charts are useful for showing proportions and percentages. They are commonly used in business reports, market share analysis, and budget distribution. Pie charts make it easy to understand the contribution of each category.



Ex:-

```
import matplotlib.pyplot as plt

sizes=[40,30,20,10]
labels=["A","B","C","D"]

plt.pie(sizes,labels=labels)
plt.show()
```

Important Customized Parameters in plt.pie():-

labels:-

used to display category names on each slice.

Ex:-

```
plt.pie(values, labels=subjects)
```

2. colors:-

used to change the color of slices.

Ex:-

```
plt.pie(values, labels=subjects, colors=['red','blue','green','gold'])
```

3. autopct:-

displays percentage values on each slice.

Ex:-

```
plt.pie(values, labels=subjects, autopct='%1.1f%%')
```

4. explode:-

used to separate a slice from the pie chart.

Ex:-
explode = (0,0.2,0,0)

plt.pie(values, labels=subjects, explode=explode)
Here the second slice moves slightly outside.

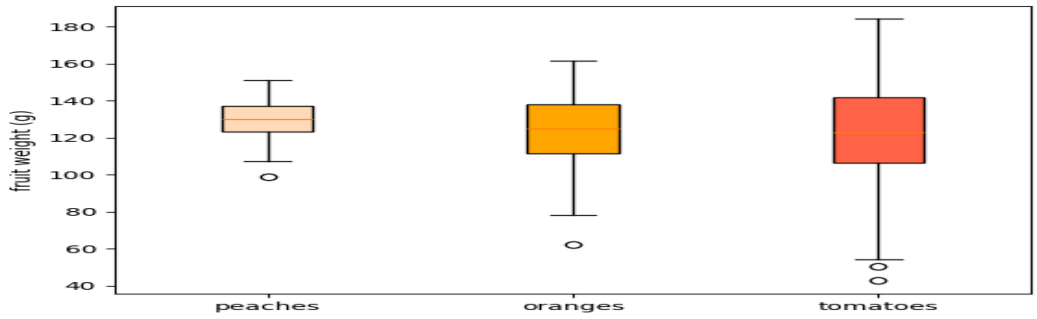
5. startangle:-
rotates the starting position of the pie chart.
Ex:-
plt.pie(values, labels=subjects, startangle=90)

Complete Example:-

```
import matplotlib.pyplot as plt

subjects = ['Maths','Science','English','Social']
marks = [30,25,20,25]

explode = (0,0.1,0,0)
plt.pie(marks,
    labels=subjects,
    colors=['red','blue','green','gold'],
    autopct='%1.1f%%',
    explode=explode,
    shadow=True,
    startangle=90)

plt.title("Subjects Marks Distribution")

plt.show()
```

Explanation of the Program
labels → shows subject names
colors → different colors for slices
autopct → shows percentage values
explode → separates one slice
shadow=True → adds shadow effect
startangle=90 → rotates chart start position

6. Box Plot
A box plot is a graphical representation used to show the statistical distribution of data. It displays important values such as the minimum, maximum, median, and quartiles of the dataset. In Matplotlib, a box plot is created using the boxplot() function. Box plots also help identify outliers in the data. They are widely used in statistical analysis and research studies. Box plots help in understanding the spread and variability of data.



Ex:-

```
import matplotlib.pyplot as plt
data=[10,20,30,40,50]
plt.boxplot(data)
plt.show()
```

Python Database:-

A database is an organized collection of related data that is stored electronically in a computer system. it helps users to store, manage, retrieve and update information easily.

Types of Databases:-

Databases are mainly divided into two major types.

1.Relational Database(RDBMS)

2.Non-Relational Database(NoSQL)

1.Relational Database(RDBMS)

A Relational Database Management System (RDBMS) stores data in the form of tables.

Each table contains rows and columns.

Rows represent records, and columns represent fields. RDBMS uses SQL (Structured Query Language) to manage data.

Ex:- MySQL, PostgreSQL, SQLite

2. Non-relational Databases (NoSQL)

A Non-relational Database (NoSQL) stores data in flexible formats such as:

Documents

Key-value pairs

Graphs

Wide-column stores

It does not require fixed table structures.

NoSQL databases are used for handling large-scale and unstructured data.

Ex:- MongoDB, Apache Cassandra

Python MySQL:-

python MySQL means connecting a python program with a MySQL database. python can communicate with MySQL using a module called mysql.connector
this allows python to store data, retrieve data, update data, delete data. it is widely used in web applications and data management systems.
python works with MySQL using the mysql-connector-python library.

What is MySQL?

MySQL is an **open-source Relational Database Management System (RDBMS)** used to store, organize, and manage data in the form of tables. It uses **Structured Query Language (SQL)** to perform operations such as inserting, updating, deleting, and retrieving data. In MySQL, data is stored in rows and columns, making it easy to access and manage. It is widely used in web applications, banking systems, and student management software. MySQL is popular because it is fast, secure, reliable, and supports multiple users at the same time.

MySQL database:-

you can download a MySQL database at https://www.mysql.com/downloads/.

installing MySQL connector:-

pip install mysql-connector-python

import module

```
import mysql.connector
```

connect to Database:

```
conn = mysql.connector.connect(
   host="localhost",
   user="root",
   password="1234",
   database="college"
)
```

Explanation

host → database server address
user → MySQL username
password → MySQL password
database → database name

create cursor:-

```
cur = conn.cursor()
```

(A cursor is used to execute SQL commands.)

create table:-

```
cur.execute("""
CREATE TABLE IF NOT EXISTS student(
   id INT PRIMARY KEY,
   name VARCHAR(50),
   marks INT
)
""")
```

insert data

```
cur.execute("INSERT INTO student VALUES(1,'Devendra',85)")
conn.commit()
```

(commit() saves changes permanently.)

read data

```
cur.execute("SELECT * FROM student")
for row in cur.fetchall():
   print(row)
```

Q1).Explain database connectivites in python. Discuss the steps involved in importing Mysql, connecting to a database, forming a query and passing the query to Mysql.

(or)

How to connect to MySQL server? Explain steps to connect with a database ?

Ans:-

Database connectivity in Python means establishing a connection between a Python application and a database such as MySQL. This allows the program to store, retrieve, modify, and delete information from the database. Python uses special modules to communicate with database servers. One commonly used module is **mysql.connector**. By using database connectivity, Python programs can handle large amounts of data efficiently.

1. Importing MySQL Module

The first step in database connectivity is importing the required MySQL connector module into the Python program. This module contains predefined classes and methods that help Python communicate with the MySQL server. Without importing this module, Python cannot understand database commands. The **mysql.connector** package acts as a bridge between Python and MySQL.

Ex: import mysql.connector

2. Connecting to the Database

After importing the module, the next step is establishing a connection to the database server. The connection requires details such as hostname, username, password, and database name. The hostname usually refers to the local machine as **localhost**. The username identifies the MySQL user account. The password provides security for database access. The database name tells Python which database should be opened for operations. If the connection is successful, Python can start sending SQL commands.

Ex:-

```
con = mysql.connector.connect(
   host="localhost",
   user="root",
   password="1234",
   database="college"
)
```

3. Creating Cursor Object

A cursor object is created after the connection is established. The cursor acts as a control structure that allows SQL queries to be executed. It works like a pointer that sends commands to the database and retrieves results. Every SQL statement is passed through the cursor object. Without the cursor, Python cannot interact with the database tables. The cursor provides methods like **execute()**, **fetchone()**, and **fetchall()**.

Ex:-

```
cur = con.cursor()
```

4. Forming SQL Query

The next step is writing the SQL query that should be executed. SQL queries are written as strings inside the Python program. The query can be used to insert, update, delete, or display records. For example, a SELECT query is used to retrieve data from a table. The query must follow proper SQL syntax; otherwise an error occurs.

Ex:-

```
query = "SELECT * FROM student"
```

5. Passing Query to MySQL

Once the query is formed, it must be passed to the MySQL database. This is done using the **execute()** method of the cursor object. The execute method sends the SQL command to the MySQL server. MySQL then processes the query and returns the result. If the query is a SELECT statement, records can be fetched afterward.

Ex:- `cur.execute(query)`

6.Fetching Data from Database

If the SQL query is a SELECT statement, the data must be retrieved from the database. Python provides methods such as **fetchone()**, **fetchmany()**, and **fetchall()**. The **fetchall()** method retrieves all rows from the result set. Each row is returned as a tuple. The data can then be displayed or processed further. Fetching data is useful when generating reports or displaying information.

Ex:-

```
rows = cur.fetchall()
for row in rows:
    print(row)
```

7. Closing the Connection

The final step is closing the database connection after all operations are completed. Closing the connection releases system resources. It also prevents unnecessary memory usage. Keeping connections open for a long time may slow down the system. Therefore, it is good practice to close the connection properly. This ensures the program ends safely.

Ex:- con.close()

Complete Example:-

```
import mysql.connector

# Step 1: Establish connection
con = mysql.connector.connect(
    host="localhost",
    user="root",
    password="1234",
    database="college"
)

# Step 2: Create cursor object
cur = con.cursor()

# Step 3: Form SQL query
query = "SELECT * FROM student"

# Step 4: Execute query
cur.execute(query)

# Step 5: Fetch all records
rows = cur.fetchall()
```

```
# Step 6: Display records
for row in rows:
   print(row)

# Step 7: Close connection
con.close()
```

Q2).Explain how to insert, update, delete and retrieve data using python Mysql connectivity. (CRUD Operations)

CRUD stands for **Create, Read, Update, and Delete**. These are the basic operations performed on a database table. Using Python with MySQL connectivity, we can easily perform these operations through SQL queries. Python uses the **mysql.connector** module to communicate with the MySQL database. Each operation is performed by writing an SQL query and executing it through a cursor object.

1. Insert Operation (Create)

The **Insert operation** is used to add new records into a database table. In Python, an SQL INSERT query is written as a string and passed to the execute() method. After inserting data, the **commit()** method is used to save the changes permanently. Without commit(), the inserted record will not be stored in the database. This operation is useful when adding new users, employees, or student details. Insert is the first step in storing data into a database.

Ex:-

```
import mysql.connector
con = mysql.connector.connect(host="localhost", user="root", password="1234",
database="college")
cur = con.cursor()
query = "INSERT INTO student VALUES(104, 'Arun', 88)"
cur.execute(query)
con.commit()
print("Record inserted successfully")
con.close()
```

2. Retrieve Operation (Read)

The **Retrieve operation** is used to read data from the database table. It uses the SQL SELECT statement to fetch records. The query is executed using execute(), and data is collected using fetchone(), fetchmany(), or fetchall(). The retrieved records can be displayed or processed in the Python program. This operation is commonly used in reports and search functions. Read operation helps users view the stored information.

Example:

```
import mysql.connector

con = mysql.connector.connect(host="localhost", user="root", password="1234",
database="college")
cur = con.cursor()

query = "SELECT * FROM student"
cur.execute(query)

rows = cur.fetchall()

for row in rows:
   print(row)

con.close()
```

3. Update Operation

The **Update operation** modifies existing records in the database. It uses the SQL UPDATE statement. The query specifies which record should be changed and what new value should be assigned. After executing the query, commit() must be called to save changes. This operation is useful for changing marks, phone numbers, or addresses. Update ensures data remains accurate and current.

Example:

```
import mysql.connector

con = mysql.connector.connect(host="localhost", user="root", password="1234",
database="college")
cur = con.cursor()
query = "UPDATE student SET marks=95 WHERE id=104"
cur.execute(query)
con.commit()
print("Record updated successfully")
con.close()
```

4. Delete Operation

The **Delete operation** removes records from the table. It uses the SQL DELETE statement. The WHERE clause is important because it specifies which record must be deleted. If WHERE is omitted, all records may be deleted accidentally. After execution, commit() saves the changes. Delete is useful when removing unwanted or old records. This operation helps maintain clean database tables.

Example:

```
import mysql.connector

con = mysql.connector.connect(host="localhost", user="root", password="1234",
database="college")
cur = con.cursor()

query = "DELETE FROM student WHERE id=104"
cur.execute(query)

con.commit()
print("Record deleted successfully")

con.close()
```

Operation	SQL Command	Purpose
Create	INSERT	Add new records
Read	SELECT	Retrieve records
Update	UPDATE	Modify existing records
Delete	DELETE	Remove records

Q3: Explain commit() and rollback() methods used in database operations.

In database operations, Python communicates with databases such as MySQL to insert, update, or delete records. Whenever changes are made to the database, those changes are not permanently stored immediately. Python provides two important methods called **commit()** and **rollback()** to control database transactions.

1. commit() Method

The **commit()** method is used to save all changes made during the current transaction permanently into the database. Whenever INSERT, UPDATE, or DELETE operations are performed, the changes remain temporary until commit() is executed. If commit() is not used, the changes may be lost when the program ends. It ensures that the modified data is stored permanently in the table. This method is mainly used after successful execution of SQL statements.

Syntax

```
connection.commit()
```

Ex:-

```
import mysql.connector
```

```
con = mysql.connector.connect(
   host="localhost",
   user="root",
   password="1234",
   database="college"
)

cur = con.cursor()
query = "INSERT INTO student VALUES(105, 'Ramesh', 82)"
cur.execute(query)
con.commit()
print("Record inserted successfully")
con.close()
```

2. rollback() Method

The **rollback()** method is used to undo changes made during the current transaction. If an error occurs while executing SQL statements, rollback() returns the database to its previous state. It prevents incomplete or wrong data from being saved. This method is useful when multiple queries are executed together. If one query fails, rollback() can cancel all related changes. It helps maintain data integrity and avoids database corruption. Rollback is mainly used for error handling.

Syntax

```
connection.rollback()
```

Ex:-

```
import mysql.connector

con = mysql.connector.connect(
   host="localhost",
   user="root",
   password="1234",
   database="college"
)
cur = con.cursor()

try:
   query = "UPDATE student SET marks=90 WHERE id=105"
   cur.execute(query)

   con.commit()
```

```
except:
   con.rollback()
   print("Transaction cancelled")

con.close()
```

5 Marks

1. What is matplotlib? Write any four features
2. What is database connections ? why is it required ?
3. Explain commit() and rollback() methods used in database operations.
4. Explain Histogram with examples.
5. Explain forming and passing a query to Mysql using python.
6. what is Matplotlib? How to install matplotlib an import ?
7. explain how to insert color names in bar chart?
8. what is MySQL? Explain need of MySQL in Python.

10 Marks

1. Explain various charts available in matplotlib. Discuss line chart, bar chart, histogram, scatter chart and pie chart with suitable examples.

2. Explain database connectivites in python. Discuss the steps involved in importing Mysql, connecting to a database, forming a query and passing the query to Mysql.

3. Explain how data visualization helps in data analysis. Describe different types of plots with suitable examples.

4. Explain how to insert, update, delete and retrieve data using python Mysql connectivity.

5. Write a detailed note on Matplotlib library. Explain installation, importing and plotting methods.

B. Devendra Msc-CS
Dept of Computer Science & Applications
Shri Gnanambica Degree College(A).