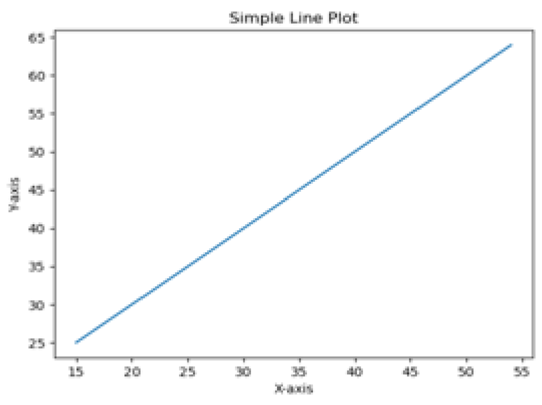
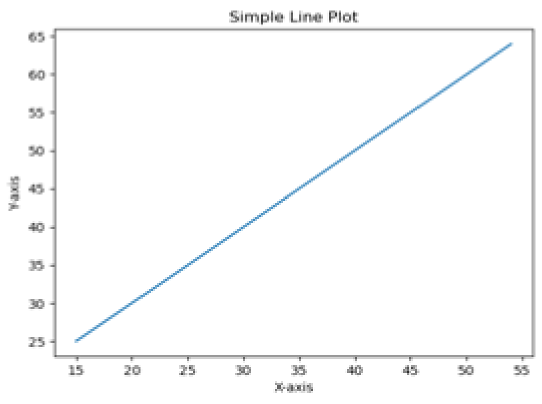
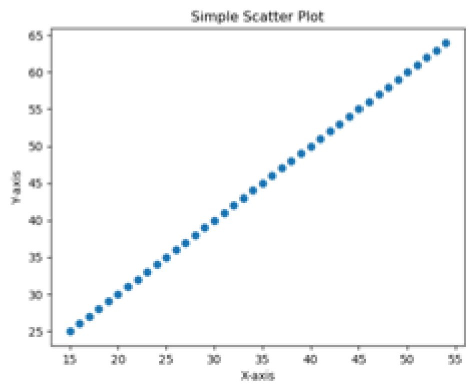
UNIT-5

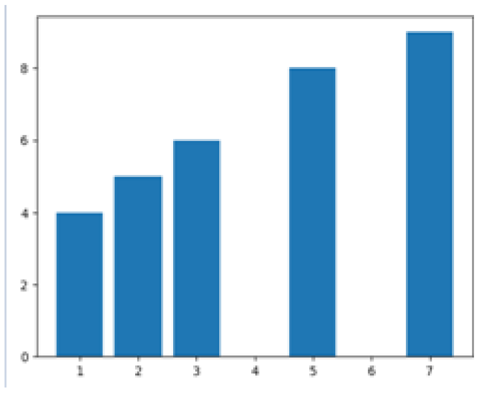
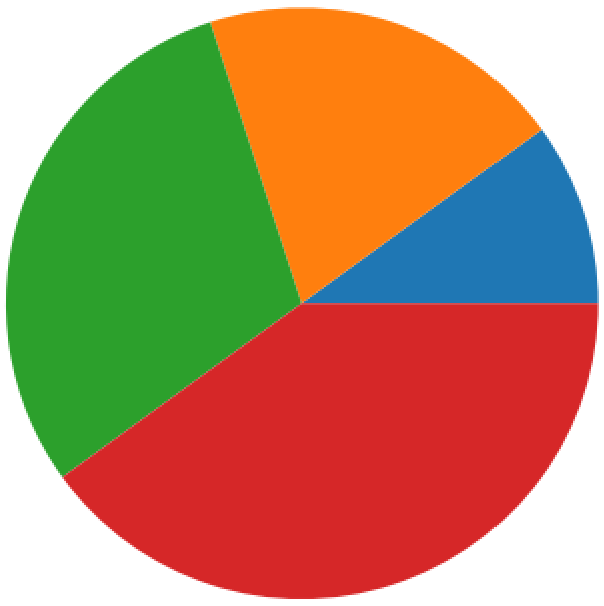
1. Introduction to Matplotlib

- ➢ Matplotlib is a powerful and very popular data visualization library in Python. To create line plots, bar plots, and scatter plots in Matplotlib using stock market data in 2022.
- ➢ These are the foundational plots that will allow you to start understanding, visualizing, and telling stories about data. Data visualization is an essential skill for all data analysts and Matplotlib is one of the most popular libraries for creating visualizations.
- ➢ Matplotlib is a plotting library used to create a wide variety of visualizations. It allows you to visualize data, making it easier to understand patterns, trends, and relationships in the data.
- ➢ Matplotlib is very flexible and customizable for creating plots. It does require a lot of code to make more basic plots with little customizations.

Key Features of Matplotlib:

- **Versatility**: You can plot line charts, scatter plots, bar charts, histograms, pie charts, and more.
- **Customizability**: It supports customizing almost every aspect of the plot (like labels, title, axes, etc.).
- **Integration**: Works seamlessly with other Python libraries like NumPy, Pandas, and Seaborn.
- **Simple Plotting** − Matplotlib allows us to create basic plots easily with just a few lines of code.
- **Customization** − we can extensively customize plots by adjusting colors, line styles, markers, labels, titles and more.
- **Multiple Plot Types** − It supports a wide variety of plot types such as line plots, scatter plots, bar charts, histograms, pie charts, 3D plots, etc.
- **Publication Quality** − Matplotlib produces high-quality plots suitable for publications and presentations with customizable DPI settings.
- **Support for Latex Typesetting** − we can use Latex for formatting text and mathematical expressions in plots.
- Matplotlib supports various types of plots which are as mentioned below. Each plot type has its own function in the library.

Name of the plot	Definition	Image
Line plot	A line plot is a type of graph that displays data points connected by straight line segments. The plt.plot() function of the matplotlib library is used to create the line plot.	
Scatter plot	A scatter plot is a type of graph that represents individual data points by displaying them as markers on a two-dimensional plane. The plt.scatter() function is used to plot the scatter plot.	
Line plot	A line plot is a type of graph that displays data points connected by straight line segments. The plt.plot() function of the matplotlib library is used to create	

	the line plot.	
Bar plot	A bar plot or bar chart is a visual representation of categorical data using rectangular bars. The plt.bar() function is used to plot the bar plot.	
Pie plot	A pie plot is also known as a pie chart. It is a circular statistical graphic used to illustrate numerical proportions. It divides a circle into sectors or slices to represent the relative sizes or percentages of categories within a dataset. The plt.pie() function is used to plot the pie chart.	

- The above mentioned are the basic plots of the matplotlib library. We can also visualize the 3-d plots with the help of Matplotlib.

You can install Matplotlib with the following command:

```
pip install matplotlib
```

Then, import it in your script using:

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

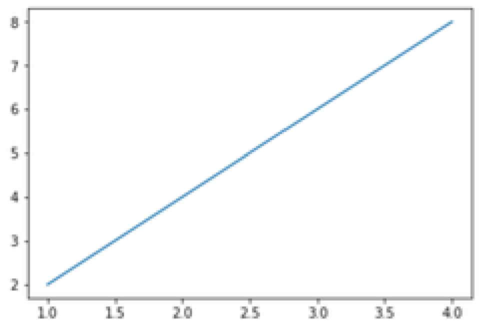
2. Line Chart

- A **line chart** is one of the most common visualizations used to display data trends over a period of time.
- a simple line chart is generated using NumPy to define data values. The x-values are evenly spaced points, and the y-values are calculated as twice the corresponding x-values.

Example:

```
import matplotlib.pyplot as plt

# Data for plotting
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Plotting a simple line chart
plt.plot(x, y)

# Adding title and labels
plt.title("Line Chart Example")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

# Display the plot
plt.show()
```

- **plt.plot(x, y)**: Plots a line chart where x is the list of values on the X-axis, and y is the list of values on the Y-axis.
- **plt.title()**, **plt.xlabel()**, **plt.ylabel()**: Add titles and labels to the plot.



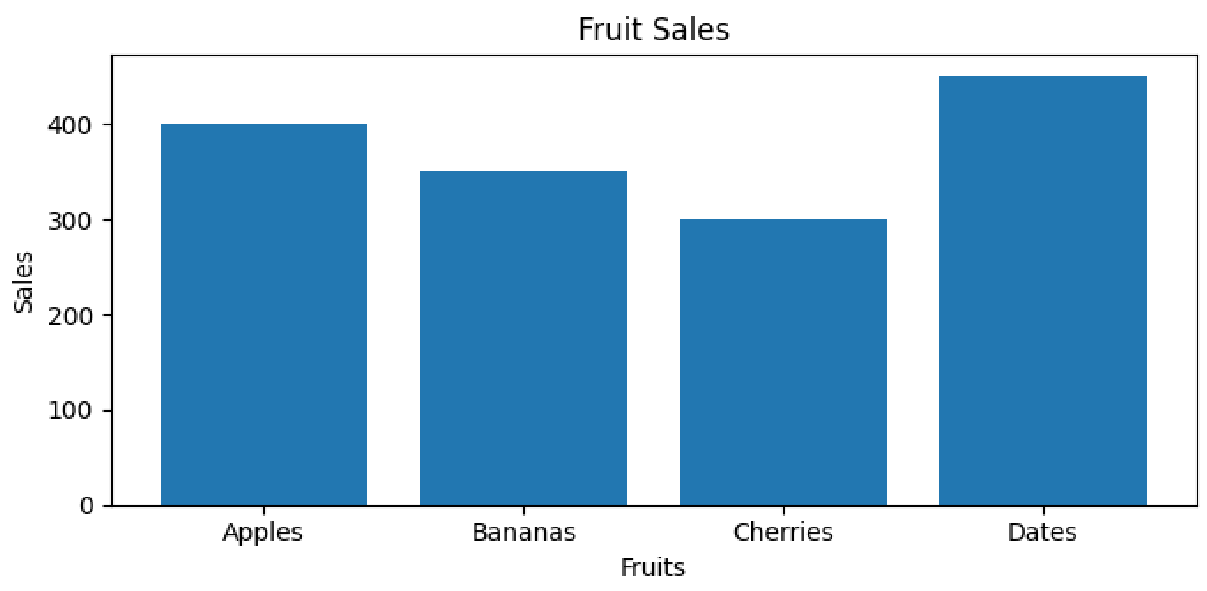
3. Bar Chart

- A **bar chart** is useful for comparing data between different categories. It is created using rectangular bars where the height of each bar is proportional to the value of the category.

- ➢ A bar plot uses rectangular bars to represent data categories, with bar length or height proportional to their values. It compares discrete categories, with one axis for categories and the other for values.

Example:

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

fruits = ['Apples', 'Bananas', 'Cherries', 'Dates']
sales = [400, 350, 300, 450]

plt.bar(fruits, sales)
plt.title('Fruit Sales')
plt.xlabel('Fruits')
plt.ylabel('Sales')
plt.show()
```



4. Histogram

- ➢ A **histogram** is used to represent the distribution of numerical data by splitting the data into bins (intervals) and counting the number of occurrences in each bin.
- ➢ A Histogram represents data provided in the form of some groups. It is an accurate method for the graphical representation of numerical data distribution.
- ➢ It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency.

Plotting Histogram in Python using Matplotlib

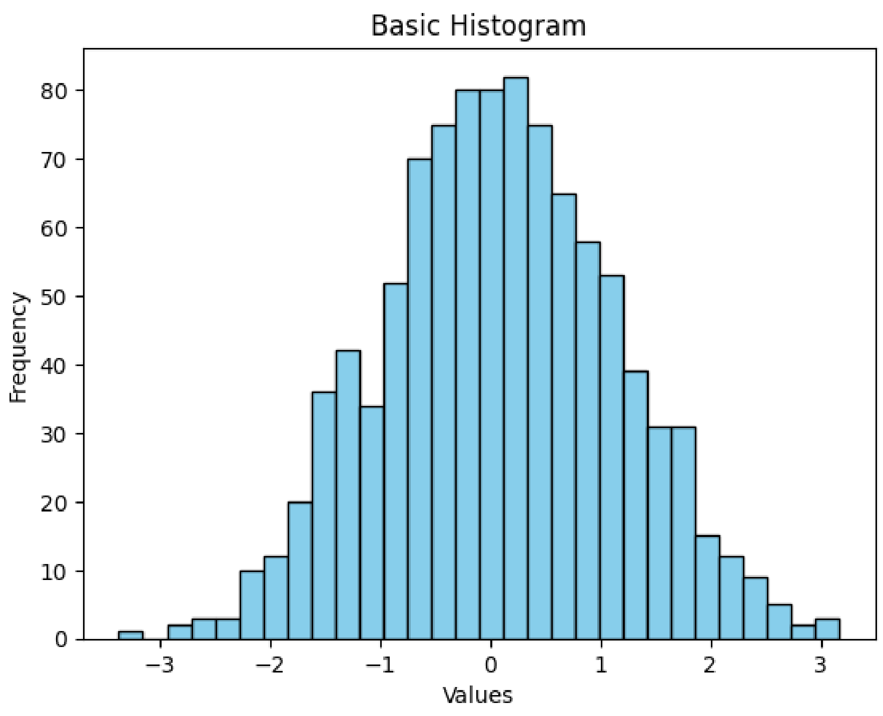
Here we will see different methods of Plotting Histogram in Matplotlib in Python:

- Basic Histogram
- Customized Histogram with Density Plot
- Customized Histogram with Watermark
- Multiple Histograms with Subplots

- Stacked Histogram
- 2D Histogram (Hexbin Plot)

```
import matplotlib.pyplot as plt

import numpy as np



# Generate random data for the histogram

data = np.random.randn(1000)



# Plotting a basic histogram

plt.hist(data, bins=30, color='skyblue', edgecolor='black')

 # Adding labels and title

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.title('Basic Histogram')



# Display the plot

plt.show()
```

- **Output:**



-
- **plt.hist()**: This function creates a histogram. The data is the list of values, and bins specifies the number of intervals to divide the data into.

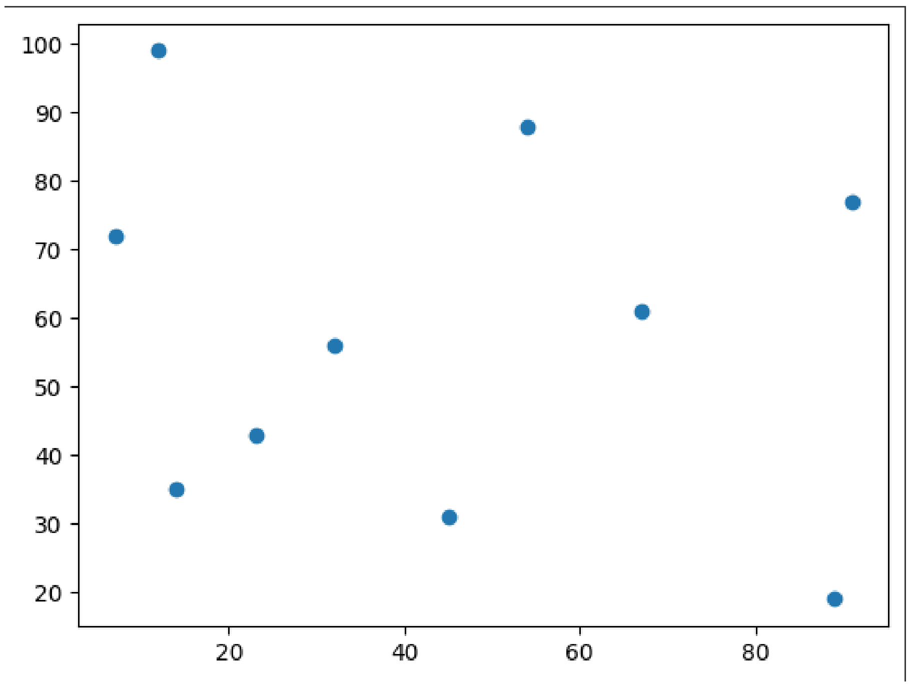
---

5. Scatter Plot

- A **scatter plot** is used to display the relationship between two continuous variables. It plots points on the graph as individual dots and is useful for identifying correlations between variables.
- **matplotlib.pyplot.scatter()** is used to create scatter plots, which are essential for visualizing relationships between numerical variables. Scatter plots help illustrate how changes in one variable can influence another, making them invaluable for data analysis.
- A basic scatter plot can be created using matplotlib.pyplot.scatter() by plotting two sets of data points on the x and y axes:

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

x = np.array([12, 45, 7, 32, 89, 54, 23, 67, 14, 91])
y = np.array([99, 31, 72, 56, 19, 88, 43, 61, 35, 77])

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

Output:



- **plt.scatter(x, y)**: This function creates a scatter plot where x and y are lists of values to be plotted on the X and Y axes, respectively.

---

6. Pie Chart

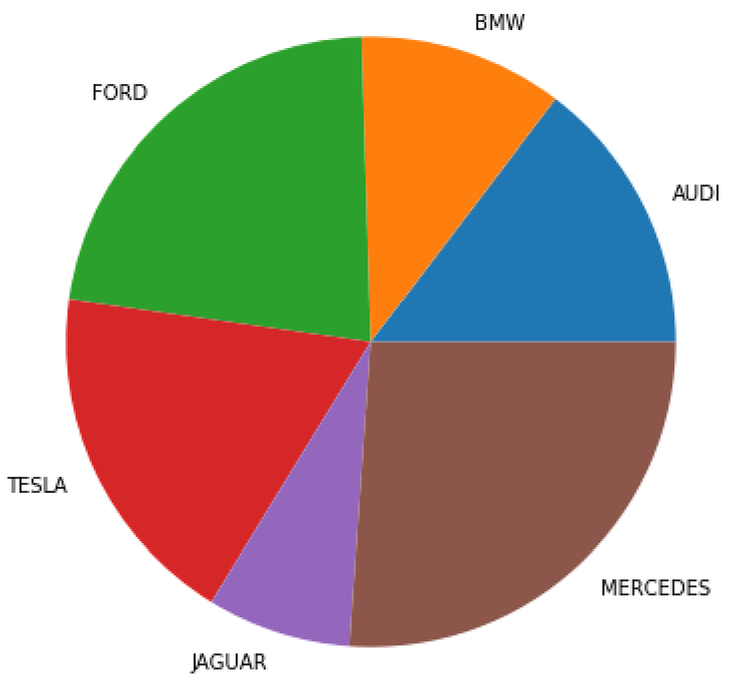
- A **pie chart** is used to represent parts of a whole. Each slice of the pie represents a category's contribution to the total.
- A **Pie Chart** is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data.
- **Pie charts in Python** are widely used in business presentations, reports, and dashboards due to their simplicity and effectiveness in displaying data distributions.
- how to create a **pie chart in Python** using the **Matplotlib** library, one of the most widely used libraries for data visualization in Python.

Example:

```
# Import libraries
from matplotlib import pyplot as plt
import numpy as np


# Creating dataset
cars = ['AUDI', 'BMW', 'FORD',
        'TESLA', 'JAGUAR', 'MERCEDES']

data = [23, 17, 35, 29, 12, 41]
```

```
# Creating plot
fig = plt.figure(figsize=(10, 7))
plt.pie(data, labels=cars)

# show plot
plt.show()
```

Output:



- **plt.pie()**: This function creates a pie chart. The sizes represent the data, labels are the categories, colors define the color of each slice, and autopct is used to display percentage values.
- **startangle=140**: Rotates the start angle of the pie chart for better visualization.

Summary of Different Plots

Plot Type	Purpose	Function to Create
Line Chart	Displays trends or relationships over time.	plt.plot()
Bar Chart	Compares quantities of different categories.	plt.bar()
Histogram	Shows the distribution of a dataset.	plt.hist()
Scatter Plot	Displays relationships between two continuous variables.	plt.scatter()
Pie Chart	Shows parts of a whole (percentage-based).	plt.pie()

Database Connectivity with MySQL in Python

- A database is basically a collection of structured data in such a way that it can easily be retrieved, managed and accessed in various ways.
- One of the simplest forms of databases is a text database. Relational databases are the most popular database system which includes the following:

- MySQL
- Oracle Database
- SQL server
- Sybase
- Informix
- IBM db2
- NO SQL

Among all these databases, **MySQL** is one of the easiest databases to work with. Let me walk you through about this in detail.

What is MySQLdb?

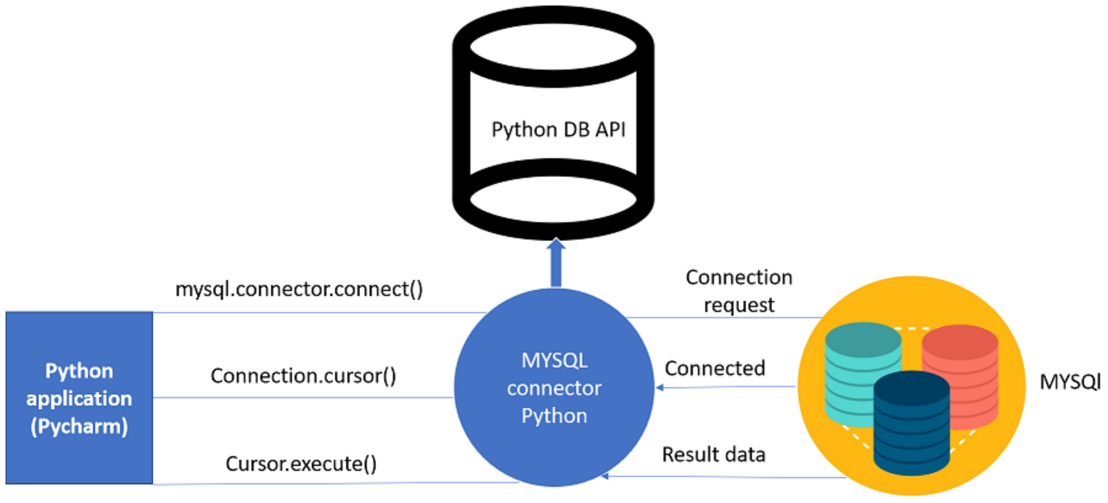
- MySQLdb is an open-source freely available relational database management system that uses Structured Query Language. Now one of the most important question here is "What is SQL?"
- SQL (Structured Query Language) is a standard language for relational databases that allow users to do various operations on data like, Manipulating, Creating, Dropping, etc. In a nutshell, SQL allows you to do anything with the data.
- Let's move ahead and dive deep into Python database connection wherein you will learn how to connect with the database.
- Python provides several libraries to connect with databases and execute queries. One of the most commonly used libraries for connecting to MySQL is mysql-connector-python.
- This allows Python applications to interact with MySQL databases for performing operations like querying, inserting, updating, and deleting data.

How does Python connect to a database?

- It is very simple to connect Python with the database. Refer the below image which illustrates a Python connection with the database where how a connection request is sent to MySQL connector Python, gets accepted from the database and cursor is executed with result data.
- Before connecting to the MySQL database, make sure you have MySQL installer installed on your computer. It provides a comprehensive set of tools which helps in installing MySQL with the following components:

    - MySQL server
    - All available connectors
    - MySQL Workbench
    - MySQL Notifier
    - Tools for Excel and Microsoft Visual Studio
    - MySQL Sample Databases
    - MySQL Documentation

- To download the MySQL installer please go through the following video which talks about the various steps that you need to follow while installing MySQL.



1. Importing MySQL Connector for Python:

Before you can interact with a MySQL database in Python, you need to install the mysql-connector-python library. You can install this package using pip:

pip install mysql-connector-python

After installing the connector, you can import it in your Python code:

import mysql.connector

2. Connecting with a MySQL Database

To interact with a MySQL database, you need to establish a connection. You can use the mysql.connector.connect() method to connect to the database.

Steps for connecting to a MySQL database:

1. **Database**: The name of the database to connect to.
2. **Username-** It is simply the username you give to work MySQL server with, the Default username is *root.*
3. **Password-** Password is given by the user when you have installed the MySQL database. I am giving password here as ‘password123’
4. **Host Name-** This basically is the server name or IP address on which your MySQL is running, If it is a ‘localhost’, then your IP address is 127.0.0.0

Example:

```
import mysql.connector

# Establishing the connection
conn = mysql.connector.connect(
    host="localhost",  # MySQL server host
    user="root",       # MySQL username
    password="password123",  # MySQL password
    database="mydatabase"   # Name of the database
)

# Check if the connection is successful
if conn.is_connected():
print("Connected to the database.")
else:
print("Connection failed.")

# Don't forget to close the connection after operations
conn.close()
```

- **conn.is_connected ()**: This checks if the connection to the MySQL database is successful.

3. Forming a Query in MySQL:

- Once you're connected to a MySQL database, you can form a query. Queries in MySQL are written in **SQL (Structured Query Language)**.

Common SQL Queries:

- **SELECT**: Retrieve data from a table.
- **INSERT**: Insert new data into a table.
- **UPDATE**: Update existing data in a table.
- **DELETE**: Delete data from a table.

Example:

```
# Forming a SELECT query to fetch all records from a table
query = "SELECT * FROM employees;"
```

You can use the same approach for other SQL operations, such as:

```
# Inserting data
query = "INSERT INTO employees (name, age, department) VALUES ('John Doe', 29,
'HR');"

# Updating data
query = "UPDATE employees SET age = 30 WHERE name = 'John Doe';"
```

```
# Deleting data
query = "DELETE FROM employees WHERE name = 'John Doe';"
```

4. Passing a Query to MySQL:

Once you have formed a query, you need to pass the query to MySQL and execute it. This is done using the cursor object. The cursor is used to execute SQL queries and retrieve results.

Steps for Executing a Query:

1. **Create a Cursor**: A cursor is an object used to execute SQL queries.
2. **Execute the Query**: Use cursor.execute() to run the query.
3. **Fetch Results**: If the query retrieves data (e.g., SELECT), use cursor.fetchall() or cursor.fetchone() to retrieve the results.

Example:

Fetching Data (SELECT Query):

```
import mysql.connector


# Establishing the connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password123",
    database="mydatabase"
)

# Creating a cursor object to interact with the database
cursor = conn.cursor()

# Forming the SELECT query
query = "SELECT * FROM employees;"

# Executing the query
cursor.execute(query)

# Fetching all results
results = cursor.fetchall()

# Display the results
for row in results:
    print(row)

# Closing the cursor and connection
cursor.close()
conn.close()
```

- **cursor.execute(query)**: Executes the query.
- **cursor.fetchall()**: Fetches all rows from the result set. You can also use cursor.fetchone() to fetch one row at a time.

Inserting Data (INSERT Query):

```
import mysql.connector

# Establishing the connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password123",
    database="mydatabase"
)
# Creating a cursor object
cursor = conn.cursor()

# Forming the INSERT query
query = "INSERT INTO employees (name, age, department) VALUES ('John Doe', 29,
'HR');"

# Executing the query
cursor.execute(query)

# Committing the transaction (important for INSERT, UPDATE, DELETE)
conn.commit()

# Closing the cursor and connection
cursor.close()
conn.close()

print("Record inserted successfully.")
```

- **conn.commit()**: Commits the transaction. This is necessary for operations like INSERT, UPDATE, and DELETE.

Updating Data (UPDATE Query):

```
import mysql.connector

# Establishing the connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password123",
    database="mydatabase"
)
```

```
# Creating a cursor object
cursor = conn.cursor()

# Forming the UPDATE query
query = "UPDATE employees SET age = 30 WHERE name = 'John Doe';"

# Executing the query
cursor.execute(query)

# Committing the transaction
conn.commit()

# Closing the cursor and connection
cursor.close()
conn.close()

print("Record updated successfully.")
```

Deleting Data (DELETE Query):

```
import mysql.connector

# Establishing the connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password123",
    database="mydatabase"
)

# Creating a cursor object
cursor = conn.cursor()

# Forming the DELETE query
query = "DELETE FROM employees WHERE name = 'John Doe';"

# Executing the query
cursor.execute(query)

# Committing the transaction
conn.commit()

# Closing the cursor and connection
cursor.close()
conn.close()

print("Record deleted successfully.")
```

Summary of Key Functions:

Function	Description
mysql.connector.connect()	Establishes a connection to the MySQL database.
conn.cursor()	Creates a cursor object to interact with the database.
cursor.execute(query)	Executes the SQL query.
cursor.fetchall()	Fetches all rows from the result set (for SELECT queries).
cursor.fetchone()	Fetches one row from the result set.
conn.commit()	Commits the transaction (necessary for INSERT, UPDATE, DELETE).
cursor.close()	Closes the cursor object after the operation.
conn.close()	Closes the connection to the database.