Practical No 4

A. Abstract Method

This code defines an abstract base class `Animal` with a method `sound()` that must be implemented by its subclasses. The `Dog` class inherits from `Animal` and provides its own implementation of `sound()`, which returns the string "Bark". An instance of `Dog` is created and its `sound()` method is called, printing "Bark".

B. Interface

The `Area` class is an interface that defines a contract for shapes to implement the `findArea` method. This interface specifies that any shape that wants to calculate its area must provide an implementation for the `findArea` method. The `Triangle` and `Rectangle` classes implement this interface by providing their own specific implementations of the `findArea` method, which allows them to calculate their respective areas.

Practical No 5

A. Single Thread

This code demonstrates the difference between running two tasks sequentially (without threading) and concurrently (using threading).

In the "Without Using Threading" section, two objects `obj1` and `obj2` are created, and their `run` methods are called sequentially. This means that `obj1.run()` will finish printing "John Doe" 5 times before `obj2.run()` starts printing "Jane Doe".

In the "Using Threading" section, two threads `objT1` and `objT2` are created, and their `run` methods are called. Since they are threads, they can run concurrently, meaning that "John Doe (Thread)" and "Jane Doe (Thread)" will be printed alternately, rather than one after the other.

B. Multitasking Thread

This Python code snippet demonstrates multithreading. It creates two threads, `t1` and `t2`, which run concurrently, printing "John Doe " and "Jane doe " respectively, 5 times each, with a 1-second delay between each print. The main thread waits for both threads to finish before printing "I Am Main Thread".

C. Daemon Thread

In this code snippet, the `thread1` function is run in a non-daemon thread because the `daemon` attribute of the `Thread` object `T` is not set to `True`. This means that when the main thread finishes executing (i.e., after printing "Main Thread Execution"), the program will wait for the `thread1` function to finish executing before exiting.

Practical No 6

A. MySQL Database Connection in python

This code snippet connects to a MySQL database on the local machine (`localhost`) using the `mysql.connector` library in Python. It uses the username `root` and password `root` for authentication, and then prints the connection object `mydb` to the console.

B. Inserting Data In Database

This code snippet is written in Python and uses the `mysql.connector` library to connect to a MySQL database.

Here's what the code does:

1. It imports the `mysql.connector` library.

2. It establishes a connection to the MySQL database using the `connect()` method of the `mysql.connector` library. The connection details are provided as keyword arguments to the `connect()` method.

3. It creates a cursor object using the `cursor()` method of the connection object.

4. It executes three SQL `INSERT` statements using the `execute()` method of the cursor object. These statements insert data into the `Employee` table of the `Company` database.

5. It prints the number of records affected by each `INSERT` statement using the `rowcount` attribute of the cursor object.

6. It commits the changes made to the database using the `commit()` method of the connection object.

In summary, this code snippet connects to a MySQL database, inserts data into the `Employee` table, and then prints the number of records affected by each `INSERT` statement.

C. Updating Data In Database

This code snippet is using the `mysql.connector` library to connect to a MySQL database named "Company" on the local host. It then creates a cursor object to execute SQL queries. The code updates the "Name" field of the "Employee" table where the "EmpID" is equal to 11, setting the name to "Iron Man". After executing the update query, it prints the number of records affected by the update. Finally, it commits the changes to the database.

D. Delete Data In Database

This Python code snippet connects to a local MySQL database named "Company" and deletes the record with `EmpID` equal to 11 from the `Employee` table. It then prints the number of records affected by the deletion and commits the changes to the database.

E. Fetching Data From Database

This code snippet is using the `mysql.connector` library to connect to a MySQL database named "Company" on the local machine. It then creates a cursor object to interact with the database. The code executes a SQL query to select all rows from a table named "Employee".

The commented out code is an example of how to fetch data from the result set. `cursor.fetchone()` fetches the next row from the result set, `cursor.fetchmany(n)` fetches the next n rows from the result set, and `cursor.fetchall()` fetches all rows from the result set.

In this code snippet, the `cursor.fetchall()` method is used to fetch all rows from the result set and the fetched data is then printed.