Question 1:

Write a Python program that uses the Command pattern. Suppose you have a Light class with methods on() and off() and a Fan class with methods start() and stop().

- Define a Command interface with a method execute().
- Make two classes that implement the Command interface: LightOnCommand and LightOffCommand. These classes should take a Light object as an argument in their __init__() method and call the on() and off() methods in their execute() method.
- Similarly, make two classes that implement the Command interface: FanStartCommand and FanStopCommand. These classes should take a Fan object as an argument in their __init__() method and call the start() and stop() methods in their execute() method.
- Next, make a RemoteControl class that has four attributes for commands: lightOnCommand, lightOffCommand, fanStartCommand, and fanStopCommand. The RemoteControl should have a method setCommand() that assigns the lightOnCommand, lightOffCommand, fanStartCommand, and fanStopCommand attributes.
- It should also have methods lightOnButtonPressed(), lightOffButtonPressed(), fanStartButtonPressed(), and fanStopButtonPressed() that call the execute() method on the corresponding command.

Question 2:

Write a Python program that uses the Command pattern. Suppose you have a Database class with methods for insert(), update(), and delete() operations.

- Define a Command interface with an execute() method.
- Make three classes that implement the Command interface: InsertCommand, UpdateCommand, and DeleteCommand. These classes should take a Database object as an argument in their constructor and invoke the insert(), update(), and delete() methods respectively.
- Lastly, make a Client class with a setCommand() method that assigns the command slot. It should also have an executeCommand() method that calls the execute() method on the relevant command.

Question 3:

You want to build a database client application that works with SQLite databases. The application should let users do different things with the data, such as create, read, update, and delete. To make the application more flexible and easy to maintain, you decide to use the Command pattern. This pattern lets you wrap each database operation as a separate command object.

To apply the Command pattern, you need to do the following steps:

- Think of the different database operations that the users might want to do, such as creating a new table, inserting a new record, retrieving a specific record, updating an existing record, or deleting a record.
- For each database operation, make a command class that contains the logic for that operation. Use the sqlite3 module to connect and communicate with the SQLite database.
- Make an invoker class that acts as the bridge between the users and the command objects. The invoker should let users choose the operation they want, enter any required parameters, and run the corresponding command.

- Make a receiver class that represents the SQLite database connection. The receiver should have methods for doing the actual database operations, such as running SQL queries or changing data structures.
- Show how the Command pattern works by creating and running commands for different database operations. Explain how the pattern separates the user interface from the database operations and makes it easier to add new operations.

```
import sqlite3

class Database:
    def __init__(self, db_file):
        self.conn = sqlite3.connect(db_file)

    def execute(self, query, params=None):
        if params is None:
            params = []
        cursor = self.conn.cursor()
        cursor.execute(query, params)
        self.conn.commit()
        return cursor.fetchall()
```

Question 4:

Use the Template Method pattern to design and code a waiter's process for taking and serving orders. The order process should have the following steps:

- Greet the customer
- Take the customer's order
- Prepare the customer's order
- Serve the customer their dish

The waiter should follow a different order of steps depending on the order type. For example, a dine-in order should be delivered to the customer's table, while a take-out order should be wrapped and given to the customer at the counter.

The Template Method pattern will help to keep the code neat and modular, and it will make it easy to add new order types in the future.