**Overview of** **Object-Oriented Paradigm in Python game**

**Encapsulation:**

This **Board** class encapsulates all the data and functionality related to the game board, like storing the state and handling updates. It hides the internal representation of the board and provides methods to interact with it. It also encapsulates the logic for game progression, turn-taking, and interaction with the **Board** class.

**Data Abstraction and Representation:**

Data abstraction is a crucial aspect of this approach. The Board class abstracts how the board is represented and manipulated within the game. This abstraction allows other parts of the code to engage with the board through a simplified interface, exemplified by methods like **display**, **winner**, and **add**. Such an interface streamlines interactions, making them more intuitive and less dependent on the underlying complexities.

**Modularity in the Game Class:**

The Game class illustrates the principle of modularity. It segregates the responsibilities of managing the game's flow from those concerning the board's representation and player interactions. This separation of concerns simplifies the overall structure and makes each component's role more distinct and manageable.

**Object Interaction Through the Main Class:**

The main class in the code demonstrates how different objects, such as the board and game, interact to deliver the application's complete functionality. It uses instances of these classes to guide the application's control flow, orchestrating the interactions and processes needed to run the game.

**Structure and Organisation:**

The overall structure and organisation of the code are primarily class-based, with each class assigned specific responsibilities. This organisation enhances clarity and manageability, making it easier to navigate and understand the various components of the game.

**Reusability and Scalability:**

The modular nature of classes like **Board** and **Game** allows for easy modification, extension, and reuse in different contexts.

**Maintenance Debugging:**

Encapsulation and modularity make the code more maintainable and accessible to debug, as each part can be tested and modified independently.