**Errors and Drawbacks of todo\_list.py Addressed in Todo\_listUpdated.py**

1. **Global State Management**:
   * **Error/Drawback**: In todo\_list.py, the tasks list is defined in the global scope. This can lead to unintended side effects if other parts of the program modify the global state.
   * **Modification**: In Todo\_listUpdated.py, the tasks list is encapsulated within the TodoList class, avoiding potential conflicts and making the state management more robust.
2. **Lack of Encapsulation**:
   * **Error/Drawback**: Functions and data are not encapsulated, leading to less modular code that is harder to maintain and extend.
   * **Modification**: By using a class (TodoList), the updated version encapsulates both data (tasks) and behavior (methods for task operations), enhancing modularity and maintainability.
3. **Scalability and Reusability**:
   * **Error/Drawback**: The procedural approach in todo\_list.py is less scalable and reusable. For example, creating multiple to-do lists would require significant changes.
   * **Modification**: The class-based structure in Todo\_listUpdated.py allows for creating multiple instances of TodoList, each with its own state, improving scalability and reusability.
4. **Code Organization and Readability**:
   * **Error/Drawback**: Mixing global variables with functions can make the code harder to follow and understand, especially as the codebase grows.
   * **Modification**: Organizing the code into a class improves readability by logically grouping related functionality together.
5. **Error Handling and Validation**:
   * **Error/Drawback**: The procedural code in todo\_list.py does not handle invalid input robustly within a cohesive structure, making error handling less centralized.
   * **Modification**: By using methods within a class, Todo\_listUpdated.py centralizes error handling and validation, making the code easier to manage and extend.

**Detailed Analysis of todo\_list.py**

**Global State Management**

* **Drawback**: The tasks list is globally defined, meaning it can be accessed and modified from anywhere in the code, leading to potential side effects and bugs.

**Lack of Encapsulation**

* **Drawback**: Functions like display\_tasks, add\_task, mark\_completed, and remove\_task operate on a global tasks list, making it difficult to isolate and test individual components.

**Scalability and Reusability**

* **Drawback**: The procedural approach is less flexible. For example, if a user wants to manage multiple to-do lists, the code would need substantial modifications.

**Code Organization and Readability**

* **Drawback**: Mixing global variables with functions makes the codebase less organized, which can become problematic as the program grows in size and complexity.

**Conclusion**

The procedural approach in todo\_list.py introduces several drawbacks, including potential issues with global state management, lack of encapsulation, reduced scalability and reusability, and less organized code. These issues are effectively addressed in Todo\_listUpdated.py by refactoring the code into an object-oriented structure. This improves modularity, maintainability, scalability, and overall code organization.