Información General del Proyecto
El proyecto "Design-Patterns-In-Python" es un repositorio con fines didácticos y educativos para la implementación de diseño, ayudando a los programadores y desarrolladores a entender como estos patrones pueden ser aplicados en Python3. El diseño tiene una estructura general que incluye clases y ejemplos individuales y específicos para cada patrón, mostrando su implementación y uso práctico. Uno de los retos que tiene el diseño es como adaptar los patrones que a menudo se conceptualizan en lenguajes de tipo estático, a un lenguaje dinámico como Python, manteniendo la legibilidad y eficiencia del código.
https://github.com/Sean-Bradley/Design-Patterns-In-Python/tree/master/singleton

Información y estructura del fragmento del proyecto donde aparece el patrón
En el repositorio, el Singleton se encuentra en un módulo que encapsula la clase Singleton. Además de la clase que implementa el patrón, el módulo incluye ejemplos de uso que demuestran la creación de instancias y la imposibilidad de crear instancias adicionales, lo que asegura una única instancia de la clase en todo el programa.

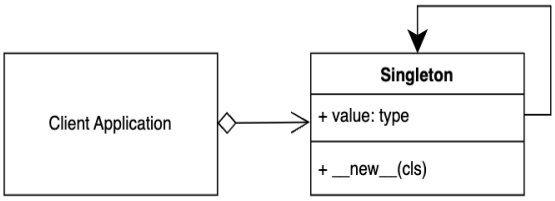
Información general sobre el patrón
El patrón Singleton restringe y limita la creación de una clase a un solo objeto. Se usa generalmente para gestionar recursos compartidos, como conexiones entre base de datos o la configuración de un sistema, que por naturaleza no debe estar duplicado.

Información del patrón aplicado al proyecto
Dentro del proyecto, el Singleton garantiza que solo se pueda instanciar una clase una vez. Esto se implementa a través de un método estático que crea una nueva instancia solo si aún no existe, y posteriormente devuelve la instancia existente en todas las llamadas posteriores.

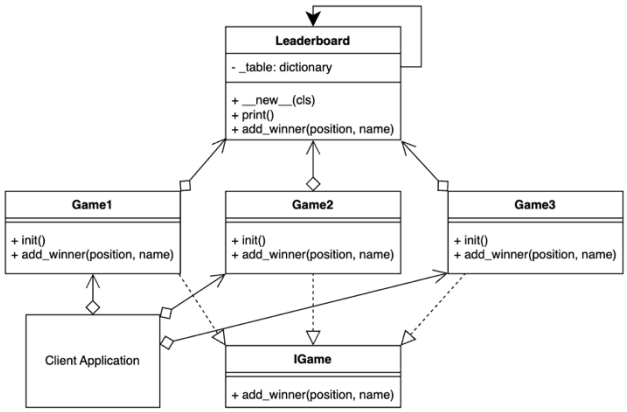
¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto?
La ventaja que hay es que existe un control estricto y controlado con respecto a como y cuando se accede a cierto recurso compartido, esto es útil ya que puede prevenir conflictos al acceder a recursos o una sobrecarga de un sistema durante múltiples instancias.

¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?
El patrón Singleton puede causar problemas a la hora de hacer pruebas ya que tiene un estado global y esto puede causar problemas si queremos escalar el programa o cuando se hagan pruebas que asumen instancias independientes.

¿De qué otras formas se le ocurre que se podrían haber solucionado los problemas que resuelve el patrón?
También se podría usar un “contenedor de inyección de dependencias” que pueda gestionar las instancias únicas de las clases, esto puede facilitar hacer pruebas y que el sistema sea más modular, lo que ayuda también en la escalabilidad.



El diagrama UML muestra el patrón de diseño Singleton. Se muestran dos componentes: una aplicación cliente y el Singleton. La aplicación cliente por naturaleza es dependiente del Singleton, esto se indica por la línea con un diamante en el extremo de la aplicación cliente. El Singleton tiene una propiedad 'value' y un método especial de clase __new__. El uso del método __new__ sugiere que la creación de instancias de la clase Singleton está controlada para asegurar que solo exista una instancia de la clase, la cual es accesible por la aplicación cliente. Este diseño garantiza y asegura que todas las solicitudes a la clase Singleton devuelvan la misma instancia, lo que es característico del patrón Singleton.



El diagrama UML representa una estructura de clases para un sistema de juegos con un tablero de líderes implementado como un Singleton, este sistema sirve como un ejemplo de implementación para Singleton. La clase `Leaderboard` tiene una propiedad privada `_table` que es un diccionario, y métodos públicos `__new__`, `print` y `add_winner`. El método `__new__` sugiere que `Leaderboard` es un Singleton. Hay tres clases de juegos (`Game1`, `Game2`, `Game3`) heredan de una interfaz `IGame`, indicada por la línea de puntos. La `Client Application` interactúa con las clases de juegos, las cuales a su vez interactúan con `Leaderboard`. Esto permite un tablero de líderes centralizado para todos los juegos. La relación de composición entre `Client Application` y `Game1` sugiere que `Client Application` crea y gestiona la instancia de `Game1`.