TP sur les anti-patterns : tester un Singleton en Python

Objectif

Ce TP vise à mettre en évidence les défis associés au test d'un Singleton, en illustrant comment les tests peuvent interférer les uns avec les autres lorsqu'un Singleton partage un état global.

Contexte

Le pattern Singleton garantit qu'une classe n'a qu'une seule instance tout au long du cycle de vie de l'application. Cela peut poser des difficultés lors de l'écriture de tests unitaires indépendants.

Instructions

1. Examiner le code Singleton suivant en Python
2. Ecrit 3 tests unitaires comme suggérés dans l'image

singleton.py

```python
class SingletonMeta(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            instance = super().__call__(*args, **kwargs)
            cls._instances[cls] = instance
        return cls._instances[cls]


class Singleton(metaclass=SingletonMeta):
    def __init__(self, value: int = 0) -> None:
        self._value = value

    def get_value(self) -> int:
        return self._value

    def set_value(self, value) -> None:
        self._value = value
```

test_singleton.py

```python
from singleton import Singleton


def test_singleton_default_value():
    # On initialise le singleton sans paramètre
    ...
    # On vérifie que la valeur par défaut est 0
    ...

def test_singleton_set_value():
    # On initialise le singleton sans paramètre
    ...
    # On change la valeur du singleton
    ...
    # On vérifie que la valeur a changé
    ...

def test_singleton_init_value():
    # On initialise le singleton avec 42
    ...
    # On vérifie que la valeur est 42
    ...
```

3. Réflexion et conclusion
   - Que se passe-t-il, pourquoi ?

   - Comment rendre ce Singleton plus testable ?