#code #python

Tests

Un test consiste à appeler la fonctionnalité avec un scénario qui correspond à un cas d'utilisation, et à vérifier que cette fonctionnalité se comporte comme prévu.

Les tests unitaires : éléments fondamentaux

Les tests unitaires se concentrent sur de petites portions de code pour garantir leur bon fonctionnement. Ils doivent être :

- **Basés sur des spécifications précises** des méthodes ou fonctions.
- Conçus pour **couvrir les préconditions et postconditions** (spécifications PRE/POST).
- Automatisés pour assurer une exécution rapide et fiable.

Concepts clés :

- **Choix des valeurs de test** : Identifier des cas représentatifs et des cas limites.
- **Utilisation des assertions (`assert`)** : Vérifier que les résultats des fonctions correspondent aux attentes.
- **Couverture des tests** : Mesurer la proportion du code testé par les cas.

Le Test Driven Development (TDD)

Le TDD, ou développement piloté par les tests, repose sur une méthodologie itérative où les tests sont rédigés avant l'implémentation du code. Ce processus suit plusieurs étapes :

1. **Écrire les spécifications des fonctions/méthodes** (PRE/POST) sur la base d'un diagramme UML et du cahier des charges.
2. **Choisir des ensembles de valeurs de tests** représentatives.
3. **Écrire des tests qui échouent initialement**, car le code n'est pas encore implémenté.
4. **Implémenter juste assez de code** pour que les tests passent.
5. **Refactorer le code** pour l'optimiser, tout en maintenant les tests validés.

La couverture du code (Code Coverage)

Un indicateur clé pour évaluer l'efficacité des tests est le **Code Coverage**, qui mesure le pourcentage de code exécuté par les tests. Un bon score de couverture indique que la majorité des instructions sont validées.

Cas pratique

```python
import datetime

class Student:
    """
    This class collects all personnal information about a student, as well as the history of his results.
    Author V. Van den Schrieck
    """

    def __init__(self, firstname: str, lastname: str, birthdate: datetime):
        """
        :pre: -
        :post: A new Student is created
        :raises: ValueError
                    - if firstname or lastname is empty
                    - if birthdate is less than 15 years in the past
        """
        if firstname == "" or lastname == "":
            raise ValueError
        self._firstname = firstname
        self._lastname = lastname
        self._birthdate = birthdate
```

```python
from unittest import TestCase
from datetime import datetime
from student import Student

class TestStudent(TestCase):

    def test_init(self):
        stud1 = Student("Toto", "Tutu", datetime(2000, 1, 1))
        self.assertEqual(stud1._firstname, "Toto", "Normal test : firstname")
        self.assertEqual(stud1._lastname, "Tutu", "Normal test : lastname")
        self.assertEqual(stud1._birthdate, datetime(2000,1,1), "Normal test : birthdate")

        stud2 = Student("Jean Charles", "Tutu", datetime(2000, 1, 1))
        self.assertEqual(stud2._firstname, "Jean Charles", "firstname with space")

        self.assertRaises(ValueError, Student, "", "Tutu", datetime(2021, 1, 1))
```

Méthode `test_init`

Cette méthode teste le constructeur (`__init__`) de la classe `Student` en vérifiant plusieurs cas :

Cas 1 : Initialisation normale

```
stud1 = Student("Toto", "Tutu", datetime(2000, 1, 1))
self.assertEqual(stud1._firstname, "Toto", "Normal test : firstname")
self.assertEqual(stud1._lastname, "Tutu", "Normal test : lastname")
self.assertEqual(stud1._birthdate, datetime(2000, 1, 1), "Normal test :
birthdate")
```

- **`assertEqual`** vérifie que la valeur réelle des attributs (`_firstname` , `_lastname` , `_birthdate`) correspond aux valeurs attendues.
- Si les valeurs diffèrent, le test échoue, et le message d'erreur ("Normal test : ...") est affiché.

Cas 2 : Gestion des prénoms avec espace

```
stud2 = Student("Jean Charles", "Tutu", datetime(2000, 1, 1))
self.assertEqual(stud2._firstname, "Jean Charles", "firstname with space")
```

- Ce test assure que le prénom peut contenir des espaces, et que l'attribut est bien initialisé.

Cas 3 : Erreur si le prénom est vide

```
self.assertRaises(ValueError, Student, "", "Tutu", datetime(2021, 1, 1))
```

- **`assertRaises`** vérifie que le constructeur lève une exception `ValueError` lorsque le prénom est vide.
- Les paramètres donnés après `ValueError` sont passés au constructeur pour reproduire le cas problématique.