Université de Bordeaux
Semestre 2

2022-2023

TD6 : Les Tests Unitaires

Compétences

- Découvrir la notion de tests et pratiquer des tests unitaires
- Découvrir la notion de couverture de code

Réaliser des tests unitaires

Regarder la vidéo suivante : https://www.youtube.com/watch?v=hBCaoN421Qs

Le module `unittest`

Pour réaliser des tests unitaires, nous utiliserons le module `unittest` dont la documentation officielle se trouve ici. Ce module s'appuie sur trois concepts importants :

- test fixture : représente la préparation nécessaire pour réaliser un test. Tout comme par exemple la création temporaire de base de données, dossiers ou même le démarrage de services.
- test case : consiste à tester une fonctionnalité précise, et ainsi tester que la sortie corresponde bien à un résultat attendu.
- test runner : gère l'exécution des tests et fournit la sortie à l'utilisateur sous forme graphique ou textuelle.

Activité préliminaire

Téléchargez les sources de cette activité (`src_etudiant.zip`). Les sources contiennent deux dossiers :

- **liste** contenant l'implémentation du type de donnée abstrait (TDA) `liste`,
- **tests** contenant un fichier `test_liste.py`

Le fichier `test_liste.py` est une classe héritant de `unittest.TestCase`, les tests sont écrits sous forme de méthodes. Les noms des méthodes doivent **impérativement** commencer par **`test`** afin d'indiquer au **`test runner`** quelles sont les méthodes de tests. De plus, chaque test doit appeler une fonction `assertion` de la classe `TestCase`. La classe `TestCase` possède plusieurs types de **`assert`**. Ici, nous utiliserons **`assertTrue`** qui permet de tester la valeur de vérité d'une expression booléenne.
Pour exécuter les tests, on utilise la fonction **`unittest.main()`**. Les tests peuvent être exécutés directement depuis VSCode ou bien en ligne de commande :

```
python3 -m unittest -v test_liste.py
```

1. Combien de tests y a-t-il dans le fichier `test_liste.py` ?
2. À l'aide du code, définissez l'objectif de chaque test.
3. À l'aide de la documentation, expliquez le rôle des méthodes **`setUp`** et **`tearDown`**
4. Lancez l'exécution des tests. Quels sont les tests valides ?
5. Corrigez le code pour que tous les tests soient valides.
6. Écrivez un nouveau test dont l'objectif est de vérifier l'ordre d'insertion des valeurs dans la liste avec **`assertEqual`**. Vous trouverez une description détaillée de cet assert dans la documentation de `unittest`

La couverture de code avec `coverage`

La couverture de code est l'ensemble des lignes exécutées par les tests unitaires. Cela ne signifie pas toujours qu'elles sont correctes, mais seulement qu'elles ont été exécutées au moins une fois sans provoquer d'erreur. Le module le plus simple est `coverage`. L'utilisation du module `coverage` se résume à une ligne de commande :

```
python3-coverage run -m unittest
```

Un fichier **`.coverage`** apparaît alors. Ce sont des données brutes plus facilement lisibles après leur conversion en un rapport de couverture.

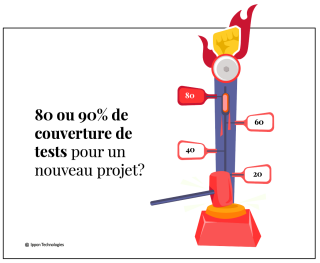
```
python3-coverage report -m
```

Ce qui donne dans notre cas :

```
               Stmts   Miss  Cover   Missing
-----------------------------------------------
/__init__.py       1      0   100%
/liste.py         41      2    95%   17, 42
test_liste.py     35      1    97%   46
-----------------------------------------------
TOTAL             77      3    96%
```

On peut également obtenir ce rapport au format HTML avec la commande suivante :

```
1 python3-coverage html -d coverage.html
```



1. À partir du rapport de couverture du code, indiquez le % de code utilisé par les tests.
2. Si le % est inférieur à 100%, indiquez la ou les lignes de code ignorée(s).
3. Ces lignes peuvent-elles être source de bug(s) ? Si oui, définissez un objectif de test et implémentez-le.
4. Des tests réussis avec une couverture de code de 100% sont-ils une garantie qu'il n'y ait pas de bug dans notre programme ?

Le projet `VideoTracker`

Une première série de tests pour le projet

En vous inspirant de l'activité préliminaire :

1. Définissez deux ou trois objectifs de test pour la tâche liée à la classe `FileRepo`. Vous devez rédiger vos tests de telle manière que des personnes extérieures à votre projet puissent procéder aux tests aisément et sans avoir besoin de vous poser des questions. Vous pouvez par exemple utiliser un tableau pour présenter vos tests

NUMÉRO	ACTION	ATTENDU	RÉSULTAT
1	…	…	…

2. Écrivez dans un fichier `test_FileRepo.py` les tests correspondant à vos objectifs.
3. Éditez le rapport de couverture de code et commentez le.