Écrire des tests unitaires - TD 1

Pierre-Antoine Guillaume

7 avril 2025

0.1 Recommandations

- Pour ce TP, j'attends que vous soyez autonome et capable d'utiliser la ligne de commande
- nommez correctement vos tests : i.e. def test_score_spare_adds_score_once_more()
- si le nom du test ne se suffit pas, ajoutez un commentaire pour mieux exprimer votre intention
- faites les exercices dans l'ordre, ils sont ordonnés par difficulté croissante
- Commitez souvent
- essayez de faire du TDD
  Des étudiants m'ont déjà dit pour ce travail « *Ça fait une heure que je n'ai pas essayé de lancer les tests, et que je n'ai pas commité, et mes tests ne passent plus je ne sais pas pourquoi* »
- procédez étape par étape, exigence par exigence : n'essayez pas de tout traiter d'un seul coup
- vous pouvez utiliser le langage que vous souhaitez, dans mes exemples j'ai parfois utilisé python, rust et php, mais prenez un outil avec lequel vous êtes à l'aise (idéalement celui de votre projet fil rouge)
- j'attends que vous décidiez également du framework de tests que vous voulez utiliser

1 Calculatrice

1.1 Détails de l'exercice

Écrivez les tests d'un module qui propose :
— une fonction somme, qui permet de faire la somme de deux éléments
— une fonction soustrait, qui permet de faire la soustraction de deux éléments
— une fonction multiplie, qui permet de faire la multiplication de deux éléments
— une fonction divise, qui permet de faire la division d'un élément par l'autre
— une fonction exponentiation, qui permet de faire la puissance d'un élément par l'autre

2 FizzBuzz

2.1 Détails de l'exercice

Le programme doit parcourir les nombres de 1 à `n` et :
— Afficher *"Fizz"* si le nombre est divisible par 3.
— Afficher *"Buzz"* si le nombre est divisible par 5.
— Afficher *"FizzBuzz"* si le nombre est divisible par 3 et 5.
— Afficher le nombre lui-même dans tous les autres cas.

2.2 Indications supplémentaires

1. Attention, les I/O sont particulièrement longs. Dans vos tests, ne faites pas les outputs.
2. Programme principal : Écrivez un programme qui parcourt les nombres de 1 à `n` et applique les règles FizzBuzz.
3. Sortie : Le programme doit afficher soit *"Fizz"*, *"Buzz"*, *"FizzBuzz"*, soit le nombre, selon les règles.
4. Testez la valeur 2'000'000'001 pour savoir si c'est Fizz, Buzz, FizzBuzz, ou bien un chiffre.

2.3 Exemple d'exécution

Pour `n = 15`, le programme doit afficher :

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

3 Bowling

3.1 Règles du bowling

- une partie se compose de 10 frames
- à chaque frame, 10 quilles sont installées et le joueur a jusqu'à deux lancés pour en faire tomber le maximum
- le score d'une frame est le nombre de quilles tombées, avec un bonus pour les strikes et les spares
- un spare consiste à faire tomber toutes les quilles après les deux lancés d'une frame.
  - le bonus associé à un spare est le nombre de quilles tombées lors du prochain lancé.
- un strike consiste à faire tomber toutes (10) les quilles dès le premier lancé d'une frame, ce qui met fin à la frame.
  - le bonus associé à un strike est le nombre de quilles tombées lors des deux prochains lancés.
- à la dernière frame, en cas de spare ou de strike, le joueur peut continuer à faire des lancés pour comptabiliser ses bonus.

Lors de la dernière frame, on ne peut pas faire plus de trois lancés.

NB : Le score le plus haut possible est 300 points.

3.2 Consigne

Vous devez implémenter la classe Game, située dans le dossier src.

Vous n'avez pas le droit de modifier *l'interface* de la classe Game, mais vous pouvez rajouter des propriétés et remplacer **pass** par des comportements appropriés.

Vous pouvez également rajouter de nouvelles méthodes, mais la classe devra être pilotée par ses méthodes roll et score.

```
class Game:
    def __init__(self):
        pass

    def roll(self, pins) -> None:
        pass

    def score(self) -> int:
        pass
```