

Behaviour Driven Development sous Python

Un guide efficace proposé par

Jérémy SELLEM
Mohamed BOUNAIME

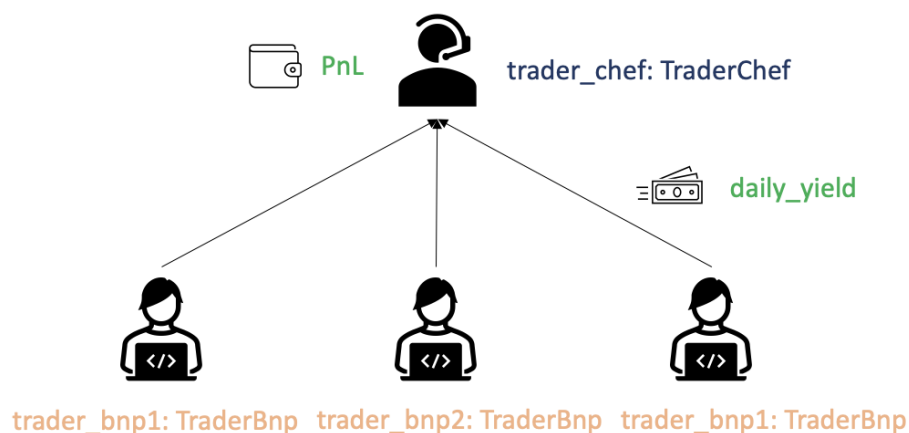
Prérequis :

- Télécharger et installer *python 3.8* disponible [ici](#)
- Installer *pytest* disponible [ici](#)
- Installer *behave* disponible [ici](#)

Sujet :

Nous nous plaçons dans l'environnement des marchés financiers. Une salle des marchés est organisée en équipes de traders. Chaque trader (TraderBnp) doit reporter à un chef des traders (TraderChef).

Le but va être de simuler la remontée quotidienne des daily yields et l'agrégation de tous ces résultats dans le P&L du chef.

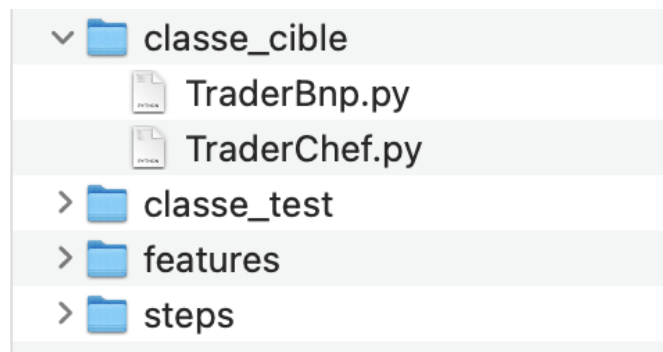


Tutorial :

1. Munissez vous des classes **TraderBnp** et **TraderChef** écrites précédemment
2. Ouvrez **PyCharm**

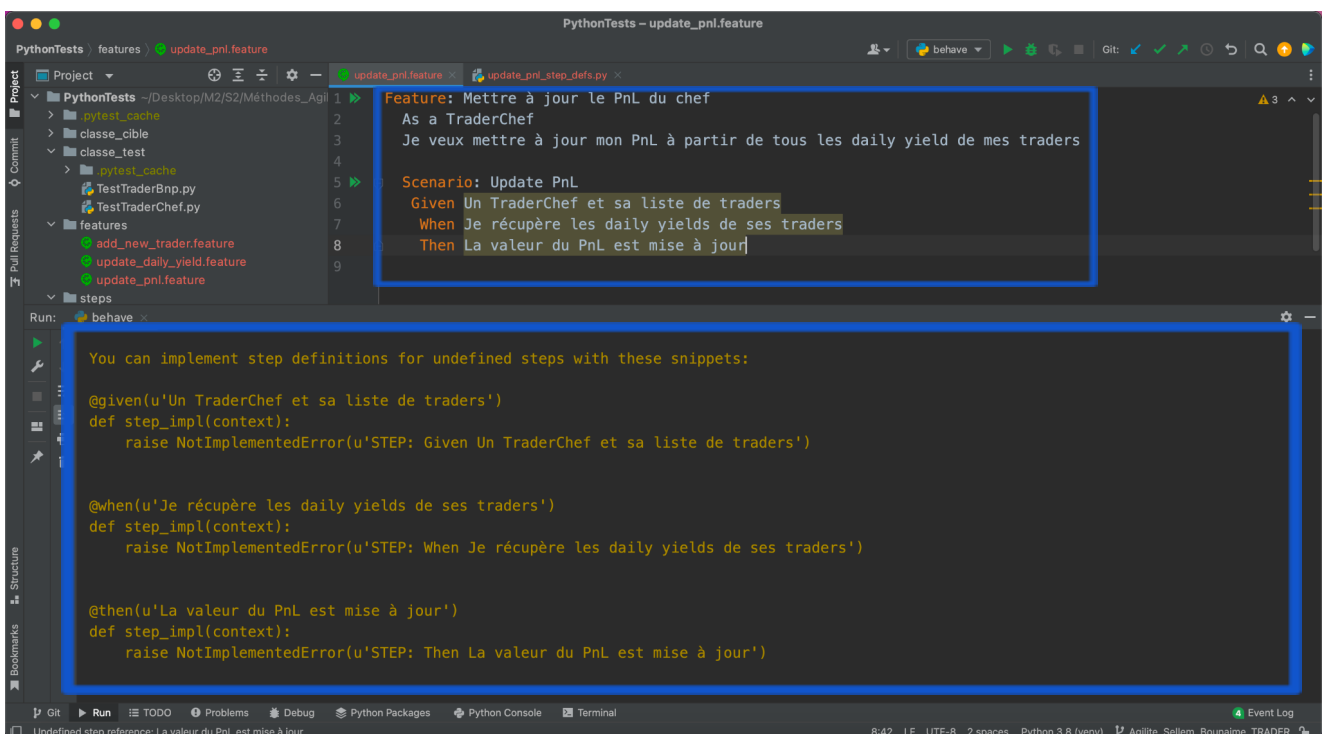
Nous allons créer des **User stories**, ce sont des **cas d'usage écrits** en langage naturel et permettant de générer des tests sur les classes cibles.

3. Créez la hiérarchie suivante :



Prenons l'exemple du cas d'usage "à la fin de la journée, le trader en chef collecte tous les daily yields de ses traders".

4. Dans le dossier **features**, créez un fichier texte **update_pnl.feature**
5. Écrire dans le langage **Gherkin** le cas d'usage ainsi que son **Scénario**
6. Exécuter la commande **behave**



7. Un message apparaît, il contient des indications à suivre pour implémenter les **step definitions** correspondantes
8. Dans le dossier **steps**, créez un fichier python **update_pnl_step_defs.py**
9. Copier/Coller les snippets précédents et formater cela dans une classe

Votre step definition `update_pnl_step_defs.py` doit ressembler à ceci :

```
from behave import *
from classe_cible.TraderChef import TraderChef
from classe_cible.TraderBnp import TraderBnp
from dataclasses import dataclass

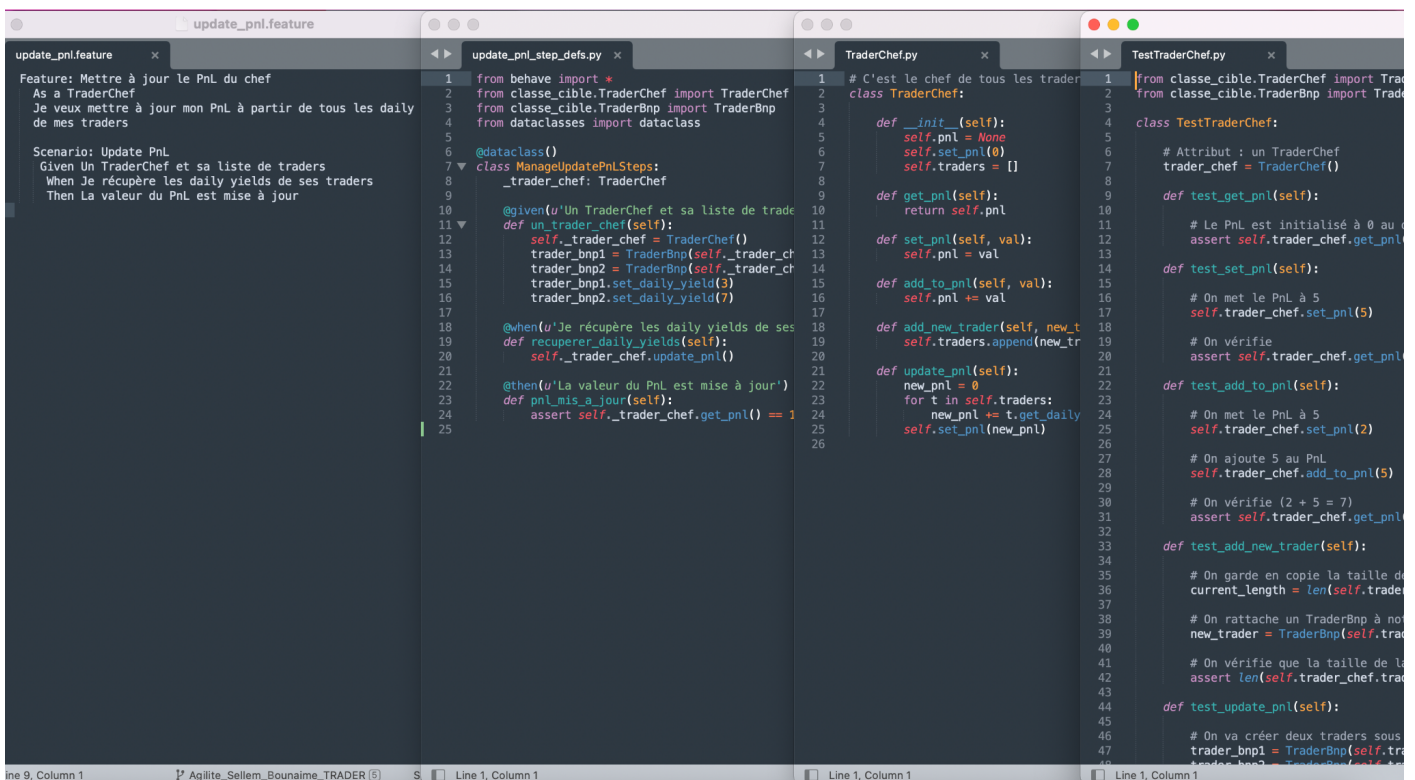
@dataclass()
class ManageUpdatePnlSteps:
    _trader_chef: TraderChef

    @given(u'Un TraderChef et sa liste de traders')
    def un_trader_chef(self):
        self._trader_chef = TraderChef()
        trader_bnp1 = TraderBnp(self._trader_chef)
        trader_bnp2 = TraderBnp(self._trader_chef)
        trader_bnp1.set_daily_yield(3)
        trader_bnp2.set_daily_yield(7)

    @when(u'Je récupère les daily yields de ses traders')
    def recuperer_daily_yields(self):
        self._trader_chef.update_pnl()

    @then(u'La valeur du PnL est mise à jour')
    def pnl_mis_a_jour(self):
        assert self._trader_chef.get_pnl() == 10
```

10. D'un point de vue global, les fichiers utiles à votre premier test doivent être similaires à ceci :



Bravo ! Vous avez mis à jour le P&L et votre chef est content de vos résultats !



Cas particulier : Scénario paramétré

Dans cet exemple, nous montrons comment vérifier que l'actualisation du **daily yield** d'un **TraderBnp** s'effectue correctement.

La **User Story** dans le langage **Gherkin** s'écrit avec le mot-clé **Outline** et fournit des exemples concrets avec un résultat attendu. Chaque ligne du tableau sera un Scénario testé :

```
Feature: Mettre à jour le Daily Yield d'un trader
  As a TraderBnp
  Je veux mettre à jour mon Daily Yield

  Scenario Outline: Update Daily Yield
    Given Un TraderBnp et son Daily Yield initialement à <current_daily>
    When J'ajoute une somme <new_daily> à son Daily Yield
    Then Je récupère la valeur du Daily Yield mis à jour <updated_daily>

    Examples:
    | current_daily | new_daily | updated_daily |
    | 4             | 3         | 7             |
    | 7.1           | 5.2       | 12.3          |
    | -1            | 4         | 2             |
```

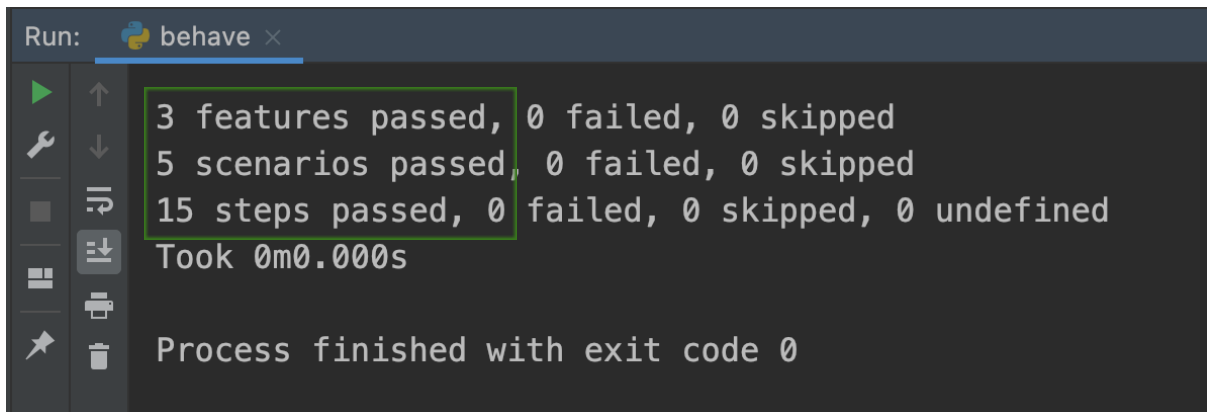
Le 3ème scénario est incorrect car $-1 + 4 = 3$. Lorsque que vous exécutez **behave**, ce **scenario** ainsi que la **feature** qui l'a appelé apparaissent en **failed** :

```
behave x
Failing scenarios:
  Users/thegreatgolfer/Desktop/M2/S2/Méthodes_Agiles/TP1/PythonTests/features/update_daily_yield.feature:13  Update Daily Yield -- @1.3

2 features passed, 1 failed, 0 skipped
4 scenarios passed, 1 failed, 0 skipped
14 steps passed, 1 failed, 0 skipped, 0 undefined
Took 0m0.001s

Process finished with exit code 1
```

Objectif : tous les **scénarios** et **features** à **passed** !



```
Run: behave x
3 features passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
15 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s

Process finished with exit code 0
```