

Tests unitaires sous Python

Un guide efficace proposé par

Jérémy SELLEM
Mohamed BOUNAIME

Prérequis :

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

Tutorial :

1. Ouvrez votre environnement de développement python préféré, nous utiliserons Visual Studio dans ce qui suit.
2. Créer un projet.

Le but est de coder une relation bidirectionnelle 0..1 à * et de la tester.

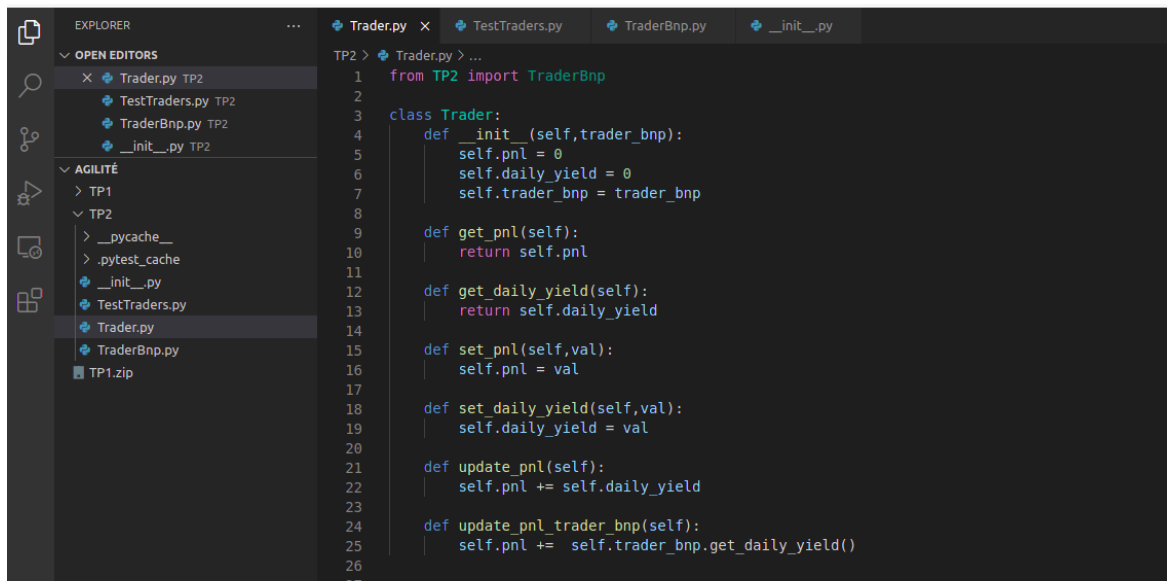
3. Importer les classes **Trader** et **TraderBnp** vues dans la Partie 1 de ce guide.



4. Code de la classe **TraderBnp**

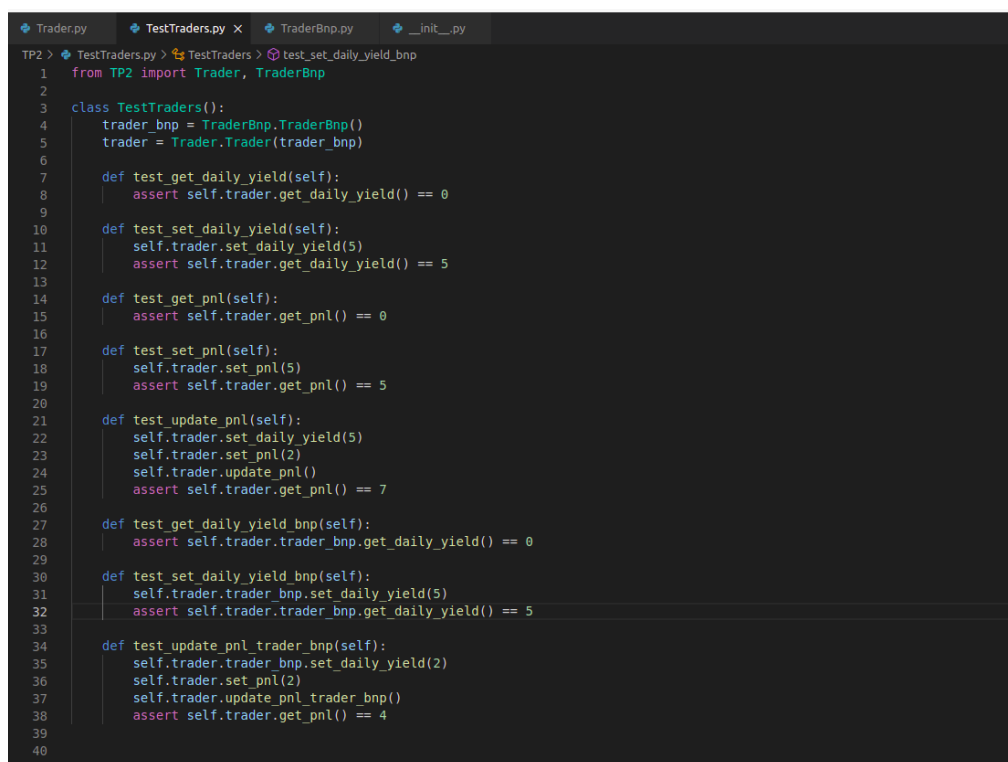
```
1
2
3 class TraderBnp:
4     def __init__(self):
5         self.daily_yield = 0
6
7     def get_daily_yield(self):
8         return self.daily_yield
9
10    def set_pnl(self, val):
11        self.pnl = val
12
13    def set_daily_yield(self, val):
14        self.daily_yield = val
15
16    def update_daily_yield(self, b):
17        self.daily_yield += b
18
```

5. Code de la classe **Trader**



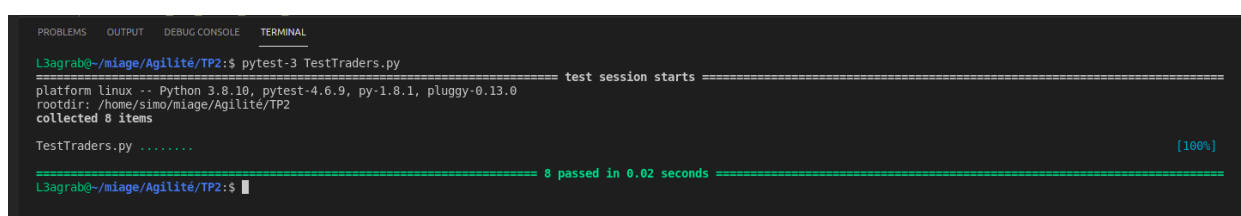
```
1 from TP2 import TraderBnp
2
3 class Trader:
4     def __init__(self, trader_bnp):
5         self.pnl = 0
6         self.daily_yield = 0
7         self.trader_bnp = trader_bnp
8
9     def get_pnl(self):
10        return self.pnl
11
12    def get_daily_yield(self):
13        return self.daily_yield
14
15    def set_pnl(self, val):
16        self.pnl = val
17
18    def set_daily_yield(self, val):
19        self.daily_yield = val
20
21    def update_pnl(self):
22        self.pnl += self.daily_yield
23
24    def update_pnl_trader_bnp(self):
25        self.pnl += self.trader_bnp.get_daily_yield()
26
27
```

6. Créer une classe de test : **TestTrader**



```
1 from TP2 import Trader, TraderBnp
2
3 class TestTraders():
4     trader_bnp = TraderBnp.TraderBnp()
5     trader = Trader.Trader(trader_bnp)
6
7     def test_get_daily_yield(self):
8         assert self.trader.get_daily_yield() == 0
9
10    def test_set_daily_yield(self):
11        self.trader.set_daily_yield(5)
12        assert self.trader.get_daily_yield() == 5
13
14    def test_get_pnl(self):
15        assert self.trader.get_pnl() == 0
16
17    def test_set_pnl(self):
18        self.trader.set_pnl(5)
19        assert self.trader.get_pnl() == 5
20
21    def test_update_pnl(self):
22        self.trader.set_daily_yield(5)
23        self.trader.set_pnl(2)
24        self.trader.update_pnl()
25        assert self.trader.get_pnl() == 7
26
27    def test_get_daily_yield_bnp(self):
28        assert self.trader.trader_bnp.get_daily_yield() == 0
29
30    def test_set_daily_yield_bnp(self):
31        self.trader.trader_bnp.set_daily_yield(5)
32        assert self.trader.trader_bnp.get_daily_yield() == 5
33
34    def test_update_pnl_trader_bnp(self):
35        self.trader.trader_bnp.set_daily_yield(2)
36        self.trader.set_pnl(2)
37        self.trader.update_pnl_trader_bnp()
38        assert self.trader.get_pnl() == 4
39
40
41
```

7. Exécuter *pytest-3* *TestTraders.py* pour lancer les fonctions de **TestTrader** depuis le dossier contenant votre projet.



```
L3agrab@~/miage/Agilité/TP2:~$ pytest-3 TestTraders.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-4.6.9, py-1.8.1, pluggy-0.13.0
rootdir: /home/simo/miage/Agilité/TP2
collected 8 items

TestTraders.py ..... [100%]

===== 8 passed in 0.02 seconds =====
L3agrab@~/miage/Agilité/TP2:~$
```

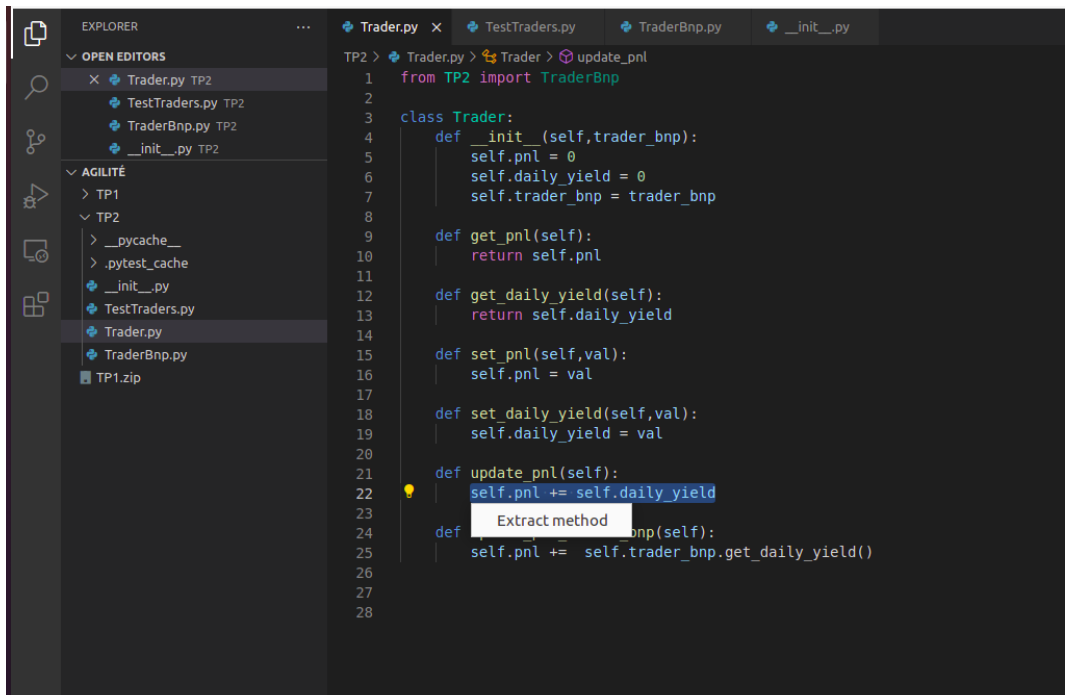
Refactoring : “opération consistant à retravailler le code source d'un programme informatique, sans toutefois y ajouter des fonctionnalités ni en corriger les bogues, de façon à en améliorer la lisibilité et, par voie de conséquence, la maintenance, ou à le rendre plus générique” ([Wikipedia](#))

8. Nous allons utiliser la technique **ExtractMethod** sur la mise à jour du PnL

Clique droit

Refactor...

Extract method



9. L'instruction est remplacée par une méthode

```
from TP2 import TraderBnp

class Trader:
    def __init__(self, trader_bnp):
        self.pnl = 0
        self.daily_yield = 0
        self.trader_bnp = trader_bnp

    def get_pnl(self):
        return self.pnl

    def get_daily_yield(self):
        return self.daily_yield

    def set_pnl(self, val):
        self.pnl = val

    def set_daily_yield(self, val):
        self.daily_yield = val

    def update_pnl(self):
        self.add_pnl_daily_yield()

    def add_pnl_daily_yield(self):
        self.pnl += self.daily_yield

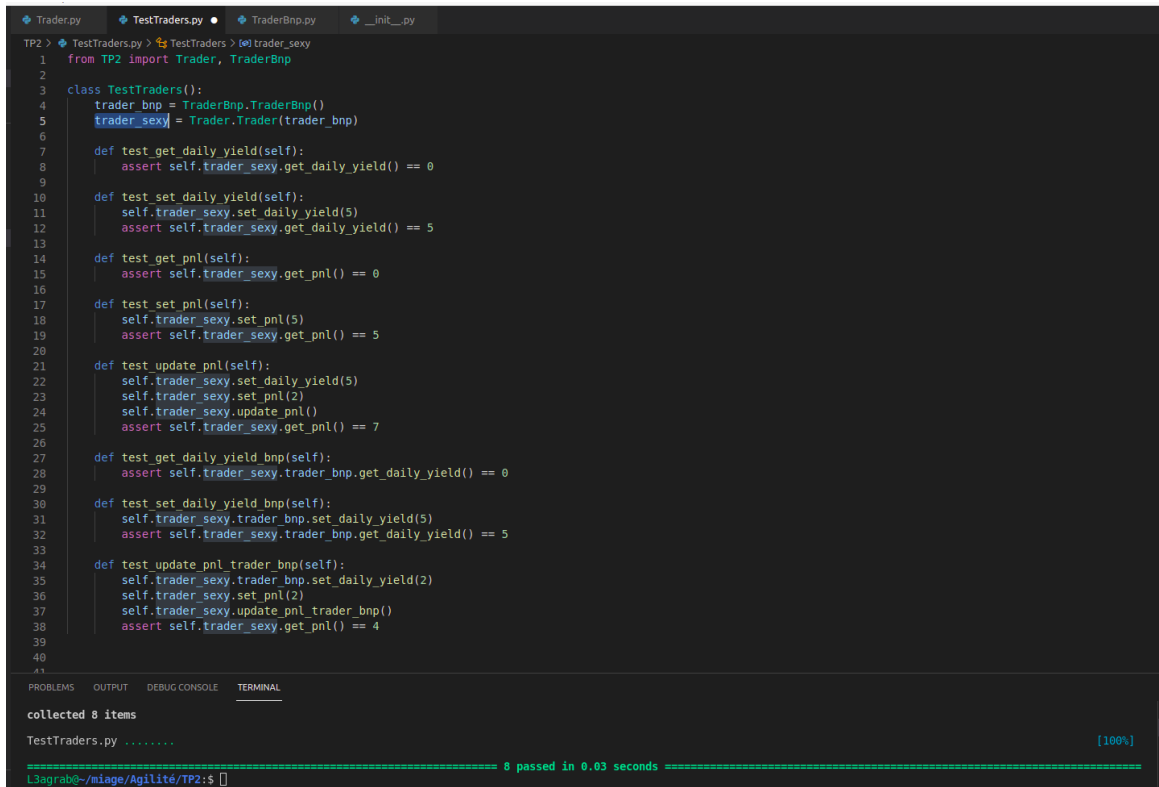
    def update_pnl_trader_bnp(self):
        self.pnl += self.trader_bnp.get_daily_yield()
```

10. Autre technique de **Refactoring**, nous allons utiliser la technique **Rename** sur le champ trader

```
TP2 > TestTraders.py > TestTraders > trader
1 from TP2 import Trader, TraderBnp
2
3 class TestTraders():
4     trader_bnp = TraderBnp.TraderBnp()
5     trader = Trader.Trader(trader_bnp)
6     trader
7     Enter to Rename, Shift+Enter to Preview
8     assert self.trader.get_daily_yield() == 0
9
10    def test_set_daily_yield(self):
11        self.trader.set_daily_yield(5)
12        assert self.trader.get_daily_yield() == 5
13
14    def test_get_pnl(self):
15        assert self.trader.get_pnl() == 0
16
17    def test_set_pnl(self):
18        self.trader.set_pnl(5)
19        assert self.trader.get_pnl() == 5
20
21    def test_update_pnl(self):
22        self.trader.set_daily_yield(5)
23        self.trader.set_pnl(2)
24        self.trader.update_pnl()
25        assert self.trader.get_pnl() == 7
26
27    def test_get_daily_yield_bnp(self):
28        assert self.trader.trader_bnp.get_daily_yield() == 0
29
30    def test_set_daily_yield_bnp(self):
31        self.trader.trader_bnp.set_daily_yield(5)
32        assert self.trader.trader_bnp.get_daily_yield() == 5
33
34    def test_update_pnl_trader_bnp(self):
35        self.trader.trader_bnp.set_daily_yield(2)
36        self.trader.set_pnl(2)
37        self.trader.update_pnl_trader_bnp()
38        assert self.trader.get_pnl() == 4
39
40
```

```
TP2 > TestTraders.py > ...
1 from TP2 import Trader, TraderBnp
2
3 class TestTraders():
4     trader_bnp = TraderBnp.TraderBnp()
5     trader_sexy = Trader.Trader(trader_bnp)
6
7     def test_get_daily_yield(self):
8         assert self.trader_sexy.get_daily_yield() == 0
9
10    def test_set_daily_yield(self):
11        self.trader_sexy.set_daily_yield(5)
12        assert self.trader_sexy.get_daily_yield() == 5
13
14    def test_get_pnl(self):
15        assert self.trader_sexy.get_pnl() == 0
16
17    def test_set_pnl(self):
18        self.trader_sexy.set_pnl(5)
19        assert self.trader_sexy.get_pnl() == 5
20
21    def test_update_pnl(self):
22        self.trader_sexy.set_daily_yield(5)
23        self.trader_sexy.set_pnl(2)
24        self.trader_sexy.update_pnl()
25        assert self.trader_sexy.get_pnl() == 7
26
27    def test_get_daily_yield_bnp(self):
28        assert self.trader_sexy.trader_bnp.get_daily_yield() == 0
29
30    def test_set_daily_yield_bnp(self):
31        self.trader_sexy.trader_bnp.set_daily_yield(5)
32        assert self.trader_sexy.trader_bnp.get_daily_yield() == 5
33
34    def test_update_pnl_trader_bnp(self):
35        self.trader_sexy.trader_bnp.set_daily_yield(2)
36        self.trader_sexy.set_pnl(2)
37        self.trader_sexy.update_pnl_trader_bnp()
38        assert self.trader_sexy.get_pnl() == 4
39
40
```

11. Exécuter de nouveau les tests pour s'assurer que le Refactoring n'a pas impacté les fonctionnalités



```
TP2 > TestTraders.py > TestTraders > trader_sexy
1 from TP2 import Trader, TraderBnp
2
3 class TestTraders():
4     trader_bnp = TraderBnp.TraderBnp()
5     trader_sexy = Trader.Trader(trader_bnp)
6
7     def test_get_daily_yield(self):
8         assert self.trader_sexy.get_daily_yield() == 0
9
10    def test_set_daily_yield(self):
11        self.trader_sexy.set_daily_yield(5)
12        assert self.trader_sexy.get_daily_yield() == 5
13
14    def test_get_pnl(self):
15        assert self.trader_sexy.get_pnl() == 0
16
17    def test_set_pnl(self):
18        self.trader_sexy.set_pnl(5)
19        assert self.trader_sexy.get_pnl() == 5
20
21    def test_update_pnl(self):
22        self.trader_sexy.set_daily_yield(5)
23        self.trader_sexy.set_pnl(2)
24        self.trader_sexy.update_pnl()
25        assert self.trader_sexy.get_pnl() == 7
26
27    def test_get_daily_yield_bnp(self):
28        assert self.trader_sexy.trader_bnp.get_daily_yield() == 0
29
30    def test_set_daily_yield_bnp(self):
31        self.trader_sexy.trader_bnp.set_daily_yield(5)
32        assert self.trader_sexy.trader_bnp.get_daily_yield() == 5
33
34    def test_update_pnl_trader_bnp(self):
35        self.trader_sexy.trader_bnp.set_daily_yield(2)
36        self.trader_sexy.set_pnl(2)
37        self.trader_sexy.update_pnl_trader_bnp()
38        assert self.trader_sexy.get_pnl() == 4
39
40
41
```

collected 8 items

TestTraders.py [100%]

===== 8 passed in 0.03 seconds =====

L3agrab@~/miage/Agilité/TP2:\$

12. On finit sur une note plus philosophique, Edward A. Murphy Jr et son équipe d'ingénieurs disaient :

« S'il y a plus d'une façon de faire quelque chose, et que l'une d'elles conduit à un désastre, alors il y aura quelqu'un pour le faire de cette façon »

Nous avons pu constater ce phénomène quand il a fallu transférer le code Java d'un ordinateur à un autre, puisque nous travaillons en binôme. Le code généré par BlueJ ne fonctionnait pas sur l'ordinateur du partenaire. Et nous avons dû le réécrire manuellement. En cela nous voyons une application de la loi de Murphy.