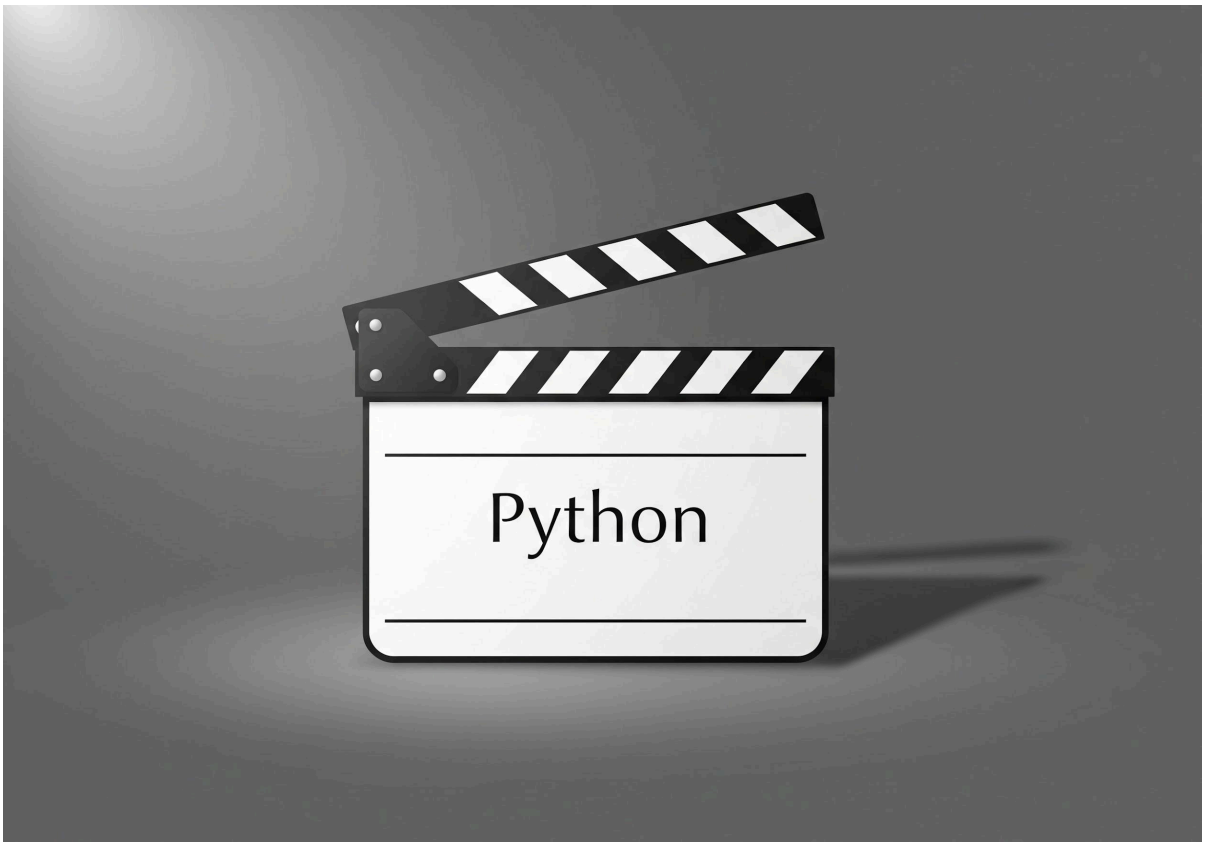
🎬 Bienvenue dans le Studio Python : Le Making-of de votre Projet



Bonjour à tous ! Aujourd'hui, nous n'allons pas seulement écrire du code, nous allons diriger un studio de production. Imaginez que votre ordinateur est le studio, et Python est le langage que nous parlons sur le plateau.

Votre projet a une histoire : il a commencé dans un "vieux studio" (Java/BlueJ) et nous venons de déménager dans un studio ultra-moderne (Python). Voici comment nous sommes organisés.

1. Le Casting et les Rôles (Les Classes)

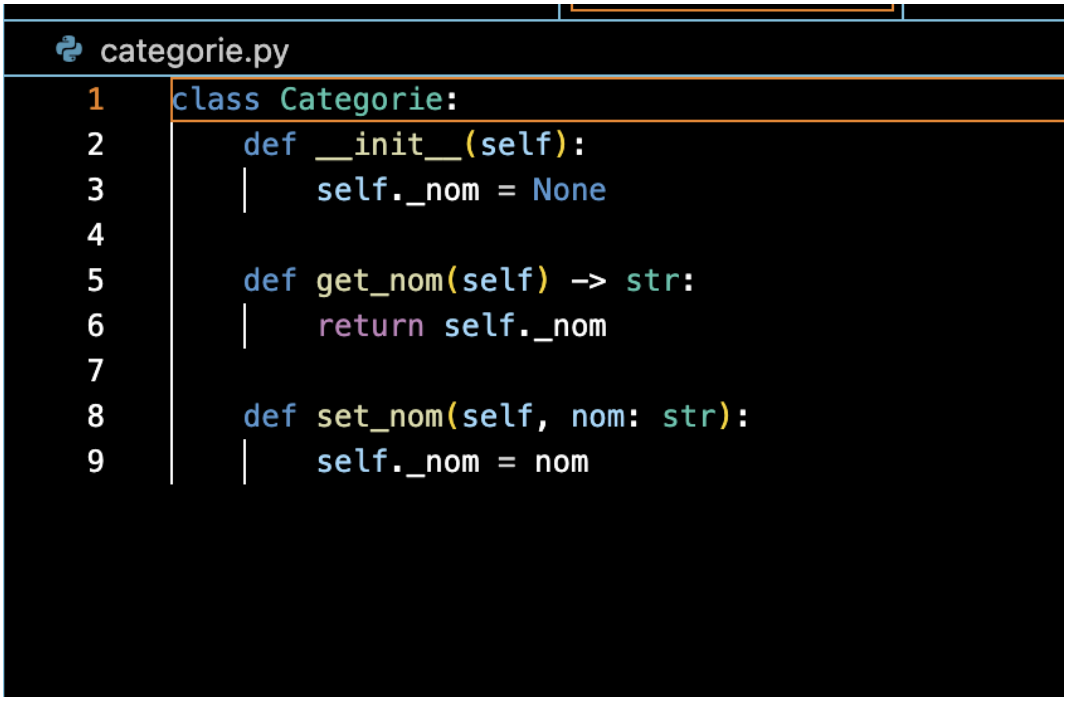
Dans tout bon film, il faut des acteurs et des rôles bien définis. Dans notre monde Python, c'est ce qu'on appelle des **Classes**.

- **Le Rôle "Film"** (`film.py`) : C'est notre star principale. Imaginez une fiche de casting vierge. Pour être un film dans notre studio, il faut avoir une durée, une date de sortie et une catégorie.

film.py

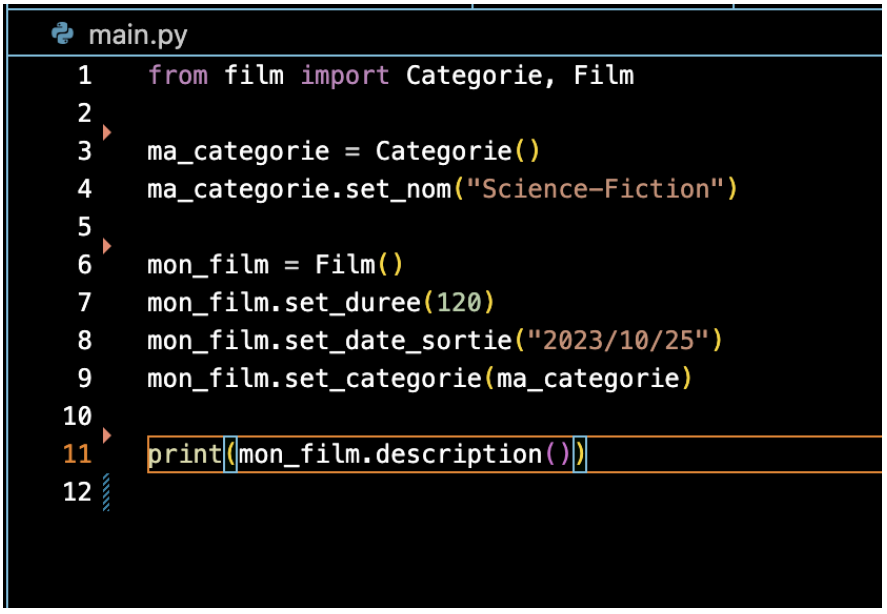
```python
from categorie import Categorie

class Film:
    def __init__(self):
        self._duree = 0
        self._date_sortie = "2001/01/01"
        self._categorie = None

    def get_duree(self) -> int:
        return self._duree

    def set_duree(self, duree: int):
        self._duree = duree

    def get_date_sortie(self) -> str:
        return self._date_sortie

    def set_date_sortie(self, date_sortie: str):
        self._date_sortie = date_sortie

    def set_categorie(self, categorie: Categorie):
        self._categorie = categorie

    def get_categorie(self) -> Categorie:
        return self._categorie

    def description(self) -> str:
        if self._categorie:
            nom_cat = self._categorie.get_nom()
        else:
            nom_cat = "Inconnue"

        return f"Je suis un film de catégorie {nom_cat} et de durée {self._duree} sorti en {self._date_sortie}"
```

- **Le Rôle "Catégorie"** (`categorie.py`) : C'est le genre du film (Science-Fiction, Drame, etc.). C'est une étiquette simple mais essentielle pour classer nos œuvres.



2. Le Réalisateur en Action (Le fichier `main.py`)

Si les fichiers précédents sont les fiches de casting, le fichier `main.py` est le réalisateur qui crie **"Action !"**.



Regardez ce qu'il fait dans le script :

1. **Il crée le décor :** Il invente une catégorie "Science-Fiction".

2. **Il recrute l'acteur :** Il crée un `mon_film`.
3. **Il donne les instructions :** "Toi, tu dures 120 minutes, tu sors le 25 octobre 2023, et tu es de la Science-Fiction".
4. **Le résultat :** Le film sait enfin qui il est et peut réciter sa description parfaite.

3. Les Répétitions en Coulisses (Les Tests Unitaires `test.py`)

Avant de montrer le film au public, on fait des répétitions privées. On veut être sûr que si on dit à l'acteur "Tu dures 111 minutes", il ne réponde pas "Je dure 0 minute".

C'est le rôle de `test.py`.

- Nous avons un inspecteur exigeant (`unittest`) qui vérifie chaque petit détail.
- Par exemple, il teste le film "Kill Bill". Il change sa date de sortie et vérifie immédiatement si le film a bien retenu la nouvelle date. Si ça ne matche pas, il arrête tout !

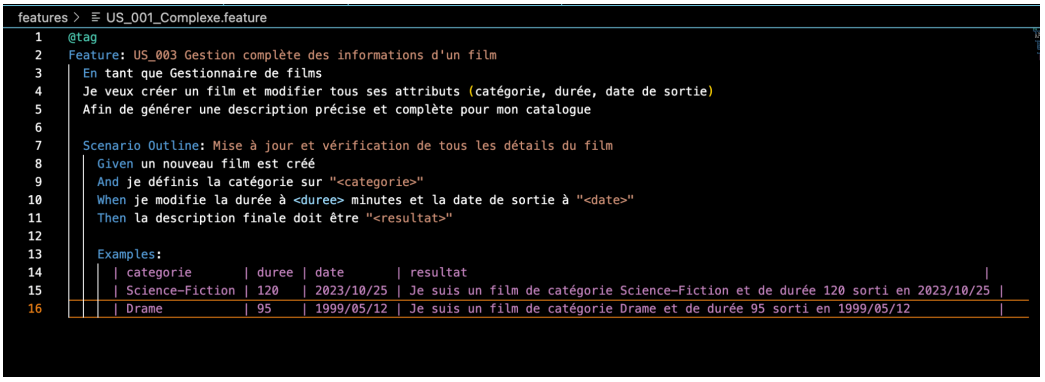
test.py

```python
import unittest
from film import Film, Categorie

class TestFilm(unittest.TestCase):


    def setUp(self):
        self.kill_bill = Film()
        self.action = Categorie()

        self.action.set_nom("Action")
        self.kill_bill.set_categorie(self.action)

    def tearDown(self):
        pass

    def test_set_duree(self):
        self.kill_bill.set_duree(111)
        self.assertEqual(111, self.kill_bill.get_duree())

    def test_set_date_sortie(self):
        self.kill_bill.set_date_sortie("2003/10/29")
        self.assertEqual("2003/10/29", self.kill_bill.get_date_sortie())

    def test_description_film(self):
        self.action.set_nom("Thriller")
        attendu = "Je suis un film de catégorie Thriller et de durée 0 sorti en 2001/01/01"
        self.assertEqual(attendu, self.kill_bill.description())

if __name__ == '__main__':
    unittest.main()
```

4. La Promesse Faite au Public (Tests Fonctionnels / Behave)

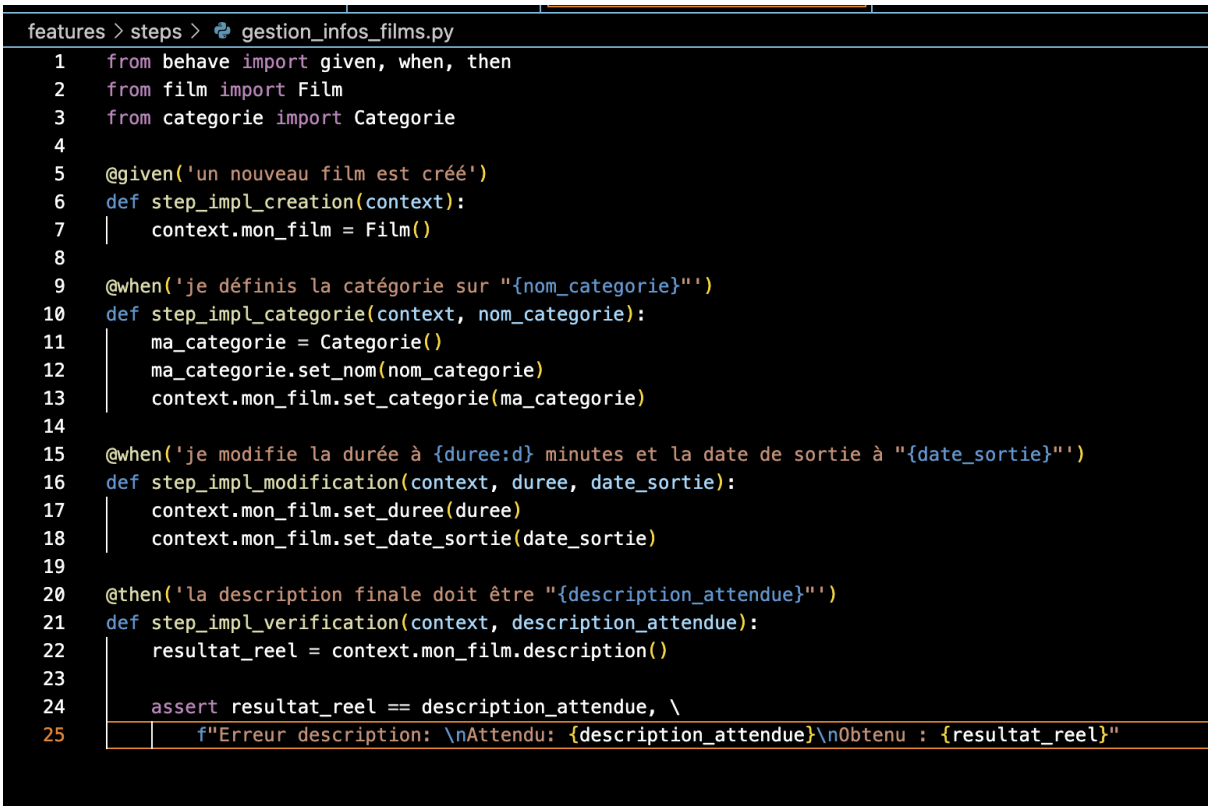
Ici, on entre dans la partie la plus "magique" pour les novices. Imaginez que le producteur (qui ne sait pas coder) écrive ses exigences sur une serviette en papier, en français (ou presque).

- **Le Scénario Client (`.feature`)** : Le producteur écrit : *"En tant que Gestionnaire, je veux modifier la durée... Quand je modifie la durée à 120 minutes... Alors la description doit être..."*. C'est lisible par un humain.



```
features > US_001_Complexe.feature
1   @tag
2   Feature: US_003 Gestion complète des informations d'un film
3     En tant que Gestionnaire de films
4     Je veux créer un film et modifier tous ses attributs (catégorie, durée, date de sortie)
5     Afin de générer une description précise et complète pour mon catalogue
6
7     Scenario Outline: Mise à jour et vérification de tous les détails du film
8       Given un nouveau film est créé
9       And je définis la catégorie sur "<categorie>"
10      When je modifie la durée à <duree> minutes et la date de sortie à "<date>"
11      Then la description finale doit être "<resultat>"
12
13      Examples:
14        | categorie       | duree | date       | resultat                                                          |
15        | Science-Fiction | 120   | 2023/10/25 | Je suis un film de catégorie Science-Fiction et de durée 120 sorti en 2023/10/25 |
16        | Drame           | 95    | 1999/05/12 | Je suis un film de catégorie Drame et de durée 95 sorti en 1999/05/12            |
```

- **La Traduction (`steps/`)** : Notre équipe technique (vous !) utilise un outil appelé `behave`. Il prend ces phrases en français et les connecte au code Python.
    - Quand le producteur dit *"un nouveau film est créé"*, Python exécute `context.mon_film = Film()`.



```
features > steps > gestion_infos_films.py
1   from behave import given, when, then
2   from film import Film
3   from categorie import Categorie
4
5   @given('un nouveau film est créé')
6   def step_impl_creation(context):
7       context.mon_film = Film()
8
9   @when('je définis la catégorie sur "{nom_categorie}"')
10  def step_impl_categorie(context, nom_categorie):
11      ma_categorie = Categorie()
12      ma_categorie.set_nom(nom_categorie)
13      context.mon_film.set_categorie(ma_categorie)
14
15  @when('je modifie la durée à {duree:d} minutes et la date de sortie à "{date_sortie}"')
16  def step_impl_modification(context, duree, date_sortie):
17      context.mon_film.set_duree(duree)
18      context.mon_film.set_date_sortie(date_sortie)
19
20  @then('la description finale doit être "{description_attendue}"')
21  def step_impl_verification(context, description_attendue):
22      resultat_reel = context.mon_film.description()
23
24      assert resultat_reel == description_attendue, \
25          f"Erreur description: \nAttendu: {description_attendue}\nObtenu : {resultat_reel}"
```

Nouvelle association bidirectionnelle : Réalisateur 📽️

On crée une nouvelle classe.

🎬 Scène 3 : L'Arrivée du "Big Boss" et le Grand Nettoyage

Le studio s'agrandit ! Il ne s'agit plus seulement d'avoir des acteurs (Films) et des genres (Catégories). Il nous faut quelqu'un pour diriger tout ça.

1. Le Nouveau Rôle : Le Réalisateur (`realisateur.py`)

*Fonctionnalité : Association bidirectionnelle 0..1 à **

Un nouveau personnage de poids entre dans le studio : **Le Réalisateur**. Mais attention, la relation entre un Réalisateur et ses Films est contractuelle et stricte (c'est ce qu'on appelle l'**Association Bidirectionnelle**).

- **Le Contrat "Donnant-Donnant" :** Dans votre code, quand Tarantino signe pour réaliser *Kill Bill*, deux choses se passent automatiquement :
    1. *Kill Bill* note "Tarantino" sur sa fiche.
    2. Tarantino ajoute *Kill Bill* à sa filmographie.
- C'est la magie de votre méthode `set_realisateur` dans `Film` et `ajouter_film` dans `Realisateur`. Si l'un le sait, l'autre le sait aussi. Impossible d'avoir un film qui croit être dirigé par Spielberg alors que Spielberg ne le connaît pas ! Vous avez "encapsulé" cette logique pour éviter les incohérences.

```
        return self._categorie

    def set_realisateur(self, realisateur: Realisateur):
        self._realisateur = realisateur

    def get_realisateur(self) -> Realisateur:
        return self._realisateur

    def description(self) -> str:
        if self._categorie:
```

realisateur.py > Realisateur

```python
from typing import List
from film import Film

class Realisateur:
    def __init__(self, nom: str):
        self._nom = nom
        self._films: List[Film] = []

    def get_nom(self) -> str:
        return self._nom

    def set_nom(self, nom: str):
        self._nom = nom

    def ajouter_film(self, film: Film):
        if film not in self._films:
            self._films.append(film)
            film.set_realisateur(self)

    def stats(self):
        t = 0
        for f in self._films:
            t += f.get_duree()

        res = f"Le réalisateur {self._nom} a produit {len(self._films)} films pour une durée de {t} min."
        return res
```

On voit bien que la fonction ajouter_film non seulement ajoute le film à la liste des films du réalisateur, mais elle précise bien au film quel est son réalisateur.

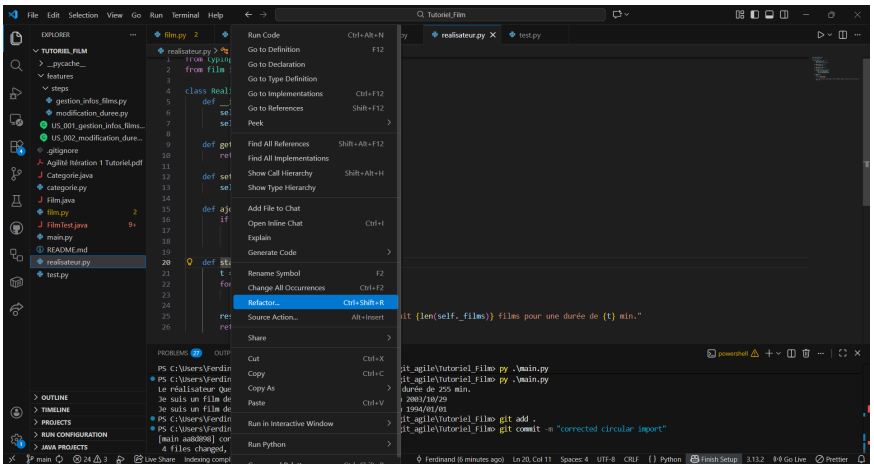
Pour être sûr de la robustesse de la classe, on ajoute un test unitaire simple qui vérifie le nom d'un nouveau réalisateur.

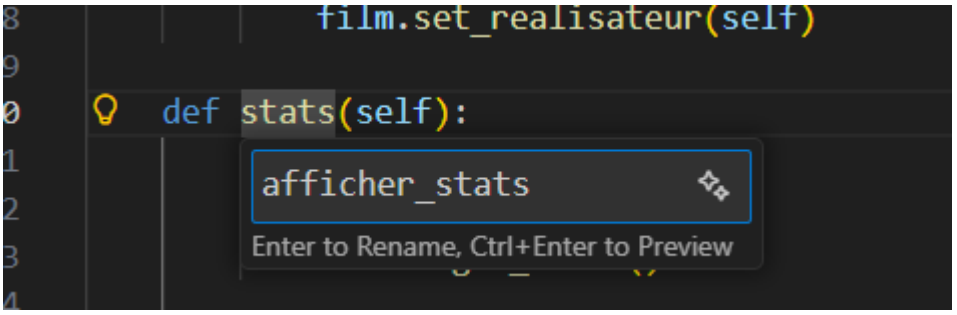
```python
    def setUp(self):
        self.kill_bill = Film()
        self.action = Categorie()
        self.real = Realisateur("Quentin Tarantino")

        self.action.set_nom("Action")
        self.kill_bill.set_categorie(self.action)

    def tearDown(self):
        pass

    def test_set_duree(self):
        self.kill_bill.set_duree(111)
        self.assertEqual(111, self.kill_bill.get_duree())

    def test_set_realisateur(self):
        self.assertEqual("Quentin Tarantino", self.real.get_nom())
```

2. La Réécriture du Scénario (Refactoring)

Le producteur a relu le script (le code) et a dit : "C'est brouillon ! On réécrit pour que ce soit plus clair". C'est l'art du **Refactoring**.

- **Changement de Nom (Rename) :** Peut-être aviez-vous une variable `d` qui est devenue `duree_minutes`. C'est comme changer le nom d'un personnage en cours de route pour qu'on comprenne mieux qui il est.
- **Découpage de Scène (Extract Method) :** Imaginez une méthode `description()` qui faisait 50 lignes. Vous l'avez probablement découpée. Au lieu que l'acteur raconte toute sa vie d'une traite, il fait appel à des sous-routines : "Donne-moi mon genre", "Donne-moi mon réalisateur". Le code devient plus lisible, comme un dialogue bien rythmé.

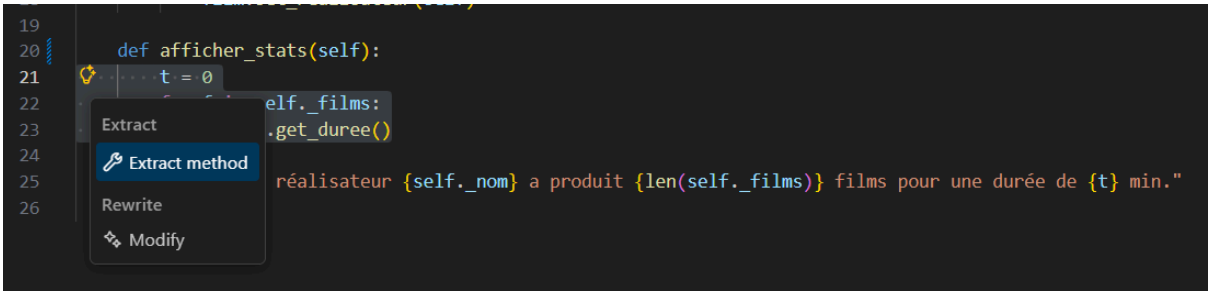




On voit bien que le changement s'est fait automatiquement dans le main.py :

```
realisateur1.ajouter_film(pulp_fiction)

print(realisateur1.afficher_stats())
print(kill_bill.description())
print(pulp_fiction.description())
```

Maintenant on veut extraire le calcul :



Et ça fonctionne :

Génial !

Test infected

Qu'est-ce que "Test Infected" ?

Être "infecté par les tests", c'est changer de mentalité : on ne voit plus les tests comme une corvée à faire à la fin, mais comme un **outil de conception** indispensable.

L'idée centrale de l'article est que **si un code n'a pas de test, il ne fonctionne pas (ou on ne peut pas le prouver)**. L'article introduit aussi le concept de **"Test-First"** (écrire le test avant le code).

On va donc ajouter deux tests unitaires :

- Un test unitaire pour voir le comportement de afficher_stats quand la liste de films est vide
- Un test bidirectionnel pour tester la bidirectionnalité

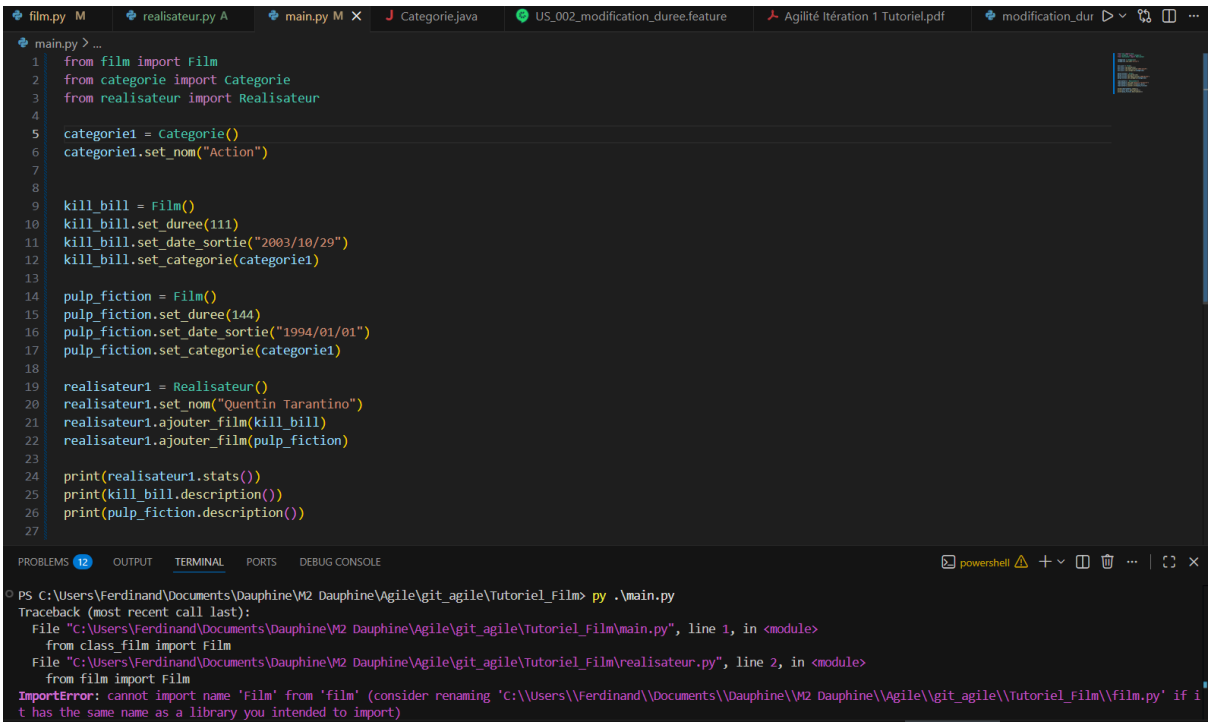
```
PS C:\Users\Ferdinand\Documents\Dauphine\M2 Dauphine\Agile\git_agile\Tutoriel_Film> py test.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
PS C:\Users\Ferdinand\Documents\Dauphine\M2 Dauphine\Agile\git_agile\Tutoriel_Film>
```

5. La Loi de Murphy du Tournage

Pour conclure, une petite leçon de sagesse apprise à la dure.

> **Loi de Murphy :** *"Si un test peut échouer, il échouera au moment de la démo devant le client."*

Dans votre projet : Vous avez sûrement vécu ce moment où tout marchait sur votre machine, mais dès que vous avez changé une petite ligne dans `Categorie` (par exemple le nom d'une variable), tout le test de `Film` s'est effondré en cascade. Une petite erreur de casting au début, et c'est tout le film qui est gâché à la fin !



Erreur de circular import : import en boucle les deux fichiers, n’en charge aucun.

film.py M | realisateur.py A | main.py M | Categorie.java | US_C

film.py > ...

```python
from categorie import Categorie
from realisateur import Realisateur

class Film:
    def __init__(self):
        self._duree = 0
        self._date_sortie = "2001/01/01"
        self._categorie = None
        self._realisateur = None

    def get_duree(self) -> int:
        return self._duree

    def set_duree(self, duree: int):
        self._duree = duree

    def get_date_sortie(self) -> str:
        return self._date_sortie

    def set_date_sortie(self, date_sortie: str):
        self._date_sortie = date_sortie

    def set_categorie(self, categorie: Categorie):
        self._categorie = categorie

    def get_categorie(self) -> Categorie:
        return self._categorie

    def set_realisateur(self, realisateur: Realisateur):
        self._realisateur = realisateur

    def get_realisateur(self) -> Realisateur:
        return self._realisateur

    def description(self) -> str:
        if self._categorie:
```

On voit ici les import.

```python
    def set_realisateur(self, realisateur: "Realisateur"):
        self._realisateur = realisateur

    def get_realisateur(self) -> "Realisateur":
        return self._realisateur

    def description(self) -> str:
```

Nouvelle version : type hinting et on a retiré le import Realisateur

main.py > ...

```python
from film import Film
from categorie import Categorie
from realisateur import Realisateur

categorie1 = Categorie()
categorie1.set_nom("Action")


kill_bill = Film()
kill_bill.set_duree(111)
kill_bill.set_date_sortie("2003/10/29")
kill_bill.set_categorie(categorie1)

pulp_fiction = Film()
pulp_fiction.set_duree(144)
pulp_fiction.set_date_sortie("1994/01/01")
pulp_fiction.set_categorie(categorie1)

realisateur1 = Realisateur("Quentin Tarantino")
realisateur1.ajouter_film(kill_bill)
realisateur1.ajouter_film(pulp_fiction)

print(realisateur1.stats())
print(kill_bill.description())
print(pulp_fiction.description())

```

Le main fonctionne cette fois

```
PS C:\Users\Ferdinand\Documents\Dauphine\M2 Dauphine\Agile\git_agile\Tutoriel_Film> py .\main.py
Le réalisateur Quentin Tarantino a produit 2 films pour une durée de 255 min.
Je suis un film de catégorie Action et de durée 111 sorti en 2003/10/29
Je suis un film de catégorie Action et de durée 144 sorti en 1994/01/01
```