Tuto PyTest :

Installer la librairie PyTest.

Créer un fichier fic1.py

1. Taper :

**def** capital\_case(x):  
 **return** x.capitalize()  
  
**def** test\_capital\_case():  
 **assert** capital\_case(**'bonjour'**) == **'Bonjour'**

Les tests sont des méthodes qui commencent par test\_ ou finissent par \_test

Pour voir le résultat du test, on tape dans la console :  
pytest fic1.py

Faire le même test en provoquant une erreur.

1. Rajouter dans le fichier

**import** pytest

**def** test\_raises\_exception\_on\_non\_string\_arguments():  
 **with** pytest.raises(TypeError):  
 capital\_case(9)

Vérifier le résultat

Du coup on va modifier notre fonction pour éviter l’erreur :

**def** capital\_case(x):  
 **if not** isinstance(x, str):  
 **raise** TypeError(**'Please provide a string argument'**)  
 **return** x.capitalize()

1. On va organiser notre travail proprement en laissant notre code dans fic1.py et tous les tests dans testFic1.py

fic1.py :

**def** capital\_case(x):  
 **if not** isinstance(x, str):   
 **raise** TypeError(**'Please provide a string argument'**)  
 **return** x.capitalize()

testFic1.py :  
  
**import** pytest  
**import** fic1  
  
**def** test\_capital\_case():  
 **assert** fic1.capital\_case(**'bonjour'**) == **'Bonjour'  
  
def** test\_raises\_exception\_on\_non\_string\_arguments():  
 **with** pytest.raises(TypeError):  
 fic1.capital\_case(9)

Puis saisir la commande : pytest testFic1.py

Travailler avec une classe :

Nous voulons créer une classe pour gérer un portefeuille :

Voici les tests à passer dans le fichier testWallet.py :

**import** pytest  
**from** wallet **import** Wallet, InsufficientAmount  
  
  
**def** test\_default\_initial\_amount():  
 wallet = Wallet()  
 **assert** wallet.balance == 0  
  
**def** test\_setting\_initial\_amount():  
 wallet = Wallet(100)  
 **assert** wallet.balance == 100  
  
**def** test\_wallet\_add\_cash():  
 wallet = Wallet(10)  
 wallet.add\_cash(90)  
 **assert** wallet.balance == 100  
  
**def** test\_wallet\_spend\_cash():  
 wallet = Wallet(20)  
 wallet.spend\_cash(10)  
 **assert** wallet.balance == 10  
  
**def** test\_wallet\_spend\_cash\_raises\_exception\_on\_insufficient\_amount():  
 wallet = Wallet()  
 **with** pytest.raises(InsufficientAmount):  
 wallet.spend\_cash(100)

Pour cela nous allons devoir coder les classes suivantes dans le fichier wallet.py :

**class** InsufficientAmount(Exception):  
 **pass  
  
class** Wallet(object):  
  
 **def** \_\_init\_\_(self, initial\_amount=0):  
 self.balance = initial\_amount  
  
 **def** spend\_cash(self, amount):  
 **if** self.balance < amount:  
 **raise** InsufficientAmount(**'Not enough available to spend {}'**.format(amount))  
 self.balance -= amount  
  
 **def** add\_cash(self, amount):  
 self.balance += amount

Nous pouvons réduire le code grâce aux fixtures dans le fichier testWalletFixtures.py:

@pytest.fixture  
**def** wallet():  
 *'''Returns a Wallet instance with a balance of 20'''* **return** Wallet(20)  
  
**def** test\_default\_initial\_amount(empty\_wallet):  
 **assert** empty\_wallet.balance == 0  
  
**def** test\_setting\_initial\_amount(wallet):  
 **assert** wallet.balance == 20  
  
**def** test\_wallet\_add\_cash(wallet):  
 wallet.add\_cash(80)  
 **assert** wallet.balance == 100  
  
**def** test\_wallet\_spend\_cash(wallet):  
 wallet.spend\_cash(10)  
 **assert** wallet.balance == 10  
  
**def** test\_wallet\_spend\_cash\_raises\_exception\_on\_insufficient\_amount(empty\_wallet):  
 **with** pytest.raises(InsufficientAmount):  
 empty\_wallet.spend\_cash(100)

Donnons des listes de paramètres à nos tests pour les multiplier en rajoutant :

@pytest.mark.parametrize(**"earned,spent,expected"**, [  
 (30, 10, 20),  
 (20, 2, 18),  
])  
**def** test\_transactions(empty\_wallet, earned, spent, expected):  
 empty\_wallet.add\_cash(earned)  
 empty\_wallet.spend\_cash(spent)  
 **assert** empty\_wallet.balance == expected