# TP - Introduction aux Tests Unitaires

## Objectifs

- Comprendre l'importance des tests dans le développement
- Écrire des fonctions simples en Python
- Créer des tests unitaires avec `unittest`
- Tester des cas positifs et négatifs
- Corriger les erreurs découvertes par les tests

## Prérequis

- Python 3.6+ installé
- Un éditeur de texte (VS Code, PyCharm, etc.)
- Modules `unittest` (inclus avec Python) ou `pytest` (à installer avec `pip install pytest`)

## Structure du projet

Créez un dossier `tp_tests` avec cette structure :

```
tp_tests/
├── fonctions.py      # Nos fonctions à tester
└── test_fonctions.py # Nos tests unitaires
```

---

## Partie 1 : Création des fonctions à tester

### Étape 1 : Créer le fichier `fonctions.py`

Créez le fichier `fonctions.py` et copiez le code suivant :

```python
def additionner(a, b):
    """Additionne deux nombres"""
    return a + b

def est_pair(nombre):
    """Vérifie si un nombre est pair"""
    return nombre % 2 == 0

def valider_email(email):
    """Valide un email simple (doit contenir @ et .)"""
    if "@" not in email:
        return False
    if "." not in email:
        return False
    return True
```

```
def calculer_moyenne(notes):
    """Calcule la moyenne d'une liste de notes"""
    if len(notes) == 0:
        return 0
    return sum(notes) / len(notes)

def convertir_temperature(celsius):
    """Convertit des degrés Celsius en Fahrenheit"""
    return (celsius * 9/5) + 32
```

---

## Partie 2 : Écriture des tests (avec unittest)

### Étape 2a : Créer le fichier `test_fonctions.py` avec unittest

Créez le fichier `test_fonctions.py` et ajoutez ce code de base :

```
import unittest
from fonctions import additionner, est_pair, valider_email, calculer_moyenne,
convertir_temperature

class TestFonctions(unittest.TestCase):

    def test_additionner_cas_positif(self):
        """Test addition avec nombres positifs"""
        resultat = additionner(2, 3)
        self.assertEqual(resultat, 5)

    def test_additionner_cas_negatif(self):
        """Test addition avec nombres négatifs"""
        resultat = additionner(-2, -3)
        self.assertEqual(resultat, -5)

    # À COMPLÉTER : Ajoutez vos tests ici

# Permet d'exécuter les tests
if __name__ == '__main__':
    unittest.main()
```

### Alternative avec pytest

Si vous préférez pytest, créez plutôt ce fichier :

```
from fonctions import additionner, est_pair, valider_email, calculer_moyenne,
convertir_temperature

def test_additionner_cas_positif():
    """Test addition avec nombres positifs"""
    resultat = additionner(2, 3)
```

```
    assert resultat == 5

def test_additionner_cas_negatif():
    """Test addition avec nombres négatifs"""
    resultat = additionner(-2, -3)
    assert resultat == -5

# À COMPLÉTER : Ajoutez vos tests ici
```

## Étape 3 : Tester la fonction `est_pair`

**Avec unittest :**

```
def test_est_pair_nombre_pair(self):
    """Test avec un nombre pair"""
    self.assertTrue(est_pair(4))

def test_est_pair_nombre_impair(self):
    """Test avec un nombre impair"""
    self.assertFalse(est_pair(3))

def test_est_pair_zero(self):
    """Test avec zéro"""
    self.assertTrue(est_pair(0))
```

**Avec pytest :**

```
def test_est_pair_nombre_pair():
    """Test avec un nombre pair"""
    assert est_pair(4) == True

def test_est_pair_nombre_impair():
    """Test avec un nombre impair"""
    assert est_pair(3) == False

def test_est_pair_zero():
    """Test avec zéro"""
    assert est_pair(0) == True
```

## Étape 4 : Tester la fonction `valider_email`

**Avec unittest :**

```
def test_valider_email_valide(self):
    """Test avec un email valide"""
    self.assertTrue(valider_email("test@example.com"))
```

```
    def test_valider_email_sans_arobase(self):
        """Test avec un email sans @"""
        self.assertFalse(valider_email("testexample.com"))

    def test_valider_email_sans_point(self):
        """Test avec un email sans point"""
        self.assertFalse(valider_email("test@example"))
```

**Avec pytest :**

```
def test_valider_email_valide():
    """Test avec un email valide"""
    assert valider_email("test@example.com") == True

def test_valider_email_sans_arobase():
    """Test avec un email sans @"""
    assert valider_email("testexample.com") == False

def test_valider_email_sans_point():
    """Test avec un email sans point"""
    assert valider_email("test@example") == False
```

---

## Partie 3 : Lancer les tests

### Étape 5 : Exécuter les tests

**Avec unittest :**

```
python test_fonctions.py
```

**Avec pytest :**

```
pytest test_fonctions.py
```

Vous devriez voir quelque chose comme :

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
```

---

# Partie 4 : Exercices à compléter

## Exercice 1 : Tester `calculer_moyenne`

Ajoutez ces tests pour la fonction `calculer_moyenne` :

**Avec unittest :**

```
def test_calculer_moyenne_liste_normale(self):
    """Test avec une liste de notes normales"""
    # TODO: Testez avec [10, 15, 20] - résultat attendu : 15
    # Utilisez self.assertEqual(resultat, valeur_attendue)
    pass

def test_calculer_moyenne_liste_vide(self):
    """Test avec une liste vide"""
    # TODO: Testez avec [] - résultat attendu : 0
    pass

def test_calculer_moyenne_une_note(self):
    """Test avec une seule note"""
    # TODO: Testez avec [18] - résultat attendu : 18
    pass
```

**Avec pytest :**

```
def test_calculer_moyenne_liste_normale():
    """Test avec une liste de notes normales"""
    # TODO: Testez avec [10, 15, 20] - résultat attendu : 15
    # Utilisez assert resultat == valeur_attendue
    pass

def test_calculer_moyenne_liste_vide():
    """Test avec une liste vide"""
    # TODO: Testez avec [] - résultat attendu : 0
    pass

def test_calculer_moyenne_une_note():
    """Test avec une seule note"""
    # TODO: Testez avec [18] - résultat attendu : 18
    pass
```

## Exercice 2 : Tester `convertir_temperature`

Ajoutez ces tests :

**Avec unittest :**

```
    def test_convertir_temperature_zero(self):
        """Test conversion 0°C = 32°F"""
        # TODO: Testez la conversion de 0°C
        pass

    def test_convertir_temperature_eau_bouillante(self):
        """Test conversion 100°C = 212°F"""
        # TODO: Testez la conversion de 100°C
        pass
```

**Avec pytest :**

```
def test_convertir_temperature_zero():
    """Test conversion 0°C = 32°F"""
    # TODO: Testez la conversion de 0°C
    pass

def test_convertir_temperature_eau_bouillante():
    """Test conversion 100°C = 212°F"""
    # TODO: Testez la conversion de 100°C
    pass
```

---

# Partie 5 : Débogage

## Étape 6 : Introduire une erreur volontaire

Modifiez la fonction `additionner` dans `fonctions.py` :

```
def additionner(a, b):
    """Additionne deux nombres"""
    return a * b  # ERREUR VOLONTAIRE : * au lieu de +
```

Relancez les tests. Vous devriez voir une erreur :

```
FAIL: test_additionner_cas_positif
AssertionError: 6 != 5
```

## Étape 7 : Corriger l'erreur

Remettez le `+` dans la fonction `additionner` et relancez les tests.

---

# Questions de réflexion

1. **Pourquoi écrire des tests ?**
    - Les tests permettent de vérifier que notre code fonctionne correctement
    - Ils nous alertent quand nous introduisons des erreurs
    - Ils documentent le comportement attendu du code
2. **Cas positifs vs négatifs :**
    - **Cas positifs** : Testent le comportement normal (email valide, nombres positifs)
    - **Cas négatifs** : Testent les cas d'erreur (email invalide, liste vide)
3. **Que teste-t-on ?**
    - Les entrées normales
    - Les cas limites (0, liste vide)
    - Les entrées invalides

---

## Pour aller plus loin

Essayez d'ajouter :

- Une fonction `diviser(a, b)` qui gère la division par zéro
- Des tests pour vérifier qu'une exception est levée avec `self.assertRaises()` (unittest) ou `pytest.raises()` (pytest)
- Une fonction de validation de mot de passe avec plusieurs critères

**Bon travail ! Vous avez écrit vos premiers tests unitaires !** 🎉