TP Séance 1 : Créer des fonctions simples et leurs tests

> **Date :** Séance 1 - Méthodologies de tests et tests unitaires
> **Modalité :** Travail individuel avec validation par binômes

📋 Objectifs pédagogiques

À la fin de ce TP, vous devrez être capable de :

- Implémenter des fonctions Python respectant des contraintes précises
- Écrire des tests unitaires couvrant cas positifs, négatifs et limites
- Diagnostiquer et corriger des erreurs de tests
- Utiliser pytest efficacement et mesurer la couverture de code

🛠️ Setup initial

Installation et vérification

```
# 1. Créer environnement virtuel
python -m venv venv_tests
source venv_tests/bin/activate  # Linux/Mac
# ou venv_tests\Scripts\activate  # Windows

# 2. Installer dépendances
pip install pytest pytest-cov

# 3. Créer structure projet
mkdir tp_seance1 && cd tp_seance1
mkdir -p src tests
touch src/__init__.py tests/__init__.py
touch src/{calculator,validator,string_utils}.py
touch tests/test_{calculator,validator,string_utils}.py
```

📋 **Livrable 1** : Capture d'écran de l'arborescence créée

🧮 Exercice 1 : Calculatrice robuste

Spécifications

Créer une calculatrice financière avec gestion d'erreurs stricte :

- **add(a, b)** : Addition avec validation de types
- **divide(a, b)** : Division avec gestion division par zéro
- **power(base, exponent)** : Puissance avec cas limites (0^négatif)

Implémentation

Fichier : `src/calculator.py`

```
def add(a, b):
    """
    Additionne deux nombres.

    Raises:
        TypeError: Si un argument n'est pas un nombre
    """
    # TODO: Valider les types avec isinstance(a, (int, float))
    # TODO: Lever TypeError avec message explicite
    # TODO: Retourner a + b
    pass

def divide(a, b):
    """
    Divise a par b.

    Returns:
        float: Quotient (toujours en float)

    Raises:
        TypeError: Si un argument n'est pas un nombre
        ZeroDivisionError: Si b est égal à 0
    """
    # TODO: Valider les types
    # TODO: Vérifier b != 0, lever ZeroDivisionError
    # TODO: Retourner float(a / b)
    pass

def power(base, exponent):
    """
    Calcule base^exponent.

    Raises:
        ValueError: Si base=0 et exponent<0
    """
    # TODO: Valider les types
    # TODO: Gérer cas 0^(négatif) -> ValueError
    # TODO: Retourner base ** exponent
    pass
```

Tests obligatoires

Fichier : `tests/test_calculator.py`

```
import pytest
from src.calculator import add, divide, power

class TestAddition:
```

```
    def test_add_integers(self):
        assert add(5, 3) == 8

    def test_add_floats(self):
        assert add(2.5, 3.7) == pytest.approx(6.2)

    def test_add_avec_zero(self):
        assert add(0, 5) == 5
        assert add(5, 0) == 5

    def test_add_negatifs(self):
        assert add(-5, -3) == -8
        assert add(10, -7) == 3

    def test_add_type_invalide(self):
        with pytest.raises(TypeError):
            add("5", 3)
        with pytest.raises(TypeError):
            add(5, "3")

class TestDivision:
    def test_divide_entiers(self):
        assert divide(10, 2) == 5.0

    def test_divide_decimal(self):
        assert divide(7, 3) == pytest.approx(2.333333)

    def test_divide_par_zero(self):
        with pytest.raises(ZeroDivisionError):
            divide(10, 0)

    def test_divide_zero_dividende(self):
        assert divide(0, 5) == 0.0

class TestPower:
    def test_power_basique(self):
        assert power(2, 3) == 8
        assert power(5, 2) == 25

    def test_power_exposant_zero(self):
        assert power(5, 0) == 1
        assert power(0, 0) == 1

    def test_power_zero_exposant_negatif(self):
        with pytest.raises(ValueError):
            power(0, -1)
```

Validation : `pytest tests/test_calculator.py --cov=src.calculator`

📧 Exercice 2 : Validateur d'email et mot de passe

Spécifications strictes

is_valid_email(email)

- Exactement un `@`
- Partie locale : 1-64 caractères alphanumériques/points/tirets
- Domaine avec extension 2-4 lettres

validate_password_strength(password)

Retourne un dictionnaire avec :

- `is_valid` : bool (True si score >= 4)
- `score` : int (0-5)
- `missing_criteria` : list des critères manquants

Critères (1 point chacun) :

- Longueur >= 8
- Au moins 1 majuscule
- Au moins 1 minuscule
- Au moins 1 chiffre
- Au moins 1 caractère spécial (!@#$%^&*)

Implémentation

Fichier : `src/validator.py`

```
import re

def is_valid_email(email):
    """Valide une adresse email."""
    if not isinstance(email, str):
        raise TypeError("L'email doit être une chaîne")

    # TODO: Vérifier qu'il y a exactement un @
    # TODO: Séparer en partie locale et domaine
    # TODO: Valider partie locale (regex: ^[a-zA-Z0-9._-]+$)
    # TODO: Valider domaine (regex: ^[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$)
    pass

def validate_password_strength(password):
    """Évalue la force d'un mot de passe."""
    if not isinstance(password, str):
        raise TypeError("Le mot de passe doit être une chaîne")

    result = {
        'is_valid': False,
        'score': 0,
        'missing_criteria': []
    }
```

```
    # TODO: Vérifier chaque critère et incrémenter score
    # TODO: Ajouter critères manquants à la liste
    # TODO: is_valid = True si score >= 4

    return result
```

Tests obligatoires

Fichier : `tests/test_validator.py`

```
import pytest
from src.validator import is_valid_email, validate_password_strength

class TestEmailValidation:
    def test_emails_valides(self):
        valides = ["user@example.com", "test.email@domain.org"]
        for email in valides:
            assert is_valid_email(email) == True

    def test_emails_invalides(self):
        invalides = ["user", "user@", "@domain.com", "user@domain"]
        for email in invalides:
            assert is_valid_email(email) == False

    def test_type_invalide(self):
        with pytest.raises(TypeError):
            is_valid_email(123)

class TestPasswordStrength:
    def test_password_fort(self):
        result = validate_password_strength("MyStrong123!")
        assert result['is_valid'] == True
        assert result['score'] >= 4

    def test_password_faible(self):
        result = validate_password_strength("weak")
        assert result['is_valid'] == False
        assert len(result['missing_criteria']) > 0
```

Exercice 3 : Utilitaires de chaînes

Implémentation

Fichier : `src/string_utils.py`

```
def capitalize_words(text):
    """Met en majuscule la première lettre de chaque mot."""
```

```
    if not isinstance(text, str):
        raise TypeError("L'argument doit être une chaîne")
    # TODO: Utiliser title() ou split/join avec capitalize()
    pass

def count_words(text):
    """Compte le nombre de mots."""
    if not isinstance(text, str):
        raise TypeError("L'argument doit être une chaîne")
    # TODO: Utiliser split() et len(), gérer chaîne vide
    pass

def remove_vowels(text):
    """Supprime toutes les voyelles."""
    if not isinstance(text, str):
        raise TypeError("L'argument doit être une chaîne")
    # TODO: Utiliser une boucle ou regex pour supprimer a,e,i,o,u,y
    pass
```

Tests obligatoires

Fichier : `tests/test_string_utils.py`

```
import pytest
from src.string_utils import capitalize_words, count_words, remove_vowels

class TestStringUtils:
    def test_capitalize_mots_multiples(self):
        assert capitalize_words("hello world") == "Hello World"

    def test_count_words_basique(self):
        assert count_words("hello world python") == 3
        assert count_words("") == 0

    def test_remove_vowels_basique(self):
        assert remove_vowels("hello") == "hll"
        assert remove_vowels("python") == "pythn"
```

Livrables finaux

Validation complète

```
# Tous les tests doivent passer
pytest tests/ -v

# Couverture minimum 90%
pytest tests/ --cov=src --cov-report=html
```

```
# Génération rapport
pytest tests/ --cov=src --cov-report=term-missing
```

À remettre

1. **Code source complet** dans l'arborescence demandée
2. **Capture d'écran** de tous les tests passants
3. **Rapport de couverture** HTML
4. **Liste des difficultés** rencontrées avec solutions

Critères d'évaluation

- ☑ **Toutes les fonctions implémentées** (30 pts)
- ☑ **Tests positifs réussis** (25 pts)
- ☑ **Tests négatifs réussis** (25 pts)
- ☑ **Gestion des exceptions** (15 pts)
- ☑ **Couverture > 90%** (5 pts)

Commandes finales de validation

```
# Structure projet
find . -name "*.py" | sort

# Exécution complète
pytest tests/ -v --tb=short

# Statistiques
pytest tests/ --co -q
```

Bonne chance ! 🚀