 

TP1 – Tests Unitaires

**Année universitaire :** 2024–2025

**Enseignant :** Arij Azzabi

**Table des matières**

1. [Contexte du TP](#_bookmark0) 2
2. [Étape 1 – Création du projet de test](#_bookmark1) 2
3. [Étape 2 – Implémentation des fonctions à tester](#_bookmark2) 2
4. [Étape 3 – Écriture des tests unitaires](#_bookmark3) 2
5. [Étape 4 – Exécution des tests](#_bookmark4) 3
6. [Étape 5 – Ajout d’un cas d’erreur ou de test inattendu](#_bookmark5) 3
7. [À rendre](#_bookmark6) 4

# Contexte du TP

Vous travaillez sur une bibliothèque logicielle contenant trois fonctions. Afin de valider leur bon fonctionnement, vous allez les tester individuellement avec un framework de test **unitaire**.

Vous avez le choix entre deux langages :

## Python + PyTest

## Java + JUnit

Indiquez ici votre choix : **Java + JUnit**

# Étape 1 – Création du projet de test

Instructions

1. Crée un dossier tp1-tests-unitaires.
2. Dans ce dossier, crée un fichier fonctions.py ou Fonctions.java.
3. Implémente les 3 fonctions suivantes (voir section suivante).
4. Crée un fichier de test : test\_fonctions.py ou FonctionsTest.java.

# Étape 2 – Implémentation des fonctions à tester

Implémente les fonctions suivantes dans ton fichier source.

1. **calculer\_ttc(prix\_ht, tva)** Retourne le prix TTC selon la formule : prix\_ht + prix\_ht \* tva
2. **verifier\_mot\_de\_passe(mot\_de\_passe)** Retourne True si le mot de passe respecte : au moins 8 caractères, une majuscule, un chiffre.
3. **convertir\_date(date\_fr)** Reçoit une date au format JJ/MM/AAAA et la convertit en AAAA-MM-JJ

# Étape 3 – Écriture des tests unitaires

Pour chaque fonction, écris **au moins 2 tests** : un cas classique, un cas limite.

Exemple PyTest

from fonctions import calculer\_ttc

def test\_calculer\_ttc\_normal():

assert calculer\_ttc(100, 0.2) == 120

def test\_calculer\_ttc\_zero():

assert calculer\_ttc(0, 0.2) == 0

Insère ici un test que tu as écrit toi-même :

# Étape 4 – Exécution des tests

* + En Python, lance : pytest test\_fonctions.py
  + En Java (JUnit), exécute la classe de test avec ton IDE

Note ici le résultat obtenu :

**=================test session starts=========================**

**platform win32 -- Python 3.13.1, pytest-8.3.5, pluggy-1.5.0**

**rootdir: C:\Users\bchaouki\_peaks\development\tp1-tests-unitaires**

**plugins: anyio-4.8.0**

**collected 5 items**

**test\_fonctions.py ..... [100%]**

**=================5 passed in 0.05s =========================**

As-tu des tests qui échouent ? **Non**

Si oui, quelles hypothèses fais-tu pour expliquer ces erreurs ?

**X**

# Étape 5 – Ajout d’un cas d’erreur ou de test inattendu

Objectif de cette étape

Les tests classiques sont importants, mais il est tout aussi important de prévoir les cas inattendus ou erronés. Dans cette étape, tu vas :

* Choisir une des 3 fonctions.
* Imaginer une utilisation incorrecte ou extrême de cette fonction.
* Écrire un test unitaire pour ce cas.
* Observer si la fonction actuelle gère bien ce comportement.

**Fonction choisie :** **verifier\_mot\_de\_passe**

**Cas de test imaginé :** **Mot de passe vide.**

## Code du test correspondant :

## def test\_verifier\_mot\_de\_passe\_vide():

## """Test avec un mot de passe vide"""

## assert verifier\_mot\_de\_passe("") == False

## Résultat obtenu (succès) : ============test session starts===================

## platform win32 -- Python 3.13.1, pytest-8.3.5, pluggy-1.5.0

## rootdir: C:\Users\bchaouki\_peaks\development\tp1-tests-unitaires

## plugins: anyio-4.8.0

## collected 6 items / 5 deselected / 1 selected

## test\_fonctions.py . [100%]

## ============1 passed, 5 deselected in 0.04s=============

## Ton interprétation :

## Comme mon test « test\_verifier\_mot\_de\_passe\_vide » est passé avec succès, cela signifie que la fonction « verifier\_mot\_de\_passe » gère correctement le cas d'un mot de passe vide. En d'autres termes, la fonction a bien renvoyé False lorsqu'un mot de passe vide a été fourni, ce qui correspond à l'attendu. Cela prouve que mon code prend en compte ce scénario comme prévu.

# À rendre

* + fonctions.py ou Fonctions.java
  + test\_fonctions.py ou FonctionsTest.java
  + Une capture des tests réussis