🔹 PARTIE 1 – Compréhension (savoir ce qu'ils font)

Exercice 1 – Docstring

Explique à quoi sert une **docstring** et où elle peut être placée dans un programme Python.

1. À quoi sert une docstring ?
2. Où peut-on en écrire ?
3. Comment afficher une docstring dans Python ?

Exercice 2 – Assert

Observe le code suivant :

```
def racine(x):
    assert x >= 0, "x doit être positif"
    return x ** 0.5
```

1. À quoi sert la ligne avec `assert` ?
2. Que se passe-t-il si `x = -4` ?

Exercice 3 – Doctest

Observe cette docstring :

```
def carre(x):
    """
    >>> carre(3)
    9
    >>> carre(-2)
    4
    """
    return x * x
```

1. À quoi servent les lignes commençant par `>>>` ?
2. Que fait le module `doctest` ?

🔹 PARTIE 2 – Analyse critique (dire si c'est bien écrit)

Exercice 4 – Docstring correcte ou non ?

Pour chaque docstring, indique si elle est **correcte**, **incomplète** ou **mal écrite**, et explique pourquoi.

A

```
def addition(a, b):
    """Addition."""
    return a + b
```

B

```
def division(a, b):
    """
    Divise a par b.

    Args:
        a (float): numérateur
        b (float): dénominateur

    Returns:
        float: résultat
    """
    return a / b
```

C

```
def moyenne(liste):
    # calcule la moyenne
    return sum(liste) / len(liste)
```

Exercice 5 – Assert bien utilisé ?

Indique si l'utilisation de `assert` est **appropriée ou non**.

A

```
def surface_carre(c):
    assert c > 0
    return c * c
```

B

```
age = int(input("Âge : "))
assert age >= 0
print("Bienvenue")
```

C

```
def moyenne(notes):
    assert len(notes) > 0
    return sum(notes) / len(notes)
```

Exercice 6 – Doctest valide ?

Dis si les doctests suivants fonctionneront correctement.

A

```
def cube(x):
    """
    >>> cube(2)
    8
    """
    return x ** 3
```

B

```
def division(a, b):
    """
    >>> division(1, 3)
    0.3333333333333333
    """
    return a / b
```

C

```
def affiche():
    """
    >>> print("a\nb")
```

```
    a
    b
    """
    print("a\nb")
```

🔹 PARTIE 3 – Rédaction guidée

Exercice 7 – Écrire une docstring

Complète la fonction suivante en ajoutant une **docstring claire et complète** :

```
def perimetre_rectangle(longueur, largeur):
    return 2 * (longueur + largeur)
```

La docstring doit préciser :

- le rôle de la fonction
- les paramètres
- la valeur de retour

Exercice 8 – Ajouter des assert

Ajoute des `assert` pour vérifier que :

- n est un entier
- n est positif ou nul

```
def factorielle(n):
    res = 1
    for i in range(2, n + 1):
        res *= i
    return res
```

Exercice 9 – Écrire un doctest simple

Ajoute un doctest à cette fonction pour tester **au moins deux cas** :

```
def est_pair(n):
    return n % 2 == 0
```

🔹 PARTIE 4 – Exercices complets (docstring + assert + doctest)

Exercice 10 – Fonction complète

Complète entièrement la fonction suivante :

```
def maximum(a, b):
    pass
```

Contraintes :

- docstring avec description + exemples doctest
- assert pour vérifier que a et b sont des nombres
- renvoyer le plus grand des deux

Exercice 11 – Gestion d'erreur + doctest

Écris une fonction `inverse(x)` qui :

- renvoie `1 / x`
- lève une erreur si `x == 0`
- contient un doctest vérifiant le bon fonctionnement et l'erreur

---

Exercice 12 – Lecture et correction

Le code suivant contient **au moins 3 erreurs ou mauvaises pratiques**. Identifie-les et propose une correction.

```
def racine(x):
    """
    >>> racine(-4)
    2
    """
    assert x >= 0
    return x ** 0.5
```