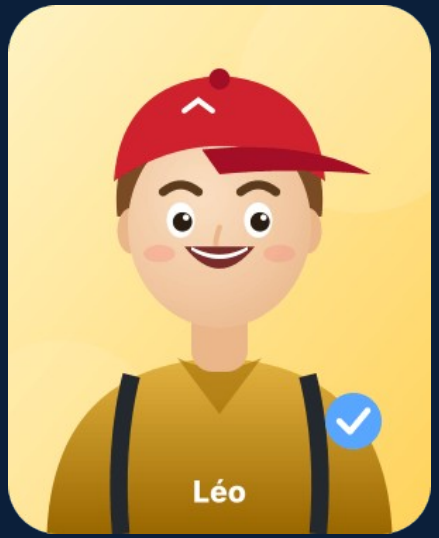
HEF-010 · MISSION 08 / 10

Correction, tests et cas limites

Un algorithme qui marche « en général » ne marche pas. Il doit marcher partout, ou dire clairement quand il ne marche pas.





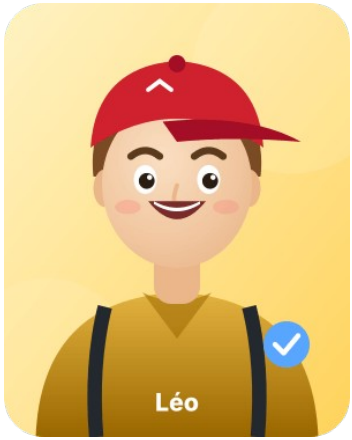
Badge : Gardien·ne de la qualité

70 min · débutant

L'histoire

Vendredi soir. Kodi livre sa fonction de recherche. Samedi matin, un utilisateur dont le nom contient un accent fait planter le site. Dimanche, une liste vide aussi.

> *« Ton code marche pour les exemples que tu as imaginés, dit Sam. Maintenant, on va imaginer ceux que tu n'as pas imaginés. »*



Dans cette mission : prouver, tester, et casser son propre code avant que d'autres le fassent.

Ta mission

À la fin, je serai capable de…

1. écrire un test automatique simple avec `assert`
2. lister des cas limites systématiquement
3. distinguer test, preuve et démonstration
4. trouver un contre-exemple à une fonction fausse
5. écrire une fonction avec ses tests

Défi d'échauffement

Défi d'échauffement

Voici une fonction censée renvoyer le deuxième plus grand nombre d'une liste :

```
def deuxieme(liste):
    liste.sort()
    return liste[-2]
```

Trouve au moins 3 entrées pour lesquelles elle plante ou donne un résultat discutable. Écris-les.

Le concept

Un test, c'est quoi ?

Un **test** est un petit programme qui vérifie qu'une fonction donne le résultat attendu sur un cas précis.

```
assert plus_grand([3, 9, 2]) == 9
assert plus_grand([5]) == 5
```

Si une condition est fausse, `assert` arrête le programme avec une erreur : le test a échoué.

Les cas limites

Cherche toujours ces familles :

Famille	Exemples
Vide / minimal	liste vide, un seul élément

Le concept

Les cas limites (suite)

Famille	Exemples
Extrêmes	très grand nombre, 0, négatifs
Doublons	plusieurs fois la même valeur
Déjà dans l'état final	liste déjà triée
Ordre inversé	pire cas pour un tri
Types inattendus	texte à la place d'un nombre

Tester ne prouve pas

Un test montre qu'il y a un bug, jamais qu'il n'y en a plus. Pour être sûr, on combine : tests variés, **invariant** (vu en mission 03), **contrat** (préconditions) et relecture.

Le concept

Les tests aléatoires comparatifs

On compare notre fonction à une fonction de référence (comme `sorted`) sur beaucoup d'entrées aléatoires :

```
import random
for _ in range(1000):
    donnees = [random.randint(-50, 50) for _ in range(random.randint(0, 10))]
    assert tri_insertion(donnees) == sorted(donnees)
```

🧠 ***Astuce*** *: écris le test* ***avant*** *la fonction. Ça force à définir précisément ce qu'elle doit faire.*

Exemple pas à pas

Exemple pas à pas

Fonction à corriger :

```
def deuxieme(liste):
    liste.sort()
    return liste[-2]
```

Cas limites trouvés :

Entrée	Problème
`[]` ou `[5]`	`IndexError`, il n'y a pas de deuxième
`[9, 9, 3]`	renvoie 9 : le deuxième plus grand est-il 9 ou 3 ? À décider !
`[3, 1, 2]`	modifie la liste d'origine (effet de bord)

Décision : « deuxième plus grand valeur distincte ». Renvoie `None` s'il n'existe pas.

Exemple pas à pas (suite)

```
def deuxieme(liste):
    valeurs = sorted(set(liste))
    if len(valeurs) < 2:
        return None
    return valeurs[-2]

assert deuxieme([]) is None
assert deuxieme([5]) is None
assert deuxieme([9, 9, 3]) == 3
liste = [3, 1, 2]
assert deuxieme(liste) == 2
assert liste == [3, 1, 2]   # la liste d'origine n'est pas modifiée
print("tous les tests passent")
```

À toi, avec aide

Voici une fonction censée compter les voyelles d'un texte :

```
def compter_voyelles(texte):
    return sum(1 for c in texte if c in "aeiou")
```

1. Trouve 3 cas qui montrent un problème.
2. Corrige-la.
3. Écris 4 tests.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Essaie des majuscules : `"AIE"`.

Indice 2

Essaie des lettres accentuées : `"é"`, `"à"`. Faut-il les compter ?

Indice 3

Décide la règle (par exemple : insensible à la casse, accents comptés) avant de corriger.

À toi, avec aide · réponse

Réponse

Problèmes : majuscules non comptées, accents ignorés, y ? (à décider).

```
def compter_voyelles(texte):
    voyelles = "aeiouyàâäéèêëîïôöùûüœ"
    return sum(1 for c in texte.lower() if c in voyelles)

assert compter_voyelles("") == 0
assert compter_voyelles("AIE") == 3
assert compter_voyelles("été") == 2
assert compter_voyelles("rythme") == 2  # y et e
print("ok")
```

La règle (y compté, accents inclus) est **un choix**, il doit être écrit dans le contrat.

À toi, en solo

Voici une fonction censée renvoyer l'indice du minimum :

```
def indice_min(liste):
    mini = 0
    for i in range(1, len(liste)):
        if liste[i] < liste[mini]:
            mini = i
    return mini
```

1. Quel est le comportement avec une liste vide ? Avec des doublons ?
2. Écris un contrat clair.
3. Ajoute une précondition `assert` et au moins 5 tests, dont un test aléatoire comparé à `liste.index(min(liste))`.

À toi, en solo · réponse

Pistes de correction

```
import random

def indice_min(liste):
    """Indice de la première occurrence du minimum. Exige: liste non vide."""
    assert len(liste) > 0, "liste non vide"
    mini = 0
    for i in range(1, len(liste)):
        if liste[i] < liste[mini]:
            mini = i
    return mini

assert indice_min([4]) == 0
assert indice_min([3, 1, 2]) == 1
assert indice_min([2, 1, 1]) == 1       # doublons : premier minimum
assert indice_min([-5, 0, -5]) == 0
for _ in range(500):
    donnees = [random.randint(-9, 9) for _ in range(random.randint(1, 8))]
    assert indice_min(donnees) == donnees.index(min(donnees))
print("ok")
```

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Tester seulement le cas heureux	Les bugs sont dans les cas limites	Lister les familles de cas limites
Tests qui dépendent les uns des autres	Un échec en cache un autre	Tests indépendants et petits
Modifier l'entrée sans le dire	Effet de bord inattendu	Le tester explicitement
Croire qu'un test prouve l'absence de bug	Fausse confiance	Combiner contrat, invariant et tests variés

Mini-boss

Mini-boss

Choisis **une fonction** de ce module (par exemple `chercher_binaire`) et construis son dossier qualité :

1. Contrat (préconditions, postconditions).
2. Liste d'au moins 8 cas limites avec le résultat attendu.
3. Les tests `assert` correspondants.
4. Un test aléatoire contre une référence.
5. Un bug que tu injectes volontairement, et la façon dont un test l'attrape.

Mini-boss · réponse

Pistes de correction

Pour `chercher_binaire` : cas limites = liste vide, 1 élément (présent/absent), cible plus petite que tout, plus grande que tout, doublons, taille paire/impaire, liste non triée (précondition). Test aléatoire : comparer `chercher_binaire(l, x) != -1` avec `x in l` sur des listes triées. Bug injecté : remplacer `<=` par `<` dans `while bas <= haut` fait échouer le test de la liste à un élément.

Quiz éclair

Quiz éclair

1. Un test montre :

A. qu'il n'y a jamais de bug
B. la vitesse
C. qu'il y a un bug, s'il échoue

2. Un cas limite typique :

A. la liste vide
B. la liste moyenne
C. le cas heureux

3. `assert` sert à :

A. afficher un texte

Quiz éclair

Quiz éclair (suite)

B. arrêter le programme si une condition est fausse
C. trier une liste

4. 🔁 *Révision mission 07* : Le glouton :

A. est toujours optimal
B. est toujours lent
C. n'est pas toujours optimal

5. 🔁 *Révision mission 03* : Une précondition :

A. ce que l'appelant doit fournir
B. la valeur de retour
C. un commentaire

Quiz éclair · réponse

Réponses

1. **C**.
2. **A**.
3. **B**.
4. **C**.
5. **A**.

Je réfléchis

Je réfléchis

1. Quel cas limite aurais-tu oublié sans cette mission ?
2. Dans ta vie, où un « ça marche d'habitude » t'a-t-il posé problème ?

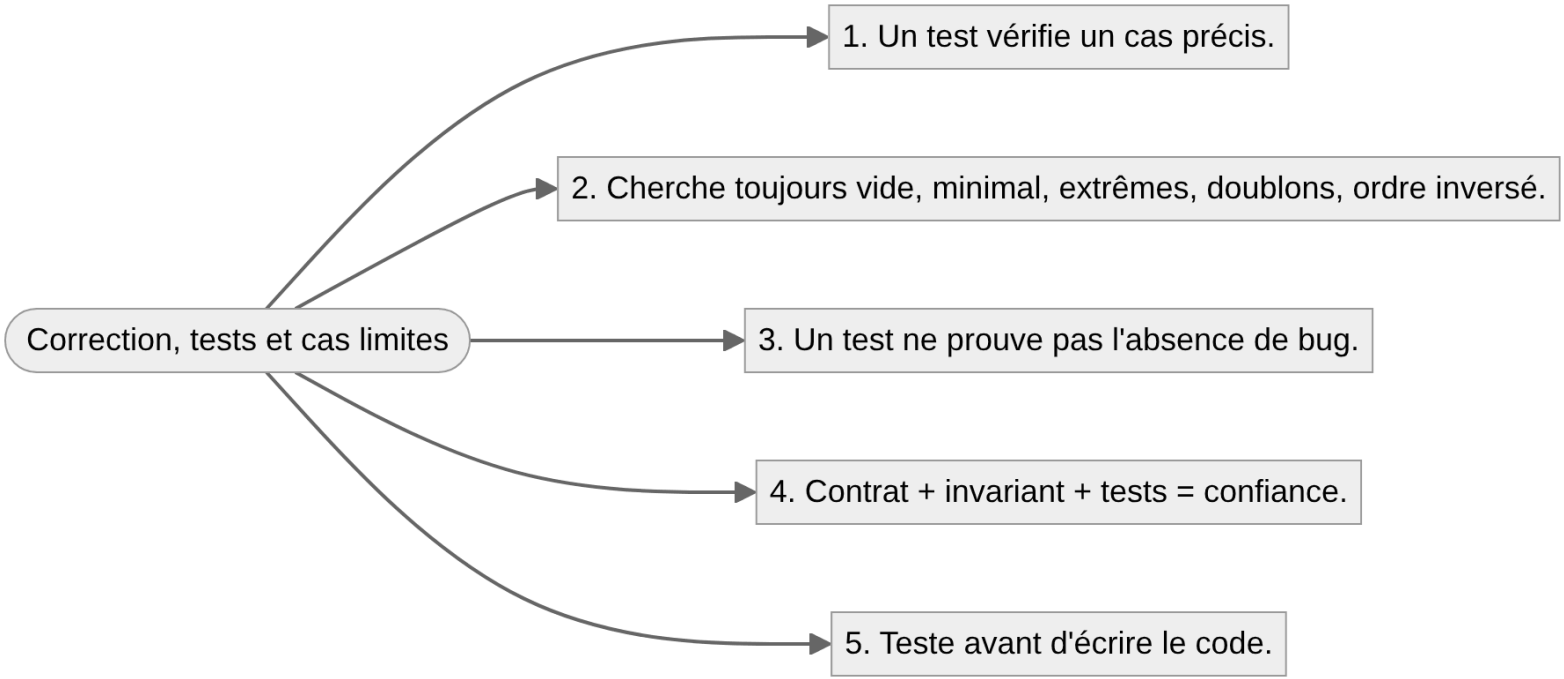
À retenir

À retenir

1. Un test vérifie un cas précis.
2. Cherche toujours vide, minimal, extrêmes, doublons, ordre inversé.
3. Un test ne prouve pas l'absence de bug.
4. Contrat + invariant + tests = confiance.
5. Teste avant d'écrire le code.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Gardien·ne de la qualité

Prochaine mission : Comparer des approches

