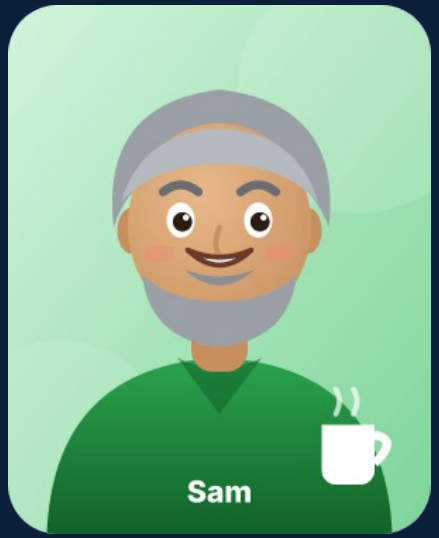
HEF-010 · MISSION 03 / 10

Fonctions, invariants et contrats

Une bonne fonction fait une chose, la fait bien, et dit clairement ce qu'elle promet.





Badge : Constructeur de briques

70 min · débutant

L'histoire

Le code de Kodi grandit : cent lignes qui calculent des moyennes, filtrent des membres, formatent des dates. Un bug apparaît. Impossible de savoir où.



> *« Découpe en fonctions, dit Sam. Chacune avec une promesse claire. Si une promesse est rompue, tu sais où chercher. »*

Dans cette mission, tu apprends à écrire des fonctions fiables.

Ta mission

À la fin, je serai capable de…

1. écrire une fonction avec des paramètres et une valeur de retour
2. décrire un contrat : ce qu'elle exige, ce qu'elle garantit
3. distinguer fonction pure et fonction avec effet de bord
4. identifier un invariant de boucle
5. documenter une fonction en une phrase claire

Défi d'échauffement

Défi d'échauffement

Tu veux donner une consigne à un ami : *« Fais-moi un thé. »* Écris ce que **tu lui fournis** (entrées), ce que **tu attends** (sortie) et **ce que tu supposes** (« il y a de l'eau », « il sait faire bouillir »). Qu'arrive-t-il si une supposition est fausse ?

Le concept

La fonction : une promesse

Une **fonction** prend des entrées (**paramètres**), fait un travail, et rend un résultat (**valeur de retour**).

```
def aire_rectangle(largeur, hauteur):
    return largeur * hauteur
```

Avantages : on réutilise, on teste, on nomme une idée.

Le contrat

Un **contrat** dit :

- **Préconditions** : ce que l'appelant doit fournir (« `liste` n'est pas vide »).
- **Postconditions** : ce que la fonction garantit (« renvoie un élément de la liste »).

Le concept

Le contrat (suite)

```
def plus_grand(liste):
    """Renvoie le plus grand élément. Exige: liste non vide."""
    assert len(liste) > 0, "précondition: liste non vide"
    meilleur = liste[0]
    for x in liste:
        if x > meilleur:
            meilleur = x
    return meilleur
```

`assert` arrête le programme si la précondition est fausse : mieux vaut échouer tôt et clairement.

Fonction pure ou effet de bord

- **Pure** : même entrée → même sortie, rien d'autre ne change. Facile à tester.
- **Effet de bord** : elle modifie autre chose (affiche, écrit un fichier, change une liste reçue).

Le concept

Fonction pure ou effet de bord (suite)

Préfère les fonctions pures quand c'est possible.

L'invariant de boucle

Un **invariant** est une propriété vraie **avant, pendant et après** chaque tour de boucle. Dans `plus_grand`, l'invariant est : *« `meilleur` est le plus grand des éléments déjà lus »*. À la fin, tous sont lus : donc `meilleur` est le plus grand.

🧠 ***Astuce** : quand une boucle est difficile à comprendre, cherche son invariant.*

Exemple pas à pas

Problème : une fonction qui renvoie la somme des carrés des nombres impairs.

1. Contrat : entrée = liste d'entiers (peut être vide) ; sortie = entier ; pour une liste vide, 0.
2. Invariant : après chaque tour, `total` vaut la somme des carrés des impairs déjà lus.

```
def somme_carres_impairs(nombres):
    """Somme des carrés des nombres impairs. Liste vide -> 0."""
    total = 0
    for n in nombres:
        if n % 2 == 1:
            total = total + n * n
    return total

print(somme_carres_impairs([1, 2, 3, 4]))   # 1 + 9 = 10
print(somme_carres_impairs([]))             # 0
```

Exemple pas à pas

Exemple pas à pas (suite)

3. Vérification de l'invariant :

Lu	`total`	Invariant vrai ?
(rien)	0	oui, aucun impair lu
1	1	oui
2	1	oui, 2 est pair
3	10	oui
4	10	oui

À toi, avec aide

Écris `est_trie(liste)` qui renvoie `True` si la liste est triée par ordre croissant, `False` sinon.

1. Écris le contrat (une phrase).
2. Écris la fonction.
3. Teste `[1, 2, 2, 5]` → True, `[3, 1]` → False, `[]` → True.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Compare chaque élément avec le suivant.

Indice 2

Dès qu'un élément est plus grand que le suivant, tu peux répondre `False`.

Indice 3

Si la boucle se termine sans problème, renvoie `True`. Parcours les positions de 0 à `len(liste) - 2`.

À toi, avec aide · réponse

Réponse

Contrat : *renvoie True si chaque élément est inférieur ou égal au suivant (liste vide ou à un élément : True).*

```
def est_trie(liste):
    for i in range(len(liste) - 1):
        if liste[i] > liste[i + 1]:
            return False
    return True

print(est_trie([1, 2, 2, 5]))  # True
print(est_trie([3, 1]))        # False
print(est_trie([]))            # True
```

Invariant : à chaque tour, la partie déjà vérifiée est triée.

À toi, en solo

Écris `moyenne_sans_extremes(notes)` : la moyenne des notes **sans** la plus petite ni la plus grande. Exige au moins 3 notes (utilise `assert`). Écris le contrat, puis la fonction, puis 3 tests.

À toi, en solo · réponse

Pistes de correction

```
def moyenne_sans_extremes(notes):
    """Moyenne sans min ni max. Exige au moins 3 notes."""
    assert len(notes) >= 3, "il faut au moins 3 notes"
    mini, maxi = notes[0], notes[0]
    total = 0
    for n in notes:
        mini = min(mini, n)
        maxi = max(maxi, n)
        total = total + n
    return (total - mini - maxi) / (len(notes) - 2)

print(moyenne_sans_extremes([10, 12, 20]))        # 12.0
print(moyenne_sans_extremes([5, 5, 5, 5]))        # 5.0
print(moyenne_sans_extremes([1, 2, 3, 4, 100]))   # 3.0
```

Points à vérifier : le contrat dit ce qui se passe avec moins de 3 notes ; les doublons du minimum ne sont retirés qu'une fois.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Fonction qui fait tout	Impossible à tester ou à nommer	Une fonction = une idée
Contrat non écrit	Chacun suppose autre chose	Écrire une phrase d'entrée/sortie
Modifier la liste reçue	Effet de bord surprenant	Travailler sur une copie si besoin
Pas de test des cas limites	Plantage sur liste vide	Tester vide, un élément, deux éléments

Mini-boss

Mini-boss

Écris un petit module de 4 fonctions pour analyser les missions terminées par un membre (liste d'entiers : minutes passées sur chaque mission) :

- `total(minutes)` ;
- `plus_longue(minutes)` (exige une liste non vide) ;
- `au_dessus_de(minutes, seuil)` qui renvoie la **nouvelle liste** des valeurs supérieures au seuil, sans modifier l'originale ;
- `resume(minutes)` qui utilise les trois autres et renvoie une phrase.

Pour chacune : contrat en une phrase et au moins 2 tests.

Mini-boss · réponse

Pistes de correction

Un bon travail : `resume` est composée des trois autres (pas de code dupliqué) ; `au_dessus_de` construit une nouvelle liste (fonction pure) ; les contrats mentionnent le cas de la liste vide ; les tests incluent une liste vide pour `total` (0) et une erreur attendue pour `plus_longue([])`.

Quiz éclair

Quiz éclair

1. Une précondition est :

A. ce que la fonction renvoie
B. ce que l'appelant doit fournir
C. un commentaire facultatif

2. Un invariant de boucle est :

A. faux au début
B. le nom de la variable
C. vrai avant, pendant et après chaque tour

3. Une fonction pure :

A. ne modifie rien d'autre que son résultat

Quiz éclair (suite)

B. affiche toujours quelque chose
C. change ses arguments

4. 🔁 *Révision mission 02* : `while` s'arrête quand :

A. le programme est long
B. la condition devient fausse
C. on appelle print

5. 🔁 *Révision mission 01* : Les cas limites font partie :

A. du bonus
B. des bugs
C. de l'énoncé du problème

Quiz éclair · réponse

Réponses

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

Je réfléchis

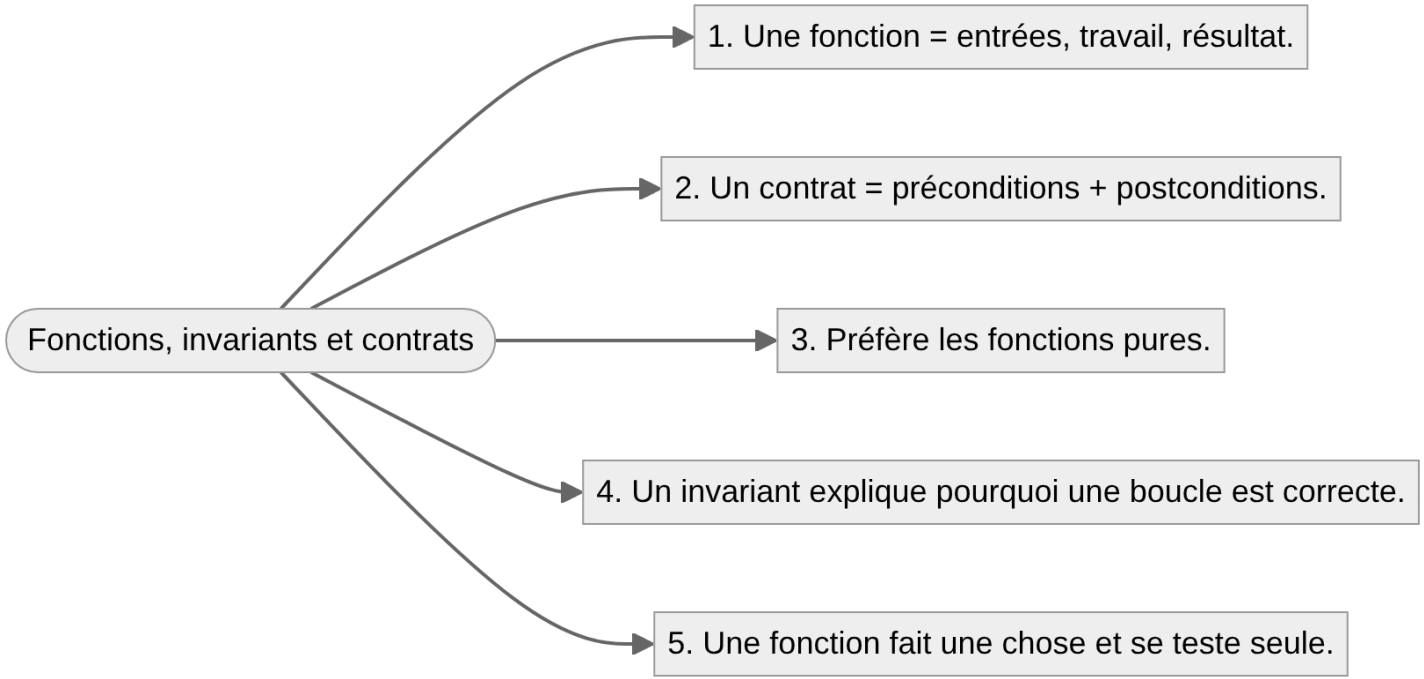
1. Dans ton thé du défi, quelle supposition était la plus risquée ? Comment l'écrire dans un contrat ?
2. Quelle fonction de ton quotidien (même hors code) a un contrat clair ?

À retenir

1. Une fonction = entrées, travail, résultat.
2. Un contrat = préconditions + postconditions.
3. Préfère les fonctions pures.
4. Un invariant explique pourquoi une boucle est correcte.
5. Une fonction fait une chose et se teste seule.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Constructeur de briques

Prochaine mission : Récursion : se résoudre soi-même

