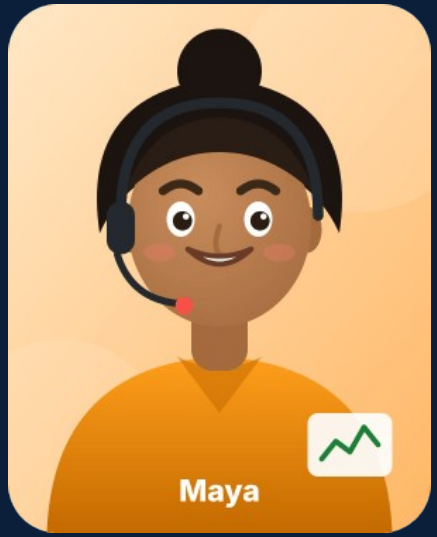
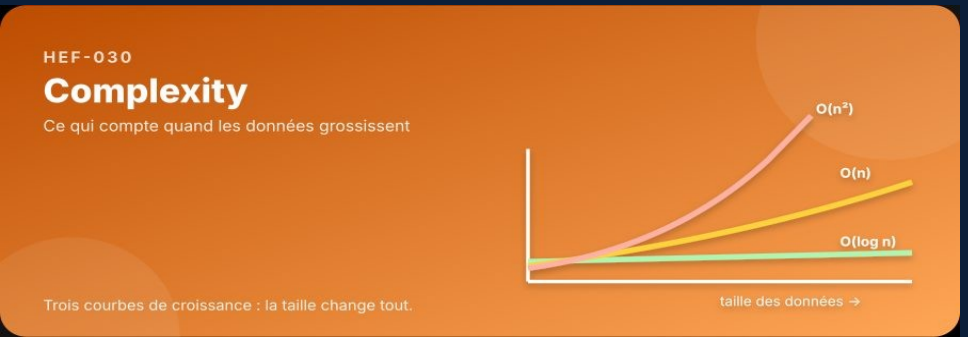
HEF-030 · MISSION 10 / 10

Optimiser avec preuves

Mesurer, comprendre, corriger, vérifier, expliquer : la démarche complète, de bout en bout.





Badge : Ingénieur·e de la performance

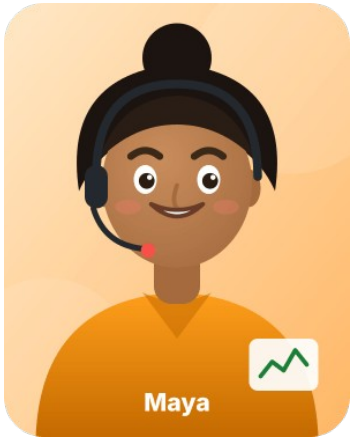
80 min · débutant

L'histoire

Le tableau de bord de HashBox est toujours lent pour les gros comptes. Cette fois, Maya demande un **rapport écrit** : pourquoi c'est lent, comment on le corrige, et comment on sait que c'est vraiment mieux.

> *« Pas de "je crois", dit Maya. Des chiffres, des preuves, et une conclusion qu'un autre ingénieur pourrait refaire. »*

Dans cette dernière mission, tu assembles tout ce que tu as appris sur la complexité.



Ta mission

À la fin, je serai capable de…

1. suivre une démarche d'optimisation complète : hypothèse, mesure, correction, preuve
2. justifier un gain par la complexité et par les mesures
3. vérifier qu'une optimisation ne change pas le résultat
4. décider quand arrêter d'optimiser
5. rédiger un rapport de performance

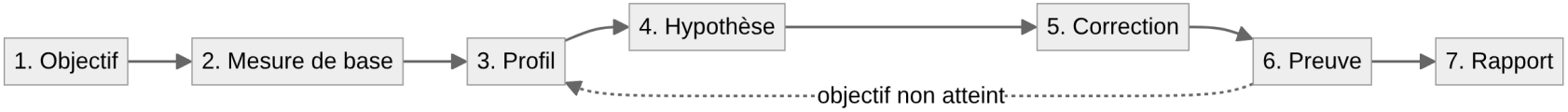
Défi d'échauffement

Défi d'échauffement

Un ami te dit : « mon application est lente, je vais tout réécrire en C ». En te souvenant de la mission 9, quelles **cinq questions** lui poses-tu avant qu'il ne se lance ?

Le concept

La démarche en 7 étapes



Description du schéma : sept étapes de gauche à droite, de l'objectif au rapport. Une flèche en pointillés revient de la preuve vers le profil si l'objectif n'est pas atteint.

1. **Objectif chiffré** : « la page doit s'afficher en moins de 1 s pour 50 000 éléments ».
2. **Mesure de base** : temps actuel, plusieurs tailles, sur des données réalistes.
3. **Profil** : où passe le temps ?
4. **Hypothèse** : « le goulot est un `in` sur une liste ; avec un ensemble il passera de O(n²) à O(n) ».

Le concept

La démarche en 7 étapes (suite)

5. **Correction** : une modification ciblée.
6. **Preuve** : (a) résultats identiques (tests), (b) nouvelle mesure, (c) la croissance observée correspond à la prévision.
7. **Rapport** : problème, méthode, résultats, risques, limites.

Vérifier l'équivalence

Une optimisation qui change le résultat est un bug. On compare les sorties de l'ancienne et de la nouvelle version sur beaucoup d'entrées aléatoires, **avant** de comparer les temps.

Quand s'arrêter ?

- Quand l'objectif est atteint.
- Quand le gain suivant ne vaut pas la complexité ajoutée (code illisible).

Le concept

Quand s'arrêter ? (suite)

- Quand le goulot est ailleurs (réseau, disque, base de données) : l'optimisation de calcul n'aidera plus.

Le rapport de performance

Section	Contenu
Contexte	Pourquoi c'est un problème, objectif chiffré
Méthode	Données, machine, outils, nombre de répétitions
Résultats avant	Tableau des temps, profil
Cause	Explication par la complexité
Correction	Ce qui a changé et pourquoi
Résultats après	Tableau des temps, courbe, preuve d'équivalence

Le concept

Le rapport de performance (suite)

Section	Contenu
Risques et limites	Mémoire, cas limites, conditions de validité
Décision	Recommandation

🧠 ***Astuce*** *: un rapport clair permet à quelqu'un de refaire ta mesure. C'est le test de qualité.*

Exemple pas à pas

Problème : « le calcul des recommandations est trop lent ».

Fonction lente :

```
import random, time

def recommandations(historiques, cible):
    # historiques : dict membre -> liste de cours suivis
    deja = historiques[cible]
    scores = {}
    for autre, cours in historiques.items():
        if autre == cible:
            continue
        communs = [c for c in cours if c in deja]       # `in` sur une liste
        if communs:
            for c in cours:
                if c not in deja:
                    scores[c] = scores.get(c, 0) + len(communs)
```

Exemple pas à pas (suite)

```
    return sorted(scores, key=lambda c: (-scores[c], c))[:5]

def recommandations_rapide(historiques, cible):
    deja = set(historiques[cible])
    scores = {}
    for autre, cours in historiques.items():
        if autre == cible:
            continue
        communs = len(deja.intersection(cours))
        if communs:
            for c in cours:
                if c not in deja:
                    scores[c] = scores.get(c, 0) + communs
    return sorted(scores, key=lambda c: (-scores[c], c))[:5]

random.seed(3)
catalogue = list(range(300))
historiques = {m: random.sample(catalogue, 60) for m in range(600)}
```

Exemple pas à pas (suite)

```
for m in range(0, 600, 60):
    assert recommandations(historiques, m) == recommandations_rapide(historiques,
   m)

for f in (recommandations, recommandations_rapide):
    t = time.perf_counter()
    for m in range(0, 600, 60):
        f(historiques, m)
    print(f.__name__, round(time.perf_counter() - t, 4))
```

Étapes visibles : équivalence vérifiée, puis mesure. L'hypothèse (le `in` sur la liste `deja`) est confirmée par la baisse du temps.

À toi, avec aide

Rédige le **plan de mesure** de l'exemple précédent :

1. objectif chiffré ;
2. tailles testées ;
3. nombre de répétitions ;
4. critère de réussite (gain minimal et équivalence) ;
5. risques (mémoire, données non représentatives).

À toi, avec aide · réponse

Indices progressifs

Indice 1

Fixe un objectif du type « moins de X ms par recommandation ».

Indice 2

Choisis au moins quatre tailles pour voir la tendance.

Indice 3

Pense à ce que tu feras si le gain est inférieur à 20 %.

Réponse

Exemple de plan :

1. Objectif : une recommandation en moins de 50 ms avec 5 000 membres de 60 cours chacun.
2. Tailles : 500, 1 000, 2 500, 5 000 membres.
3. Répétitions : 5 par taille, médiane.
4. Réussite : résultats identiques sur 100 membres aléatoires **et** gain d'au moins 30 % sur la plus grande taille.
5. Risques : l'ensemble de `deja` ajoute de la mémoire (négligeable ici) ; des historiques plus longs en production pourraient changer l'équilibre ; mesurer sur des données semblables à la production.

À toi, en solo

À toi de mener l'enquête. Choisis l'un des deux problèmes suivants et applique les 7 étapes, du code à un mini-rapport d'une page :

A. Trouver les paires de mots d'une liste de 5 000 mots qui sont des anagrammes.

B. Calculer pour chaque jour d'une période de 100 000 jours la somme des ventes des 7 derniers jours (fenêtre glissante).

Écris d'abord la version simple, mesure, corrige, prouve l'équivalence, rapporte.

Pistes de correction

Pistes. **A** : version simple = toutes les paires, comparaison des lettres triées ($O(n^2 \times k \log k)$) ; version rapide = dictionnaire signature → liste de mots ($O(n \times k \log k)$). **B** : version simple = `sum(ventes[i-6:i+1])` à chaque jour ($O(n \times 7)$, puis $O(n \times w)$ pour une fenêtre plus large) ; version rapide = fenêtre glissante qui ajoute le nouveau jour et retire le plus ancien ($O(n)$). Pour chaque cas : tests d'équivalence sur des entrées aléatoires, tableau de temps, estimation de l'exposant, section « limites » (par exemple fenêtre très petite : le gain disparaît).

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Optimiser sans objectif	On ne sait pas quand s'arrêter	Fixer un objectif chiffré
Oublier la preuve d'équivalence	Un résultat différent passe inaperçu	Tester avant de mesurer
Rapport sans méthode	Personne ne peut le refaire	Noter données, machine, répétitions
Optimiser le mauvais endroit	Gain négligeable	Profil d'abord

Mini-boss

Mini-boss

Dossier final du module HEF-030. Choisis un programme que tu as écrit dans les modules HEF-010 ou HEF-020 (une recherche, un tri, un graphe…) ou le problème B ci-dessus.
Produis un **rapport de performance** complet :

1. objectif chiffré ;
2. mesure de base pour 4 tailles, avec estimation de l'exposant ;
3. profil et goulot ;
4. hypothèse expliquée par la complexité ;
5. correction minimale ;
6. preuve : tests d'équivalence + nouvelle mesure + comparaison avec la prévision ;
7. limites et recommandation.

Livrables : `bench.py` (code et mesures) et `RAPPORT.md`.

Mini-boss · réponse

Pistes de correction

Critères d'un bon rapport : l'objectif est chiffré ; la méthode est reproductible (graine aléatoire fixée, nombre de répétitions, version de Python) ; l'hypothèse est confirmée **ou infirmée** honnêtement ; l'exposant mesuré est cohérent avec le Big O théorique ; la correction est minimale et lisible ; la section « limites » dit dans quel cas la solution n'apporte plus de gain.

Quiz éclair

Quiz éclair

1. La première étape d'une optimisation est :

A. fixer un objectif chiffré et mesurer
B. réécrire le code
C. changer de langage

2. Avant de comparer les temps, on vérifie :

A. la couleur du terminal
B. que les résultats sont identiques
C. la taille du fichier

3. On arrête d'optimiser quand :

A. le code est joli

Quiz éclair

Quiz éclair (suite)

B. la nuit tombe
C. l'objectif est atteint ou que le gain ne vaut plus la complexité

4. 🔁 *Révision mission 09* : `tottime` désigne :

A. le temps passé dans la fonction elle-même
B. le temps total avec sous-appels
C. le nombre d'appels

5. 🔁 *Révision mission 08* : Pour réduire le bruit d'une mesure, on :

A. la fait une seule fois
B. la répète et prend la médiane
C. la fait pendant un téléchargement

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quelle étape de la démarche te semble la plus difficile à respecter sous pression ?
2. Dans quel autre domaine de ta vie pourrais-tu appliquer « mesurer avant d'agir » ?

À retenir

1. Objectif chiffré → mesure → profil → hypothèse → correction → preuve → rapport.
2. Une optimisation correcte conserve le résultat.
3. La complexité explique, les mesures confirment.
4. On s'arrête quand l'objectif est atteint.
5. Un bon rapport est reproductible.

Félicitations

Félicitations

Tu as terminé **HEF-030 — Complexité** et le **bloc Fondations**. Tu sais compter le coût d'un algorithme, parler Big O, choisir la bonne structure, mesurer proprement, profiler, optimiser avec des preuves et rédiger un rapport.

→ Prochain bloc : **Systèmes** — commence par **HEF-040 — Architecture des ordinateurs**

MISSION ACCOMPLIE

+120 XP

Badge débloqué : Ingénieur·e de la performance

Fin du module HEF-030. Bravo !

