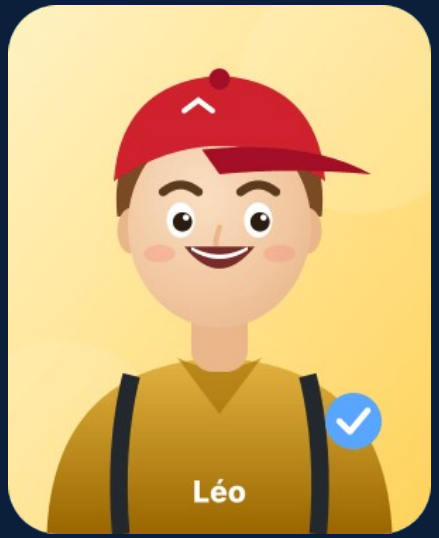
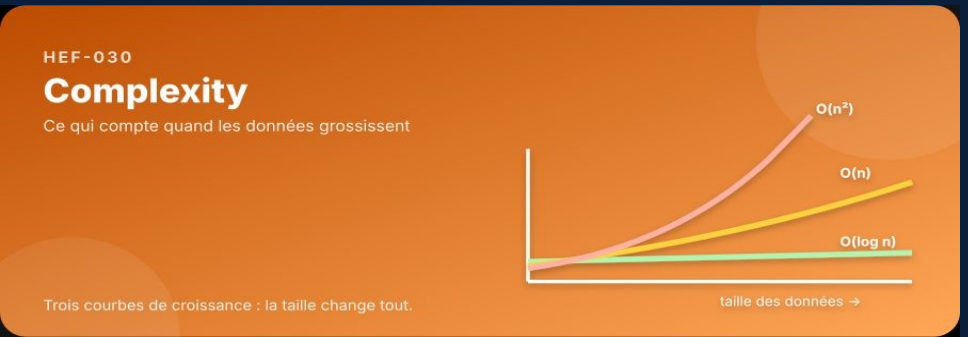
HEF-030 · MISSION 08 / 10

Mesurer pour de vrai

La théorie prédit, la mesure confirme. Savoir mesurer proprement est une compétence à part entière.





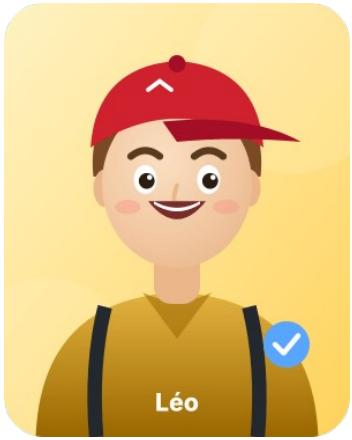
Badge : Scientifique du code

70 min · débutant

L'histoire

Kodi a optimisé une fonction et annonce fièrement : « C'est deux fois plus rapide ! » Sam demande : « Combien de fois as-tu mesuré ? Sur quelles données ? Et sur quelle machine ? » Kodi n'a mesuré qu'une fois, sur 10 éléments, pendant qu'il écoutait de la musique.

> *« Une mesure sans protocole n'est qu'une opinion », dit Sam.*



Dans cette mission, tu apprends à construire des benchmarks fiables et à lire leurs résultats.

Ta mission

À la fin, je serai capable de…

1. mesurer proprement avec `time.perf_counter` et `timeit`
2. répéter les mesures et utiliser la médiane
3. mesurer pour plusieurs tailles de données
4. estimer l'exposant de croissance à partir de mesures
5. identifier les causes de mesures trompeuses

Défi d'échauffement

Défi d'échauffement

Tu veux savoir si ton trajet à vélo est plus rapide que le bus. Tu le fais une fois : vélo 22 min, bus 25 min. Peux-tu conclure ? Liste **cinq raisons** pour lesquelles ta mesure pourrait être trompeuse. Que ferais-tu pour la rendre fiable ?

Le concept

Les bases d'une bonne mesure

1. **Répéter** : une mesure isolée est bruitée.
2. **Prendre la médiane** (ou le minimum) plutôt que la moyenne, moins sensible aux pics.
3. **Varier n** pour voir la tendance, pas un seul point.
4. **Isoler** ce qu'on mesure : ne pas inclure la génération des données.
5. **Contrôler** : mêmes données pour comparer deux versions.
6. **Noter l'environnement** : machine, version de Python, charge.

`time.perf_counter`

Le concept

`time.perf_counter` (suite)

```
import time

debut = time.perf_counter()
sorted(range(100000, 0, -1))
duree = time.perf_counter() - debut
print(round(duree, 5))
```

`timeit`

`timeit` répète le code et réduit le bruit.

```
import timeit

t = timeit.repeat("sorted(donnees)", setup="donnees = list(range(1000, 0, -1))", n
   umber=100, repeat=5)
print(min(t) / 100)      # meilleur temps moyen par exécution
```

Le concept

Un petit banc d'essai réutilisable

```
import time, statistics

def mesurer(f, donnees, repetitions=5):
    temps = []
    for _ in range(repetitions):
        t = time.perf_counter()
        f(donnees)
        temps.append(time.perf_counter() - t)
    return statistics.median(temps)
```

Estimer l'exposant

Si le temps vaut $c \times n^k$, alors en doublant n le temps est multiplié par 2^k. Le **rapport** entre deux mesures donne k :

Le concept

Estimer l'exposant (suite)

Rapport T(2n) / T(n)	Croissance probable
≈ 1	O(1)
≈ 2	O(n)
un peu plus de 2	O(n log n)
≈ 4	$O(n^2)$
≈ 8	$O(n^3)$

Pièges de mesure

Piège	Effet
Chauffer le cache / première exécution	première mesure plus lente
Trop petit n	bruit plus grand que le signal

Le concept

Pièges de mesure (suite)

Piège	Effet
Données favorables (déjà triées)	résultat non représentatif
Autre programme en cours	pics aléatoires
Mesurer la création des données	on mesure autre chose

🧠 ***Astuce*** *: une courbe vaut mieux qu'un chiffre. Trace le temps pour 5 valeurs de n.*

Exemple pas à pas

Exemple pas à pas

Problème : estimer le Big O de `a_doublon` à partir de mesures.

```
import time, statistics

def a_doublon(liste):
    for i in range(len(liste)):
        for j in range(i + 1, len(liste)):
            if liste[i] == liste[j]:
                return True
    return False

def mesurer(f, donnees, repetitions=5):
    temps = []
    for _ in range(repetitions):
        t = time.perf_counter()
        f(donnees)
        temps.append(time.perf_counter() - t)
    return statistics.median(temps)
```

Exemple pas à pas (suite)

```
precedent = None
for n in (250, 500, 1000, 2000):
    t = mesurer(a_doublon, list(range(n)))
    rapport = "" if precedent is None else f"x{t / precedent:.1f}"
    print(n, round(t, 4), rapport)
    precedent = t
```

Lecture : à chaque doublement de n, le rapport est autour de 4 : la croissance est **quadratique**, ce qu'on attendait.

Remarque : on a utilisé une liste sans doublon (pire cas). Avec une liste qui commence par deux valeurs identiques, la fonction s'arrête tout de suite : mesurer le meilleur cas aurait donné une conclusion fausse.

À toi, avec aide

Mesure `sorted` sur des listes aléatoires de n = 20 000, 40 000, 80 000, 160 000 éléments.

1. Génère les données **avant** la mesure.
2. Répète 5 fois et prends la médiane.
3. Calcule les rapports T(2n)/T(n). Que concluds-tu ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

Utilise `random.random()` pour générer.

Indice 2

Appelle `sorted(donnees)` dans la mesure ; ne mesure pas la génération.

Indice 3

Pour du O(n log n), le rapport est un peu au-dessus de 2.

À toi, avec aide · réponse

Réponse

```
import random, time, statistics

def mesurer(f, donnees, repetitions=5):
    temps = []
    for _ in range(repetitions):
        t = time.perf_counter()
        f(donnees)
        temps.append(time.perf_counter() - t)
    return statistics.median(temps)

precedent = None
for n in (20000, 40000, 80000, 160000):
    donnees = [random.random() for _ in range(n)]
    t = mesurer(sorted, donnees)
    print(n, round(t, 4), "" if precedent is None else f"x{t / precedent:.2f}")
    precedent = t
```

Les rapports sont un peu supérieurs à 2 : cohérent avec O(n log n).

À toi, en solo

Écris un outil `courbe(f, generateur, tailles)` qui, pour une fonction `f`, un générateur de données `generateur(n)` et une liste de tailles, renvoie la liste `[(n, temps_median)]` et affiche une **estimation de l'exposant** `k` par régression simple : k ≈ log(T2/T1) / log(n2/n1), moyenné sur les paires consécutives.

Teste-le sur : parcours simple (k ≈ 1), double boucle (k ≈ 2), `sorted` (k un peu au-dessus de 1).

À toi, en solo · réponse

Pistes de correction

```
import math, random, statistics, time

def courbe(f, generateur, tailles, repetitions=5):
    points = []
    for n in tailles:
        donnees = generateur(n)
        temps = []
        for _ in range(repetitions):
            t = time.perf_counter()
            f(donnees)
            temps.append(time.perf_counter() - t)
        points.append((n, statistics.median(temps)))
    exposants = [math.log(t2 / t1) / math.log(n2 / n1)
                 for (n1, t1), (n2, t2) in zip(points, points[1:]) if t1 > 0 and t
   2 > 0]
    k = sum(exposants) / len(exposants)
    return points, k

def parcours(l):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    s = 0
    for x in l:
        s += x
    return s

def double(l):
    c = 0
    for a in l:
        for b in l:
            c += 1
    return c

gen = lambda n: list(range(n))
for nom, f, tailles in (("parcours", parcours, (200000, 400000, 800000)), ("double
   ", double, (200, 400, 800))):
    _, k = courbe(f, gen, tailles)
    print(nom, "k ≈", round(k, 2))
```

À toi, en solo · réponse

Pistes de correction (suite)

Les valeurs de k sont proches de 1 et 2. Un écart de ±0,3 est normal : le bruit de mesure existe.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Une seule mesure	Bruit pris pour un résultat	Répéter et prendre la médiane
Mesurer la préparation des données	On mesure autre chose	Séparer génération et mesure
Un seul n	Impossible de voir la tendance	Au moins 4 ou 5 tailles
Comparer sur des données différentes	Comparaison injuste	Mêmes données pour les deux versions

Mini-boss

Mini-boss

Banc d'essai complet. Compare trois façons de détecter un doublon : double boucle, `len(set(l)) != len(l)`, et tri puis comparaison des voisins.

1. Génère trois jeux : sans doublon (pire cas), avec doublon au début, avec doublon à la fin.
2. Mesure pour n = 1 000, 2 000, 4 000, 8 000 (réduis n pour la double boucle si besoin).
3. Remplis un tableau et estime l'exposant de chaque méthode sur chaque jeu.
4. Écris un rapport de 10 lignes : conclusions, limites de ta mesure, recommandation.

Mini-boss · réponse

Pistes de correction

```
import statistics, time

def v1(l):
    for i in range(len(l)):
        for j in range(i + 1, len(l)):
            if l[i] == l[j]:
                return True
    return False

def v2(l):
    return len(set(l)) != len(l)

def v3(l):
    s = sorted(l)
    return any(s[i] == s[i + 1] for i in range(len(s) - 1))

def mes(f, d, r=3):
    t = []
    for _ in range(r):
```

Mini-boss · réponse

Pistes de correction (suite)

```
        a = time.perf_counter(); f(d); t.append(time.perf_counter() - a)
    return statistics.median(t)

for n in (500, 1000, 2000):
    jeux = {"sans": list(range(n)), "debut": [0, 0] + list(range(2, n)), "fin": li
   st(range(n - 1)) + [0]}
    for nom, d in jeux.items():
        print(n, nom, *(round(mes(f, d), 5) for f in (v1, v2, v3)))
```

Rapport attendu : v1 est O(n²) sauf si le doublon est au début ; v2 est O(n) ; v3 est O(n log n). La limite : les mesures dépendent de la machine et de Python.

Quiz éclair

Quiz éclair

1. Pour réduire le bruit d'une mesure, on :

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

2. Si T(2n)/T(n) ≈ 4, la croissance est :

A. quadratique
B. linéaire
C. constante

3. On ne doit pas inclure dans la mesure :

A. l'appel de la fonction

Quiz éclair

Quiz éclair (suite)

B. la génération des données
C. le calcul du résultat

4. 🔁 *Révision mission 06* : Trier avant de chercher est rentable quand :

A. on fait une seule recherche
B. les données changent sans cesse
C. on fait beaucoup de recherches

5. 🔁 *Révision mission 07* : Un cache sans limite risque de :

A. consommer toute la mémoire
B. ralentir le processeur
C. effacer les données

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quelle « mesure rapide » as-tu déjà prise pour une vérité ?
2. Comment expliquerais-tu à un collègue pourquoi on prend la médiane ?

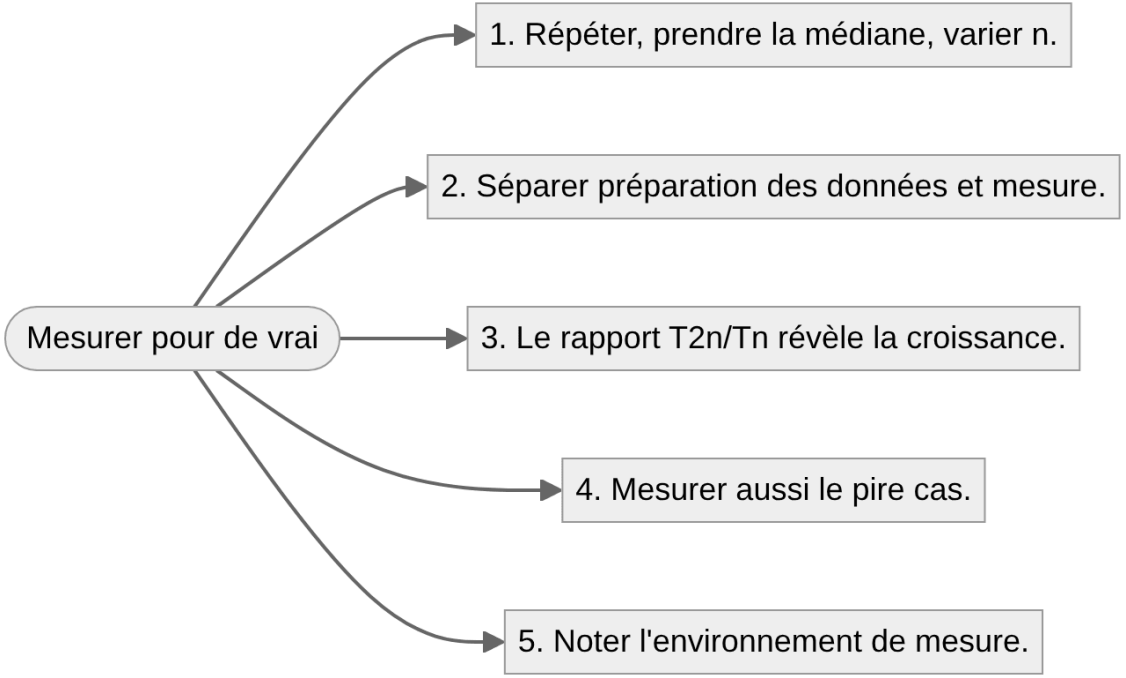
À retenir

À retenir

1. Répéter, prendre la médiane, varier n.
2. Séparer préparation des données et mesure.
3. Le rapport T(2n)/T(n) révèle la croissance.
4. Mesurer aussi le pire cas.
5. Noter l'environnement de mesure.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Scientifique du code

Prochaine mission : Profilage : trouver où le temps passe

