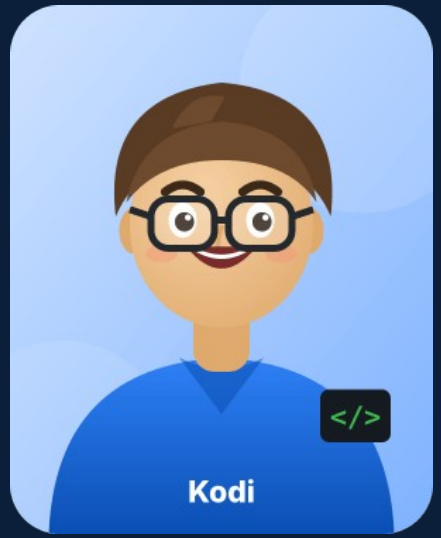
HEF-010 · MISSION 09 / 10

Comparer des approches

Deux solutions correctes ne se valent pas forcément. Savoir choisir, c'est savoir comparer.



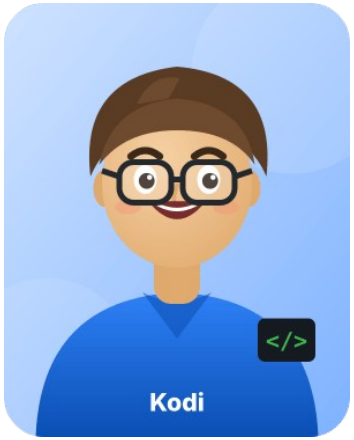


Badge : Arbitre

70 min · débutant

L'histoire

Maya veut **dédupliquer** la liste d'inscrits : retirer les doublons. Kodi propose deux solutions. Sam lui demande laquelle est la meilleure.



> *« Ça dépend. De quoi ? » répond Kodi, qui commence à comprendre.*

Dans cette mission, tu apprends à comparer des solutions sur des critères précis.

Ta mission

À la fin, je serai capable de…

1. comparer deux algorithmes selon temps, mémoire et clarté
2. mesurer le temps d'exécution avec `time`
3. compter les opérations d'un algorithme
4. utiliser un ensemble pour éviter une double boucle
5. justifier un choix par des données

Défi d'échauffement

Défi d'échauffement

Pour savoir si une liste de 1 000 noms contient des doublons, tu as deux idées : (A) comparer chaque nom avec tous les autres ; (B) ranger les noms dans une boîte et voir si l'un y est déjà. **Laquelle te semble plus rapide et pourquoi ?** Estime le nombre de comparaisons pour (A).

Le concept

Critères de comparaison

Critère	Question
Correction	Donne-t-elle toujours le bon résultat ? (en premier !)
Temps	Combien d'opérations quand les données grandissent ?
Mémoire	Combien d'espace en plus ?
Clarté	Quelqu'un d'autre la comprend-il ?
Hypothèses	A-t-elle besoin de données triées, de nombres… ?

🧠 ***Une solution rapide mais fausse ne vaut rien.*** *La correction passe avant.*

Le concept

Deux manières de compter

1. **Compter les opérations** : on incrémente un compteur dans le code.
2. **Mesurer le temps** :

```
import time
debut = time.perf_counter()
# ... code à mesurer ...
duree = time.perf_counter() - debut
```

La mesure dépend de la machine ; le comptage d'opérations, non. Les deux se complètent.

L'ensemble (`set`) pour tester l'appartenance

Le concept

L'ensemble (`set`) pour tester l'appartenance (suite)

Un ensemble Python permet de savoir si une valeur est présente en un temps quasi constant, quelle que soit sa taille. Il évite la double boucle.

```
def a_doublons_lent(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 a_doublons_rapide(liste):
    vus = set()
    for x in liste:
        if x in vus:
            return True
        vus.add(x)
    return False
```

Le concept

L'ensemble (`set`) pour tester l'appartenance (suite)

La première fait de l'ordre de n × n comparaisons, la seconde de l'ordre de n. Elle utilise **plus de mémoire** : on échange de la mémoire contre du temps. C'est un **compromis**.

Exemple pas à pas

Mesure sur 2 000 éléments sans doublon (pire cas) :

```
import time

def a_doublons_lent(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 a_doublons_rapide(liste):
    vus = set()
    for x in liste:
        if x in vus:
            return True
        vus.add(x)
    return False
```

Exemple pas à pas (suite)

```
donnees = list(range(2000))
for f in (a_doublons_lent, a_doublons_rapide):
    t = time.perf_counter()
    resultat = f(donnees)
    print(f.__name__, resultat, round(time.perf_counter() - t, 5), "s")
```

Comptage théorique : la version lente fait 2000 × 1999 / 2 ≈ 2 millions de comparaisons ; la rapide en fait 2 000.

Verdict : si les listes sont petites (10 éléments), la différence est invisible et la version lente est plus simple à lire. Au-delà, la rapide gagne.

À toi, avec aide

Compare deux façons de savoir si un mot est dans une liste de 5 000 mots, appelée 1 000 fois :

- A : `mot in liste` (liste Python) ;
- B : `mot in ensemble` (set construit une fois).

Écris le code de mesure puis note les temps. Conclus en une phrase.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Construis la liste avec `[str(i) for i in range(5000)]`.

Indice 2

Construis `ensemble = set(liste)` une seule fois, avant la mesure.

Indice 3

Mesure avec `time.perf_counter()` autour d'une boucle de 1 000 recherches.

À toi, avec aide · réponse

Réponse

```
import time

liste = [str(i) for i in range(5000)]
ensemble = set(liste)
cibles = [str(4999 - i) for i in range(1000)]

t = time.perf_counter()
for c in cibles:
    c in liste
tl = time.perf_counter() - t

t = time.perf_counter()
for c in cibles:
    c in ensemble
te = time.perf_counter() - t

print("liste :", round(tl, 5), "s  ensemble :", round(te, 5), "s")
```

Réponse (suite)

L'ensemble est nettement plus rapide (de l'ordre de dizaines à centaines de fois), au prix de la mémoire et de la construction initiale.

À toi, en solo

Écris deux versions de `intersection(a, b)` (éléments présents dans les deux listes, sans doublons dans le résultat) :

1. une version avec deux boucles ;
2. une version avec un ensemble.

Vérifie qu'elles donnent le même résultat (comme ensembles) sur des listes aléatoires, puis compare leurs temps sur 3 000 éléments. Écris un petit verdict (3 lignes) avec le critère de choix.

À toi, en solo · réponse

Pistes de correction

```
import random, time

def inter_lente(a, b):
    res = []
    for x in a:
        for y in b:
            if x == y and x not in res:
                res.append(x)
    return res

def inter_rapide(a, b):
    return list(set(a) & set(b))

for _ in range(200):
    a = [random.randint(0, 20) for _ in range(random.randint(0, 10))]
    b = [random.randint(0, 20) for _ in range(random.randint(0, 10))]
    assert set(inter_lente(a, b)) == set(inter_rapide(a, b))

a = [random.randint(0, 5000) for _ in range(3000)]
```

À toi, en solo · réponse

Pistes de correction (suite)

```
b = [random.randint(0, 5000) for _ in range(3000)]
for f in (inter_lente, inter_rapide):
    t = time.perf_counter()
    f(a, b)
    print(f.__name__, round(time.perf_counter() - t, 4), "s")
```

Verdict : correctes toutes les deux ; la version ensemble gagne largement quand les listes grandissent ; on garde la version à deux boucles seulement pour de très petites listes ou pour préserver l'ordre.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Comparer sur un seul exemple	Résultat non représentatif	Plusieurs tailles de données
Mesurer une seule fois	Bruit de la machine	Répéter et prendre la médiane
Optimiser avant de comparer	Complexifier sans gain	Mesurer d'abord
Ignorer la mémoire	Solution rapide mais gourmande	Toujours noter le coût en mémoire

Mini-boss

Mini-boss

Tu dois proposer, pour une liste de 100 000 inscrits, la meilleure façon de **retirer les doublons en gardant l'ordre d'apparition**.

1. Écris deux solutions différentes.
2. Prouve qu'elles sont correctes sur 5 cas limites.
3. Mesure sur 1 000, 10 000 et 100 000 éléments.
4. Rédige une recommandation d'une demi-page : critères, résultats, compromis.

Mini-boss · réponse

Pistes de correction

Solution lente : pour chaque élément, vérifier `if x not in resultat` (liste). Solution rapide : un ensemble `vus` plus une liste `resultat`. Les mesures montrent que la lente devient inutilisable à 100 000 éléments, la rapide reste instantanée. Recommandation : la version à ensemble, en notant le surcoût mémoire et l'hypothèse que les éléments soient « hachables ».

Quiz éclair

1. Avant de comparer la vitesse de deux algorithmes, il faut vérifier :

A. qu'ils ont le même nom
B. qu'ils sont corrects
C. qu'ils sont courts

2. Un ensemble permet de tester la présence :

A. lentement
B. uniquement sur des nombres triés
C. très rapidement, au prix de la mémoire

3. Un compromis, c'est :

A. gagner en temps en payant en mémoire, par exemple

Quiz éclair (suite)

B. un bug
C. un test

4. 🔁 *Révision mission 08* : Un test montre :

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

5. 🔁 *Révision mission 07* : Un glouton :

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

Quiz éclair · réponse

Réponses

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

Je réfléchis

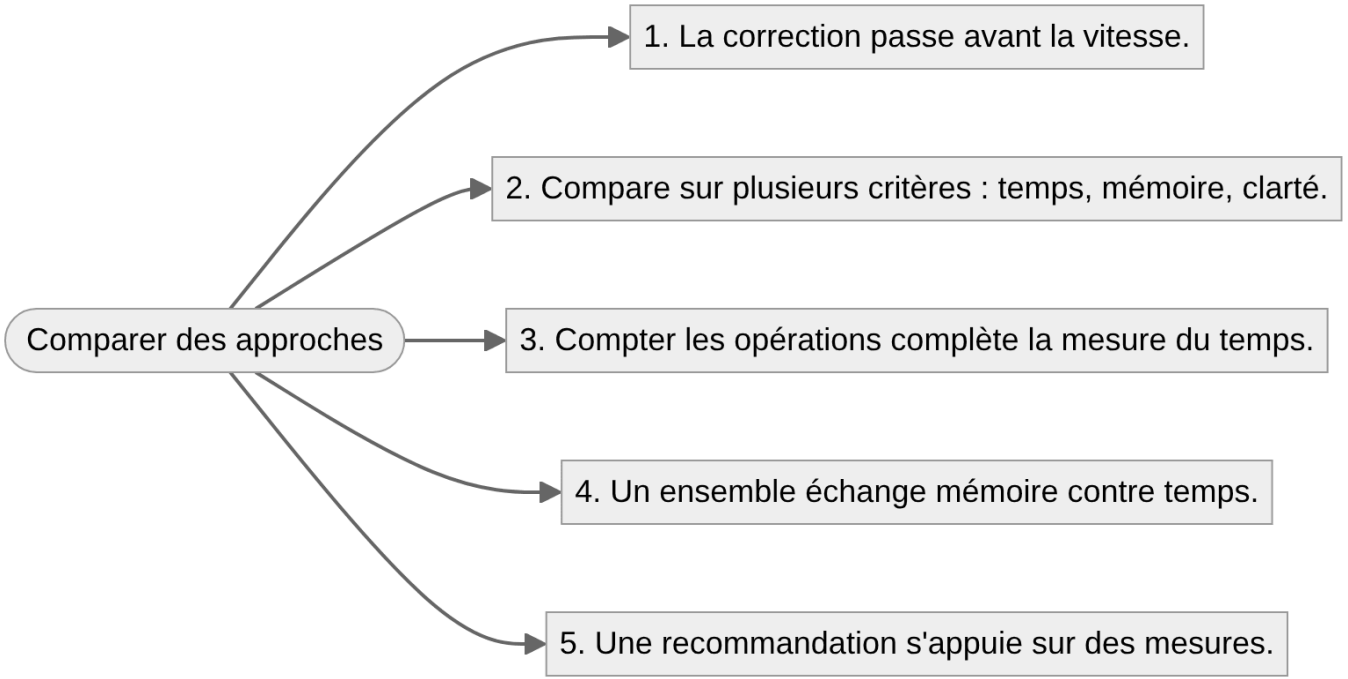
1. Quel critère de comparaison te paraît le plus souvent oublié ?
2. Dans un choix de ta vie, as-tu comparé plusieurs options sur des critères écrits ?

À retenir

1. La correction passe avant la vitesse.
2. Compare sur plusieurs critères : temps, mémoire, clarté.
3. Compter les opérations complète la mesure du temps.
4. Un ensemble échange mémoire contre temps.
5. Une recommandation s'appuie sur des mesures.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Arbitre

Prochaine mission : Face à un problème inédit

