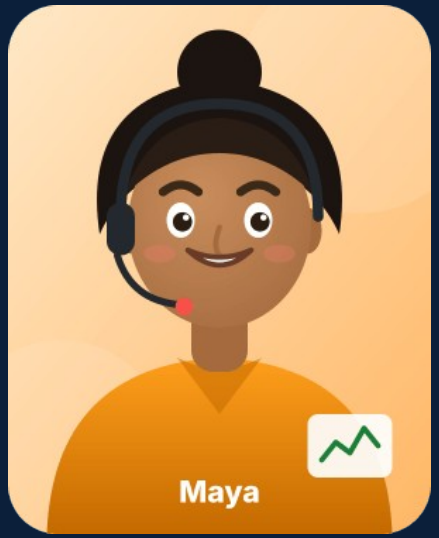
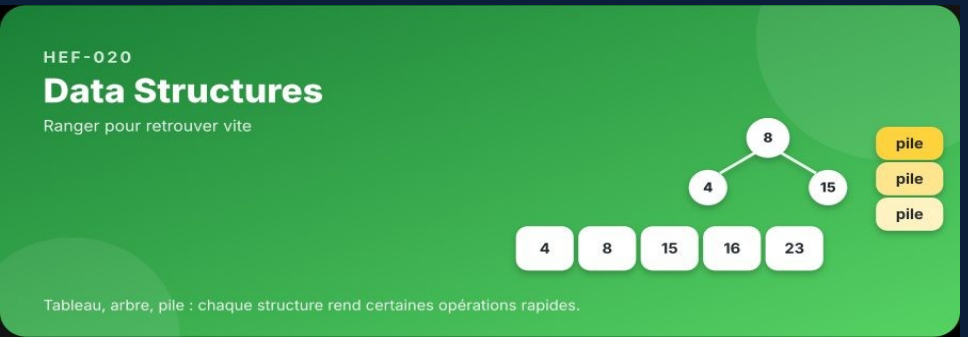
HEF-020 · MISSION 10 / 10

Choisir, prouver, comparer

La vraie compétence : choisir la bonne structure, expliquer pourquoi, et prouver qu'elle tient ses promesses.





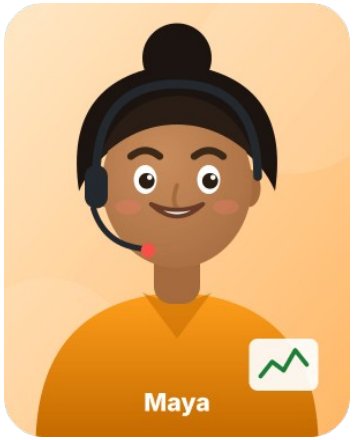
Badge : Ingénieur·e des données

80 min · débutant

L'histoire

Dernière mission du module. Maya donne à Kodi trois besoins, et lui demande de choisir **la** bonne structure pour chacun, avec des preuves :

1. un historique des 1 000 dernières actions ;
2. une recherche de membres par préfixe de nom (« Mar… ») ;
3. un classement en temps réel des 10 meilleurs scores.



« Pas d'intuition seule, dit Sam. Les opérations, les coûts, les invariants. »

Tu rassembles tout le module dans une démarche de choix.

Ta mission

À la fin, je serai capable de…

1. lister les opérations et leur fréquence pour un besoin
2. comparer plusieurs structures sur des critères précis
3. énoncer l'invariant d'une structure
4. écrire des tests de propriétés pour une structure
5. justifier un choix par un compromis

Défi d'échauffement

Défi d'échauffement

Tu dois stocker **les 100 000 produits d'une boutique**. Écris trois opérations que tu feras (rechercher par nom, lister par prix, ajouter…). Pour chacune, quelle structure du module te vient en tête ? Une seule structure suffit-elle ?

Le concept

La démarche en 5 étapes

1. **Opérations** : quelles actions, avec quelle fréquence ?
2. **Contraintes** : taille des données, ordre, doublons, mémoire.
3. **Candidates** : 2 ou 3 structures plausibles.
4. **Comparaison** : coût des opérations clés, simplicité, mémoire.
5. **Preuve** : invariant et tests.

Tableau de synthèse

Structure	Lire par position	Chercher une valeur	Ajouter	Retirer	Ordre ?
Tableau (`list`)	rapide	lent	fin : rapide	milieu : lent	oui

Tableau de synthèse (suite)

Structure	Lire par position	Chercher une valeur	Ajouter	Retirer	Ordre ?
Liste chaînée	lent	lent	début : rapide	rapide si position connue	oui
Pile / file	:	:	rapide	rapide (une extrémité)	LIFO / FIFO
Dictionnaire / ensemble	:	rapide	rapide	rapide	non garanti
BST équilibré	:	rapide (log n)	rapide (log n)	rapide (log n)	trié
Tas	:	min : immédiat	rapide (log n)	min : rapide	partiel
Graphe	:	selon parcours	:	:	:

Le concept

Les invariants

Un **invariant de structure** est une propriété qui doit toujours être vraie :

Structure	Invariant
Pile	on retire toujours le dernier ajouté
BST	gauche < nœud < droite
Tas-min	parent ≤ enfants
Ensemble	pas de doublons

Pour le vérifier, on écrit une fonction `verifier_invariant()` et on l'appelle dans les tests, après chaque opération.

Le concept

Les tests de propriétés

Au lieu d'exemples fixes, on teste des **propriétés** sur beaucoup d'entrées aléatoires :

- insérer puis retirer tous les éléments d'un tas donne une liste triée ;
- le parcours en ordre d'un BST est trié ;
- une pile rend les éléments dans l'ordre inverse de leur entrée.

```
import heapq, random

for _ in range(300):
    donnees = [random.randint(-100, 100) for _ in range(random.randint(0, 20))]
    tas = []
    for x in donnees:
        heapq.heappush(tas, x)
    sortie = [heapq.heappop(tas) for _ in range(len(tas))]
```

Le concept

Les tests de propriétés (suite)

```
    assert sortie == sorted(donnees)
print("propriété du tas vérifiée")
```

🧠 ***Astuce*** *: combiner deux structures est courant. Un dictionnaire + une liste ou un tas résout souvent des besoins qu'une seule ne couvre pas.*

Exemple pas à pas

Besoin 1 : historique des 1 000 dernières actions.

Opérations : ajouter (très fréquent), lire toutes les actions (rare). Taille bornée. Candidat : `deque(maxlen=1000)` : l'ajout supprime automatiquement la plus ancienne.

Besoin 2 : recherche par préfixe.

Opérations : trouver tous les noms commençant par « Mar ». Candidats : liste triée + recherche binaire sur le préfixe, ou BST. Le choix : liste triée et `bisect`.

```
import bisect

noms = sorted(["Marie", "Marc", "Maya", "Omar", "Sara", "Marion"])

def par_prefixe(noms_tries, prefixe):
    debut = bisect.bisect_left(noms_tries, prefixe)
```

Exemple pas à pas (suite)

```
    fin = bisect.bisect_left(noms_tries, prefixe + "\uffff")
    return noms_tries[debut:fin]

print(par_prefixe(noms, "Mar"))   # ['Marc', 'Marie', 'Marion']
```

Besoin 3 : classement des 10 meilleurs.

Opération : un nouveau score arrive ; on veut toujours les 10 meilleurs. Candidat : tas-min de taille 10 (mission 07).

Tableau de décision :

Besoin	Structure	Raison
Historique	`deque(maxlen=1000)`	ajout rapide, taille bornée automatiquement
Préfixe	liste triée + `bisect`	recherche binaire, résultat contigu

Exemple pas à pas (suite)

Besoin	Structure	Raison
Top 10	tas-min de taille 10	seul le minimum du tas compte

À toi, avec aide

Pour chaque besoin, remplis : opérations, structure, raison.

1. Vérifier si un e-mail est déjà inscrit (100 000 e-mails, ajouts fréquents).
2. Annuler les dernières actions d'un éditeur.
3. Servir les demandes dans l'ordre d'arrivée.
4. Trouver le plus court trajet entre deux stations de métro.

Indices progressifs

Indice 1

Pour 1, pense à « est-il présent ? » rapide.

Indice 2

Pour 2 et 3, pense à LIFO et FIFO.

Indice 3

Pour 4, c'est un problème de relations avec distance en nombre d'arrêts.

À toi, avec aide · réponse

Réponse

1. Présence rapide, ajouts → **ensemble** (ou dictionnaire).
2. Dernier ajouté, premier annulé → **pile**.
3. Premier arrivé, premier servi → **file** (deque).
4. Stations reliées → **graphe** (liste d'adjacence) + **BFS**.

À toi, en solo

Écris une classe `Classement` qui garde les **N meilleurs scores** avec le nom du joueur.

- `ajouter(nom, score)` ;
- `top()` renvoie la liste `[(nom, score), …]` du meilleur au moins bon ;
- un score d'un joueur existant n'est retenu que s'il améliore le sien.

Ajoute `verifier_invariant()` (au plus N entrées, pas de doublon de joueur) et un test de propriétés sur 300 séquences aléatoires comparées à une version naïve.

À toi, en solo · réponse

Pistes de correction

```
import random

class Classement:
    def __init__(self, n):
        self.n = n
        self._scores = {}

    def ajouter(self, nom, score):
        if nom not in self._scores or score > self._scores[nom]:
            self._scores[nom] = score
        if len(self._scores) > self.n:
            pire = min(self._scores, key=lambda k: (self._scores[k], k))
            del self._scores[pire]

    def top(self):
        return sorted(self._scores.items(), key=lambda x: (-x[1], x[0]))

    def verifier_invariant(self):
        assert len(self._scores) <= self.n
```

À toi, en solo · réponse

Pistes de correction (suite)

```
def naif(n, evenements):
    meilleur = {}
    for nom, s in evenements:
        meilleur[nom] = max(meilleur.get(nom, s), s)
    return sorted(meilleur.items(), key=lambda x: (-x[1], x[0]))[:n]

for _ in range(300):
    evs = [(random.choice("abcdef"), random.randint(0, 20)) for _ in range(random.
   randint(0, 15))]
    c = Classement(3)
    for e in evs:
        c.ajouter(*e)
        c.verifier_invariant()
    attendu = naif(3, evs)
    assert [s for _, s in c.top()] == [s for _, s in attendu]
print("ok")
```

Pistes de correction (suite)

Remarque : on compare les scores (pas les noms) car en cas d'égalité entre joueurs hors du top, le départage peut varier.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Choisir par habitude	Toujours une liste, même quand c'est lent	Lister d'abord les opérations
Optimiser trop tôt	Complexité inutile pour 50 éléments	Mesurer avant de complexifier
Oublier l'invariant	Bug silencieux	Écrire `verifier_invariant()` et l'appeler dans les tests
Une seule structure à tout faire	Compromis forcé	Combiner (dictionnaire + tas, etc.)

Mini-boss

Projet final du module — dossier de choix. Pour **chacun des cinq besoins** ci-dessous, produis une fiche d'une demi-page :

1. file d'attente d'un support client (priorité aux clients premium) ;
2. correcteur orthographique (« ce mot existe-t-il ? ») ;
3. suggestion « membres que tu pourrais connaître » ;
4. historique de navigation avec retour arrière ;
5. agenda où l'on cherche les rendez-vous entre deux dates.

Chaque fiche : opérations et fréquence, structure choisie et alternative écartée, invariant, coût attendu, un petit code de démonstration avec test.

Mini-boss · réponse

Pistes de correction

Pistes : (1) tas avec compteur d'arrivée ; (2) ensemble de mots ; (3) graphe + amis d'amis (BFS à distance 2) ; (4) deux piles (arrière/avant) ; (5) liste triée par date + `bisect`, ou BST. Une bonne fiche justifie le choix par les opérations les plus fréquentes et nomme honnêtement ce qu'on perd.

Quiz éclair

Quiz éclair

1. Pour choisir une structure, on commence par :

A. lister les opérations et leur fréquence
B. choisir la plus récente
C. utiliser une liste

2. Un invariant de structure est :

A. un commentaire
B. une propriété toujours vraie
C. un test unique

3. Un test de propriété :

A. vérifie un seul exemple

Quiz éclair

Quiz éclair (suite)

B. remplace le code
C. vérifie une règle sur beaucoup d'entrées aléatoires

4. 🔁 *Révision mission 09* : BFS trouve dans un graphe non pondéré :

A. le plus court chemin en nombre d'arêtes
B. le chemin le plus long
C. un chemin au hasard

5. 🔁 *Révision mission 04* : Une table de hachage associe :

A. des nœuds à des feuilles
B. des clés à des valeurs
C. des listes à des tris

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quel besoin réel de ta vie professionnelle ou scolaire pourrait être traité avec les structures du module ?
2. Quelle structure te semblait étrangère au début et te paraît maintenant évidente ?

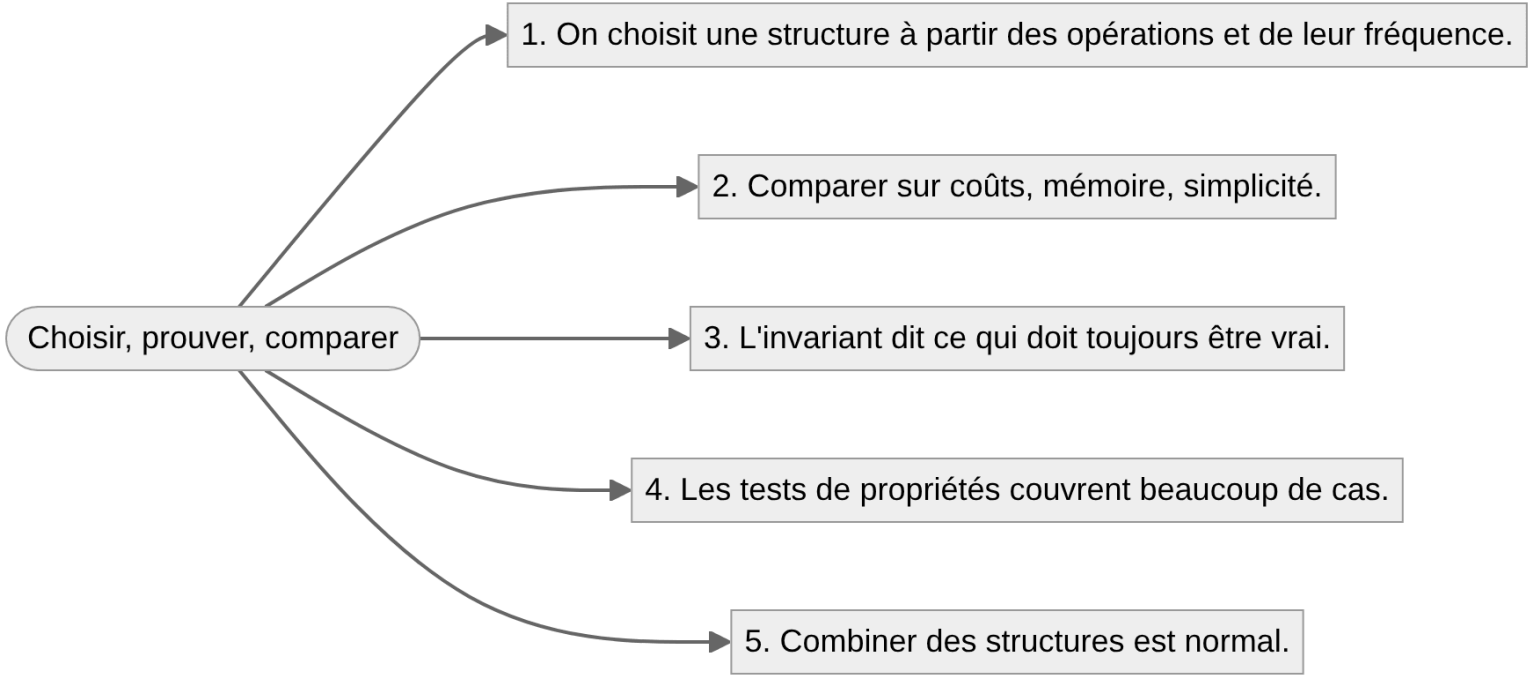
À retenir

À retenir

1. On choisit une structure à partir des opérations et de leur fréquence.
2. Comparer sur coûts, mémoire, simplicité.
3. L'invariant dit ce qui doit toujours être vrai.
4. Les tests de propriétés couvrent beaucoup de cas.
5. Combiner des structures est normal.

À retenir

Le schéma de la mission



Félicitations

Félicitations

Tu as terminé **HEF-020 — Structures de données**. Tu sais décrire une structure par ses opérations, en implémenter les principales, choisir la bonne selon l'usage, et prouver qu'elle tient ses promesses.
→ Prochain module : **HEF-030 — Complexité**

MISSION ACCOMPLIE

+120 XP

Badge débloqué : Ingénieur·e des données

Fin du module HEF-020. Bravo !

