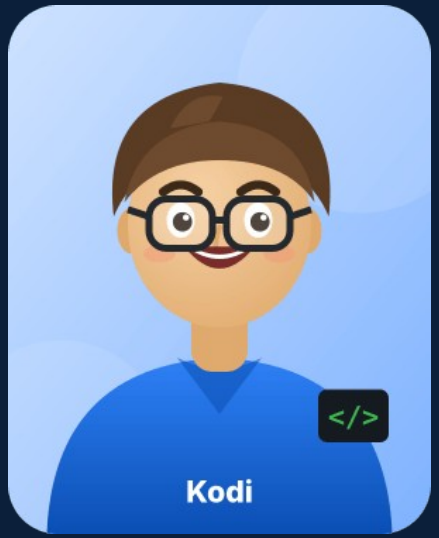
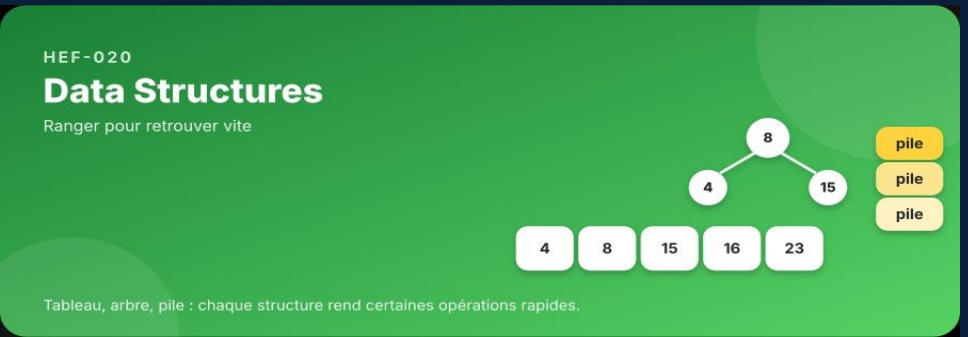
HEF-020 · MISSION 01 / 10

Ranger l'information : le modèle abstrait

Avant de choisir comment stocker, on décide ce qu'on veut pouvoir faire.





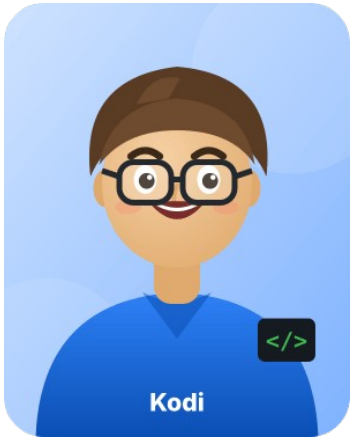
Badge : Architecte du rangement

65 min · débutant

L'histoire

HashBox grandit : 5 000 membres, 200 missions, des milliers de résultats de quiz. Kodi stocke tout dans une seule grande liste Python. Chaque recherche devient lente, chaque ajout provoque des erreurs.

> *« Tu mélanges deux questions, dit Sam. Qu'est-ce que je veux faire avec mes données ? et comment je les range en mémoire ? Commence par la première. »*



Cette mission introduit l'idée la plus importante du module : séparer ce qu'on veut faire de comment on le fait.

Ta mission

À la fin, je serai capable de…

1. définir une structure de données avec mes mots
2. distinguer un type abstrait (ADT) de son implémentation
3. décrire une structure par ses opérations
4. dire quelles opérations sont coûteuses ou rapides
5. choisir une structure à partir des opérations nécessaires

Défi d'échauffement

Défi d'échauffement

Pense à ta **bibliothèque** (ou à une liste de courses, ou à ton téléphone). Quelles actions fais-tu le plus souvent : ajouter, retrouver, supprimer, parcourir, trier ? Classe-les par fréquence. Selon ce classement, comment organiserais-tu les objets ?

Le concept

Structure de données

Une **structure de données** est une façon d'organiser l'information en mémoire pour que certaines opérations soient simples et rapides.

Analogie : une cuisine. Les couteaux sur un aimant (accès rapide), les conserves au fond du placard (peu utilisées). Le rangement dépend de l'usage.

Type abstrait de données (ADT)

Un **ADT** décrit une structure par son **comportement** (ses opérations), sans dire comment elle est faite à l'intérieur.

ADT	Opérations typiques
Liste	ajouter, lire à une position, supprimer, parcourir

Le concept

Type abstrait de données (ADT) (suite)

ADT	Opérations typiques
Pile	empiler, dépiler, regarder le sommet
File	enfiler, défiler
Dictionnaire	associer une clé à une valeur, retrouver par clé
Ensemble	ajouter, tester l'appartenance

Interface et implémentation

- L'**interface** : ce qu'on peut faire (les opérations et leurs règles).
- L'**implémentation** : comment c'est fait (tableau, liste chaînée, arbre…).

On peut changer l'implémentation sans changer le code qui utilise l'interface. C'est le même principe que les contrats de la mission « Fonctions » du module HEF-010.

Le concept

Interface et implémentation (suite)

```
class CompteurDeVisites:
    """Interface: enregistrer(page), total(page). Rien sur l'implémentation."""
    def __init__(self):
        self._visites = {}          # implémentation cachée

    def enregistrer(self, page):
        self._visites[page] = self._visites.get(page, 0) + 1

    def total(self, page):
        return self._visites.get(page, 0)

c = CompteurDeVisites()
c.enregistrer("accueil")
c.enregistrer("accueil")
print(c.total("accueil"))   # 2
```

On pourrait remplacer le dictionnaire par une liste de couples : le code qui utilise `CompteurDeVisites` ne changerait pas.

Le concept

Le vocabulaire des coûts

Pour chaque opération, on se demande : combien d'étapes ? Est-ce constant, ou proportionnel à la taille ? On formalisera cela dans HEF-030 (complexité).

🧠 ***Astuce*** *: choisis la structure en commençant par la liste des opérations les plus fréquentes.*

Exemple pas à pas

Besoin : garder les 3 dernières pages visitées par un membre.

1. Opérations : ajouter une page, oublier la plus ancienne si on dépasse 3, lire les 3 pages.
2. Choix d'ADT : une **file** de taille limitée.

3. Interface

```
class Historique:
    """Garde les n derniers éléments."""
    def __init__(self, taille):
        self._taille = taille
        self._elements = []

    def ajouter(self, page):
        self._elements.append(page)
        if len(self._elements) > self._taille:
```

Exemple pas à pas (suite)

```
            self._elements.pop(0)

    def contenu(self):
        return list(self._elements)

h = Historique(3)
for p in ["a", "b", "c", "d"]:
    h.ajouter(p)
print(h.contenu())   # ['b', 'c', 'd']
```

4. Ce qu'on remarque : l'utilisateur de `Historique` ignore que c'est une liste. Si demain on utilise une autre structure plus efficace, `ajouter` et `contenu` restent les mêmes.

À toi, avec aide

Décris l'ADT « **Panier** » d'une boutique de formations :

1. Liste ses opérations (au moins 4).
2. Pour chacune, écris une phrase de contrat.
3. Écris une classe `Panier` en Python avec ces opérations (implémentation libre).

À toi, avec aide · réponse

Indices progressifs

Indice 1

Opérations possibles : ajouter, retirer, vider, compter, total.

Indice 2

Contrat de `retirer` : que se passe-t-il si la formation n'est pas dans le panier ?

Indice 3

Une liste ou un dictionnaire convient comme implémentation.

À toi, avec aide · réponse

Réponse

```
class Panier:
    """Ajouter/retirer des formations (par nom) avec leur prix."""
    def __init__(self):
        self._articles = {}          # nom -> prix

    def ajouter(self, nom, prix):
        self._articles[nom] = prix   # remplace si déjà présent

    def retirer(self, nom):
        self._articles.pop(nom, None)  # sans erreur si absent

    def compter(self):
        return len(self._articles)

    def total(self):
        return sum(self._articles.values())

p = Panier()
p.ajouter("HEF-000", 20)
```

À toi, avec aide · réponse

Réponse (suite)

```
p.ajouter("HEF-010", 30)
p.retirer("HEF-000")
print(p.compter(), p.total())   # 1 30
```

À toi, en solo

Choisis **trois situations** de ta vie (ou d'une appli que tu aimes) et pour chacune :

1. liste les opérations nécessaires ;
2. nomme l'ADT le plus adapté (liste, pile, file, dictionnaire, ensemble) ;
3. écris l'interface en une phrase par opération.

Exemple de situations : historique de navigation, annuaire de contacts, file d'attente à une caisse.

À toi, en solo · réponse

Pistes de correction

Exemples de bonnes réponses : historique de navigation → **pile** (retour = dépiler) ; annuaire → **dictionnaire** (nom → numéro) ; file à la caisse → **file** (premier arrivé, premier servi). Une bonne réponse nomme les opérations *avant* de nommer la structure.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Choisir la structure avant l'usage	Mauvais choix, tout ralentit	Lister d'abord les opérations
Exposer l'implémentation	Le code utilisateur dépend d'un détail	Passer par des méthodes
ADT flou	Chacun interprète autrement	Écrire le contrat de chaque opération
Tout mettre dans une liste	Recherches lentes	Se demander : qu'est-ce que je fais le plus souvent ?

Mini-boss

Mini-boss

Tu conçois le **carnet de mentors** de HashBox. Besoins : ajouter un mentor, le retrouver par son nom, le supprimer, lister les mentors disponibles un jour donné.

1. Écris l'interface (opérations + contrat).
2. Implémente une première version avec une **liste de dictionnaires**.
3. Implémente une seconde version avec un **dictionnaire nom → mentor**.
4. Vérifie avec les mêmes tests que les deux versions se comportent pareil.
5. Dis laquelle serait plus rapide pour « retrouver par nom » et pourquoi.

Mini-boss · réponse

Pistes de correction

Un bon travail : les deux classes ont la même interface (`ajouter`, `trouver`, `supprimer`, `disponibles`) ; un seul fichier de tests s'applique aux deux ; la version dictionnaire retrouve par nom sans parcourir toute la liste. L'explication évoque que la liste doit être parcourue élément par élément alors que le dictionnaire accède directement par la clé.

Quiz éclair

Quiz éclair

1. Un ADT décrit une structure par :

A. ses opérations
B. son code interne
C. sa taille mémoire

2. L'interface d'une structure, c'est :

A. comment elle est faite à l'intérieur
B. ce qu'on peut faire avec
C. son nom de fichier

3. Pour choisir une structure, on commence par :

A. choisir la plus récente

Quiz éclair

Quiz éclair (suite)

B. utiliser toujours une liste
C. lister les opérations les plus fréquentes

4. 🔁 *Révision HEF-010, mission 08* : Un cas limite typique est :

A. la liste vide
B. la liste moyenne
C. le cas heureux

5. 🔁 *Révision HEF-010, mission 03* : Un contrat comprend :

A. uniquement un nom
B. préconditions et postconditions
C. du code

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Quelle opération fais-tu le plus souvent dans ton rangement quotidien ? Est-il adapté ?
2. Quel changement d'implémentation pourrais-tu faire sans que personne ne s'en aperçoive ?

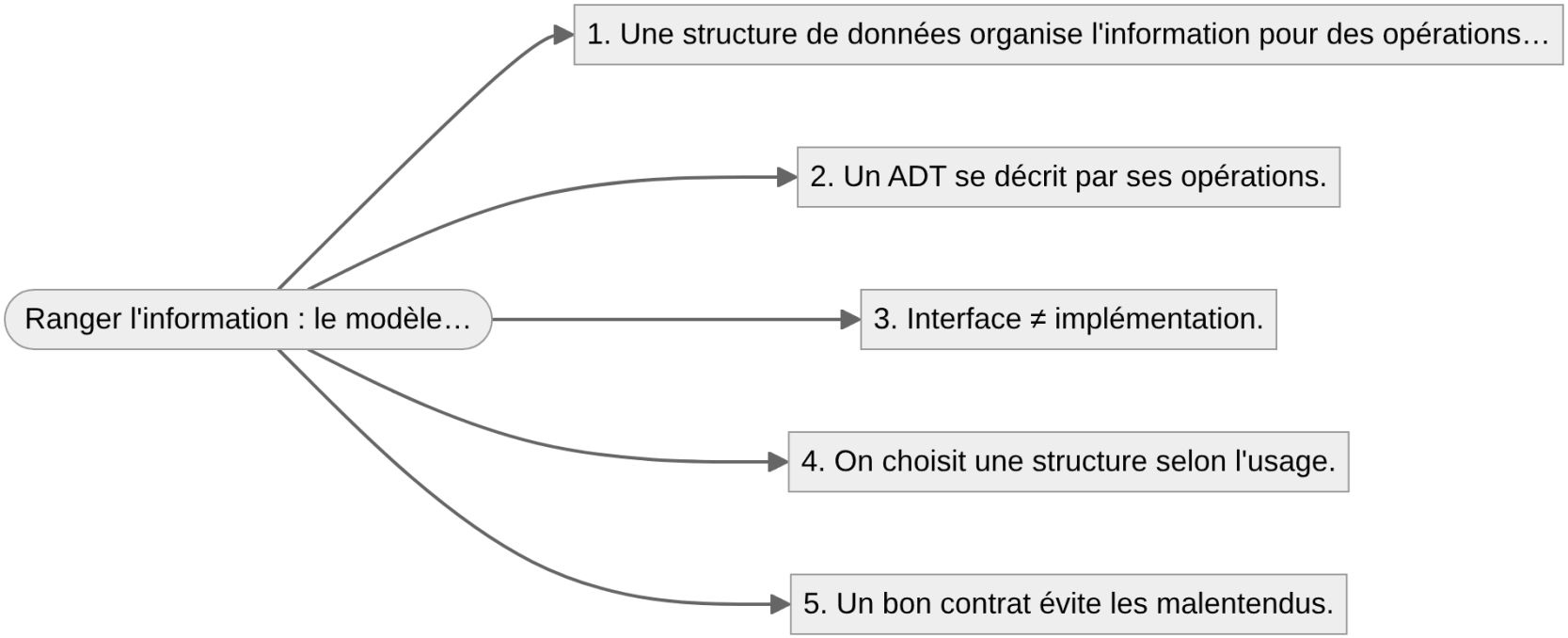
À retenir

À retenir

1. Une structure de données organise l'information pour des opérations précises.
2. Un ADT se décrit par ses opérations.
3. Interface ≠ implémentation.
4. On choisit une structure selon l'usage.
5. Un bon contrat évite les malentendus.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Architecte du rangement

Prochaine mission : Tableaux et listes chaînées

