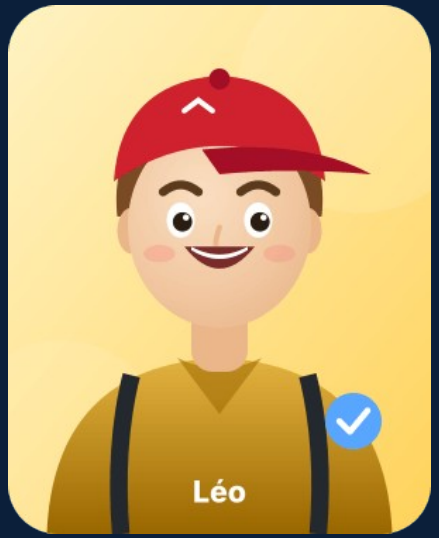
HEF-010 · MISSION 04 / 10

Récursion : se résoudre soi-même

Pour résoudre un gros problème, résous un problème plus petit… du même genre.





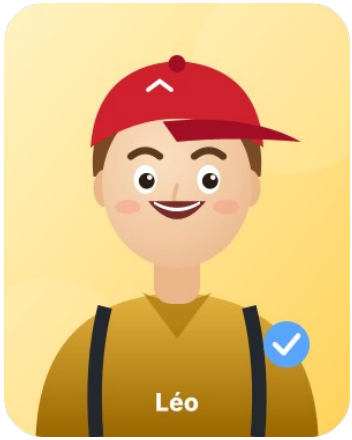
Badge : Maître des poupées russes

75 min · débutant

L'histoire

Les dossiers de HashBox sont rangés dans des dossiers, qui contiennent des dossiers, qui contiennent des dossiers. Maya veut **compter tous les fichiers**, à n'importe quelle profondeur.
Kodi écrit trois boucles imbriquées. Puis quatre. Puis il abandonne.

> *« Et si un dossier se comptait lui-même de la même façon ? » propose Sam.*



Bienvenue dans la récursion.

Ta mission

À la fin, je serai capable de…

1. expliquer ce qu'est une fonction récursive
2. identifier le cas de base et le cas récursif
3. faire la trace d'un appel récursif
4. expliquer pourquoi l'absence de cas de base est un bug
5. comparer une solution récursive et une boucle

Défi d'échauffement

Défi d'échauffement

Tu es dans une file d'attente et tu veux savoir **ta position**, mais tu ne vois pas le début de la file. Tu peux seulement demander à la personne devant toi. Comment obtiens-tu ta position ? Écris la règle que chaque personne suit.

Le concept

L'idée

Une fonction est **récursive** quand elle s'appelle elle-même, sur un problème **plus petit**.
Analogie : les poupées russes. Pour compter les poupées, tu en ouvres une, tu comptes 1, puis tu comptes les poupées de l'intérieur de la même façon. Quand tu ouvres une poupée vide, tu t'arrêtes.

Les deux ingrédients

1. **Cas de base** : le cas si simple qu'on répond directement. Sans lui, la fonction ne s'arrête jamais.
2. **Cas récursif** : on réduit le problème et on s'appelle.

Le concept

Les deux ingrédients (suite)

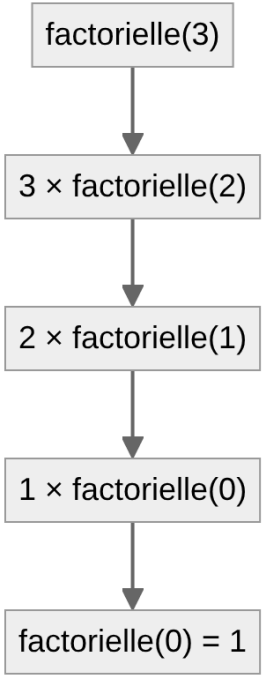
```
def factorielle(n):
    if n == 0:          # cas de base
        return 1
    return n * factorielle(n - 1)   # cas récursif
```

Ce qui se passe en mémoire

Chaque appel attend le résultat du suivant. On empile les appels, puis on les dépile en remontant.

Le concept

Ce qui se passe en mémoire (suite)



Le concept

Ce qui se passe en mémoire (suite)

Description du schéma : factorielle(3) appelle factorielle(2), qui appelle factorielle(1), qui appelle factorielle(0). Ce dernier renvoie 1, et les résultats remontent en se multipliant.

Le danger

Sans cas de base, ou si le problème **ne diminue pas**, les appels s'empilent jusqu'à l'erreur `RecursionError`. En Python, la limite par défaut est d'environ 1000 appels.

> 🧠 ***Astuce en 3 questions*** *: (1) Quel est le cas de base ? (2) Le problème diminue-t-il à chaque appel ? (3) Si j'ai le résultat du plus petit, comment construire le mien ?*

Exemple pas à pas

Problème : calculer la somme d'une liste, sans boucle.

Réflexion :

- Cas de base : liste vide → 0.
- Cas récursif : somme = premier élément + somme du reste.

```
def somme(liste):
    if len(liste) == 0:
        return 0
    return liste[0] + somme(liste[1:])

print(somme([4, 2, 7]))   # 13
```

Trace :

Exemple pas à pas

Exemple pas à pas (suite)

```
somme([4, 2, 7])
= 4 + somme([2, 7])
= 4 + (2 + somme([7]))
= 4 + (2 + (7 + somme([])))
= 4 + (2 + (7 + 0))
= 13
```

Remarque : ici une boucle est plus simple et plus rapide. La récursion brille quand le problème est **naturellement imbriqué** : dossiers, arbres, grammaires.

À toi, avec aide

Écris `puissance(x, n)` qui calcule x puissance n (n entier >= 0), **sans** utiliser `**`.

1. Cas de base ?
2. Cas récursif ?
3. Teste : `puissance(2, 5)` = 32 ; `puissance(7, 0)` = 1.

À toi, avec aide · réponse

Indices progressifs

Indice 1

Quand n vaut 0, le résultat est 1.

Indice 2

x puissance n = x × (x puissance n−1).

Indice 3

Vérifie que n diminue de 1 à chaque appel.

À toi, avec aide · réponse

Réponse

```
def puissance(x, n):
    if n == 0:
        return 1
    return x * puissance(x, n - 1)

print(puissance(2, 5))  # 32
print(puissance(7, 0))  # 1
```

À toi, en solo

Un dossier est représenté par un dictionnaire : `{"nom": ..., "fichiers": nombre, "sous_dossiers": [dossiers...]}`.

Écris `compter_fichiers(dossier)` qui renvoie le nombre total de fichiers, à toute profondeur. Teste avec un dossier qui contient 2 fichiers et deux sous-dossiers de 3 et 4 fichiers (total 9).

À toi, en solo · réponse

Pistes de correction

```
def compter_fichiers(dossier):
    total = dossier["fichiers"]
    for sous in dossier["sous_dossiers"]:
        total = total + compter_fichiers(sous)
    return total

racine = {"nom": "racine", "fichiers": 2, "sous_dossiers": [
    {"nom": "a", "fichiers": 3, "sous_dossiers": []},
    {"nom": "b", "fichiers": 4, "sous_dossiers": []},
]}
print(compter_fichiers(racine))  # 9
```

Le cas de base est implicite : un dossier sans sous-dossiers ne déclenche aucun appel.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Oublier le cas de base	`RecursionError`	Écrire le cas de base en premier
Le problème ne diminue pas	Boucle d'appels infinie	Vérifier que l'argument se rapproche du cas de base
Récursion là où une boucle suffit	Plus lent et plus fragile	Choisir la récursion pour les structures imbriquées
Recalculer les mêmes sous-problèmes	Exemple : Fibonacci naïf devient très lent	Mémoriser les résultats (on y reviendra)

Mini-boss

Mini-boss

Écris trois fonctions récursives :

1. `inverser(texte)` qui renvoie le texte à l'envers ;
2. `est_palindrome(texte)` qui renvoie True si le texte se lit pareil dans les deux sens ;
3. `aplatir(liste)` qui transforme `[1, [2, [3, 4]], 5]` en `[1, 2, 3, 4, 5]`.

Pour chacune, écris le cas de base et le cas récursif en une phrase avant le code.

Mini-boss · réponse

Pistes de correction

```
def inverser(texte):
    if len(texte) <= 1:
        return texte
    return inverser(texte[1:]) + texte[0]

def est_palindrome(texte):
    if len(texte) <= 1:
        return True
    return texte[0] == texte[-1] and est_palindrome(texte[1:-1])

def aplatir(liste):
    resultat = []
    for element in liste:
        if isinstance(element, list):
            resultat = resultat + aplatir(element)
        else:
            resultat.append(element)
    return resultat
```

Mini-boss · réponse

Pistes de correction (suite)

```
print(inverser("kodi"))                  # idok
print(est_palindrome("radar"))           # True
print(aplatir([1, [2, [3, 4]], 5]))      # [1, 2, 3, 4, 5]
```

Quiz éclair

Quiz éclair

1. Un cas de base sert à :

A. arrêter les appels récursifs
B. accélérer le code
C. nommer la fonction

2. Sans cas de base, on obtient :

A. un résultat correct
B. une erreur de récursion infinie
C. une boucle for

3. La récursion est particulièrement adaptée :

A. aux additions simples

Quiz éclair

Quiz éclair (suite)

B. aux affichages
C. aux structures imbriquées comme les dossiers

4. 🔁 *Révision mission 03* : Un invariant de boucle est :

A. vrai avant, pendant et après chaque tour
B. un commentaire
C. un nom de variable

5. 🔁 *Révision mission 03* : Une précondition est :

A. la valeur de retour
B. ce que l'appelant doit fournir
C. un test

Quiz éclair · réponse

Réponses

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

Je réfléchis

1. Dans l'exemple de la file d'attente, quel était ton cas de base ?
2. Quelle situation de ta vie a une structure « poupées russes » ?

À retenir

1. Une fonction récursive s'appelle sur un problème plus petit.
2. Il faut toujours un cas de base.
3. Le problème doit diminuer à chaque appel.
4. Les appels s'empilent puis se dépilent.
5. Récursion = idéale pour les structures imbriquées.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Maître des poupées russes

Prochaine mission : Chercher efficacement

