3. Exemple pratique 2 : Approche inspirée de la vie réelle (40 min)

Objectif : Utiliser la récursivité pour explorer un arbre généalogique et calculer la profondeur d'une famille

Dans ce projet, nous allons modéliser un **arbre généalogique** en utilisant la récursivité pour explorer les générations successives d'une famille. L'objectif est de calculer la **profondeur de l'arbre**, c'est-à-dire la distance maximale entre la personne la plus ancienne (la racine) et l'une de ses descendants les plus jeunes (les feuilles). Cela nous permettra de savoir combien de générations séparent la racine de la feuille la plus éloignée.

Contexte : Un arbre généalogique

Dans un arbre généalogique :

- Chaque personne a un **nom**.
- Chaque personne peut avoir une liste d'**enfants**.
- Les personnes au sommet de l'arbre (sans parents) sont des **racines**, et celles sans enfants sont des **feuilles**.

L'objectif est d'utiliser la récursivité pour traverser cet arbre et déterminer la profondeur la plus longue, correspondant au nombre de générations entre la racine et la feuille la plus éloignée.

Étape 1 : Modélisation d'une personne dans l'arbre

Nous allons créer une classe `Personne` pour représenter chaque individu dans l'arbre généalogique. Chaque `Personne` aura deux propriétés :

- **nom** : le nom de la personne.
- **enfants** : une liste d'enfants.

Voici comment modéliser une personne en Python :

```
class Personne:
    def __init__(self, nom):
        self.nom = nom
        self.enfants = []

    def ajouter_enfant(self, enfant):
        self.enfants.append(enfant)
```

Explication :

- La méthode `__init__` initialise un objet `Personne` avec un **nom** et une **liste vide d'enfants**.
- La méthode `ajouter_enfant` permet d'ajouter un enfant à la liste `enfants`.

Étape 2 : Création d'un arbre généalogique

Nous allons maintenant construire un exemple d'arbre généalogique avec plusieurs générations.

```
# Création des personnes
```

```
grand_pere = Personne("Grand-père")
pere = Personne("Père")
oncle = Personne("Oncle")
enfant1 = Personne("Enfant 1")
enfant2 = Personne("Enfant 2")

# Ajouter les enfants
grand_pere.ajouter_enfant(pere)
grand_pere.ajouter_enfant(oncle)
pere.ajouter_enfant(enfant1)
pere.ajouter_enfant(enfant2)
```

Dans cet arbre généalogique :

- **Grand-père** a deux enfants : **Père** et **Oncle**.
- **Père** a deux enfants : **Enfant 1** et **Enfant 2**.

Cet arbre peut être visualisé comme suit :

```
Grand-père
  ├── Père
  │    ├── Enfant 1
  │    └── Enfant 2
  └── Oncle
```

Étape 3 : Fonction récursive pour calculer la profondeur

L'idée est de créer une fonction récursive qui va explorer chaque branche de l'arbre généalogique pour déterminer sa **profondeur maximale**. Cette profondeur correspond au nombre de générations entre la racine (le grand-père) et la feuille la plus éloignée (le dernier enfant sans descendants).

La logique de la fonction récursive :

1. **Cas de base** : Si une personne n'a pas d'enfants, elle est une feuille et la profondeur est de 1 (car c'est une génération).
2. **Cas récursif** : Si une personne a des enfants, on calcule la profondeur de chaque enfant, puis on prend le maximum de ces profondeurs et on ajoute 1 pour inclure la génération actuelle.

Voici le code Python pour cette fonction :

```
def profondeur_arbre(personne):
    # Cas de base : la personne n'a pas d'enfants
    if not personne.enfants:
        return 1
    else:
        # Calculer la profondeur de chaque enfant
        profondeurs = [profondeur_arbre(enfant) for enfant in personne.enfants]
        # Prendre la profondeur maximale des enfants et ajouter 1 (pour la
génération actuelle)
        return 1 + max(profondeurs)
```

Explication détaillée :

- **Cas de base** : Si la personne n'a pas d'enfants (`if not personne.enfants`), on renvoie `1`, car cette personne est une feuille.

- **Cas récursif** : Pour chaque enfant, on appelle récursivement la fonction `profondeur_arbre(enfant)`. Ensuite, on prend la **profondeur maximale** des enfants et on ajoute `1` pour inclure la génération actuelle.

Étape 4 : Calcul de la profondeur de l'arbre généalogique

Nous allons maintenant utiliser la fonction pour calculer la profondeur totale de l'arbre généalogique à partir du **grand-père**, qui est la racine de l'arbre.

```
# Calcul de la profondeur de l'arbre
profondeur_totale = profondeur_arbre(grand_pere)
print(f"La profondeur de l'arbre généalogique est de {profondeur_totale}.")
```

Sortie attendue :

```
La profondeur de l'arbre généalogique est de 3.
```

Explication :

- **Grand-père** est à la **génération 1**.
- **Père** et **Oncle** sont à la **génération 2**.
- **Enfant 1** et **Enfant 2** sont à la **génération 3**.

La génération la plus profonde de cet arbre est donc **3**, car c'est le nombre de générations qui séparent le grand-père (racine) des enfants (feuilles).

Objectif atteint :

Grâce à la récursivité, nous avons exploré chaque branche de l'arbre généalogique pour calculer la profondeur maximale. Chaque appel récursif descend dans une branche de l'arbre (une nouvelle génération), jusqu'à ce qu'il atteigne une personne sans enfants (une feuille).

Améliorations possibles :

1. **Affichage de l'arbre** : Vous pouvez ajouter une fonctionnalité pour afficher visuellement l'arbre généalogique avec les niveaux de génération.
2. **Recherche de la génération la plus petite** : En plus de la profondeur maximale, vous pourriez aussi calculer la génération minimale si certaines branches de l'arbre s'arrêtent plus tôt.
3. **Arbres plus complexes** : Vous pouvez étendre cet exemple avec plus de générations et des structures familiales plus compliquées.

Cet exemple montre comment la récursivité permet d'explorer des structures hiérarchiques, comme les arbres généalogiques, de manière simple et efficace. C'est un modèle utile pour comprendre comment gérer la récursion dans des problèmes liés à la vie réelle, notamment dans des systèmes où les relations parent-enfant sont clés.