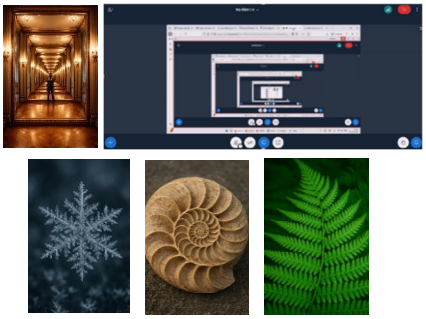
La récursivité

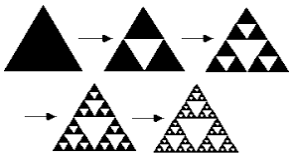
Cours

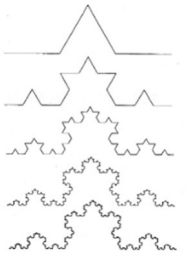
☑ Comprendre la notion de récursivité.

1. Qu'est-ce-que le principe de récursivité ?

Dans la vie quotidienne comme dans la nature, le principe de la récursivité est visible dans de nombreuses formes : le reflet d'un miroir dans un autre miroir ou l'écran partagé d'un ordinateur dans une visioconférence qui se répètent à l'infinie, un flocon de neige dont chaque branche ressemble au motif principal, une coquille de nautile qui s'enroule en spirale selon une structure répétée, ou encore une fougère où chaque segment reproduit l'ensemble de la plante.







La récursivité constitue le fondement même des fractales. Une fractale est une figure dont chaque partie reproduit la forme générale, mais à des échelles différentes.

Le triangle de Sierpinski et la courbe de Koch sont construits par répétition d'une même règle simple. Ces objets illustrent parfaitement la puissance de la récursivité : la répétition d'une fonction simple pour former une structure complexe.

2. La récursivité en informatique.

En programmation, la récursivité est une approche dans laquelle **une fonction s'appelle elle-même pour résoudre un problème en le divisant en sous-problèmes plus simples**.

Fonction puissance_aux

Fonction puissance_aux appelée

```
def puissance_aux(a,b):
    if b == 0: # Cas de base
        return 1
    return a * puissance_aux(a,b-1)

def puissance(a,b): #b est un nombre entier positif
    if b >= 0 and type(b) == int:
        return puissance_aux(a,b)
```

Boucle infinie

Pour qu'une fonction récursive fonctionne correctement il faut respecter quelques règles simples afin d'**éviter les boucles infinies.**

- ✓ **Un argument différent** `puissance_aux(a,b):` ≠ `puissance_aux(a,b-1)`
- ✓ **Un cas de base :** `if b == 0:`
- ✓ **Une vérification de la saisie des arguments par les utilisateurs le cas échéant**

```
if b >= 0 and type(b) == int:
```

La récursivité

Cours

☑ Analyser le fonctionnement d'un programme récursif.

3. Analysons le programme ci-dessous.

```
def somme_liste(lst):
    """
    Retourne la somme des éléments de lst (liste d'entiers) de façon récursive.
    Exemple : somme_liste([1, 2, 3]) -> 6
    """
    if not lst:            # cas de base : liste vide
        return 0
    return lst[0] + somme_liste(lst[1:])  # étape récursive
```

b. Ce programme est-il récursif ?

La fonction `somme_list` s'appelle-t-elle elle-même ?

Oui, ici la fonction s'appelle elle-même en retournant le premier élément de la liste et en le supprimant de celle-ci : `somme_liste(lst[1:])`

La fonction `somme_list` contient-elle un cas de base ?

Oui, ici la fonction retourne 0 quand la liste est vide : `if not lst:`

✓ Cette fonction est donc récursive.

a. Quels sont les différentes étapes pour la liste suivante [5, 3, 8, 15] ?

Dans un programme récursif, les différentes étapes intermédiaires sont stockées successivement en mémoire dans ce que l'on appelle une **pile**. En Python l'instruction `lst[1:]` renvoie la liste sans le premier élément de la liste.

Voici les étapes du processus lorsque l'utilisateur tape la commande :
`somme_list ([5, 3, 8, 15])`