Activité - Récursivité

1. **Fonction récursive - Définition**

En informatique et en mathématiques, une fonction qui s'appelle elle-même est appelée fonction récursive.

2. Un exemple pour comprendre la notion de récursivité

Considérons une fonction f qui, à tout réel x, associe 3x et imaginons qu'on souhaite lui faire calculer 3^5. Comment faire ?

$$1 \xmapsto{f} 3 \times 1 = 3 \quad a$$

$$3 \xmapsto{f} 3 \times 3 = 3^2 \quad b$$

$$3^2 \xmapsto{f} 3^2 \times 3 = 3^3 \quad c$$

$$3^3 \xmapsto{f} 3^3 \times 3 = 3^4 \quad d$$

$$3^4 \xmapsto{f} 3^4 \times 3 = 3^5 \quad e$$

En Python, le code à écrire serait par exemple :

```python
def f(x):
    return 3*x

a = f(1)    # On calcule 3 à la puissance 1
b = f(a)    # On calcule 3 à la puissance 2
c = f(b)    # On calcule 3 à la puissance 3
d = f(c)    # On calcule 3 à la puissance 4
e = f(d)    # On calcule 3 à la puissance 5
print(f"Le nombre 3 à la puissance 5 est égal à {e}.")
```

Résultat affiché dans la console Python :

```
Le nombre 3 à la puissance 5 est égal à 243.
```

On pourrait aussi écrire d'une manière plus compacte le code suivant :

```python
def f(x):
    return 3*x

e = f(f(f(f(f(1)))))    # On calcule 3 à la puissance 5
print(f"Le nombre 3 à la puissance 5 est égal à {e}.")
```

On constate que la fonction f appelle la fonction f qui appelle la fonction f qui appelle la fonction f qui appelle une dernière fois la fonction sur le nombre 1. Comme mentionné en introduction, une fonction qui s'appelle elle-même est une fonction récursive.

Limitation de la solution proposée

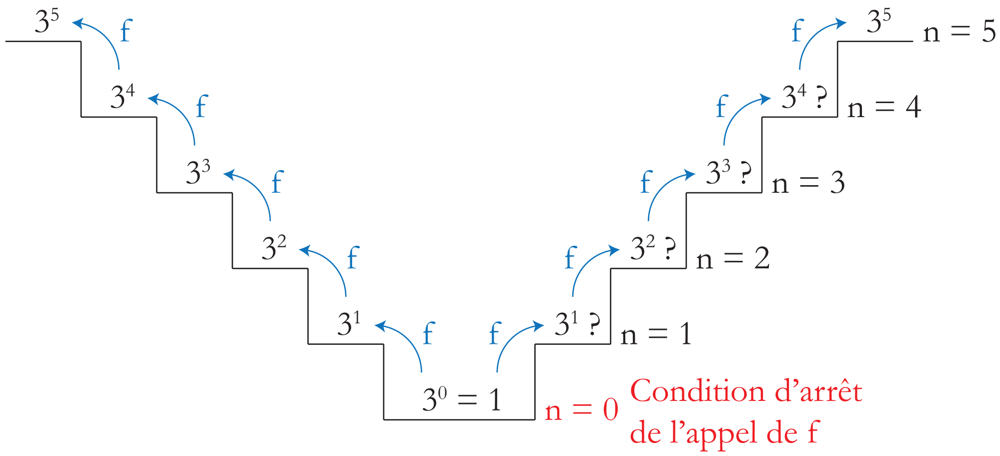
Le problème de ce dernier code est qu'il ne nous offre aucune souplesse pour le calcul de 3^8 ou 3^{25}. Il faudrait à chaque fois réécrire une suite imbriquée de huit f ou de vingt-cinq f dans le code pour obtenir le résultat demandé. Une modification du code s'avère donc nécessaire !

Nous souhaiterions pouvoir calculer 3^n pour n'importe quelle valeur entière de n, n étant un nombre choisi par l'utilisateur du programme. Le programme, une fois écrit, n'a donc pas à être modifié.
On veut implémenter la fonction p telle que :

$$n \xmapsto{p} \underbrace{3^n}_{p(n)}$$

Comment faire ?

Examinons à l'aide d'un schéma ce qui se passe lorsque l'on veut calculer 3^5 avec f ?



Pour calculer 3^5, la fonction f a besoin de 3^4, mais :
  pour calculer 3^4, la fonction f a besoin de 3^3, mais :
    pour calculer 3^3, la fonction f a besoin de 3^2, mais :
      pour calculer 3^2, la fonction f a besoin de 3^1, mais :
        pour calculer 3^1, la fonction f a besoin de 3^0, c'est-à-dire 1.

Une fois 3^0, c'est-à-dire 1, donné. La fonction calcule 3, 3^2, 3^3, puis 3^4 et retourne enfin 3^5.

Ce processus s'implémente mathématiquement aisément à l'aide de la fonction récursive définie par :

$$p(n) = \begin{cases} si\ n = 0, alors\ retourne\ 1 \\ sinon\ retourne\ 3 \times p(n-1) \end{cases}$$

Pourquoi p(5) est-il égal à 3^5 avec cette curieuse définition ?

Détails des calculs réalisés par la fonction récursive p

$p(5)$ $n = 5$

$n = 0$?

$3 \times p(4)$ $n = 4$

$n = 0$?

$3 \times (3 \times p(3))$ $n = 3$

$n = 0$?

$3 \times (3 \times (3 \times p(2)))$ $n = 2$

$n = 0$?

$3 \times (3 \times (3 \times (3 \times p(1))))$ $n = 1$

$n = 0$?

$3 \times (3 \times (3 \times (3 \times (3 \times p(0)))))$ $n = 0$

$n = 0$

$3^5 \times 1 = 3^5$

Implémentation en langage Python de la fonction récursive p

Le code proposé ci-dessous résout le problème posé.

```
def p(n):    # Définition de la fonction récursive p
    if n == 0:
        return 1
    else:
        return 3 * p(n - 1)

# Saisie de la puissance

n = int(input("Entrer la puissance à laquelle vous souhaitez élever le
nombre 3 : "))

# Affichage du résultat

print(f"Le nombre 3 à la puissance {n} est égal à {p(n)}.")
```

Exemple de résultat affiché :

```
Entrer la puissance à laquelle vous souhaiter élever le nombre 3 : 8
Le nombre 3 à la puissance 8 est égal à 6561.
```

Le concept de récursivité est un concept très puissant mais qui est considéré comme délicat à manipuler. Son utilisation requiert prudence et rigueur.

3. Exercice

On souhaite écrire un programme qui permette de calculer $1 \times 2 \times 3 \times 4 \times ... \times n$, où n est un nombre entier. On note n! un tel nombre et on le prononce "factorielle n".

3.1. Schématiser la situation.

3.2. Proposer une fonction récursive f adaptée à la résolution du problème.

3.3. Proposer un programme en Python qui, pour un nombre entier naturel n entré au clavier, retourne n!