Récursivité

1 Un premier exemple

La somme des n premiers entiers est la somme :

$$0 + 1 + 2 + \cdots + (n-1) + n \tag{1}$$

On peut calculer cette somme à l'aide du programme suivant :

```python
def somme(n):
    s = 0
    for i in range(n+1):
        s += i
    return s
```

La fonction `somme(n)` est **itérative**, à l'aide d'une boucle `for` elle répète des instructions un certain nombre de fois.

Une autre façon de programmer `somme` est de remarquer que `somme(n) = somme(n-1) + n` :

$$\text{somme(n)} = \underbrace{\boxed{0 + 1 + 2 + \cdots + (n-1)}}_{\text{somme(n-1)}} + n \tag{2}$$

Cette formule est valable pour `n` entier strictement positif, pour `n = 0`, on a simplement `somme(0) = 0`.

On obtient une **formulation récursive**, la définition de `somme` fait appel à `somme` :

$$\text{somme(n)} = \begin{cases} 0 & \text{si } n = 0 \\ n + \text{somme(n-1)} & \text{si } n > 0 \end{cases}$$

Cette formulation donne directement le programme suivant :

```python
def somme(n):
    if n == 0:
        return 0
    else:
        return n + somme(n-1)
```

La nouvelle fonction `somme` ainsi obtenue est une **fonction récursive**, elle contient un appel à elle-même.

2 Fonction récursive

Une fonction est dite récursive si elle contient un appel à elle-même. Elle est toujours constituée de un ou plusieurs **cas de base (ou conditions d'arrêt)** et de un ou plusieurs **cas récursifs**.
Les cas récursifs sont ceux qui renvoient à la fonction entrain d'être définie et les cas de base sont ceux qui donne un résultat.

```python
def somme(n):
    if n == 0:
        return 0      <---- cas de base
    else:
        return n + somme(n-1)   <--- cas récursif
```

L'appel à `somme(n-1)` dans le corps de la fonction est un **appel récurssif**.

Prenons pour autre exemple la suite de Fibonacci qui doit son nom à Leonardo Fibonacci. Dans un problème récréatif posé dans l'ouvrage *Liber abaci* publié en 1202, il y décrit la croissance d'une population de lapins. La fonction `fibonacci(n)` donne le nombre de lapins au bout de `n` mois et est définie de la manière suivante :

$$\text{fibonacci}(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{fibonacci}(n-2) + \text{fibonacci}(n-1) & \text{si } n > 1 \end{cases}$$

Ce qui donne la fonction python ci-dessous :

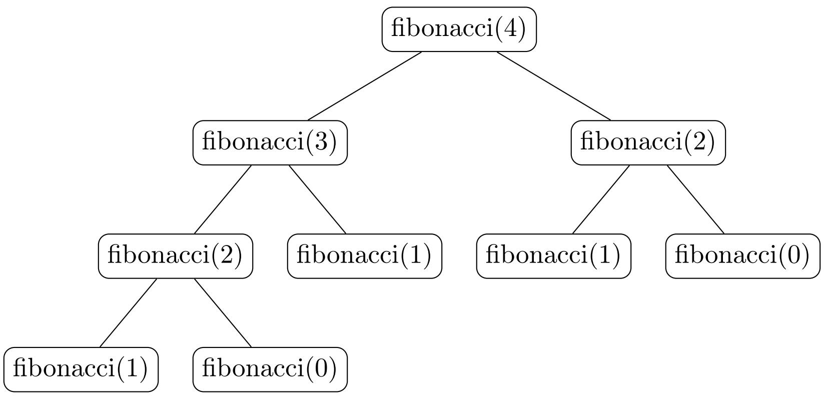
```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

cas de base

cas récursif double

Cette fonction a deux cas de base et un cas **récursif double**, il y a deux appels récursifs dans le cas récursif.

L'ensemble des appels récursifs lors de l'éxécution de `fibonacci(n)` pour un certain entier positif `n` peut être représenté sous la forme d'un **arbre des appels**. Par exemple pour `fibonacci(4)`, on obtient l'arbre suivant :



Auteur : Pascal Seckinger