# Calcul d'une somme ou d'un produit de termes

Parmi les procédures les plus courantes, nous serons souvent amenés à calculer les termes successifs d'une somme ou d'un produit. S'il existe de nombreuses façons de procéder, on retiendra que celles-ci reposent avant tout sur une **structure itérative**.

Par exemple, si on cherche à déterminer la somme des $n$ premiers cubes définie pour tout $n \in \mathbb{N}$ par :

$$S_n = \sum_{k=0}^{n} k^3$$

alors on initialise une variable $S$ à 0, puis on ajoute à chaque étape la valeur $k^3$ pour $k$ parcourant les entiers de 0 à $n$.

Concrètement, on pourra utiliser une boucle `for` ou `while` à l'aide d'un compteur :

```python
def somme(n):
    S=0
    for k in range(0,n+1):
        S=S+k**3
    return S
```

```python
def somme(n):
    S,k=0,0
    while k <= n:
        S=S+k**3
        k=k+1
    return S
```

**Remarques**

1. Dans ce dernier programme, on pourra observer que $S$ et $k$ ont été affectés simultanément : on parle aussi d'**affectations en parallèle**... et c'est là une des facilités propre au langage Python.
2. On aurait aussi pu initialiser $S$ à 1, la première valeur significative de la somme à condition d'adapter les valeurs prises par $k$ : on veillera donc à ce que le compteur suive le calcul réellement effectué.

De la même façon, on peut calculer le produit $P_n$ défini pour tout $n \in \mathbb{N}^*$ par :

$$P_n = \prod_{k=1}^{n} \frac{k+1}{k}$$

Il suffit cette fois d'initialiser une variable $P$ à 1, puis on multiplie à chaque étape par $\dfrac{k+1}{k}$ pour $k$ parcourant les entiers de 1 à $n$.

Concrètement, on aura encore :

```python
def produit(n):
    P=1
    for k in range(0,n+1):
        P=P*(k+1)/k
    return P
```

```python
def produit(n):
    P,k=1,1
    while k <= n:
        P=P*(k+1)/k
        k=k+1
    return P
```

Pour finir, on rappelle qu'il existe une commande déjà intégrée au langage, la commande `sum`. Si dans la plupart des cas, on nous demandera de redéfinir une telle procédure à l'aide des boucles `for` ou `while`, on retiendra qu'on peut aussi lancer les instructions suivantes :

```
In : L=[k**3 for k in range(0,10)]; sum(L)
                                                2025
```

Malheureusement, il n'existe pas de fonction similaire pour obtenir le produit des éléments d'une même liste.

## Exercice 1 - Calcul de la somme des coefficients binomiaux

Soit $n \in \mathbb{N}$. On rappelle que pour tout $k \in [\![0, n]\!]$, on pose : $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$.

1. Construire la fonction *facto* qui pour tout entier $n$ donné, renvoie la valeur de $n!$ définie par :
$$n! = \begin{cases} \prod_{k=1}^{n} k \text{ , si } n \geq 1 \\ 1 \text{ , si } n = 0 \end{cases}$$
2. Définir le programme *somme* qui pour tout entier $n$ donné, renvoie la somme $(S_n)$ définie par $S_n = \sum_{k=0}^{n} \binom{n}{k}$.
3. Donner alors pour tout $n \in \mathbb{N}$, $S_n$ sous une forme explicite. Justifier rapidement votre résultat.

## Exercice 2 - Comportement asymptotique de $\zeta(x)$

Sous réserve d'existence, on définit la fonction $\zeta$ par :
$$\zeta(x) = \sum_{k=1}^{+\infty} \frac{1}{k^x} = \lim_{n \to +\infty} \sum_{k=1}^{n} \frac{1}{k^x}$$
et on pose pour tout $n \in \mathbb{N}^*$, $S_n = \sum_{k=1}^{n} \dfrac{1}{k^x}$.

1. Dans le langage Python, construire la fonction *zeta* qui, pour tout couple $(x, n)$ donné, renvoie la valeur de $S_n$.
2. Modifier votre programme pour que celui-ci renvoie, à chaque étape, le couple $(k, S_k)$ et ceci afin d'étudier le comportement asymptotique de la suite $(S_n)$.
3. Tester votre programme pour différentes valeurs de $x$ et pour $n = 1000$. Que pouvez-vous en déduire ?
4. On fixe $x = 2$ et on admet que $\zeta(2) = \frac{\pi^2}{6}$.
Construire la fonction *approximation* qui pour tout $\epsilon > 0$ donné, calcule les valeurs de $S_n$ tant que $|S_n - \frac{\pi^2}{6}| > \epsilon$, puis renvoie le premier entier $n_0$ pour lequel :
$$|S_{n_0} - \frac{\pi^2}{6}| \leq \epsilon$$