

TD 1 : Syntaxe de base et fonctions

Exercice 1 :

1. Ecrire une fonction « **somme_div(n)** » qui retourne la somme de tous les diviseurs d'un nombre passé en paramètre. (n doit être positif)

Un nombre entier est parfait s'il est égal à la somme de ses diviseurs (sauf lui-même).

Ex : $6 = 1 + 2 + 3$ est parfait.

2. Ecrire une fonction « **parfait (m)** » qui teste si un nombre passé en paramètre **m** est parfait et qui retourne **True** s'il l'est et **False** sinon.

3. Ecrire une fonction « **premier (x)** » qui teste si un nombre passé en paramètre **x** est premier et qui retourne **True** s'il l'est et **False** sinon.

Exercice 2 :

Un **nombre narcissique** (ou nombre d'Armstrong) est un entier naturel non nul égal à la somme de ses chiffres élevés à la puissance (p), où (p) est le nombre de chiffres du nombre.

Par exemple : $153 = 1^3 + 5^3 + 3^3$ et $548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6$

Travail demandé :

Écrire un programme Python permettant de vérifier si un entier n saisi par l'utilisateur est narcissique.

Le programme devra être décomposé en fonctions :

nombre_chiffres(n) # Retourne le nombre de chiffres de n supposé positif

somme_puissances_chiffres(n, p) # Retourne la somme des chiffres de n élevés à la puissance p

est_narcissique(n) # Retourne True si n est narcissique et False sinon

Exercice 3 :

Un joueur lance un certain nombre de fléchettes. Chaque fléchette peut rapporter 0, 20 ou 50 points selon la zone touchée.

Par exemple, pour 10 fléchettes et un score total de 300 points, une possibilité est :

$3 \times 0 + 5 \times 20 + 2 \times 50 = 300$ avec $3 + 5 + 2 = 10$

Écrire une fonction : **possibilites(nb_flechettes, score)** qui affiche toutes les possibilités permettant d'obtenir exactement score points avec nb_flechettes lancées.

Exercice 4 :

Implémentez une fonction **fibo(n)** qui renvoie le n^{ème} terme de la suite de Fibonacci en utilisant à la fois une approche **itérative** et une approche **réursive**.

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_n = U_{n-2} + U_{n-1} \end{cases}$$

Exercice 5 : Approximation de π

On utilise la formule de Leibniz :

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

Écrire une fonction **approx_pi(n)** qui calcule une approximation de π à l'aide des n premiers termes. Tester pour n = 10, 100, 1000, 10000 et comparer avec math.pi.