Exercice 1 : Dans l'activité exponentiation la méthode de mesure de la durée d'exécution des fonctions n'est pas très rigoureuse. En effet, le temps mesuré prend en compte toute l'activité de la machine et pas seulement l'exécution de la fonction Python. Il est plus judicieux de compter le nombre d'opérations réalisées lors de l'exécution de la fonction.

- 1. À l'aide d'une variable gloable COMPTEUR, compter le nombre d'opérations effectuées par les fonctions puissance_perso, puissance_recursif et puissance_recursif_rapide lors du calcul de 2701¹⁹⁴⁰⁶.
- 2. Écrire la fonction puissance_iteratif_rapide qui reprend la méthode de calcul de puissance_recursif_rapide.

Remarque

Il est toujours possible de transformer un algorithme itératif en récursif et réciproquement.

3. Compter le nombre d'opérations de cette fonction.

Exercice 2 : La somme des entiers s'écrit :

$$0+1+2+...+n$$

- 1. Donner une définition récursive de la somme des entiers.
- 2. Implémenter la fonction somme (n: int) \rightarrow int.

Exercice 3 : La fonction factorielle est définie par :

$$n! = 1 \times 2 \times 3... \times n$$
 si $n > 0$ et $0! = 1$

- 1. Donner une définition récursive qui correspond au calcul de la fonction factorielle.
- 2. Implémenter la fonction factorielle(n: int) \rightarrow int.

Exercice 4 : Soit u_n la suite d'entiers définie par $u_0 > 1$ et :

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3 \times u_n + 1 & \text{sinon} \end{cases}$$

Écrire la fonction **syracuse(u:int)** \rightarrow **None** qui affiche les valeurs successives de la suite u_n tant que $u_n > 1$. L'appel de la fonction s'effectuera avec une valeur de u_0 quelconque.

Exercice 5:

- 1. Écrire une fonction récursive entiers(i: int, k: int) \rightarrow None qui affiche les entiers entre i et k. Par exemple, entiers(0,3) doit afficher 0 1 2 3.
- 2. Écrire une fonction récursive **impairs(i: int, k: int)** \rightarrow **None** qui affiche les nombres impairs entre i et k.

Exercice 6:

- 1. Construire par compréhension un tableau de 10 entiers aléatoires compris entre 1 et 100.
- 2. Écrire la fonction itérative somme (tab: list) \rightarrow int qui renvoie la somme des entiers de tab.
- 3. Écrire la fonction récursive **somme_rec** équivalente. Le nombre de paramètres pourra évoluer par rapport à la fonction précédente.

Exercice 7:

1. Construire par compréhension un tableau de 30 entiers aléatoires compris entre 1 et 1000.



- 2. Écrire la fonction itérative mini(tab: list) \rightarrow int qui renvoie le minimum de tab.
- 3. Écrire la fonction récursive mini_rec équivalente. Le nombre de paramètres pourra évoluer par rapport à la fonction précédente.

Exercice 8 : L'algorithme d'Euclide est l'un des plus anciens algorithmes (300 avant JC). Il permet de déterminer le Plus Grand Commun Diviseur (PGCD) de deux entiers.

Détermination du PGCD de $20\ {\rm et}\ 35$:

- $-35 = 20 \times 1 + 15$
- $-20 = 15 \times 1 + 5$
- $-15 = 5 \times 3 + 0$
- Le PGCD est 5.
- 1. Écrire la fonction itérative pgcd(a: int, b: int) → int qui renvoie le Plus Grand Commun Diviseur de a et b. On donne comme précondition : a < b.
- 2. Écrire la fonction récursive pgcd_rec(a: int, b: int) → int qui renvoie le Plus Grand Commun Diviseur de a et b.

Exercice 9 : Écrire une fonction récursive nombre_chiffres(n: int) \rightarrow int qui renvoie le nombre de chiffres qui compose n.

Exercice 10 : La formulation récursive ci-après permet de calculer les coefficients binomiaux :

$$C(n,p) = \left\{ \begin{array}{ll} 1 & \text{si } p=0 \text{ ou } n=p \\ C(n-1,p-1) + C(n-1,p) & \text{sinon} \end{array} \right.$$

- 1. Écrire une fonction récursive $C(n: int, p: int) \rightarrow int$ qui renvoie la valeur de C(n,p).
- 2. Le triangle de Pascal est une présentation des coefficients binomiaux sous la forme d'un triangle. Dessiner le triangle de Pascal à l'aide d'une double boucle **for** pour n variant de 0 à 10.

 $\begin{array}{c} 1 \\ 1 \ 1 \\ 1 \ 2 \ 1 \\ 1 \ 3 \ 3 \ 1 \\ 1 \ 4 \ 6 \ 4 \ 1 \\ \end{array}$

