
TRAVAUX PRATIQUES n° 12

Réversivité

Dans ce TP, toutes les fonctions doivent être écrites de manière **réursive** (sauf indication contraire).

Une fonction réursive repose toujours sur :

- ★ un **cas de base** (situation où l'on sait répondre sans rappel) ;
- ★ un **appel réursif** sur un problème plus petit, qui rapproche du cas de base.

Exercice 1 (*Introduction*) : Tester les fonctions réursives suivantes. Identifier le cas de base et l'appel réursif dans chaque fonction.

```
1 def factorielle(n) :  
2     if n == 0 :  
3         return 1  
4     else :  
5         return n * factorielle(n-1)  
6  
7 def somme(L) :  
8     if L == [] :  
9         return 0  
10    else :  
11        return L[0] + somme(L[1:])
```

Exercice 2 (*Calcul d'une somme*) : Écrire une fonction `somme_bicarree(n)` prenant en argument un entier naturel n et renvoyant la valeur de la somme $\sum_{k=0}^n k^4$.

Cas de base : $n == 0$.

Exercice 3 (*Réursivité sur les listes*) : On suppose dans cet exercice que les listes fournies sont non vides lorsqu'on calcule un minimum.

- 1) Écrire une fonction `produit(L)` prenant en argument une liste de nombres L et renvoyant le produit de tous les éléments de L .

Cas de base : $L == []$ (par convention, le produit vaut alors 1).

- 2) Écrire une fonction `minimum(L)` prenant en argument une liste de nombres L **non vide** et renvoyant le minimum des éléments de L .

Exercice 4 (*Fonction puissance*) :

- 1) Écrire une fonction `puissance_0(x, n)` **non réursive** qui, étant donnés un réel x et un entier naturel n , calcule et renvoie la valeur de x^n .
- 2) Écrire une fonction `puissance_1(x, n)` qui fait la même chose que `puissance_0` mais de manière réursive en utilisant la formule de récurrence :

$$\begin{cases} x^0 = 1 \\ \forall n \in \mathbb{N}^*, \quad x^n = x \times x^{n-1} \end{cases}$$

- 3) Écrire une fonction `puissance_2(x, n)` qui fait la même chose que `puissance_1` mais en utilisant la formule de récurrence :

$$\begin{cases} x^0 = 1 \\ \forall n \in \mathbb{N}^*, \quad x^n = \begin{cases} (x \times x)^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x \times (x \times x)^{\frac{n-1}{2}} & \text{si } n \text{ est impair} \end{cases} \end{cases}$$

- 4) Tester les instructions suivantes et expliquer ce qu'il se passe.

```
>>> puissance_1(5, 2000)
>>> puissance_2(5, 2000)
```

- 5) Décommenter puis exécuter le code suivant, qui permet de comparer les deux algorithmes itératif et récursif. Quel est le meilleur ?

```
1 from time import time
2
3 t0 = time()
4 puissance_0(5, 100000)
5 t1 = time()
6 print("Le temps d'exécution de puissance_0(5, 100000) est {}".format(t1-t0))
7
8 t0 = time()
9 puissance_2(5, 100000)
10 t1 = time()
11 print("Le temps d'exécution de puissance_2(5, 100000) est {}".format(t1-t0))
```

Exercice 5 (Fibonacci) :

- 1) Écrire une fonction `Fibonacci(n)` qui prend en argument un entier naturel `n` et qui renvoie la valeur du n^{e} terme de la suite de Fibonacci définie par $F_0 = F_1 = 1$ et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.
- 2) Tester votre fonction pour $n = 35$, puis pour $n = 40$. Que constate-t-on ? Expliquer pourquoi l'exécution devient très lente (et pourquoi il est inutile d'essayer $n = 100$ avec cette version naïve).

Exercice 6 (Coefficients binomiaux) :

- 1) Écrire une fonction `binom_1(n, p)` qui prend en argument deux entiers naturels `n` et `p` et qui renvoie la valeur du coefficient binomial $\binom{n}{p}$ en utilisant la formule de Pascal :

$$\forall n, p \in \mathbb{N}^*, \quad \binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

avec les cas de base : $\binom{n}{0} = 1$, $\binom{n}{n} = 1$, et $\binom{n}{p} = 0$ si $p > n$.

- 2) Écrire une fonction `binom_2(n, p)` qui prend en argument deux entiers naturels `n` et `p` et qui renvoie la valeur du coefficient binomial $\binom{n}{p}$ en utilisant la formule :

$$\forall n, p \in \mathbb{N}^*, \quad \binom{n}{p} = \frac{n}{p} \binom{n-1}{p-1}$$

avec le cas de base $\binom{n}{0} = 1$ (et $\binom{n}{p} = 0$ si $p > n$).

- 3) Comparer ces deux fonctions en exécutant les instructions suivantes :

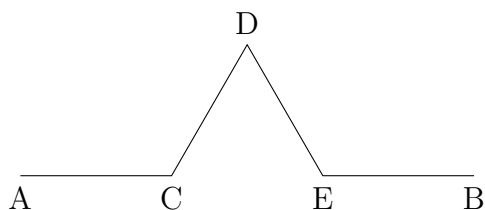
```
>>> binom_1(100, 2)
>>> binom_2(100, 2)
>>> binom_1(100, 50)      # Attention : peut être très long !
>>> binom_2(100, 50)
```

Exercice 7 (*Permutations (pour aller plus loin)*) : Écrire une fonction `permutations(L)` qui prend en argument une liste `L` et qui renvoie la liste de toutes les permutations des éléments de `L`.

Par exemple `permutations([1, 2, 3])` doit renvoyer la liste `[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]` mais pas nécessairement dans cet ordre.

Exercice 8 (*Flocon de von Koch (pour aller plus loin)*) : Pour construire le flocon de von Koch, on procède de la manière suivante :

- ★ Étape 0 : la courbe est un segment $[AB]$.
- ★ Étape 1 : on remplace le segment $[AB]$ par la figure suivante de sorte à avoir $AC = CD = DE = CE = EB$:



- ★ Étape 2 : on applique la même transformation à tous les segments $[AC]$, $[CD]$, $[DE]$ et $[EB]$.
 - ★ On recommence ainsi de suite pendant un certain nombre d'étapes.
- 1) Définir trois fonctions Python qui, à partir des coordonnées de deux points A et B du plan, calculent les coordonnées des points C , D et E de l'étape 1 respectivement.

Les coordonnées de ces points sont obtenus par les formules suivantes : si $\overrightarrow{AB}(x, y)$, alors

$$\begin{aligned}\overrightarrow{AC} & \left(\frac{x}{3}, \frac{y}{3} \right) \\ \overrightarrow{AD} & \left(\frac{1}{2}x - \frac{\sqrt{3}}{6}y, \frac{1}{2}y + \frac{\sqrt{3}}{6}x \right) \\ \overrightarrow{AE} & \left(\frac{2x}{3}, \frac{2y}{3} \right)\end{aligned}$$

On représentera les coordonnées d'un point par une liste de deux éléments.

- 2) Écrire une fonction récursive `von_koch(A, B, n)` qui prend en arguments deux points du plan A et B (liste de leurs coordonnées) et un entier n et qui renvoie la liste des coordonnées des points de la courbe de von Koch après n étapes.

Cas de base : $n == 0$ (on renvoie alors $[A, B]$).

- 3) Tracer alors la courbe obtenue. On rappelle les instructions pour tracer un graphe :

```
import matplotlib.pyplot as plt
plt.figure()
plt.title("Flocon de von Koch")
plt.plot(abscisses, ordonnees)
plt.show()
```