

# TD: programmation réursive:

## Exercice 1: compter à rebour:

1. Écrire une fonction Python réursive:
  - qui prend en entrée un entier naturel  $n$
  - qui affiche les entiers  $n, n-1, n-2, \dots, 1$ .
2. Faire de même avec une fonction itérative.

## Exercice 2: compter:

1. Écrire une fonction Python réursive:
  - qui prend en entrée un entier naturel  $n$
  - qui affiche les entiers  $1, 2, 3, \dots, n$ .
2. Faire de même avec une fonction itérative.

## Exercice 3: fonction somme: Écrire une fonction Python réursive:

- qui prend en entrée un entier naturel  $n$  non nul
- qui renvoie la somme  $1+2+3+\dots+n$

## Exercice 4: escalier:

Pour construire un escalier de  $n$  marches, il suffit de savoir construire une marche et un escalier de  $n-1$  marches ;)

Traduire ce procédé de construction en python. On utilisera une fonction réursive et le module turtle ([http://math.univ-lyon1.fr/irem/Formation\\_ISN/formation\\_recurzivite/tortue/doc/turtleDoc.html](http://math.univ-lyon1.fr/irem/Formation_ISN/formation_recurzivite/tortue/doc/turtleDoc.html))

**Exercice 5: utilisation des listes:** On s'intéresse au programme suivant:

```
0 def SommeListe(L) :  
  # cas de base  
 2   if L == [] : return 0  
  
  # cas general  
 4   return L[-1] + SommeListe(L[:-1]) # L[-1] : dernier element de liste  
 6                                     # L[:-1] : liste L tronquee  
                                     # de son dernier element
```

recursive5.py

On peut décomposer le programme ainsi sur un exemple:

```
sommeListe([1,2,3,4]) = 4 + sommeListe([1,2,3])  
                    = 4 + (3 + sommeListe([1,2]))  
                    = 4 + (3 + (2 + sommeListe([1])))  
                    = 4 + (3 + (2 + (1 + sommeListe([]))))  
                    = 4 + (3 + (2 + (1 + 0)))  
                    = 4 + (3 + (2 + 1))  
                    = 4 + (3 + 3)  
                    = 4 + 6  
                    = 10
```

Figure 1: Exemple avec la liste [1,2,3,4]

Ce programme prend donc en entrée une liste et renvoie la somme des nombres de cette liste. Écrire une nouvelle fonction `ProduitListe()` qui permet de renvoyer le produit des éléments d'une liste.

**Exercice 6: minimum d'une liste:** Écrire une fonction Python récursive:

- qui prend en entrée une liste de nombres
- qui renvoie le minimum de cette liste

**Exercice 7: triangle de Sierpinski:** Écrire une fonction Python récursive utilisant le module `turtle` permettant d'obtenir le dessin suivant:

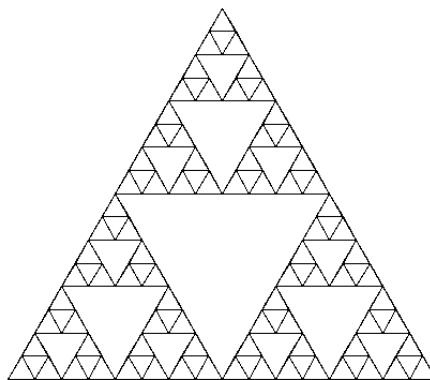


Figure 2: triangle de Sierpinski

### Exercice 8: longueur d'une chaîne de caractères:

1. Expliquer le code suivant:

```
0 ch1 = ""
  print(not ch1)
2
  ch2 = "Hello"
4  print(not ch2)
```

recursive8.py

2. Soit une chaîne de caractères, écrire un algorithme récursif permettant de déterminer sa longueur (sans utiliser la fonction `len()`).

### Exercice 9: suites:

1. Soit la suite  $u_{n+1} = 3u_n + 2$  et  $u_0 = 1$ .
  - 1.1 Calculer les 5 premiers termes de cette suite.
  - 1.2 Écrire une fonction récursive `suite(n)` permettant de calculer  $u_n$ .
2. Même question avec  $u_{n+1} = u_n^2 + n$  et  $u_0 = 1$ .

### Exercice 10: parité:

Un nombre  $N$  est pair si  $(N-1)$  est impair, et un nombre  $N$  est impair si  $(N-1)$  est pair. Écrire deux fonctions récursives mutuelles `pair(N)` et `impair(N)` permettant de savoir si un nombre  $N$  est pair et si un nombre  $N$  est impair.

### Exercice 11: palindrome:

Un mot est un palindrome si on peut le lire dans les deux sens de gauche à droite et de droite à gauche. Exemple KAYAK est un palindrome. Écrire une fonction récursive permettant de vérifier si un mot est palindrome.