# TP 8 – RÉCURSIVITÉ SUR LES ENTIERS

Info1.Algo1 - 2022-2023 Semestre Impair

## Rappels

On appelle **fonction récursive** une fonction qui s'appelle *(directement ou indirectement)* elle-même.

```
def f(n):
  if n==0: # Cas d'arret
      return 1
  else: # Cas recursif
      return 2*f(n-1) # Diminution taille du probleme
```

## Exercice 1 - fonctions définies par récurrence ★

Dans cet exercice, compléter le fichier **ex01_recurrence.py** et valider chacune des 2 fonctions de test.

**1)** Écrire la fonction récursive `factorielle` qui calcule la factorielle $n!$ d'un entier $n$ donné en paramètre. On rappelle que la factorielle est définie par :

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \quad \text{si } n \geq 1 \end{cases}$$

**2)** Écrire la fonction récursive `suite_geometrique` qui calcule le terme $u_n$ d'une suite géométrique $(u_n)$ en fonction du terme initial $u_0$, de la raison $r$ et du rang $n$ donnés en paramètres. On rappelle qu'une suite géométrique est définie par récurrence par :

$$\begin{cases} u_n = u_0 & \text{si } n = 0 \\ u_n = r \cdot u_{n-1} & \text{si } n \geq 1 \end{cases}$$

## Exercice 2 - somme des carrés ★

Dans le fichier **ex02_somme_carres.py**, compléter la fonction récursive `somme_carres` qui accepte en paramètre un entier `n` positif ou nul et retourne la somme des carrés des entiers de `0` à `n`.

**Exemple :** `somme_carres(4)` doit retourner `30` ($1^2 + 2^2 + 3^2 + 4^2$).

## Exercice 3 - chiffres d'un nombre ★

**1)** Dans le fichier **ex03_chiffres.py** compléter la fonction récursive `somme_chiffres` qui calcule et retourne la somme des chiffres d'un entier positif ou nul donné en paramètre.

**Exemple :** `somme_chiffres(45301)` doit retourner `13` (4+5+3+0+1).

**2)** Compléter la fonction récursive `premier_chiffre` qui détermine et retourne le premier chiffre d'un entier positif ou nul donné en paramètre.

**3)** Compléter la fonction récursive `occurences_chiffre` qui accepte en paramètres :

- un entier `n`
- un chiffre `c` (c'est-à-dire un entier vérifiant `0<=c<=9`)

et retourne le nombre d'occurences du chiffre `c` dans l'écriture en base `10` du nombre `n`.

## Exercice 4 - puissance rapide ★

**1)** Dans le fichier **ex04_puissance.py** compléter la fonction récursive `puissance` qui accepte en paramètres un entier relatif $a$ et un entier naturel $n$ et retourne le résultat du calcul de $a^n$. On rappelle que la puissance $a^n$ est définie par :

$$\begin{cases} a^0 = 1 \\ a^n = a \cdot a^{n-1} & \text{si } n \geq 1 \end{cases}$$

Avec l'application de cette définition, votre fonction doit valider la fonction de test pour la partie **non-optimisée**.

Vous devez alors observer une erreur de type `RecursionError`.

**2)** D'après la **documentation de Python**, l'appel de `sys.getrecursionlimit()` (du module `sys`) donne la valeur actuelle de la limite de récursion, c'est-à-dire la profondeur maximum de la pile de l'interpréteur.

Afficher la valeur de cette limite (ne pas oublier l'`import` adéquat). Interpréter l'erreur observée.

**3)** Une solution pourrait être de modifier la limite de récursion en utilisant la fonction `setrecursionlimit` du module `sys`, mais nous allons préférer ici une solution algorithmique. On remarque en effet que, si $n$ est pair, alors :

$$a^n = \left(a^{n/2}\right)^2$$

Modifier la fonction `puissance` de façon à traiter ce deuxième cas récursif (le cas récursif existant permettant alors de traiter le cas où $n$ est impair). La fonction doit alors valider la fonction de test pour la partie **optimisée**.

## Exercice 5 - minimum et maximum ★

**1) a)** Dans le fichier **ex05_minimum_maximum.py** est donnée la fonction récursive `minimum_chiffres` qui accepte en entrée un nombre entier positif ou nul `n` et retourne le minimum de ses chiffres (en base `10`). Cependant si la fonction valide les premiers tests unitaires, elle semble ne pas retourner lorsque le nombre de chiffres devient trop grand.

Expliquer la nature du problème rencontré ici (grâce à un affichage de la valeur de `n` pour chaque appel de `minimum_chiffres(n)` par exemple).

**b)** Corriger la fonction `minimum_chiffres` de manière à ce qu'elle valide tous les tests unitaires.

**2)** En vous inspirant de la version corrigée de `minimum_chiffre`, compléter la fonction `minimum_maximum_chiffres` qui accepte en entrée un nombre entier positif ou nul `n` et retourne un tuple `(mini,maxi)`, ou `mini` est le minimum des chiffres de `n` et `maxi` leur maximum.

## Exercice 6 - Syracuse ★

On considère un nombre entier plus grand strictement positif :

- s'il est pair, on le divise par 2.
- s'il est impair, on le multiplie par 3 et on ajoute 1.

En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur.

**Exemple :** à partir de 14, on construit la suite des nombres : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2... C'est ce qu'on appelle la **suite de Syracuse** du nombre 14.

Dans cet exemple, après que le nombre 1 a été atteint, la suite des valeurs (1,4,2,1,4,2...) se répète indéfiniment en un cycle de longueur 3, appelé cycle trivial.

Dans le fichier **ex06_syracuse.py**, compléter la fonction récursive `syracuse` qui retourne la suite de Syracuse d'un nombre `n` donné en paramètre, sous forme de liste, en s'arrêtant dans le cas où l'on rencontre la valeur `1`.

**Remarque :** Dans cet exercice, il n'y a pas de diminution de la taille du problème dans le cas où n est impair, mais la **conjecture de Syracuse** (appelée aussi conjecture de Collatz) dit qu'en suivant ce schéma on finit toujours par parvenir à 1.

## Exercice 7 - PGCD ★

On souhaite écrire la fonction récursive `pgcd` qui calcule et retourne le PGCD (**plus grand commun diviseur**) de deux entiers positifs `a` et `b` (avec `b` non nul).

Pour cela, on se base sur les propriétés suivantes :

- Le PGCD de `a` et `0` est `a`.
- Si `b` est non nul, le PGCD de `a` et `b` est aussi le PGCD de `b` et `a%b`.

**1)** Identifier le cas d'arrêt, le cas récursif et la taille du problème. Pourquoi la taille du problème est-elle bien diminuée lors de l'appel récursif ?

**2)** Compléter dans le fichier **ex07_pgcd.py** la fonction récursive `pgcd`.

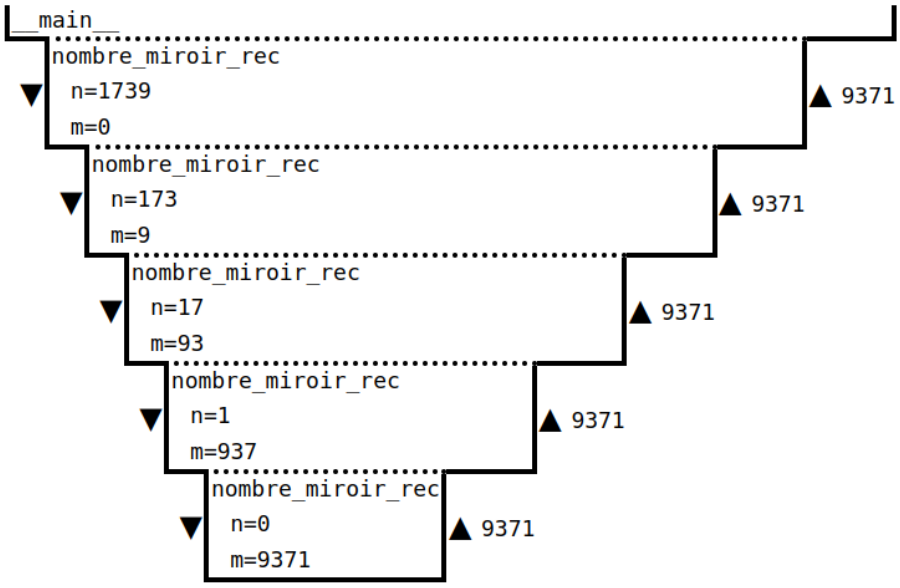
## Exercice 8 - nombre miroir ★★

Dans cet exercice, on souhaite écrire une **fonction récursive** qui accepte en paramètre un entier positif et retourne le nombre miroir correspondant à cet entier.

**Exemple :** Si l'entier donné en paramètre est `1739`, la fonction doit retourner l'entier `9371`.

**1)** Dans le fichier **ex08_nombre_miroir.py**, Compléter tout d'abord la fonction récursive `nombre_miroir_rec` qui accepte un second paramètre entier `m` égal à `0` lors de l'appel initial et qui va au fur et à mesure des appels récursifs contenir la version inversée en miroir de l'entier `n` donné en paramètre.

**Exemple :** L'appel `nombre_miroir_rec(1739,0)` génère les appels suivants :




**2)** Compléter la fonction `nombre_miroir` qui se contente d'effectuer l'appel correct de la fonction `nombre_miroir_rec`.

## Exercice 9 - récursivité mutuelle ★

Dans son ouvrage *Liber abaci* publié en 1202, Leonardo Fibonacci décrit la croissance d'une population de lapins :

*Quelqu'un a déposé un couple de lapins dans un certain lieu, clos de toutes parts, pour savoir combien de couples seraient issus de cette paire en une année, car il est dans leur nature de générer un autre couple en un seul mois, et qu'ils enfantent dans le second mois après leur naissance.*

Ce que l'on peut retraduire de la façon suivante :

- Au départ, on place un couple de lapins adultes dans un champ.
- Chaque couple de lapins adultes met un mois pour donner naissance à une paire de bébés lapins qui formeront un nouveau couple. Ceci se répète chaque mois.

- Les bébés lapins mettent un mois à devenir adultes et ne meurent jamais.

Si l'on note $a_n$ le nombre de couples d'adultes et $b_n$ le nombre de couples de bébés à l'instant $n$, on a :

$$\begin{cases} a_0 = 1 \\ a_n = a_{n-1} + b_{n-1} \quad \text{si } n \geq 1 \end{cases}$$

et

$$\begin{cases} b_0 = 0 \\ b_n = a_{n-1} \quad \text{si } n \geq 1 \end{cases}$$

Dans le fichier **ex09_lapins.py** compléter les fonctions **mutuellement récursives** `bebes` et `adultes` qui acceptent en paramètre le numéro du mois `n` et retournent respectivement le nombre de bébés et le nombre d'adultes au mois `n`.

**Indications :**

- La fonction `bebes` doit appeler la fonction `adultes`, et réciproquement, de façon à respecter les définitions par récurrence données ci-dessus.
- Ce n'est qu'**une fois que les deux fonctions sont écrites** que l'on peut les tester.

## Exercice 10 - mémoïsation ★★★ *(pour aller plus loin)*

La suite de Fibonacci $(F_n)$ (pour $n$ entier positif) est définie par récurrence de la façon suivante :

$$\begin{cases} F_n = 0 & \text{si } n = 0 \\ F_n = 1 & \text{si } n = 1 \\ F_n = F_{n-1} + F_{n-2} & \text{sinon.} \end{cases}$$

Le fichier **ex10_memoisation.py** propose une fonction `fibonacci` qui accepte en paramètre l'entier `n` et retourne le terme de la suite $(F_n)$ de rang `n`. Si la fonction est pertinente pour de petites valeurs de `n`, elle est très inefficace lorsque `n` devient trop grand.

> **Afin d'éviter d'effectuer un nombre inconsidéré de fois le même calcul, on se propose dans ce exercice de mémoriser les valeurs déjà calculées de manière à pouvoir s'en reservir. Le principe, tel qu'il va être présenté, porte le nom de "mémoïsation".**

Pour ce faire, on utilise un dictionnaire `dico` qui s'enrichit au fur et à mesure des différentes valeurs de $F_n$ :

- Ce dictionnaire est passé en paramètre lors de tous les appels récursifs.
- À un moment donné, ce dictionnaire pourra par exemple contenir les premières valeurs de de $F_n$ la façon suivante : `{0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8}`

**1)** Écrire une nouvelle fonction récursive `fibonacci_memo` qui accepte en paramètres :

- Un entier positif ou nul `n`
- Le dictionnaire `dico`

Cette fonction commence par vérifier si la valeur recherchée n'est pas déjà dans le dictionnaire. Si c'est le cas, elle retourne cette valeur. Sinon, elle implémente les même cas d'arrêt et cas récursif que la fonction `fibonacci` proposée initialement.

**Important :**

- Avant de retourner, la fonction `fibonacci_memo` ne doit pas oublier de **sauvegarder la valeur nouvellement calculée** dans le dictionnaire.
- Les appels récursifs doivent constituer en un appel de la fonction `fibonacci_memo` en passant en second paramètre le même dictionnaire que celui reçu.

**2)** Écrire une nouvelle fonction `fibonacci` qui accepte en paramètre l'entier `n` et se contente de retourner le résultat de `fibonacci_memo(n,{})`. Cette nouvelle fonction doit pouvoir valider l'ensemble des test unitaires.