# recursivite_extrait_pour_pdf

September 21, 2019

## 1 Exercice 1 : factorielle en récursif

On souhaite programmer une fonction `factorielle(n)` qui retourne la factorielle de l'entier n passé en paramètre.
Par exemple on a `factorielle(7)=7*6*5*4*3*2*1` autrement dit `factorielle(7) = 7 * factorielle(6)`.
Ce qui conduit aisément à la relation suivante :
`factorielle(n) = n * factorielle(n-1)` pour un entier `n>=1`et par convention `factorielle(0)=1`.

La programmer en récursif :

```
In [0]: def factorielle(n):
            pass
```

## 2 Exercice 2 : Des listes russes . . . aux listes syldaves

### 2.1 Listes russes

Par analogie avec une poupée russe, on considère dans cet exercice une ń liste russeż. Sans chercher à formaliser ce concept, on se donne une liste de départ qui peut :

- soit être vide
- soit contenir une autre liste qui elle-même peut :
  - soit être vide
  - soit contenir une autre liste qui elle-même peut :
    - soit être vide
    - soit contenir une autre liste qui elle-même peut :
      - etc. . .

Bien entendu on suppose que ce processus s'arrête à un moment donné. Autrement dit on suppose que la liste de départ (la *matriona*) a un nombre fini de descendantes (les *matriochkas*).

### 2.1.1 Question 1

La fonction `engendrer_liste_russe` fournie ci-dessous renvoie une liste russe qui contient un nombre aléatoire de listes emboîtées les unes dans les autres.

Écrire une fonction `compter_liste_russe` programmée en récursif qui, en prenant en argument une liste russe, permet de déterminer combien de listes sont emboîtées les unes dans les autres (dans la liste russe passée en argument).

### 2.1.2 Question 2

Pouvait-on, ou pas, la programmer raisonnablement en itératif ?

```
In [6]: import random

        def engendrer_liste_russe():
            nb = random.randint(9, 98)
            L = []
            for i in range(nb):
                L = [L]
            return L

        ma_liste_russe = engendrer_liste_russe()

        def compter_liste_russe(L):
            pass
```

## 2.2 Listes syldaves

On s'intéresse maintenant aux listes syldaves dont le concept est très proche des listes russes. Sans chercher à formaliser ce concept, on se donne une liste de départ qui peut :

- soit être vide
- soit contenir *deux* autres listes dont chacune peut, indépendamment de l'autre :
  - soit être vide
  - soit contenir *deux* autres listes dont chacune peut, indépendamment de l'autre :
    - soit être vide
    - soit contenir *deux* autres listes dont chacune peut, indépendamment de l'autre :
      - etc...

Bien entendu on suppose que ce processus s'arrête à un moment donné.

Autrement dit chaque poupée syldave peut désormais contenir 0 ou 2 enfants ...

Autrement dit on modélise ici un **arbre** généalogique (descendance d'un ancêtre) simplifié où les unions (mariages etc.) ne sont pas représentées et où chaque individu ne peut avoir que 0 ou 2 descendants ...

### 2.2.1 Question 3

La fonction `engendrer_liste_syldave` fournie ci-dessous permet de retourner une liste syldave obtenue aléatoirement.

Écrire une fonction `compter_listes_syldaves` programmée en récursif qui, en prenant en argument une liste syldave, permet de déterminer combien de listes sont, *in fine*, contenues dans la liste syldave.

### 2.2.2 Question 4

Pouvait-on, ou pas, la programmer raisonnablement en itératif ?

```
In [3]: import random

        def engendrer_liste_syldave(*ottokar):
            if len(ottokar)== 2 :
                n = ottokar[0]
                if n == 0 and ottokar[1]==0:
                    return []
                else:
                    sceptre1, sceptre2 = min(random.randint(0,3), 1), min(random.randint(0,3),
                    return [engendrer_liste_syldave(max(sceptre1*(n-1),0), max(ottokar[1]-1, 0
            else :
                return engendrer_liste_syldave(random.randint(8, 10), 2)

        ma_liste_syldave = engendrer_liste_syldave()


        def compter_liste_syldave(L):
            pass
```