Correction TD : Récursivité n°1

Exercice 1 :

Reprenons le premier exemple du cours :

Soit la suite des nombres pairs :

$$U_0 = 0$$
$$U_1 = 2$$
$$U_2 = 4$$
$$U_3 = 6$$
$$U_4 = \mathbf{8}$$

1. Donner une définition par récurrence de cette suite :

$$\begin{cases} U_0 = 0 \\ U_{n+1} = U_n + 2 \end{cases}$$

2. Créer une **fonction récursive** `nième_nombre_pair` qui prendra en paramètre un nombre `n` et qui renverra le n^{ième} nombre pair.

3. tester votre fonction avec n = 5. Pour indication le cinquième nombre pair est 8.

```
def nième_nombre_pair(n):
    if n==1:
        return 0
    else:
        return nième_nombre_pair(n-1)+2

n=int(input("Entrer n : "))
print(nième_nombre_pair(n))
```

Exercice 2 :

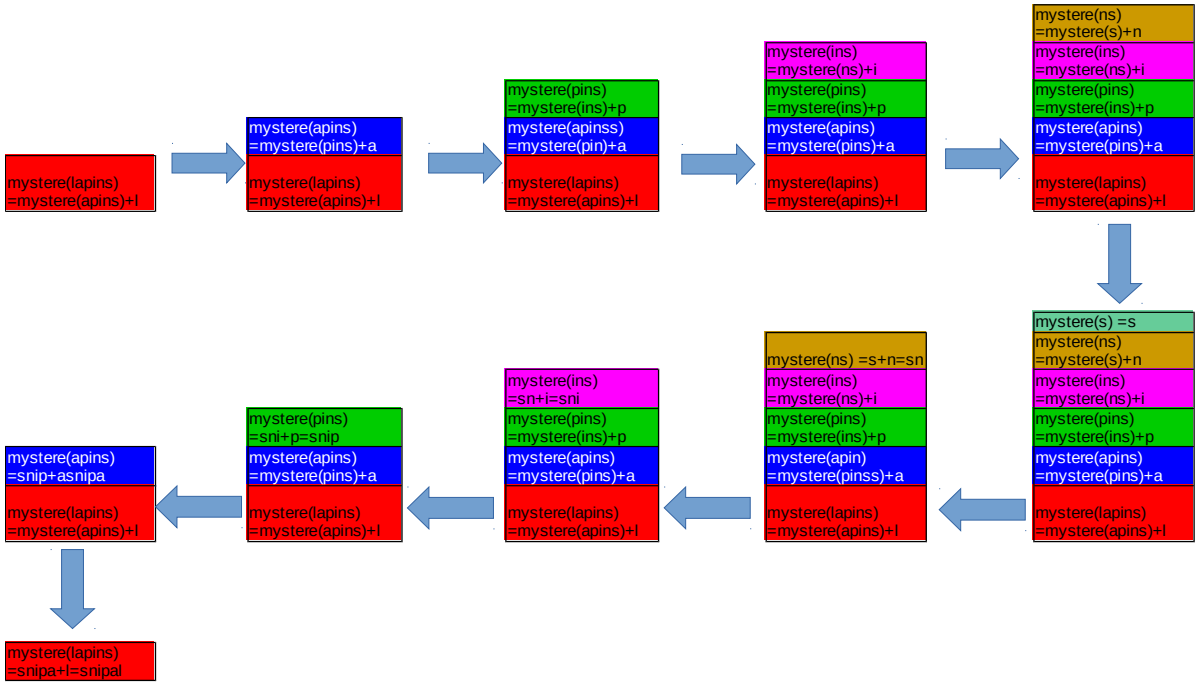
Soit la fonction suivante :

```
def mystere(l):
    if len(l)==1:
        return l[0]
    else :
        return  mystere(l[1:])+l[0]
```

1. Que cette fonction ?

Elle renverse une chaîne de caractère.

2. Compléter comme dans le cours, la pile des appels récursifs avec le mot « lapin ».



Exercice 3 :

Pour savoir si un nombre entier naturel n est un multiple de 13, on peut procéder de la façon suivante :

- Si n = 0 alors n est bien un mutiple de 13 (En effet 0 = 13 × 0)
- Sinon il suffit de procéder de la même façon avec (n – 13)

1. Utiliser ce principe pour écrire une **fonction récursive** `multiple_treize` qui prendra en paramètre un entier naturel `n` et qui renverra `True` si n est un multiple de 13 et `False` sinon.

2. tester votre fonction avec les nombre 481 et 587.

```
def multiple_treize(n):
    if n==0:
        return True
    if n<0:
        return False
    return multiple_treize(n-13)

print(multiple_treize(481))
print(multiple_treize(587))
```

Exercice 4 :

On appelle S_n la somme des nombre entiers de 1 à n.

$$S_n = 1 + 2 + 3 + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + n$$

On peut remarquer que :

$$S_n = S_{n-1} + n$$

1. En utilisant cette remarque, écrire une **fonction récursive** `somme_recursive` qui prendra en paramètre un entier naturel `n` et qui renverra le résultat de la somme des entiers de 1 à n.

2. Tester votre fonction avec n = 100. (Pour vérification, je rappelle que $S_n = \frac{n(n+1)}{2}$)

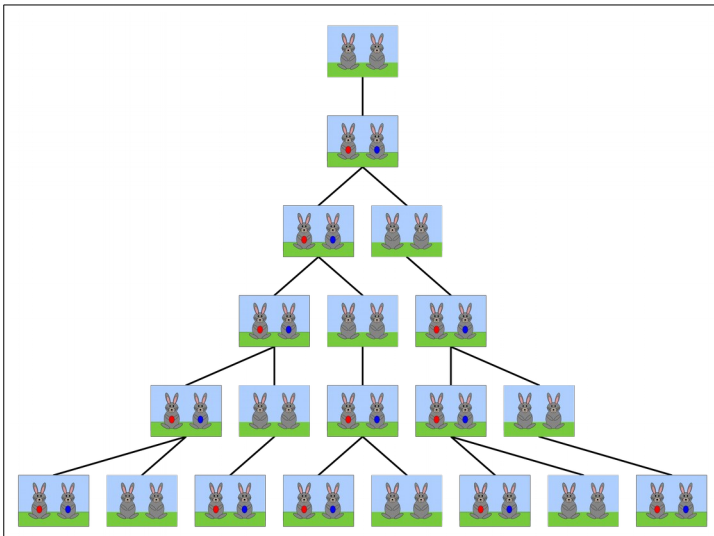
```
def somme_recursive(n):
    if n==0 :
        return 0
    return somme_recursive(n-1)+n

print(somme_recursive(100))
```

Exercice 5 :

Leonardo Pisano Fibonacci (v. 1175 – v. 1250) est le plus connu des mathématiciens du Moyen Âge. Il est surtout connu par la suite de nombres qui porte son nom.

Elle aurait été découverte en comptabilisant les lapins suite à leur reproduction. Fibonacci met au point la formule qui permet de déduire la quantité de lapins de la saison suivante à partir des quantités des saisons précédentes.

Cette formule devient la première formule avec récursivité connue de l'histoire.



Étape 1 : un couple de lapereaux

Étape 2 : un couple de lapins adultes dont la femelle est donnera naissance à un couple de lapereaux le mois suivant

Étape 3 : Le couple de départ (enceinte à nouveau) + le couple de lapereaux

Étape 4 : Le couple de départ (enceinte à nouveau) + un second couple (enceinte) + un nouveau couple de lapereaux

……

……..

1. Soit la suite $(F_n)_{n\geq 1}$ qui donne le nombre de couple de lapins à chaque étape. Compléter la définition par récurrence (ci dessous) de cette suite :

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

2. Écrire une **fonction récursive** Fibo_recursif qui prendra en paramètre un entier naturel `n` et qui renverra le $n^{ième}$ terme de la suite de Fibonacci.

```
def fibo_recursif(n):
    if n==1 or n==2:
        return 1
    return fibo_recursif(n-1)+fibo_recursif(n-2)

print(fibo_recursif(6))
```

Exercice 6 :

La méthode du paysan russe est un très vieil algorithme de multiplication de deux nombres entiers positifs.

Cette méthode fut utiliser par les premier ordinateur avant que la multiplication ne soit directement intégrée dans les processeur sous forme de circuit électronique.

En voici une version itérative en langage Python :

```
def multiplication_russe(x,y):
    p=0
    while x>0:
        if x%2==1:
            p=p+y
        x=x//2
        y=y+y
    return p
```

1. Appliquer à la main cette fonction pour effectuer la multiplication de 105 par 253 puis vérifier avec l'ordinateur que le résultat sue vous avez trouvé est le bon.

x	y	p
105	253	**0**
52	**506**	**253**
26	**1012**	**253**
13	**2024**	**253**
6	**4048**	**2277**
3	**8096**	**2277**
1	**16192**	**10373**
0	**32384**	**26565**

2. On admet que cet algorithme repose sur les relations suivantes :

$$x \times y = \begin{cases} 0 & si \quad x = 0 \\ (x\ //\ 2) \times (y + y) & si \quad x\,est\ pair \\ (x//2) \times (y + y) + y & si \quad x\,est\,impair \end{cases}$$

En utilisant cette remarque, écrire **une fonction récursive** `multpilication_russe_recursive` qui prendra en paramètre deux entier naturel `x` et `y` et qui renverra le résultat de la multiplication de x par y.

```
def multiplication_russe_recursive(x,y):
    if x==0:
        return 0
    if x%2==0:
        return multiplication_russe_recursive(x//2,y+y)
    else:
        return multiplication_russe_recursive(x//2,y+y)+y

print(multiplication_russe_recursive(105,253))
```

Exercice 7 : *(bonus : le triangle de pascal)*

Le triangle de Pascal donne les coefficient binomiaux $\binom{n}{p}$ sous la forme d'un triangle (se dit « p parmi n »).

p → n (nb épreuves) ↓	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

1. En observant le triangle, on peut écrire :

$\binom{n}{p} = 1$ si p = 0 ou si n = p

$\binom{n}{p} = \binom{n-1}{p} + \binom{n-1}{p-1}$ sinon

2. Écrire une fonction récursive `Coef_bin` qui prend en paramètre deux nombres entiers naturels `n` et `p` (tels que p ≤ n) et qui renvoie le coefficient binomial « p parmi n ».

3. A l'aide d'une boucle for, faire afficher le triangle de Pascal (pour n allant de 0 à 10).

```
def coef_bin(n,p):
    if p==0 or n==p:
        return 1
    return coef_bin(n-1,p)+coef_bin(n-1,p-1)

for i in range(10):
    L=[]
    for j in range(i+1):
        L.append(coef_bin(i,j))
    print(L)
```