Récursivité

Informatique #02

Partie A – Pour commencer...

Exercice 1 — factorielle

1. Écrire une fonction récursive `fact(n)` qui renvoie la factorielle d'un entier `n` donné.
2. Écrire une fonction `fact2(n)` similaire mais implémentée à l'aide d'une boucle.
3. Construire un tableau aléatoire de 100 entiers compris entre 1 et 5000 puis calculer les factorielles correspondantes à l'aide des fonctions précédentes. Comparer les temps de calcul.

```
In [1]: def fact(n):
            if n <= 1:
                return 1
            return n*fact(n-1)
In [2]: def fact2(n):
            res = 1
            for i in range(1,n+1):
                res *= i
            return res
In [3]: fact(5), fact2(6)
Out[3]: (120, 720)
In [4]: import random as rd
        import time, sys
        sys.setrecursionlimit(100000)
        T = [rd.randint(0, 5000) for i in range(100)]
In [5]: t0 = time.time()
        res1 = [fact(t) for t in T]
        t1 = time.time()
        res2 = [fact2(t) for t in T]
        t2 = time.time()
        print(t1-t0,t2-t1)
Out[5]: 0.38621020317077637 0.2839620113372803
```

Exercice 2 — puissance

1. Écrire une fonction récursive `puis(x, n)` qui retourne x^n pour x réel et n entier naturel.
2. Écrire une fonction analogue itérative.

```
In [7]: def puis(x, n):
            if n == 0:
                return 1
            return x * puis(x, n-1)
In [8]: puis(3,4)
Out[8]: 81
```

Exercice 3 — somme des entiers

Écrire une fonction récursive `somme_ent(n)` qui calcule la somme des entiers compris entre 0 et n.

```
In [9]: def somme_ent(n):
            if n == 0:
                return 0
            return n + somme_ent(n-1)
In [10]: somme_ent(10)
Out[10]: 55
```

Exercice 4 — triangle de Pascal

1. Écrire une fonction `binom(n, k)` qui calcule $\binom{n}{k}$ à l'aide de la formule :

$$\forall (k,n) \in \mathbb{N}^* \times \mathbb{N}^* \quad \binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

2. En déduire une fonction `trianglepascal(n)` qui affiche les $n+1$ premières lignes du triangle de Pascal.
3. Essayer d'optimiser la fonction précédente.

```
In [11]: def binom(n, k):
             if k == 0:
                 return 1
             elif n == 0:
                 return 0
             return binom(n-1, k-1) + binom(n-1, k)
In [12]: binom(5, 3), fact(5)//(fact(3)*fact(2))
Out[12]: (10, 10)
In [14]: def trianglepascal(n):
             for i in range(n+1):
                 l = []
                 for j in range(i+1):
                     l += [binom(i,j)]
                 print(i,l)
In [15]: trianglepascal(7)
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]
```

Exercice 5 — somme des éléments d'une liste

Écrire une fonction `sommelist(L)` qui calcule récursivement la somme des éléments d'une liste L.

```
In [16]: def sommelist(L):
             if len(L) == 0:
                 return 0
             return L[0]+sommelist(L[1:])
In [17]: sommelist([1, 2, 3, 4, 5, 1])
Out[17]: 16
In [18]: def sommelist_rec_bis(L, k, s):
             if k == 0:
                 return s
             return sommelist_rec_bis(L, k-1,s+L[k-1])
         def sommelist_bis(L):
             return sommelist_rec_bis(L, len(L), 0)
In [19]: sommelist_bis([1, 2, 3, 4, 5, 1])
Out[19]: 16
```

Partie B – Autour de la suite de Fibonacci

Exercice 6 — Fibonacci récursive v1

On rappelle que la suite de Fibonacci vérifie la relation de récurrence suivante :

$$F_0 = 0, \quad F_1 = 1, \quad \forall n \in \mathbb{N} \quad F_{n+2} = F_{n+1} + F_n$$

1. Écrire une fonction récursive `fibo_rec1(n)` qui retourne la valeur de F_n.
2. Quelle est sa complexité ?

```
In [20]: def fibo_rec1(n):
             if n == 0:
                 return 0
             elif n == 1:
                 return 1
             else:
                 return fibo_rec1(n-1)+fibo_rec1(n-2)
In [21]: fibo_rec1(7)
Out[21]: 13
```

Exercice 7 — Fibonacci itérative

1. Écrire une fonction itérative `fibo_it(n)` qui retourne la valeur de F_n.
2. Quelle est sa complexité ?

```
In [22]: def fibo_it(n):
             a, b = 1, 0
             for i in range(n):
                 a, b = a+b, a
             return b
In [23]: fibo_it(7)
Out[23]: 13
```

Exercice 8 — Fibonacci récursive v2 (récursivité terminale)

1. Écrire une fonction récursive `fibo_tempo(a, b, k)` qui prend en argument F_n et F_{n-1} puis qui retourne F_{n+k}.
2. En déduire une nouvelle fonction récursive `fibo_rec2(n)` qui renvoie F_n.
3. Quelle est sa complexité ?

```
In [24]: def fibo_tempo(a, b, k):
             if k == 0:
                 return b
             return fibo_tempo(a+b, a, k-1)

         def fibo_rec2(n):
             return fibo_tempo(1, 0, n)

In [25]: fibo_rec2(7)

Out[25]: 13
```

Exercice 9 — Fibonnaci récursive v3 (mémoïsation)

Écrire une fonction récursive `fibo_rec(n)` qui renvoie F_n en sauvagardant dans une liste ou bien un dictionnaire les termes de la suite déjà calculés.

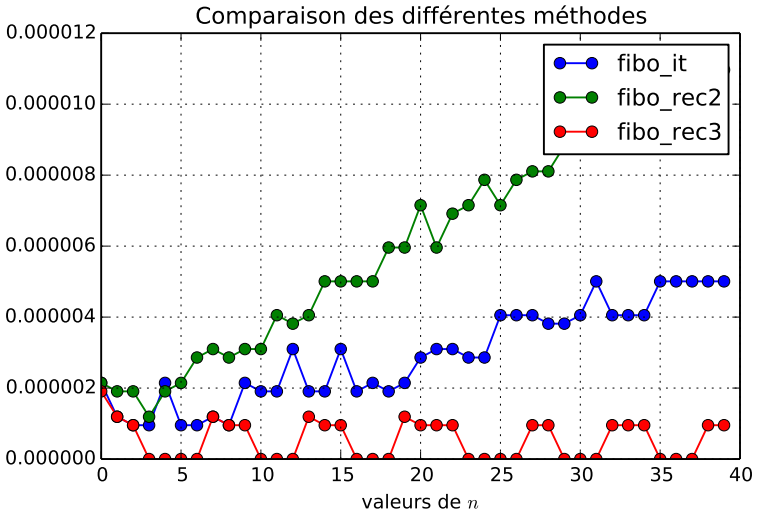
```
In [26]: memo = {0:0, 1:1}
         def fibo_rec3(n):
             if not n in memo:
                 memo[n] = fibo_rec3(n-1) + fibo_rec2(n-2)
             return memo[n]

In [27]: fibo_rec3(7)

Out[27]: 13
```

Exercice 10 — comparaison des différentes méthodes (🌶)

1. Comparer les temps de calcul dans les 4 cas. On pourra prendre $n = 34$.
2. Comparer graphiquement les temps de calcul des 4 méthodes.

```
In [28]: import time
```

```
In [29]: n = 34
         t0 = time.time()
         fibo_rec1(n)
         t1 = time.time()
         fibo_rec2(n)
         t2 = time.time()
         fibo_rec3(n)
         t3 = time.time()
         fibo_it(n)
         t4 = time.time()
         print([t1-t0, t2-t1, t3-t2, t4-t3])

[4.62979722023, 0.00012278556823, 0.000267982482910, 0.0001029968261]

In [30]: import numpy as np
         import matplotlib.pyplot as plt
         import time

In [31]: def comparaison_f(n, fibo):
             x = np.arange(n)
             y = np.zeros(n)
             for i in range(n):
                 t0 = time.time()
                 fibo(i)
                 t1 = time.time()
                 y[i] = t1 - t0
             plt.plot(x, y, marker = 'o')

         def comparaison(n):
             memo = {0:0, 1:1} # réinitialisation du dictionnaire
             comparaison_f(n, fibo_it)
             comparaison_f(n, fibo_rec2)
             comparaison_f(n, fibo_rec3)
             comparaison_f(n, fibo_rec1)
             plt.title('Comparaison des différentes méthodes')
             plt.legend(['fibo_it', 'fibo_rec2',
                                    'fibo_rec3','fibo_rec1'])
             plt.xlabel('valeurs de $n$')
             plt.ylabel('temps (en s)')
             plt.grid(True)
             plt.savefig('Fibonacci.pdf')
             plt.show()

In [33]: comparaison(40)
```



Partie C – Arithmétique

Exercice 11 — algorithme d'Euclide

Écrire une fonction `euclide(a, b)` permettant de calculer le pgcd de deux entiers de façon récursive.

```
In [44]: def euclide(a, b):
             if b == 0:
                 return a
             return euclide(b, a % b)
In [45]: euclide(29029, 6090)
Out[45]: 203
```

Exercice 12 — conversion d'entiers (☕)

1. Écrire une fonction `listechiffres(n)` d'un entier `n` qui retourne la liste des chiffres de cet entier (en base 10) sous forme de liste.
2. Écrire deux fonctions `codbinaire_it(n)` et `codbinaire_rec(n)` qui retournent une chaîne de caractères représentant l'entier `n` converti en base 2 en procédant soit de façon itérative, soit de façon récursive.
3. Écrire enfin une fonction `codage(n, base)` permettant de convertir tout entier `n` dans une base donnée, en supposant que la base est inférieure ou égale à 16.
   Écrire une fonction `euclide(a, b)` permettant de calculer le pgcd de deux entiers de façon récursive.

```
In [46]: def listechiffres(n):
             L = []
             while n != 0:
                 L = [n%10] + L
                 n = n // 10
             return L
In [47]: listechiffres(789)
Out[47]: [7, 8, 9]
In [48]: def codbinaire(n):
             L = ''
             while n != 0:
                 L = str(n % 2) + L
                 n = n // 2
             return L
In [49]: codbinaire(235)
Out[49]: '11101011'
In [50]: def codbinaire_rec(n):
             if n < 2:
                 return str(n)
             return codbinaire_rec(n // 2) + str(n % 2)
In [51]: codbinaire_rec(235)
Out[51]: '11101011'
In [52]: def codage(n, base):
             conv = '0123456789ABCDEF'
             if n < base:
                 return conv[n]
             return codage(n // base, base) + conv[n % base]
In [53]: codage(23514, 2)
Out[53]: '101101111011010'
In [54]: codage(23514, 16)
Out[54]: '5BDA'
```

Exercice 13 — exponentiation rapide

1. Écrire une fonction permettant, à l'aide de l'algorithme d'exponentiation rapide vu en cours, de calculer x^n pour un réel x et un entier n donnés, en n'utilisant que des multiplications.
2. Comparer le temps d'exécution avec l'algorithme naïf.

```
In [55]: def exprapide(x, n):
             if n == 1:
                 return x
             elif n % 2 == 0:
                 return exprapide(x*x, n //2)
             else:
                 return x*exprapide(x*x, (n-1) //2)
In [56]: exprapide(2, 3)
Out[56]: 8
In [57]: def expnormal(x, n):
             if n == 1:
                 return x
             else:
                 return x*expnormal(x, n-1)
In [58]: import time
         import sys
         sys.setrecursionlimit(30000)
         t0 = time.time()
         expnormal(1.004, 20000)
         t1 = time.time()
         exprapide(1.004, 20000)
         t2 = time.time()
         print(t1-t0, t2-t1)
0.02352118492126465 0.00012087821960449219
```

Partie D – Suites récurrentes

Exercice 14 — conjecture de Syracuse

On considère la suite de Syracuse définie par $u_0 = c$ et :

$$\forall n \in \mathbb{N} \quad u_{n+1} = \begin{cases} \dfrac{u_n}{2} \text{ si } u_n \text{ est pair} \\ 3u_n + 1 \text{ sinon} \end{cases}$$

On conjecture, mais ce résultat n'a jamais été démontré, que quel que soit c, la suite $(u_n)_{n\in\mathbb{N}}$ atteint la valeur 1, c'est-à-dire qu'il existe $N \in \mathbb{N}$ tel que $u_N = 1$.

1. Écrire une fonction récursive `syracuse(c)` qui retourne les différentes valeurs prises par la suite jusqu'à qu'elle atteigne 1.
2. Écrire une fonction analagogue mais cette fois-ci de manière itérative.

```
In [34]: def syracuse(n):
             if n == 1:
                 return 1
             elif n % 2 == 0:
                 return syracuse(n//2)
             else:
                 return syracuse(3*n+1)
In [35]: def syracuse_it(n):
             L = []
             s = n
             while s != 1:
                 L += [s]
                 if s % 2 == 0:
                     s = s // 2
                 else:
                     s = 3*s+1
             return L+[1]
In [36]: syracuse(38)
Out[36]: 1
In [37]: print(syracuse_it(38))
[38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52,
                      26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

Exercice 15 — approximation de $\sqrt{2}$

– Méthode de Héron

```
In [37]: def approx1(n):
             if n == 0:
                 return 5
             return approx1(n-1)/2+1/approx1(n-1)
In [38]: print(approx1(5), math.sqrt(2))
1.4142135857968836 1.4142135623730951
```

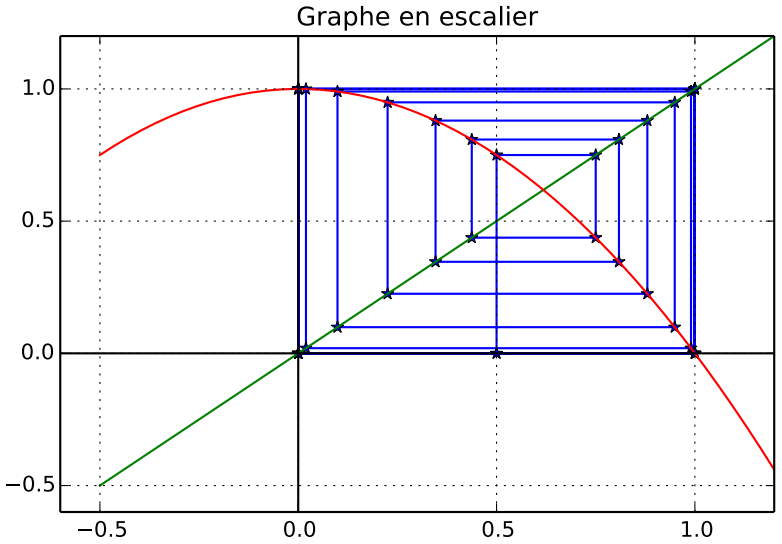
– Méthode Théon de Smyrne

```
In [39]: def p(n):
             if n == 0:
                 return 1
             return p(n-1)+2*q(n-1)

         def q(n):
             if n == 0:
                 return 1
             return p(n-1)+q(n-1)

         def approx2(n):
             return p(n)/q(n)

In [40]: print(approx2(5), math.sqrt(2))

1.4142857142857144 1.4142135623730951
```

Exercice 16 — suites récurrentes et graphes en escalier (🔥)

Écrire une fonction `escargot(f, n, u0, xmin, xmax)` permettant de visualiser le graphe en colimaçons engendré par une suite récurrente de la forme $u_{n+1} = f(u_n)$, les arguments `xmin` et `xmax` permettant de régler la fenêtre d'affichage.

```
In [41]: import numpy as np
         import matplotlib.pyplot as plt

         def escargot(f, n, u0, xmin, xmax):
             abs = [u0]
             ord = [0]
             x = u0
             for i in range(n):
                 abs += [x, f(x)]  # on pourrait éviter de calculer
                                   plusieurs fois f(x)
                 ord += [f(x), f(x)]
                 x = f(x)
             plt.plot(abs, ord, marker = '*')

             t = np.linspace(xmin, xmax)
             plt.plot(t, t) # 1ère bissectrice
             plt.plot(t, f(t)) # courbe fonction
             plt.axhline(color='k')
             plt.axvline(color='k')
             plt.title('Graphe en escalier')
             plt.grid(True)
             plt.savefig('suite.pdf')
             plt.show()

In [42]: escargot(lambda x: np.sqrt(1+x), 10, 4, 0, 4.5)

In [43]: escargot(lambda x: 1-x**2, 20, 1/2, -0.5, 1.2)
```



Partie E – Miscellaneous

Exercice 17 — palindrome

1. Écrire une fonction récursive `inverse(mot)` qui renverse la chaîne de caractères `mot`.
2. En déduire une fonction `palindrome(mot)` qui détecte la présence d'un palindrome.

```
In [59]: def inverse(mot):
             if mot == '':
                 return mot
             return inverse(mot[1:]) + mot[0]
In [60]: M = 'anticonstitutionnellemment'
In [61]: inverse(M)
Out[61]: 'tnemmellennoitutitsnocitna'
In [62]: def palindrome(mot):
             return mot == inverse(mot)
In [63]: palindrome('kayak')
Out[63]: True
```

Exercice 18 — liste d'anagrammes (☕)

1. Écrire une fonction récursive `anagramme(mot)` qui retourne la liste des anagrammes d'un mot donné.
2. Améliorer la fonction précédente pour éliminer les éventuels doublons.

```
In [64]: def anagramme(mot):
             if len(mot) == 1:
                 return [mot]
             else:
                 res = []
                 for w in anagramme(mot[1:]):
                     for pos in range(len(w)+1):
                         res += [w[:pos]+mot[0]+w[pos:]]
             return res
In [65]: print(anagramme('moto'))
['moto', 'omto', 'otmo', 'otom', 'mtoo', 'tmoo', 'tomo', 'toom',
'mtoo', 'tmoo', 'tomo', 'toom', 'moot', 'omot', 'oomt', 'ootm',
'moot', 'omot', 'oomt', 'ootm', 'moto', 'omto', 'otmo', 'otom']
In [66]: print(list(set(anagramme('moto'))))
['moot', 'omot', 'omto', 'tmoo', 'otmo', 'oomt',
'toom', 'otom', 'ootm', 'tomo', 'mtoo', 'moto']
```