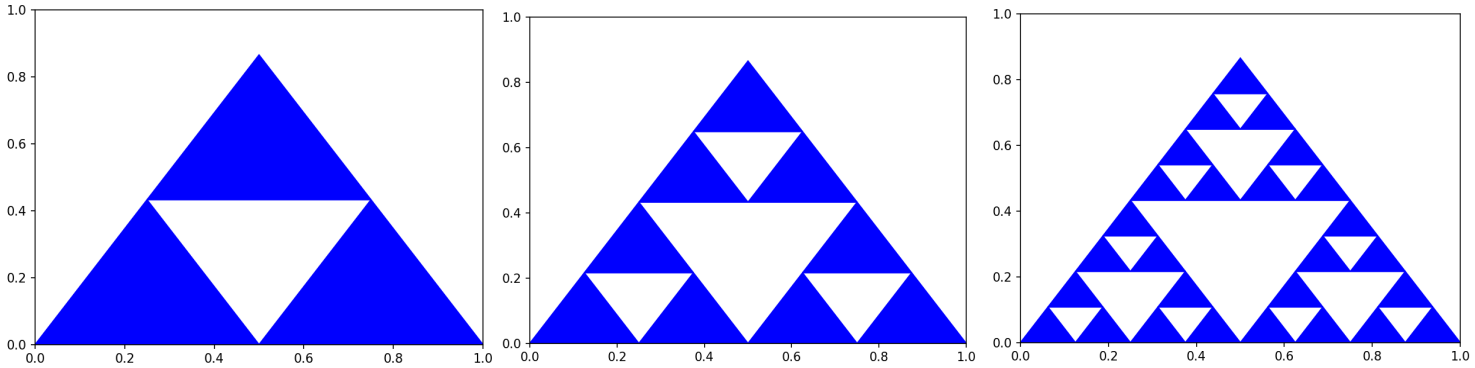
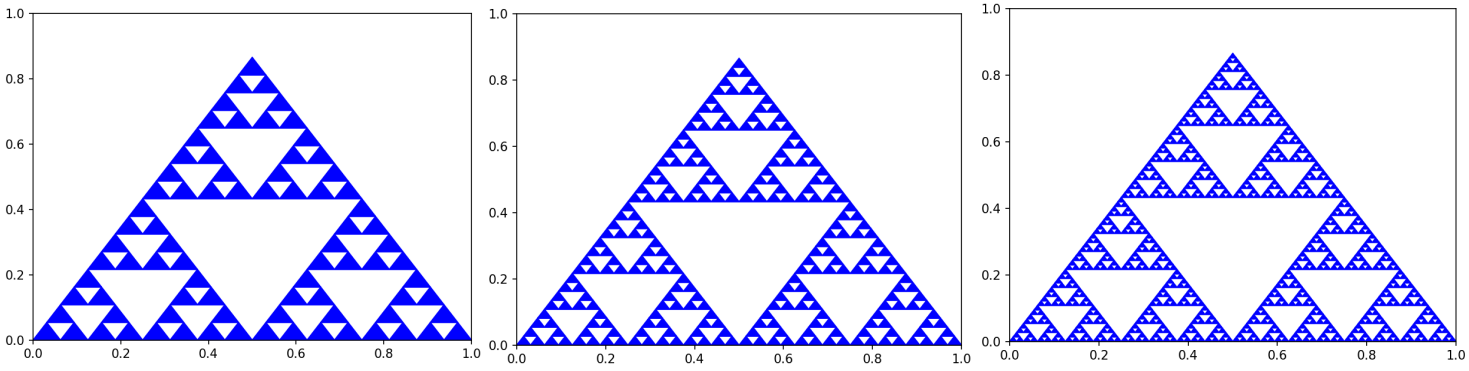
Informatique - Exercices d'entraînement 2



— 100% Récursivité —



L'objectif de cette seconde série d'exercices est de compiler quelques énoncés d'entraînement à la programmation récursive.

Vous trouverez dans ce document :

Amusez-vous bien, et comme il est écrit à l'ouverture de Pyzo : 'Happy coding !'

0 - Préambule - Test de validité

Si vous ne pouvez pas battre votre ordinateur aux échecs, défiez-le au kick-boxing.

Une fois que l'on a codé une fonction, il est indispensable de vérifier qu'elle produit le résultat attendu. Pour se rassurer, plusieurs vérifications sont recommandées. Mais il peut être laborieux de faire ces vérifications une par une, et on peut grâce aux listes faire n vérifications en une seule instruction, avec n arbitraire.

Un petit exemple vaut mieux qu'un long discours. Le code ci-dessous est celui d'une fonction qui reçoit comme paramètre un entier n, et qui retourne la valeur de n^2+1.

```
def f(n):
        return n**2 +1
```

Pour la tester, on peut produire en une seule instruction l'affichage de $f(0)$,..., $f(9)$ en écrivant dans le shell :

```
>>>[f(k) for k in range(10) ]
```

Instruction qui produira l'affichage espéré :

```
[1, 2, 5, 10, 17, 26, 37, 50, 65, 82]
```

Dans un certain nombre d'exercices de ce document, des tests de validité seront proposés ; mais libre à vous de les modifier et de choisir ceux que vous préférerez.

1 - Exemple fondamental - Factorielle d'un entier

L'informatique n'est pas une science exacte ; on n'est jamais à l'abri d'un succès.

En Maths, la factorielle d'un entier naturel n peut être définie de deux manières.
La première consiste à poser :

$$\begin{cases} 0! = 1 \\ \forall\, n \in \mathbb{N}^*, \quad n! = \displaystyle\prod_{k=1}^{n} k \end{cases}$$

Méthode 1

Et la seconde consiste à poser :

$$\begin{cases} 0! = 1 \\ \forall\, n \in \mathbb{N}^*, \quad n! = n \times (n-1)! \end{cases}$$

Méthode 2

La différence entre ces 2 méthodes n'est pas la valeur de $n!$ obtenue (heureusement), mais le mode de calcul de cette valeur.

De nouveau, un petit exemple vaut mieux qu'un long discours :

➤ **Calcul de $5!$ avec la méthode 1**

$$5! = 1 \times 2 \times 3 \times 4 \times 5 = 2 \times 3 \times 4 \times 5 = 6 \times 4 \times 5 = 24 \times 5 = 120$$

➤ **Calcul de $5!$ avec la méthode 2**

$$5! = 5 \times 4! = 5 \times 4 \times 3! = 5 \times 4 \times 3 \times 2! = 5 \times 4 \times 3 \times 2 \times 1! = 5 \times 4 \times 3 \times 2 \times 1 \times 0! = 5 \times 4 \times 3 \times 2 \times 1 \times 1$$

D'une certaine manière, les deux méthodes font les même calculs, mais pas dans le même sens.

En tous les cas, ces deux méthodes seront codées différemment en Python :

Méthode 1 (itérative)

```
def FAC1(n):
    res =1
    for k in range(1,n+1):
        res = res * k
    return res
```

Méthode 2 (récursive)

```
def FAC2(n):
    if n ==0:
        return 1
    else:
        return n * FAC2(n-1)
```

La fonction de gauche est appelée une fonction **itérative**. Celle de droite est appelée une fonction **récursive** ; sa particularité est qu'elle "s'appelle elle-même", jusqu'à ce que la condition d'arrêt écrite en-dessous du "def" soit vérifiée.

L'objectif de ce document est de fournir quelques exemples illustrant la récursivité.

Dans TOUS les exercices qui suivent, la consigne est la même : coder une fonction RECURSIVE répondant au problème.

2 - Des exercices pour démarrer

"Pour comprendre la récursivité,
vous devez d'abord comprendre la récursivité."

Exercice 1. — Ecrire le code d'une fonction $F(n)$ qui affiche n fois "Je dois ranger mon bureau".

Exercice 2. — Ecrire le code d'une fonction Somme(n), qui reçoit comme paramètre un entier $n \geqslant 1$, et qui retourne la valeur de :

$$S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Test de validité (dans le shell) : `[Somme(k) for k in range(1,5)]`

Exercice 3. — **Valeur approchée de $\zeta(2)$.**

Ecrire le code d'une fonction Z2(n), qui reçoit comme paramètre un entier $n \geqslant 1$, et qui retourne la valeur de :

$$S_n = 1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2}$$

Test de validité (dans le shell) : `[Z2(k) for k in range(1,5)]`

Exercice 4. — **Simulation de tirages de lettres.**

Ecrire le code d'une fonction SIM(n), qui reçoit comme paramètre un entier n, et qui affiche n lettres minuscules choisies aléatoirement.

Remarque. On pourra introduire la chaîne de caractères :

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
```

et utiliser la célèbre fonction randint.

Test de validité (dans le shell) : `SIM(50)`

3 - Incontournables du programme - Sur les listes

Dans le monde, il y a 10 catégories de personnes : celles qui connaissent le binaire et celles qui ne le connaissent pas.

Exercice 5. — Minimum d'une liste

Ecrire une fonction `MinR(L)` de paramètre une liste non-vide d'entiers, qui retourne le plus petit élément de cette liste.

Test de validité (dans le shell) : `L=[randint(0,10) for k in range(12)]`(création d'une liste aléatoire d'entiers), puis `L`(affichage de la liste), puis `MinR(L)`(affichage du min).

Exercice 6. — Maximum d'une liste

Ecrire une fonction `MaxR(L)` de paramètre une liste non-vide d'entiers, qui retourne le plus grand élément de cette liste.

Exercice 7. — Moyenne d'une liste

Ecrire une fonction `MoyR(L)` de paramètre une liste non-vide d'entiers, qui retourne la moyenne des éléments de cette liste.

Indication : on commencera par écrire une fonction `SomR(L)` qui calcule récursivement * la somme des éléments de la liste L.

Exercice 8. — Ecrire le code Python d'une fonction NB4R(L) qui reçoit comme paramètre une liste d'entiers L, et qui renvoie le nombre d'apparitions de l'entier 4 dans la liste L.

Par exemple l'instruction NB4R([1,2,4,5,7,4,12]) doit produire le résultat : 2

Exercice 9. — Ecrire le code Python d'une fonction NBR(L,n) qui reçoit comme paramètres une liste d'entiers L et un entier n, et qui renvoie le nombre d'apparitions de l'entier n dans la liste L.

Par exemple l'instruction NBR([1,2,4,2,7,4,2],2) doit produire le résultat : 3.

Exercice 10. — Ecrire le code Python d'une fonction Pairs(L) qui reçoit comme paramètre une liste d'entiers L, et qui renvoie le nombre d'entiers pairs de L.

Par exemple l'instruction PairsR([1,2,4,5,7,11,12]) doit produire le résultat : 3

∗. Est-ce la peine de le préciser ?

Exercice 11. — Dichotomie pour l'appartenance à une liste

Il s'agit de l'exercice 2 du TP14 (le premier sur la récursivité).

On peut déterminer si un élément x est présent dans une liste L à l'aide d'un algorithme dichotomique, *si l'on suppose que L est triée dans l'ordre croissant.*

Pour ce faire, si L contient au moins un élément, on la coupe en deux en calculant l'indice m de l'élément situé au milieu de la liste. Puis on compare $L[m]$ et l'élément x :

- s'ils sont égaux, on a trouvé x ;
- si $x < L[m]$, comme L est triée dans l'ordre croissant, si x se trouve dans L, il se situe forcément dans la portion à strictement à gauche de $L[m]$
- sinon, si $x > L[m]$, alors si x se trouve dans L, il se situe dans la portion située strictement à droite de $L[m]$

Ecrire une fonction `DIR(x,L)` qui implémente l'algorithme de recherche dichotomique détaillé ci-dessus. On pourra utiliser des slicings judicieusement choisis.

Exercice 12. — Une perle rare - L'unique fonction itérative de ce document

Ecrire une fonction **itérative** separe(L) qui prend en argument une liste L supposée non vide, et qui renvoie deux listes Lg et Ld telle que Lg contient les éléments de L qui sont inférieurs ou égaux à $L[0]$ (à l'exception du premier élément), et Ld ceux qui sont strictement supérieurs à $L[0]$.

Ainsi, separe([6, 7, 1, 6, 9, 2]) doit renvoyer ([1, 6, 2], [7, 9]).

On remarquera en particulier que seul le deuxième 6 figure dans la liste.

Exercice 13. — Algorithme de tri rapide récursif

Ecrire une fonction récursive TRAP(L) qui reçoit comme paramètres une liste d'entiers, et qui renvoie la liste L triée dans l'ordre croissant, en s'inspirant de la description de cet algorithme donnée dans l'énoncé du TP14.

4 - Incontournables du programme - Sur les suites

Ce n'est pas un hasard si l'informatique et la théorie du chaos se sont développées simultanément.

Exercice 14. — Ecrire le code Python d'une fonction SCHIFFR(n) qui reçoit comme paramètre un entier n, et qui renvoie la somme des chiffres de n dans son écriture décimale.

Par exemple, SCHIFFR(123) doit renvoyer 6.

Exercice 15. — **SRL1** Soit (u_n) la suite réelle définie par $u_0 = 2$ et :

$$\forall\, n \in \mathbb{N},\ u_{n+1} = \frac{2u_n + 1}{3}$$

Ecrire une fonction F12(N) en Python qui reçoit comme paramètre un entier naturel N, et qui retourne la valeur de u_N.

Exercice 16. — **SRL2** Soit (u_n) la suite réelle définie par $u_0 = 1$, $u_1 = 3$ et :

$$\forall\, n \in \mathbb{N},\ u_{n+2} = u_{n+1} - u_n + 1$$

Ecrire une fonction F13(N) en Python qui reçoit comme paramètre un entier naturel N, et qui retourne la valeur de u_N.

Exercice 17. — **Suite de Fibonacci** La suite de Fibonacci est la suite (F_n) définie en posant :

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

Ecrire une fonction en Python `FiboR(p)` qui prend en paramètre un entier naturel p, et qui renvoie le terme F_p de la suite de Fibonacci.

Test de validité (dans le shell) : `[FiboR(plouf) for plouf in range(8)]`

Exercice 18. — **Coefficients binomiaux**

Ecrire une fonction en Python `BINR(n,p)` qui reçoit comme paramètres deux entiers naturels n et p, et qui renvoie le coefficient binomial $\binom{n}{p}$.

On rappelle que :

— Si $p > n$: $\binom{n}{p} = 0$

— Pour tout n : $\binom{n}{0} = 1$

— Pour tout n et tout p tel que $1 \leqslant n$: $\binom{n}{p} = \binom{n}{p-1} + \binom{n-1}{p-1}$

Test de validité (dans le shell) : `[BINR(4,j) for j in range(5)]` (affiche la ligne 4 du triangle de Pascal)

Exercice 19. — Algorithme de Syracuse

L'algorithme de Syracuse consiste à faire subir à un entier naturel N l'algorithme suivant :

Si N est pair, alors on transforme N en $N/2$;

Si N est impair, alors on transforme N en $3N+1$.

Par exemple, en appliquant à répétition cet algorithme à l'entier 1, on obtient la suite de valeurs :

$$1\,;\ \ 4\,;\ \ 2\,;\ \ 1\,;\ \ 4\,;\ \ 2\,;\ \ 1\,;\ \ 4\,;\ \ 2\,;\ \ 1\,;\ldots$$

Par exemple (bis), en appliquant à répétition cet algorithme à l'entier 5, on obtient la suite de valeurs :

$$5\,;\ \ 16\,;\ \ 8\,;\ \ 4\,;\ \ 2\,;\ \ 1\,;\ \ 4\,;\ \ 2\,;\ \ 1\,;\ldots$$

Ecrire une fonction en Python `SYR(n,x)` qui reçoit comme paramètres deux entiers naturels n et x, et qui retourne le n-ème terme dans l'algorithme de Syracuse appliqué à x.

Test de validité (dans le shell) : `[SYR(k,5) for k in range(9)]` (on doit obtenir la ligne ci-dessus)

Exercice 20. — Algorithme pour le calcul du PGCD

Ecrire une fonction en Python `PGCD(a,b)` qui reçoit comme paramètres deux entiers naturels a et b, et qui renvoie le PGCD de a et b (noté $a \wedge b$).

On rappelle que :

- $a \wedge 0 = a$
- $a \wedge b = b \wedge r$ où r désigne le reste dans la division euclidienne de a par b

Test de validité (dans le shell) : `[PGCD(15,j) for j in range(5)]`

Intermède - Une brève histoire du temps

Dans le module time de Python, on dispose d'une fonction time(), qui renvoie "l'heure du système" au moment où cette fonction est exécutée. †

On peut faire jouer à cette fonction le rôle d'un chronomètre pour mesurer la durée d'exécution d'une série d'instructions.

Exemple. Le code ci-dessous fait calculer à la machine la somme $S = \sum_{k=0}^{10\,000\,000} k^2$.

Ce calcul ne présente qu'un seul intérêt : il demande du temps à la machine, et la durée d'exécution n'est pas 0.0 (ce qui est le cas par exemple pour $S' = \sum_{k=0}^{10\,000} k^2$).

```
from time import *

T_DEBUT =time()
S =0
for k in range(10**7):
    S =S+k**2
T_FIN =time()

DUREE =T_FIN -T_DEBUT

print('Durée (en secondes): ',DUREE)
```

L'exécution de ce code produit l'affichage : `Durée (en secondes): 2.2870800495147705`

Ce qui conduit à deux réflexions :

— Youpi ! On va pouvoir mesurer des durées, et comparer les complexités temporelles des programmes ! ‡

— On n'a peut-être pas besoin de cette cargaison de décimales, les centièmes de secondes seront très largement suffisants. Pour cela, on peut tronquer la DUREE à 10^{-2} par exemple via l'instruction :

```
DUREE = int(100*(T_FIN - T_DEBUT))/100
```

†. Très précisément, la fonction time() renvoie le nombre de secondes écoulées depuis le 1er janvier 1970 à 00 :00 :00. Cette date (que l'on appelle *epoch*, ou vulgairement *année zéro de l'informatique*) n'a pas été choisie au hasard (cela remonte à la création d'UNIX, le premier système d'exploitation moderne de l'histoire).

Ce système est utilisé par UNIX et par de nombreux langages de programmation pour gérer les dates sans avoir à se soucier de la gestion des noms des jours, des mois de longueurs différentes, des années bissextiles, *etc*...

‡. OK, ce n'est peut-être pas la première chose qui vous est venue à l'esprit.

Un exemple de comparaison de la complexité temporelle :

itératif VS récursif

Soit (u_n) la suite définie par : $u_0 = 0$, $u_1 = 1$ et pour tout n entier naturel : $u_{n+2} = 3u_{n+1} - 2u_n$.
On a écrit ci-dessous deux fonctions permettant de calculer le terme de rang n de cette suite.

Méthode 1 (itérative)

```
def S_ITER(n):
    u,v=0,1
    if n==0:
        return u
    elif n==1:
        return v
    else:
        for k in range(n):
            u,v=v,3*v-2*u
        return u
```

Méthode 2 (récursive)

```
def S_REC(n):
    if n in [0,1]:
        return n
    else:
        return 3*S_REC(n-1) -2 * S_REC(n-2)
```

Et voici le temps nécessaire au calcul des 40 premiers termes de la suite avec chaque fonction :

```
T_DEBUT =time()
L = [S_ITER(k) for k in range(40)]
T_FIN =time()

DUREE =int(100*(T_FIN -T_DEBUT))/100

print('Durée (version  itérative ): ',DUREE)

### Affichage produit dans le shell

>>>Durée (version itérative ):  0.0
```

```
T_DEBUT =time()
L = [S_REC(k) for k in range(40)]
T_FIN =time()

DUREE =int(100*(T_FIN -T_DEBUT))/100

print('Durée (version  récursive ): ',DUREE)

### Affichage produit dans le shell

>>>Durée (version récursive ):  54.09
```

Sur cet exemple, la version récursive consomme beaucoup plus de temps que la version itérative !!!

Mais ce n'est pas le cas général, et sans rentrer dans les détails, c'est ici la nature même du problème qui rend la version récursive beaucoup moins efficace (au moins en termes de temps) que la version itérative.

Remarque. Il est amusant de demander à la machine le calcul de u_{10^4} avec la fonction récursive S_REC. Un joli message d'erreur apparaît :

```
RecursionError: maximum recursion depth exceeded in comparison
```

Message qui signifie grossièrement qu'avant de faire exploser la complexité temporelle, la fonction S_REC fait déjà exploser la complexité spatiale (espace mémoire alloué aux calculs nécessaires pour cette fonction)...

5 - Incontournables du programme - Sur les chaînes de caractères

Il y a trois types de programmes : ceux avec des bugs que vous connaissez, ceux avec des bugs que vous ne connaissez pas, et ceux avec les deux.

Exercice 21. — Ecrire une fonction `NBVOYR(mot)` qui reçoit une chaîne de caractères en paramètre, et qui retourne le nombre de voyelles (minuscules, sans accents) de cette chaîne.

Exercice 22. — Ecrire une fonction `PAL(mot)` qui reçoit une chaîne de caractères en paramètre, et qui retourne True si cette chaîne est un palindrome, False sinon (rappel : 'rotor', 'bob', 'ANNA' sont des palindromes ; 'zeugma' et 'triacontakaitétragone' n'en sont pas).

Exercice 23. — Ecrire une fonction `SCHR(mot1,mot2)` qui reçoit comme paramètres deux chaînes de caractères, et qui détermine si mot2 est une sous-chaîne de mot1.

6 - Ultime incontournable - Dichotomie pour l'équation $f(x) = 0$

"Et qu'on en finisse avec ces sornettes qui feraient rire mon cheval."[§]

Exercice 24. — Ecrire une fonction `DICHOR(f,a,b,epsi)` qui reçoit comme paramètres une fonction f (définie au préalable), deux réels a et b (tels que $a < b$), et un réel epsi, et qui retourne un encadrement d'une solution de $f(x) = 0$ dans $[a, b]$ s'il en existe une, et qui retourne 'Wazaaaaaa' sinon.

Test de validité :

- Définir $f(x) = x^2 - 2$ dans l'éditeur ;
- Dans le shell `DICHOR(f,0,2,0.01)` renvoie un encadrement à 10^{-2} près de $\sqrt{2}$;
- Dans le shell `DICHOR(f,0,1,0.01)` renvoie 'Wazaaaaa'

§. Maurice Druon (1918-2009).