Chapitre 14 - La récursivité

Définition :

Un algorithme est dit **récursif s'il s'appelle lui-même** directement ou indirectement via l'appel d'une ou de plusieurs autres fonctions qui elles-mêmes finissent par l'appeler.

La récursivité est un concept fondamental en informatique qui met naturellement en pratique un mode de pensée puissant qui consiste à pouvoir découper la tâche à réaliser en sous-tâches de mêmes natures mais plus petites qui finalement sont simples à résoudre.

1- EXEMPLE POUR COMPRENDRE LE FONCTIONNEMENT :

```
def f(n):
    if n == 1 :
        print(f"Arrêt des appels récursifs")
    else :
        print(f"appel recursif f({n-1})")
        f(n-1)
        print(f"n = {n-1}")
```

Exécution de `>>> f(4)` :

Empilement dans la Pile d'exécution	*Dépilement* – Affichage dans la console

Fonctionnement :

Lors de l'exécution d'un algorithme récursif, les appels récursifs successifs sont stockés dans une pile, c'est la **pile d'exécution**.
Plus précisément, la pile d'exécution est un emplacement mémoire destiné à stocker les paramètres, les variables locales ainsi que les adresses mémoires de retour des fonctions en cours d'exécution.

2- AUTRE EXEMPLE : SOMME DES ENTIERS INFERIEURS A n

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

Exécution de `>>> s(4)` :

Empilement dans la Pile d'exécution	*Dépilement*

L'exécution dans la console de `s(4)` renvoie ainsi la valeur …… .

Le code de cette fonction permet ainsi de calculer la somme des entiers inférieurs ou égal à n . Pour calculer cette même somme on peut utiliser une version dite **itérative** de ce code. On en donne ci-dessous 2 exemples :

```
def sommeIteratif(n) :
    s = 0
    while n >= 0 :
        s = s + n
        n = n - 1
    return s
```

ou

```
def sommeIteratifBis(n) :
    s = 0
    for i in range(n+1) :
        s = s + i
    return s
```

A savoir :

- Lorsque l'on écrit une fonction récursive en Python, on partage généralement son code en deux parties :
  - Partie 1 comprenant une condition d'arrêt pour stopper les appels récursifs.
  - Partie 2 comprenant les appels récursifs.

- La récursivité est un moyen de répéter des blocs d'instructions sans utiliser de boucle while ou for.

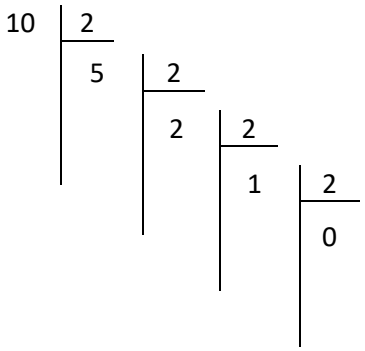
La programmation récursive n'est ni meilleure, ni pire que, la programmation itérative. Les fonctions récursives peuvent généralement aussi être programmées de façon itérative. Cependant, en cas de nombreux appels récursifs, la mémoire de la machine sera trop fortement sollicitée et l'exécution ralentie, voire impossible. Le nombre d'appels récursifs est généralement limité à 1000. On peut repousser cette limite jusqu'à 2000, en utilisant la fonction *setrecursionlimit()* de la bibliothèque *sys* .

```
import sys
#print(sys.getrecursionlimit())
sys.setrecursionlimit(2000)
```

Le choix entre une solution récursive ou une solution itérative est généralement guidé par le type de problème à résoudre car certains problèmes s'écrivent *naturellement* de façon récursive.

3- AUTRE EXEMPLE : CONVERSION DECIMALE → BINAIRE

Exemple de la conversion binaire de 10 :




```
def conv(n) :
    if n <= 1 :
        return str(n)
    else :
```

Exécution de `conv(10)` :

Empilement dans la Pile d'exécution	*Dépilement*

L'exécution dans la console de `conv(10)` renvoie ainsi la valeur ……

On aurait pu utiliser une version dite **itérative** de ce code. On en donne ci-contre 1 exemple :

```
def convIteratif(n) :
    bin = ""
    while n//2 != 0 :
        bin = str(n%2) + bin
        n = n // 2
    return "1" + bin
```

4- AUTRE EXEMPLE : MAXIMUM D'UNE LISTE

Exemple du maximum de la liste `l = [4,9,-2,12,1]` :

```
def maximum(l):
    if len(l) == 1:
        return l[0]
    else:
        a = l[0]
        b = maximum(l[1:])
        if a > b :
        else :
```

Exécution de `>>> maximum([4,9,-2,12,1])` :

Empilement dans la Pile d'exécution	*Dépilement*
	`maximum([12,1])`
	`maximum([-2,12,1])`
	`maximum([9,-2,12,1])`
	`maximum([4,9,-2,12,1])`

L'exécution dans la console de `maximum([4,9,-2,12,1])` renvoie ainsi la valeur ……

5- AUTRE EXEMPLE : TRI D'UNE LISTE

Exemple du tri de la liste `l = [4,9,-2,12,1]`

```
def tri(l) :
    if len(l) == 1 :
        return l
    else :
        max = maximum(l)
        l.remove(max)
        l = tri(l)
        l.append(max)
        return l
```

Info : la méthode *remove()* permet de supprimer de la liste le premier élément dont la valeur est égale à celle mise en argument.

Exécution de `tri([4,9,-2,12,1])` :

Empilement dans la Pile d'exécution	*Dépilement*
`l = tri(l) ; l.append(  )`	
`l = tri(l) ; l.append(  )`	
`l = tri(l) ; l.append(  )`	
`l = tri(l) ; l.append(  )`	

L'exécution dans la console de `tri([4,9,-2,12,1])` renvoie ainsi la valeur ……

Remarque : En n'ayant aucun retour comme dans le script ci-contre, la liste est triée si elle est définie dans le programme principal :

```
>>> tri([-2,40,-88])
```

ne renvoie rien.

```
>>> l = [4,9,-2,12,1]

>>> tri(l)

>>> print(l)
[-2, 1, 4, 9, 12]
```

Par contre si elle est définie dans le programme principal (variable globale) la liste est tout de même triée.

```
def tri(l) :
    if len(l) == 1 :
        pass
    else :
        max = maximum(l)
        l.remove(max)
        tri(l)
        l.append(max)
```

6- AUTRE EXEMPLE : SUPPRIMER LES ESPACES DANS UNE PHRASE

La fonction *esp()* ci-contre renvoie la chaine de caractère mise en argument, mais sans les caractères « *espace* ». L'algorithme est ici itératif.

```
def esp(phrase) :
    new = ''
    for c in phrase :
        if c != ' ' : new = new + c
    return new
```

On donne ci-contre une version récursive à compléter :

```
def esp(phrase) :
    if len(phrase) == 1 :
        return phrase
    else :
        c = ""
        if phrase[0] != ' ' : c = phrase[0]
        return
```

Exemple de l'exécution de : `>>> esp('j e')`

Empilement dans la Pile d'exécution	*Dépilement*

L'exécution dans la console de `>>> esp('j e')` renvoie ainsi la valeur ……

7- AUTRE EXEMPLE : REPERER UN PALINDROME

Définition du Palindrome sur Wikipédia :

« Le **palindrome** (adjectif et substantif masculin), du grec πάλιν / *pálin* (« en arrière ») et δρόμος / *drómos* (« chemin, voie »), aussi appelé **palindrome de lettres,** est une figure de style désignant un mot ou une phrase dont l'ordre des lettres reste le même qu'on les lise de gauche à droite ou de droite à gauche, comme dans la phrase « Esope reste ici et se repose » ou encore « La mariee ira mal » à un accent près. »

La fonction *pal()* ci-dessous renvoie True si la chaine de caractère en argument est un palindrome. Elle renvoie False sinon.

```
def pal(mot) :
    if len(mot) < 2 : return True
    else : return (mot[0] == mot[-1]) and                    )
```

Exemple de l'exécution de : `pal('ratar')`

Empilement dans la Pile d'exécution	*Dépilement*

L'exécution dans la console de `pal('ratar')` renvoie ainsi la valeur ……

Autres exécutions :

```
phrase = esp("Esope reste ici et se repose".lower())
print(pal(phrase))
phrase = esp("La mariee ira mal".lower())
print(pal(phrase))
```

8- AUTRE EXEMPLE : DESSIN DE FRACTALES

Définition d'un fractale sur Wikipédia : Une figure **fractale** est un objet mathématique qui présente une structure similaire à toutes les échelles. C'est un objet géométrique « infiniment morcelé » dont des détails sont observables à une échelle arbitrairement choisie. En zoomant sur une partie de la figure, il est possible de retrouver toute la figure ; on dit alors qu'elle est « autosimilaire ».

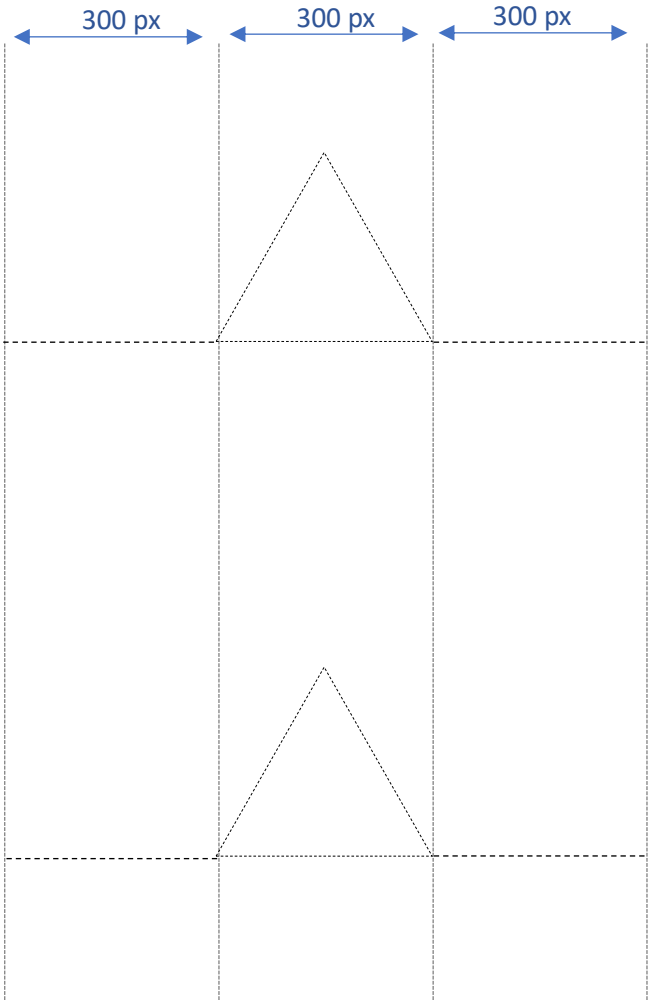
On donne ci-dessous la fonction *koch()* qui permet de tracer un dessin de ce type.

```
from turtle import forward,left

def koch(a,n) :
    if n == 0 : forward(a)
    else :
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
        right(120)
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
```

- Exemple de l'exécution de `koch(900,1)` :

```
def koch(a,n) :
    if n == 0 : forward(a)
    else :
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
        right(120)
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
```



- Exemple de l'exécution de `koch(900,2)` :

```
def koch(a,n) :
    if n == 0 : forward(a)
    else :
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
        right(120)
        koch(a//3,n-1)
        left(60)
        koch(a//3,n-1)
```

Ordre d'appels des fonctions *: k() pour koch() , l() pour left() , r() pour right() , f() pour forward()*

- Appel 1 : - Appel 2 : - Appel 3 : - Appel 4 : - Appel 5 : - Appel 6 : - Appel 7 : - Appel 8 : - Appel 9 : - Appel 10 : - Appel 11 :	- Appel 12 : - Appel 13 : - Appel 14 : - Appel 15 : - Appel 16 : - Appel 17 : - Etc ….