# T NSI Récursivité

## 1. Définitions :

**Programmation itérative :** consiste à répéter un même bloc d'instructions jusqu'à obtenir le résultat souhaité.

**Programmation récursive :** est une technique de programmation qui remplace les instructions de boucle (while, for, etc.) par des appels de **fonctions** (ou de **structures informatiques**) qui s'appellent elles-mêmes.

**Une fonction récursive** est une fonction qui contient au moins un appel à elle-même. Une fonction récursive possède toujours :

- Un ou plusieurs cas de base
- Un cas récursif (s'appelle elle-même)

**Un langage récursif** est un langage dans lequel on peut programmer des fonctions récursives. Python est un langage récursif.

Les fonctions récursives, permettent, pour certains types de problèmes

- d'écrire plus facilement les programmes
- de vérifier plus facilement que les programmes sont corrects

Exemple : on veut écrire un programme qui :

- Prend un entier n > =0
- Affiche les nombres de 1 à n par ordre croissant

```
n= int(input("Donner le nombre d'entiers n à afficher "))


def afficher_iteratif(n) :
    i = 1
    while( i <= n) :
        print(i)
        i+= 1

def afficher_recursif(n):
    #cas de base
    if n == 1 :
      print("Décollage")
      print(n)
    #cas recursif
    else :
        print(f"appel afficher recursif n : {n}")
        afficher_recursif(n-1)
        print(n)

afficher_iteratif((n))
afficher_recursif(n)
```

# T NSI Récursivité

Faite dérouler à la main l'exécution de la fonction récursive :

# 2. Différents exemples d'utilisation de la méthode récursive :

## 2.1. Calcul du factoriel :

On rappelle que en mathématiques, la **factorielle** d'un entier naturel *n*, notée *n*!, ce qui se lit soit " factorielle de n " soit " factorielle n ", est le produit des nombres entiers strictement positifs inférieurs ou égaux à *n*.

Soit : $$n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Produire une code qui retourne le factoriel d'un nombre entier n de manière :

**Itératif : fonction factoriel_itératif**

Remplacer les ? pour réaliser la fonction `factoriel_iteratif`

```
def factoriel_iteratif(n) :
  if n ??? :
    print(f"factoriel iteratif {n}: 1")
    return ?
  else :
    fact = factoriel_iteratif(0)
    for i in range(????,????):
      fact = ??????
      print(f"factoriel iteratif {i}  : {fact}")
    return ?????

print(factoriel_iteratif(n))
```

Combien d'étapes de calcul pour réaliser le factoriel de 5 ?

**Recursive : fonction factoriel_recursif ( différentier le cas particulier )**

Remplacer les ? pour réaliser la fonction `factoriel_iteratif ,` la fonction pourrai être écrite avec moins de lignes et en s'affranchissant de la variable intermédiaire fact . Ii il s'agit de mettre en évidence les différents appels à la fonction.

```
def factoriel_recursif(n) :
  print(f"factoriel_recursif({n}) ")
  if n ?????:
    fact = ????
    return ?????
  else :
    fact = ?????????
    print(f"factoriel_recursif({n}) :{fact} ")
    return ???????


print(factoriel_recursif(n))
```

Toujours pour le factoriel de 5 combien d'appels à la fonction sont effectués. Expliquer le fonctionnement.

## 2.2. Calcul de suites mathématiques

On définit une suite mathématique U par $U_n = \begin{cases} 5 \; pour \;\; n = 0 \\ U_{n+1} = \; U_n + 3 \end{cases}$

Calculer les différents termes de $U_0$ à $U_5$ à la main. Proposer un code permettant de le faire automatiquement de manière récursive.

Inspirer vous du calcul de factoriel pour obtenir :

```
Donner le nombre d'entiers n à afficher 5
Appel à U5
Appel à U4
Appel à U3
Appel à U2
Appel à U1
Appel à U0
partiel  U0 : 5
partiel  U1 : 8
partiel  U2 : 11
partiel  U3 : 14
partiel  U4 : 17
partiel  U5 : 20
20
```

## 2.3. Une histoire de lapins la suite de Fibonacci

Les naissances des lapins selon Fibonacci

En janvier un jeune couple de lapins est réuni.
En février, ce couple donne naissance à un couple de lapereaux.
La suite suit les règles suivantes :

- Un couple adulte donne naissance à un couple de lapereaux tous les mois ;
- Par contre un couple de lapereaux doit attendre un mois avant d'atteindre sa maturité et, adulte, se mettre à procréer tous les mois.

Ce qui établit une règle de procréation : pour le mois suivant il y aura les lapins existaient déjà + les nouveaux

**Soit Fn = nombre de lapins au mois n**

**$F_0 = 0$**

**$F_1 = 1$**

**$F_n = F_{n-1} + F_{n-2}$**


Leonardo da Pisa, dit Fibonacci

Proposer un algorithme récursif qui calcul le nombres de lapins au nième mois.

Calculer pour n de 0 à 10.

### 2.3.1. Le tri rapide ou Quick Sort

Le tri rapide est un autre algorithme de tri, basé sur la récursivité, qui est très utilisé pour sa relative simplicité et sa rapidité.

Il consiste à choisir un nombre de la liste au hasard, que l'on appelle nombre pivot, et auquel on compare les autres valeurs de la liste.

Si la valeur comparée est inférieure au nombre pivot, on la place dans une liste que l'on nomme "inferieure" par exemple, sinon on la place dans une liste "superieure". Une fois toutes les valeurs comparées, on obtient normalement 2 listes : la liste "inferieure" contenant toutes les valeurs inférieures au pivot et la liste "supérieure" contenant toutes celles supérieures.

Il ne reste plus qu'à répéter cette méthode pour les 2 listes, jusqu'à ce que celles-ci ne contiennent plus qu'une seule valeur.

Compléter le code suivant pour réaliser cet algorithme

```
ll =[-10,5,8,45,-96,12,897,-50]
def tri_rapide(list):
    inferieure = []
    pivot = []
    superieure = []
    #cas particulier
    if len(list) < 2:
        return ??????
    #cas general
    pivot_valeur =  ????????
    for i in list:
        if i < pivot_valeur:
            ??????????????????
        elif i > pivot_valeur:
            ??????????????????????
        else:
            pivot.append(i)
    return ??????????????+ pivot + ????????????????


tri_rapide(ll)
```

2.4. Fractales

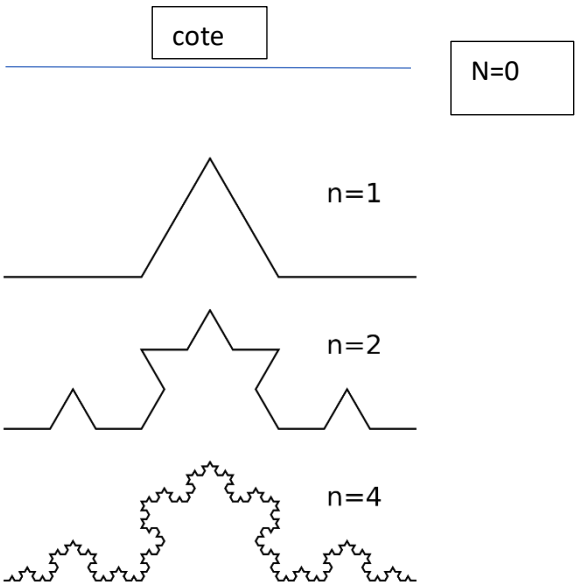
# Courbe de Koch

On peut la créer à partir d'un segment de droite, en modifiant récursivement chaque segment de droite de la façon suivante :

1. On divise le segment de droite en trois segments de longueurs égales.
2. On construit un triangle équilatéral ayant pour base le segment médian de la première étape.
3. On supprime le segment de droite qui était la base du triangle de la deuxième étape.

Au bout de ces trois étapes, l'objet résultant a une forme similaire à une section transversale d'un chapeau de sorcière.

La courbe de Koch est la limite des courbes obtenues, lorsqu'on répète indéfiniment les étapes mentionnées ci-avant.




- A l'aide de turtle pour une longueur "cote" ( longueur du segment ) écrivez le code d'une fonction kock qui trace la première courbe pour n =1

- On va écrire une fonction récursive qui prend comme attribue cote et un nombre n

Différentier le ca n = 0 qui donne la courbe pour n = 0

Les autres cas ou chaque segment est lui-même une courbe de Koch au 1/3