Travaux pratiques - Récursivité

Exercice 1 Suite de Fibonacci

L'objectif de cet exercice est d'implémenter les fonctions récursive et itérative pour calculer le terme de rang n de la suite de Fibonacci. Nous allons également étudier le coût en temps de calcul de chacune des versions de cette fonction.

1. Implémenter la fonction fibonnaci récursive du TD et tester à l'aide du code suivent pour $n = 5$. Vérifier que votre résultat est 5.

```
f = fibonacci_rec(5)
print(f)
```

2. Implementer la fonction itérative et tester avec $n = 5$.
3. Tester votre fonction avec $n = 50$. Expliquer le comportement de cet appel de fonction.
4. Nous allons maintenant utiliser le module `time` pour mesurer la durée d'exécution des fonctions récursive et itérative. Etudoer la dicumentation python du modele `time`, puis copier et compléter le code suivant pour afficher le temps d'exécution en nano secondes des fonctions récursive et itérative pour $n = 5$.

```
import time
def fibonacci_rec(n):
    ...
    code de la fonction recursive
    ...

def fibonacci_it(n):
    ...
    code de la fonction iterative
    ...

#PROGRAMME PRINCIPAL
n = 25
start = time.time_ns()
f_rec = fibonacci_rec(n)
end = ...
d_rec = ...
print("n = ", n, " fib_rec(n) = ",...," duration = ",...," nanosecondes")

start = ...
f_it = fibonacci_it(n)
end = ...
d_it = ...
print("n = ", n, " fib_it(n) = ",f_it," duration = ",d_it," nanosecondes")
```

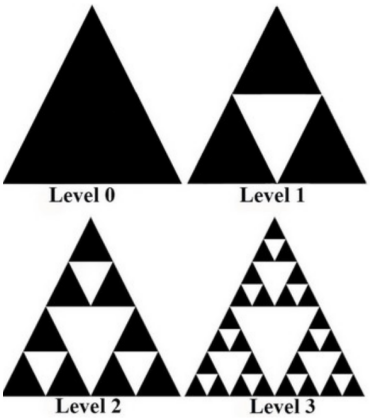
5. A l'aide du code précédent, compléter le tableau des durées d'exécution suivant :

Durée en nanosecondes pour $n =$	1	5	10	20	30	50	100	500	1000
Fibonacci récursive									
Fibonacci itérative									

6. Implémenter la fonction récursive avec mémoire (TD ex 7) et ajouter une ligne dans votre tableau des durées d'exécution. Comparer les résultats.

Exercice 2 Triangle de Sierpinski

Le triangle de Sierpinski est une fractale définie de manière récursive. La figure est composée de petits triangles identiques, chacun d'eux étant une version réduite du triangle principal.



1. Expliquer les étapes pour construire une triangle de Sierpinski de niveau 1 et de longueur 120.
2. En déduire le cas de base et les appels récursifs d'une fonction récursive permettant de construire un triangle de Sierpinski de niveau n.
3. Pour afficher un triangle de Sierpinski, nous allons utiliser le module `Turtle`. Voici quelques commandes `Turtle` à connaitre pour construire un triangle à l'aide d'un objet `t` de type `Turtle` (Cet objet sera instancié dans le programme principal et passé en paramètre de la fonction récursive).
   Le principe de traçage de figures géométriques ressemble fortement à celui du logiciel **Scratch** connu depuis la classe 3ème. Il s'agit de déplacer un crayon dans un repère. On peut positionner le crayon en mode écriture (`pendown`) ou en mode déplacement uniquement (`penup`).
   - Avancer le crayon de n pixels : `t.forward(n)`
   - Reculer le crayon de n pixels : `t.backward(n)`
   - Faire pivoter le crayon de x degrés dans le sens direct : `t.right(x)`
   - Faire pivoter le crayon de x degrés dans le sens indirect : `t.left(x)`
   - Positionner le crayon sur un point de coordonnées $(x; y)$ dans le repère : `t.goto(x,y)`. Le centre de l'écran a pour coordonnées $0; 0)$.

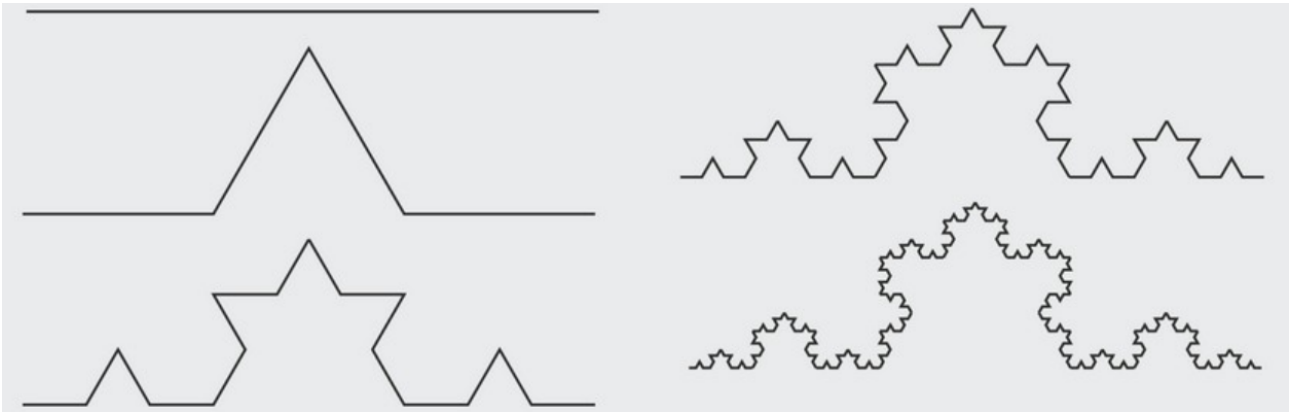
   Copier et compléter le code suivant de la fonction récursive `dessiner_triangle_sierpinski` qui dessine le triangle de Sierpinski à un niveau de profondeur donné. La fonction prend en paramètre un objet de type `Turtle`, la mesure des côtés du triangle (exprimé en pixel) et le niveau de profondeur.

```
import turtle

def dessiner_triangle_sierpinski(t, ordre, longueur):
    # A COMPLETER


# Tester la fonction
turtle.speed(2)
turtle.penup()
turtle.goto(-150, -150)
turtle.pendown()
dessiner_triangle_sierpinski(turtle, 3, 300)

# Attendre que la fenetre Turtle soit fermee
turtle.done()
```

4. Testez votre fonction en dessinant le triangle de Sierpinski à différents niveaux de profondeur.

Exercice 3 **Courbe de Koch**

La courbe de Koch est une autre fractale définie de manière récursive. À chaque itération, chaque segment est remplacé par quatre segments de longueurs égales formant un triangle équilatéral.



1. La première courbe (un segment) est le niveau 0. La seconde courbe avec uniquement un triangle est le niveau 1.
   Expliquer les étapes pour construire la courbe de de niveau 1 et de longueur 120.
2. En déduire le cas de base et les appels récursifs d'une fonction récursive permettant de construire un courbe de Koch de niveau n.
3. Pour afficher une courbe de Koch, nous allons utiliser le module `Turtle` comme pour l'exercice précédent.
   Copier et compléter le code suivant de la fonction récursive `dessiner_courbe_koch` qui dessine une courbe de Koch à un niveau de profondeur donné et de longueur donnée. La fonction prend en paramètre un objet de type `Turtle`, la longueur de la courbe (exprimé en pixel) et le niveau de profondeur.

```
import turtle

def dessiner_courbe_koch(t, ordre, longueur):
    #A COMPLETER

# Courbe de Koch
turtle.reset()
turtle.speed(2)
turtle.penup()
turtle.goto(-150, 0)
turtle.pendown()
dessiner_courbe_koch(turtle, 4, 300)

# Attendre que la fenetre Turtle soit fermee
turtle.done()
```

4. Testez votre fonction en dessinant des courbes de Koch à différents niveaux de profondeur.