Contrôle n°1 : récursivité

Exercice 1 : un peu de bases

1) Ecrire une fonction « multiplie » de paramètres « chaiN » et « n » qui va renvoyer la chaine obtenue après « n » concaténations de « chaiN ». la fonction sera complètement documentée et contiendra une boucle.
2) Quels sont les deux parties obligatoires dans toute fonction récursive.

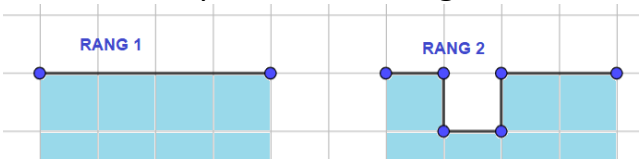
Exercice 2 : Suite de Fibonacci

A titre de rappel la suite de Fibonacci est la suite caractérisée par $\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$.

1) Propose le code d'une fonction récursive en python fibo(n) donnant le n^{ième} terme de la suite de Fibonacci.
2) A votre avis que va-t-il se passer si l'on appelle la fonction écrite avec $n = 10000$? Pourquoi ?

Exercice 3 : Une petite fractale

En s'inspirant du travail fait avec le flocon de koch, on veut créer en python avec la librairie turtle une figure à base carrée qui va être creusée de plus en plus par le même motif.
Ce programme contiendra deux fonctions : une qui sera récursive et qui tracera des lignes brisées :



Au rang d'après on appliquera à chacun des cinq segments précédemment dessiné la même transition.
L'autre fonction aura deux paramètres : la dimension de base du carré et « n » la profondeur de la fractale. Elle fera quatre appels de la fonction récursive pour dessiner le carré initial.

1) Sur votre copie vous indiquerez comment compléter les parties **1** , **2** et **3**
2) Expliquer le but des 4 premières lignes de la fonction fractale
3) A quoi correspond « _ » dans la cinquième ligne de la fonction fractale ?

```
from turtle import *

def fract(longueur, n):
        if n == 1:
            1
        else:
            2

def fractale(taille, etape):
    speed(0)
    penup()
    goto(-taille/2, taille/2)
    pendown()
    for _ in range(4):
        3

fractale(300, 3)
done()
```

```
def f(n):
    """n est un entier strictement positif"""
    if n==1 : return 0
    else : return 1 + f(n//2)
```

Exercice 4

En utilisant la fonction ci-contre remplir le tableau suivant :

n	1	2	3	4	5	6	7	8	9	10	11	12
$f(n)$												

Bonus : à votre avis à quoi sert la fonction ?

Exercice 5

D'après Wikipedia la **conjecture de Syracuse**, encore appelée **conjecture de Collatz**, **conjecture d'Ulam**, **conjecture tchèque**, **problème de Kakutani** ou **problème 3*x* + 1**, est l'hypothèse mathématique selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1.

Une **suite de Syracuse** est une suite d'entiers naturels définie de la manière suivante : on part d'un nombre entier strictement positif ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et l'on ajoute 1. En répétant l'opération, on obtient une suite d'entiers strictement positifs dont chacun ne dépend que de son prédécesseur.

Par exemple, à partir de 14, on construit la suite des nombres : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1 et ainsi de suite. C'est la suite de Syracuse du nombre 14. Après que le nombre 1 a été atteint, la suite des valeurs 1, 4, 2, 1, 4, 2… se répète indéfiniment en un cycle de longueur 3, appelé cycle trivial.

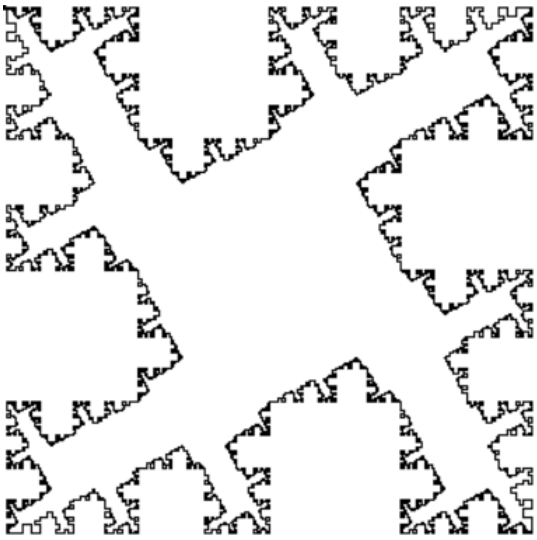
On veut créer une fonction donnant le nombre d'étapes nécessaires pour arriver à 1 quand on a une suite de Syracuse part d'un nombre donné U.
Sur votre copie vous indiquerez comment compléter les parties **1** et **2** .

Syracuse(14,0) provoquera l'affichage « 1 est atteint pour la première fois au rang 17 »

Programme incomplet :

```
def Syracuse(u :int,n=0 :int) :
    """u est la valeur du terme observé
    et n est le rang de ce terme"""
    if u==1 :
        1
    else:
        2
```

Bonus pour les petits futés :
Soit `phrase` une chaîne de caractère. Proposer une fonction **récursive** `inverse(phrase:str)->str` qui accepte en entrée une chaîne `phrase` en argument, et qui renvoie en sortie une chaîne inversée. Exemple : `allo` devient `olla`



Exemple de la fractale générée par l'exercice 3

Correction Contrôle n°1 : récursivité

Exercice 1 : un peu de bases

1) def multiplie(chaN : string, n : int) -> string :
   New = ‘’
   for i in range(n): New+=chaN
   return New
2) toute fonction récursive aura une partie dédiée à l’arrêt et une partie s’appelant elle-même.

Exercice 2 : Suite de Fibonacci

1) def fibo(n):
   if n==0 or n==1 : return n
   return fibo(n-1)+fibo(n-2)
2) on va avoir un message « erreur de récursion », mais avant ça, ça va prendre un temps très très long de calcul

Exercice 3 : Une petite fractale

1) voir ci-contre
2) utiliser la vitesse maximale , commencer la figure à un endroit avantageux sans laisser de trace du déplacement.
3) C’est un compteur ne générant pas de création inutile de variable.

```
from turtle import *

if n == 1:
    forward(longueur)
  else:
    fract(longueur / 4, n - 1)
    right(90)
    fract(longueur / 4, n - 1)
    left(90)
    fract(longueur / 4, n - 1)
    left(90)
    fract(longueur / 4, n - 1)
    right(90)
    fract(2*longueur / 4, n - 1)


def fractale(taille, etape):
  speed(0)
  penup()
  goto(-taille/2, taille/2)
  pendown()
  for _ in range(4):
    fract(taille, etape)
    t.right(90)

fractale(300, 3)
done()
```

Exercice 4

```
def f(n):
  """n est un entier strictement positif"""
  if n==1 : return 0
  else : return 1 + f(n//2)
```

Que fait l'algorithme récursif suivant? donné en pseudo-code ..

n	1	2	3	4	5	6	7	8	9	10	11	12
$f(n)$	0	1	1	2	2	2	2	3	3	3	3	3

Ça donne la plus grande valeur de m tel que $n \geq 2^m$

Exercice 5

```
def Syracuse(u :int,n=0) :
    """u est la valeur du terme observé
    et n est le rang de ce terme"""
    if u==1 :
      print(f"1 est atteint pour la première
fois au rang {n}")
    else:
      if u%2==0 : Syracuse(u/2,n+1)
      else : Syracuse(3*u+1,n+1)
```

Exercice Bonus

```
def inverse(phrase:str)->str :
  if len(phrase)==0 : return phrase
  return inverse(phrase[1:])+phrase[0]
```