Exo 1 : On cherche à calculer la somme $0+1+2+\ldots N$ sans utiliser la formule mathématique.

a) Écrire une fonction `somN(N : int) -> int`, qui renvoit la somme des entiers naturels de 0 à N.

b) Écrire la fonction récursive `somN_rec(N : int) -> int`, qui répond à la question.

c) Déroulé l'état de la pile d'exécution pour $N = 4$.

Exo 2 : de manière rudimentaire si $(x,y) \in \mathbb{Z} \times \mathbb{N}$, le calcul de la somme $x+y$ se fait également par la relation : $s(x,y) = s(x+1,y-1)$ et $s(x,0) = x$.

Proposer une fonction récursive qui calcule la somme $x+y$.

Exo 3 : Écrire une fonction récursive `puissR(x : float, n : int) -> float`, qui renvoie x^n ($n \in \mathbb{N}$).

Rq : une méthode beaucoup plus efficace existe en remarquant que $x^{2k} = x^k \times x^k$ et $x^{2k+1} = x^k \times x^k \times x$.

Exo 4 : Écrire une fonction récursive `rechercheTab(tab : list, v : int) -> bool`, qui renvoie `True` si v est une valeur de `tab`, `False` sinon.

Exo 5 : Une fonction de recherche dichotomique récursive `rechercherDicho(tab : list, v : int) -> bool`, qui renvoie `True` si v est une valeur de `tab`, `False` sinon, peut se définit ainsi :

◇ si le tableau est de longueur 1, la fonction renvoie `True` si v est la valeur du tableau, `False` sinon.

◇ sinon la fonction renvoie le résultat d'elle-même appliquée au demi-tableau de gauche ou le résultat d'elle-même appliquée au demi-tableau de droite

Implémenter la fonction `rechercherDicho()` et la tester.

Exo 6 : Un palindrome est mot qui se lit de la même manière à l'endroit comme à l'envers (en omettant les espaces et autres signes de ponctuation). *e.g.* "elle", "kayak", "Élu par cette crapule" et "Engage le jeu que je le gagne" sont des palindromes tandis que "il", "papa" ou "Seth a été à Sète" n'en sont pas.

On rappelle que `"elephant"[2:-3]` renvoie la chaîne `eph` (du 2è caractère au 3è caractère en partant de la fin exclu)

a) Écrire une fonction itérative `palindromeI(mot : str) -> bool`, qui vérifie si `mot` est un palindrome.

b) Un mot vide est-il un palindromme ? un mot contenant un seul caractère ?

c) Supposons que nous ne puissions comparer qu'un couple de lettres, lesquelles vériferiez-vous pour savoir si le mot peut être un palindrome ? Comment se ramener à ce cas en avançant dans la comparaison du mot ?

d) Écrire une fonction récursive `palindromeR(mot : str) -> bool`, qui vérifie si `mot` est un palindrome.

e) Dérouler la pile d'exécution pour le mot `"kayak"`.

Exo 7 : Connaissant $u_0 \in \mathbb{N}$, la suite de Syracuse est définie sur $\mathbb{N}$ par $u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{sinon} \end{cases}$.

a) calculer les 6 premiers termes pour $u_0 = 4$, que remarquez-vous ?

b) calculer les 7 premiers termes pour $u_0 = 10$, que remarquez-vous ?

On décide donc d'arrêter le calcul de u_{n+1} dès que $u_n = 1$,

c) écrire une fonction récursive `syracuseR(u0 : int) -> list`, qui renvoye toutes les valeurs de u_0 à $u_n = 1$.

d) écrire une fonction itérative `syracuseI(u0 : int) -> list`, qui renvoye toutes les valeurs de u_0 à $u_n = 1$.

Exo 8 : Soit $a \in \mathbb{R}_+$, la méthode d'Héron d'Alexandrie (I^erS ap. JC) permet d'approcher très rapidement la valeur de $\sqrt{a}$. On définit donc la suite (u_n) par $u_n = \begin{cases} 1 & \text{si n=0} \\ \frac{1}{2}\left(u_{n-1} + \frac{a}{u_{n-1}}\right) & \text{sinon} \end{cases}$.

On souhaite calculer $\sqrt{5}$,

a) préciser la formule définissant (u_n)

b) quelle est la condition de terminaison ?

c) écrire la fonction récursive `racine5R(n : int) -> float`, qui renvoye la valeur u_n.

d) écrire une fonction itérative `racine5I(n : int) -> float`, qui renvoye la valeur u_n.

Exo 9 : Expliquer ce que fait cette fonction récursive où $(n,m) \in \mathbb{N} \times \mathbb{N}$.

```
def mystere(m : int, n : int) -> int :
    if m==0 or m==n :
        return 1
    return mystere( m-1, n-1 ) + mystere( m, n-1 )
```

Exo 10 : Expliquer ce que fait cette fonction récursive où $n \in \mathbb{N}^*$.

```
def mystere(n : int) -> None :
    if n==0:
        pass    # rien
    mystere( n//2 ) # division entière par 2
    print( n%2, end='' ) # end='' permet de ne pas aller a la ligne a la fin du print
```

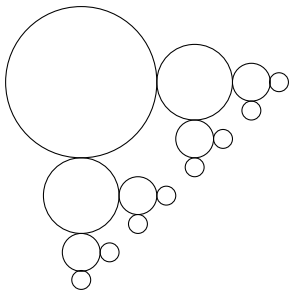
Exo 11 : Sans passer par la fonction `str()`, écrire une fonction récursive `nb2Chiffres( n : int ) -> int`, qui renvoie le nombre de chiffres dans l'écriture de $n \in \mathbb{N}$.
e.g. `nb2Chiffres(123)` renverra 3 et `nb2Chiffres(43210)` renverra 5.

on pourra remarquer que l'entier `n//10` possède un chiffre de moins que l'entier `n`

Exo 12 : (de aberthelot) On dispose d'un paquet de $n \in \mathbb{N}^*$ cartes numérotés de 1 à n triées dans l'ordre de sorte que 1 soit en dessous. On prend la carte supérieure et on la replace au-dessous du paquet, puis on jette la carte du dessus du paquet. Quelle sera la dernière carte survivante ?

Écrire une fonction qui retourne la valeur de la carte survivante.

Exo 13 : (de aberthelot) On souhaite obtenir la figure ci-contre de manière récursive (ici profondeur 4).



La figure est formée d'un cercle et de deux copies de ce cercle ayant subies une réduction d'un facteur 2, ces deux petits cercles étant tangents extérieurement au cercle initial et tels que les lignes des centres sont parallèles aux axes du repère. Ces deux petits cercles deviennent à leur tour "cercle initial" pour poursuivre la figure.
On suppose que la fonction `cercle(coord : tuple, r : float) -> None`, est déjà définie et dessine un cercle de centre de coordonnées `coord` et de rayon `r`.
Proposer une fonction `python` qui réponde au problème.

Exo 14 : On souhaite tracé le triangle de Sierpinski (*cf.* la bédé *les fractals* de Ian Stewart) en partant d'un triangle équilatéral. Voici les résultats attendus :

triangle_sierpinski(n)	n=1	n=2	n=3
résultat			

a) À partir des points $A(x_A,y_A)$, $B(x_B,y_B)$, $C(x_C,y_C)$ sommets du triangle parent ABC, quelles sont les coordonnées des points sommets des trois triangles fils ?

b) Compléter la fonction récursive d'appel principal `triangle_sierpinski(n : int) -> None`, qui trace le triangle de Sierpinski de profondeur n.

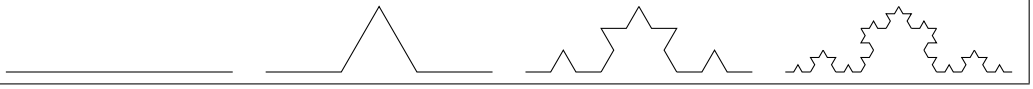
```
import matplotlib.pyplot as plt
from numpy import sqrt
def triangle(A : tuple, B : tuple, C : tuple) -> None : # trace le triangle ABC
    plt.fill( [ A[0], B[0], C[0] ], [ A[1], B[1], C[1] ], 'k')

def triangle_sierpinski(n : int, A : tuple =(0,0), B : tuple = (1,0), C : tuple = (1/2,
↪  sqrt(3)/2) ) -> None : # par défaut, triangle équilatéral de côté 1
    if n == 1 :
        ...
    else :
        ... # plusieurs lignes
    plt.show()
```

Rq : sous spyder, vérifier que `Tools/Preferences/Ipython console/Graphics/backend -> automatic` (relancer le kernel si changement)

Exo 15 : On souhaite dessiner le flocon de von Koch.

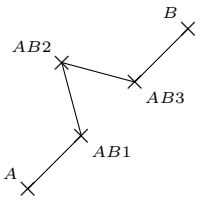
Voici les résultats attendus ou chaque simili triangle est un triangle équilatéral sans la base :

flocon(n)	n=0	n=1	n=2	n=3
résultat				

a) Soient $A(x_A,y_A)$, $B(x_B,y_B)$ deux points du plan, déterminer les coordonnées des points $AB1$, $AB3$ respectivement au tiers et deux tiers du segment $[AB]$ (*i.e.* $AB1$ est deux fois plus près de A que de B, tandis que $AB3$ est deux fois plus près de B que de A).

On admettra que le point $AB2$, image du point $AB3$ par la rotation de centre $AB1$ d'angle $\pi/3$, a pour coordonnées :

$$\begin{cases} x = x_{AB1} + (x_{AB3} - x_{AB1}) \times \frac{1}{2} - (y_{AB3} - y_{AB1}) \times \frac{\sqrt{3}}{2} \\ y = y_{AB1} + (x_{AB3} - x_{AB1}) \times \frac{\sqrt{3}}{2} + (y_{AB3} - y_{AB1}) \times \frac{1}{2} \end{cases}$$



b) Considérons un segment $[AB]$, quelle opération est faite sur ce segment entre l'étape n et $n+1$ du flocon de von Koch ?

c) Compléter la fonction suivante et tracer un flocon complet.

```
import matplotlib.pyplot as plt
from numpy import sqrt
def segment(A : tuple, B : tuple) -> None : # trace le segment [AB]
   plt.plot( [ A[0], B[0] ], [ A[1], B[1] ], linewidth=1, color='k' )
def flocon(n : int, A : tuple = (0,0), B : tuple = (1,0)) -> None :
   if n == ... :
      ...
   else :
      AB1 = ...
      AB3 = ...
      AB2 = ...
      flocon(..., A, AB1)
      ... # plusieurs lignes
   plt.show()
```

Exo 16 : On considère la fonction suivante (dûe à Mc Carthy et généralisée par D. Knuth),

```
def f(n : int) -> int :
   if n > 100 :
      return n - 10
   else :
      return f( f( n+11 ) )
```

a) calculer `f(101)` en précisant l'état de la pile d'exécution : $\boxed{f \mid n = 101 \mid \to \dots}$ pile d'exécution $\to \boxed{\dots}$

b) calculer `f(100)` en précisant la pile d'exécution :

$\boxed{f \mid n = 101 \mid \to \dots}$ pile d'exécution $\to$ $\begin{array}{|c|c|l|}\hline f & n = 111 & \to \dots \\ \hline f & n = 101 & \to f(f(100+11)) \\ \hline \end{array}$ pile d'exécution $\to \dots$

c) de même calculer `f(99)` et `f(98)` en précisant la pile d'exécution.

d) Quelle conjecture peut-on faire ? Pourquoi s'appelle-t-elle la fonction91 de Mc Carthy ?

e) Calculer `f(102)` ; `f(103)` et corriger votre conjecture.

f) Implémenter la fonction f et compléter la fonction récursive `verif(n : int) -> bool`, de sorte que `verif(90)` vérifie tous les résultats de $f(n)$ pour $90 \leqslant n \leqslant 101$.

```
def verif(n : int) -> bool :
   print( n ) # pour connaître les valeurs vérifiées
   if n == ... :
      return f(101) == 91
   else :
      return f(n) == ... and verif( ... )
```