TP d'informatique n°9
(Python Maths)

Ecriture de fonctions autour de la suite de Syracuse

But du TP
- Ecrire les premières structures conditionnelles et les premières instructions itératives
- Consolider les connaissances sur les structures conditionnelles et itératives.
- S'exercer sur la syntaxe des fonctions
- Réaliser des travaux sur les listes.

Vous penserez **à sauvegarder régulièrement** votre travail dans un fichier dans votre espace personnel. Pour le nom de fichier, n'utilisez ni accent, ni espace, ni caractères spéciaux (autres que – et _) : en effet, certains projets nécessitent l'ouverture de fichiers que l'on aura écrits au préalable : or cet appel peut s'avérer délicat si le nom de ce fichier possède des caractères "non habituels"

Dans ce TP, on travaille avec la suite de Collatz, ou suite de Syracuse.
Cette suite est définie par :

On part d'un nombre entier naturel n non nul ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur.

Par exemple, en partant de n = 7, les 10 premiers termes de la suite sont :
7, 22, 11, 34, 17, 52, 26, 13, 40, 20

1. Ecrire une séquence d'instructions qui, d'abord demande à l'utilisateur un entier n naturel, et affiche les 20 premiers termes de la suite de Syracuse de premier terme n.

On utilisera l'algorithme suivant :

```
n ← entré par l'utilisateur
pour k allant de 1 à 20 faire          Rem : k ne joue ici qu'un rôle de compteur
    si 2 divise n alors
        n ← quotient de n par 2
    sinon
        n ← 3 × n + 1
    afficher (n)
```

Tester cette séquence avec plusieurs valeurs de n inférieur à 1000.

On constate rapidement que si la suite atteint la valeur 1, la suite devient périodique à partir de ce rang. La conjecture de Syracuse énonce que, pour toute valeur $n \in \mathbb{N}^*$, la suite atteint toujours la valeur 1.

2. Ecrire une fonction **syr** prenant comme argument un entier naturel n et qui retourne l'entier n/2 si n est pair et l'entier 3n +1 si n est impair.

3. Ecrire une fonction **suite_syracuse** prenant comme arguments un entier naturel p et un entier naturel u0 et qui retourne le terme de rang p de la suite de Syracuse débutant à u0.
Afficher le 100^{e} terme de toutes les suites de toutes les suites de Syracuse démarrant avec un u0 compris entre 1 et 100. Même chose avec le 1000^{e} terme.

Remarque : pour rendre plus lisible les affichages, on peut utiliser une "méthode" s'appliquant aux chaines de caractères et permettant d'écrire cette chaine sur une longueur de k espaces en la centrant (sous la forme .center(k)) et à demander à ne passer à la ligne dans l'affichage que tous les 10 termes par exemple (option end = " " lorsque u n'est pas multiple de 10).
Cela donne la séquence d'instructions :

```
for u in range(1,101):
        affiche = str(suite_syracuse(100,u)).center(6)
        if u%10 == 0:
                print(affiche)
        else:
                print(affiche, end = " ")
```

4. Ecrire une fonction **syracuse_atteint_1** prenant comme argument un entier naturel u0 et un entier p et qui retourne True si la suite de Syracuse débutant à u0 atteint 1 avant le p-ième terme et qui retourne False sinon.
Déterminer le plus petit entier u0 pour lequel aucun des 200 premiers termes de la suite de Syracuse débutant à u0 n'est égal à 1.
Montrer, avec Python, que toutes les suites de Syracuse démarrant avec un u0 compris entre 1 et 1000, finissent par atteindre la valeur 1.

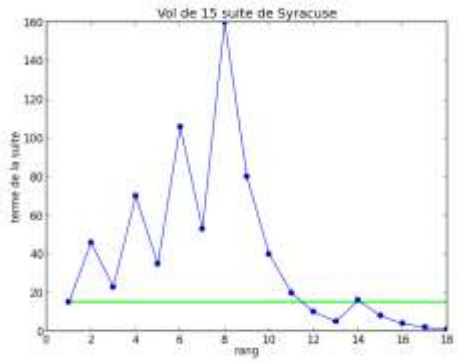
5. Soit u0 un entier naturel. On appelle vol de u0, la liste des termes successifs de la suite de Syracuse démarrant à u0 jusqu'à ce que l'on tombe sur un 1. **E**crire une fonction **vol** prenant comme argument un entier naturel u0 et qui retourne l'orbite de u0.

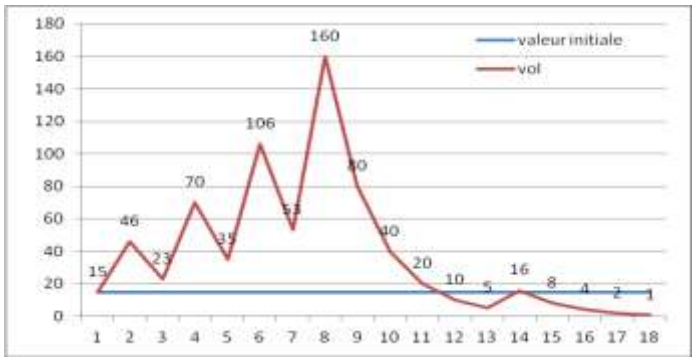
On utilisera, à l'intérieur du processus "def" pour définira la fonction vol, l'algorithme suivant :

```
n ← u0
Liste ← [n]
tant que n ≠ 1 faire                    Rem : on "espère" que la boucle s'arrête bien…
        n ← syr(n)
        Liste ← Liste + [n]             ici le + correspond à la concaténation
Retourner : Liste
```

Donner les orbites (ou vols…) de 27, 73 et 97.

Dans une question subsidiaire en fin de TP, si on a le temps, on pourra tracer le vol d'une suite pour un nombre donné (on peut aussi le faire avec OpenOffice…)





Pour un vol donné, il est possible de mesurer un certain nombre de caractéristiques :
- la plus grande valeur atteinte durant le vol : ce que l'on appelle l'*altitude*
- la longueur de la suite obtenue : *durée de vol*
- le nombre d'étapes avant de passer strictement en-dessous du nombre de départ : *durée de vol en altitude*
- le nombre d'étapes minimum après lequel on ne repasse plus au-dessus de la valeur initiale : *durée de vol avant la chute*

Par exemple, les caractéristiques du vol 15, (vol dont le graphe est dans la figure précédente) sont :

Caractéristiques du vol 15	Valeur
Altitude	160
Durée de vol	18
Durée de vol en altitude	11
Durée de vol avant la chute	14

6. Ecrire une fonction **altitude** recevant comme argument un entier naturel u0 et qui détermine l'altitude atteint par le vol u0.
Ecrire une fonction **altitudemax** recevant comme arguments deux entiers naturels n et m, avec n < m, et qui retourne une liste de 2 entiers : le premier est l'entier u0 compris entre n et m pour lequel le vol atteint l'altitude maximale, et le second est la valeur de cette altitude maximale.
Déterminer, pour chaque intervalle [p * 1000 + 1, (p +1) * 1000] pour p entre 0 et 9, l'entier initiant le vol de plus haute altitude

7. Ecrire une fonction **rechlongvol** recevant comme arguments deux entiers naturels n et m, avec n < m, et qui retourne une liste de 3 termes : le premier est l'entier u0 compris entre n et m pour lequel le vol est le plus long, le second est la durée de ce vol et le troisième terme est ce vol.
Déterminer, pour chaque intervalle [p * 1000 + 1, (p +1) * 1000] pour p entre 0 et 9, l'entier initiant le vol de plus longue durée

8. Ecrire une fonction **dureevolaltitude** recevant comme argument un entier naturel u0 et qui détermine la durée de vol en altitude du vol u0.

Déterminer les entiers entre 1 et 1000 initiant les 2 plus longues durées de vol en altitude.

9. Ecrire une fonction **dureevolchute** recevant comme argument un entier naturel u0 et qui détermine la durée de vol avant la chute du vol u0.
Déterminer les entiers entre 1 et 1000 initiant les 2 plus longues durées de vol avant la chute.

10. Ecrire une fonction **dessinvol** recevant comme argument un entier naturel u0 et qui dessine le vol u0. On s'inspirera du fichier contenant la fonction esca vue pour le TP sur les suites récurrentes.

TP d'informatique n°9 (Python Maths)

1. Ecrire une séquence d'instructions qui, d'abord demande à l'utilisateur un entier n naturel, et affiche les 20 premiers termes de la suite de Syracuse de premier terme n.

On utilisera l'algorithme suivant :

```
n = int (input("Donnez le premier terme de la suite "))
for k in range(20):
    if n % 2 == 0 :
        n = n // 2
    else :
        n = 3*n + 1
    print(n)
```

Tester cette séquence avec plusieurs valeurs de n inférieur à 1000.

On constate rapidement que si la suite atteint la valeur 1, la suite devient périodique à partir de ce rang. La conjecture de Syracuse énonce que, pour toute valeur $n \in \mathbb{N}^*$, la suite atteint toujours la valeur 1.

2. Ecrire une fonction **syr** prenant comme argument un entier naturel n et qui retourne l'entier n/2 si n est pair et l'entier 3n +1 si n est impair.

```
def syr(n) :
    if n % 2 == 0:
        return(n//2)
    else :
        return(3*n+1)
```

3. Ecrire une fonction **suite_syracuse** prenant comme arguments un entier naturel p et un entier naturel u0 et qui retourne le terme de rang p de la suite de Syracuse débutant à u0.

```
def = suite_syracuse(p, u0):
    a = u0
    for k in range(p) :
        a = syr(a)
    return(a)
```

Afficher le 100e terme de toutes les suites de toutes les suites de Syracuse démarrant avec un u0 compris entre 1 et 100. Même chose avec le 1000e terme.

Voici l'affichage (compact) des 100 premiers termes de rang 100

4	1	1	2	2	2	1	4	1	4	2	4	4	2	2	1	4	2	2	1
1	4	4	1	2	1	53	4	4	4	10	2	2	1	1	4	4	4	1	2
80	2	2	1	1	1	16	2	4	4	4	2	2	106	106	1	2	1	2	1
1	20	20	4	4	4	4	2	2	2	4	1	23	1	2	1	1	2	2	4
1	160	160	4	4	4	4	2	4	2	2	2	2	5	5	4	184	1	1	1

Remarque : pour rendre plus lisible les affichages, on peut utiliser la séquence d'instructions :

```
for u in range(1,101):
    affiche = str(suite_syracuse(100,u)).center(6)
    if u%20 == 0 :    print(affiche)
    else :            print(affiche, end = " ")
```

4. Ecrire une fonction **syracuse_atteint_1** prenant comme argument un entier naturel u0 et un entier p et qui retourne True si la suite de Syracuse débutant à u0 atteint 1 avant le p-ième terme et qui retourne False sinon.

```
def = syracuse_atteint_1(u0,p):
    a , k = u0 , 1
    while  k < p  :
        a , k = syr(a) , k + 1
            if  a == 1 :
                return (True)
    return(False)
```

Déterminer le plus petit entier u0 pour lequel aucun des 200 premiers termes de la suite de Syracuse débutant à u0 n'est égal à 1. **On trouve 2463 pour lequel le 200° terme est 40**
Montrer, avec Python, que toutes les suites de Syracuse démarrant avec un u0 compris entre 1 et 1000, finissent par atteindre la valeur 1. **Puisque pour tous les entiers < 2463, le 200° terme de la suite est 1, 2 ou 4, on sait que toutes les suites démarrant avec u0 <1000 atteignent 1.**

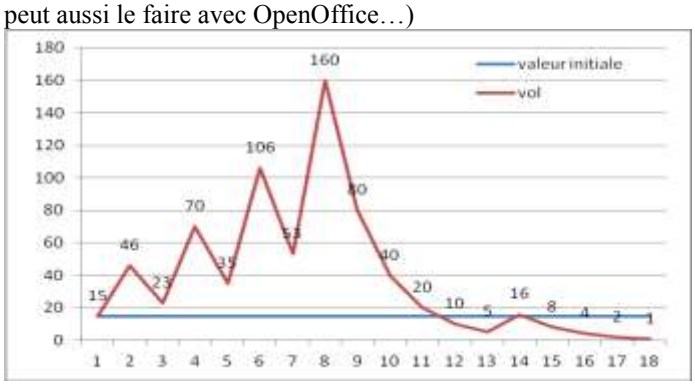
5. Soit u0 un entier naturel. On appelle vol de u0, la liste des termes successifs de la suite de Syracuse démarrant à u0 jusqu'à ce que l'on tombe sur un 1. **E**crire une fonction **vol** prenant comme argument un entier naturel u0 et qui retourne l'orbite de u0.

On utilisera, à l'intérieur du processus "def" pour définira la fonction vol, l'algorithme suivant :

```
n = u0
Liste = [n]
while  n != 1 :
        n  =  syr(n)
        Liste = Liste + [n]
return( Liste)
```

Les longueurs des vol 27, 73 et 97 sont 112, 116, 119

Dans une question subsidiaire en fin de TP, si on a le temps, on pourra tracer le vol d'une suite pour un nombre donné (on peut aussi le faire avec OpenOffice…)



Pour un vol donné, il est possible de mesurer un certain nombre de caractéristiques :
- la plus grande valeur atteinte durant le vol : ce que l'on appelle l'*altitude*
- la longueur de la suite obtenue : *durée de vol*
- le nombre d'étapes avant de passer strictement en-dessous du nombre de départ : *durée de vol en altitude*
- le nombre d'étapes minimum après lequel on ne repasse plus au-dessus de la valeur initiale : *durée de vol avant la chute*

Par exemple, les caractéristiques du vol 15, (vol dont le graphe est dans la figure précédente) sont :

Caractéristiques du vol 15	Valeur
Altitude	160
Durée de vol	18
Durée de vol en altitude	11
Durée de vol avant la chute	14

6. Ecrire une fonction **altitude** recevant comme argument un entier naturel u0 et qui détermine l'altitude atteint par le vol u0.
Une version simple est d'écrire : altitude = lambda n: max(vol(n))
On peut aussi transformer la fonction vol en remplaçant "Liste" par un compteur
Ecrire une fonction **altitudemax** recevant comme arguments deux entiers naturels n et m, avec n < m, et qui retourne une liste de 2 entiers : le premier est l'entier u0 compris entre n et m pour lequel le vol atteint l'altitude maximale, et le second est la valeur de cette altitude maximale.

```
def altitudemax(n,m):
        leader = n
        for k in range(n,m+1):
                if altitude(k) > altitude(leader):
                        leader = k
        return(leader, altitude(leader), vol(leader))
```

Déterminer, pour chaque intervalle [p * 1000 + 1, (p +1) * 1000] pour p entre 0 et 9, l'entier initiant le vol de plus haute altitude **On trouve 703, 1819, 2047, 3071, 4591, 5673, 6121, 7935, 8161, 9663, qui atteignent des altitudes respectives de 250 504, 1 276 936, 1 276 936, 1 276 936, 8 153 620, 6 810 136, 8 153 620, 2 316 760, 8 153 620, 27 114 424.**

7. Ecrire une fonction **rechlongvol** recevant comme arguments deux entiers naturels n et m, avec n < m, et qui retourne une liste de 3 termes : le premier est l'entier u0 compris entre n et m pour lequel le vol est le plus long, le second est la durée de ce vol et le troisième terme est ce vol.

```
def rechlongvol(n,m):
    leader = n
    for k in range(n,m+1):
        if len(vol(k)) > len(vol((leader)):
            leader = k
    return(leader, len(vol((leader)), """vol(leader)""")
```

Déterminer, pour chaque intervalle [p * 1000 + 1, (p +1) * 1000] pour p entre 0 et 9, l'entier initiant le vol de plus longue durée **On trouve 871, 1161, 2919, 3711, 4379, 5567, 6171, 7963, 8959, 9257, qui atteignent de durées de vol respectives de 179, 182, 217, 238, 215, 236, 262, 252, 247, 260**

8. Ecrire une fonction **dureevolaltitude** recevant comme argument un entier naturel u0 et qui détermine la durée de vol en altitude du vol u0.

```
def dureevolaltitude(u0):
    n = u0
    compteur = 0
    while n >= u0 :
        n = syr(n)
        compteur = compteur + 1
    return(compteur)
```

Déterminer les entiers entre 1 et 1000 initiant les 2 plus longues durées de vol en altitude.
On trouve 703 et 27 qui ont des durées de vol en altitude respectives de 132 et 96

9. Ecrire une fonction **dureevolchute** recevant comme argument un entier naturel u0 et qui détermine la durée de vol avant la chute du vol u0.

```
def dureevolchute(u0):
    Liste = vol(u0)
    n = len(Liste)
    k = 0
    x = Liste[n - 1 - k]
    while x < u0 :
        k = k + 1
        x = Liste[n - 1 - k]
    return(n-k)
```

Déterminer les entiers entre 1 et 1000 initiant les 2 plus longues durées de vol avant la chute.
On trouve 871 et 937 qui ont des durées de vol en altitude respectives de 156 et 135

10. Ecrire une fonction **dessinvol** recevant comme argument un entier naturel u0 et qui dessine le vol u0. On s'inspirera du fichier contenant la fonction esca vue pour le TP sur les suites récurrentes

```
import matplotlib.pyplot as plt

def dessinvol(u0):
    Y = vol(u0)
    n = len(vol(u0))
    X = [0]*n
    for k in range(n):
        X[k] = k+1
    plt.plot(X,[0]*n,'k-')
    plt.plot(X,[u0]*n,'g-')
    plt.plot(X, Y, 'bo-')
    plt.show()
    plt.close()

dessinvol(15)
```