

TP d'informatique n°6

(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 :

$n \leftarrow$ *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
         $n \leftarrow$  quotient de  $n$  par 2
    sinon
         $n \leftarrow 3 \times n + 1$ 
    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 **sy** 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 u_0 et qui retourne le terme de rang p de la suite de Syracuse débutant à u_0 . Afficher le 100^e terme de toutes les suites de Syracuse démarrant avec un u_0 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 chaînes de caractères et permettant d'écrire cette chaîne 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 u_0 et un entier p et qui retourne True si la suite de Syracuse débutant à u_0 atteint 1 avant le p -ième terme et qui retourne False sinon.

Déterminer le plus petit entier u_0 pour lequel aucun des 200 premiers termes de la suite de Syracuse débutant à u_0 n'est égal à 1.

Montrer, avec Python, que toutes les suites de Syracuse démarrant avec un u_0 compris entre 1 et 1000, finissent par atteindre la valeur 1.

5. Soit u_0 un entier naturel. On appelle vol de u_0 , la liste des termes successifs de la suite de Syracuse démarrant à u_0 jusqu'à ce que l'on tombe sur un 1. Ecrire une fonction `vol` prenant comme argument un entier naturel u_0 et qui retourne l'orbite de u_0 .

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

$n \leftarrow u_0$

Liste $\leftarrow [n]$

tant que $n \neq 1$ **faire**

$n \leftarrow \text{syr}(n)$

 Liste \leftarrow Liste + $[n]$

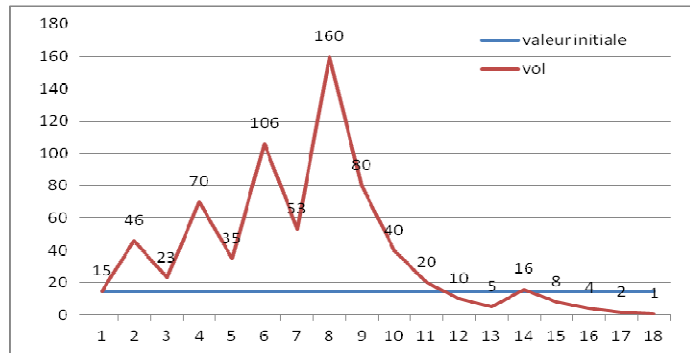
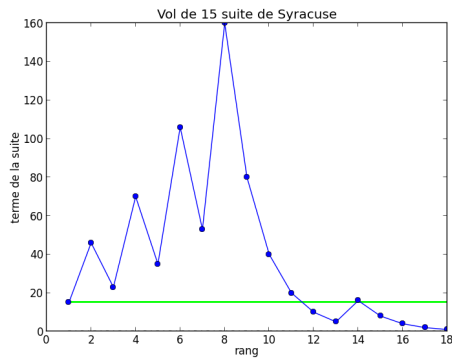
Rem : on "espère" que la boucle s'arrête bien...

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 u_0 et qui détermine l'altitude atteinte par le vol u_0 .

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 u_0 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 u_0 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 u_0 et qui détermine la durée de vol en altitude du vol u_0 .

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 u_0 et qui détermine la durée de vol avant la chute du vol u_0 .

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 u_0 et qui dessine le vol u_0 . On s'inspirera du fichier contenant la fonction `esca` vue pour le TP sur les suites récurrentes.