Initiation interactive à Python 4 : les boucles

Aller vers : Initiation interactive à Python 3 : Les tests

Indéniablement, le gros avantage des ordinateurs, c'est qu'ils obéissent sans se fatiguer. Qu'on souhaite refaire 100, 1000 ou 1 million de fois la même chose... il n'y a qu'à demander. Tous les langages de programmation possèdent un moyen pour réaliser des opérations répétitives : on appelle ça des boucles. Python possède plusieurs types de boucles. Les boucles `while` et les boucles `for`. La boucle `while` est la plus *simple* et elle permet de répéter une séquence d'instructions tant qu'une condition particulière est remplie.

Boucle while

Voici un premier exemple de boucle `while`. Le programme permet d'afficher tous les multiples de 3 sans toutefois atteindre la valeur 100 :

Le programme précédent se lit :

- initialiser n à 0
- répéter tant que n est strictement plus petit que 100 :
  - afficher n
  - ajouter 3 à n

Attention à la ligne `n = n + 3`. Elle ne signifie sûrement pas que n est égal à n plus 3 (ce serait difficile). Le = en Python (et dans beaucoup de langages) n'a pas de rapport avec l'égalité en mathématiques. C'est le symbole d'*affectation* qui permet, en Python, de donner un nom à un objet. Ici, si n vaut par exemple 6, on commence par calculer `n + 3`, qui vaudra alors 9, puis on *nomme* ce nouvel objet n. La variable n qui désignait 6 désigne maintenant 9.

Modifiez le programme qui précède pour réaliser ce qui suit :

1. affichez tous les multiples de 2 plutôt que les multiples de 3, toujours en vous arrêtant vers 100.
2. maintenant que vous savez afficher tous les nombres pairs, que faut-il modifier pour afficher tous les nombres impairs ?
3. si plutôt que de calculer et stocker `n + 2` à chaque tour de boucle, vous calculez `n * 2`, vous devez pouvoir obtenir les nombres utilisés dans le jeu 2048. Mais pour les obtenir tous, il faudra aussi change le *test* du `while` pour ne pas vous arrêter à 100.

Maintenant que nous savons faire des boucles, nous pouvons calculer la suite de Collatz issue de 27 sans nous fatiguer, nous allons utiliser une boucle `while` indiquant : Tant que n est différent de 1, refaire le bloc d'instructions qui consiste à calculer le nombre suivant :

Notez que le symbole différent s'écrit `!=`. Notez bien l'indentation du code : le bloc dans le `while` contient 5 lignes, dont `print(n)` car on veut afficher tous les nombres de la suite. Si vous décalez ce `print(n)` pour l'aligner avec la marge de gauche, cette instruction ne sera plus **dans** la boucle, mais **après** la boucle, et ça changera tout. Essayez...

La suite obtenue est assez longue... pensez à utiliser l'ascenseur à droite de la zone d'exécution pour voir toutes les valeurs.

Boucle for

Naturellement, il est possible de réaliser le même travail de façon différente. Nous pourrions vouloir stocker les valeurs obtenues (dans une objet qu'on appelle en Python une liste), compter la longueur de la suite, rechercher le maximum de la suite etc... Toutes ces questions sont faciles à résoudre avec un programme de quelques lignes, mais nécessitent un peu plus de connaissances que les tests et les boucles `while`. En particulier, la boucle `for` peut parfois simplifier un peu l'écriture. La boucle `for` de Python peut faire *beaucoup* de choses, mais elle peut aussi *simplement* compter d'une valeur à une autre. Pour afficher tous les nombres de 10 à 20 inclus, nous pouvons écrire :

Mais il est souvent plus lisible d'écrire :

Le code précédent se lit : pour n variant de 10 à 21 (non inclus), afficher n.

Le fait de devoir écrire 21 pour aller jusqu'à 20 peut dérouter... cette particularité ne vient pas de la boucle `for` mais de la fonction `range(mini, maxi)` qui «contient» en quelque sorte tous les entiers de la valeur `mini` incluse à la valeur `maxi` **non incluse**. Cette particularité est dûe à la manipulation très courante des listes, dont nous n'avons pas encore parlé.

Le programme suivant affiche la table de 3 :

Notez que nous avons écrit `t = 3` en début de code, pour ensuite utiliser `t` à deux endroits différents. Si maintenant nous voulons afficher la table de 5, nous n'avons **qu'une seule expression** à modifier en début de code, nous n'avons pas besoin de tout relire et de remplacer partout les 3 par de 5. C'est donc une bonne idée d'avoir créé cette variable `t` en début de programme.

Modifiez le programme précédent pour qu'il affiche effectivement la table de 5.

Puis modifiez le à nouveau pour afficher **toutes les tables** de 1 à 9... Attention, pour faire ceci, il vous faudra 2 boucles `for`, l'une dans l'autre...

Si vous souhaitez un document un peu plus complet, auquel vous reporter pour savoir ce que sont : les fonctions, les listes, les itérateurs... jetez un oeil à ce document : Introduction à l'algorithmique avec Python. Naturellement, il existe beaucoup d'autres documents du même genre sur Internet.

Aller vers Initiation interactive à Python 5 : Listes