Initiation interactive à Python 3 : Les tests

Aller vers Initiation interactive à Python 2 : Calculatrice, variables, IO

L'exécution d'un programme n'est pas toujours aussi linéaire que ce que nous venons de voir. Il faut parfois varier les instructions à exécuter selon que certaines conditions sont remplies ou non. Dans un programme, *un test* permet d'exécuter une séquence d'instructions ou une autre, selon le résultat d'une comparaison par exemple. Python permet de réaliser des tests avec l'instruction `if`.

Addition ou multiplication ?

Nous allons illustrer les tests avec cet exemple : le programme, après vous avoir demandé d'entrer deux nombres, vous proposera de les multiplier ou de les ajouter. Pour choisir l'opération, vous devrez entrer * ou + et valider.

Attention à l'indentation !!! Après `if`, l'interpréteur Python attend un bloc : le bloc à exécuter si le test est vrai. Ce bloc est matérialisé par un décalage vers la droite et ne comporte qu'une seule ligne : `c = a * b`. De même, après `else`, on indique par un bloc ce qu'il faudra exécuter si le test est faux (une seule ligne : `c = a + b`. La dernière ligne `print…` n'est pas décalée car elle est *en dehors* des blocs et sera exécutée après l'un ou l'autre des blocs du test.

Par convention, lorsqu'on indente un bloc, on le décale de 4 espaces.

Essayez ce programme. Que se passe-t-il si, plutôt que d'entrer * ou +, vous entrez autre chose ? Essayez par exemple d'entrer `-`. Voyez vous un moyen de corriger ce défaut ?

Triangles

Nous allons illustrer les tests avec cet exemple : étant données 3 nombres, par exemple 2, 4 et 5, est-il possible de dessiner un triangle dont les côtés ont pour longueur ces nombres ? Une propriété peut nous aider à répondre : si la somme des 3 longueurs est inférieure au double de la plus grande des trois longueurs, alors c'est possible. Sinon, ça ne l'est pas [1)].

Nous allons tout de suite éliminer une des difficultés en utilisant la fonction `max`, qui permet de trouver le plus grand élément parmi plusieurs. Par exemple : `max([4,7,2])` vaudra 7. Notez les crochets, nécessaires ici car on communique en à `max` une **liste** de nombres.

Voici un programme qui réalise ce que nous souhaitons :

Vous pouvez tester ce programme pour les longueurs:

- (1,2,10) : impossible de construire un triangle
- (3,4,5) : construction possible

Dans le cas où la construction est possible (et uniquement dans ce cas), nous aimerions aussi savoir si le triangle est rectangle. Un triangle est rectangle si la somme des carrés des deux petits côtés est égale au carré du plus grand côté. Nous pouvons aussi dire que la somme des carrés de **tous** les côtés est égale au double du carré du plus grand côté.

Par exemple, le triangle de côté 3, 4, 5 est rectangle car la somme des carrés de ses côtés (3*3+4*4+5*5 = 50) est égale au double du carré du plus grand côté (2 * 5*5 = 50)[2].

Il est possible d'écrire un test **à l'intérieur** du bloc d'un autre test. L'idée est de traduire ceci:

- Si le triangle peut être construit:
  - Afficher : “Il est possible de construire le triangle”
  - Si le triangle est rectangle:
    - Afficher : “Le triangle est rectangle”
- Sinon
  - Afficher : “Il est impossible de construire le triangle”

Modifiez le programme qui précède pour qu'il affiche d'une part si la construction est possible, et aussi si le triangle est rectangle.

Suite de Collatz

Voici un autre exemple : la suite de Collatz (qui possède beaucoup d'autres noms) se construit à partir d'un nombre de départ. Si ce nombre est impair, on obtient le nombre suivant en multipliant par 3 et en ajoutant 1. Si le nombre est pair, on obtient le nombre suivant en divisant par 2. Si cette suite est très connue c'est parce qu'il semble que, *quel que soit le nombre de départ*, la suite finit toujours par *atteindre* le nombre 1, ce qui n'a rien d'évident et n'a toujours pas été démontré (alors que le problème a l'air tout simple).

Avec un peu d'arithmétique, des tests (et bientôt des boucles… patience…), nous pouvons expérimenter avec la suite de Collatz.

Testez le programme qui précède. Il vous demande un nombre, puis teste s'il est pair ou non. Pour cela, on calcule le reste de la division entière du nombre par 2 (opération `%`). Si le reste de cette division est nul (notez que pour tester l'égalité de deux nombres, on emploie ==), c'est que le nombre est pair. Dans ce cas, on le divise par 2 (division entière `//`) et on appelle n le résultat (ce qui *efface* la valeur de n initiale). Sinon (mot clé `else`), on multiplie le nombre par 3 et on ajoute 1, puis on appelle ce résultat n. Finalement n est affiché.

Vous pouvez essayer pour quelques valeurs. Si vous entrez 3 (qui est impair), vous obtiendrez 10 (3*3+1). Si vous entrez 10, vous obtiendrez 5 (10/2) et ainsi de suite… Essayez de calculer la suite jusqu'à atteindre le nombre 1 :

3 ⇒ 10 ⇒ 5 ⇒ ???

Une fois le nombre 1 atteint, vous pouvez continuer d'écrire la suite pour voir ce qui se passe.

Maintenant, essayez à nouveau en partant de 27 :

27 ⇒ 82 ⇒ 41 ⇒ 124 ⇒ 62 ⇒ 31 ⇒ ???

C'est un peu long... et fastidieux. Écrire un programme sert justement à ne pas devoir faire des tâches répétitives, mais à demander à l'ordinateur de les faire pour nous...

Lire la suite : Aller vers Initiation interactive à Python 4 : les boucles

1)
le cas extrême où la somme des trois longueurs est exactement le double de la plus grande des 3 correspond à un triangle plat... pas très intéressant
2)
avec ce que nous connaissons de Python pour le moment, le plus simple est de faire comme ça