Les boucles inconditionnelles

Spé NSI - Lycée du parc

Année 2020-2021

Introduction

Pour répéter la même opération un certain nombre de fois, on dispose des boucles inconditionnelles. L'objectif de ce chapitre est d'apprendre à s'en servir de manière pertinente.

I Répétition simple d'une ou plusieurs instructions

L'instruction `print()` (avec un agument vide) provoque le passage à la ligne dans l'affichage. Si on veut provoquer trois passages à la ligne on peut écrire :

```
1 print()
2 print()
3 print()
```
1

Lorsqu'on doit répéter une ou plusieurs instructions peut-on toujours procéder de cette manière ?

Lorsque l'on veut répéter un certain nombre de fois une (ou plusieurs) instruction(s) on utilise la boucle inconditionnelle `for` :

```
1 for _ in range(3): print()
```
2

On indique simplement dans l'argument de la fonction `range()` le nombre de répétitions à effectuer (une répétition s'appelle un *tour* ou plus précisément un *tour de boucle*) et l'instruction a répéter se place derrière le caractère « : » (c'est le *corps de la boucle*).

Exercice 1

Écrire un programme qui demande d'abord à l'utilisateur un texte quelconque puis un nombre et qui affiche le texte autant de fois que le nombre choisi.

Exercice 2 Dans la console essayer les instructions suivantes :

```
for _ in range(2.4): print('toto')
```
3

Puis

```
for _ in range(-2): print('toto')
```
4

et

```
for _ in range(4) print('toto')
```
5

Quelle conclusion en tirer ?

Par défaut, l'instruction `print` ajoute un retour à la ligne à la fin de l'affichage. On peut modifier ce comportement en précisant le caractère de fin :

```
print('N', end = '')
print('SI')
```
6

donne :

```
NSI
>>>
```

```
print('N')
print('SI')
```
7

donne :

```
N
SI
>>>
```

Exercice 3

Écrire un programme qui demande à l'utilisateurs un nombre de tirets et un message et qui produit ensuite un affichage du type (ici avec 6 tirets) :

```
------mon_message------
```

Lorsqu'on veux répéter plusieurs fois une séquence d'instructions, on les regroupe dans un bloc. Pour cela il suffit de les décaler de quatre espaces vers la droite à partir de la marge :

en Python c'est l'indentation qui structure les blocs

La syntaxe est :

```
for _ in range(nb_repetitions):
    instruction     #
    instruction     # bloc d'instructions
    instruction     #
```
8

Exercice 4

On veux faire tracer 20 lignes de manière aléatoire à la tortue du module `turtle`. Déterminer l'erreur dans le code suivant :

```
1 from turtle import *
2 from random import randint
3
4 for _ in range(20):
5     x = randint(-100, 100)
6     y = randint(-100, 100)
7 goto(x, y)
```

9

Rectifier et tester

Entraînement 1

Écrire un programme qui demande à l'utilisateurs un nombre de tirets et un message et qui produit ensuite un affichage du type :

```
--------------------------
--------mon_message--------
--------------------------
```

Indication : pour obtenir le nombre de caractères d'une chaîne, on utilise la fonction `len` :

```
1 >>> texte = 'mon_message'
2 >>> len(texte)
3 11
```

10

II Répétition d'instructions avec différentiation

Jusqu'à maintenant, les instructions que l'on a répétées étaient strictement identiques mais il très souvent nécessaire de modifier certains éléments au fur et à mesure de l'exécution des tours de la boucle. On peut par exemple utiliser un compteur :

```
1 compteur = 0
2 for _ in range(10):
3     print(compteur)
4     compteur = compteur + 1
```

11

Que va produire ce code ?

Tester pour vérifier.

Exercice 5

Écrire un programme qui demande un nombre à l'utilisateur et fait afficher la table de multiplication correspondante sous la forme :

```
multiplicateur : 4
1 x 4  =  4
2 x 4  =  8
.
.
10 x 4  =  40
```
12

Syntaxe générale de `range`

On a jusqu'à maintenant utilisé la fonction `range` de manière naïve. En fait, la fonction `range` produit un *itérateur* que la boucle `for` ne fait que parcourir. La fonction **range** possède trois arguments dont deux sont optionnels :

- **range**(n) retourne un itérateur parcourant les entiers consécutifs entre 0 et n exclu.
- **range**(m, n) retourne un itérateur parcourant les entiers consécutifs entre m compris et n exclu.
- **range**(m, n, s) retourne un itérateur parcourant les entiers consécutifs entre m compris et n exclu avec un pas de s.

On peut avoir accès à la valeur produite par l'itérateur en précisant une variable lors de la mise en place de la boucle (on parle alors d'indice de boucle). Que produisent les codes suivants :

```
for k in range(10):
    print(k)
```
13

```
for k in range(2, 8):
    print(k)
```
14

```
for k in range(1, 10, 2):
    print(k)
```
15

```
for k in range(10, 0, -1):
    print(k)
```
16

III Utilisation d'un accumulateur

On souhaite additionner tous les nombres de 1 à 100. Compléter le code suivant

```
S = 0
for k in range(...., ......):
    S =
print('La somme des 100 premiers entiers est : ', S)
```
17

On peut suivre l'exécution d'une boucle à l'aide d'un tableau d'état :

Exercice 6

Quelle est la valeur de a après l'exécution du code ci-dessous ?
On utilisera le tableau d'état ci-contre.

```
1 a = 1
2 for k in range(4):
3     a = a + k
```

ligne	a	k

Entraînement 2 

Écrire un programme qui calcule la somme des carrés des 100 premiers entiers naturels, c'est à dire $1 + 2^2 + 3^2 + 4^2 + \cdots + 100^2$.

Entraînement 3 

Écrire un programme qui demande un entier n à l'utilisateur puis n notes et qui affiche ensuite la moyenne de ces notes.

Exercices

Exercice 7

Faire tracer une grille 10x10 par la tortue de `turtle`.

Exercice 8

Faire tracer une spiral par la tortue de `turtle`. Indication : on utilisera la fonction `circle`

Exercice 9

Écrire un programme qui demande un nombre n à l'utilisateur et qui calcule ensuite le produit des nombres de 1 à n.

Exercice 10

Écrire un programme qui demande un nombre n à l'utilisateur et faire tracer un escalier de n marches par la tortue de `turtle`.

Exercice 11

Écrire un programme qui demande des nombres n et a à l'utilisateur et faire tracer un polygone régulier de n côtés de longueur a par la tortue de `turtle`.