


Cours	Cours	
BTS SNIR	Python	
2 ^{ème} année	Les boucles	

Les boucles, tout comme les conditions, sont une structure de contrôle Python de base et il est donc essentiel de comprendre comment elles fonctionnent et de savoir les utiliser.

Présentation des boucles en Python

Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code, c'est-à-dire d'exécuter un code « en boucle » tant qu'une condition donnée est vérifiée.

Lorsqu'on code, on va en effet souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n'écrire le code qu'on doit exécuter plusieurs fois qu'une seule fois.

Nous allons ainsi pouvoir utiliser les boucles pour parcourir les valeurs d'une liste ou pour afficher une suite de nombres.

Nous avons accès à deux boucles en Python :

- La boucle `while` ("tant que...") ;
- La boucle `for` ("pour...").

Le fonctionnement général des boucles sera toujours le même : on pose une condition qui sera généralement liée à la valeur d'une variable et on exécute le code de la boucle « en boucle » tant que la condition est vérifiée.

Pour éviter de rester bloqué à l'infini dans une boucle, vous pouvez donc déjà noter qu'il faudra que la condition donnée soit fausse à un moment donné (pour pouvoir sortir de la boucle). Selon le type de condition, on va avoir différents moyens de faire cela. Nous allons voir les plus courants dans la suite de cette leçon.

La boucle Python `while`

La boucle `while` va nous permettre d'exécuter un certain bloc de code « tant qu'une » condition donnée est vérifiée. Sa syntaxe est la suivante :

```
[>>> x = 0
[>>> while x < 10:
[...     print(x)
[...     x += 1
[...
0
1
2
3
4
5
6
7
8
9
```

On commence ici par créer une variable `x` et on stocke la valeur 0 dedans.

On crée ensuite notre boucle `while` qui va baser sa condition de sortie autour de la valeur de la variable `x`.

Littéralement, cette boucle signifie “tant que `x` stocke une valeur strictement inférieure à 10, affiche la valeur de `x` puis ajoute 1 à cette valeur”.

Lors du premier tour dans la boucle, `x` stocke la valeur 0 qui est bien inférieure à 10. On rentre donc dans la boucle, on affiche la valeur de `x` et on ajoute 1 à cette valeur.

Une fois arrivé en fin de boucle, on retourne au début de la boucle. On teste à nouveau si `x` contient une valeur inférieure à 10. C’est le cas puisque `x` stocke désormais 1. On affiche à nouveau la valeur de `x` et on lui ajoute à nouveau 1.

On retourne à nouveau au début de la boucle et etc. Jusqu’à ce que la condition de sortie soit vérifiée, c’est-à-dire jusqu’à ce que `x` stocke une valeur supérieure ou égale à 10. Dans ce cas là, la boucle est ignorée et on passe à l’instruction suivante.

Note : Lorsqu’on ajoute 1 à une variable, on dit qu’on l’incrémente. À l’inverse, lorsqu’on enlève 1 à la valeur d’une variable, on dit qu’on la décrémente. Les opérations d’incrément et de décrémentation sont très fréquentes au sein des boucles. On s’en sert généralement pour que la condition d’exécution de la boucle soit fausse à un moment donné.

La boucle Python `for`

La boucle Python `for` possède une logique et une syntaxe différente de celles des boucles `for` généralement rencontrées dans d’autres langages.

En effet, la boucle `for` Python va nous permettre d’itérer sur les éléments d’une séquence (liste, chaîne de caractères, etc.) selon leur ordre dans la séquence.

La condition de sortie dans cette boucle va être implicite : on sortira de la boucle après avoir parcouru le dernier élément de la séquence.

La syntaxe de cette boucle va être la suivante :

```
>>> liste = [7, "Pierre", "trail", 29]
>>> for i in liste:
...     print(i)
...
7
Pierre
trail
29
```

La fonction range()

On va pouvoir utiliser la fonction `range()` pour itérer sur une suite de nombres avec une boucle `for`.

Cette fonction permet de générer une suite de valeurs à partir d'un certain nombre et jusqu'à un autre avec un certain pas ou intervalle.

Dans son utilisation la plus simple, nous allons nous contenter de passer un nombre en argument (entre les parenthèses) de `range()`. Dans ce cas, la fonction génèrera une suite de valeurs de 0 jusqu'à ce nombre - 1 avec un pas de 1. `range(5)` par exemple génère les valeurs 0, 1, 2, 3 et 4.

Si on précise deux nombres en arguments de cette fonction, le premier nombre servira de point de départ pour la génération de nombres tandis que le second servira de point d'arrivée (en étant exclus). `range(5, 10)` par exemple permet de générer les nombres 5, 6, 7, 8 et 9.

Finalement, on peut préciser un troisième et dernier nombre en argument de `range()` qui nous permet de préciser son pas, c'est-à-dire l'écart entre deux nombres générés. Ecrire `range(0, 10, 2)` par exemple permet de générer les nombres 0, 2, 4, 6 et 8.

On va pouvoir utiliser la fonction `range()` plutôt qu'une variable de type séquence avec nos boucles `for` pour itérer sur une suite de nombres.

```
>>> for n in range(5):
...     print(n)
...
0
1
2
3
4
>>>
>>> for n in range(5, 10):
...     print(n)
...
5
6
7
8
9
>>> for n in range(0, 10, 2):
...     print(n)
...
0
2
4
6
8
```

Les instructions break et continue

Les instructions `break` et `continue` sont deux instructions qu'on retrouve dans de nombreux langages et qui sont souvent utilisées avec les boucles mais qui peuvent être utilisées dans d'autres contextes.

L'instruction `break` permet de stopper l'exécution d'une boucle lorsqu'une certaine condition est vérifiée. On l'inclura souvent dans une condition de type `if`.

Par exemple, on va pouvoir stopper l'exécution d'une boucle lorsqu'une variable contient une valeur en particulier.

```
[>>> for n in range(9):
[...     if n % 2 == 0:
[...         print(n, " est un chiffre pair")
[...     else:
[...         print(n, " est un chiffre impair")
[...
0  est un chiffre pair
1  est un chiffre impair
2  est un chiffre pair
3  est un chiffre impair
4  est un chiffre pair
5  est un chiffre impair
6  est un chiffre pair
7  est un chiffre impair
8  est un chiffre pair
```

L'instruction `continue` permet d'ignorer l'itération actuelle de la boucle et de passer directement à l'itération suivante. Cette instruction va donc nous permettre d'ignorer toute ou partie de notre boucle dans certaines conditions et donc de personnaliser le comportement de notre boucle.

```
[>>> for n in range(9):
[...     if n % 2 == 0:
[...         continue
[...     else:
[...         print(n)
[...
1
3
5
7
```