PYTHON – Boucles

Introduction

Les boucles vont permettre de répéter une certaine opération autant de fois que nécessaire.

Deux types de situation peuvent se présenter :

- le nombre de répétitions n'est pas connu à l'avance, mais on connaît un test d'arrêt de la boucle :
    - Exemple : un jeu consiste à lancer un dé jusqu'à obtenir un six. Nous ne savons pas à l'avance combien de fois il faudra lancer le dé.
    - Pour ce genre de situation, les langages de programmation proposent la boucle **TANT QUE** :

```
TANT QUE ( je n'ai pas obtenu six ) :
    lancer le dé
FIN_TANT_QUE
```

- le nombre de répétitions est connu à l'avance : on utilise alors une boucle POUR :
    - Exemple : un jeu consiste à lancer 3 fois de suite un dé. Si l'on obtient au moins un six, on gagne; sinon, on perd. Nous savons à l'avance combien de fois il faudra lancer le dé (3 fois).
    - Pour ce genre de situation, les langages de programmation proposent la boucle **POUR** :

```
POUR i ALLANT_DE 1 A 3 :
    lancer le dé
FIN_POUR
```

Remarque : La variable *i* joue ici le rôle d'un **compteur** : elle compte les tours de boucles réalisés.

La boucle while

Syntaxe

En python, **la boucle TANT QUE s'écrit while**

La boucle while se retrouve dans la plupart des autres langages de programmation.

Elle permet de répéter un **bloc d'instructions** tant qu'une condition est vraie

Syntaxe

```
while (condition):
    # instruction 1
    # instruction 2
    # ...
    # instruction N
```

N'oubliez pas les deux points à la fin de la ligne while

Exemple :

```
a = 0
while (a < 7): # (n'oubliez pas le double point !)
        a = a + 1 # (n'oubliez pas l'indentation et l'incrémentation !)
        print(a)
```

Nous avons ainsi construit notre première boucle de programmation python, laquelle répète un certain nombre de fois le **bloc d'instructions indentées**.

Voici comment cela fonctionne :

- Avec l'instruction ***while***, Python commence par évaluer la validité de la condition fournie entre parenthèses (***a>7***).
- **Si la condition se révèle fausse (FALSE)**, alors tout le bloc qui suit est ignoré et l'exécution du programme se termine.
- **Si la condition est vraie (TRUE)**, alors Python exécute tout le bloc d'instructions constituant le corps de la boucle :
    - Instruction1 : l'instruction ***a = a + 1*** incrémente d'une unité le contenu de la variable **a** (ce qui signifie que l'on affecte à la variable **a** une nouvelle valeur, qui est égale à la valeur précédente augmentée d'une unité).
    - Instruction2 : l'appel de la fonction ***print()*** affiche la valeur courante de la variable **a**.
    - lorsque ces deux instructions ont été exécutées, nous avons assisté à une première itération, et le programme boucle, c'est-à-dire que l'exécution reprend à la ligne contenant l'instruction ***while***. La condition qui s'y trouve est à nouveau évaluée, et ainsi de suite.
      Dans notre exemple, si la condition ***a < 7*** est encore vraie, le corps de la boucle est exécuté une nouvelle fois et le bouclage se poursuit.

Résultat dans la console :

```
1
2
3
4
5
6
7
>>>
```

N'oubliez pas d'incrémenter la variable a, sinon, vous créez ce qu'on appelle une **boucle infinie.**

Si votre ordinateur se lance dans une boucle infinie à cause de votre programme, pour interrompre la boucle, vous devrez taper **CTRL + C** dans la fenêtre de l'interpréteur.

Exemple de boucle infinie :

```
a = 0
while (a < 7): # (n'oubliez pas le double point !)
        print(a)
```

Autre exemple :

```
a = 0
while (a < 4):
        a = a + 1
        print("a =",a)
print("Aurevoir")
```

```
a = 1
a = 2
a = 3
a = 4
Aurevoir
```

Instructions	**Valeur des variables**				
a = 0	0				
while (a < 4):	True	True	True	True	False
a = a + 1	1	2	3	4	
print("a =",a)	"a=1"	"a=2"	"a=3"	"a=4"	
print("Aurevoir")					"Aurevoir"

Mot-clé break

Le mot-clé break permet **d'interrompre une boucle**, quelle que soit la condition de la boucle.

```
while 1: # 1 est toujours vrai -> boucle infinie
   lettre = input("Tapez 'Q' pour quitter : ")
   if lettre == "Q":
      print("Fin de la boucle")
      break
```

1. La boucle while a pour condition 1, c'est-à-dire une condition qui sera *toujours* vraie. Autrement dit, c'est une boucle infinie.
2. On demande à l'utilisateur de taper une lettre (un 'Q' pour quitter). Tant que l'utilisateur ne saisit pas cette lettre, le programme lui redemande de taper une lettre.
3. Quand il tape 'Q', le programme affiche « Fin de la boucle » et **la boucle s'arrête grâce au mot-clé break**.
4. Python sort immédiatement de la boucle et exécute le code qui suit la boucle, s'il y en a.

Parfois, break est véritablement utile et fait gagner du temps. Mais **ne l'utilisez pas à outrance**, préférez une boucle avec une condition claire plutôt qu'un bloc d'instructions avec un break (par exemple tester directement si lettre != « Q »dans le while)

Déterminer les bonnes conditions d'une boucle while

Prenons le cas d'une saisie au clavier, où le programme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non).

Mais tôt ou tard, l'utilisateur risque de taper autre chose que la réponse attendue.

Alors, **on met en place un contrôle de saisie**, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues.

Solution n° 1 :

```
Rep = input("Voulez-vous un café? (O/N)") #on fait une 1ère lecture de Rep
while Rep != "O" and Rep != "N":
   Rep = input("Voulez-vous un café? (O/N)")
```

si l'on effectue une lecture préalable de Rep, la boucle qui suit sera exécutée uniquement dans l'hypothèse d'une mauvaise saisie initiale. **Si l'utilisateur saisit une valeur correcte à la première demande de Rep, le programme passera sur la boucle sans entrer dedans**.

Solution n° 2 :

```
Rep = "X"
while Rep != "O" and Rep != "N":
   Rep = input("Voulez-vous un café? (O/N)")
```

Une autre possibilité consiste à ne pas lire, mais à affecter arbitrairement la variable avant la boucle. Dans notre exemple, on peut affecter Rep avec n'importe quelle valeur, à part « O » et « N ». Cette affectation a pour résultat de **provoquer l'entrée obligatoire dans la boucle**, la boucle s'exécute au moins une fois.

Pour l'utilisateur, cette différence entre les 2 solutions est mineure ; mais pour le programmeur, il importe de bien comprendre que les cheminements des instructions ne seront pas les mêmes dans un cas et dans l'autre.

Pour terminer, remarquons que nous pourrions peaufiner nos solutions en ajoutant des affichages de libellés qui font encore un peu défaut

Solution n° 1 :

```
Rep = input("Voulez-vous un café? (O/N)")
while Rep != "O" and Rep != "N":
   print(“Répondre par O ou N. Réessayez.")
   Rep = input("Voulez-vous un café? (O/N)")
print(“Saisie acceptée.")
```

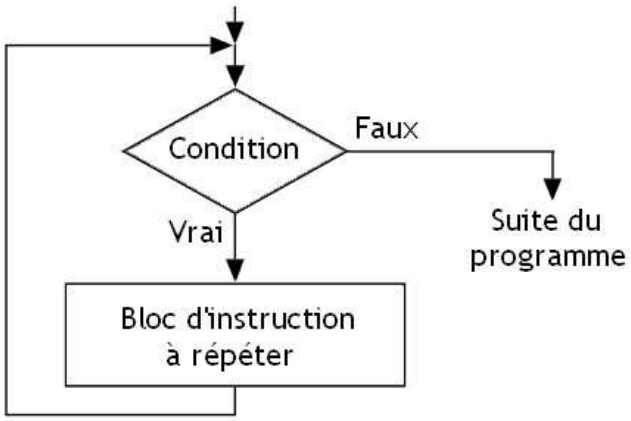
Solution n° 2 :

```
Rep = "X"
while Rep != "O" and Rep != "N":
   Rep = input("Voulez-vous un café? (O/N)")
   if Rep != "Y" and Rep != "N":
      print("mauvaise réponse. Réessayez")
print(“Saisie acceptée.")
```

Erreurs fréquentes

1. Ecrire une structure while dans laquelle la condition n’est jamais TRUE. Le programme ne rentre alors jamais dans la boucle !
2. Ecrire une boucle dans laquelle la condition ne devient jamais FALSE. L’ordinateur tourne alors dans la boucle sans pouvoir en sortir. C’est une « **boucle infinie** » .

Résumé



La boucle For

notre structure while ne sait pas à l'avance combien de tours de boucle elle va effectuer

Or, il arrive très souvent qu'on ait besoin d'effectuer un nombre **déterminé** de passages dans la boucle

C'est pourquoi une autre structure de boucle est à notre disposition : la structure **for**

La boucle for…in et range

En anglais, **range** signifie ampleur, gamme, envergure, étendue. Même si cette traduction n'est pas correcte, nous penserons intervalle

1) Syntaxe 1:

```
for compteur in range(n): #pour compteur variant de 0 à n-1
        # instruction 1
        # instruction 2
        #...
        # instruction p
```

La variable compteur est de type Entier

range(n) indique au compteur de parcourir tous les entiers de l'intervalle [0;n-1] avec un pas de 1 (incrémentation de 1 -> qui augmente de 1 à chaque tour)

Exemple :

```
for i in range(7) :
    print (i)
```

Résultat dans la console :

```
0
1
2
3
4
5
6
>>>
```

Le compteur *i* passe alors par les valeurs 0,1,2,3,4,5,6

2) Syntaxe 2:

```
for compteur in range(p,n): #pour compteur variant de p à n-1
        #instruction 1
        #instruction 2
        #...
        #instruction p
```

range (p,n) indique au compteur de parcourir tous les entiers de l'intervalle [p;n-1] avec un pas de 1 (incrémentation de 1 -> qui augmente de 1 à chaque tour)

Exemple :

```
for i in range(2,7) :
   print (i)
```

Résultat dans la console :

```
2
3
4
5
6
>>>
```

Le compteur *i* passe alors par les valeurs 2,3,4,5,6

3) Syntaxe 3:

```
for compteur in range(p,n,k): #pour compteur variant de p à n-1 avec un pas de k
        #instruction 1
        #instruction 2
        #...
        #instruction p
```

range (p,n,k) indique au compteur de parcourir tous les entiers de l'intervalle[p;n-1] avec un pas de k (incrémentation de k -> qui augmente de k à chaque tour)

Exemples :

```
for i in range(7,2,-1) :
   print (i)
```

Résultats dans la console :

```
7
6
5
4
3
>>>
```

```
for i in range(1,7,2) :
   print (i)
```

```
1
3
5
>>>
```

Erreur à ne pas faire

Examinons le porgramme suivant :

```
for Y in range (1,15):
   Y = Y * 2
   print(" Boucle numéro ",Y)
```

Vous remarquerez que nous gérons « en double » la variable Y, ces deux gestions étant contradictoires.

- D'une part, la ligne « for » augmente la valeur de Y de 1 à chaque passage.
- D'autre part la ligne « Y = Y * 2 » double la valeur de Y à chaque passage.

De telles manipulations perturbent complètement le déroulement normal de la boucle, et sont causes d'exécutions instables.

Des boucles dans des boucles

une boucle peut tout à fait contenir d'autres boucles

```
for i in range (15):
   print(" Il est passé par ici ")
   for j in range (6):
      print(" Il repassera par là ")
```

Dans cet exemple, le programme écrira une fois "il est passé par ici" puis six fois de suite "il repassera par là", et ceci quinze fois en tout. A la fin, il y aura donc eu 15 x 6 = 90 passages dans la deuxième boucle (celle du milieu), donc 90 écritures à l'écran du message « il repassera par là »

Des boucles peuvent donc être **imbriquées**

Important : lorsqu'on imbrique des boucles, il faut absolument utiliser des compteurs différents pour chaque boucle (i et j dans l'exemple)

Traduction d'une boucle For en boucle while

la structure « for » n'est pas du tout indispensable ; **on pourrait fort bien programmer toutes les situations de boucle uniquement avec un « while »**

Nous pouvons faire facilement la traduction entre une boucle for et une boucle while du langage Python. Considérons l'exemple suivant de traduction

```
#Affichage des entiers de 0 à 10
for i in range(0,11):
        print(i)
```

```
#Affichage des entiers de 0 à 10
i=0 #initialisation de i
while (i<= 10):
        print(i)
        i+=1 #incrémentation de i
```

Remarque:

Dans la boucle for une seule instruction suffit pour initialiser et incrémenter le compteur i : l'instruction range.

Dans la boucle while, deux instructions sont nécessaires: i=0 et i+=1.

L'intérêt du « for » est d'éviter au programmeur de gérer lui-même la progression de la variable qui lui sert de compteur.

Dit d'une autre manière, la structure « for » est un cas particulier de while : celui où le programmeur peut dénombrer à l'avance le nombre de tours de boucles nécessaires.