Chapitre 6 : Structures Répétitives

I. Les structures Répétitives :

Pour répéter un traitement un nombre de fois que ce soit connu ou inconnu les languages de programmation on recours à des structures qu'on appelle des boucles . Python n'est pas une exception , et alors python aussi a ces propres boucles

1- Boucle While (tantque) :

Syntaxe :

```
while condition :

    instruction_one
    instruction_two
    instruction_three
    …
    instruction_n
```

Il est maintenant important de se rappeler que:

- si vous voulez exécuter **plus d'une instruction à l'intérieur d'une** while , vous devez (comme avec if) mettre en **retrait** toutes les instructions de la même manière
- une instruction ou un ensemble d'instructions exécutées à l'intérieur de la boucle while est appelé **corps de** la **boucle**
- si la condition est False (égale à zéro) dès lors qu'elle est testée pour la première fois, le corps n'est pas exécuté une seule fois (notez l'analogie de ne rien faire s'il n'y a rien à faire)
- le corps devrait être en mesure de modifier la valeur de la condition, car si la condition est True au début, le corps peut fonctionner en continu à

l'infini - notez que faire une chose diminue généralement le nombre de choses à faire).

Exemple 1 :

```
x=int(input("Entrez un nombre"))
while x!=0:
    x= int(input("Entrez un nombre"))
print("BYE")
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
Entrez un nombre 5
Entrez un nombre 3
Entrez un nombre 8
Entrez un nombre 7
Entrez un nombre 3
Entrez un nombre0
BYE
```

Exemple 2 :

```
#Programme qui calcule combien de nombre paires entrés par l'utilisateur
#Et combien de nombre impaires
nbre_paires = 0
nbre_impaires = 0
nombre=int(input("Entrez un nombre ou tapez 0 pour sortir :"))
while nombre!=0:
    if nombre % 2 == 0 :
        nbre_paires +=1
    else:
        nbre_impaires+=1
    nombre= int(input("Entrez un nombre ou tapez 0 pour sortir :"))
print("le nombre des nombres paires est :",nbre_paires)
print("Le nombre des nombres impaires est :",nbre_impaires)
```

Résultat :

```
==== RESTART: C:/Users/TOSHIBA/AppData/Local/Programs/Python/Python38/if.py ====
Entrez un nombre ou tapez 0 pour sortir :5
Entrez un nombre ou tapez 0 pour sortir :2
Entrez un nombre ou tapez 0 pour sortir :16
Entrez un nombre ou tapez 0 pour sortir :11
Entrez un nombre ou tapez 0 pour sortir :18
Entrez un nombre ou tapez 0 pour sortir :0
le nombre des nombres paires est : 3
Le nombre des nombres impaires est : 2
```

Certaines expressions peuvent être simplifiées sans changer le comportement du programme.

Essayez de vous rappeler comment Python interprète la vérité d'une condition et notez que ces deux formes sont équivalentes :

while number != 0: et **while number :**

La condition qui vérifie si un nombre est impair peut également être codée sous ces formes équivalentes :

if number % 2 == 1: et **if number % 2:**

➢ La boucle infinie :

Une boucle infinie, également appelée **boucle sans fin** , est une séquence d'instructions dans un programme qui se répète indéfiniment (boucle sans fin.)

Voici un exemple de boucle qui n'est pas en mesure de terminer son exécution :

Exemple :

```
while True:
    print("Je suis coincé dans la boucle infinie")

print("BYE")
```

➢ Utilisation d'un compteur

Regardez l'extrait ci-dessous:

```
counter = 5
while counter != 0:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

Ce code est destiné à imprimer la chaîne "**Inside the loop**." et la valeur stockée dans la variable counter pendant une boucle donnée exactement cinq fois. Une fois que la condition n'est pas remplie (la variable counter atteint 0), la boucle est quittée et le message "**Outside the loop.**"ainsi que la valeur stockée dans countersont imprimés.

2- La Boucle For (Pour) :

Un autre type de boucle disponible en Python vient de l'observation qu'il est parfois plus important de **compter les "tours" de la boucle** que de vérifier les conditions.

Imaginez que le corps d'une boucle doit être exécuté exactement cent fois. Si vous souhaitez utiliser la boucle while pour le faire, cela peut ressembler à ceci:

```
i = 0
while i < 100:
# do_something()
i += 1
```

Ce serait bien si quelqu'un pouvait faire ce comptage ennuyeux pour vous. Est-ce possible?

Bien sûr que oui - il y a une boucle spéciale pour ce genre de tâches, et elle est nommée **for**.

En fait, la boucle for est conçue pour effectuer des tâches plus compliquées - **elle peut «parcourir» de grandes collections de données élément par élément** . Nous allons vous montrer comment faire cela bientôt, mais pour le moment, nous allons présenter une variante plus simple de son application.

Jetez un oeil à l'extrait:

```
for i in range(100):
# do_something()
```

Il y a de nouveaux éléments. Laissez-nous vous en parler :

- le mot-clé *for* ouvre la boucle for note - il n'y a aucune condition après cela vous n'avez pas à penser aux conditions, car elles sont vérifiées en interne, sans aucune intervention;
- toute variable après le mot clé *for* est la **variable** de **contrôle** de la boucle (compteur) il compte les tours de boucle et le fait automatiquement

• le mot clé *in* introduit un élément de syntaxe décrivant la plage de valeurs possibles affectées à la variable de contrôle

• la fonction range()(c'est une fonction très spéciale) est chargée de générer toutes les valeurs souhaitées de la variable de contrôle; dans notre exemple, la fonction créera (on peut même dire qu'elle **alimentera** la boucle avec) les valeurs suivantes de l'ensemble suivant: 0, 1, 2 .. 97, 98, 99; note: dans ce cas, la fonction range() démarre son travail à partir de 0 et le termine une étape (un nombre entier) avant la valeur de son argument;

Nos prochains exemples seront un peu plus modestes quant au nombre de répétitions de boucle.

Exemple :

```
for i in range(10):
print("The value of i is currently", i)
```

Exécutez le code pour vérifier si vous aviez raison.

Remarque:

• la boucle a été exécutée dix fois (c'est l'argument de la fonction range())

• la dernière valeur de la variable de contrôle est 9(pas 10, car **elle commence à partir** 0 , pas à partir de 1)

- **La fonction Range :**

- Range () est l'appel de cette fonction qui peut être équipée de deux arguments, pas d'un seul:

```
for i in range(2, 8):
print("The value of i is currently", i)
```

Dans ce cas, le premier argument détermine la (première) valeur initiale de la variable de contrôle.

Le dernier argument montre la première valeur à laquelle la variable de contrôle ne sera pas affectée.

Remarque: la range() fonction **accepte uniquement des entiers comme arguments** et génère des séquences d'entiers.

Pouvez-vous deviner la sortie du programme ? Exécutez-le pour vérifier si vous étiez en ce moment aussi.

La première valeur affichée est 2(tirée du range ()premier argument de.)

Le dernier est 7(bien que le range() deuxième argument de soit 8).

- La fonction range() peut également accepter **trois arguments**

```
for i in range(2,8,3):
    print("hello")
print("BYE")
```

Le troisième argument est le **pas** - c'est une valeur ajoutée pour contrôler la variable à chaque tour de boucle (comme vous vous en doutez, la **valeur par défaut du pas est 1**).

Résultat :

```
==== RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\d$.py ====
hello
hello
BYE
```

Remarque: si l'ensemble généré par la range()fonction est vide, la boucle n'exécutera pas du tout son corps.

Tout comme ici - il n'y aura pas de sortie:

```
for i in range(1, 1):
print("The value of i is currently", i)
```

Remarque: l'ensemble généré par le range()doit être trié **par ordre croissant** . Il n'y a aucun moyen de forcer le range()à créer un ensemble sous une forme différente. Cela signifie que le range()deuxième argument de doit être supérieur au premier.

Ainsi, il n'y aura pas de sortie ici non plus:

```
for i in range(2, 1):
print("The value of i is currently", i)
```

3- Les déclarations Break et Continue

Jusqu'à présent, nous avons traité le corps de la boucle comme une séquence d'instructions indivisible et inséparable qui sont exécutées complètement à chaque tour de la boucle. Cependant, en tant que développeur, vous pourriez être confronté aux choix suivants :

• il semble qu'il ne soit pas nécessaire de continuer la boucle dans son ensemble; vous devez vous abstenir de poursuivre l'exécution du corps de la boucle et aller plus loin;

• il semble que vous devez commencer le tour suivant de la boucle sans terminer l'exécution du tour en cours.

Python fournit deux instructions spéciales pour l'implémentation de ces deux tâches. Ces deux instructions sont les suivantes:

• **break**- quitte la boucle immédiatement et met fin inconditionnellement au fonctionnement de la boucle; le programme commence à exécuter l'instruction la plus proche après le corps de la boucle;

• **continue**- se comporte comme si le programme avait soudainement atteint la fin du corps; le tour suivant est commencé et l'expression de la condition est testée immédiatement.

Ces deux mots sont des **mots clés**.

Exemple :

```
# break - example
print("The break instruction:")
for i in range(1, 6):
    if i == 3:
        break
    print("Inside the loop.", i)
print("Outside the loop.")
# continue - example
print("\nThe continue instruction:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Inside the loop.", i)
print("Outside the loop.")
```

Résultat :

```
==== RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\d$.py ==
The break instruction:
Inside the loop. 1
Inside the loop. 2
Outside the loop.

The continue instruction:
Inside the loop. 1
Inside the loop. 2
Inside the loop. 4
Inside the loop. 5
Outside the loop.
```

4- L'instruction else :

Les boucles **while** et **for** en python seulement ont aussi une instruction **else** comme if qui s'exécute **toujours une fois, que la boucle soit entrée dans son corps ou non**.

Exemple avec for :

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

else:
    print("ecrire else:")
```

Exemple avec while :

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

II. Exercices :

Exercice 1

Créez une for boucle qui compte de 0 à 10 et imprime des nombres impairs à l'écran.

Exercice 2

Créez une while boucle qui compte de 0 à 10 et imprime des nombres impairs à l'écran.

Exercice 3

Créez un programme avec une for boucle et une break instruction. Le programme doit parcourir les caractères d'une adresse e-mail, quitter la boucle lorsqu'il atteint le @symbole et imprimer la partie précédente @sur une seule ligne.

Exercice 4

Créez un programme avec une for boucle et une continue instruction. Le programme doit parcourir une chaîne de chiffres, remplacer chacun 0par x et imprimer la chaîne modifiée à l'écran.