Structures De Contrôles

En générale, les instructions d'un programme sont exécutés d'une manière séquentielle, la première instruction, ensuite la deuxième, après la troisième et ainsi de suite. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs chemins d'exécution, ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle pour contrôler et choisir les chemins d'exécutions ou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures de contrôles conditionnelles et structures de contrôle répétitives (itératives).

Structures de contrôle conditionnelle (alternatives)

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est-ce-que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents.

Structure de contrôle alternative simple ' if '

Un test simple contient un seul bloc d'instructions.

Syntaxe :

```
if ( conditions ) :
        Instruction 1
        Instruction 2
              ⋮
        Instruction n
```

Si les conditions (une ou multiples) sont vérifiées, alors le Bloc d'instructions sera exécuté.

Si les conditions ne sont pas vérifiées, alors le Bloc d'instructions ne sera pas exécuté.

Exemple 1 : avec condition vrai

```
x = 15
if x > 10:
    print(x, "est plus grand que 10")
print("Fin")
```

Affichage après exécution

```
15 est plus grand que 10
Fin
```

Exemple 2 : avec condition fausse

```
x = 3
if x > 10:
    print(x, "est plus grand que 10")
print("Fin")
```

Affichage après exécution

```
Fin
```

Structure de contrôle alternative double ' if….else '

Un test double contient deux blocs d'instructions.

Syntaxe :

```
if ( conditions ) :
        bloc instructions 1
else :
        bloc d'instructions 2
```

Si les conditions (une ou multiples) sont vérifiées, alors le Bloc d'instructions 1 sera exécuté et le Bloc d'instructions 2 sera ignoré.

Si les conditions ne sont pas vérifiées, alors le Bloc d'instructions 2 sera exécuté et le Bloc d'instructions 1 sera ignoré

Exemple 1 : avec condition vrai

```
x = 15
if x > 10 :
        Print (x, "'est plus grand que 10")
else :
        Print (x, "n'est  grand que 10")
Print ("fin")
```

Affichage après exécution

```
15 est plus grand que 10
fin
```

Exemple 2 : avec condition fausse

```
x = 8
if x > 10 :
        Print (x, "'est plus grand que 10")
else :
        Print (x, " pas plus grand que 10")
Print ("fin")
```

Affichage après exécution

```
8 pas plus grand que 10
fin
```

Structure de contrôle alternative multiple ' if-elif-elif….else '

Dans le cas de plusieurs alternatives (conditions), il est possible de combiner plusieurs structures if-else.

Syntaxe :

```
if ( conditions 1 ) :
        bloc d'instruction 1
elif (condition 2) :
        bloc d'imstruction 2
              ⋮
elif (condition n) :
        bloc d'imstruction n
else:  Bloc d'instructions n+1      # Facultative
                                    # elle est exécuté lorsqu'aucune condition n'a été vérifiée
```

Les conditions (Conditions 1), (Conditions 2) … (Conditions n) sont évaluées l'une après l'autre jusqu'à ce que l'une de celles soit vérifiée, le bloc d'instructions lié à la condition vérifiée est exécuté et le traitement de la commande sera terminée.

Le Bloc d'instructions n+1 (else) sera exécuté si seulement si toutes les conditions (1,2…n) ne sont pas vérifiées.

Exemple :

```
x = int(input('donner un nombre')
if (x > 10) :
        Print ( "x est plus grand que 10")
elif (x < 10) :
        Print ("x  est plus petit que 10")
else :
        Print ("x  est égal à   10")
```

Affichage après exécution

Si la valeur de x est supérieure à 10

```
x est plus grand que 10
fin
```

Si la valeur de x est inférieure à 10

```
x est plus petit que 10
fin
```

Si la valeur de x vaut 10

```
x est égal  10
fin
```

Structures De Contrôle Répétitives (Itératives)

Les structures répétitives nous permettent de répéter un traitement d'un nombre fini de fois.

On dit que l'on réalise une **itération** de la boucle à chaque fois que le corps (bloc d'instructions indentée) de la boucle est exécuté.

La boucle while

La structure de contrôle répétitive while utilise une expression logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on la quitte.

Cette boucle est utilisée quand on ne connait pas le nombre de fois que la boucle doit être itérée.

Syntaxe :

while (conditions) :

    Bloc instructions

Exemples

```
x = 1
while x < 10:
    print("x a pour valeur", x)
    x = x * 2
print("Fin")
```

Affichage après exécution

```
x a pour valeur 1
x a pour valeur 2
x a pour valeur 4
x a pour valeur 8
Fin
```

Le mot-clé **while** signifie **tant que** en anglais. Le corps de la boucle (c'est-à-dire le bloc d'instructions indentées) sera répété tant que la condition est vraie.

Dans l'exemple ci-dessus, x sera multiplié par 2 tant que sa valeur reste inférieure à 10.

Remarque : Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté.

Si la condition reste toujours vraie, alors le corps de la boucle est répété **indéfiniment**.

La boucle for

La structure de contrôle répétitive **for** utilise un **indice** entier qui varie (avec un incrément = 1 par défaut) d'une valeur initiale jusqu'à une valeur finale.

À la fin de chaque itération, l'indice est incrémenté d'une manière automatique.

Remarque : Cette boucle est utilisée lorsqu'on connait le nombre d'itérations

Syntaxe :

```
for indice in séquence :
    bloc d'instructions
```

Séquence : est une collection de valeurs peut être générée avec la fonction **range()**

Exemples

Au lieu d'écrire

```
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
print('bonjour')
```

Il est possible d'utiliser la boucle **for**

```
for  i in range(0,10):
    print('bonjour')
```

La fonction range()

La fonction range permet de créer une liste de nombres compris entre un nombre de départ (inclus) et un nombre de fin (exclus)

Range() a trois paramètres : début, fin et pas

```
for i in range(10): # de 0 à 9 (10 non inclus)
for i in range(1,10): # de 1 à 9
for i in range(0,100,5): # de 0 à 95 par pas de 5
```

Remarque :

- Attention à l'indentation toujours
- On peut « casser » la boucle avec l'instruction ***break***
- On peut passer directement à l'itération suivante avec l'instruction ***continue***
- Des boucles imbriquées sont possibles
- Le bloc d'instructions peut contenir des conditions

Les instructions break et continue, et la clause else dans les boucles

L'instruction break

L'instruction **break** permet de « casser » l'exécution d'une boucle (**while** ou **for**). Elle fait sortir de la boucle et passer à l'instruction suivante.

Exemple

```
for i in range(10):
    print("debut iteration", i)
    print("bonjour")
    if i == 2:
        break
    print("fin iteration", i)
print("apres la boucle")
```

Affichage après exécution :

```
debut iteration 0
bonjour
fin iteration 0
debut iteration 1
bonjour
fin iteration 1
debut iteration 2
bonjour
apres la boucle
```

L'instruction continue

L'instruction **continue** permet de passer prématurément au tour de boucle suivant. Elle fait continuer sur la prochaine itération de la boucle.

Exemple

```
for i in range(4):
    print("debut iteration", i)
    print("bonjour")
    if i < 2:
        continue
    print("fin iteration", i)
print("apres la boucle")
```

Affichage après exécution :

```
debut iteration 0
bonjour
debut iteration 1
bonjour
debut iteration 2
bonjour
fin iteration 2
debut iteration 3
bonjour
fin iteration 3
apres la boucle
```

La clause else dans une boucle

La clause **else** dans un boucle permet de définir un bloc d'instructions qui sera exécuté à la fin seulement si la boucle s'est déroulée complétement sans être interrompue par un **break**.

Contrairement aux instructions présentes après la boucle, qui s'exécutent dans tous les cas (avec ou sans interruption par un **break**), le bloc d'instructions défini dans la clause **else** ne s'exécutera pas lors de l'interruption par un **break**. Après l'interruption, on passera directement aux instructions après la boucle.

Autrement dit, le bloc de la clause **else** est exécuté lorsque la boucle se termine par épuisement de la liste (avec **for**) ou quand la condition devient fausse (avec **while**), mais pas quand la boucle est interrompue par un **break**. Ceci est illustré dans la boucle suivante, qui recherche des nombres premiers :

```
for n in range(2, 8):
    for x in range(2, n):
        if n % x == 0:
            print(n, "egale", x, "*", n/x)
            break
    else:
        print(n, "est un nombre premier")
```

Affichage après exécution

```
2 est un nombre premier
3 est un nombre premier
4 egale 2 * 2.0
5 est un nombre premier
6 egale 2 * 3.0
7 est un nombre premier
```

Exercices

Exercice 1 :

Ecrire un programme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif ou nul.

Exercice 2 :

Ecrire un programme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif.

Exercice 3 :

Ecrire un programme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit :

(cas où l'utilisateur entre le nombre 7) Table de 7 :

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

.

7 x 9 = 63

Exercice 4 :

Ecrire un programme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre.

Par exemple, si l'on entre 5, le programme doit calculer : 1 + 2 + 3 + 4 + 5 = 15

Remarque : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

Exercice 5 :

Ecrire un programme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne. (while true)

Exercice 6 :

Ecrire un programme qui demande successivement 20 nombres à l'utilisateur, et qui lui affiche ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12

Entrez le nombre numéro 2 : 14

etc.

Entrez le nombre numéro 20 : 6

Le plus grand de ces nombres est : 14

Modifiez ensuite le programme pour affiche la position de ce nombre :

C'était le nombre numéro 2

Exercice 7 :

Réécrire le programme précédent, mais cette fois-ci on ne connaît pas d'avance combien l'utilisateur souhaite saisir de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice 8 :

Ecrire un programme qui affiche un triangle, dont la taille est donnée par l'utilisateur

Exemple pour une taille de 5ligne

```
    *
   ***
  *****
 *******
*********
```

Exercice 9 : (Apres chaine)

Un entier est dit distinct s'il est composé de chiffres distincts (différents).

Ecrire un programme python qui permet de saisir un entier n (n > 0), puis de vérifier et d'afficher si cet entier est distinct ou non.

Exemple

- **N=1273** est dit distinct car il est formé par les chiffres 1,2, 7 et 3 qui sont tous distincts, donc, le programme affichera : **cet entier est distinct**
- **N=1565** est dit non distinct car il est formé par les chiffres 1,5, 6, 5 qui ne sont pas tous distincts (le chiffre 5 se répète deux fois, donc le programme affichera : **cet entier est non distinct**