Cours structures conditionnelles et répétitives

September 9, 2025

1 Les structures conditionnelles

La structure conditionnelle est une structure dont les instructions sont exécutées selon des conditions.

1.1 Les opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer les objets en Python :

- < : **inférieur à**;
- > : **supérieur à**;
- <= : **inférieur ou égal à**;
- >= : **supérieur ou égal à**;
- == : **égal à**;
- != : **différent de**.

Les réponses de ces comparaisons sont de type booléen (***True*** ou ***False***).

[1]:
```
print("Est-ce que 100 est supérieur à 300 ? :", 100 > 300)
print("Est-ce que 70 est égal à '70' ? :", 70 == '70')
print("Est-ce que 'Dark' est égal à 'Light' ? :", 'Dark' == 'Light')

x1 = 5
x2 = 5.0
x3 = 1000

print("Est-ce que x1 est inférieur ou égal à x2 ? :", x1 <= x2)
print("Est-ce que x1 est équivalent à x3 ? :", x1 == x3)
print("Est-ce que x1 est inférieur à x3 ? :", x1 < x3)
```

```
Est-ce que 100 est supérieur à 300 ? : False
Est-ce que 70 est égal à '70' ? : False
Est-ce que 'Dark' est égal à 'Light' ? : False
Est-ce que x1 est inférieur ou égal à x2 ? : True
Est-ce que x1 est équivalent à x3 ? : False
Est-ce que x1 est inférieur à x3 ? : True
```

1.2 Les opérateurs logiques

- **and** : ET logique, retourne ***True*** (Vrai) lorsque les deux conditions considérées sont remplies et ***False*** (Faux) sinon;
- **or** : OU logique, retourne ***True*** lorsqu'au moins l'une des conditions considérées est remplie et ***False*** sinon;
- **not** : NON logique, retourne ***True*** si la condition considérée n'est pas remplie et ***False*** sinon.

Remarque : L'ordre de priorité : **not** > **and** > **or**.

[2]:
```
100 > 300 and 1 > 3
```

[2]:
```
False
```

[3]:
```
1 > 3 or 0 < 1
```

[3]:
```
True
```

[4]:
```
b1 = True
b2 = True
b3 = False

print( "True and True =", b1 and b2)
print( "True or False =", b2 or b3)
print("True and True and False =", b1 and b2 and b3)
print("True or True and False =", b1 or b2 and b3)
print("(True or True) and False =", (b1 or b2) and b3)
print("not True =", not b1)
```

```
True and True = True
True or False = True
True and True and False = False
True or True and False = True
(True or True) and False = False
not True = False
```

1.3 Bloc d'instructions – Indentation

Un bloc est défini par une indentation obtenue en décalant le début des instructions vers la droite en utilisant des espaces (habituellement 4 espaces mais ce n'est pas obligatoire). Toutes les instructions d'un même bloc sont placées au même niveau d'indentation.

1.4 Structure conditionnelle simple (un cas possible)

1.4.1 Syntaxe

if condition **:**

bloc d'instructions

La condition est une expression booléenne. Le **:** (deux-points) à la fin de la ligne introduit le bloc d'instructions qui sera exécuté si la condition est remplie (c'est-à-dire si elle prend pour valeur ***True***).

1.4.2 Exemple

[5]:
```
x = eval(input("Entrer un nombre "))

if x == 0 :
    print("Entrer un nombre non nul")
```

```
Entrer un nombre 0
Entrer un nombre non nul
```

1.5 Structure conditionnelle alternée (deux cas possibles)

1.5.1 Syntaxe

if condition **:**

bloc 1 d'instructions

else :

bloc 2 d'instructions

Si la condition vaut ***True*** alors le bloc 1 d'instructions sera exécuté, et le bloc 2 sera ignoré, sinon le bloc 2 d'instructions sera exécuté et le bloc 1 sera ignoré.

1.5.2 Exemple

[6]:
```
x = eval(input("Entrer la moyenne "))

if x >= 10 :
    print("Admis")
else :
    print("Non admis")
```

```
Entrer la moyenne 17
Admis
```

1.6 Structure conditionnelle alternée (plusieurs cas possibles)

1.6.1 Syntaxe

if condition1 **:**

bloc 1 d'instructions

elif condition2 **:**

bloc 2 d'instructions

...

else :

    dernier bloc d'instructions

Si la condition1 vaut ***True*** alors le bloc 1 d'instructions sera exécuté, et les autres blocs seront ignorés, sinon si la condition2 vaut ***True*** alors le bloc 2 d'instructions sera exécuté, et les autres blocs seront ignorés et ainsi de suite. Si toutes les conditions valent ***False*** le dernier bloc d'instructions sera exécuté.

[7]:
```
x = eval(input("Entrer un nombre "))

if x > 0 :
    print("Le nombre entré est positif")
elif x < 0 :
    print("Le nombre entré est négatif")
else :
    print("Le nombre entré est nul")
```

```
Entrer un nombre 77
Le nombre entré est positif
```

2 Instructions répétitives

En programmation, les boucles permettent de faire des itérations, c'est-à-dire de répéter le même bloc d'instructions. On trouve deux types de boucles : ***while*** et ***for***.

2.1 for

L'instruction ***for*** est utilisée pour répéter un nombre connu de fois un bloc d'instructions ou itérer sur les éléments d'un objet itérable.

2.1.1 Syntaxe

for élément ***in*** itérable ***:***

    bloc d'instructions

Dans l'en-tête de la boucle, on précise après le mot-clé ***for*** le nom d'une variable (élément dans l'exemple ci-dessus) qui prendra successivement toutes les valeurs qui sont données après le mot-clé ***in***.

Si vous devez itérer sur une suite de nombres, la fonction native ***range()*** est faite pour cela. Elle génère des suites arithmétiques :

2.1.2 Syntaxe

for item ***in range(start, stop, step):***

    bloc d'instructions

- La valeur par défaut de l'argument ***step*** est ***1*** et de l'argument ***start*** est ***0***.

- L'argument ***start*** est inclus dans le résultat tandis que L'argument ***start*** est exclu.

2.1.3 Exemples

[2]:
```
for item in range(0,10,1):   #équivalent à range(10)
    print(item)
```

```
0
1
2
3
4
5
6
7
8
9
```

[5]:
```
for item in range(0, 10, 2):
    print(item)
```

```
0
2
4
6
8
```

[6]:
```
#il vaut mieux utiliser _ au lieu d'un nom si on a pas besoin de la valeur␣
 ↪courante de la variable dans la boucle

for _ in range(5):
    print("bla-bla-bla")
```

```
bla-bla-bla
bla-bla-bla
bla-bla-bla
bla-bla-bla
bla-bla-bla
```

2.2 while

L'instruction ***while*** est utilisée pour exécuter des instructions de manière répétée tant qu'une expression est vraie.

2.2.1 Syntaxe

while condition ***:***

    bloc d'instructions

2.2.2 Exemples

[3]:
```
i = 1
while i<7 :
    print("Entrée numéro :", i)
    i = i + 1 #ou i += 1
```

```
Entrée numéro : 1
Entrée numéro : 2
Entrée numéro : 3
Entrée numéro : 4
Entrée numéro : 5
Entrée numéro : 6
```

[5]:
```
x = eval(input("Entrer un nombre strictement positif "))
while x<=0 :
    x = eval(input("Entrer un nombre strictement positif ! "))
print("Merci.")
```

```
Entrer un nombre strictement positif -1
Entrer un nombre strictement positif ! -17
Entrer un nombre strictement positif ! 0
Entrer un nombre strictement positif ! 1
Merci.
```