Python les conditions

If, elif, else, indentation

Les conditions en résumé

- Les structures conditionnelles permettent d'écrire des programmes qui peuvent s'exécuter différemment selon que certaines conditions sont vérifiées ou non.

Les programmes comme les trains peuvent prendre des chemins différents suivant les conditions(destination, heure, travaux...)

Nous avons vu précédemment que les programmes étaient une suite d'instructions. Il est cependant possible d'utiliser des **blocs** d'instructions qui ne seront pas exécutés dès leur apparition dans le programme à l'opposé des instructions écrites dans le flot principal(à gauche de la ligne).

En python, les blocs sont délimités grâce à des espaces et introduits grâce à deux points `:`.

```
entrée dans un bloc:(condition à respecter)
    ligne 1 du bloc
    ligne 2 du bloc
    ...
retour dans l'exécution normale du programme
ligne à ligne.
```

Aujourd'hui, nous allons utiliser des structures de contrôle qui permettent d'exécuter: des *blocs*, en fonction de conditions. On parle de **structure conditionnelle**.

Opérateurs de comparaison

Pour tester une condition, on utilise un opérateur de comparaison qui donne un résultat booléen: `True` ou `False`.

Soit x et y des variables de type `int` ou `float`, les opérateurs de comparaison sont:

- `x < y`: x est-il inférieur à y ?
- `x <= y`: x est-il inférieur ou égal à y ?
- `x > y`: x est-il supérieur à y ?
- `x >= y`: x est-il supérieur ou égal à y ?
- `x == y`: x est-il égal à y ?
- `x != y`: x est-il différent de y ?

> ⚠ **Avertissement**
>
> Comme vous l'avez remarqué pour tester une égalité on utilise le double signe égal `==`. Tout simplement, car le signe égal seul `=` est déjà utilisé pour l'affectation de variables.

Exemples

```
1  print(3 < 5)
```

>>SORTIE

```
True
```

```
1  print(3 <= 5)
```

>>SORTIE

```
True
```

Ou une erreur si on cherche à comparer *l'incomparable*:

```
1  3 < "5"
```

>>SORTIE

```
---------------------------------------------------------------
TypeError                                 Tracebac
Cell In[8], line 1
----> 1 3 < "5"

TypeError: '<' not supported between instances of
```

On peut constater que ces instructions retournent systématiquement un objet de type booléen.

```
1  type(3 > 5)
```

```
bool
```

Exemple

`3 + 4 < 2 * 12` renvoie `True` car les opérations `+` et `*` sont exécutées avant la comparaison `<`.

Les parenthèses sont donc inutiles (`(3 + 4) < (2 * 12)`).

Programme à structure conditionnelle :

Dans ce type de programme, le code sera toujours interprété ligne par ligne, cependant certains blocs de code ne seront exécutés que si une condition donnée est vraie.

1. L'instruction **if :**

Le bloc d'instruction contenu n'est exécuté que si la condition est vérifiée.

Syntaxe

```
1  if condition:
2      instruction 1
3      instruction 2
4      ...
```

```
1  a = 3
2  if a > 0:
3      a = -a
4  print(a)
```

>>SORTIE

```
-3
```

Dans le programme précédent l'instruction `if a > 0:` est toujours exécutée par l'interpréteur, car elle est dans le flux normal du programme(en début de ligne).

Par contre, l'instruction `a = -a` n'est exécutée que si la condition `a > 0` a renvoyé `True`, il s'agit d'un **bloc** relatif à l'instruction `if a > 0:` qui le précède. Elle est **indentée** avec quatre espaces. Si j'affecte la valeur négative -5 à `a`, ce bloc ne sera pas exécuté, et le changement de signe n'aura pas lieu.

```
1  a = -5
2  if a > 0:
3      a = -a
4  print(a)
```

>>SORTIE

```
-5
```

2. L'instruction **else** : **Indique le bloc d'instruction à exécuter si la condition if n'est pas vérifiée.**

```
if condition:
    bloc d'instructions exécuté si la condition est v
else:
    bloc d'instructions exécuté si la condition est f
```

```
a = -3
if a > 0:
    a = -a
else:
    a = a*100

print(a)
```

```
>>SORTIE

-300
```

Comme initialement la variable a n'était pas positive, c'est le bloc d'instructions `else` qui a été exécuté.

3. L'instruction **elif** : **Indique le bloc d'instruction à exécuter si la condition if n'est pas vérifiée.**

Cette instruction permet de réaliser des tests imbriqués, c'est un raccourci pour les instructions `else` et `if`.

Syntaxe

```
if condition 1:
    bloc d'instructions exécuté si la condition 1 est
elif condition 2:
    bloc d'instructions exécuté si la condition 1 est
else:
    bloc d'instructions exécuté si les conditions 1 e
```

```
a = 3
if a == 7:
    print("C'est un chiffre porte bonheur")
elif a == 3:
    print("Quelques modifications sont nécessaires")
    a += 4

print(a)
```

```
a = 3
if a == 7:
    print("C'est un chiffre porte bonheur")
elif a == 3:
    print("Quelques modifications sont nécessaires")
    a += 4

print(a)
```

```
>>SORTIE

Quelques modifications sont nécessaires
7
```

Les opérateurs booléens :

On peut combiner des conditions avec les opérateurs booléens.

Les opérateurs booléens français sont `ET`, `OU`, `NON` et leur correspondant en Python sont `and`, `or` et `not`.

- `and` renvoie `True` si les deux arguments ont pour valeur `True`.
- `or` renvoie `True` si au moins un des deux arguments a pour valeur `True`.
- `not` renvoie l'inverse de son argument `False` si l'argument est `True`, et vice-versa.

Exemples

- `1 < 2 and 3 < 4` renvoie `True`
- `1 < 2 and 3 > 4` renvoie `False`
- `1 < 2 or 3 < 4` renvoie `True`
- `1 < 2 or 3 > 4` renvoie `True`
- `not 1 < 2` renvoie `False`

Les opérateurs de comparaison sont prioritaires sur les opérateurs booléens. Quand on écrit `1 < 2 and 3 < 4` les opérations sont évaluées avant l'opération `and`; les parenthèses sont superflues: `(1 < 2) and (3 < 4)`.