

# Informatique et données pour les sciences

## Algorithmique, Boucles, Tests

8 octobre 2025

# Booléens

---

Les booléens, du nom du mathématicien Georges Boole, sont des variables ou expressions qui ne peuvent prendre que la valeur vrai ou faux. Les valeurs par défaut sont **True** et **False** (attention à la majuscule!).

```
>>>type(True)
<class 'bool'>
>>>type(False)
<class 'bool'>
```

L'opérateur `==` permet de comparer deux valeurs et retourne une valeur booléenne :

```
>>> 5 == 5
True
>>> 5 == 6
False
```

# Booléens

---

Outre l'opérateur `==`, il existe 6 autres opérateurs de comparaison usuels :

- ▶ `x!=y`    #x n'est pas égal y
- ▶ `x>y`    #x strictement supérieur à y
- ▶ `x<y`    #x strictement inférieur à y
- ▶ `x>=y`    #x supérieur ou égal à y
- ▶ `x<=y`    #x inférieur ou égal à y

Ces opérateurs vous sont familiers grâce aux mathématiques, mais attention à la différence entre `=` (affectation) et `==` (comparaison). De plus les symboles tels que `=<` ou `=>` n'existent pas !

# Opérateurs logiques

---

Il y a trois opérateurs logiques en Python : **and**, **or** et **not**. Par exemple

```
>>> 5 > 4 and 8 == 2 * 4
True
>>> True and False
False
>>> False or True
True
>>> not False
True
```

## Exécution conditionnelle : if

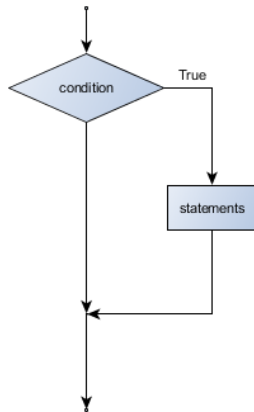
L'instruction **if** permet de vérifier des conditions et de changer le comportement du programme en fonction, par exemple :

```
nourriture = 'foie de morue'

if nourriture == 'foie de morue':
    print('Mon prefere !')
    print("J'ai envie de le dire 100 fois")
    print(100 * (nourriture + '! '))
```

À noter :

- ▶ : est nécessaire, il permet de distinguer la condition des actions réalisées si la condition est vérifiée.
- ▶ l'indentation est elle aussi nécessaire pour déterminer ce qui dépend de la condition.



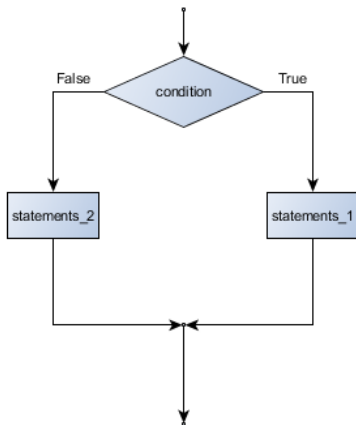
# Exécution conditionnelle : if / else

---

Il est fréquent de vouloir faire une chose si la condition est vraie et une autre dans le cas contraire, par exemple :

```
nourriture = 'foie de morue'

if nourriture == 'foie de morue':
    print('Mon prefere !')
else:
    print("Non merci !")
```

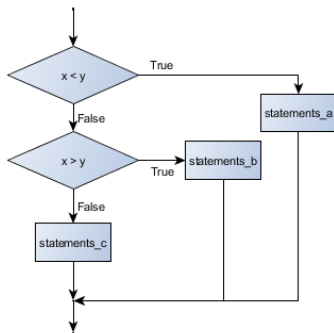


## Conditions à la chaîne : elif

Lorsqu'on a affaire à plus de deux possibilités, on peut augmenter le nombre de branches :

```
if choice == 'a':  
    print("Vous avez choisi 'a'.")  
elif choice == 'b':  
    print("Vous avez choisi 'b'.")  
elif choice == 'c':  
    print("Vous avez choisi 'c'.")  
else:  
    print("Mauvais choix.")
```

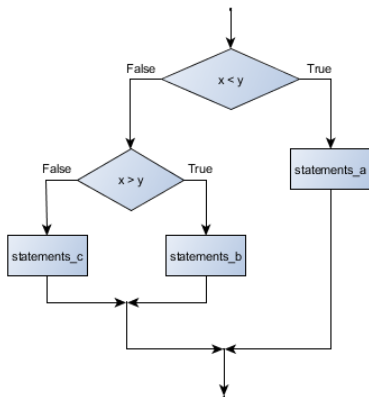
À noter : **elif** est une abbréviation de **else if**. Les conditions sont évaluées dans l'ordre, attention à la logique de vos programmes !



# Imbrication

On peut aussi imbriquer des conditions :

```
if x < y:
    STATEMENTS_A
else:
    if x > y:
        STATEMENTS_B
    else:
        STATEMENTS_C
```





# Imbrication

---

Les imbrications peuvent rapidement devenir difficiles à lire. Évitez les quand c'est possible, par exemple :

```
if 0 < x:  
    if x < 10:  
        print("0 < x < 10")
```

est moins lisible que :

```
if 0 < x and x < 10:  
    print("0 < x < 10")
```

au passage, Python autorise :

```
if 0 < x < 10:  
    print("0 < x < 10")
```

# Boucle for

---

Comme vu lors des dernières séances, la boucle for est particulièrement utile lorsqu'on veut itérer sur une liste d'éléments, par exemple :

```
for ami in ['Camille', 'Loup', 'Eva']:  
    salutation = "Salut " + ami + " !"  
    print(salutation)
```

ou encore

```
for i in range(5):  
    print('i vaut :', i)
```

## Boucle for

---

**for** est aussi particulièrement utile pour créer des listes, à l'aide d'une syntaxe particulière à Python :

```
>>> numbers = [1, 2, 3, 4]
>>> [x**2 for x in numbers]
[1, 4, 9, 16]
>>> [x**2 for x in numbers if x**2 > 8]
[9, 16]
>>> [[x, x**2, x**3] for x in numbers]
[[1, 1, 1], [2, 4, 8], [3, 9, 27], [4, 16, 64]]
>>> files = ['bin', 'Desktop', '.bashrc', '.ssh']
>>> [name for name in files if name[0] != '.']
['bin', 'Desktop']
>>> letters = ['a', 'b']
>>> [n * letter for n in numbers for letter in letters]
['a', 'b', 'aa', 'bb', 'aaa', 'bbb', 'aaaa', 'bbbb']
```

# Boucle for

---

Cette syntaxe particulière à Python :

```
[expr for item1 in seq1 for item2 in seq2 ...  
    ... for itemx in seqx if condition]
```

est équivalente à :

```
output_sequence = []  
for item1 in seq1:  
    for item2 in seq2:  
        ...  
        for itemx in seqx:  
            if condition:  
                output_sequence.append(expr)
```

## break et continue

---

On peut sortir immédiatement d'une boucle à l'aide de **break**. On utilise **continue** pour ne sauter que l'itération en cours, par exemple :

```
for i in [12, 16, 17, 24, 29]:  
    if i % 2 == 1:  
        break  
    print(i)  
print("fini")
```

donne :

```
12  
16  
fini
```

```
for i in [12, 16, 17, 24, 29]:  
    if i % 2 == 1:  
        continue  
    print(i)  
print("done")
```

donne :

```
12  
16  
24  
fini
```

On évitera par défaut ces instructions, tout comme on évitera de modifier l'incrément `i` d'une boucle `for` dans le corps de la boucle.

## Boucle while

---

Il est préférable, si on veut utiliser une condition d'arrêt, d'utiliser la boucle **while** qui exécute un nombre indéterminé de fois les instructions qu'elle contient tant que la condition n'est pas vérifiée :

```
number = 0
prompt = "What is the meaning of life,\n"
        "the universe, and everything? "

while number != "42":
    number = input(prompt)
```

À noter : si le nombre vaut 42 dès le départ, les instructions dans la boucle ne sont jamais mises en oeuvre.

## Boucle while

---

Attention aux **conditions d'arrêt d'une boucle while**, qui peut facilement devenir une **boucle infinie** ! On pourra par exemple utiliser un incrément pour s'assurer de la terminaison de la boucle :

```
nom = 'Jonathan'
devine = input("Devine mon nom: ")
pos = 0

while devine != nom and pos < len(nom):
    print("Rate ! Indice : lettre ", end='')
    print(pos + 1, "est", nom[pos] + ". ", end='')
    devine = input("Essaie encore : ")
    pos = pos + 1

if pos == len(nom) and name != guess:
    print("Dommage ! Le nom etait", nom + ".")
else:
    print("\nBien joue,", pos + 1, " essaie !")
```