



# Python - Instructions de contrôle

[Mise à jour le :3/7/2022]

- **Sources**
  - **Documentation** sur Python.org : [référence du langage](#), [if](#), [while](#) et [for](#), [fonctions natives](#) (built-in)
- **Lectures connexes**
  - **Real Python**
    - [Python enumerate\(\): Simplify Looping With Counters](#)
    - [The Python range\(\) Function \(Guide\)](#)
    - [Python "for" Loops \(Definite Iteration\)](#)
- **Mots-clés** : opérateurs de comparaison, tests, boucles

Les mots ci-dessous sont dits “réservés”. Ils ne peuvent pas être utilisés comme nom de variable. Les mots soulignés sont une nouveauté de Python 3. Les mots en **gras** sont utilisés dans cette page.

|              |                 |            |                 |              |
|--------------|-----------------|------------|-----------------|--------------|
| <b>and</b>   | <b>continue</b> | finally    | is              | raise        |
| as           | def             | <b>for</b> | lambda          | return       |
| assert       | del             | from       | <u>None</u>     | <b>True</b>  |
| <u>async</u> | <b>elif</b>     | global     | <u>nonlocal</u> | try          |
| <u>await</u> | <b>else</b>     | <b>if</b>  | <b>not</b>      | <b>while</b> |
| <b>break</b> | except          | import     | <b>or</b>       | with         |
| class        | <b>False</b>    | <b>in</b>  | <b>pass</b>     | yield        |

- **Fonctions natives (built-in)**<sup>1)</sup> utilisées dans les exemples : **enumerate()**, **list()**, **print()**, **range()**.

## Généralités

- **Structure d'une instruction de contrôle “if ... elif ... else ...”**

\*.py

```
if test1:
    <bloc d'instruction 1>
elif test2:
    <bloc d'instruction 2>
elif test3:
```

```
<bloc d'instruction 3>
...
else:
    <bloc d'instruction n>
```



Dans une structure **if elif else** un seul bloc d'instruction est exécuté.

#### • Mécanisme d'évaluation d'un test



Un test peut contenir n'importe quelle expression !

L'expression contenue dans le test est exécutée. Elle produit un objet sur lequel Python appelle la fonction built-in **bool(objet)**.

Selon l'objet renvoyé :

- la fonction built-in `bool(objet)`  $\Rightarrow$  `objet.__bool__()` qui renvoie **True** ou **False**

ou

- la fonction built-in `bool(objet)`  $\Rightarrow$  `objet.__len__()`
  - si `__len__()` retourne **0** alors **False**
  - si `__len__()` retourne autre chose alors **True**



Un objet **vide** est considéré comme **faux**. Un objet **non-vidé** est considéré comme **vrai**.

#### Exemples d'expressions

- **Un type built-in**
  - Est **Faux** : `False` `0` `None` `[]` `{}` `()` `"` (liste, dictionnaire, ensemble et chaîne de caractères vides)
  - Est **Vrai** : tout le reste

\*.py

```
d = {'marc':10}
if d: # dictionnaire non vide, on affiche
    print(d) # Résultat {'marc': 10}
```

#### • Une Comparaison

- `>` `>=` `<` `<=` `==` `!=`

\*.py

```
a=10;b=12
if a != b:
    print('La comparaison est fausse')
```

### • Un test d'appartenance

\*.py

```
if 'a' in 'marc':
    print('ok') # Résultat : ok
```

### • Un retour de fonction

- On évalue l'objet retourné.

\*.py

```
s='123'
if s.isdecimal():
    print(2*int(s)) # Résultat : 246
```

### • Un opérateur de test booléen and , or, not

\*.py

```
s='123'
if '1' in s and s.isdecimal():
    print(2*int(s)) # Résultat : 246
```

## 1. Les opérateurs dans les instructions de contrôle

Les instructions de contrôle utilisent des opérateurs logiques et de comparaison.

### 1.1 Les opérateurs de comparaison

Les opérateurs de comparaison permettent de réaliser les conditions placées dans les structures de contrôle (**prédicats**).

| égal | supérieur | inférieur | supérieur ou égal | inférieur ou égal | différent | se situe dans |
|------|-----------|-----------|-------------------|-------------------|-----------|---------------|
| ==   | >         | <         | >=                | ≤                 | !=        | in            |



Ne pas confondre l'opérateur d'égalité == avec l'opérateur d'affectation =

Exemple

[exop.py](#)

```
a >=5 # se traduit par a supérieur ou égal à 5
5<=a<=10 # pour un intervalle
```

## 1.2 Les opérateurs logiques

Les opérateurs **not**, **or** et **and** sont utilisés pour combiner des conditions.

Exemple

[exop.py](#)

```
a >=5 and a<=10 # à placer dans un test comme dans l'exemple ci-dessous
```

## 2. Les tests (ou structures alternatives)

Ce type d'instruction permet au code de suivre différents chemins.

### 2.1 if ... else ...

Syntaxe

```
if prédicat:
    bloc de code 1 # si le prédicat est vrai alors on exécute le bloc de code 1
else:
    bloc de code 2 # sinon on exécute le bloc de code 2
```



On dit du code décalé vers la droite qu'il est **indenté**. En Python l'indentation est **fondamentale**. Elle fait intégralement partie de la syntaxe du langage.

Exemple 1 : comparaison d'une variable à une valeur

[extest1.py](#)

```
a=5
if a<10:
    print("C'est vrai")
else:
    print("c'est faux") # Résultat : C'est vrai
```

Exemple 2 : utilisation de **in**

exin.py

```
chaine = "Bonjour. Comment allez-vous ?"
for lettre in chaine:
    if lettre in "AEIOUYaeiouy": # lettre est une voyelle
        print(lettre)
    else: # lettre est une consonne
        print("*")
```

## 2.2 if ... elif ... else ...



On utilise **elif** lorsqu'il est nécessaire d'enchaîner plusieurs tests.

### Syntaxe

```
if prédicat 1:
    bloc de code 1 # si le prédicat 1 est vrai alors on exécute le bloc de
code 1
elif prédicat 2:
    bloc de code 2 # sinon si le prédicat 2 est vrai alors on exécute le bloc
de code 2
else:
    bloc de code 3 # sinon on exécute le bloc de code 3
```



**elif** est une **concaténation** de **else if**. On peut utiliser autant de elif que nécessaire.

### Exemples

extest2a.py

```
a=5
if a<0:
    print("a est positif est inférieur à 5")
elif a >=5 and a<=10:
    print("a appartient à [5,10]")
elif a >10 and a<=20:
    print("a appartient à [11,20]")
else:
    print("a est supérieur à 20") # Résultat : a appartient à [5,10]
```

extest2b.py

```
mot = input("Saisissez un mot: ")
if 'e' in mot:
```

```
print("Le mot", mot, "contient le caractère 'e'.")
else:
    print("Le mot", mot, "ne contient pas le caractère 'e'.")
```

## 2.3 L'instruction pass



Python ne disposant pas d'accolades pour délimiter les blocs de code, il existe une instruction **pass**, qui ne fait rien.

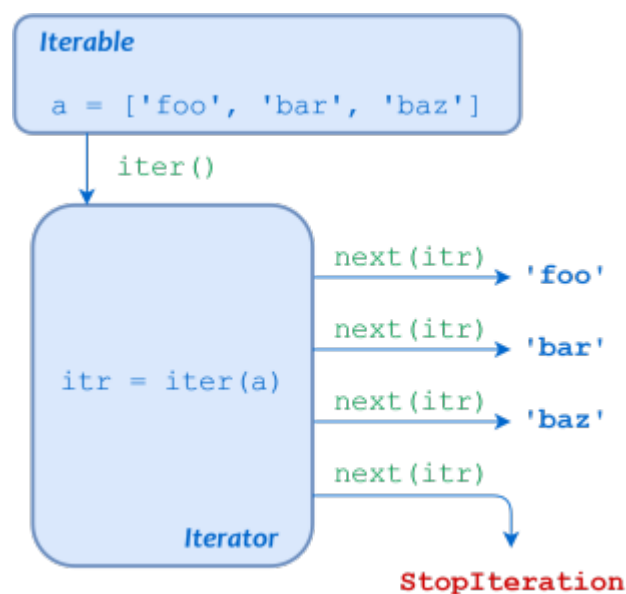
Exemple

\*.py

```
if condition:
    pass          # pass remplace un bloc de code que l'on a supprimé.
                  Cette instruction évite d'inverser le test
else:
    action(s)
```

## 3. Les boucles (ou structures répétitives)

Ce type d'instruction permet au programme de **répéter**, de **compter** ou d'**accumuler** avec très peu d'instructions.



### 3.1 La boucle while



On utilise une boucle **while** lorsqu'on ne connaît pas le nombre de répétition à effectuer. Les instructions du corps de la boucle sont exécutées tant qu'une condition



est vraie.

## Syntaxe

```
while prédicat:  
    bloc de code # le bloc de code est  
    # exécuté tant que la condition est vraie
```



Afin que la boucle puisse s'interrompre, il faut veiller à faire évoluer dans le corps de la boucle au moins une des variables intervenant dans la condition.

## Exemple 1

exboucle1.py

```
compteur = 0 # Initialisation de la variable utilisée dans la boucle  
while compteur < 10:  
    print(compteur) # Résultat : 0 1 2 3 4 5 6 7 8 9  
    compteur = compteur + 1 # Cette instruction assure l'arrêt de la  
    boucle pour compteur = 10
```

## Exemple 2

\*.py

```
while True:  
    bloc de code # cette boucle ne se termine jamais ! Cette structure  
    est utilisée dans quelques cas particuliers.
```

## 3.2 La boucle for



On utilise une boucle **for** lorsqu'on connaît le nombre de répétitions à effectuer. Elle est utilisée pour **parcourir** une séquence (une **liste**, un **tuple**, un **dictionnaire**, un **ensemble** ou une **chaîne**)

- **Cas 1 : itérer sur une séquence (for element in sequence)**

L'instruction `for element in sequence` permet d'itérer sur une collection de données, tels une **liste** ou un **dictionnaire**.

`element` est une variable créée par `for`. Elle prend successivement chacune des valeurs figurant dans la séquence parcourue.

## Syntaxe

```
for nom_variable in collection:  
    bloc de code
```

## Exemple

exboucle2.py

```
ma_liste = ["chien", "chat", "souris", "cheval"]  
for x in ma_liste:  
    print(x) # A chaque itération x prend la valeur pointée dans la  
             liste,  
             # la variable de boucle est gérée automatiquement.  
             # Résultat affiché dans la console : chien chat souris  
             cheval
```

### • Cas 2 : répéter l'exécution du code (for et la fonction range())



Pour répéter l'exécution de code un nombre de fois spécifié, on utilise la fonction range(). La fonction range() renvoie une séquence de nombres, commençant par 0, incrémente de 1 (par défaut) et se termine par un nombre spécifié.

## Syntaxe

\*.py

```
for variable in range(début, fin, pas):  
    # au premier tour de boucle variable = début  
    # à chaque tour variable = variable + pas  
    # la boucle s'arrête lorsque variable = fin
```

### Exemple 1 : boucle for avec **un seul paramètre**

\*.py

```
# cas particulier : si la variable n'est pas utilisée, on peut la  
remplacer par _  
for _ in range(3):  
    print("A") # Affichage A A A
```

\*.py

```
# Avec un indice (ou compteur) de boucle  
for i in range(3):
```



```
print(i) # Affichage : 0 1 2
```

Exemple 2 : boucle for avec **deux paramètres**

\*.py

```
for i in range(1,4):  
    print(i) # la première valeur de i est 1, la boucle s'arrête pour  
i=4  
# Affichage : 1 2 3
```

Exemple 3 : boucle for avec **trois paramètres**

\*.py

```
for i in range(1,11,2):  
    print(i) # la première valeur de i=1, l'incrément est 2, la boucle  
s'arrête à 11  
# Affichage : 1 3 5 7 9
```

### 3.3. Les mots-clés break, continue et pass

- **break**



Le mot-clé **break** permet d'**interrompre** une boucle.

Exemple

exbreak.py

```
while 1: # 1 est toujours vrai -> boucle infinie  
    lettre = input("Entrer Q pour quitter : ")  
    if lettre=="Q":  
        print("Fin de boucle !")  
        break
```

- **continue**



Le mot-clé **continue** permet de **poursuivre l'exécution** d'une boucle en repartant du *while* ou du *for*.

Exemple

## excontinue.py

```
i=1
while i<20:
    if i%3 == 0:
        i = i + 4 # on incrémente i de 4
        print("i est maintenant égale à",i)
        continue # retourne au while sans exécuter les lignes ci-
        dessous
    print("La variable i =",i)
    i = i + 1 # si i%3!=0 on incrémente i de 1
```

- **pass**



Python ne disposant pas d'accolades pour délimiter les blocs de code, il existe une instruction **pass**, qui ne fait rien.

### Exemple

\*.py

```
liste = list(range(10))
print('avant', liste)
while liste.pop() != 5:
    pass
print('après', liste)
```

## 3.4 La fonction built-in enumerate()



enumerate permet de boucler sur un iterable tout en disposant d'un compteur.

### Exemples

- Liste simple

\*.py

```
lst=['a','b','c','d','e','f','g','h','i']
for compteur, valeur in enumerate(lst):
    print(compteur, valeur) # Résultat : 0 a
                                1 b
                                2 c etc.
```

On peut fixer la première valeur de la variable compteur.

\*.py

```
lst = ['apple', 'banana', 'grapes', 'pear']
for compteur, valeur in enumerate(lst, 1):
    print(compteur, valeur)

# Résultat
# 1 apple
# 2 banana
# 3 grapes
# 4 pear
```

- Liste de listes

\*.py

```
lst = [
    ['a', 'b', 'c', 'd'],
    ['e', 'f', 'g', 'h'],
    ['i', 'j', 'k', 'l'],
    ['m', 'n', 'o', 'p'],
]

for x, ligne in enumerate(lst):
    print(x, ligne)
    for y, val in enumerate(ligne):
        print(x, y, val)

# Résultat
# 0 ['a', 'b', 'c', 'd']
# 0 0 a
# 0 1 b
# 0 2 c
# 0 3 d etc.
```

- Création d'une liste de tuples

\*.py

```
lst=['a','b','c','d','e','f','g','h','i']
list(enumerate(lst,1))
# Résultat [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e'), (6, 'f'),
(7, 'g'), (8, 'h'), (9, 'i')]
```



## Quiz

- [Conditional Statements in Python](#)

## Pour aller plus loin ...

- [For-Else: A Weird but Useful Feature in Python](#)
- [Python's all\(\): Check Your Iterables for Truthiness](#)
- [How Can You Emulate Do-While Loops in Python?](#)

## Résumé

- Les conditions sont identifiées par les mots-clés **if** (si), **elif** (sinon si) et **else** (sinon).
- Les mots-clés **if** et **elif** doivent être suivis d'un test (appelé aussi **prédicat**)
- Les booléens sont des données soit vraies (**True**) soit fausses (**False**)
- Une boucle sert à **répéter** une portion de code en fonction d'un **prédicat**.
- On peut créer une boucle grâce au mot-clé **while** suivi d'un prédicat.
- On peut parcourir une séquence grâce à la syntaxe **for element in sequence**:
- Python ne disposant pas d'accolades pour délimiter les blocs de code, il existe une instruction **pass**, qui ne fait rien.

<sup>1)</sup>

Fonctions toujours disponibles.

From:

<http://webge.fr/dokuwiki/> - **WEBGE Wikis**

Permanent link:

<http://webge.fr/dokuwiki/doku.php?id=python:bases:controle&rev=1660038875>

Last update: **2022/08/09 11:54**

