

NOTE 3

Contrôle et déroulement du flux d'exécution

Gérer le déroulement du programme

Jusqu'à maintenant, les instructions d'un programme ont été interprétées les unes après les autres, dans l'ordre où elles sont écrites.

L'ordre d'exécution des instructions des algorithmes n'est pas toujours séquentielle, on peut le modifier avec des structures conditionnelles ou des structures répétitives

- Structures conditionnelles

if effectue une action si une condition est vraie (true) ou l'omet si elle est fausse (false).

if/else effectue une action si une condition est vraie (true) ou une autre action si elle est fausse (false)

if/elif/elif/else effectue une action parmi plusieurs, selon la valeur d'une expression et elle peut posséder un cas par défaut

- Structures répétitives

Répétition d'instructions jusqu'à ce qu'une condition soit satisfaite.

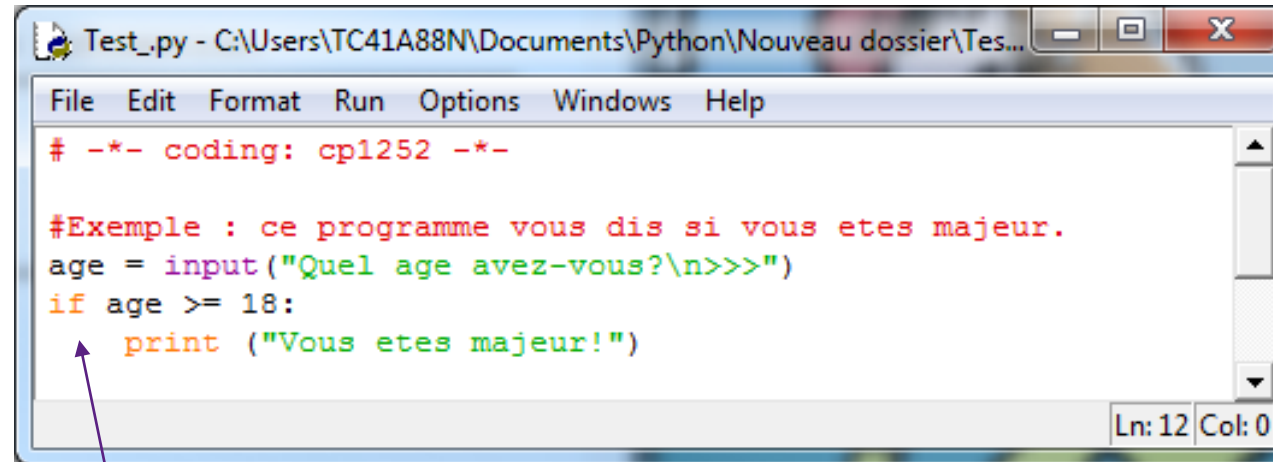
while
for

La conditionnelle if

Dans un programme, les conditions permettent de faire des choix dépendants des différentes données que l'on peut retrouver dans notre programme.

Syntaxe :

*if expression logique :
instruction*



```
Test_.py - C:\Users\TC41A88N\Documents\Python\Nouveau dossier\Tes...
File Edit Format Run Options Windows Help

# -*- coding: cp1252 -*-

#Exemple : ce programme vous dis si vous etes majeur.
age = input("Quel age avez-vous?\n>>>")
if age >= 18:
    print ("Vous etes majeur!")

Ln: 12 Col: 0
```

```
>>>
Quel age avez-vous?
>>>18
Vous etes majeur!
>>> =====
>>>
Quel age avez-vous?
>>>15
>>> |
```

Il est important que les instructions qui se font quand la condition est vraie soient indentées vers la droite pour faire partie du « if »

If/elif et else

Il est possible de

- gérer plusieurs conditions avec elif
- Gérer un cas par défaut avec else

Syntaxe :

if *expression logique*:

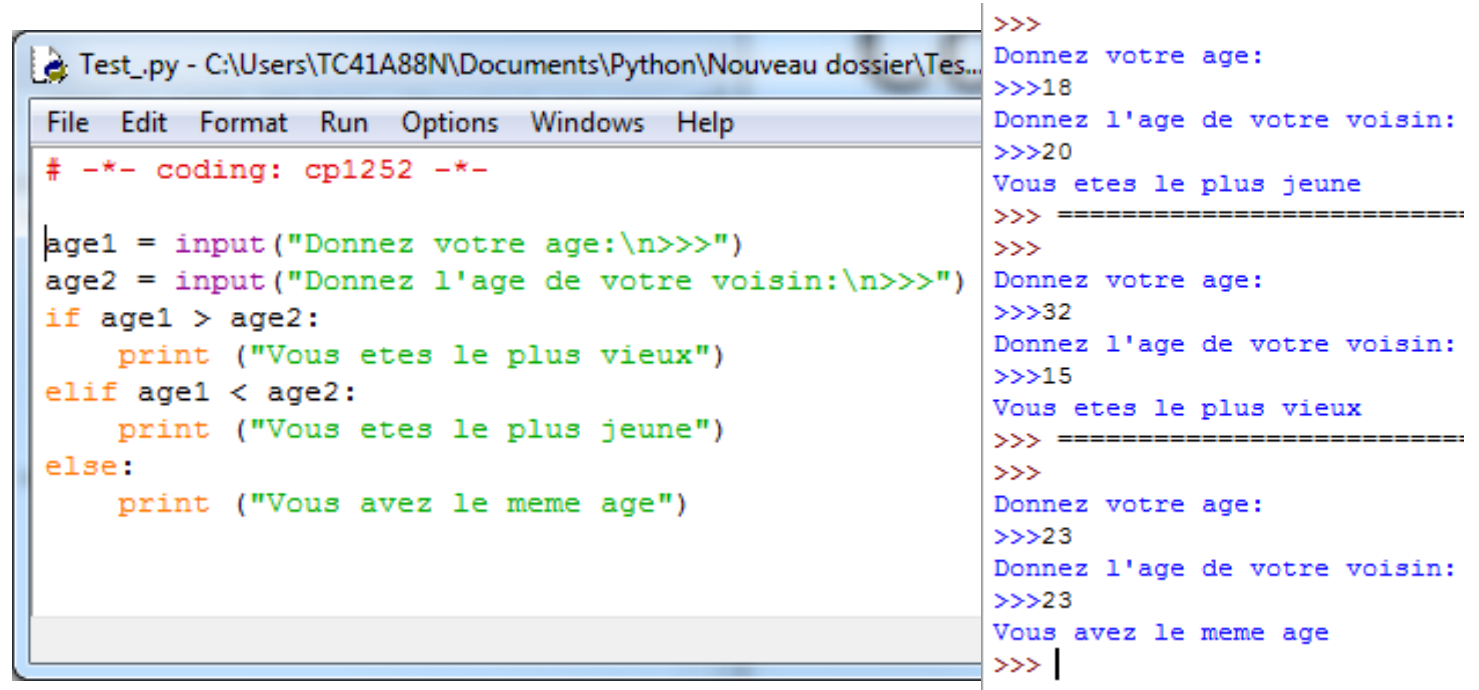
instruction

elif *expression logique*:

instruction

else :

instruction



The screenshot shows a Python IDE window titled 'Test_.py'. The code in the editor is as follows:

```
# -*- coding: cp1252 -*-

age1 = input("Donnez votre age:\n>>>")
age2 = input("Donnez l'age de votre voisin:\n>>>")
if age1 > age2:
    print("Vous etes le plus vieux")
elif age1 < age2:
    print("Vous etes le plus jeune")
else:
    print("Vous avez le meme age")
```

The output on the right shows three test cases:

```
>>>
Donnez votre age:
>>>18
Donnez l'age de votre voisin:
>>>20
Vous etes le plus jeune
>>> =====
>>>
Donnez votre age:
>>>32
Donnez l'age de votre voisin:
>>>15
Vous etes le plus vieux
>>> =====
>>>
Donnez votre age:
>>>23
Donnez l'age de votre voisin:
>>>23
Vous avez le meme age
>>> |
```

Le « if » doit toujours être placé en premier et le « else » (optionnel) à la fin!

Les tabulations sont nécessaires pour délimiter les instructions à faire dans chaque cas

Ne pas confondre : = l'opérateur d'affectation et == qui est l'opérateur de comparaison d'égalité.

◦ L'instruction switch n'existe pas en Python !

Opérateur ternaire

Cet opérateur est une alternative au if-else.

Syntaxe :

expression logique ? expression₁ : expression₂

Si l'expression logique est vraie, la valeur de expression₁ est retournée.
Autrement, la valeur de expression₂ est retournée comme résultat.

Exemple

```
age=18 ? print('tu es majeur): print('tu es mineur)
```



Équivalent de

If age=18 :

print('tu es majeur)

else :

print('tu es mineur)

Pour des raisons de lisibilité, les parenthèses peuvent donc être nécessaires.

Attention, cet opérateur peut nuire à la lisibilité du programme.

Opérateurs de comparaison

- Les opérateurs de comparaison permettent de construire des expressions logiques qui servent dans les conditions des boucles **while** et les **if**

Opérateur	signification	Exemple	Résultat
<=	strictement inférieur à	3<=5	True
<	inférieur ou égal à	'pizza'<'bière'	False (bière est avant pizza dans le dictionnaire)
>	strictement supérieur à	5>8	False
>=	supérieur ou égal à	10>=8	True
==	égal à	5==5	True
!=	différent de	5!=7	True
in	Appartient	5 in [3,4,5]	True
not in	n'appartient pas	'chat' not in [3,'chat',5]	False

Opérateurs logiques

- Les opérateurs logiques permettent de construire des expressions logiques plus complexes

Opérateur	signification	Exemple	Résultat
Expression1 and Expression2	Et (il faut que les deux expressions soient vraies)	5<6 and 6<10	True
Expression1 or Expression2	Ou (non exclusif, une seule vraie suffit)	5>6 or 6==10	False

- On utilise les tables de vérité pour déterminer la valeur des expressions

Expression 1	Expression2	Expression1 AND Expression2	Expression1 OR Expression2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Les boucles

Les boucles sont quasi indispensable en programmation.

Elles permettent de répéter des instructions un certain nombre de fois.

En Python, on peut compter sur **les boucles while** et **les boucles for**

Les boucles do while n'existent pas en python

Les boucles while

Une boucle while répète les instruction tant que la condition d'entrée est vérifiée (True).

Les instructions présentes dans la boucle doivent être décalées d'une tabulation de plus que l'initialisation de la boucle

Syntaxe :

```
while expression logique :  
    instruction
```

Aussi longtemps que l'expression logique est vraie, on exécute l'instruction ou le bloc d'instructions. Autrement, on sort de la boucle.

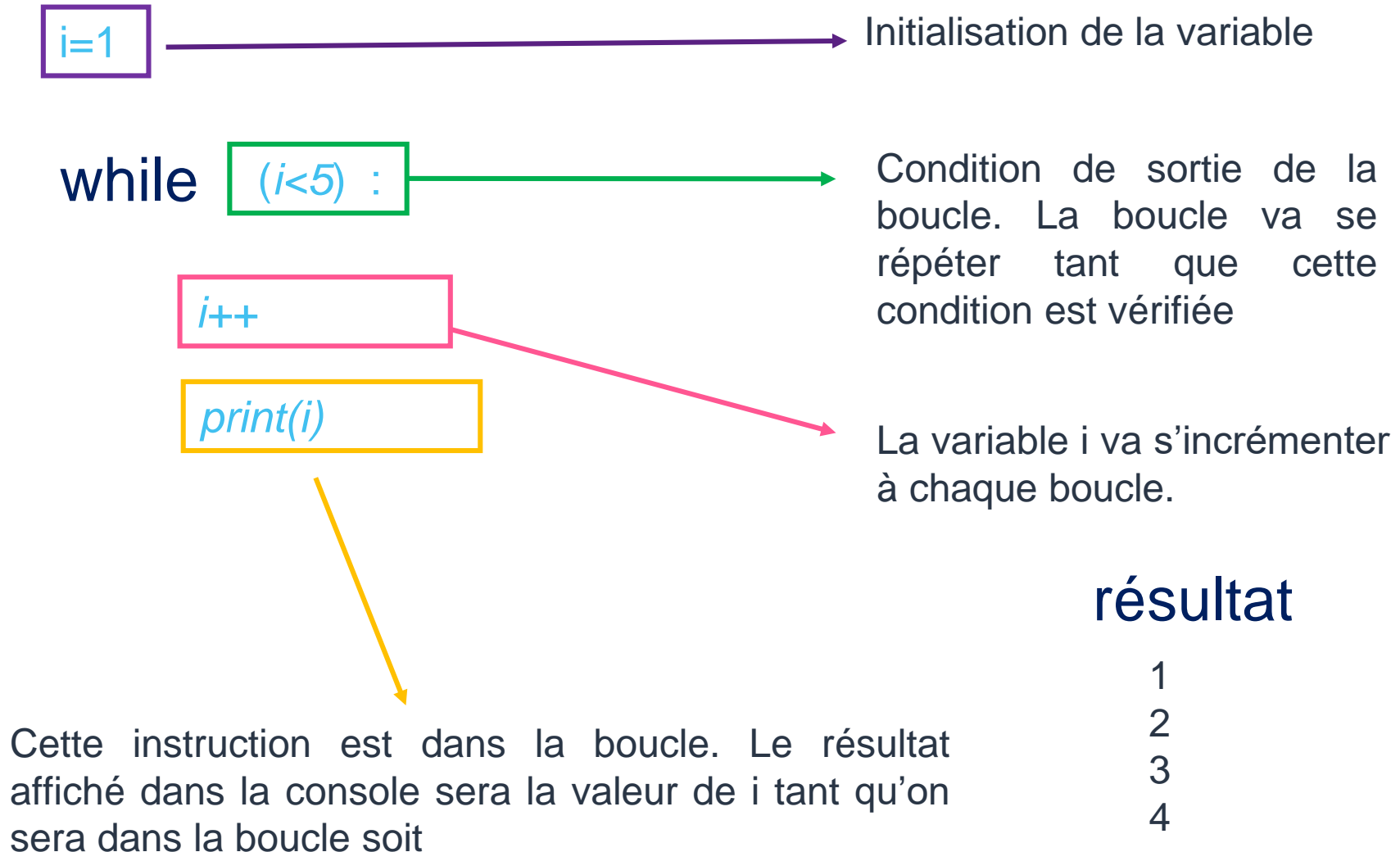
Exemple

```
i=1  
while i<5 :  
    print(i)  
    i++
```

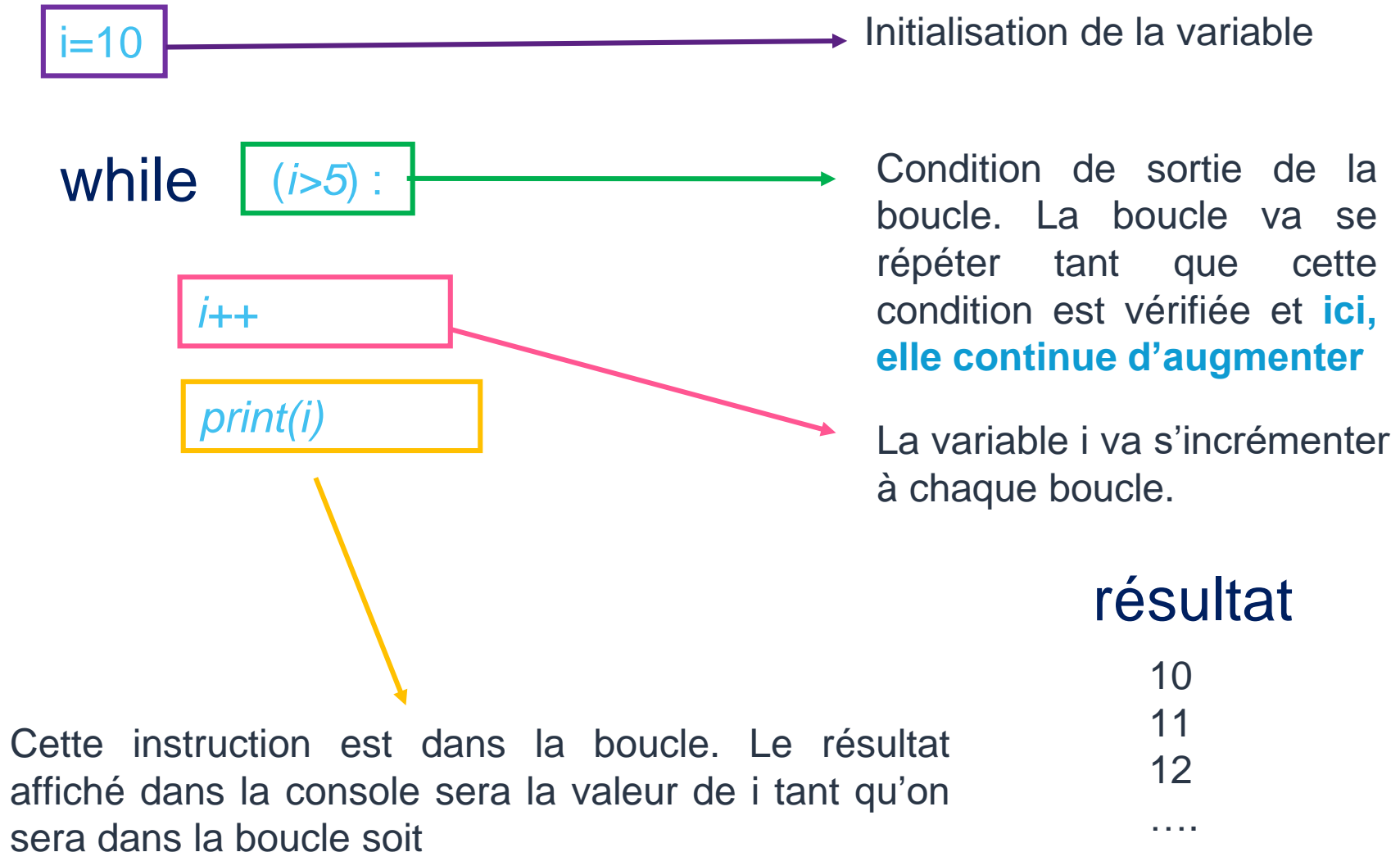
Au début de l'exécution, si l'expression logique est fausse, on passe à l'instruction qui suit la boucle while.

Attention, ce genre de boucles peut entrainer des boucles infinies!

Les boucles while



Les boucles infinies



résultat

10
11
12
....

Les boucles for

La répétition contrôlée par compteur est gérée complètement par la structure de répétition **for**.

Les instructions présentes dans la boucle doivent être décalées d'une tabulation de plus que l'initialisation de la boucle

Syntaxe :

```
for variable in range (max):  
    instructions
```

variable est une variable qui va prendre toutes les valeurs possibles entre min et max-1, par défaut min vaut 0

· **max** est la valeur maximale qui ne sera jamais atteinte par la boucle

Syntaxe :

```
for variable in range (min,max,pas):  
    instructions
```

pas est la valeur qui va incrémenter ou décrémenter variable (par défaut 1). Le pas peut être négatif

instructions est le bloc d'instruction à exécuter à chaque tour de boucle

La fonction range


On utilise la fonction **range** avec la boucle **for** pour incrémenter ou décrémenter la variable

```
for i in range (5):  
    print(i)
```

On part de 0
on n'atteint jamais la valeur 5
On gagne 1 à chaque tour

résultat

0
1
2
3
4




+1
Pas par défaut

```
for i in range (2,10, 3):  
    print(i)
```

On part de 0
on n'atteint jamais la valeur 10
On gagne 3 à chaque tour

résultat

2
5
8




+3

```
for i in range (2,5):  
    print(i)
```

On part de 2
on n'atteint jamais la valeur 5
On gagne 1 à chaque tour

résultat

2
3
4




+1

```
for i in range (5,0,-2):  
    print(i)
```

On part de 5
on n'atteint jamais la valeur 0
On perd 2 à chaque tour

résultat

5
3
1



-2

Les boucles for sur les listes et les chaines de caractères

La boucle **for** permet de parcourir tous les éléments d'une liste.

```
Maliste=['coucou','hello','wilkommen']  
for i in Maliste :  
    print(i)
```

résultat

coucou
hello
wilkommen




A chaque tour de boucle, la variable prend la valeur d'un élément de la liste

Une chaine est considérée comme une liste de caractères

```
Machaine='abc'  
for i in machaine :  
    print(i)
```

résultat

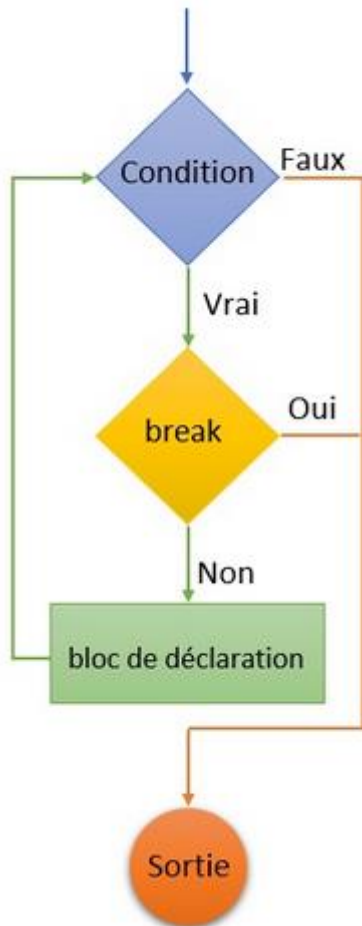
a
b
c



A chaque tour de boucle, la variable prend la valeur d'un caractère de la liste

L'instruction break

L'instruction break provoque la sortie immédiate dans une structure while, for.
L'exécution du programme se poursuit avec la première instruction suivant cette structure.



On utilise souvent l'instruction break pour sortir d'une boucle plus tôt.

Exemple

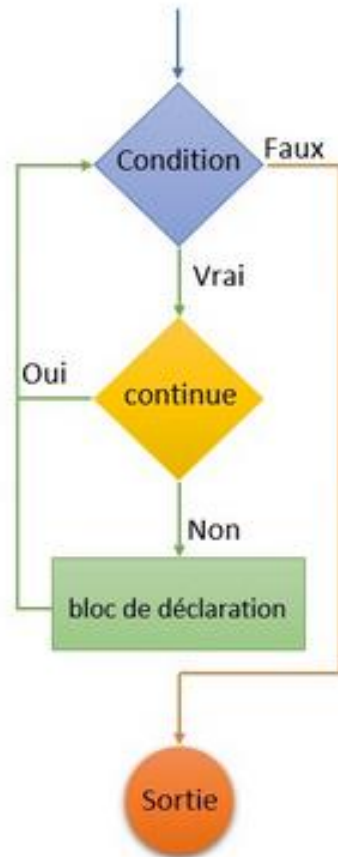
```
Machaine='abc'  
for i in machaine :  
    print(i)  
    if i=='b' :  
        break
```

On parcourt tous les caractères de la chaîne pour savoir si b existe dedans

On utilise break pour sortir dès qu'il est trouvé

L'instruction continue

continue va permettre de remonter en haut de la boucle sans exécuter les instructions qui se trouve après



Exemple

```
x=2
while x<7:
    if x%2==0:
        x++
        continue
    else :
        x++
        print(x)
```

résultat

4
6

Au départ, x vaut 2

On rentre dans la boucle

il est divisible par 2,

on l'augmente de 1

on retourne en haut de la boucle directement

On re-rentre dans la boucle

il vaut 3, on va dans le

« else »,

il augmente de 1 (x=4)

On l'affiche ...

ESiEE[it]
L'école de l'expertise numérique



une école de la



www.esiee-it.fr