INFO-H-100 - Informatique

Séance d'exercices 2
Introduction à Python
Tests et boucles

Université Libre de Bruxelles
Faculté des Sciences Appliquées

2017-2018

Expressions booléenes

Une expression booléenne est une expression dont la valeur est soit vrai (`True`) , soit faux (`False`). Ces expressions sont de type `bool`.

```
>>> 5 == 5
True
>>> 5 != 5
False
```

Elle peut se composer d'opérandes et de comparateurs :
`<` `<=` `>` `>=` `!=` `==`

Ne pas confondre = (assignation) et == (comparaison d'égalité).

Tests simples

L'instruction `if` permet de tester une condition et d'exécuter du code si cette condition est vérifiée.

```
x = 2
if x >= 0 :
  print('x est positif')
```

Le code exécuté est constitué du code indenté qui suit l'instruction `if`.

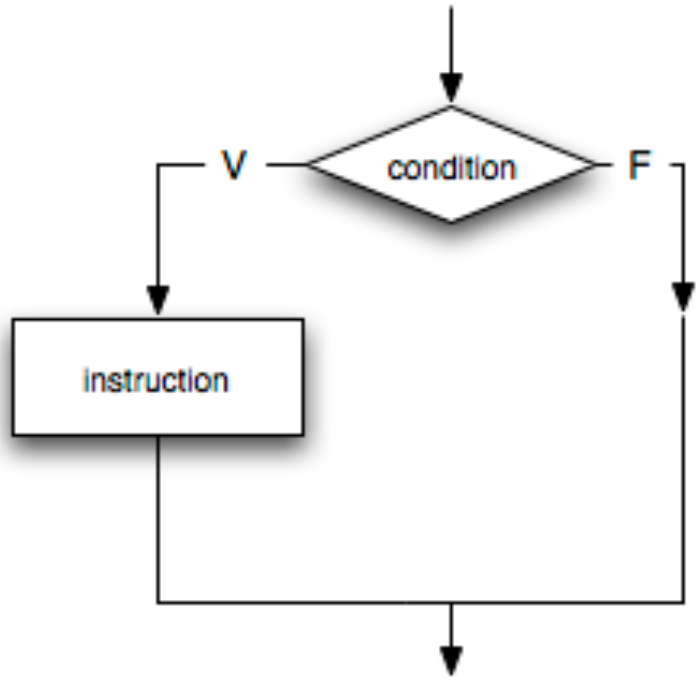
Une deuxième forme d'instruction `if` est disponible :

```
x = 2
if x % 2 == 0 :
  print('x est pair')
else :
  print('x est impair')
```

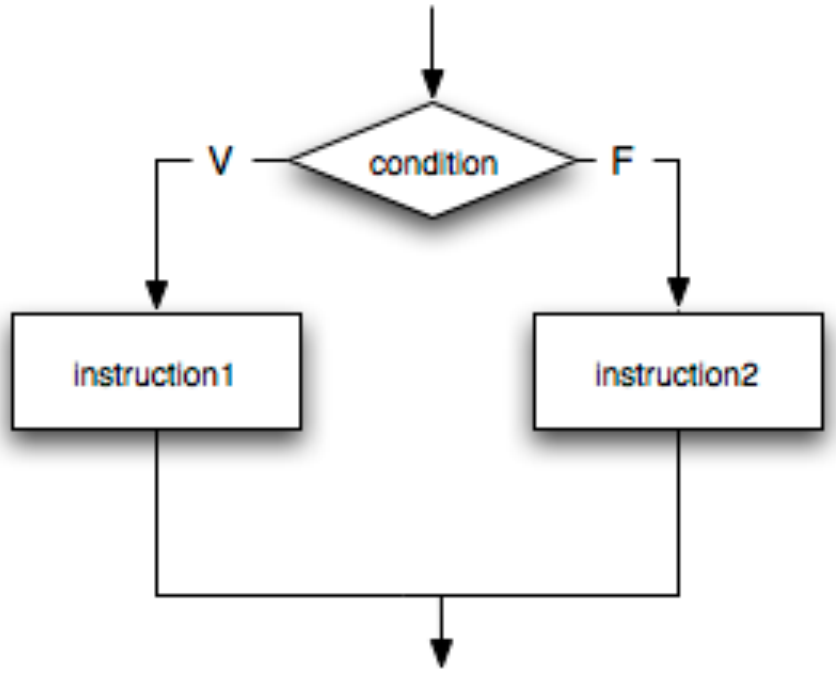
Dans ce cas, les instructions indentées après le `else` sont exécutées uniquement si la condition du `if` est fausse.

Tests simples

```
if condition:
  instruction
```




```
if condition:
  instruction1
else:
  instruction2
```



Test chaînés et imbriqués

On peut chaîner des conditions à l'aide de `elif` (else if) :

```
if x > y :
        print 'x est plus grand que y'
elif x < y :
        print 'x est plus petit que y'
else :
        print 'x est egal a y'
```

On peut également imbriquer des conditions à l'aide de l'indentation :

```
if x == y :
        print 'x est egal a y'
else :
        if x < y :
                print 'x est plus petit que y'
        else :
                print 'x est plus grand que y'
```

Composition de tests

Il y a trois opérateurs logiques : `and` (et), `or` (ou) et `not` (non).

On construit des expressions booléennes en utilisant les opérateurs de comparaison et logiques.

```
>>> x = 5
>>> 0 < x and x < 10
True
>>> x % 2 == 0 or x % 3 == 0
False
>>> not x > 10
True
```

En Python, il est possible de faire des comparaisons multiples :

```
>>> x = 5
>>> 0 < x < 10
True
```

Algèbre booléenne

- `a and b` est vrai si et seulement si `a` est vrai et `b` est vrai
- `a or b` est faux si et seulement si `a` est faux et `b` est faux

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Loi de De Morgan :

- `not(a and b)` est équivalent à `(not a) or (not b)`
- `not(a or b)` est équivalent à `(not a) and (not b)`

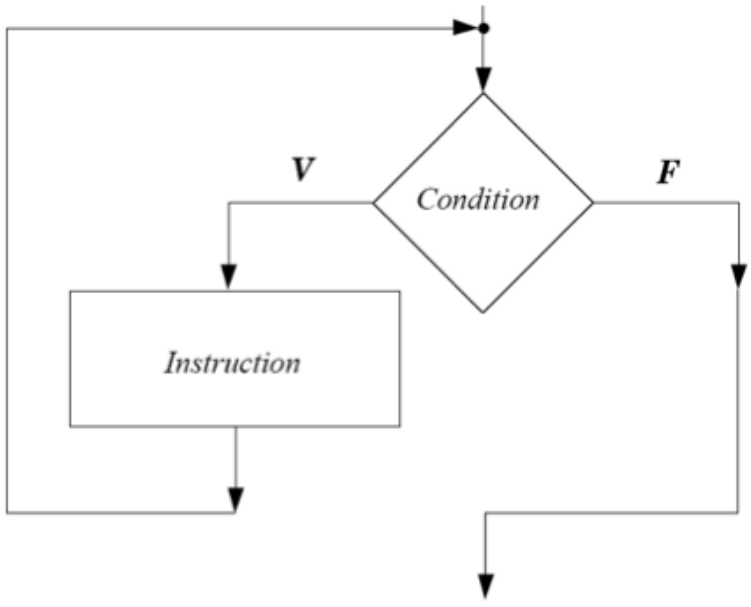
Exemple d'expressions équivalentes :

```
not(0 >= x or x >= 1000)
0 < x and x < 1000          #plus lisible
```

Boucle `while`

La structure itérative `while` permet de répéter un bloc d'instructions tant qu'une condition est vérifiée.

```
while condition:
    instructions
```



1. La condition est évaluée.
2. Si la condition est vérifiée, on exécute les instructions et on retourne au point 1.

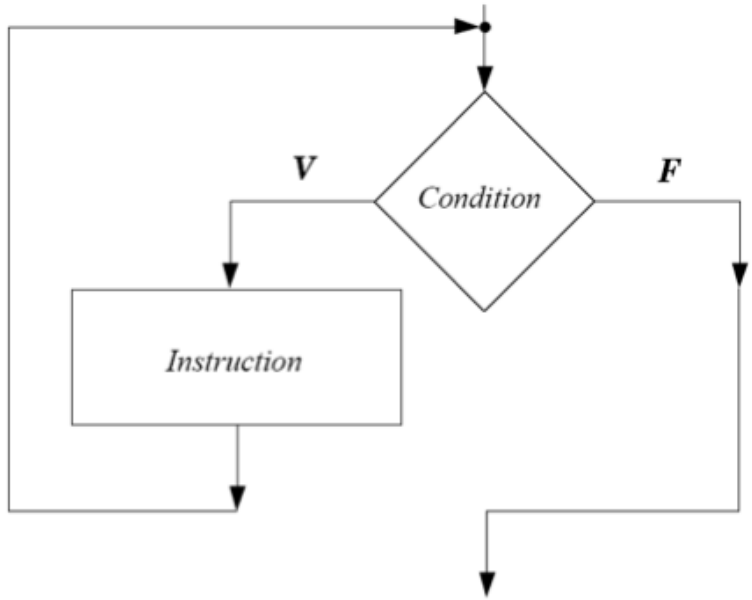
   Si la condition nest pas vérifiée on "sort" de la boucle.

Boucle `while` : exemple

```
x = 1

while x < 8:
    print(x,end="")
    x = x + 2

print(x)
```



Boucles `for`

```
>>> for lettre in 'Python':
...     print('Lettre considérée :', lettre)

Lettre considérée : P
Lettre considérée : y
Lettre considérée : t
Lettre considérée : h
Lettre considérée : o
Lettre considérée : n
```

```
>>> for i in range(3):
...     print(i)

0
1
2
```

Consultez la documentation pour trouver d'autres opérations.

Boucles `while` et `for`

```
for i in range(len(liste)):
        print(liste[i])
```

```
i = 0
while i < len(liste):
        print(liste[i])
        i += 1
```

Contrairement aux boucles `for` qui itèrent sur toute une séquence, les boucles `while` permettent d'arrêter les itérations lorsqu'une condition n'est plus vérifiée.

Prochain TP

- **fonctions, tuples**

Exercices

- Exercices 3 : Controle de flux.