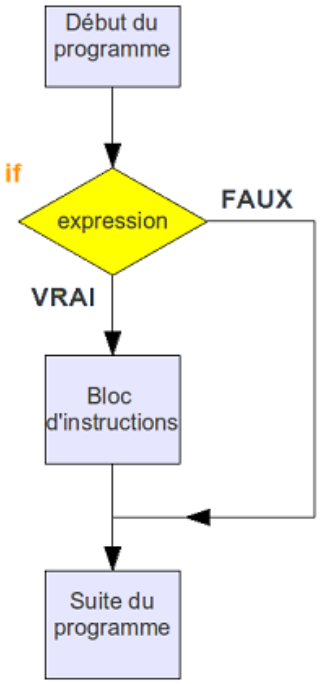
PYTHON – Tests

Condition minimale if

La condition if permet d'exécuter

- une ou plusieurs instructions dans un cas,
- d'autres instructions dans un autre cas



if va servir d'aiguillage et permettre au programme de suivre des chemins différents selon les circonstances

Exemple :

« *Allez tout droit jusqu'au prochain carrefour et là regardez à droite. Si la rue est autorisée à la circulation, alors prenez la. Mais si en revanche elle est en sens interdit, alors continuez jusqu'à la prochaine à droite* ».

Dans cet exemple, on prévoit, en fonction d'une situation donnée, deux façons différentes d'agir.

Cela suppose que l'on sache analyser la condition que nous avons fixée à son comportement *(« la rue est-elle en sens interdit ? »)* pour effectuer la série d'actions correspondante.

Cette structure logique est appelée **test**. On peut également parler de **structure conditionnelle**.

Syntaxe avec une condition

```
if booléen :
```

Un booléen est une expression dont la valeur est VRAI ou FAUX. Cela peut donc être :

- une variable (ou une expression) de type booléen
- une condition

Exemple avec une condition

```
if a > 0:
```

le test conditionnel se compose, dans l'ordre :

- du mot clé ***if*** qui signifie « si » en anglais ;
- de la **condition** proprement dite (***a > 0*** dans l'exemple) ; une condition est composée de trois éléments :
  - une valeur (a)
  - un opérateur de comparaison (>)
  - une autre valeur (0)
- du signe **deux points, « : »**, qui termine la condition et est indispensable : Python affichera une erreur de syntaxe si vous l'oubliez.

Syntaxe avec une instruction

```
if booléen :
    #instructions
```

Arrivé à la première ligne (if), la machine examine la valeur du booléen :

- Si ce booléen a pour valeur VRAI, elle exécute la série d'instructions. Cette série d'instructions peut être très brève comme très longue.
- En revanche, dans le cas où le booléen est faux, l'ordinateur saute directement aux instructions situées après la fin du bloc if.

Notez pour l'instant que l'**on décale le(s) instruction(s) par rapport au if**

Exemple avec une instruction

```
# Programme testant le if
a = 5
if a > 0:
        print("a est supérieur à 0.")
```

Code dans un fichier essai.py

- On affecte la valeur 5 à la variable a.
- On trouve le test conditionnel (if a > 0:)
- Dans le cas où a est supérieur à 0, on affiche le message « a est supérieur à 0 ». *Cette ligne est décalée d'une tabulation par rapport au if*

```
a est supérieur à 0.
>>>
```

Résultat dans la console

Syntaxe avec plusieurs instructions

précédemment, notre bloc n'était constitué que d'une seule instruction.

Rien ne nous empêche de mettre plusieurs instructions dans ce bloc, par exemple :

```
a = 5
b = 8
if a > 0:
    # On incrémente la valeur de b
    b += 1
    # On affiche les valeurs des variables
    print("a =", a ,"et b =", b)
```

Code dans un fichier essai.py

```
a = 5 et b = 9
>>>
```

Résultat dans la console

Indentation

On entend par indentation un certain décalage vers la droite, obtenu par un (ou plusieurs) espaces ou tabulations.

Les indentations sont essentielles pour Python. Il ne s'agit pas, comme dans d'autres langages tels que le C++ ou Java, d'un confort de lecture mais bien d'un **moyen pour l'interpréteur de savoir où se trouvent le début et la fin d'un bloc.**

```
a = 3
if ( a > 10 ) :
    a = a + 10
    a = 2 * a
print (a)
```

```
3
>>>
```

```
a = 3
if ( a > 10 ) :
    a = a + 10
a = 2 * a
print (a)
```

```
6
>>>
```

Les opérateurs de comparaison

Les conditions doivent nécessairement introduire de nouveaux opérateurs, dits **opérateurs de comparaison**

Opérateur	Signification littérale
<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

Si l'on veut que la comparaison ait un sens, il faut que les deux valeurs de la comparaison soient du même type.

Ces opérateurs de comparaison peuvent s'utiliser avec des nombres ou avec des caractères. Les caractères sont codés par la machine dans l'ordre alphabétique, les majuscules étant systématiquement placées avant les minuscules

Remarques:

- Ne pas confondre en Python : a == b qui correspond à l'égalité des deux variables a et b.
  avec a = b en Python qui correspond à a prend pour valeur b.

```
if (a==b):
    print("a est égal à b")
```

- a ≠ b en algorithmique sera codé en langage Python par a<>b ou bien a!= b.

Exemple

Une condition peut être aussi la comparaison de deux expressions comme :

1 > 2	FALSE
"t" < "w"	TRUE
"Maman" > "Papa"	FALSE
"maman" > "Papa"	TRUE

Une condition peut être aussi la comparaison de deux expressions comme :

```
(a + b - 3) x c  <=  (5 x y - 2) / 3
```

Condition complète if, else

Syntaxe

Le mot-clé **else**, qui signifie « sinon » en anglais, permet de définir une première forme de complément à notre instruction if

```
if booléen :
        #instructions 1
else:
        #instructions 2
#instructions 3
```

Dans le cas de la structure complète :

- Dans le cas où le booléen est VRAI, on exécute la série d'instructions 1. Au moment où on arrive au mot « else », la machine saute directement aux instructions situées après la fin du bloc if (instructions 3)
- Dans le cas où le booléen est FAUX, la machine saute directement à la première ligne située après le « else » et exécute l'ensemble des « instructions 2 ».
- Dans tous les cas, les instructions situées juste après le bloc if (instructions 3) seront exécutées normalement.

Exemple :

```
a = 5
if a > 0:
        print("a est supérieur à 0.")
else:
        print("a est inférieur ou égal à 0.")
```

Si a est strictement supérieur à 0, on affiche qu'il est positif ;

sinon, on affiche que a est nul ou inférieur à 0 .

```
a est supérieur à 0.
>>>
```

Résultat dans la console

Python exécute soit l'un, soit l'autre, et jamais les deux.

Notez 4 points importants sur cette instruction else:

- Elle doit se trouver au **même niveau d'indentation** que l'instruction **if** qu'elle complète.
- Elle **se termine** également par **deux points** puisqu'il s'agit d'une condition, même si elle est sous-entendue.
- Elle est **facultative**
- Elle ne peut figurer **qu'une fois**, clôturant le bloc de la condition. Deux instructions else dans une même condition ne sont pas envisageables et n'auraient de toute façon aucun sens.

Le mot-clé and

Il arrive souvent que nos conditions doivent tester plusieurs prédicats.

- le mot clé **and** qui signifie « et » en anglais, permet de tester plusieurs prédicat à la fois
- Le ET a le même sens en informatique que dans le langage courant. Pour que "Condition1 ET Condition2" soit VRAI, il faut impérativement que Condition1 soit VRAI et que Condition2 soit VRAI. Dans tous les autres cas, "Condition 1 et Condition2" sera faux.

```
a = 5
# on cherche à tester à la fois si a est supérieur ou égal à 2 et inférieur ou égal à 8
if a>=2 and a<=8:
   print("a est dans l'intervalle.")
else:
   print("a n'est pas dans l'intervalle.")
```

```
a est dans l'intervalle.
>>>
```

Résultat dans la console

Le mot-clé or

Il arrive souvent que nos conditions doivent tester plusieurs prédicats.

- le mot clé **or** qui signifie « ou » permet de tester un prédicat ou un autre prédicat
- Il faut se méfier un peu plus du OU. Pour que "Condition1 OU Condition2" soit VRAI, il suffit que Condition1 soit VRAIE ou que Condition2 soit VRAIE, ou que les deux conditions soient VRAIES.

```
a = 5
#on cherche à savoir si a n'est pas dans l'intervalle. La variable ne se trouve pas dans l'intervalle si elle est inférieure à 2 ou supérieure à 8
if a<2 or a>8:
   print("a n'est pas dans l'intervalle.")
else:
   print("a est dans l'intervalle.")
```

```
a est dans l'intervalle.
>>>
```

Résultat dans la console

Le mot-clé not

Le **not** inverse une condition : not(Condition1) est VRAI si Condition1 est FAUX, et il sera FAUX si Condition1 est VRAI.
C'est l'équivalent pour les booléens du signe "moins" que l'on place devant les nombres.

```
a = 5
#on cherche à savoir si a n'est pas inférieur à 2 -> donc on cherche à savoir si a est supérieur à 2 !
if not(a<2):
   print("a n'est pas dans l'intervalle.")
else:
   print("a est dans l'intervalle.")
```

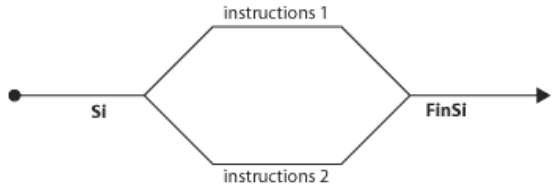
```
a n'est pas dans l'intervalle.
>>>
```

Résultat dans la console

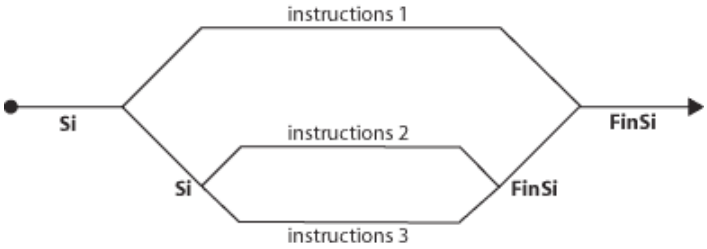
Tests imbriqués

Graphiquement, on peut très facilement représenter un SI comme un aiguillage de chemin de fer.

Un SI ouvre donc deux voies, correspondant à deux traitements différents.



Mais il y a des tas de situations où deux voies ne suffisent pas.



Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeuse).

Une première solution serait la suivante :

```
temp = input("Entrez la température de l'eau : ")
if temp <= 0:
        print("C'est de la glace ")
if temp > 0 and temp < 100:
        print("C'est de l'eau")
if temp > 100:
        print("C'est de la vapeur")
```

c'est un peu laborieux. On oblige la machine à examiner trois tests successifs alors que tous portent la température de l'eau.

Il serait plus rationnel **d'imbriquer les tests** de cette manière

```
temp = input("Entrez la température de l'eau : ")
if temp <= 0:
        print("C'est de la glace ")
else:
        if temp < 100:
                print("C'est de l'eau")
        else:
                print("C'est de la vapeur")
```

Nous avons fait des économies : au lieu de devoir taper trois conditions, dont une composée, nous n'avons plus que deux conditions simples.

Mais aussi, et surtout, nous avons fait des économies sur le temps d'exécution de l'ordinateur.

Si la température est inférieure à zéro, celui-ci écrit dorénavant « C'est de la glace » et passe directement à la fin, sans être ralenti par l'examen d'autres possibilités (qui sont forcément fausses).

Cette deuxième version n'est donc pas seulement plus simple à écrire et plus lisible, elle est également plus performante à l'exécution.

Les structures de tests imbriqués sont donc un outil indispensable à la simplification et à l'optimisation des programmes.

L'instruction elif:

il est également possible (mais non obligatoire), que le programme devienne :

```
temp = input("Entrez la température de l'eau : ")
if temp <= 0:
        print("C'est de la glace ")
elif temp < 100:
        print("C'est de l'eau")
else:
        print("C'est de la vapeur")
```

Dans le cas de tests imbriqués, **le Sinon (else) et le Si (if) peuvent être fusionnés en un SinonSi (elif)**. On peut ainsi enchaîner les elifi les uns derrière les autres pour simuler un aiguillage à autant de branches que l'on souhaite.

Le mot clé elif est une contraction de « else if », que l'on peut traduire très littéralement par « sinon si ».

```
a=0
if a > 0: # Positif
        print("a est positif.")
elif a < 0: # Négatif
        print("a est négatif.")
else: # Nul
        print("a est nul.")
```

```
a est nul.
>>>
```

Résultat dans la console

Notez 5 points importants sur cette instruction elif

- Elle est sur le même niveau **d'indentation** que le if initial.
- Elle se termine aussi par **deux points**.
- Entre le elif et les deux points se trouve une **nouvelle condition**
- Elle est **facultative**
- Vous pouvez mettre autant de elif que vous voulez après une condition en if.

Faut-il mettre un ET ? Faut-il mettre un OU ?

Dans une condition composée employant à la fois des opérateurs ET et des opérateurs OU, la présence de parenthèses possède une influence sur le résultat, tout comme dans le cas d'une expression numérique comportant des multiplications et des additions.

NB : S'il n'y a dans une condition que des ET, ou que des OU, en revanche, les parenthèses ne changent strictement rien.

```
X = int(input("saisissez un chiffre"))
A = X > 12
B = X > 2
C = X < 6
D = (A and B) or C
E = A and (B or C)
print( D, E )
```

```
D = (X > 12 and X > 2 ) or X < 6
E = X > 12 and (X > 2 or X < 6)
```

Si on remplace X par 3 : alors on remarque que D sera VRAI alors que E sera FAUX.