# La structure conditionnelle : `if`

## 1 Lien avec les variables booléennes

**Rappels**

- Une **variable booléenne** (de type `bool` en Python) ne peut prendre que deux valeurs : `True` et `False`.
- Une **expression** est un ensemble de variables (et/ou de valeurs), et d'opérateurs.
- Une **expression booléenne** est une expression renvoyant un booléen.

On utilise une expression booléenne pour **comparer différentes données** : différentes valeurs, stockées dans une variable ou bien dont on donne directement la valeur.

| Opérateur en Python | Signification |
|---|---|
| == | égal |
| != | différent |
| > | strictement supérieur |
| < | strictement inférieur |
| >= | supérieur ou égal |
| <= | inférieur ou égal |

**Exemple :** On considère que `a` vaut $7$, `nb_pommes` vaut $3$, `nombre1` vaut $25.8$ et `nombre2` vaut $137.2$.

| Expression en Python | Signification | Valeur |
|---|---|---|
| `a == 5` | | |
| `0 != nb_pommes` | | |
| `nombre1 <= nombre2` | | |
| `123 > 12` | | |

/!\ La réponse à ces tests sera toujours `True` ou `False` !

## 2 Les structures conditionnelles en programmation

On peut vouloir qu'un algorithme ne fasse pas les mêmes opérations en fonction des données du problème. On utilise alors l'instruction conditionnelle "Si... alors... sinon... alors...".

**Exemple :** On veut faire deviner le prix d'un objet à un utilisateur stocké dans une variable `prix_objet`. S'il rentre la bonne valeur dans la variable `prix`, on affiche qu'il a trouvé. Sinon, on affiche le contraire.

Voici l'algorithme utilisé en **pseudo-code** (ou encore en **langage naturel**) :

```
Si prix égal à prix_objet
Alors afficher "vous avez trouvé le prix"
Sinon afficher "vous n'avez pas trouvé"
```

On veut donner plus d'indications au joueur : on veut lui préciser si la valeur qu'il a entrée est mauvaise, inférieure ou supérieure à celle attendue.

Voici l'algorithme utilisé en **pseudo-code** (ou encore en **langage naturel**) :

```
Si prix égal à prix_objet
Alors afficher "vous avez trouvé le prix"
Sinon
    Si prix inférieur à prix_objet
    Alors fficher "la valeur est inférieure au prix de l'objet"
    Sinon afficher "la valeur est supérieure au prix de l'objet"
```

# 3 La structure conditionnelle `if` en Python

### Définition

La structure conditionnelle `if booleen` exécute un bloc d'instructions si le booléen `booleen` a pour valeur `True`.
Si le booléen `booleen` a pour valeur `False`, le programme continue **après** le bloc d'instructions.
La syntaxe en Python est la suivante :

```
if booleen:
    instruction_1
    instruction_2
    ...
suite_des_instructions
```

Le booléen `booleen` exprime une condition pour que le bloc d'instructions qui le suit soit exécuté. Cette condition s'obtient notamment par la comparaison de variables à l'aide des opérateurs vus précédemment :

- `a == c`: égalité entre les variables a et c ;
- `a != c`: non-égalité (équivalent à `not a == c`) ;
- a < c, a > c, a <= c, a >= c : opérateurs de comparaisons (inégalité stricte ou large).

On utilise également les mots-clef `else` (sinon) et `elif` (sinon si) comme illustré ci-dessous.

| À préférer ! | | Moins bien… |
|---|---|---|
| `if booleen :`<br>`  instructions_1`<br>`else:`<br>`  instructions_2` | $\Longleftrightarrow$ | `if booleen :`<br>`  instructions_1`<br>`if not ( booleen ):`<br>`  instructions_2` |
| `if booleen_1 :`<br>`  instructions_1`<br>`elif booleen_2 :`<br>`  instructions_2` | $\Longleftrightarrow$ | `if booleen_1 :`<br>`  instructions_1`<br>`else :`<br>`  if booleen_2 :`<br>`  instructions 2` |

### /!\ Syntaxe Python

La syntaxe Python est importante !

- Les " : " : indispensables pour marquer la fin de l'expression booléenne utilisée ;
- l'**indentation** est l'espacement par rapport au bord. Elle sert à indiquer quel bloc d'instructions est à exécuter uniquement si la condition est vraie, et quel bloc est à exécuter uniquement si la condition est fausse.

De manière générale en Python, toute **instruction indentée** est une sous-instruction de l'instruction moins indentée la précédent.

**Exercice 1** – *(sur papier)*

1. Remplir la table de trace du programme suivant :

```
1  seuil = 100
2  x = 124
3  if x > seuil:
4      x = seuil
```

| N° de ligne | seuil | x |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |

2. En déduire ce que fait ce programme.

**Exercice 2** – *(À faire sur papier puis à vérifier sur ordinateur)*

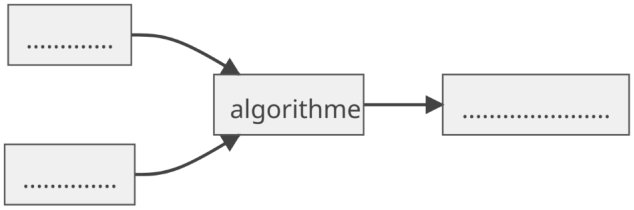
Qu'affiche le script ci-dessous ?

```
1  a = 5
2  b = 7
3  v = False
4  if a > b or v :
5      a = 8
6      v = True
7  v = not(v)
8  if a > b or v :
9      b = 9
10 print(a, b, v)
```

**Exercice 3** – *(Minimum et maximum. À faire sur papier puis à vérifier sur ordinateur)*

1. On veut écrire un programme qui stocke dans une variable `minimum` le minimum entre deux entiers `a` et `b` **différents l'un de l'autre** à l'aide d'une instruction conditionnelle.

   (a) Compléter les entrées et sorties du problème sur le schéma suivant.



(b) À l'aide d'une structure conditionnelle et d'une variable nommée `minimum`, compléter le programme suivant effectuant l'opération demandée.

```
a = 9
b = 2
...
```

(c) Quelle valeur contient la variable `minimum`, après exécution du programme ? (On peut s'aider de la fonction `print`.)

2. Écrire un script qui permette d'obtenir le maximum entre deux entiers.

3. Écrire une fonction `trouve_maximum` prenant en paramètres deux nombres `a` et `b` et renvoyant le plus grand des deux. Tester la fonction en l'appelant avec les valeurs précédentes et à l'aide de la fonction `print`.

**Exercice 4** – *(Devoir réussi ? À faire sur papier puis sur ordinateur)*

Écrire un programme qui :

- stocke la valeur 12 dans une variable `note` ;
- si cette valeur est strictement inférieure à 10, il affiche "Vous avez eu moins que la moyenne." ;
- sinon, il affiche "Vous avez eu plus que la moyenne".

Exécuter le programme.

Faire un 2ème test, en affectant la valeur 7 à `note`.

**Exercice 5** – *(Conditions multiples : triangles. À faire sur papier puis à vérifier sur ordinateur)*

1. Écrire une fonction `est_un_triangle` qui prend en paramètres 3 longueurs entières `a`, `b` et `c` et qui renvoie `True` si ces trois longueurs peuvent être les longueurs des trois côtés d'un triangle, `False` sinon.
2. Recopier votre fonction en lui ajoutant de la documentation sous forme de *docstring*. Rappel : un *docstring* se place sous la signature de la fonction, entre triple guillemets doubles.

   Exemple :

   Soit `x` est un entier et `y` une chaîne de caractères.

```
def toto(x, y):
    """ toto(int, str) -> int
    Phrase qui explique rapidement ce que fait la fonction."""

    # corps de la fonction
    return x
```