Python 2 : Structure if/else

1) La structure conditionnelle if/else

Si l'on veut un programme qui effectue des opérations uniquement si une condition est vérifiée, on peut utiliser la structure conditionnelle if.

Exemple :

```
x=float(input('rentrer un réel x')
if x>=0 :
        print('x est positif')
else :
        print('x est négatif')
```

En jaune se situe la condition, qui, si elle est remplie, va faire que le programme va exécuter les lignes en vert, en revanche si la condition n'est pas remplie, ce sont les lignes en rouge qui s'exécutent (celles après *else*, qui veut dire sinon en anglais).

Les espaces qui se situent avant les commandes en rouge et vert s'appellent **l'indentation** (sur l'ordinateur **on appuie une fois sur la commande tab**).

Notons que si l'on ne veut rien faire après le else, on peut ne pas écrire la commande else tout simplement (ce qui revient au même que d'écrire else suivi de x=x par exemple).

2) Importance de l'indentation

L'indentation est essentielle car elle permet de délimiter ce qui est exécuté de ce qui ne l'est pas, par exemple, comparons ces deux programmes :

```
x=float(input('rentrer un réel x'))
if x>0:
        x=x+1
        print(x)        #le print fait partie de l'instruction conditionnelle if


x=float(input('rentrer un réel x'))
if x>0:
        x=x+1
print(x)                # le print ne fait pas partie de l'instruction conditionnelle if
```

Le premier, lorsque l'on rentre 6, affiche 7, et lorsque l'on rentre -1, il n'affiche rien car -1 ne vérifie pas la condition donc toute la partie dans l'espace n'est pas exécutée.
Le second, lorsque l'on rentre 6 il affiche 7, et lorsque l'on rentre -1, il affiche -1, car le print est en dehors de l'indentation donc il est exécuté de toute manière.

3) Syntaxe d'une condition

Il faut faire attention de respecter la syntaxe quand on écrit une condition. Voici un tableau de correspondance.

Tests logiques	**Symboles en Python**
=	==
<	<
>	>
=/=	!=
≤	<=
≥	>=

Notamment on écrit : if x==0 : si on veut que la condition à vérifier soit x=0.
Le sens du symbole = dans Python est une affectation, le == permet juste de vérifier, on ne change pas la valeur de x.

Si on veut mettre plusieurs conditions, on peut utiliser les connecteurs logiques.

Connecteurs	**Python**
et	and
ou	or
non	not

Par exemple :
*x**2==2 or x==0* correspond à la condition x=2, x=-2 ou x=0, donc une boucle if avec cette condition va se lancer si x vaut l'une de ses trois valeurs.
x>y and x<z correspond à la condition : y<x<z

Notamment on a le programme :

```
if x<y and x<z :
        print('x est strictement plus petit que y et que z')
else :
        print('x est plus grand que y ou que z')
```

4) La commande elif

Si on veut faire plus que 2 cas, on peut utiliser la commande elif, voir l'exemple :

```
x=float(input('rentrer un réel x'))
if x>0 :
        print('x est strictement positif')
elif x==0:
        print('x est égal à 0')
else :
        print('x est strictement négatif')
```

5) Les booléens (pour ceux qui veulent aller plus loin)

Il existe un type de variable appelé booléen. Ce sont des variables qui valent ou bien True ou bien False (il n'y a donc que deux valeurs possibles).
On peut faire des opérations entre deux booléens, ce sont les connecteurs logiques vus en partie 3. Par exemple :

```
b=True
y=b and False
print(y)
```

Va afficher False, car True and False donne False.

Les règles sont les suivantes :

True and True = True	True or True = True
True and False = False	True or False = True
False and False = False	False or False = True
not True = False	not False = True

Le rôle de ces booléens est de fabriquer des conditions pour les boucles if (et while que nous verrons plus tard). En effet une boucle if se lance si le booléen (qui est la condition) est égale à True, elle ne se lance pas s'il vaut False.

On a par exemple le code :

```
b=True
if b :
        print('b vaut la valeur booléenne True')
else :
        print('ce message ne s'affichera pas)
```