L. n°2

Conditions, boucles et fonctions en Python

I. Les conditions — TP2 exos 1 à 6

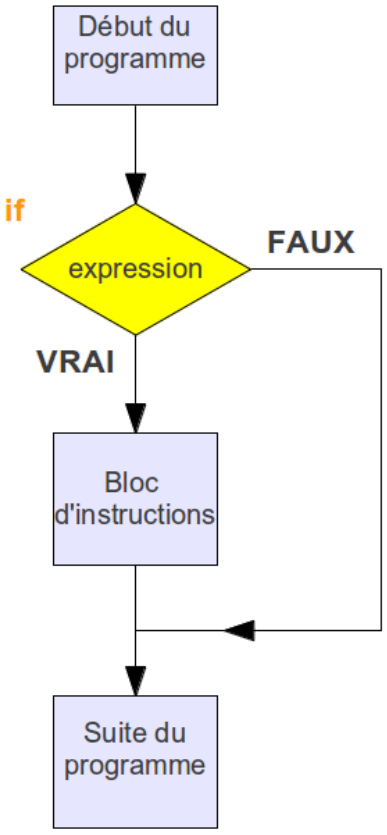
1. L'instruction *if*

Syntaxe :

```
if expression:                  # ne pas oublier le signe de ponctuation ':'
    bloc d'instructions         # attention à l'indentation
# suite du programme
```

➔ Si **l'expression est vraie** (`True`) alors **le bloc d'instructions** est exécuté.

➔ Si **l'expression est fausse** (`False`) on passe directement à la suite du programme



2. L'instruction *else*

Une instruction `else` est toujours associée à une instruction `if`.

Syntaxe :

```
if expression:
    bloc d'instructions 1              # attention à l'indentation
else:                                  # else est au même niveau que if
    bloc d'instructions 2              # attention à l'indentation
# suite du programme
```

➔ Si **l'expression est vraie** (`True`) alors le **bloc d'instructions 1** est exécuté.

➔ Si **l'expression est fausse** (`False`) alors c'est le **bloc d'instructions 2** qui est exécuté.

3. L'instruction *elif*

Une instruction `elif` (contraction de else et if) est toujours associée à une instruction `if`.

Syntaxe :

```
if expression 1:
    bloc d'instructions 1
elif expression 2:
    bloc d'instructions 2
elif expression 3:
    bloc d'instructions 3       # ici deux instructions elif, mais il n'y a pas
else:
    bloc d'instructions 4
# suite du programme
```

➔ Si **l'expression 1 est vraie** (`True`) alors le **bloc d'instructions 1** est exécuté et on passe à la suite du programme.

➔ Si **l'expression 1 est fausse** (`False`) alors on teste l'**expression 2** :

- si **l'expression 2 est vraie** on exécute le bloc d'instructions 2, et on passe à la suite du programme.
- **l'expression 2 est fausse** alors on teste l'**expression 3**, …, etc.

Le **bloc d'instructions 4** est donc exécuté si toutes les expressions précédentes sont fausses. C'est donc le bloc "par défaut".

Parfois il n'y a rien à faire. Dans ce cas, on peut omettre l'instruction `else`.

II. Les boucles TP3 exos 1 à 8

Une boucle permet d'exécuter une portion de code plusieurs fois de suite.

1. L'instruction *while*

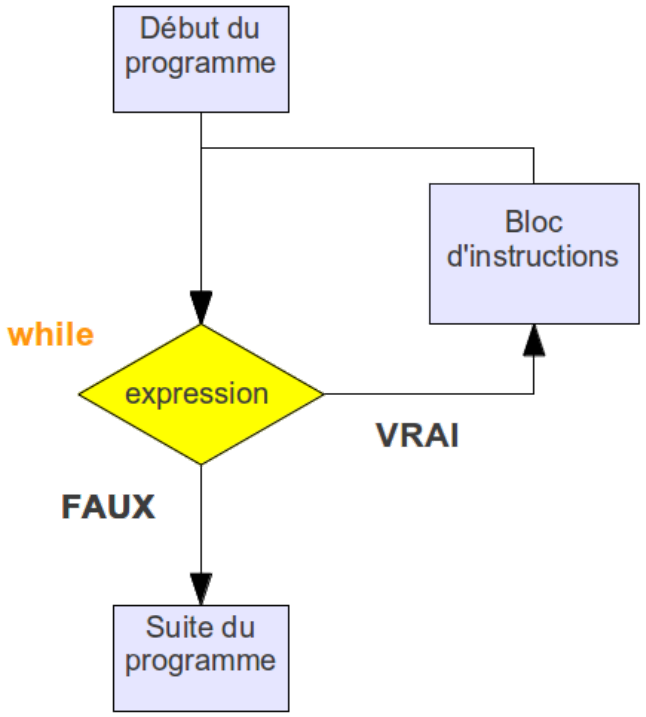
Syntaxe :

```
while expression:                # ne pas oublier le signe de ponctuation ':'
    bloc d'instructions          # attention à l'indentation
# suite du programme
```

➔ Si **l'expression est vraie** (`True`) le bloc d'instructions est exécuté, puis l'expression est à nouveau évaluée.

➔ Le cycle continue jusqu'à ce que **l'expression soit fausse** (`False`)

➔ On passe alors à la suite du programme.



2. L'instruction *for*

Syntaxe :

```
for élément in séquence :
    bloc d'instructions
# suite du programme
```

Les éléments de la séquence sont issus d'une chaîne de caractères ou bien d'une liste.

Astuce :
Si vous connaissez le nombre de boucles à effectuer, utiliser une boucle `for`.
Autrement, utiliser une boucle `while` mais attention à ne pas faire de boucle sans fin !

III. Les fonctions

Intérêt des fonctions :

Une fonction est une portion de code que l'on peut appeler au besoin (c'est une sorte de sous-programme).

L'utilisation des fonctions évite des **redondances** dans le code : on obtient ainsi des programmes plus courts et plus lisibles.

Syntaxe :

```
def NomDeLaFonction(parametre1,parametre2,parametre3,...):
    """ Documentation
    qu'on peut écrire
    sur plusieurs lignes """    # docstring entouré de 3 guillemets (ou apostrophes)

    bloc d'instructions         # attention à l'indentation

    return resultat             # la fonction retourne le contenu de la variable resultat
```