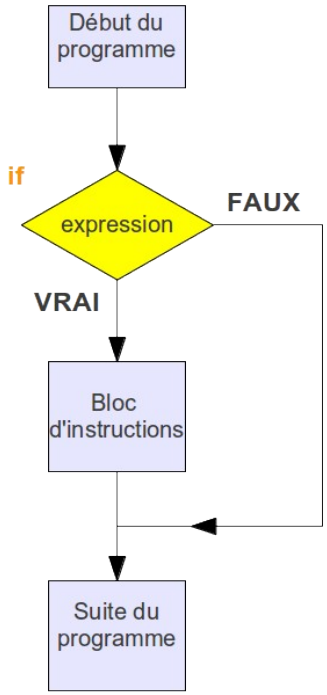
ACT3 : CONDITIONS, BOUCLES ET FONCTIONS EN PYTHON

I. INSTRUCTION CONDITIONNELLE

L'instruction conditionnelle **if** permet de soumettre un bloc de code à une condition.

Les conditions sont des expressions booléennes, vraie ou fausse, **True** ou **False**.



==	égal à	**!=**	différent de
<	plus petit que	**>**	plus grand que
< =	plus petit ou égal	**> =**	plus grand ou égal

Exemple 1 :

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note >= 10.0:
    # ce bloc est exécuté si (note >= 10.0) est vrai
    print("J'ai la moyenne")
else:
    # ce bloc est exécuté si (note >= 10.0) est faux
    print("C'est en dessous de la moyenne")
print("Fin du programme")
```

L'instruction **elif** (contraction de « else-if ») permet d'enchaîner les conditions. Pour traiter plusieurs cas, **elif** est préférable à une succession de **if**.

Exemple 2 : Comparer les cas...

```
# Avec des if, saisie de « 16 »

chaine = input("Note sur 20 : ")
note = float(chaine)
if note >= 15.0:
    print("Très bien !!")
if note >= 10.0:
    print("Au dessus de la moyenne !")
if note <= 10.0:
    print("En dessous de la moyenne")
print("Fin du programme")
```

```
# Avec des elif, saisie de « 16 »

chaine = input("Note sur 20 : ")
note = float(chaine)
if note >= 15.0:
    print("Très bien !!")
elif note >= 10.0:
    print("Au dessus de la moyenne !")
else:
    print("En dessous de la moyenne")
print("Fin du programme")
```

Exercice 1 :

Ecrire un script qui donne la mention obtenue lorsqu'on saisie la note moyenne au bac.

Exercice 2 :

Ecrire un script qui demande un entier n ($n \leq 255$) et qui donne le nombre de bits nécessaires pour le coder.

Exercice 3 :

Ecrire un script qui donne les solution de l'équation $ax^2 + bx + c = 0$

On utilisera la racine carrée de la bibliothèque math : *from math import sqrt* en première ligne du programme, puis on peut utiliser *sqrt(x)* pour la racine carrée de x.

II. BOUCLES

Une boucle est un bloc d'instructions qui se répète un certain nombre de fois.

On utilise deux types de boucles : les boucles bornées **for** (« pour ») et les boucles non-bornées **while** (« tant que »).

FOR

La boucle **for** exécute un certain nombre de fois le bloc d'instruction, en faisant varier un compteur (*variable d'itération*) dans un intervalle.

L'intervalle est un intervalle d'entiers et se note **range** :

- **range(5)** → 0, 1, 2, 3, 4 *« 5 valeurs »*
- **range(2, 5)** → 2, 3, 4 *« 3 valeurs »*
- **range(0, 5, 2)** → 0, 2, 4 *« 3 valeurs, par pas de 2 »*

Exemple 3 : for

```
chaine = 'Bonsoir'
for lettre in chaine:
    # lettre est la variable d'itération
    print(lettre)
```

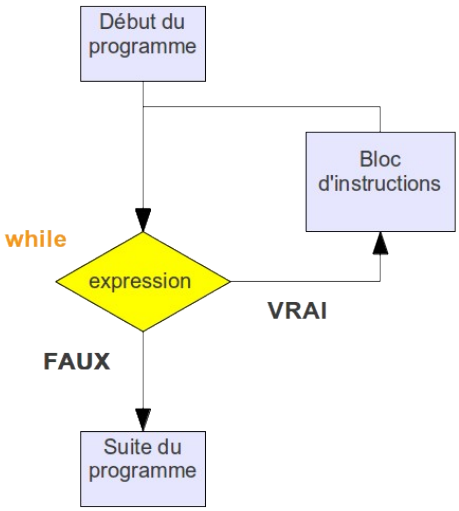
Exemple 4 : for

```
for i in range(1, 5) :
    print(i)
print("Fin de la boucle")
```

WHILE

La boucle **while** nécessite une expression booléenne qui vaut **True** pour que le bloc d'instruction s'exécute.

Lorsque cette expression vaut **False**, on passe à la suite du programme.



Exemple 5 : while

```
compteur = 1   # initialisation de la variable de comptage
while compteur < 5:
    # ce bloc est exécuté tant que la condition (compteur < 5) est vraie
    print(compteur, compteur < 5)
    compteur += 1    # incrémentation du compteur, compteur = compteur + 1
print(compteur < 5)
print("Fin de la boucle")
```

Exercice 4 :

Ecrire un script qui affiche une table de multiplication, avec **for** et avec **while**.

Exercice 5 :

1) Ecrire un script qui demande un nombre binaire et qui le convertit en décimal, avec **for**. Un *input()* renvoit une chaîne de caractères, pour transformer un caractère en entier, on utilise *int()*.
2) Sur le même principe, mais en base 5.

Exercice 6 :

1) Ecrire un script qui choisit un nombre au hasard entre 1 et 100 et qui demande à l'utilisateur de trouver ce nombre.

   On utilisera *randint()* de la bibliothèque random : *from random import randint* en première ligne du programme, puis on peut utiliser *randint(x, y)* pour générer un nombre entier aléatoire entre x et y inclus.
2) Ajouter un nombre de coups limite.

III. LES 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.

```
def nom_de_la_fonction(parametre1, parametre2, parametre3, ...):
    """ Documentation qu'on peut écrire
    sur plusieurs lignes """

    bloc d'instructions        # attention à l'indentation
    return resultat
```

Exemple 6 : Farenheit

```
def fahrenheit(degre_celsius):
    """ Conversion degré Celsius en degré Fahrenheit """
    return degre_celsius*9.0/5.0 + 32.0

print(fahrenheit(100))
   # affiche le résultat 212.0
```

Exercice 7 :

1) Ecrire une fonction **max2(a, b)** qui renvoie le plus grand de 2 entiers a et b.
2) Utiliser la fonction précédente pour écrire une fonction **max3(a, b, c)** qui renvoie le plus de 3 entiers a, b et c.

Exercice 8 :

Ecrire une fonction (procédure) **table(a)** permettant d'afficher la table de multiplication de a.

Exercice 9 :

Ecrire une fonction **pythagore()** qui demande les longueurs des 3 côtés d'un triangle est qui précise si ce triangle est rectangle ou pas : Vrai ou Faux.

Exercice 10 :

On appelle *factorielle n* (n entier naturel supérieur à 1) le nombre 1x2x3x...xn.
Ecrire une fonction **factorielle(n)** permettant de calculer *factorielle n*.
(Attention, ne tester que des nombres inférieurs à 10)