- Langages et Programmation -

CONTRÔLE DU FLUX D'EXÉCUTION AVEC PYTHON

Plan du chapitre

I. INTRODUCTION

Jusqu'à présent, nous avons vu des algorithmes simples qui se déroulent en **séquence** de la première à la dernière instruction. Cependant, ces instructions en séquence ne suffisent pas à exécuter des algorithmes plus complexes où :

- Certaines séquences d'instructions ne peuvent être exécutées que sous certaines conditions : **structures conditionnelles (if… else)**
- Certaines séquences d'instructions nécessitent d'être exécutées un certain nombre de fois : **structures itératives (boucle for** et **boucle while)**

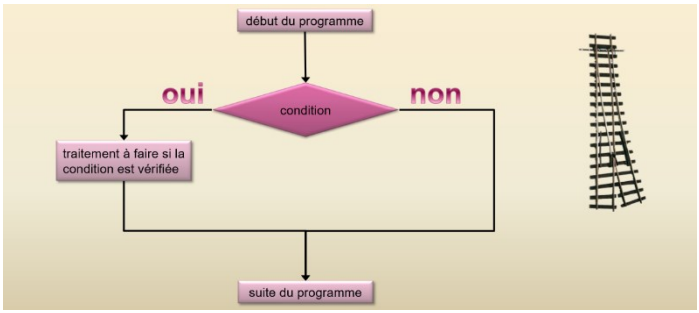
Le chemin suivi par le programme est appelé **le flux d'exécution** et les instructions qui le modifient sont appelées **les instructions de contrôle de flux.**

Retour au plan

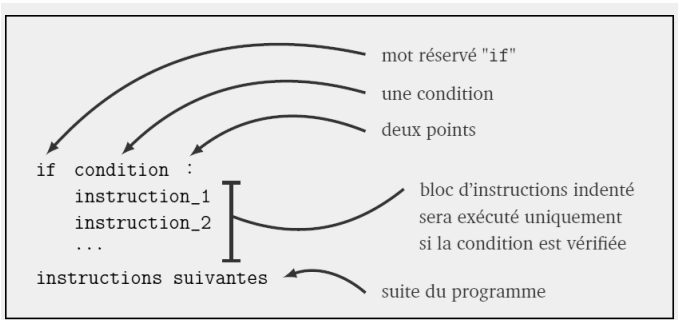
II. LES STRUCTURES CONDITIONNELLES

L'instruction conditionnelle *Si... (if...)*

Dans certains cas, on ne veut exécuter une instruction que sous une condition.



Syntaxe :



Après avoir tapé *if(condition)* et appuyé sur entrée, l'éditeur va alors automatiquement **indenter** (décalage vers la droite) les lignes d'instructions suivantes. **En effet, toutes les instructions qui doivent être effectuées si la condition est vraie doivent être indentées, et elles doivent avoir impérativement la même indentation.**

Exemple : le programme suivant affiche *"candidat reçu"* mais uniquement que celui-ci ait une moyenne supérieure ou égale à 10 :

```
1 m = float(input('Moyenne au bac = '))
2 if m>=10 :
3     print ('candidat reçu')
4 print ('programme terminé')
```

Retour au plan

L'évaluation de la condition

En fait la condition évaluée m>=10 renvoie un booléen : True (vrai) ou False (faux). Il est possible de le vérifier en rajoutant l'instruction suivante en fin de programme :

```
1 m = float(input('Moyenne au bac = '))
2 boolean = m>=10
3 if boolean :
4     print ('candidat reçu')
5 print ('la condition évaluée était : ', boolean)
6 print ('boolean est bien de type : ',type(boolean))
```

Retour au plan

Opérateurs de comparaison et opérateurs logiques

Afin d'écrire une condition, on utilise **les opérateurs de comparaison** suivants :

opérateur de comparaison	signification
==	égal
!=	différent
<=	inférieur ou égal
<	strictement inférieur
>=	supérieur ou égal
>	strictement supérieur

Attention : la comparaison *(a égal à 4)* s'écrit bien *(a==4)* et non pas *(a=4)* qui est l'affectation de la valeur 4 à la variable a.

Ces opérateurs de comparaisons peuvent être combinés aux **opérateurs logiques (ou opérateurs booléens)** suivants :

expression logique	rôle
a and b	Vraie si a **et** b sont vraies
a or b	vraie si a **ou** b (ou les deux) sont vraies
not(a)	si a est vraie, not(a) est fausse, et inversement

Exemple : le programme suivant détermine si le candidat aura une mention BIEN. Pour cela sa note doit être supérieure ou égale à 12 **et** inférieure strictement à 14 :

```
m = float(input('Moyenne au bac = '))
if m>= 12 and m<14 :
    print ('mention BIEN')
```

Là encore, l'expression évaluée : *m>= 10 and m<12* renvoie un booléen :

```
m = float(input('Moyenne au bac = '))
boolean = m>= 12 and m<14
if boolean :
    print ('mention BIEN')
print ('la condition évaluée était : ', m>= 10 and m<12)
print ('boolean est bien de type : ', type(boolean))
```

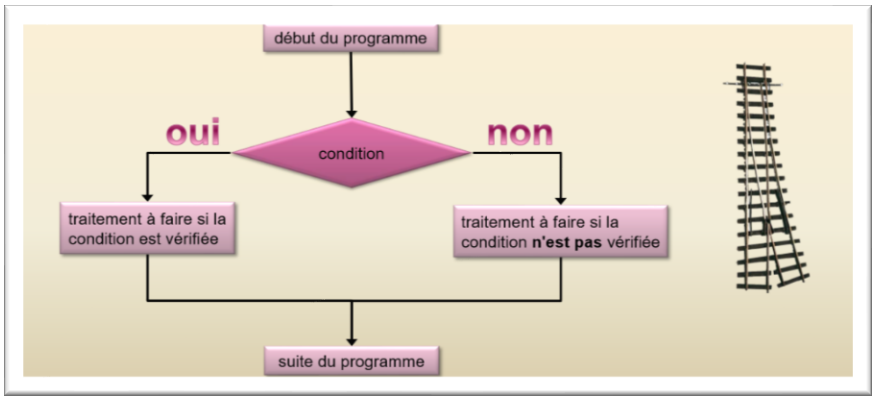
Remarque n°1 : En python (contrairement au langage Java par exemple), la condition *if (m>= 12 and m<14)* : peut être remplacée par la condition *if(12 <= m <140*:

Remarque n°2 : En python, la condition évaluée n'est pas obligatoirement entre parenthèses, mais dans le cas d'expressions complexes, les parenthèses sont vivement conseillées.

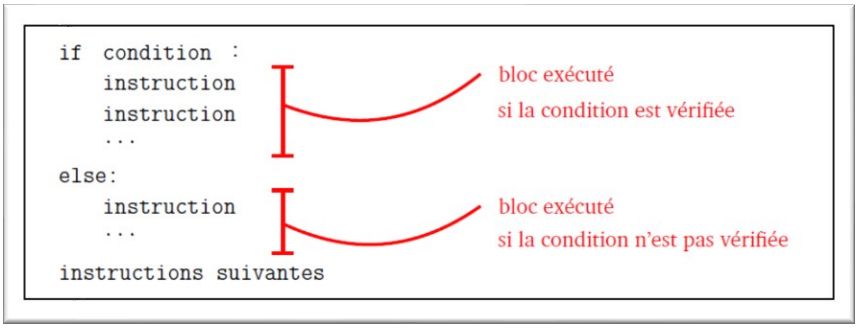
Retour au plan

L'*instruction conditionnelle Si… … Sinon (if… else…)*

Dans certains cas on ne veut exécuter une séquence d'instructions que si la condition évaluée est vraie et exécuter une autre séquence d'instruction si la condition évaluée est fausse.



Syntaxe :



Exemple : le programme renvoie systématiquement une réponse : *"candidat reçu"* (si condition évaluée vraie) sinon *"candidat non reçu"* si condition évaluée fausse) :

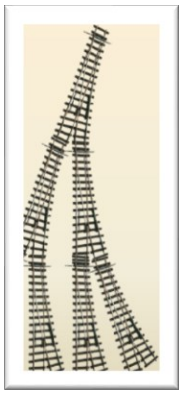
```
m = float(input('Moyenne au bac = '))
if m>=10 :
    print ('candidat reçu')
else :
    print ('candidat non reçu')
print ('programme terminé')
```

Retour au plan

Tests conditionnels imbriqués

Il est possible d'imbriquer un test conditionnel à l'intérieur d'un autre test :

```
m = float(input('Moyenne au bac = '))
if m>=10 :
    print ('candidat reçu')
    if 12 <= m <14 :
        print ('mention ASSEZ BIEN')
    if 14 <= m <16 :
        print ('mention BIEN')
    if m>=16 :
        print ('mention TRÈS BIEN')
else :
    if 8 <= m < 10 :
        print ('candidat autorisé à passer les épreuves du 2nd groupe')
    else :
        print ('candidat non reçu')
print ('programme terminé')
```



Des tests successifs peuvent également être réalisés avec l'instruction *elif (else if)*

```
m = float(input('Moyenne au bac = '))
if m>=16 :
    print ('candidat reçu mention TRES BIEN')
elif m>= 14 :
    print ('candidat reçu mention BIEN')
elif m>= 12 :
    print ('candidat reçu mention ASSEZ BIEN')
elif m>= 10 :
    print ('candidat reçu mention PASSABLE')
elif m>= 8 :
    print ('candidat autorisé à passer les épreuves du 2nd groupe')
else :
    print ('candidat non reçu')
print ('programme terminé')
```

Retour au plan

Tester autre chose que des nombres

Il est possible de tester autre chose que des nombres, par exemple des caractères

```
caract = input('caractère = ')
if  caract == 'H' :
    print ('lettre H')
else :
    print ('lettre autre que H')
print ('programme terminé')
```

Il est même possible de savoir si la lettre tapée se trouve avant ou après une lettre donnée dans l'alphabet :

```
caract = input('caractère = ')
if  caract == 'H' :
    print ('lettre H')
elif  caract > 'H' :
    print ('la lettre se trouve après la lettre  H')
else :
    print ('la lettre se trouve avant la lettre  H')
print('programme terminé')
```

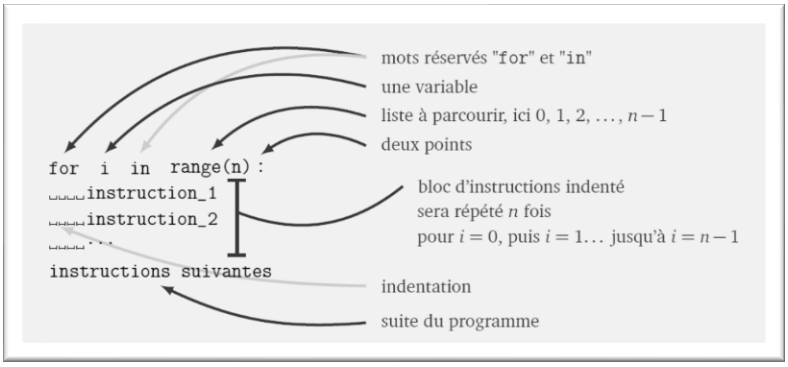
Retour au plan

III. LES STRUCTURES ITÉRATIVES

La boucle for (boucle inconditionnelle bornée)

Les traitements informatiques nécessitent souvent l'exécution de tâches répétitives.
La boucle *for* permet d'effectuer **un nombre de fois bien déterminé le corps de boucle (séquence d'instructions).** Elle s'utilise généralement avec la fonction range qui permet de générer une séquence de nombres **entiers** allant de **0 à n exclu.**

Syntaxe :



Après avoir tapé *for i in range(n)* et appuyé sur entrée, l'éditeur va alors automatiquement **indenter** les lignes d'instructions suivantes. En effet, toutes les instructions à répéter doivent être indentées, et elles doivent avoir impérativement la même indentation.

L'instruction suivante permet de voir la liste des nombres générés par la fonction *range()* :

```
print(list(range(10)))
```

Exemple n°1 : Le programme suivant donne la table de multiplication par n :

```
n=int(input("Quelle table de multiplication souhaitez-vous ? "))
for i in range (11) :
    print (i, ' x ', n, ' = ', i*n)
```

La fonction range peut accepter 3 arguments : *range (valeur_début, valeur_fin, pas).* Les 3 arguments *(valeur_début, valeur_fin, pas)* sont obligatoirement des entiers !

Exemple n°2 : Le programme suivant affiche la liste des nombres pairs allant de 100 à 120 :

```
for i in range(100, 121, 2) :
    print(i)
```

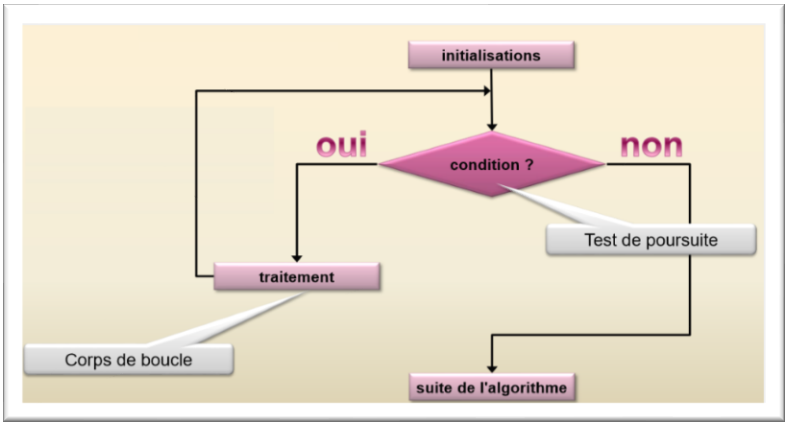
Il est également possible de faire parcourir à la boucle une liste quelconque :

```
for i in [10, 13, 15, 17,25] :
    print (i)
```

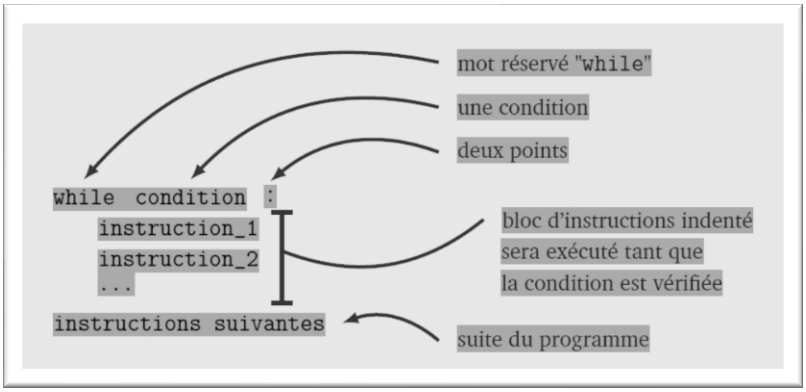
Retour au plan

La boucle while (tant que) (boucle conditionnelle non bornée)

La boucle *while* est assez peu différente de la boucle for. On l'utilise lorsque l'on ne sait a priori pas combien de boucles vont être effectuées. On l'utilise également lorsque l'on veut réaliser des incrémentations avec un pas qui ne soit pas entier, par exemple lors d'un tracé de courbe.
La boucle « tant que » exécute le corps de boucle (instructions indentées) tant qu'une condition évaluée dans le test de poursuite est vraie. Dès que la condition devient fausse, elle passe aux instructions suivantes.



Syntaxe :



Attention : lorsque que l'on utilise une boucle *while*, il faut s'assurer que la condition évaluée dans le test de poursuite deviendra fausse à un moment ou un autre, sinon … le programme va boucler indéfiniment !

Exemple : Le programme suivant indique tous les entiers x dont le carré est inférieur à un nombre n donné et indique également leur nombre (0 compris).

```
n= float(input("nombre n = "))
x=0
while (x**2 <= n) :
    print (x)
    x=x+1           # ou    x +=1
print ("Il y a ",x," nombres entiers dont le carré est inférieur ou égal à ",n)
```

Retour au plan

Imbrication de boucles

Il est possible d'imbriquer des boucles, c'est-à-dire que dans le bloc d'une boucle (boucle extérieure), on utilise une nouvelle boucle (boucle intérieure).

Exemple : Il est possible de modifier le programme de la table de multiplication vu précédemment afin qu'il affiche toutes les tables de multiplication de 1 jusqu'à 10. Pour cela, il suffit d'imbriquer le programme précédent à l'intérieur d'une boucle qui fait varier n de 1 à 10

```
for n in range (1,11) :
    print("table de multiplication par ",n)
    for i in range (11) :
        print (i, ' x ', n, ' = ', i*n)
    print("")
```

Question : combien d'itérations (une exécution du corps de boucle) vont être effectuées ?

Retour au plan

Parcours d'une chaîne de caractères.

- Solution directe : parcours des éléments

Une chaine de caractères (type string) est composée de … caractères ! Il est possible de parcourir ces caractères avec une structure *for… in…*

Exemple : le programme suivant demande une chaîne de caractères et insère un "." entre chaque caractère :

```
chaine = input("Chaine de caractères =  ")
for i in chaine :
    print (i,end=".")    # end="." substitue un point au retour à la ligne par défaut
```

- Solution indirecte : parcours des indices

Il est possible d'accéder à chaque caractère d'une chaîne à l'aide de son indice de position. **Le premier caractère est toujours indexé 0.**

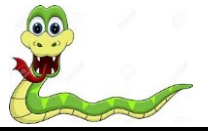
Exemple : pour afficher le 1er et 5ème caractère d'une chaîne :

```
chaine = "NUMERIQUE"
print("Le 1er caractere est ",chaine[0])
print("Le 5eme caractere est ",chaine[4])
```

Pour balayer l'ensemble des caractères de la chaîne de caractères :

```
chaine = input("chaine de caracteres = ")
for i in range(len(chaine)) :          # la fonction len() retourne la longueur de la chaine
    print (chaine[i], end=".")
```

Retour au plan



IV. EXERCICES

Exercice n°1 :

Réaliser un programme qui simule un lancer de 3 dés.
Le programme devra :

- Afficher le résultat du lancer ;
- Indiquer s'il y a des doubles et si oui, indiquer la valeur ;
- Indiquer s'il y a un triple et si oui, indiquer la valeur.

Exercice n°2 :

Résoudre à l'aide d'un programme le problème suivant :
Une balle est lâchée d'une hauteur initiale h et rebondit à chaque fois au 2/3 de sa hauteur. On considère qu'elle ne rebondit plus à partir d'une hauteur de 5 cm. Calculer le nombre de rebonds de la balle

Exercice n°3 :

A l'aide d'un programme, vérifier que la probabilité de tirer un 6 lors d'un jeter de dé est bien de 1/6.

Exercice n°4 :

Réaliser un programme qui demande une chaîne de caractères et une lettre et indiquera le nombre de fois que la lettre à été rencontrée dans la chaîne de caractères.

Exercice n°5 :

Réaliser un programme qui demande un mot et détermine si oui ou non ce mot est un palindrome, c'est-à-dire qu'il peut se lire indifféremment dans les 2 sens, par exemple radar, S.O.S, …

Exercice n°6 :

Réaliser un programme qui demande une chaîne de caractères et la recopie dans une nouvelle variable en l'inversant.

Retour au plan