

Chapitre 10 : Boucles Imbriquées

Boucles avec itérateur numérique

Boucles utilisées lorsque le **nombre d'itérations est connu (ou calculé)** avant l'entrée dans la boucle (pour exemple, un parcours complet de tableau) : - l'itérateur numérique est de type entier et doit être initialisé avant l'entrée dans la boucle - la condition d'arrêt porte sur l'itérateur - l'itérateur est incrémenté (ou décrémenté) par pas entier

Boucles "conditionnelles"

Boucles dont le **nombre d'itérations n'est pas connu à l'avance** (par exemple, le parcours partiel d'un tableau).

Ces deux types de boucles peuvent toujours se traduire par un while

Boucles imbriquées

Deux boucles sont imbriquées si une boucle est incluse dans une autre. À chaque itération de la boucle englobante, la boucle interne est exécutée.

Exemple :

```
In [ ]: n=10
        m=5
        i=0 # Initialisation de l'itérateur de la boucle externe
        # Boucle externe
        while(i<n):
            j=0 # Initialisation de l'itérateur de la boucle interne
            # Boucle interne
            while(j<m):
                print(i, ' ', j)
                j+=1
            i+=1
```

L'instruction `print(i, ' ', j)` est exécutée 50 fois : il y a 10 itérations de la boucle englobante (i varie de 0 à 9), et pour chacune de ces itérations la boucle interne (`while(j<m)`) est exécutée 5 fois (j varie de 0 à 4).

Remarque : pour faciliter l'écriture et la lecture d'un programme, les boucles internes peuvent être remplacées par un appel de fonction.

```
In [ ]: def affich(i,m) :  
        """ Affiche i, m fois sous la forme """  
        j=0  
        while(j<m):  
            print(str(i)+' '+str(j))  
            j+=1  
  
        # programme principal  
        n=10  
        m=5  
        i=0  
        while(i<n):  
            affich(i,m)  
            i+=1
```