Boucles while et for

En informatique, une boucle est une instruction qui permet de répéter un certain nombre de fois une suite d'opérations. Il existe deux types de boucles : les boucles for et les boucles while.

Boucle While

```
>>> while condition1 :
>>>     première ligne du bloc d'instructions 1
>>>     …
>>>     dernière ligne du bloc d'instructions 1
>>> suite du programme
```

Dans la première ligne, le mot-clé while permet au compilateur d'identifier le début d'une instruction conditionnelle. Ensuite il évalue la condition1. Si cette condition a la valeur vraie, le compilateur effectue le bloc d'instructions 1 puis réévalue la condition1. Sinon, il sort de la boucle et enchaîne avec la suite du programme. Ainsi le programme effectue le bloc d'instructions « tant que » la condition1 est vraie.

Comme pour l'instruction conditionnelle, la boucle est une instruction structurée. Les deux points annoncent le début du bloc d'instructions qui doit se trouver en retrait par rapport au reste du programme. Le retrait doit être le même à chaque ligne. Quand il n'y a plus de retrait, le compilateur sait que le bloc est terminé.

Chaque fois que le compilateur effectue le bloc d'instructions, on dit qu'il a effectué une itération.

Le principal danger dans ce type de boucle est que la condition1 ne prenne jamais la valeur « faux ». Dans ce cas, l'algorithme ne se termine pas. Le bloc d'instructions doit donc modifier l'environnement pour rendre la condition1 fausse au bout d'un temps fini.

Attention : la variable évaluée dans la condition doit exister au départ. Il faut qu'on lui ait déjà affecté une valeur au préalable. On dit qu'on initialise la variable.

Si la condition est fausse dès le départ, le corps de boucle n'est jamais effectué.

Il existe deux cas pour décider de la fin d'une boucle selon que l'on connaît ou pas à l'avance le nombre d'itérations à effectuer.

Dans le premier cas, la condition portera sur une variable de boucle, comme n dans l'exemple suivant :

```
>>> a=2 ; n=1 ;
>>> while n<5 :
>>>     a=a+2
>>>     n=n+1
>>> print(a)
```

Dans le cas où on ne sait pas à l'avance combien d'itérations vont être effectuées, il faudra être certain que la condition est fausse au bout d'un temps fini, comme dans l'exemple suivant :

```
>>> a= input('entrez un nombre entier naturel strictement supérieur à
1 :')
>>> x=int(a)
>>> n=1 ; P=1
>>> while P<1000 :
>>>     P=P*x
>>>     n=n+1
>>> print('la puissance', n-1,'-ème de', a , ' est supérieure à 1000')
```

Exercices :

1) Écrire un programme qui affiche les 20 premiers termes de la table de multiplication par 4 en signalant au passage (à l'aide d'un astérisque) ceux qui sont des multiples de 6 (sous la forme `4 ; 8 ; 12 * ;`...)(on pourra utiliser l'argument `end=' '` dans la fonction `print`, pour ne pas que le compilateur passe à la ligne).

2) Écrire un programme qui demande un entier n et un réel x et qui calcule la puissance n-ème de x.

3) Écrire un programme qui demande un entier naturel n et qui calcule n!

4) Écrire un programme qui génère un nombre aléatoire entre 1 et 100 puis qui fait deviner ce nombre à l'utilisateur en lui demandant des propositions et en indiquant si la proposition est supérieure, inférieure ou égale au nombre à deviner (pour générer un nombre aléatoire entre n et m, il faut importer le module `random` et utiliser l'instruction `randrange(n,m)`).

5) Écrire un programme qui demande à l'utilisateur un nombre n et qui calcule la somme des entiers naturels de 1 à n (en utilisant une boucle).

6) Écrire un programme qui demande à l'utilisateur un entier n et qui calcule les n premiers terme de la suite définie par $u_0=1$ et $u_{(n+1)}=3u_n+2$ pour tout n entier naturel.

Boucle for

La boucle for est utilisée quand on veut répéter un bloc d'instructions un certain nombre de fois connu à l'avance (fixé par le programmeur ou par l'utilisateur).

```
>>> for <indice> in <liste> :
>>>     <bloc d'instructions>
>>>...
```

Comme toute instruction structurées, ne pas oublier les deux points et les indentations.

Pour générer une liste de nombre allant de 0 à n (où n est un entier naturel), on utilisera la fonction `range()`.

Exercices :

1) Écrire un programme permettant de calculer une moyenne. Il demande à l'utilisateur combien il a de nombres dans sa liste. Puis il demande successivement tous les nombres de la liste et renvoie la moyenne.

2) Réécrire le programme de l'exercice 3 précédent, en utilisant une boucle for.