

Boucle non bornée while

On utilise généralement la boucle **while** lorsqu'on souhaite répéter un nombre de fois une même instruction et qu'on ne sait pas combien de fois cette instruction va être répétée.

On connaît alors une condition d'arrêt, c'est-à-dire un test qui permet de savoir si l'instruction va être répétée ou non.

Par exemple, si on veut ajouter 1 à un nombre tant que ce nombre est inférieur à 50, on utilisera une boucle **while** : *"TANT QUE nombre < 50, ajouter 1 à nombre"*.

En Python, la structure de la boucle **while** est la suivante :

```
while condition:  
    instruction
```

Dans notre exemple, pour un nombre initial égal à 8, on écrira donc :

```
deg PYTHON  
nombre = 8  
while nombre<50:  
    nombre=nombre+1
```

A la fin de la boucle, on aura :

```
deg PYTHON  
>>> from exemple import *  
>>> nombre  
50  
>>> |
```

En effet, lorsque **nombre** contient 49 la condition est toujours vérifiée et l'instruction s'exécute. **nombre** est donc incrémenté de 1 et vaut alors 50.

La condition n'est plus vérifiée et la boucle **while** se termine.

La fiche sur les instructions conditionnelles contenait la syntaxe des conditions possibles en Python. Elles sont répétées ci-dessous.

Condition	Syntaxe Python
x est égal à y	<code>x==y :</code>
x est différent de y	<code>x!=y :</code>
x est strictement supérieur à y	<code>x>y :</code>
x est strictement inférieur à y	<code>x<y :</code>
x est supérieur ou égal à y	<code>x>=y :</code>
x est inférieur ou égal à y	<code>x<=y :</code>

Exercice

Écrire une fonction `max_cube(n)` qui prend un entier naturel n en argument et qui renvoie le plus grand entier dont le cube est inférieur ou égal à n .

Analyse de l'énoncé

Prenons pour exemple le cas $n = 30$.

On cherche le plus grand entier naturel (0, 1, 2, ...) dont le cube est inférieur à 30.

Naturellement nous testons les entiers un par un :

- Si l'entier est 0 : $0^3 = 0 < 30$;
- Si l'entier est 1 : $1^3 = 1 < 30$;
- Si l'entier est 2 : $2^3 = 8 < 30$;
- Si l'entier est 3 : $3^3 = 27 < 30$;
- Si l'entier est 4 : $4^3 = 64 > 30$.

Le cube de 4 étant trop grand, l'entier recherché est 3 : c'est en effet le plus grand nombre dont le cube est inférieur à 30.

Grâce à cet exemple, on a pu identifier la structure de la fonction que l'on va écrire :

- On initialise une variable **entier** à 0;
- On teste si "**entier** au cube" est inférieur ou égal à n ;
- Si c'est le cas, on ajoute 1 à **entier**;
- Si ce n'est pas le cas, la réponse est **entier-1** (car **entier** au cube dépasse n);
- On recommence à l'étape 2.

Vous reconnaissez une structure de boucle **while** :

```
while condition:
    instruction
```

Ici la condition est "entier au cube est inférieur ou égal à n " :

```
entier**3 <= n
```

Et l'instruction est "ajouter 1 à entier" :

```
entier = entier + 1
```

Résolution

```
def max_cube(n):  
    entier=0  
    while entier**3<=n:  
        entier=entier+1  
    return entier-1
```

Remarque pour finir

Pour incrémenter un nombre d'une même valeur à chaque fois, vous pouvez écrire : `entier+=1` à la place de `entier=entier+1`.

Autre exercice

Écrire une fonction `facteurs(n)` qui prend un nombre entier en argument et renvoie la liste des facteurs de sa décomposition en facteurs premiers.

```
def facteurs(n):  
    liste_div=[]  
    diviseur=2  
    while n>1:  
        if n%diviseur==0:  
            liste_div.append(diviseur)  
            n=n/diviseur  
        else:  
            diviseur=diviseur+1  
    return liste_div
```

Quelques remarques sur le corrigé

On crée une liste des diviseurs premiers de n que l'on va remplir à chaque fois que l'on en trouve un : `liste_div`.

On commence par tester si n est divisible par 2.

Si c'est le cas, on stocke 2 dans `liste_div` et on remplace n par $n/2$.

Puis on recommence par tester si 2 divise le nouveau n .

Si ce n'est pas le cas, on ajoute 1 à 2 et on teste donc si le nouveau n est divisible par 3.

Ainsi de suite jusqu'à ce qu'à force de diviser n par ses diviseurs premiers, on arrive à $n = 1$.

Pour ajouter un élément à une liste, on utilise la fonction `append`. Pour ajouter l'élément 5 à une liste `L`, on écrira :

```
>>> L=[1,2]  
>>> L.append(5)  
>>> L  
[1,2,5]
```