Python 4 : La boucle while

1) La boucle while

La boucle while permet de répéter une opération tant (*while* veut dire tant que) qu'une condition est vérifiée.

Par exemple :

```
x=1
while x<10:         # la condition se situe entre le while et les :
    x=2*x           # ce qui se situe dans la boucle est identifiable grâce à l'indentation
print(x)            # cette instruction est en dehors de la boucle donc n'est exécutée qu'une fois
```

Le programme va multiplier x par 2 tant que x est strictement inférieur à 10, une fois que la condition n'est plus vérifiée, la boucle prend fin et le programme continue (ici il affiche x).

Donc dans ce programme, x vaut 1 au départ, puis comme x<10, alors x prend la valeur 2x donc 2, puis comme 2<10, x prend la valeur 2x donc 4, puis comme 4<10, x prend la valeur 2x donc 8, puis comme 8<10, x prend la valeur 2x donc 16, puis comme 16>=10, alors la boucle ne se répète plus, et le programme va afficher x, donc 16.

- Point très important, si on ajoute dans le programme précédent la variable n comme ceci :

```
x=1
n=0            # on n'oublie pas déclarer la variable n
while x<10:
    x=2*x
    n=n+1      # à chaque itération (car l'instruction est dans la boucle), n augmente de 1
print(x)       # x ne s'affiche qu'une seule fois car l'instruction est en dehors de la boucle
```

Le n va jouer un rôle de **compteur**, il s'ajoute de 1 à chaque itération de la boucle, donc il compte le nombre de fois que la boucle s'est lancée. Ici n vaut 4.

- Comme dans les boucles for et if, faire attention à <u>l'indentation</u> à la <u>syntaxe de la condition</u>.

- Notez également qu'une condition qui est tout le temps vérifiée entraine une boucle qui se répète sans fin et donc un programme qui ne se termine jamais, par exemple :

```
S=1
while S>0:
    S=S+1
print(S)
```

Ici le programme n'affichera jamais S car il exécute la boucle sans fin car S est toujours >0.

2) Méthodes usuelles

a) Savoir quand les termes d'une suite vérifient une condition

Si on a la donnée d'une suite récurrente comme ceci :
On connait le premier terme u_0 et on a une formule de récurrence, pour tout n, $u_{n+1} = f(u_n)$
Si on veut savoir, par exemple, à partir de quel rang n la suite dépasse un certain nombre A, on peut écrire :

```
u=u0                    #la valeur de u0 est à adapter dans chaque cas !
n=0                     # n est notre compteur
while u<A:              # tant que un est plus petit que A, on calcule le terme suivant
    u=f(u)              #la fonction f est à adapter dans chaque cas !
    n=n+1
print(n)                #ici n correspond au premier rang n tel que un>=A
```

b) Approcher la limite d'une suite

Si on a la donnée d'une suite récurrente comme ceci :
On connait le premier terme u_0 et on a une formule de récurrence, pour tout n, $u_{n+1} = f(u_n)$.
Si de plus, on sait qu'elle converge vers une limite l et qu'on a une inégalité du type :
Pour tout entier n, $|u_n - l| \leq \frac{1}{n+1}$ qu'on appelle inégalité (1)

Alors on peut écrire :

```
u=u0                    # à adapter avec la valeur du premier terme
n=1
while 1/(n+1)>10**(-4):          #tant que 1/(n+1) est plus grand que 0,001 on calcule les
termes de un
    u=f(u)              # On calcule le terme un+1 grâce à la formule de récurrence
    n=n+1               # on n'oublie pas de faire avancer n de 1
print(u)
```

On sort de la boucle une fois que $\frac{1}{n+1}$ est inférieur à 0,001, donc d'après l'inégalité (1), cela veut dire également que $|u_n - l|$ est inférieur à 0,001, donc que u_n est proche de sa limite l à 0,001 près[1], donc que u_n est une approximation de sa limite au millième.

Bien sûr, on adaptera le 0,001 en fonction de la précision de l'approximation cherchée, et on adapte aussi le $\frac{1}{n+1}$ en fonction de l'inégalité que l'on possède (on peut avoir exp(-n), 1/n^2 etc.)
On adapte aussi le f en fonction de l'exercice (si $u_{n+1} = 2u_n - 1$, on écrit u=2*u-1).

[1] Rappelons que $|a - b|$ correspond à la distance entre les réels a et b, donc $|u_n - l|$ correspond à l'écart entre un et sa limite.