CHAPITRE 3

Boucle conditionnelles (ou while)

1. BOUCLES WHILE

Une boucle conditionnelle, ou **boucle while** est une boucle qui s'arrête lorsqu'une certaine condition n'est plus remplie. Elle s'écrit ainsi :

```
while condition:
    bloc d instructions
    bloc d instructions
```

où `condition` est une expression de type booléen. Tant que le booléen vaut `True`, le bloc d'instructions va se répéter. Dès qu'il prend la valeur `False`, on sort de la boucle et on passe à la suite.

Attention

Il faut être bien sûr que la boucle va s'arrêter. Cela sera sûrement la cause de plantages lors de nos premiers TP.

Exemple 1 — Que fait la fonction `PE` dans le programme suivant ?

```
def PE(x):
    k=0
    while k < x:
        k = k+1
    return(k-1)
```

2. DIFFÉRENCES ENTRE LES BOUCLES FOR ET WHILE

La principale différence entre les boucles For et While c'est qu'on ne sait pas à l'avance (sauf si on fait une étude théorique) combien de fois on passera dans la boucle. On peut

toujours remplacer une boucle for par une "fausse boucle while" en remplaçant

```
for i in range(n):
    instructions
```

par

```
i = 0
while i < n:
    instructions
    i = 1 + 1
```

Le sens inverse est parfois possible, notamment en utilisant des instructions conditionnelles, mais pas toujours. Il est donc important de bien choisir à l'avance si c'est une boucle For ou une boucle While qui répondra le mieux au problème. La question à se poser est "est-ce que je peux trouver à l'avance le nombre de passage dans la boucle?" Si oui, autant faire une boucle For qui va limiter les erreurs humaines et mieux gérer les petits problèmes. Sinon, on fait une boucle While!

3. QUELQUES EXEMPLES

Détermination d'une valeur de seuil. Si on définit une la suite récurrente (u_n) par

$$u_0 = 3, u_{n+1} = u_n^2 - 1,$$

on peut vérifier que la suite u_n diverge en croissant vers $+\infty$. Formellement, cela se réécrit

$$\forall A > 0, \exists N \in \mathbf{N}, \forall n \geqslant N, u_n \geqslant A.$$

Il peut être intéressant de trouver ce premier rang N pour lequel $u_N \geqslant A$. On peut le coder ainsi.

```
def seuil(A): #prend le réel A en entree
    compteur = 0 #on initialise le compteur
    u = 3 #on initialise le calcul de la suite à u0 = 3
    while u < A: #on va calculer les termes de la suite tant que \
un < A
        u=u**2 - 1 #on calcul un+1
        compteur +=1 #on incrémente le compteur
    return compteur #on retourne N
```

Remarque 3.1 — Ainsi, on ne calcule que les termes de la suite qui sont nécessaires.

Inutile de coder la fonction qui à un entier associe u_n : elle serait source de beaucoup de calculs supplémentaires, par exemple on calculerait plusieurs fois les mêmes termes.

Remarque 3.2 — L'exemple précédent introduit un **compteur**, qui compte (comme son nom l'indique) le nombre de passage dans la boucle. C'est très fréquent dans les exercices qui demandent l'utilisation d'une boucle While/

Approximation d'une limite ou d'une somme. Venons en au suites convergente : si on sait qu'une suite converge mais qu'on ne connaît pas sa limite, on peut essayer de l'approximer de façon algorithmique. Cela se base sur la remarque suivante : si $u_n \to \ell \in \mathbf{R}$, alors $u_{n+1} - u_n \to 0$. On part du principe que si $u_{n+1} - u_n$ est assez petit, alors u_n est assez proche de ℓ.

On propose alors l'algorithme suivant, sur deux exemples.

Le premier : on rappelle que la suite définie par

$$u_0 = 1 \text{ et } u_{n+1} = \frac{u_n + a/u_n}{2}$$

converge vers le réel $\sqrt{a}$. Le code suivant, qui en entrée prend un seuil ϵ et un réel a, renvoie le premier terme u_n tel que $|u_{n+1} - u_n| \leqslant \epsilon$, donc une approximation de $\lim u_n = \sqrt{a}$.

```
1 def racine(a, eps):
2     u = 1
3     x = 0.5*(u + a/u)
4     while abs(x-u) > eps:
5         u = x
6         x = 0.5*(u + a/u)
7     return x
```

Exemple 2 — Modifier le code précédent pour qu'en plus il renvoie le nombre de passage dans la boucle.

Deuxième exemple, on verra ultérieurement dans le programme que la somme

$$S_n = \sum_{k=1}^{n} \frac{1}{k^2} \text{ converge .}$$

On souhaite trouver une approximation de la limite, notée

$$\lim_n \sum_{k=1}^{n} \frac{1}{k^2} = \sum_{k=1}^{+\infty} \frac{1}{k^2}.$$

Dans ce programme là, on sera obligé de garder en mémoire le compteur (comme la relation de récurrence dépend de n :

$$S_{n+1} = S_n + \frac{1}{(n+1)^2}.$$

```
def somme(eps):
    S = 1 #initialise la somme
    n = 1 #initialise le compteur
    aux = 1.5 #initailise la variable auxiliaire à S_2
    while abs(S-aux) > eps:
        n = n+1 #incrémente l'entier
        S = aux #change la somme
        aux = aux + 1/((n+1)*(n+1)) # la variable auxiliaire
    return S
```

Cette approche n'est pas extrêmement rigoureuse. Souvent dans les exercices il y aura une étude mathématiques préalable pour établir la condition.

Exemple 3 — ***Exercice type*** On définit (u_n) et (v_n) deux suites par

$$u_n = \sum_{k=1}^{n} \frac{1}{k^2} \text{ et } v_n = u_n + \frac{1}{n}.$$

1. Montrer que (u_n) et (v_n) convergent vers la même limite que l'on notera ℓ.
2. Montrer que pour tout $n \in \mathbf{N}^*$, $u_n < \ell < v_n$ puis que $|\ell - \frac{u_n+v_n}{2}| < \frac{v_n-u_n}{2}$..
3. Écrire une fonction Python qui prend en entrée un flottant `epsilon` et renvoie une approximation de ℓ à `epsilon` près.

Une fois les deux premières questions traitées, on peut proposer le code suivant :

```
def somme(epsilon):
    u = 1 #initialise la somme u_n
    v = 2 #initialise la somme v_n
    n = 1 #initialise le compteur
    while abs(u-v) > 2*epsilon:
        n = n+1 #incrémente l'entier
        u = u + 1/(n**2) #mise à jour u_n
        v = u + 1/n #mise a jour v_n
    return (u+v)/2
```

Le programme fonctionne bien car à chaque passage dans la boucle, `v`$= v_n$ et `u`$= u_n$. A la sortie, $u - v \leqslant 3\varepsilon$ donc $|\frac{u+v}{2} - \ell| \leqslant \varepsilon$ donc `(u+v)/2` est une sortie convenable.