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 PES dans le programme suivant ?

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

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
```

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

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

Il peut être intéressant de trouver ce premier rang N pour lequel $u_{\mathrm{N}} \geqslant \mathrm{A}$. 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.

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 $\lim u_n = \sqrt{a}$.

```
def racine(a, eps):
    u = 1
    x = 0.5*(u + a/u)
    while abs(x-u) > eps:
        u = x
        x = 0.5*(u + a/u)
    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 S2
    while abs(S-aux) > eps:
                n += 1 #incrémente l'entier
                S = aux #change la somme
                aux = aux + 1/((n+1)*(n+1)) # la variable auxiliaire
    return S
```

EXERCICES SUR LES BOUCLES WHILE

Exercice 1 Écrire une fonction qui prend en entrée un réel positif x et renvoie le plus petit entier $n \geqslant 2$ tel que $\frac{1}{n\ln(n)} \leqslant x$.

Exercice 2 Vous savez peut être qu'un entier naturel n est un nombre premier si et seulement si il n'admet aucun diviseur (non trivial) inférieur ou égal à $\sqrt{n}$. En s'inspirant de cette remarque, écrire une fonction qui détermine si un entier n est premier ou pas.

Exercice 3 On admet que

$$\lim_n \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} = \ln(2).$$

Écrire un programme qui prend en entrée un seuil `epsilon` et renvoie une approximation de $\ln(2)$ à ϵ près, ainsi que l'entier n pour lequel on a tronqué la somme.

Exercice 4

1. Écrire un programme qui prend en entrée un seuil ϵ et donne une approximation de la limite de

   $$\sum_{k=0}^{n} \frac{(-1)^n x^{2n}}{(2n)!}$$

   en s'arrêtant dès que terme que l'on rajoute est plus petit que ϵ.
2. On admet (pour l'instant) que

   $$\forall x \in \mathbb{R}, \cos(x) = \lim_n \sum_{k=0}^{n} \frac{(-1)^n x^{2n}}{(2n)!}.$$

   Écrire une fonction qui prend en entrée un flottant x et un seuil ϵ et qui renvoie une approximation de $\cos(x)$ avec la somme précédente, ainsi que la valeur de n pour laquelle on a tronqué la somme.

EXERCICES BILAN SUR LE DÉBUT DE L'ANNÉE.

Exercice 5 La suite de Fibonacci est la suite récurrente linéaire d'ordre 2 définie par

$$F_0 = F_1 = F_1 \text{ et } \forall n \in \mathbb{N}, F_{n+2} = F_{n+1} + F_n.$$

Écrire une fonction `Fibo` qui prend en entrée un entier n et renvoie F_n.

Exercice 6 Écrire une fonction qui prend en entrée une liste et renvoie son plus petit élément, ainsi que son indice dans la liste.

Exercice 7 Écrire une fonction qui prend entier trois flottants a,b,c et renvoie le domaine de la fonction

$$x \mapsto \ln(ax^2 + bx + c).$$

Exercice 8 On définit une suite récurrente par $W_0 = \frac{\pi}{2}$ et

$$\forall n \in \mathbb{N}, nW_nW_{n+1} = \frac{\pi}{2}.$$

Coder une fonction qui prend en entrée n et renvoie W_n.

Exercice 9 On définit une suite récurrente par

$$u_0 = a \in \mathbb{N}^* \text{ et } \forall n \in \mathbb{N}, u_n = \begin{cases} \frac{u_n}{2} \text{ si } u_n \text{ est pair.} \\ 3u_n + 1 \text{ sinon .} \end{cases}$$

Coder une fonction qui affiche les n premiers termes de cette suite. Faire une conjecture.

Exercice 10 Coder une fonction qui prend en entrée deux entiers positifs k et n et renvoie le coefficient binomial $\binom{n}{k}$.

1. En utilisant les factorielles (une fonction auxiliaire sera la bienvenue..
2. Sans utiliser les factorielles (on s'aidera du triangle de Pascal).