Feuille de cours 4 (informatique) : Boucle while

On a vu que pour répéter des instructions un certain nombre de fois on peut utiliser une boucle **for**. La syntaxe utilisant **range** fait clairement apparaître qu'on utilise une boucle **for** lorsqu'on sait à l'avance le nombre de fois qu'on veut répéter les instructions (par exemple pour les répéter `n` fois, on peut utiliser **range**( )).

Lorsqu'on ne sait pas à l'avance combien de fois on veut répéter des instructions, on peut utiliser une boucle **while** ("tant que"), on va alors répéter des instructions tant qu'une certaine condition est vérifiée.

1 Syntaxe de la boucle **while**

La condition est exprimée à l'aide d'un booléen.

```
1 while condition :
2     # bloc d'instructions
3 # instructions en dehors de la boucle
```

Tant que le booléen `condition` a la valeur `True`, Python exécute le bloc d'instructions de la ligne 2. Dès que le booléen `condition` prend la valeur `False`, il sort de la boucle et passe à la ligne 3. En particulier, Python "n'entre pas" dans la boucle si la condition n'est pas satisfaite initialement.

Remarque : Encore une fois, les deux points et l'indentation sont essentiels.

Il y a deux différences fondamentales avec la boucle **for** :

- la boucle **while** n'affecte pas de variable. Il faut donc définir les variables apparaissant dans la condition au préalable : c'est à vous d'initialiser la condition **en amont de la boucle**.
- la boucle **while** n'agit pas naturellement sur la condition qu'elle teste (pas d'incrémentation) : c'est à vous de mettre à jour la condition **à l'intérieur de la boucle**.

Exercice 1
Qu'affiche Python à l'exécution des scripts suivants ?

```
1 k = 1
2 while k<5 :
3     k = 2*k
4 print(k)
```

```
1 k = 1
2 while k>5 :
3     k = 2*k
4 print(k)
```

```
1 k = 10
2 while k>5 :
3     k = 2*k
4 print(k)
```

Attention ! Pour que la boucle s'arrête (on dit "pour qu'elle termine"), il faut être sûr que le booléen va prendre la valeur `False` au bout d'un moment ! En particulier, si on oublie de mettre à jour la condition dans une boucle (ou si on la met à jour de manière incorrecte), on boucle indéfiniment ! Si cela vous arrive en TP, que faire ?

Exercice 2
Écrire une fonction `nb_puiss_2` prenant en argument un entier `n` et renvoyant le nombre de fois que cet entier est divisible par 2, c'est-à-dire l'entier k tel que $n = 2^k m$ avec m impair.

2 Algorithmes de seuil

La boucle `while` est très souvent utilisée pour trouver le plus petit entier n tel qu'une certaine quantité u_n est au-dessus ou en-dessous d'un certain seuil M.

Si on note `u_n` une variable contenant u_n alors les fonctions suivantes renvoient...

```
def f(M):
    n = 0
    u_n = u_0
    while u_n < M :
        # update pour u_n
        n = n+1
    return n
```

```
def f(M):
    n = 0
    u_n = u_0
    while u_n >= M :
        # update pour u_n
        n = n+1
    return n
```

Remarque 1
Attention, si la mise à jour de `u_n` fait intervenir `n`, alors l'ordre dans lequel on update les variables `u_n` et `n` compte !

Exercice 3
Écrire une fonction `seuil1` prenant en argument un réel `M` et renvoyant le plus petit entier naturel n tel que $n^2 - 3n + 1 \geq$ `M`. Pourquoi votre boucle **`while`** termine-t-elle ?

Exercice 4
Écrire une fonction `seuil2` prenant en argument un réel `M` et renvoyant le plus petit entier naturel n tel que $\sum_{k=0}^{n} \sqrt{k} \geq M$.

Exercice 5
Écrire une fonction `seuil3` prenant en argument un réel `M` et renvoyant le plus petit entier naturel non nul n tel que $n! > M$. On rappelle que $n! = 1 \times 2 \times 3 \times \cdots \times n = \prod_{k=1}^{n} k$.

Exercice 6
Soit (u_n) la suite définie par $u_0 = 1$ et : $\forall n \geq 0,\ u_{n+1} = \dfrac{u_n}{n+1}$. On peut montrer que (u_n) décroît vers 0 puis écrire une fonction `seuil4` prenant en argument un réel ε et renvoyant la plus grande valeur de u_n telle que $u_n < \varepsilon$.

3 Algorithmes d'approximation numérique : exemple du calcul de $\sqrt{2}$

Vous êtes-vous déjà demandé ce qu'il se passe dans votre calculatrice (ou dans Python) lorsque vous demandez la valeur de $\sqrt{2}$? En effet, elle ne contient pas en mémoire la valeur de $\sqrt{2}$ (s'il fallait stocker toutes les valeurs des racines carrées, votre petite calculatrice ne suffirait pas !).

En fait, elle utilise un algorithme qui lui permet de calculer une valeur approchée de $\sqrt{2}$. Cet algorithme peut se ramener à calculer une suite (u_n) définie par récurrence et telle que $\lim\limits_{n\to+\infty} u_n = \sqrt{2}$. Si on prend un N "assez grand" alors u_N sera une approximation de $\sqrt{2}$.

Mais comment choisir ce N assez grand ? On peut proposer plusieurs méthodes en s'appuyant sur le dessin ci-dessus :

- on peut demander que la quantité u_N soit proche à ε près de u_{N+1}, autrement dit que la distance entre u_N et u_{N+1} soit plus petite que ε :

- on pourrait aussi directement demander que u_N soit proche à ε près de $\sqrt{2}$, autrement dit que :

  Mais cela pose problème car

  On peut donc plutôt demander que soit proche de à ε près, c'est-à-dire que :

Dans tous les cas, dans ce genre d'algorithmes d'approximation, on choisit une valeur ε **petite** et on s'arrête quand une certaine quantité a est proche d'une autre quantité b à ε près, c'est-à-dire quand

Que signifie que ε doit être "petit" ? C'est à vous d'en juger en gardant en tête que si l'objectif est de calculer $\sqrt{2} \simeq 1,4$, il serait étrange de penser que $\varepsilon = 1$ soit "petit". En fait la valeur de ε à choisir dépend de la précision voulue : plus ε sera petit, plus on obtiendra une meilleure approximation de $\sqrt{2}$. En TP, on pourra essayer avec $\varepsilon = 10^{-3}$, $\varepsilon = 10^{-7}$, etc.

Notez enfin qu'en Python, la fonction valeur absolue existe (sans avoir besoin de l'importer via une bibliothèque), il s'agit de la fonction **`abs`**.

Exercice 7 (méthode de Héron)
Soit la suite (u_n) définie par $u_0 = 1$ et : $\forall n \geq 0,\ u_{n+1} = \dfrac{u_n}{2} + \dfrac{1}{u_n}$.
On peut montrer que (u_n) converge vers $\sqrt{2}$. On peut donc calculer les termes de cette suite pour obtenir une approximation de $\sqrt{2}$: cette méthode pour calculer la valeur de $\sqrt{2}$ est appelée méthode de Héron.
Fixons $\varepsilon > 0$ une quantité "petite" et considérons que u_N est une bonne approximation de $\sqrt{2}$ lorsque $|u_N^2 - 2| \leq \varepsilon$. Écrivons alors une fonction `approx` prenant en argument ε et renvoyant une approximation de $\sqrt{2}$ d'autant meilleure que ε est petit.

Remarque 2
Pouvait-on faire les fonctions précédentes avec une boucle `for` ?

4 For vs While

Une boucle **for** peut toujours être réalisée à l'aide d'une boucle **while** (mais l'inverse n'est pas vrai). Par exemple la syntaxe

```
for k in range(a,b) :
    # Vos instructions
```

peut être remplacée par

```
k
while       :
    # Vos instructions
    k=k+1
```

Exercice 8
Dans chacun des cas suivants, écrire avec l'autre boucle un programme équivalent :

```
u = 1
for k in range(8):
    u = u + k
```

```
u = 2
k = 3
while k < 10 :
    u = u * k
    k = k + 1
```

```
u = 2
k = 3
while k < 10 :
    u = u * k
    k = k + 2
```

Quand c'est possible, on préférera utiliser une boucle **for** plutôt qu'une boucle **while**.

5 Entraînement

Exercice 9

1. Pour $n \in \mathbb{N}^*$ on note $H_n = \sum_{k=1}^{n} \frac{1}{k}$. On peut montrer que $H_n \underset{n\to+\infty}{\longrightarrow} +\infty$. Écrire une fonction prenant en argument $M \in \mathbb{R}$ et renvoyant le plus petit entier $n \in \mathbb{N}^*$ tel que $H_n \geq M$.
2. Soit (u_n) la suite définie par $u_0 = 2$ et $\forall n \in \mathbb{N},\ u_{n+1} = -n - u_n^2$. Écrire une fonction prenant en argument un réel $M \in \mathbb{R}$ et renvoyant le plus petit entier $n \in \mathbb{N}$ tel que $u_n < M$. Pourquoi votre fonction termine-t-elle ?

Exercice 10

Dans cet exercice, on souhaite écrire une fonction Python prenant en argument une valeur ε et renvoyant une approximation de $\ln(2)$ qui soit d'autant meilleure que ε est petite.

Pour cela, on va utiliser la quantité $S_n = \sum_{k=0}^{n} \frac{(-1)^k}{k+1}$. On peut démontrer (mais on ne demande pas de le faire) que : $\lim_{n\to+\infty} S_n = \ln(2)$.

On souhaite donc écrire une fonction Python renvoyant une valeur de S_N qui soit proche de $\ln(2)$. Pour savoir si S_N est proche de $\ln(2)$, on choisit de fixer $\varepsilon > 0$ petit, et de dire sur S_N est proche de $\ln(2)$ lorsque $|S_{N+1} - S_N| \leq \varepsilon$.

1. Simplifier $S_{N+1} - S_N$ et reformuler la condition $|S_{N+1} - S_N| \leq \varepsilon$ de manière plus simple.
2. Écrire une fonction Python prenant en argument ε et renvoyant l'approximation de $\ln(2)$ correspondante calculée par la méthode décrite ci-dessus.