Python - Boucles

1 Boucles conditionnelles

La **boucle conditionnelle** est un programme permettant l'exécution d'instructions tant qu'une condition est (ou n'est pas) vérifiée. On parle aussi de **boucle while**. La syntaxe Python d'une boucle conditionnelle est :

```
while condition:
    instructions
```

1. Notez, comme toujours en Python, que l'indentation est signifiante : le début du bloc d'instructions est marqué par les deux points, l'ensemble du bloc d'instructions doit avoir la même indentation, et le retour à l'indentation du `while` indique la fin du bloc d'instruction.
2. Une boucle while est exécutée tant que la condition est satisfaite : il est donc primordial de vérifier que celle-ci s'arrête à un moment !
3. En particulier, la condition doit dépendre d'objets modifiés par le bloc d'instruction !

Exercice 1

1. Donner l'entier contenu par la variable `i` à l'issue des instructions suivantes.
```
i=1
while i<100:
    i*=2
```
2. Expliquer pourquoi exécuter le code suivant n'est pas une bonne idée.
```
i=1
while i>0:
    print(i)
    i=i+1
```

Exercice 2 Un premier seuil

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$u_0 > 0, \quad \text{et } \forall n \in \mathbb{N},\ u_{n+1} = \frac{u_n}{2} - 3n.$$

1. Montrer que (u_n) est majorée par son premier terme, et non minorée.
2. En déduire qu'il existe $n_0 \in \mathbb{N}$ tel que :

$$\forall n < n_0,\ u_n \geq 0, \quad \text{et } \forall n \geq n_0,\ u_n < 0.$$

3. Écrire une fonction `seuil(u)` qui prend en argument le premier terme de la suite u_0, et renvoie le rang du dernier terme de la suite positif.

Exercice 3 Syracuse

Pour tout $u_0 \in \mathbb{N}^*$, on considère la suite donnée par la relation de récurrence suivante

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$$

1. Montrer que si $u_0 = 1$, alors (u_n) est périodique et déterminer sa période.

2. En déduire que si $u_k = 1$ pour un certain $k \in \mathbb{N}$, la suite est périodique à partir d'un certain rang.

3. Écrire á l'aide d'une boucle while un programme qui demande à l'utilisateur un entier naturel N, et renvoie tous les termes de la suites (u_n) partant de $u_0 = N$, jusqu'à ce que la valeur 1 soit atteinte. On prendra soin de demander à Python de faire tous les calculs sur des entiers.
4. Tester votre programme avec quelques valeurs.
5. Ce programme termine-t-il toujours ?

Représentation graphique On peut demander à Python de représenter graphiquement le vol de la suite. Pour cela, on utilise la commande `plot` de la librairie `matplotlib.pyplot`. Entrer les commandes suivantes :

```
import matplotlib.pyplot as plt
```

1. Modifier votre programme pour que, au lieu d'afficher les termes de la suite, il les stocke dans une liste, notée `L`. On ulitisera la commande `L.append(a)` qui permet d'ajouter l'objet `a` à la fin de la liste `L`.
2. Entrer alors les instructions :

```
x=list(range(len(L)))
plt.plot(x,L)
plt.show()
```

2 Boucles inconditionnelles

Nous avons vu la semaine dernière que les boucles conditionnelles, ou boucles *while*, permettent de demander l'exécution d'une instruction quand on ne sait pas a priori combien d'étapes sont nécessaires. En pratique, dans de nombreux cas, on connaît à l'avance combien d'itération il faut effectuer, et la boucle conditionnelle est inutile. On peut alors demander explicitement à la machine d'effectuer ce nombre d'opérations : c'est le principe de la **boucle inconditionnelle**, ou **boucle for**.

La syntaxe la plus commune est la suivante :

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

Remarques :

1. dans la syntaxe ci-dessus, `i` est le compteur de boucle : c'est une nouvelle variable, définie localement dans la boucle. Les instructions peuvent dépendre de cette variable. Elle est incrémentée automatiquement à chaque passage dans la boucle.
2. Le programme ci-dessus demande `n` passages dans la boucle : la variable `i` parcours les entiers de 0 à $n-1$. Attention à ça !
3. Comme toujours, l'indentation est signifiante : le début du bloc d'instructions est marqué par les deux points, l'ensemble du bloc d'instructions doit avoir la même indentation, et le retour à l'indentation du `for` indique la fin du bloc d'instruction.

Une très courte introduction aux listes

Les listes sont un objet central en Python, et nous y reviendrons plus en détail plus tard. Voyons juste quelques commandes de survie.

— Création de liste :

```
L=[1, 2, 3, 4]    #entre crochets, éléments séparés par une virgule
L=[]              #liste vide
L=list(range(n))  #conversion d'un autre type d'objet en liste
```

— Accès au éléments d'une liste : tester dans la console les commandes suivantes.

```
>>> L=[21, 32, 43]
>>> L[0]
>>> L[2]
>>> L[3]
```

— Méthodes applicables sur les listes :

```
len(L)            # longueur de L
L1 + L2           # concaténation des deux listes L1 et L2
L.append(a)       # ajoute l'objet a à la fin de la liste L
L.remove(a)       # retrait de la première occurence de a
del(L[i])         # suppression de l'attribut d'indice i
```

Avec graphiques

On peut obtenir des graphiques en Python avec la bibliothèque *matplotlib.pyplot*, qui va essentiellement tenter de simuler le comportement de Matlab. Les instructions sont les suivantes :

```
import matplotlib.pyplot as plt   # importer la librairie
plt.plot(X1,Y1)                   # tracé d'une courbe
plt.plot(X2,Y2)                   # tracé d'une seconde courbe
. . . .
plt.show()                        # tout afficher dans une fenêtre graphique
```

Dans cette syntaxe, la commande `plt.plot(X,Y)` prend en argument deux **listes de même tailles** X et Y, et tracera la ligne brisée reliant les points dont les coordonnées sont stockées dans X (abscisses) et Y (ordonnées).

Exercice 4

Prévoir ce qu'affichera chacune des trois instructions suivante. Tester les.

```
for k in range(12):
    print(k**2)

for i in 'Saint-Jean de Passy':
    print(i)

for j in [1,3,1,3,1,3]:
    print(j**3)
```

On retiendra qu'en Python, une boucle *for* fonctionne avec tout objet **itérable**, c'est-à-dire que l'on peut parcourir dans l'ordre. L'indice de boucle prendra alors toutes les valeurs contenues dans cet itérable.

Exercice 5 Factorielle et sommes

1. Ecrire une fonction `factoriel(n)` prenant en argument un entier n et renvoyant $n!$. Tester votre fonction (avec de petites valeurs de n.)
2. a) Écrire une fonction `somme(n)` prenant en argument un entier n et renvoyant $\sum_{k=1}^{n} k$. Tester votre fonction, et comparer le résultat avec la valeur théorique.

   b) Modifier la fonction précédante en `somme(n,a)` de façon à ce qu'elle renvoie $\sum_{K=1}^{n} k^a$. Tester votre fonction avec $a = 1, 2, 3$. Comparer avec les valeurs théoriques.

Exercice 6 Une suite récurrente

1. On définit la suite u par $u_0 \in \mathbb{R} \setminus \{2\}$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \dfrac{2u_n - 3}{u_n - 2}$.

2. Montrer que la suite est bien définie (on pourra admettre provisoirement ce résultat, et garder sa résolution pour votre soirée).
3. Écrire une fonction `suite1(n,u0)` qui prend en arguments un entier naturel n, un réel u_0, et qui renvoie la valeur de u_n.
4. Modifier le programme précédent pour renvoyer la liste `U` contenant les coefficients $u_0, u_1, \ldots, u_n$. *(On pourra ajouter au fur et a mesure la variable u dans une liste U.)*
5. Tester votre programme avec $u_0 = 1$, $u_0 = 3/2$, et $u_0 = 4$. Que constatez-vous ?
6. Si la suite est convergente, quelles sont les valeurs possibles de la limite ℓ ?
7. Déterminer en fonction de u_0 la valeur du terme général u_n.

Exercice 7 Une suite récurrente d'ordre 2

Soit u la suite définie par $u_0 = 1, u_1 = 3$, et pour tout $n \in \mathbb{N}$, $u_{n+2} = 2u_{n+1} + 3u_n$.

1. Écrire une fonction `suite2(n,u0,u1)` qui prend comme argument un entier n, deux réels u_0 et u_1, et renvoie la liste des $(n+1)$ premiers termes de la suite.
2. Conjecturer le terme général de la suite, et le démontrer.
3. Même question avec $u_0 = 1, u_1 = -1$.

Exercice 8 Le problème de Josèphe

Flavius Josèphe est un historiographe romain d'origine judéenne du I^{er} siècle. Il raconte que, lors de la première guerre judéo-romaine, il faisait partie d'un groupe de 40 soldats juifs piégés dans une cave par les soldats romains. Ils décidèrent, plutôt que de se rendre de se suicider selon le schéma suivant. Ils formèrent un cercle avec une origine, un soldat numéroté 1. Celui-ci se suicide, puis une personne sur trois (parmi les survivants) en faisant le tour du cercle autant de fois que nécessaire. Josèphe, peu enclin à se suicider, décide de se positionner de façon à être le dernier rescapé, et ainsi avoir le loisir de se rendre aux romains.

1. Écrire un programme `Josephus(n)` prenant en argument le nombre de soldats, et renvoyant la bonne position pour Josèphe.
2. Modifier votre programme en `Josephus(n,r,p)`, où n est le nombre de soldats, r le nombre voulu de rescapés, et p le pas entre chaque suicidés. Il renverra une liste de r positions.