Lycée Louis-Le-Grand, Paris 2013/2014
MPSI 4 – Informatique pour tous
J.-P. Becirspahic, A. Troesch

TP n° 2 : Listes, structures itératives

Les exercices 1 à 3 sont repris de la fin du TP n° 1. Les élèves les ayant déjà traités peuvent passer directement à l'exercice 4. Les exercices 4 à 6 doivent aller assez vite.

Exercice 1 – Structures conditionnelles simples

Il est fréquent de devoir différencier l'action à effectuer suivant les cas. On utilise pour cette situation la structure conditionnelle

```
if bool:
    instructions
else:
    instructions
```

Le booléen `bool` est le plus souvent obtenu sous forme d'un test (`==`, `!=`, `>`, `>=`, `<`, `<=`, `is`, `in`*).*
Si la discussion porte sur plus de deux termes, on peut ajouter des tests intermédiaires grâce à `elif` *(abréviation de else if).*

Écrire dans un programme une fonction prenant en paramètre une année, renvoyant un booléen, égal à `True` si et seulement si l'année est bissextile. On demandera une année à l'utilisateur, et on lui affichera en réponse un texte disant si l'année est bissextile ou non.
On rappelle que depuis octobre 1582, une année n est bissextile si et seulement si n est divisible par 4, sauf si n est divisible par 100, mais pas par 400. On rappelle également qu'avant 1582, les années bissextiles étaient exactement les années multiples de 4.

Exercice 2 – Structures itératives conditionnelles

Les structures itératives (boucles) permettent de répéter un bloc d'instructions un grand nombre de fois. Dans cette exercice, nous étudions la boucle :

```
while bool:
    instructions
```

qui effectue les instructions données dans le bloc suivant les deux-points tant que le booleen `bool` a la valeur `True`.

1. Soit pour tout $n \in \mathbb{N}^*$, $S_n = \sum_{k=1}^{n} \frac{1}{n}$. Écrire un programme déterminant la plus petite valeur de n pour laquelle $S_n > A$, A étant un réel entré par l'utilisateur. N'essayez pas votre programme avec des valeurs de A supérieures à 22.
2. On rappelle que si a et b sont deux entiers strictement positifs si r le reste de la division euclidienne de a par b, alors, si $r \neq 0$, le pgcd de a et b est égal au pgcd de b et r. En répétant cette opération jusqu'à obtenir un reste nul, on peut donc calculer le pgcd (c'est l'algorithme d'Euclide).
   Écrire un programme demandant à l'utilisateur deux entiers a et b, puis calculer et afficher leur pgcd.

Exercice 3 – On définit la suite de Syracuse par $u_0 \in \mathbb{N}^*$, et

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

On veut vérifier la propriété suivante : il existe un rang N tel que $u_N = 1$ (et à partir de ce rang, la suite boucle sur la séquence 4, 2, 1, 4, 2, 1, etc.). Écrire un programme demandant à l'utilisateur une valeur initiale u_0, calculant les différents termes de la suite tant qu'ils ne sont pas égaux à 1, et affichant pour terminer la première valeur de N pour laquelle $u_N = 1$, ainsi que la plus grande valeur obtenue pour u_n.

Exercice 4 – Manipulations élémentaires de listes

Les questions suivantes sont à faire dans l'interpréteur Python.

1. Créer la liste `l=[1,2,3,4,5]`, et extraire le terme d'indice 4. Que remarquez-vous ?
2. Redéfinissez le terme d'indice 4 comme étant la somme des deux précédents.
3. Essayez de sommer, de multiplier deux listes entre elles ; de sommer, de multiplier une liste par un entier. Observations ?
4. Comparer l'effet sur l'adressage mémoire des opérations `l = l + [1]`, `l += [1]` et `l.append(1)`.

La fonction `randint(n,m)` *du module* `random` *réalise un générateur aléatoire uniforme sur l'intervalle d'entiers* $[\![n, m]\!]$. *La fonction* `sample(population,k)` *du module* `random` *permet de créer une liste constituée de* k *éléments deux à deux distincts de la population. La population doit donc être de taille supérieure à* k. *Il peut s'agit d'une liste ou d'un ensemble, ou encore de l'itérateur rm* ***`range(n)`*** *qui parcourt les entiers de* 0 *à* $n-1$.

4. Générer une liste `li` de longueur aléatoire comprise entre 10 et 10000, et contituée d'entiers distincts choisis au hasard entre 1 et 10000.
5. Extraire le dernier terme de la liste `li`, sans modifier la liste, et sans chercher à déterminer la longueur de la liste.
6. En s'aidant de la fonction `help` (pour cette question et celles qui suivent), déterminer la longueur de la liste `li`.
7. Toujours grâce à l'aide, déterminer le rang de l'éventuelle occurrence de 1. S'il n'y en a pas, rajouter l'élément en fin de liste. Déplacer la valeur 1 en début de liste.
8. Stockez dans une variable x la valeur d'indice 5 de la liste, et supprimez cette valeur du tableau (en une seule instruction).
9. À l'aide des méthodes associées aux listes, triez le tableau en ordre décroissant.
10. À l'aide de la fonction `sum`, déterminer la somme des éléments de la liste `li`

Exercice 5 – Techniques de saucissonnage sur les listes

On rappelle (voir le cours) qu'étant donné une liste `li`, `li[m:n]` *renvoie un tableau contitué des termes d'indices* m *à* $n-1$ *du tableau* `li`. *Les questions suivantes sont à faire dans l'interpréteur Python.*

1. (a) Définir une liste `liste` d'entiers, de longueur 10.
   (b) Extraire la tranche `[2:6]`, visualisez le résultat, triez cette tranche par ordre décroissante, replacez-la dans la liste initiale. Ce faisant, contrôlez l'effet de ces opérations sur les adresses.
2. (a) Insérez la liste `liste` à l'intérieur d'elle même, à l'indice 3. Proposez deux méthodes pour le faire.
   (b) Supprimez l'insertion que vous venez de faire.

Exercice 6 – Mutabilité et copies

On vérifie dans cet exercice les comportements relatifs à la mutabilité des listes étudiés en cours.

1. Créer une liste `liste1`, la copier dans `liste2`. Comparez les identifiants de `liste1` et `liste2`.
2. Modifiez un attribut de `liste1`. Quel est l'effet sur `liste2` ?
3. Effectuez une copie `liste3` de `liste1` par saucissonnage. Quel est l'effet sur les adresses ? Modifier un attribut de `liste1`. Quel est l'effet sur `liste3` ?
4. Créer une liste `liste4`, dont l'attribut `liste4[0]` est elle-même une liste. Créer une copie `liste5` de `liste4` par saucissonnage.
5. Comparez les adresses de `liste4` et `liste5`. Comparer les adresses des attributs `liste4[0]` et `liste5[0]`. Modifier un des attributs de la liste `liste4[0]`, et vérifier l'effet de cette modification sur `liste5`.
6. Créer le tuple `couple=([1,2],3)`. Peut-on modifier l'attribut d'indice 1 de `couple` ? Modifier un des attributs de `couple[0]`, et voir l'effet sur `couple`.
   Ce n'est pas parce que le contenu de `couple` *ne peut pas être modifié en place que les contenus des attributs ne peuvent pas être modifiés, si ces attributs sont mutables.*

Exercice 7 –

1. Dans le module `numpy.random` se trouve une fonction nommée `randint`. Utilisez-la pour créer une liste de 100 entiers tirés au hasard entre 0 et 99.
2. Calculer le nombre d'éléments de $[\![0, 99]\!]$ qui n'appartiennent pas à cette liste.
3. Recommencer cette expérience un grand nombre de fois et calculer la moyenne du nombre d'absents.
4. Comparer à la valeur théorique.

Exercice 8 – **In and Out Shuffle**

Une méthode traditionnelle pour mélanger un paquet de cartes consiste à couper le paquet en deux puis à entrelacer ces deux parties. Lorsque les deux parties sont égales et que l'entrelacement se fait carte par carte, le mélange est dit parfait.

Il y a deux types de mélanges parfaits :

- le *out shuffle* lorsqu'on reconstitue le jeu en commençant par la première carte de la première moitié ;
- le *in shuffle* lorsqu'on reconstitue le jeu en commençant par la première carte de la seconde moitié.

On rappelle que `li[a::p]` extrait les termes d'une liste de p en p, à partir de l'indice a.

1. Écrire une fonction `out_shuffle` effectuant le mélange *out shuffle* d'une liste contenant un nombre pair d'éléments.

   Pour un jeu de 52 cartes, combien de mélanges doit-on effectuer pour retrouver la liste dans son état initial ?
2. Mêmes questions avec le *in shuffle*.

Exercice 9 – À l'aide du crible d'Ératosthène, créer une liste contitué des entiers premiers inférieurs ou égaux à 1000.

Exercice 10 –

1. Créer une liste `entiers` de tous les entiers de 2 à 100.
2. Définir par compréhension une liste `diviseurs` contituée des couples (n, diviseurs de n), pour tout n élément de la liste `entiers`, les diviseurs de n étant donnés sous forme d'une liste.
3. En déduire la liste `premiers` de tous les entiers premiers de 2 à 100.
4. Que pensez-vous de cette méthode par rapport au crible d'Ératosthène ?

Exercice 11 – **Calculs de suites récurrentes, et de sommes**

Les questions sont indépendantes.

1. Soit la suite définie par $u_0 = 0$, et pour tout $n \in \mathbb{N}$, $u_{n+1} = \sqrt{3u_n + 4}$. On montre facilement que cette suite converge vers 4 en croissant.
   (a) Écrire un programme demandant à l'utilisateur un entier n en renvoyant tous les termes de la suite jusqu'à u_n.
   (b) Écrire un programme renvoyant le plus petit entier n pour lequel $u_n > 3,99999999$.
2. Calculer $\displaystyle\sum_{n=0}^{1000} u_n$ où $u_0 = 1$ et $\forall n \geqslant 0$, $u_{n+1} = \dfrac{1}{u_n + 1}$.
3. Écrire un programme affichant les n premiers termes de la suite définie par $u_0 = 0$, $u_1 = 2$, pour tout $k \in \mathbb{N}$, $u_{k+2} = \sin u_k + 2\cos u_{k+1}$ (la valeur de n étant demandée à l'utilisateur). La suite $(u_n)_{n\in\mathbb{N}}$ semble-t-elle convergente ?
4. Soit $f(x, y) = x\cos y + y\cos x$. On définit une suite u_n par $u_0 = 0$, $u_1 = 1$, $u_{n+2} = f(u_{n+1}, u_n)$ si n est pair, et $u_{n+2} = f(u_{n+1}, u_{n-1})$ si n est impair. Afficher les N premières valeurs de (u_n).

Exercice 12 – L'entier n étant donné par l'utilisateur, afficher les n premières lignes du triangle de Pascal. On veillera à aligner (suivant leurs unités) les valeurs situées sur une même colonne.

Exercice 13 – Écrire, pour $(b, c) \in [\![2, 16]\!]^2$ un convertisseur de l'écriture d'un entier positif n de la base b à la base c. On fera entrer b, n (en base b) et c par l'utilisateur. On utilisera les lettres 'a', 'b', 'c', 'd', 'e' et 'f' pour désigner les chiffres suivant le chiffre 9 en bases 11 à 16.