Feuille de cours 5 (informatique) : Listes 1 : création et parcours par éléments

Une liste est une séquence d'éléments de types quelconques, rangés dans un certain ordre, listés entre crochets et séparés par des virgules. Voici quatre exemples de listes :

```
L1 = [1,2,3]
L2 = [3,2,1]
L3 = [1,1,1,1,1]
L4 = [3,'banane',[1,2]]
```

Remarque 1

- l'ordre des éléments compte (ainsi pour Python `L1` $\neq$ `L2`)
- un élément peut être présent plusieurs fois dans une liste (comme dans `L3`)
- les éléments d'une liste ne sont pas forcément de même type (comme dans `L4`)

La strucure de liste est pratique pour stocker un grand nombre de données sans avoir besoin de créer une variable pour chacune d'entre elles.

Important : on sera souvent amené à utiliser la **liste vide** c'est-à-dire la liste ne contenant aucun élément : il s'agit de `L = []`.

1 Création de listes

Lorsqu'une liste ne contient que peu d'éléments, tous connus à l'avance, on peut la saisir directement "à la main", par exemple pour créer une liste `L` correspondant au 5-uplet $(3, -2, \pi, 0, 1)$ on écrira :

En pratique cependant, on est le plus souvent amené à demander à Python de créer lui-même la liste. Pour cela, il existe essentiellement deux méthodes.

1.1 Création par ajouts successifs d'éléments

Pour créer une liste `L`, on peut :

- initialiser `L` à la liste vide `[]`, puis
- lui rajouter successivement des éléments grâce à une boucle `for`.

Important : pour rajouter un élément `x` à une liste `L`, on utilise la commande `L.append(x)`. Avec cette commande, l'élément `x` est rajouté **à la fin** de la liste `L`.

```
L = []
for ... :
  L.append(element a ajouter)
```

Remarque 2
Notez que la commande `L.append(x)` ne crée pas une nouvelle liste, mais *modifie* la liste `L` (ainsi, il ne faut **pas** écrire `L = L.append(x)` mais simplement `L.append(x)`).

Exercice 1
Que contient `L` après exécution du code ci-contre ?

```
1 L = []
2 L.append(4)
3 L.append(7)
4 L.append(2)
```

Que contient `M` après exécution du code ci-contre ?

```
1 M = []
2 for k in range(4):
3     M.append(k)
```

Exercice 2
1. Écrire une fonction prenant en argument un entier `n` supérieur à 3 et renvoyant la liste `L = [3,4,5,...,n]`.
2. Écrire une fonction prenant en argument `n` $\in \mathbb{N}$ et renvoyant la liste `L = [1,4,9,16,...,n^2]`.

1.2 Création "en compréhension"

Il existe une deuxième façon de créer des listes en donnant la forme de leurs éléments, c'est d'utiliser la syntaxe

`[f(k) for k in L]`

où `L` est elle-même une liste ou un range, et où `f(k)` désigne n'importe quelle expression dépendant (ou non) de `k`.

Par exemple, `[k**2+2 for k in range(4)]` vaut

Exercice 3
Comment créer la liste `L2 = [1,4,9,...,n^2]` à partir de la liste `L1 = [1,2,3,...,n]` ?

Exercice 4
Définir une liste `Lbis` contenant les mêmes éléments que `L` mais créée par une autre méthode :

```
L=[]
for k in range(10):
    L.append(2*k)
```

```
L=[k**2 for k in range(2,7)]
```

1.3 Liste des termes d'une suite

Il arrive très souvent qu'on veuille obtenir la liste des termes succcessif d'une suite c'est-à-dire une liste du type `L = [`u_0`,`u_1`,`u_2`,...,`u_n`]`.

Quelle méthode utiliser si la suite (u_n) est donnée de manière explicite c'est-à-dire sous la forme $(u_n) = (f(n))$?

Quelle méthode utiliser si la suite (u_n) est définie par récurrence par une relation du type $u_{n+1} = f(u_n)$ ou $u_{n+1} = f(u_n, n)$?

Exercice 5
Dans chacun des cas suivants, écrire une fonction prenant en argument n et renvoyant la liste `[`u_0`,`u_1`,`u_2`,...,`u_n`]`.

1. $\forall n \in \mathbb{N},\ u_n = \sqrt{n} + 1$
2. $u_0 = 2$ et $\forall n \in \mathbb{N},\ u_{n+1} = \sqrt{u_n} + 1$
3. $u_0 = 3$ et $\forall n \in \mathbb{N},\ u_{n+1} = \dfrac{n}{1 + u_n}$.

2 Parcours d'une liste par ses éléments

Une fois une liste créée, on peut la parcourir pour consulter un à un ses éléments. On utilise pour cela une boucle `for` avec la syntaxe suivante :

```
for x in L :
    # instructions exploitant (ou non) l'element x de la liste
```

Lors de cette boucle `for`, la variable `x` prend alors successivement la valeur de tous les éléments de `L`, *de la gauche vers la droite*. Par exemple, dans la boucle `for x in [5,-2,4.3,1]`, `x` prend, dans cet ordre, les valeurs :

Remarque 3
Il n'est pas recommandé d'appeler `k` la variable de la boucle `for` comme on le fait usuellement pour une boucle `for` utilisant un `range`. En effet, ici `x` désigne un élément de `L` donc ce n'est pas nécessairement un entier !

Exercice 6
Réalisez le "suivi" des variables `x` et `a` dans la boucle for ci-dessous :

```
a = 0
for x in [2,5,0,3,1]:
    a = a + x
```

Exercice 7
Écrire une fonction Python prenant en argument une liste de nombres réels `L` et renvoyant la somme de ses éléments.

Remarque 4
En fait, cette fonction existe déjà en Python, il s'agit de la fonction `sum`. Ainsi, `sum(L)` est la somme des éléments de `L`.

Exercice 8
Écrire une fonction Python prenant en argument une liste `L` et renvoyant son nombre d'éléments.

Remarque 5
En fait, cette fonction existe déjà en Python, il s'agit de la fonction `len`. Ainsi, `len(L)` est le nombre d'éléments de `L`.

Un algorithme classique est celui permettant d'obtenir le *minimum d'une liste* `L`.
On propose pour cela une méthode consistant à :

- initialiser une variable `mini` au premier élément de la liste qu'on obtient par `L[0]` (nous reviendrons sur comment accéder à un élément particulier d'une liste dans la feuille de cours "Listes 2"), puis
- parcourir les éléments de la liste, et actualiser la variable `mini` si on rencontre un élément plus petit.

Compléter la fonction suivante pour qu'elle renvoie le minimum d'une liste obtenu selon cette méthode :

```
1 def minimum(L):
2     mini =
3     for x in
4         if
5             mini = x
6     return mini
```

Quelles valeurs prennent les variables `x` et `mini` au cours de l'exécution de `minimum([4,5,1,2,3,0,-1,1])` ?

3 Premières opérations sur les listes

Concaténation :
On peut concaténer deux listes `L1` et `L2`, c'est-à-dire obtenir la liste des éléments de `L1` suivis des éléments de `L2`. Pour cela, on utilise l'opération `+`. Si par exemple `L1 = [1,2,'truc']` et `L2 = [2,'machin',2]` alors `L1 + L2` vaut `[1,2,'truc',2,'machin',2]`.
Pour concaténer plusieurs fois une liste avec elle-même, on peut utiliser l'opération `*` (avec un entier). Si par exemple `L = [2,True]` alors `L*3` vaut `[2,True,2,True,2,True]`.

Attention : Python n'effectue pas la somme des coefficients des listes. Ainsi `[2,6] + [1,3]` ne vaut pas `[3,9]` mais

Nombre d'éléments (longueur) : `len(L)` renvoie le nombre d'éléments de `L`.
Par exemple `len([2,'paf',[1,2]])` vaut

Somme des éléments : `sum(L)` renvoie la somme des éléments de `L` (si `L` contient des nombres). Par exemple `sum([4,9,-2])` vaut 11.

Appartenance :
On peut tester l'appartenance d'un élément à une liste grâce à l'opération `in`, le résultat obtenu est un booléen. Par exemple si `L = [2,3,6]` alors `2 in L` vaut `True` et `4 in L` vaut `False`.

Ajout d'un élément en fin de liste : `L.append(x)` rajoute `x` à la fin de la liste `L`.

Remarque 6
Il est aussi possible de supprimer ou de modifier des éléments d'une liste, voire feuille de cours "Listes 2".

4 Entraînement

Exercice 9
Donner trois manières de créer une liste contenant 1000 fois le nombre 1.

Exercice 10
Écrire une fonction prenant en argument un entier `n` (supérieur à 2) et renvoyant la liste `L`$=\left[\frac{1}{2},\ \frac{1}{3},\ \frac{1}{4},\ \ldots,\ \frac{1}{n}\right]$.

Exercice 11
Soit (u_n) la suite définie par : $u_1 = 2$ et $\forall n \in \mathbb{N}^*,\ u_{n+1} = 2u_n^2 + 3$. Écrire une fonction prenant en argument n et renvoyant la liste `[`u_1`,`u_1`,`u_2`,...,`u_n`]`.

Exercice 12
Écrire une fonction `produit` prenant en argument une liste `L` et renvoyant le produit de tous ses éléments. Par exemple `produit([2,5,3])` doit renvoyer 30.

Exercice 13
Écrire une fonction `maximum` prenant en argument une liste de nombres `L` et renvoyant son maximum.

Exercice 14
Écrire une fonction prenant en argument une liste de nombres réels `L` et renvoyant le nombre de fois que `L` contient `1`. Par exemple pour `L = [7,1,1,2,1,-1]` votre fonction doit renvoyer 3.

Exercice 15
Écrire une fonction `appartient` prenant en argument une liste `L` et une variable `a` et renvoyant `True` si `a` apparaît dans `L` et `False` sinon. On n'utilisera pas la fonction `in`.