

LES LISTES

1. Notion de liste

1.1 Définition

Définition. Une **liste** est une collection ordonnée d'objets.

On peut accéder à chacun des objets individuellement si l'on connaît son **indice**.

1.2 Syntaxe Python

Si une variable `L` est une liste alors son élément d'indice `i` s'obtient par la commande

```
| L[i]
```

En particulier, le **premier** objet de `L` a pour indice 0, c'est

```
| L[0]
```

1.3 Exemples

Considérons la liste `L=[8,1,'bonjour',14.523,'mardi',[5,2]]`.

- Dans le shell, on accède aux objets de `L` ainsi :

```
>>> L[0]
8      # réponse
>>> L[2]
'bonjour' # réponse
```

- Les objets dans une liste peuvent être de types différents.

Ici `L[0]` et `L[1]` sont des entiers, `L[2]` et `L[4]` sont des chaînes de caractères, `L[3]` est un réel (un flottant). L'objet `L[5]` est lui même une liste.

```
>>> type(L[0])
<class 'int'>
>>> type(L[2])
<class 'str'>
>>> type(L[3])
<class 'float'>
>>> type(L[5])
<class 'list'>
```

1.4 Quelques commandes en Python

- La longueur d'une liste `L` est le nombre d'objets qui la composent.

En Python, on utilise la commande `len`. Avec la liste `L` précédente, on a

```
>>> len(L)
6      # réponse
```

Attention ! En Python, le premier indice est 0. Donc une liste `L` contient `len(L)` objets dont les indices vont de 0 à `len(L)-1`.

- La liste vide est `[]`. Sa longueur est 0.
- La *concaténation* de deux listes `L1` et `L2` est la liste `L1 + L2` constituée de toute la liste `L1` suivie de toute la liste `L2`. Par exemple, si on entre dans l'éditeur :

```
L1 = [5, 'coucou', 3.14]
L2 = ['dodo', 78]
L3 = L1 + L2
```

alors, dans le shell on a :

```
>>> L3
[5, 'coucou', 3.14, 'dodo', 78]    # réponse
```

Il est possible de changer les éléments individuels d'une liste. Par exemple, en tapant dans l'éditeur :

```
L = [8,1, 'bonjour', 14.523, 'mardi', [5,2]]
L[2] = 4
L[4] = 'vendredi'
L[5] = 8.7
```

alors, dans le shell on obtient

```
>>> L
[8, 1, 4, 14.523, 'vendredi', 8.7]    # réponse
```

2. Construire une liste en Python

2.1 Construction directe

Si la liste contient peu d'objets on peut affecter directement les valeurs.

Par exemple, pour la liste `L` des quatre premiers carrés, on tape

```
L = [0, 1, 4, 9]
```

Il est fastidieux de créer ainsi la liste des 100 premiers carrés. Mais, comme on connaît sa longueur, on peut écrire,

```
L = [0] * n # initialisation d'une liste de longueur n ✓
           n remplie arbitrairement
for i in range(100):
    L[i] = i**2 # on modifie les éléments de L
```

2.2 La méthode `append`

Cette méthode consiste à augmenter la liste depuis son extrémité droite, élément par élément.

Elle peut servir quand on ne sait pas au départ quelle sera la longueur de la liste.

La syntaxe python est la suivante,

```
L = [] # initialisation avec la liste vide
L.append(élément_à_inclure)
```

Remarques.

- Bien penser à initialiser L.
- **Attention !** La méthode `append` modifie la liste L sur place et prend elle même en charge la nouvelle affectation de L.
Il est donc **inutile et faux** d'écrire `L = L.append (...)`

2.3 Construction par compréhension

Cette construction est très rapide et assez élégante. Elle consiste à affecter directement la liste en décrivant ses éléments à l'aide d'une boucle `for`.

La syntaxe python est la suivante,

```
| L = [élément_à_ajouter for ...]
```

Voici la construction par compréhension de la liste des 100 premiers carrés,

```
| L = [i**2 for i in range(100)]
```

► **Attention !** La commande `range` ne prend en paramètres **que des entiers**. Si l'un des paramètre est un nombre à virgule alors on obtient un message d'erreur.

```
>>> L = [i for i in range(0.1, 10, 1)]  
Traceback (most recent call last):  
  File "<pyshell#46>", line 1, in <module>  
    L=[i for i in range(0.1,10,1)]  
TypeError: 'float' object cannot be interpreted as an integer
```

3. Parcourir une liste en Python

Une liste L étant donnée, on souhaite la parcourir élément par élément, du premier au dernier.

C'est une boucle `for` qui sera utilisée.

3.1 Parcourir à l'aide de l'indice

La syntaxe python est la suivante,

```
| n = len(L) # n est le nombre d'objets dans L  
| for i in range(n):  
|     # on utilise L[i]
```

Remarques.

- La liste L contient n objets indexés de 0 à $n - 1$.
Dans la boucle `for i in range(n):`, le compteur i va de 0 à $n - 1$.
Donc tout va bien.
- **Attention !** En cas de dépassement, comme par exemple demander `L[n]` à une liste L contenant n éléments, c'est à dire telle que `n = len(L)`, le message d'erreur dans le shell sera

```
>>> L[n]  
IndexError: list index out of range
```

3.2 Parcourir à l'aide des éléments

La syntaxe `python` est la suivante,

```
for x in L:
    # on utilise x
```

Remarque. Plus de problème d'indice. Ce parcours efficace de toute la liste `L` permet de travailler directement avec ses objets.

4. Trancher une liste en Python

Une liste `L` étant donnée, on souhaite en isoler une partie contenant des éléments avec des indices consécutifs.

La syntaxe `python` est la suivante, dans l'éditeur puis dans le shell :

```
sous_L = L[i:j]

>>> sous_L
[ L[i], ... L[j-1]] # réponse
```

Remarques.

- La sous liste `L[i:j]` est une liste constituée des éléments `L[i]`, `L[i+1]`, ..., `L[j-1]`.
- La sous liste `L[i:j]` contient $j - i$ éléments.
- Son dernier élément n'est pas `L[j]`, c'est `L[j-1]`.

Exemple. Étant donnée la liste `L` des 10 premiers carrés, on a

```
L = [i**2 for i in range(10)]
sous_L = L[2:8]

>>> L
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> sous_L
[4, 9, 16, 25, 36, 49]
>>> sous_L[0]
4
>>> len(sous_L)
6 # 6 = 8 - 2
```

5. Listes et affectation

► **Attention !** L'instruction `L2 = L1` ne crée pas une copie de la liste `L1` ; les deux variables `L1` et `L2` désignent alors un même objet !

```
>>> L1 = [8, 1, -2, 9, 4]
>>> L2 = L1
>>> L1[0] = 2018
>>> L2
[2018, 1, -2, 9, 4]
```

Si on souhaite créer une copie d'une liste, on pourra :

- construire la nouvelle liste par compréhension :

```
| >>> L2 = [L1[i] for i in range(len(L1))]
```

ou encore :

```
| >>> L2 = [e for e in L1]
```

- utiliser une sous-liste contenant tous les éléments :

```
| >>> L2 = L1[0:len(L1)]
```

6. Éléments régulièrement répartis

Ce paragraphe est à mettre en lien avec le TP 3 sur les tracés.

Pour construire une liste de flottants (nombres à virgule réels) on utilise la commande `linspace` issue de la bibliothèque `numpy`.

Si on tape dans l'éditeur :

```
| import numpy as np #importation de la bibliothèque  
| L = np.linspace(0, 10, 9)
```

alors on a dans le shell :

```
| >>> L  
| array([ 0. ,  1.25,  2.5 ,  3.75,  5. ,  6.25,  7.5 ,  8.75, 10. ]) ✓  
| >>> L[1]  
| 1.25  
| >>> type(L)  
| <class 'numpy.ndarray'>
```

En toute rigueur, la commande `linspace` ne produit pas une liste mais un tableau de la bibliothèque `numpy`. Nous utiliserons ce type de variable pour représenter des matrices.

De façon générale, la commande

`np.linspace(a,b,n)`

avec a et b flottants et n un entier naturel, produit les n valeurs d'une subdivision régulière de l'intervalle $[a, b]$. Cette subdivision contient a comme plus petite valeur et elle contient b comme plus grande valeur.

```
| >>> L = np.linspace(0.5, 10.5, 11)  
| >>> L  
| array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5, 10.5]) ✓  
|                                     #11 valeurs
```