1 Listes

Les listes permettent de représenter une séquence de données de n'importe quel type. Pour déclarer une liste, on peut par exemple déclarer ses éléments entre deux crochets, et séparés par des virgules :

```
>>> L = [1, 5, 3, 4, 2]
>>> L
[1, 5, 3, 4, 2]
```

On accède aux éléments de la liste, par le nom de la liste, et l'**indice** (ou position) de l'élément dans la liste entre crochets :

```
>>> L[0]
1

>>> L[4]
2
```

L'indice du premier élément est 0, celui du second est 1, etc.

Le nombre d'élément d'une liste s'obtient grâce à la fonction **`len()`** :

```
>>> len(L)
5
```

Les indices des éléments d'une liste `L` varient de 0 à **`len`**`(L)-1`.

Ainsi le script suivant permet de parcourir les éléments d'une liste par indice croissant :

```
>>> for k in range(len(L)):
...     print(L[k])
...
1
5
3
4
2
```

Les listes sont **mutables** : on peut modifier l'élément d'une liste :

```
>>> L[0] = 12
>>> L
[12, 5, 3, 4, 2]
```

Attention : on ne peut pas ajouter un nouvel élément par cette méthode : `L[5] = 13` produira une erreur.

2 Opérations similaires sur les chaînes de caractères

La fonction **`len()`** peut aussi prendre en paramètre une chaîne de caractères ; elle renverra le nombre de caractères de la chaîne :

```
>>> len('ABCD')
4
```

Dans une chaine de caractère aussi, on peut accéder à ses éléments via leur indice :

```
>>> C = 'ABCD'
>>> for k in range(len(C)):
...     print(C[k]):
...
A
B
C
D
```

Par contre, les chaînes de caractères sont **non-mutables** : on ne peut pas changer un caractère dans une chaîne de caractère :

```
>>> C[0] = 'Z'
---------------------------------------------------------
TypeError
----> 1 C[0]='Z'

TypeError: 'str' object does not support item assignment
```

Attention : les chaînes de caractères sont, comme les listes, des **séquences** : on peut accéder à ses éléments (les caractères) via leur indice. Par contre, à la différence des listes, les chaînes de caractères sont **non-mutables** : on ne peut pas modifier ses caractères.

2.1 Ajout d'élément dans une liste

En Python, donnée une liste `L` et une expression `e`, l'instruction `L.append(e)` ajoute la valeur de `e` à la fin de la liste `L`.

```
>>> L = []
>>> L.append(1)
>>> L.append(2)
>>> L
[1, 2]
```

☞ La liste : `[]` désigne la **liste vide** ; elle ne contient aucun élément ; `len([])` retourne 0. Son usage est pratique pour déclarer une liste vide avant d'y insérer des éléments à l'aide de la méthode `append()`.

On parle de la **méthode** `append()` des listes ; son fonctionnement est similaire à une fonction, à ceci près que la liste `L` à laquelle elle s'applique ne figure par en argument mais en préfixe, suivi d'un point.

2.2 Suppression d'élément dans une liste

La méthode `pop()` effectue l'opération contraire de `append()` : elle retire l'élément en fin de liste, et le renvoie :

```
>>> L = [1, 2, 3]
>>> L.pop()
3
>>> L
[1, 2]
```

Si on lui passe en argument un indice `i`, la méthode `pop(i)` retire non pas le dernier élément, mais celui d'indice `i` :

```
>>> L = [1, 2, 3]
>>> L.pop(1)
2
>>> L
[1, 3]
```

Pour retirer l'élément à l'indice `i`, sans le renvoyer, on peut aussi utiliser l'instruction **`del`** `L[i]` :

```
>>> L = [1,2,3]
>>> del L[1]
>>> L
[1, 3]
```

☞ La méthode `L.pop(i)` est plus générale que l'usage de **`del`** `L[i]` ; on n'utilisera dans la suite que la méthode `pop()`. Cependant il vaut mieux connaître aussi l'usage de **`del`**.

2.3 Extraction de liste

Pour recopier une sous-liste extraite d'une liste, on utilisera la syntaxe suivante :

LISTE[Indice départ (inclus) : Indice arrivée (exclus) : Pas]

python crée la liste obtenue en copiant l'élément de `LISTE` d'indice `Indice de depart`, puis tous les éléments obtenus en ajoutant successivement `Pas` à l'indice, tant qu'on ne dépasse pas `Indice d'arrivee`. **Par défaut :** Pas = 1

- si Pas > 0 : **par défaut** départ = 0 ; arrivée = **`len`**(LISTE)
- si Pas < 0 : **par défaut** départ = **`len`**(LISTE)−1 : arrivée = -1

Exemple.

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f']
>>> L[0:2]       # de l'indice 0 à l'indice 1
['a', 'b']
>>> L[2:4]       # de l'indice 2 à l'indice 3
['c', 'd']
>>> L[::2]       # indices pairs
['a', 'c', 'e']
>>> L[1::2]      # indices impairs
['b', 'd', 'f']
```

2.4 Exercices

Exercice 1.

1. Écrire une fonction `suite(n)` prenant en paramètre un entier `n` et qui renvoie la liste des $(n+1)$ premiers termes $u_0, u_1, \dots, u_n$ de la suite (u_n) définie par :
$$u_0 = 1 \quad ; \quad u_{n+1} = -u_n^2 + 1$$
2. En déduire les listes des termes de rangs pairs $u_0, u_2, u_4, \dots u_{100}$ et des termes de rangs impairs $u_1, u_3, u_5, \dots u_9 9$.
3. Établir une conjecture sur la suite (u_n) et la démontrer.

Exercice 2.

Écrire une fonction `fibonacci(n)` prenant en paramètre un entier positif n et qui renvoie les $(n+1)$ premiers termes $F_0, F_1, \dots, F_n$ de la suite de Fibonacci :
$$F_0 = 0,\ F_1 = 1,\ \forall\, n \in \mathbb{N}, F_{n+2} = F_n + F_{n+1}.$$

Exercice 3.

1. Écrire une fonction `somme(L)` prenant en paramètre une liste de nombres `L` et qui renvoie la somme de ces nombres.
2. Utiliser cette fonction pour calculer $\sum_{k=1^1 00} k^4$.

Exercice 4.

1. Écrire une fonction `coefBinomiaux(n)` prenant en argument un entier `n` positif et qui renvoie la liste formée des coefficients binomiaux $\binom{n}{0}$, $\binom{n}{1}$,... $\binom{n}{n}$. On partira de la liste `[ 1 ]` (pour `n` égal à 0) en on construira successivement les listes des $\binom{m}{k}$ ($k \in [\![0, m]\!]$) pour m variant de 1 à n à l'aide de la relation de Pascal :

$$\binom{m+1}{k} = \binom{m}{k-1} + \binom{m}{k}$$

et des 2 valeurs extrémales :

$$\binom{m}{0} = \binom{m}{m} = 1$$

2. Écrire une fonction `formuleBinome(n)` prenant en argument un entier positif `n` et qui écrit dans la console le développement de $(a+b)^n$ donné par la formule du binôme.