# Mémento : les listes en Python

Une liste est une collection ordonnée d'objets repérés par leur indice : le 1er élément a pour indice 0, le 2e élément a pour indice 1… Une liste en Python est de type list et ses éléments sont de type quelconque. On parle alors de liste *hétérogène* même si, en pratique, on utilise le plus souvent des listes *homogènes*, dont tous les éléments sont de même type.

## 1. Initialisation d'une liste

On peut donner une valeur à une liste *en extension* : la liste est notée entre crochets, ses éléments séparés par une virgule.

```
L1 = [ 1, 3, 5 ]                  # L1 est une liste d'entiers comportant trois éléments
L2 = [ ]                          # L2 est une liste vide
L3 = [ [1, 2], [0, 9, 3] ]        # L3 est une liste de listes d'entiers…
L4 = [ "lundi", "mardi" ]         # L5 est une liste de chaînes de caractères
```

On peut également convertir un objet de type str (chaîne) ou range (intervalle) en list (liste) :

```
L5 = list ( "salut" )             # L5 vaut alors [ "s", "a", "l", "u", "t" ]
L6 = list ( range(5) )            # L6 vaut alors [ 0, 1, 2, 3, 4 ]
L7 = list ( range(1,15,2) )       # L7 vaut alors [ 1, 3, 5, 7, 9, 11, 13 ]
```

Il est enfin possible de définir une liste *en compréhension*, en utilisant le format général suivant :

maListe = [ *expression avec élément* **for** élément **in** uneListe **if** *condition sur élément* ]

```
L8 = [ x*x for x in range(5) ]                  # L8 vaut alors [ 0, 1, 4, 9, 16 ]
L9 = [ y+1 for y in L8 if (1 < y) and (y < 14) ]  # L9 vaut alors [ 5, 10 ]
```

## 2. Fonctions de base sur les listes

- Accès aux éléments d'une liste : si L est la liste [ 1, 3, 5, 10 ], alors L[0] est l'entier 1, L[2] est l'entier 5, etc. Si L est une liste de listes, par exemple L = [ [1, 2], [1, 5, 10] ], alors L[1][2] est l'entier 10.
- Longueur d'une liste (nombre d'éléments) : len(L) retourne le nombre d'éléments de la liste L. Ainsi, si L est la liste [ 1, 3, 5, 10 ], alors len(L) retourne la valeur 4. Les éléments d'une liste L sont donc L[0], …, L[len(L) – 1].
- Concaténation de deux listes

```
L1 = [ 1, 12 ] ; L2 = [ 8, 5 ] ; L3 = L1 + L2          # L3 vaut alors [ 1, 12, 8, 5 ]
```

- « Multiplication » d'une liste : concaténation itérée…

```
L4 = [ 1, 3 ]
L5 = L4 * 3                       # L5 vaut alors [ 1, 3, 1, 3, 1, 3 ]
L6 = [ 0 ] * 8                    # L6 vaut alors [ 0, 0, 0, 0, 0, 0, 0, 0 ]
```

- Test de présence d'un élément dans une liste : il est possible de tester la présence d'un élément dans une liste grâce à l'opérateur in. Ainsi, si L est la liste [ 1, 3, 5, 10 ], alors « 7 in L » retourne la valeur False, alors que « 10 in L » retourne la valeur True.
- Nombre d'occurrences d'un élément dans une liste : L.count(8) retourne le nombre d'occurrences de 8 dans la liste L.
- Indice d'un élément dans une liste : L.index(25) retourne l'indice de l'élément 25 dans la liste L (erreur si 25 absent).

## 3. Extraction de sous-listes

Il est possible d'extraire une sous-liste d'une liste donnée grâce à l'opérateur ':' qui permet de définir l'intervalle d'indices concerné (l'écriture « deb : fin » correspond à l'intervalle « [ deb, fin [ »). Si le premier indice n'est pas spécifié, la valeur 0 est prise par défaut. Si le deuxième indice n'est pas spécifié, la valeur len(laListe) est prise par défaut.

```
L1 = [ 1, 3, 5, 7, 9, 11, 13, 15 ]
L2 = L1 [ 1:4 ]                   # L2 vaut alors [ 3, 5, 7 ]
```

```
L3 = L1 [ :5 ]          # L3 vaut alors [ 1, 3, 5, 7, 9 ]
L4 = L1 [ 5: ]          # L4 vaut alors [ 11, 13, 15 ]
L5 = L1 [ : ]           # L5 vaut alors [ 1, 3, 5, 7, 9, 11, 13, 15 ]
```

## 4. Ajout d'éléments dans une liste

- Ajout d'un élément en fin de liste : append, ou concaténation d'une liste singleton

```
L1 = [ 1, 3 ]
L1.append (18)          # L1 vaut alors [ 1, 3, 18 ]
L1 = L1 + [ 35 ]        # L1 vaut maintenant [ 1, 3, 18, 35 ]
```

On peut ainsi « réécrire » la construction en compréhension des listes L8 et L9 (voir section 1) ainsi :

```
L8 = [ ]                        L9 = [ ]
for x in range(5) :             for y in L8 :
    L8.append(x*x)                  if (1 < y) and (y < 14) :
                                        L9.append(y+1)
```

- Ajout de plusieurs éléments en fin de liste : extend (équivalent à la concaténation)

```
L2 = [ 6, 8, 24 ]
L2.extend ( [ 3, 4, 5 ] )       # L2 vaut alors [ 6, 8, 24, 3, 4, 5 ]
```

- Insertion d'un élément en position quelconque : insert

```
L3 = [ 1, 4, 7, 11 ]
L3.insert (3,45)        # insère l'entier 45 qui aura l'indice 3, L3 vaut alors [ 1, 4, 7, 45, 11 ]
```

## 5. Remplacement d'éléments dans une liste

```
L1 = [ 2, 4, 6, 8, 10, 12, 14 ]
L1[ 2 ] = 21            # L1 vaut alors [ 2, 4, 21, 8, 10, 12, 14 ]
L1[ 4:6 ] = [ 0, 15 ]   # L1 vaut maintenant [ 2, 4, 21, 8, 0, 15, 14]
```

## 6. Suppression d'éléments dans une liste

- Suppression d'un élément d'indice donné : del

```
L1 = [ 0, -3, 5, -14, 77 ] ; del ( L1[2] )      # L1 vaut alors [ 0, -3, -14, 77 ]
```

- Suppression de la 1re occurrence d'un élément donné : remove

```
L2 = [ 1, 4, 7, 4, 15 ] ; L2.remove (4)         # L2 vaut alors [ 1, 7, 4, 15 ]
```

- Suppression (et récupération) du dernier élément d'une liste : pop

```
L3 = [ 8, 67, 32, 28 ] ; elem = L3.pop()        # elem vaut alors 28, et L3 vaut [ 8, 67, 32 ]
```

## 7. Parcours des éléments d'une liste

- Parcourir les éléments d'une liste L pour effectuer un traitement : for elem in L…

## 8. Identification vs recopie

**Attention.** L'affectation de listes en Python, par exemple « L1 = L2 », est une *identification* ! Les deux listes L1 et L2 ont maintenant le « même contenu » et, ainsi, toute modification de L2 modifie également L1 (et réciproquement) !

- Si l'on souhaite que L2 soit une *copie* de L1, et donc que L1 et L2 puissent évoluer indépendamment l'une de l'autre, il faut écrire « L2 = list (L1) » ou, comme vu précédemment, « L2 = L1 [ : ] ».
- Si L1 est une liste de listes, on devra aussi copier les éléments de L1 en écrivant « L2 = [ list (elem) for elem in L1 ] ».