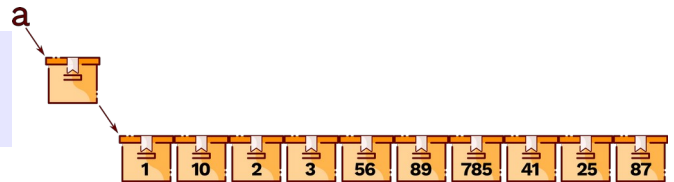


# Construction de listes

## 1. Déclaration d'une liste (rappel)

Sous Python, on peut construire une liste comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets. Exemple :

```
>>>a=[1,10,2,3,56,89,785,41,25,87]
>>>a
[1,10,2,3,56,89,785,41,25,87]
```



La première instruction crée une variable `a` dont le contenu fait référence à une liste de 10 entiers.

## 2. Construction de grandes listes

Si on doit construire une liste vraiment grande, il devient difficile de le faire en énumérant tous ses éléments. Différentes manières d'initialiser la liste peuvent alors être utilisées.

### Construction d'une liste avec une valeur unique.

On peut utiliser l'opérateur `*` pour indiquer la longueur de la liste :

```
>>>a=[0]*1000
>>>len(a)
1000
>>> a[0]
0
>>> a[-1]
0
```

On indique la taille après le symbole `*`, ici 1000 et entre crochets la valeur affectée à chaque case de la liste, ici 0. On obtient donc une liste de taille 1000 dont toutes les cases contiennent la valeur 0.

### Listes par compréhension

La liste en compréhension permet d'écrire des boucles `for` plus concises. Très utiles dès lors que l'on souhaite créer une nouvelle liste basée sur une pré-existante. Par exemple, on peut utiliser une liste en compréhension pour créer une liste contenant les carrés des 1000 premiers entiers. Classiquement ce programme peut s'écrire :

```
a =[0]*1000
for i in range(1000) :
    a[i] = i*i
```

Python propose une syntaxe simplifiée pour combiner l'allocation de la liste et son remplissage par la notation suivante :

```
>>> a = [i*i for i in range(1000)]
>>> a
[0, 1, 4, 9, 16, ..., 996004, 998001]
```

Cette nouvelle construction mélange les crochets, qui explicitent la construction d'une liste, et les mots-clés de la boucle `for` de Python, qui explicitent le remplissage de la liste avec une boucle. Cette structure s'appelle une **liste par compréhension**.

Dans cette construction, le parcours de la variable `i` n'est pas limitée à un intervalle d'entiers construits avec `range`. On peut parcourir une autre liste :

```
>>> t = [3*i+1 for i in range(10)]
>>> a = [x *x for x in t]
>>> t
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28]
>>> a
[1, 16, 49, 100, 169, 256, 361, 484, 625, 784]
```

Il est possible de ne conserver que certaines valeurs prises par la variable, en ajoutant une condition booléenne à la compréhension, avec le mot-clé `if`

```
>>> a = [i * i for i in range(30) if i % 4 == 1]
>>> a
[1, 25, 81, 169, 289, 441, 625, 841]
```

