

Manipulation de Listes en Python

Modification, Ajout, Suppression et Gestion de la Mémoire

Basé sur la liste : fam

État des lieux : La liste de départ

Nous utilisons la liste `fam` contenant les prénoms et tailles de différents membres de la famille. Voici l'état initial des données et leurs index correspondants.

```
fam = ['Liz', 1.73, 'Emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

Index	0	1	2	3	4	5	6	7
Value	'Liz'	1.73	'Emma'	1.68	'mom'	1.71	'dad'	1.89

Notez bien que l'Indexation commence toujours à 0.

Modification d'un élément unique

Les données évoluent. Par exemple, la taille de 'dad' n'est plus à jour (il perd des centimètres avec l'âge). Il faut remplacer `1.89` par `1.86`.

```
fam[7] = 1.86
```

... (Indices 0-5 inchangés) ...			
+-----+-----+-----+			
Index 6 7			
+-----+-----+-----+			
Value 'dad' 1.86 ← Valeur mise à jour			
+-----+-----+-----+			

L'accès par crochets [] permet d'écraser l'ancienne valeur assignée à cet index.

Modification par plage (Slicing)

Vous pouvez modifier plusieurs éléments simultanément en sélectionnant une plage (slice).

```
fam[0:2] = ['Lisa', 1.74]
```

Index	0	1	2	...
Value	'Lisa'	1.74	'Emma'	...

▲ ▲
|_____|_ Plage remplacée

La syntaxe [début:fin] cible les éléments de l'index 0 inclus à l'index 2 exclu. Une nouvelle liste est assignée à cette sélection.

Ajout d'éléments : L'opérateur +

Contrairement aux entiers, l'opérateur `+` ne fait pas de somme mathématique. Il 'colle' deux listes ensemble. Ajoutons 'moi' et une taille à la liste.

```
fam_ext = fam + ['moi', 1.79]
```

... (Indices 0-5) ...

Index	6	7	8	9
Value	'dad'	1.86	'moi'	1.79

^ ^
|_____|_ Nouveaux éléments

La liste originale reste intacte si le résultat n'est pas réassigné, ou vous pouvez stocker le résultat dans une nouvelle variable (fam_ext).

Suppression d'éléments

Pour retirer un élément, on utilise l'instruction `del`. Supprimons 'Emma' qui se trouve à l'index 2.

```
del(fam[2])
```

Index	0	1	2	3	
Value	'Liz'	1.73	(Vide)	1.68	...
					▲ Élément supprimé

L'élément à l'index 2 est physiquement retiré de la structure.

Attention au décalage d'index !

Après une suppression, tous les éléments suivants se décalent vers la gauche pour combler le vide. Les index changent.

Avant suppression :

Index	2	3
Value	'Emma'	1.68

Élément à supprimer

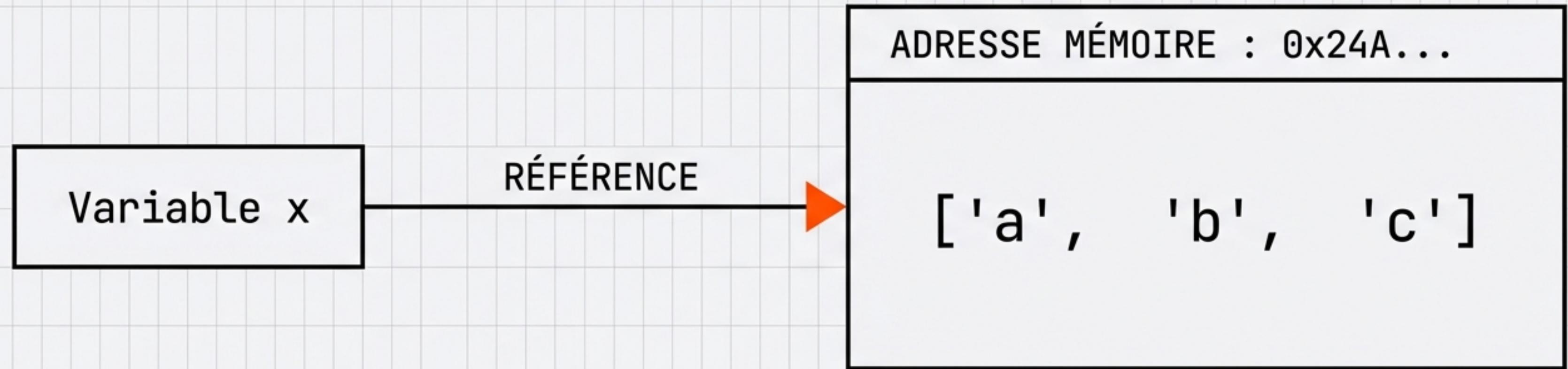
Après suppression de 'Emma' :

Index	2	3
Value	1.68	'mom'

Attention : Si vous exécutez `del(fam[2])` une seconde fois, vous ne supprimerez pas 'Emma' (déjà partie), mais sa taille ('1.68), qui occupe désormais l'index 2.

Sous le capot : La Mémoire

Que se passe-t-il vraiment quand on crée `x = ['a', 'b', 'c']` ?

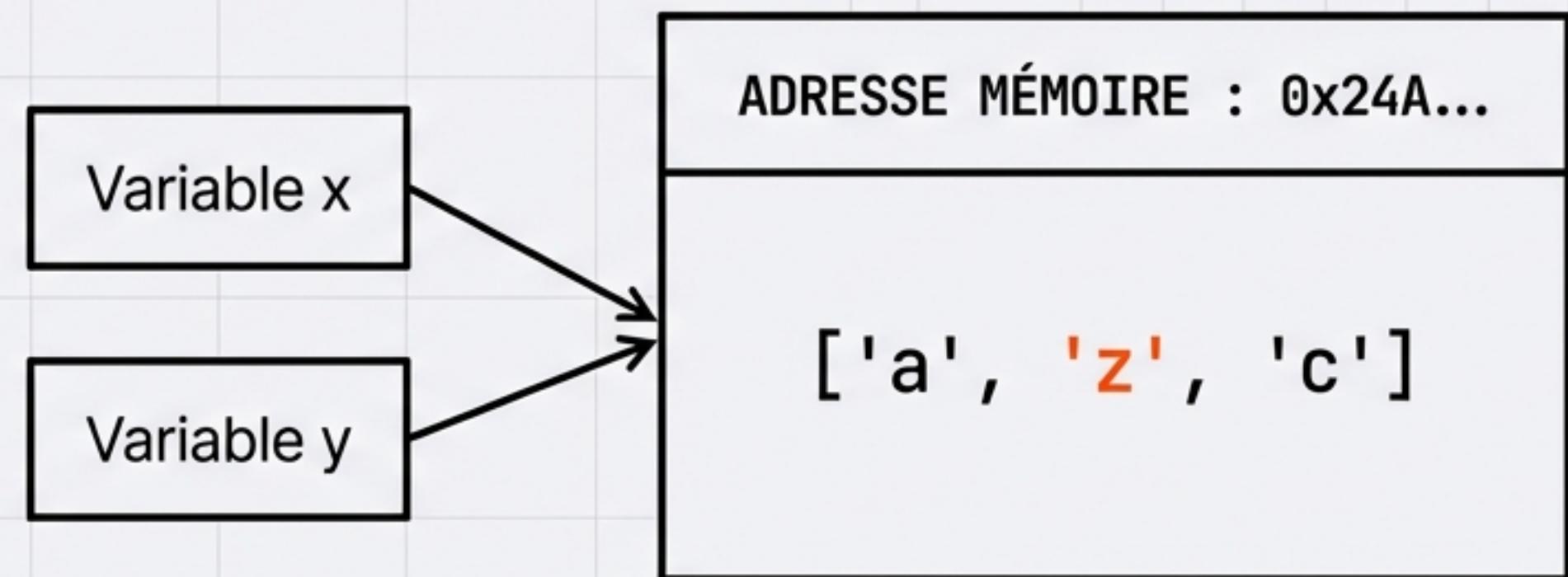


La variable `x` ne contient pas la liste elle-même. Elle contient une **référence** (l'adresse) de l'endroit où la liste est stockée en mémoire.

Le piège de la copie par référence

Si vous copiez une liste avec "=", vous copiez seulement l'adresse, pas les valeurs.

```
x = ['a', 'b', 'c']
y = x
y[1] = 'z'
```



Résultat pour y : `['a', 'z', 'c']`

Résultat pour x : `['a', 'z', 'c']` **<-- MODIFIÉ AUSSI !**

`'x'` et `'y'` pointent vers la **même** liste en mémoire. Modifier l'un affecte l'autre.

La solution : Créer une vraie copie

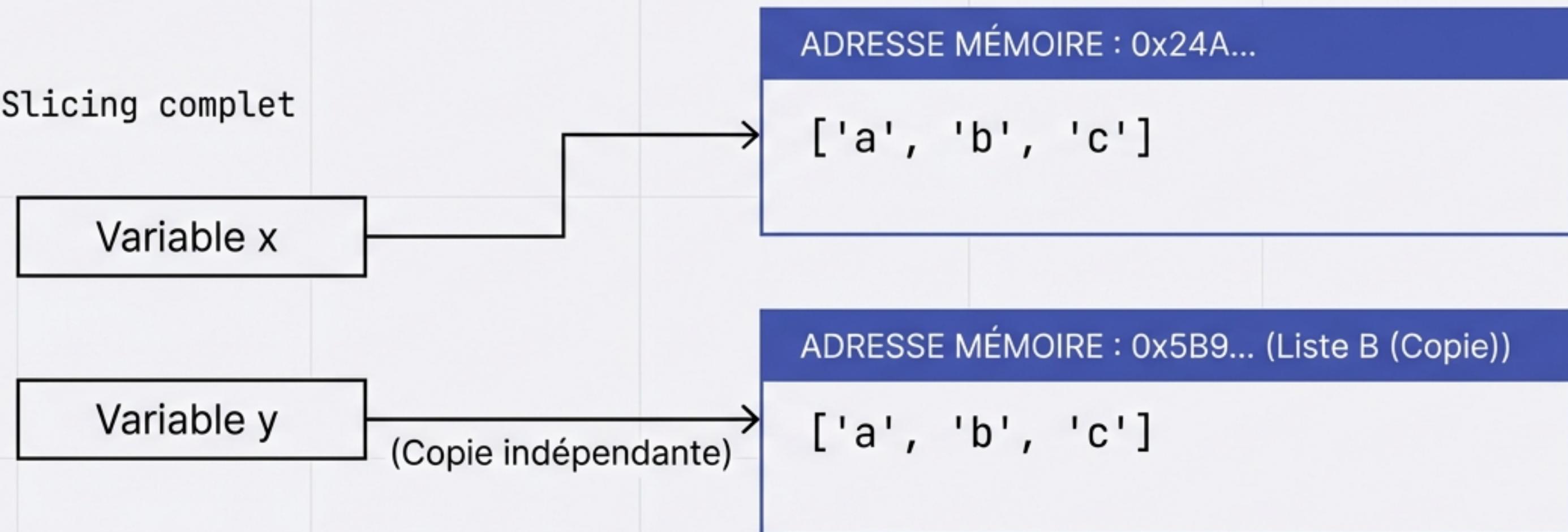
Pour manipuler une copie sans affecter l'original, il faut générer une nouvelle liste en mémoire.

```
# Méthode 1 : Fonction list()
```

```
y = list(x)
```

```
# Méthode 2 : Slicing complet
```

```
y = x[:]
```



Modifier y **ne changera plus** x.

Récapitulatif des commandes

ACTION	SYNTAXE	DESCRIPTION
----- + ----- + -----		
Modifier (Un)	<code>liste[x] = val</code>	Change la valeur à l'index x
Modifier (Plage)	<code>liste[x:y] = [...]</code>	Remplace une tranche
Ajouter	<code>liste + [...]</code>	Colle deux listes
Supprimer	<code>del(liste[x])</code>	Retire l'élément (décalage!)
Copier (Lien)	<code>y = x</code>	Copie la référence (Attention)
Copier (Vrai)	<code>y = list(x) ou x[:]</code>	Crée une nouvelle liste