# Modification d'une liste en Python

**1.** Une variable informatique est constituée d'un **identifiant** (= le "nom" de la variable) et d'une **valeur**. En fait, l'identifiant n'est qu'un moyen mnémotechnique d'accéder à un emplacement de la mémoire, dans lequel la valeur est enregistrée.

On peut accéder à l'adresse d'un objet avec la fonction `id`.

```
x = 12
print(id(x))
A = [1,2,3]
print(id(A))
```

**2.** Quand un objet est simple (entier, réel), il occupe peu de place en mémoire. Quand on modifie la valeur d'un tel objet, Python crée un nouvel objet, à un nouvel emplacement.

```
x = 12
print(id(x))
x = 13
print(id(x))
```

L'instruction `y=x` associe un second identifiant au même objet (même adresse en mémoire, donc même valeur). Si on modifie ensuite la valeur affectée à l'identifiant `x`, un nouvel objet est créé et associé à cet identifiant. On a deux adresses distinctes.

```
x = 12
y = x
print(id(x)==id(y))     # True
x = 11
print(id(x)==id(y))     # False
```

**3.** Quand un objet est composé (liste...), il peut occuper beaucoup de place en mémoire. Pour éviter de gaspiller la place, la modification du contenu de l'objet ne crée pas un nouvel objet.

```
A = [1,2,0]
print(id(A))
A[-1] = 3
print(id(A))
```

L'**affectation** `B=A` associe un second identifiant au même objet. Si on modifie la valeur affectée à l'identifiant `A`, on modifie simultanément la valeur affectée à l'identifiant `B` (puisqu'aucun nouvel objet n'est créé) : on n'a toujours qu'une seule adresse en mémoire.

```
A = [4,5,6]
B = A                   # même objet, mêmes valeurs
print(id(A)==id(B))     # True
A[0] = 0
print(id(A)==id(B))     # True
print(A, B)             # [0,5,6] [0,5,6]
```

**4.** La **copie superficielle** (*shallow copy*) permet de créer un nouvel objet avec les mêmes valeurs. On peut alors modifier les valeurs de l'un sans modifier les valeurs de l'autre.

```
A = [4,5,6]
B = A.copy()            # nouvel objet, mêmes valeurs
print(id(A)==id(B))     # False
print(A==B)             # True
A[0] = 0
print(A==B)             # False
print(A, B)             # [0,5,6] [4,5,6]
```

**5.** On doit calculer les puissances d'une matrice $A$. Imaginons que cette matrice soit représentée par une *liste de listes* `A`. Les puissances $A^k$ seront représentées par une *liste de listes* `B`.

Initialement, c'est-à-dire pour $k = 1$, la valeur de `B` est aussi la valeur de `A`. Mais la valeur de `B` doit être modifiée à chaque étape. Il faut donc définir un autre objet avec les mêmes valeurs.

La copie superficielle ne marche pas : si on modifie un coefficient de `B`, on modifie ce coefficient dans `A`. Bien que les objets soient distincts, leurs valeurs restent les mêmes !

```
A = [[1,2],[3,4]]
B = A.copy()            # objets distincts, mêmes valeurs
print(id(A)==id(B))     # False
print(A==B)             # True
B[0][0] = 0
print(A==B)             # True
```

En fait, on a réalisé une copie de la liste `A` et on peut donc modifier la valeur de `B` sans modifier la valeur de `A`.

```
B.append([5,6])
print(A==B)             # False
```

Mais comme les éléments de la liste `A` sont des *listes* (c'est-à-dire des objets composés), la copie superficielle n'a pas créé de nouveaux objets pour définir les éléments de `B`.

```
A = [[1,2],[3,4]]
B = A.copy()
print(id(A)==id(B))       # False
print(id(A[0])==id(B[0])) # True
B[0][0] = 0
print(id(A[0])==id(B[0])) # True
B.append([5,6])
print(id(A[0])==id(B[0])) # True
```

Comme `A` et `B` sont des objets différents composés des mêmes éléments, si on modifie une liste dans la liste `A`, cette modification apparaît aussi dans la liste `B` (puisqu'il n'y a qu'une seule liste).

**6.** Pour créer deux objets différents avec les mêmes valeurs, il faut donc réaliser une **copie profonde** (*deep copy*).

Ça peut se faire avec les moyens du bord, en créant pas à pas une nouvelle liste de listes à partir de `A`.

```
B = []
for ligne in A:
  L = []
  for x in ligne:
    L.append(x)
  B.append(L)
```

On peut aussi profiter du travail des autres.

```
import copy

B = copy.deepcopy(A)
```

**7.** Le mieux est d'utiliser le travail des autres (en pensant à dire *merci*) et de **ne pas travailler avec des listes de listes**.

```
import numpy as np

A = np.matrix([[1,2],[3,4]])
B = A.copy()
print(A==B)             # mêmes valeurs
print(id(A)==id(B))     # objets différents
B[0,0] = 0
print(A==B)             # valeurs différentes
A.dot(B)                # produit matriciel A.B
```

Mais voilà... On est parfois obligé de se plier à une discipline qui n'est pas toujours éclairée...