

*J.Courtin*

*PC\* LVH - Caen*

---

# INFORMATIQUE

Notion d'objet  
et  
Référence partagée

---

# 1 - Notion d'objet et références partagées

En programmation objet, toute variable est une écriture symbolique qui pointe vers une adresse mémoire contenant l'objet : on parle d'instance de l'objet.

Chaque objet est ainsi défini par une collection de propriétés qui seront gardées en mémoire dès que l'objet est créé : on parle d'instance de l'objet.

- Il peut y avoir plusieurs instances d'un même type d'objet :

```
L1 = [1,2,3]
```

```
L2 = ['A','B','C'] # Deux objets de la classe liste
```

- deux noms de variables distincts peuvent désigner ou pointer la même instance :

```
maListe = L1 # L1 et maListe désigneront la même adresse mémoire.
```

Cela permet de ne pas garder inutilement en mémoire la même information.

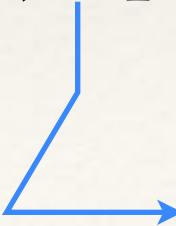
 Référence partagée

CONCLUSION : Ne pas confondre la variable et l'objet pointé temporairement.

«espace de nommage»

```
>>> dir()
```

adresse mémoire



## Exemple :

code

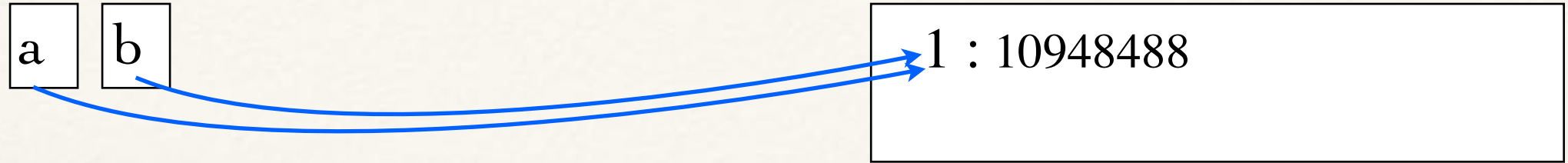
```
>>> a=1  
>>> b=a
```

variables

a b

instance d'objet

1 : 10948488



The diagram illustrates the state of memory after the code execution. Two variables, 'a' and 'b', are shown in boxes. Two blue arrows originate from these boxes and point to a single object instance '1 : 10948488' in a larger box. This indicates that both variables reference the same object in memory.

`b = a` : signifie que `b` va pointer la même instance que celle désignée par `a`  
[et surtout pas que `b` et `a` seraient égaux : ce n'est que provisoire]

code

```
>>> a=2
```

variables

a b

instance d'objet

1 : 10948488

2 : 10948476



The diagram illustrates the state of memory after the code execution. Variable 'a' is shown in a box with a red arrow pointing to object '2 : 10948476'. Variable 'b' is shown in a box with a blue arrow pointing to object '1 : 10948488'. This indicates that 'a' now points to a new object, while 'b' still points to the original object.

`a = 2` : `a` va pointer une autre instance d'entier, mais `b` reste inchangée.

```
>>> print( a-b, id(a-b), a/b, id(a/b) )
```

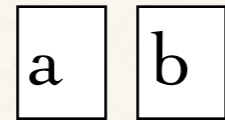
```
(1, 10948488, 2, 10948476)
```

**Conséquence : les entiers 1 et 2 ne sont représentés qu'une fois en mémoire.**

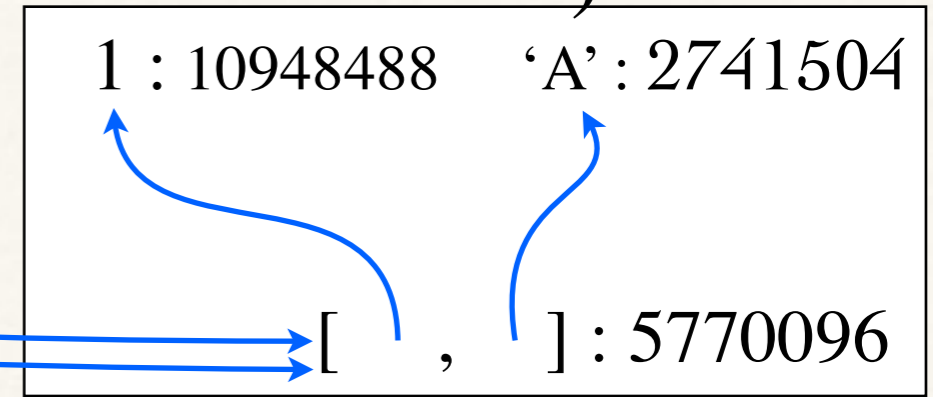
code

```
>>> a=[1,'A']
>>> b=a
```

variables



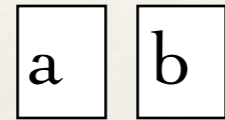
instance d'objet



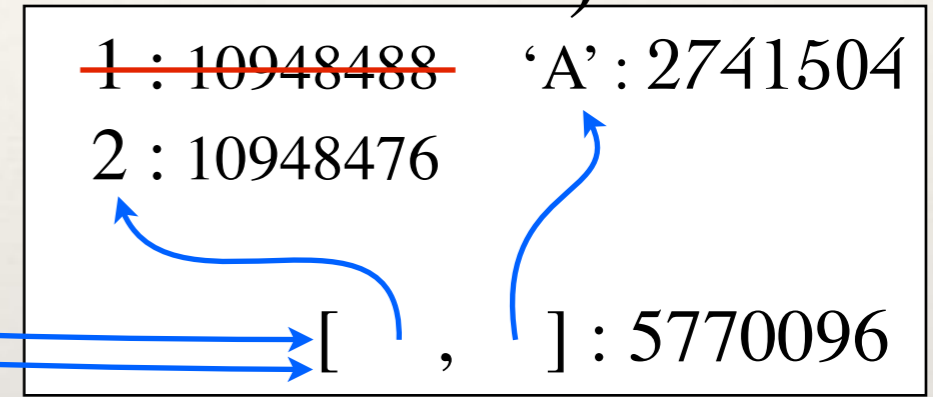
code

```
>>> a[0]=2
```

variables



instance d'objet



code

```
>>> b[0]
2
```

```
>>> id(b)
5770096
```

**Rq:** L'affectation `b=a` ou `a[0]=2` ne fait qu'aiguiller une variable vers un objet ou vers un autre.

La commande **del(variable)** permet de supprimer la variable de «l'espace de nommage» [=> le gestionnaire pourra si nécessaire libérer la mémoire qu'il occupait à condition qu'aucune autre variable ne pointe sur cet objet (Python tient les comptes...)]

↙ sys.getrefcount()

# Module copy : permet de créer un nouvel objet

code

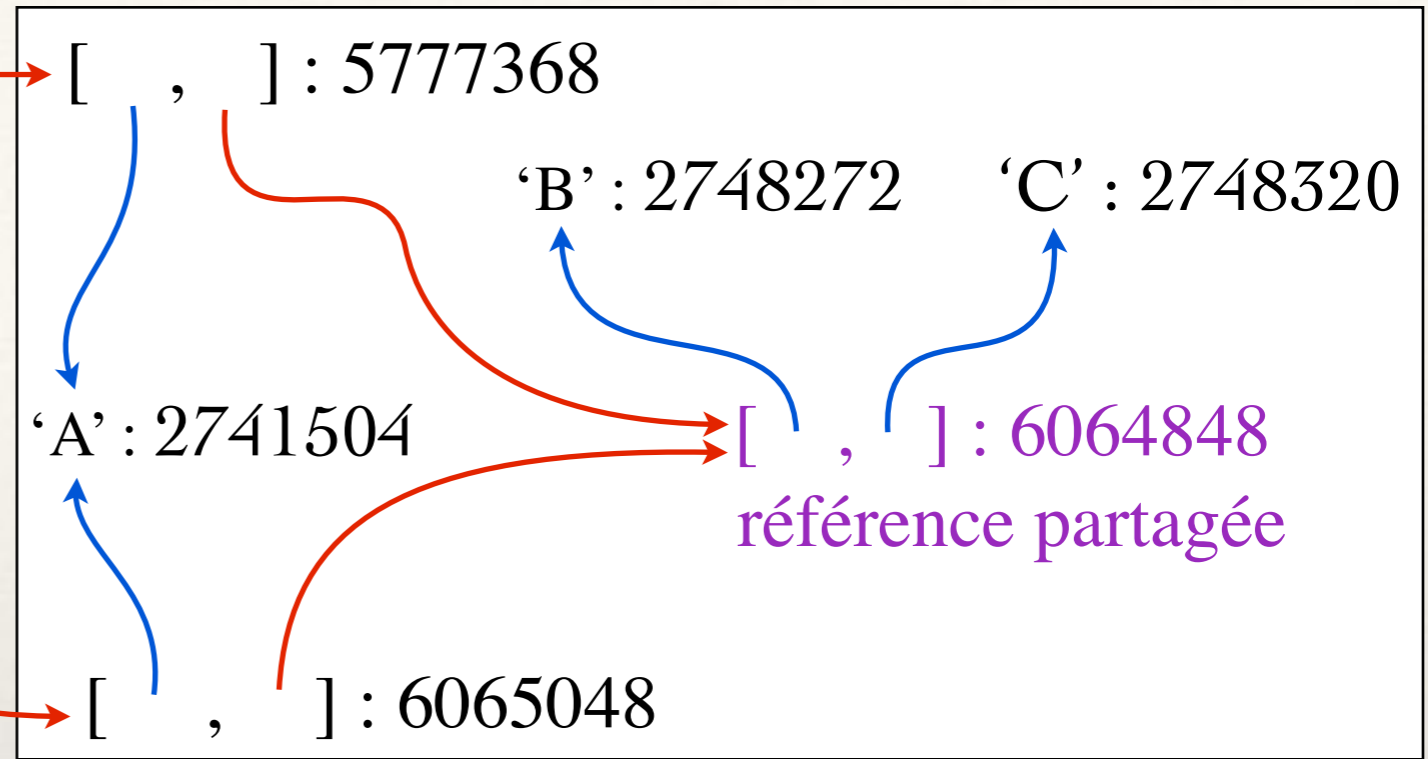
```
>>> a=['A',['B','C']]
>>> id(a), id(a[1])
(5777368, 6064848)
>>> b =copy.copy(a)
>>> id(b), id(b[1])
(6065048, 6064848)
```

variables

a

b

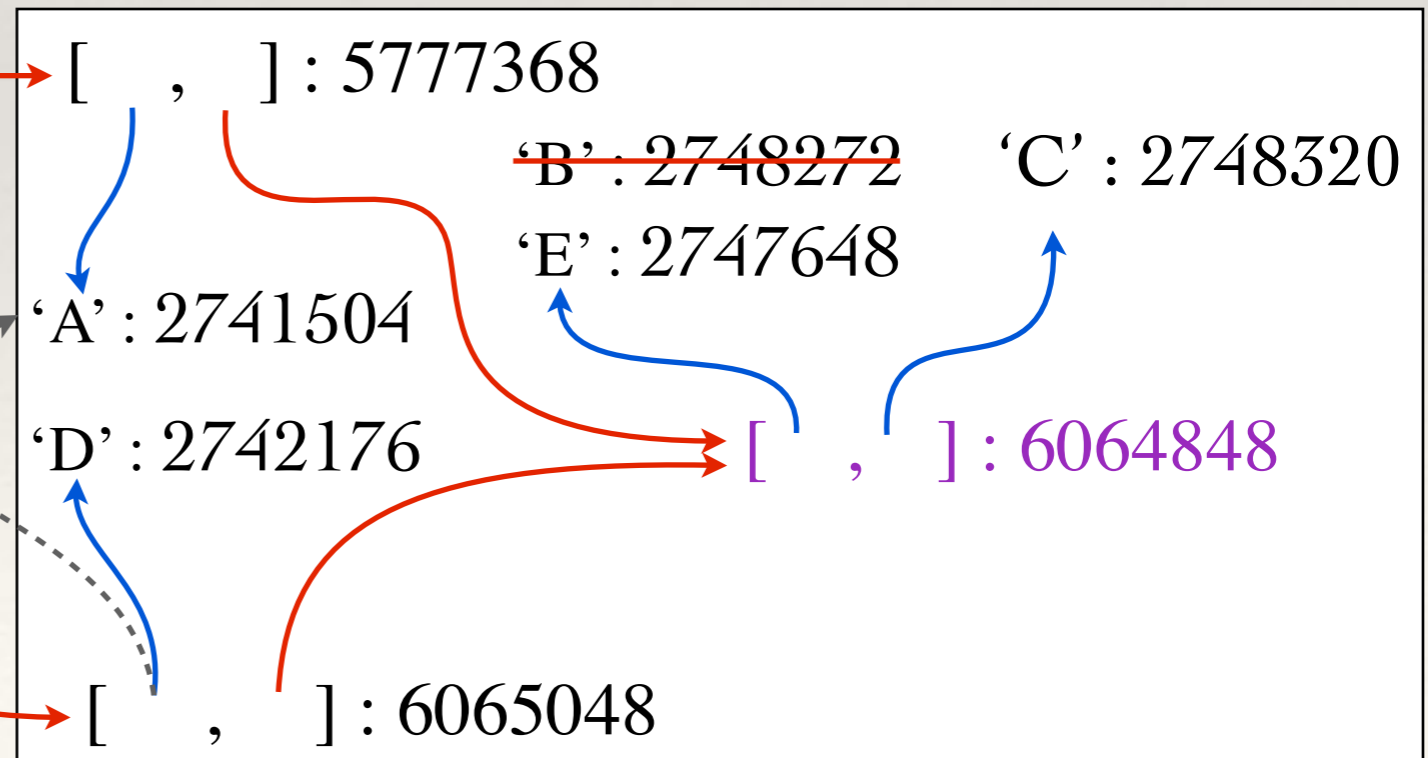
instance d'objet



```
>>> b[0]='D'
>>> b
['D',['B','C']]
>>> a[1][0]='E'
['A',['E','C']]
>>> b
['D',['E','C']]
```

a

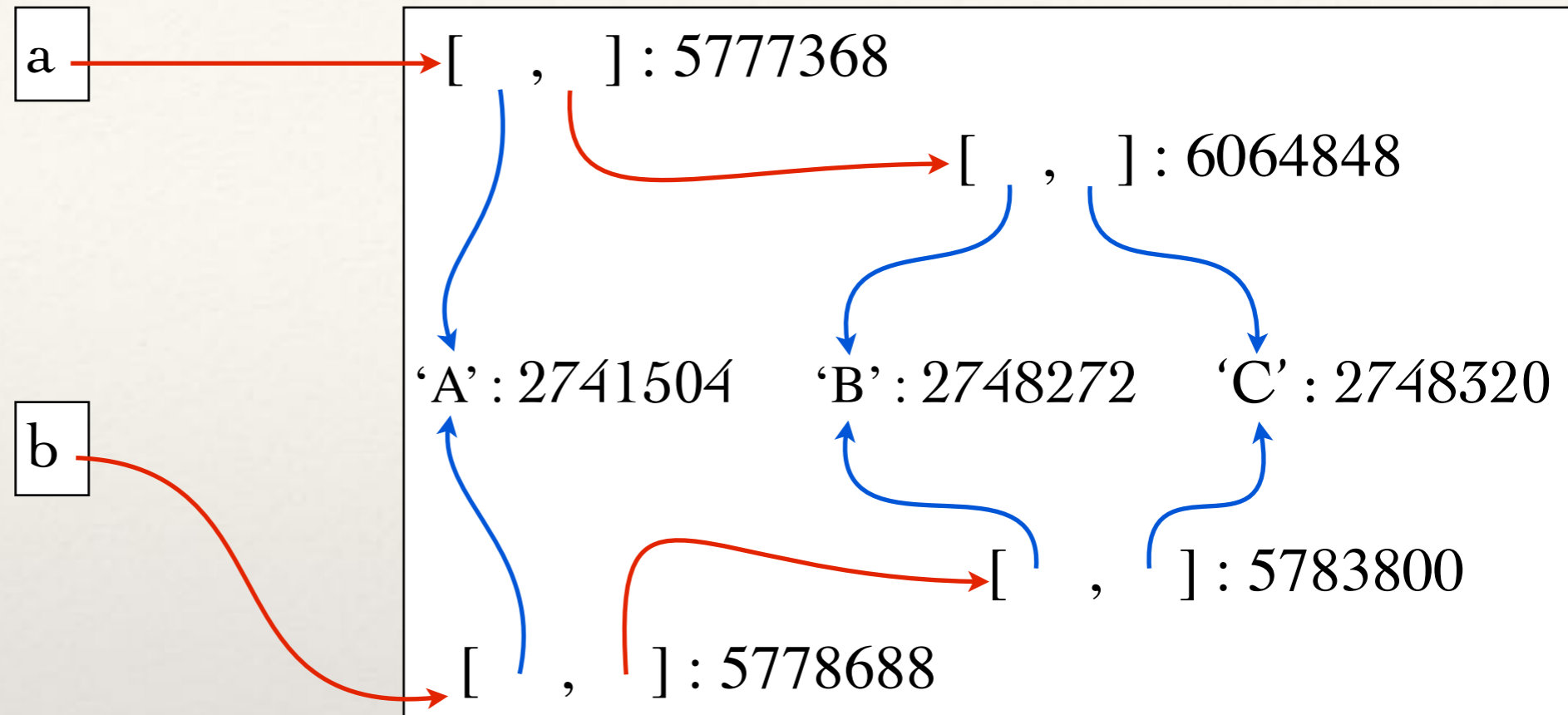
b



# deepcopy : copie récursive de l'objet

code                      variables                      instance d'objet

```
>>> a=['A',['B','C']]
>>> id(a), id(a[1])
(5777368, 6064848)
>>> b =copy.deepcopy(a)
>>> id(b), id(b[1])
(5778688, 5783800)
```



- b est initialisée avec la même structure et les mêmes valeurs que a  
=> c'est structurellement le même «objet» mais sous d'autres identifiants.
- a et b sont totalement indépendantes : plus de référence partagée
- Récursif => Ceci reste vrai quel que soit le niveau d'imbrication [,[,[,[ [ ] ] ] ] ]

## Conséquence :

Une altération de a ne se répercutera plus sur b et réciproquement.

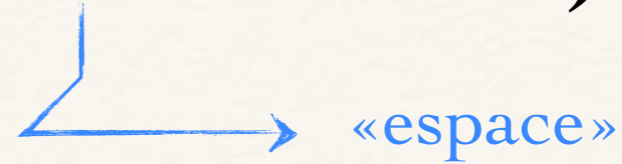
# En Python : « Tout est objet !!! »

=>

Penser objet tu dois !



# Quelle est la taille en mémoire des objets ?



Le module `sys` permet d'accéder à la taille des objets `sys.getsizeof()` :

```
>>> sys.getsizeof("Toto")  
25
```

```
>>> sys.getsizeof(3.14159)  
16
```

```
>>> sys.getsizeof(None)  
8
```

Le résultat est indiqué en bytes [1 byte assimilé à un octet soit 8 bits]

Globalement, la création d'objets structurés impose de garder plus d'information, et se traduit à la création par un surcoût en mémoire :

- L'information supplémentaire permet en revanche de gagner du temps de calcul grâce à des stratégies algorithmiques plus performantes.  
[ex : tri-fusion, fonction de hachage]
- Le surcoût devient souvent négligeable pour des données de tailles importantes et l'information n'est pas dupliquée [références partagées]

Il s'agit surtout d'un compromis **temps vs espace** qui s'avère généralement avantageux car on dispose aujourd'hui de beaucoup de mémoire.



