Types de données

Jean-Pierre Becirspahic
Lycée Marcelin Berthelot

Notion de type en informatique

Sur un ordinateur, toutes les données manipulées sont représentées par une suite de bits (une quantité qui vaut 0 ou 1) regroupées en octets (= 8 bits). Cette suite de bits est appelée la valeur de la donnée.

Notion de type en informatique

Sur un ordinateur, toutes les données manipulées sont représentées par une suite de bits (une quantité qui vaut 0 ou 1) regroupées en octets (= 8 bits). Cette suite de bits est appelée la valeur de la donnée.
Problème : connaître la valeur de la donnée n'est pas suffisant pour déterminer de quel objet il s'agit, car des objets de natures différentes peuvent posséder la même valeur.

L'entier 88 et le caractère `'X'` possèdent tous deux la valeur `01011000` dans un codage usuel sur un octet.

Notion de type en informatique

Sur un ordinateur, toutes les données manipulées sont représentées par une suite de bits (une quantité qui vaut 0 ou 1) regroupées en octets (= 8 bits). Cette suite de bits est appelée la valeur de la donnée.
Problème : connaître la valeur de la donnée n'est pas suffisant pour déterminer de quel objet il s'agit, car des objets de natures différentes peuvent posséder la même valeur.

L'entier 88 et le caractère `'X'` possèdent tous deux la valeur `01011000` dans un codage usuel sur un octet.

Solution : une donnée est représentée par sa valeur et par un type qui décrit la façon dont doit être interprétée cette suite de bits.

```
>>> type(88)
<class 'int'>

>>> type('X')
<class 'str'>
```

Notion de type en informatique

- lorsqu'on envoie une donnée par l'intermédiaire du clavier, une analyse syntaxique est réalisée pour déterminer le type de l'objet souhaité, puis sa valeur est calculée.

$$2 \xrightarrow{\text{analyse syntaxique}} \texttt{int} \xrightarrow{\text{représentation}} (\texttt{int}, \texttt{00000010})$$

$$2. \xrightarrow{\text{analyse syntaxique}} \texttt{float} \xrightarrow{\text{représentation}} (\texttt{float}, \texttt{00010000})$$

Notion de type en informatique

- lorsqu'on envoie une donnée par l'intermédiaire du clavier, une analyse syntaxique est réalisée pour déterminer le type de l'objet souhaité, puis sa valeur est calculée.

$$\texttt{2} \xrightarrow{\text{analyse syntaxique}} \texttt{int} \xrightarrow{\text{représentation}} (\texttt{int}, \texttt{00000010})$$

$$\texttt{2.} \xrightarrow{\text{analyse syntaxique}} \texttt{float} \xrightarrow{\text{représentation}} (\texttt{float}, \texttt{00010000})$$

- lorsqu'un objet doit être affiché sur l'écran, son type permet de déterminer quelle forme lui donner.

$$(\texttt{int}, \texttt{01011000}) \xrightarrow{\text{affichage à l'écran}} \texttt{88}$$

$$(\texttt{str}, \texttt{01011000}) \xrightarrow{\text{affichage à l'écran}} \texttt{'X'}$$

$$(\texttt{float}, \texttt{01011000}) \xrightarrow{\text{affichage à l'écran}} \texttt{1024.}$$

Opérations sur les types

À chaque type de donnée est associé un certain nombre de fonctions et de méthodes. Par exemple, l'algorithme d'addition du type `int` est différent de celui du type `float` :

```
>>> 1 + 2 - 3
0
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

Opérations sur les types

À chaque type de donnée est associé un certain nombre de fonctions et de méthodes. Par exemple, l'algorithme d'addition du type `int` est différent de celui du type `float` :

```
>>> 1 + 2 - 3
0
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

On peut dans une certaine mesure convertir une donnée d'un type à un autre ; la fonction correspondante porte le nom du type d'arrivée :

```
>>> float('12')
12.0
>>> str(1024)
'1024'
>>> int(3.14)
3
```

Opérations sur les types

À chaque type de donnée est associé un certain nombre de fonctions et de méthodes. Par exemple, l'algorithme d'addition du type `int` est différent de celui du type `float` :

```
>>> 1 + 2 - 3
0
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

On peut dans une certaine mesure convertir une donnée d'un type à un autre ; la fonction correspondante porte le nom du type d'arrivée :

```
>>> float('12')
12.0
>>> str(1024)
'1024'
>>> int(3.14)
3
```

```
>>> 5 / 2
2.5
>>> 2 * 12.5
25.0
>>> 1 + '3'
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

Variables

Observons le code suivant :

```
>>> x = 5
>>> x = x / 2
>>> x
2.5
```

Variables

Observons le code suivant :

```
>>> x = 5
>>> x = x / 2
>>> x
2.5
```

La variable x, initialement de type **int**, est transformée en une variable de type **float**.

Variables

Observons le code suivant :

```
>>> x = 5
>>> x = x / 2
>>> x
2.5
```

La variable x, initialement de type `int`, est transformée en une variable de type `float`.
Dans certains langages, cette transformation est impossible. Lors de sa création, une variable est associée à un espace mémoire fixe dont la taille est choisie pour contenir le type souhaité.
Un exemple en C :

```
int x = 5 ;
x = x * 2 ;
```

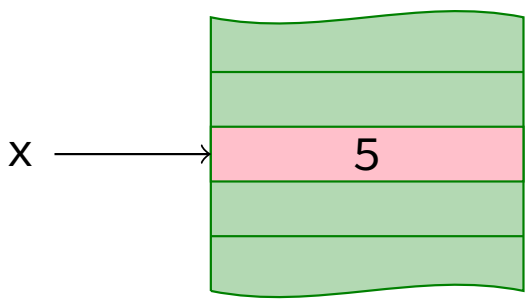
Variables

Observons le code suivant :

```
>>> x = 5
>>> x = x / 2
>>> x
2.5
```

La variable x, initialement de type **int**, est transformée en une variable de type **float**.
Dans certains langages, cette transformation est impossible. Lors de sa création, une variable est associée à un espace mémoire fixe dont la taille est choisie pour contenir le type souhaité.
Un exemple en C :

```
int x = 5 ;
x = x * 2 ;
```



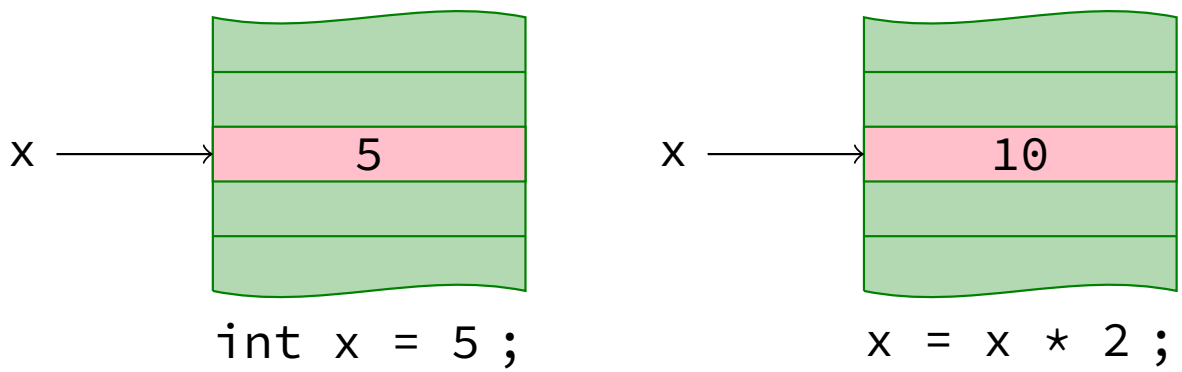
Variables

Observons le code suivant :

```
>>> x = 5
>>> x = x / 2
>>> x
2.5
```

La variable x, initialement de type **int**, est transformée en une variable de type **float**.
Dans certains langages, cette transformation est impossible. Lors de sa création, une variable est associée à un espace mémoire fixe dont la taille est choisie pour contenir le type souhaité.
Un exemple en C :

```
int x = 5 ;
x = x * 2 ;
```



Variables

Comme la taille de la représentation dépend du type de donnée, le mécanisme d'affectation à un emplacement fixe de la mémoire est impossible en Python.

Variables

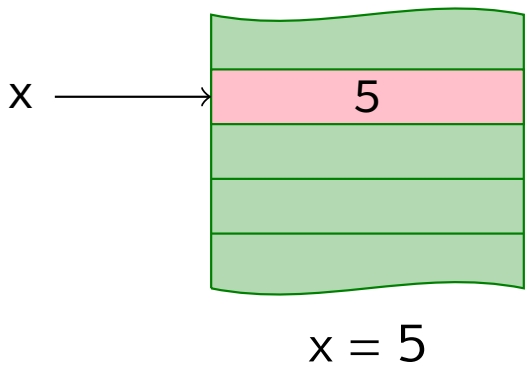
Comme la taille de la représentation dépend du type de donnée, le mécanisme d'affectation à un emplacement fixe de la mémoire est impossible en Python.

```
x = 5
x = x / 2
```

Variables

Comme la taille de la représentation dépend du type de donnée, le mécanisme d'affectation à un emplacement fixe de la mémoire est impossible en Python.

```
x = 5
x = x / 2
```

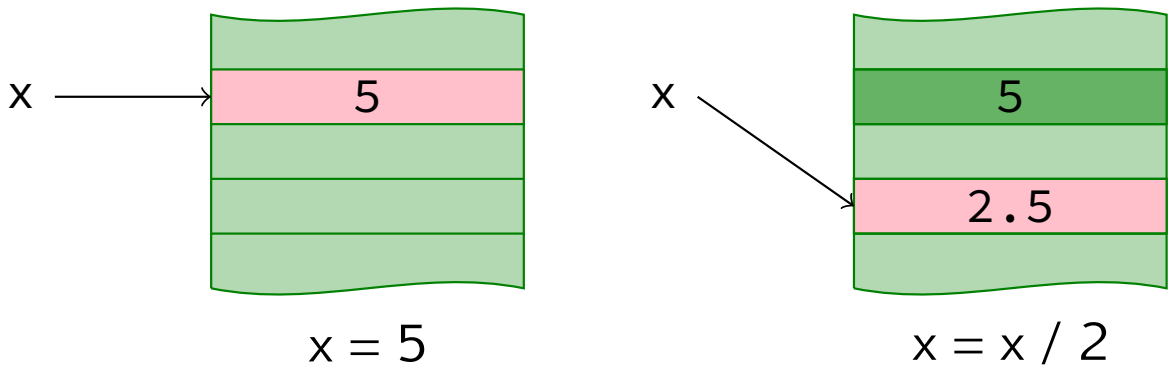


x = 5

Variables

Comme la taille de la représentation dépend du type de donnée, le mécanisme d'affectation à un emplacement fixe de la mémoire est impossible en Python.

```
x = 5
x = x / 2
```

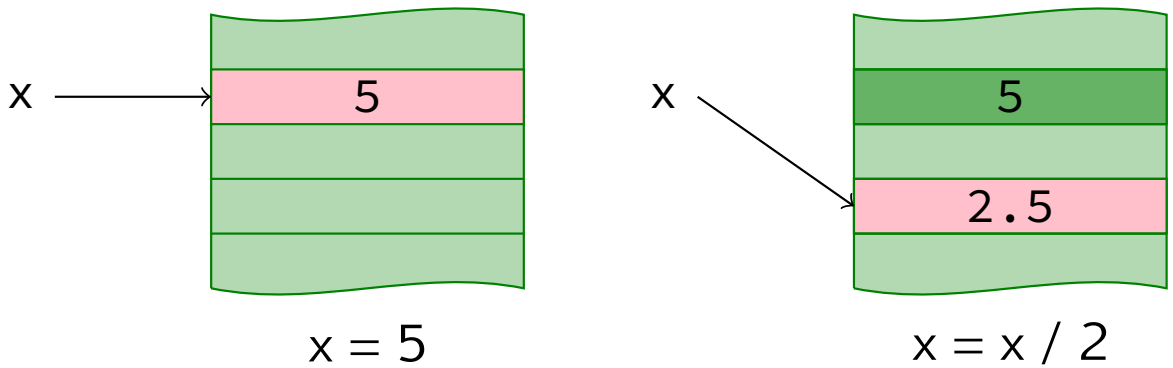


Toute modification d'une variable est stockée et référencée dans un autre emplacement mémoire, de taille adaptée.

Variables

Comme la taille de la représentation dépend du type de donnée, le mécanisme d'affectation à un emplacement fixe de la mémoire est impossible en Python.

```
x = 5
x = x / 2
```



Toute modification d'une variable est stockée et référencée dans un autre emplacement mémoire, de taille adaptée.

Remarque. Un objet Python est donc caractérisé par un type, une valeur et un identifiant (qu'on peut assimiler à une adresse mémoire).

```
>>> x = 5
>>> id(x)
4356169376
```

```
>>> x = x / 2
>>> id(x)
4360146128
```

Listes

À la différence des variables, les listes Python sont stockées à un emplacement fixe, tout en conservant la capacité d'être modifiables. À ce titre elles sont qualifiées de mutables.

```
>>> L = [1, 2, 3]
>>> id(L)
4360984768
>>> L[0] = L[0] / 2
>>> L.append('a')
>>> L
[0.5, 2, 3, 'a']      # la valeur de L a été modifiée
>>> id(L)
4360984768            # son emplacement en mémoire n'a pas changé
```

Listes

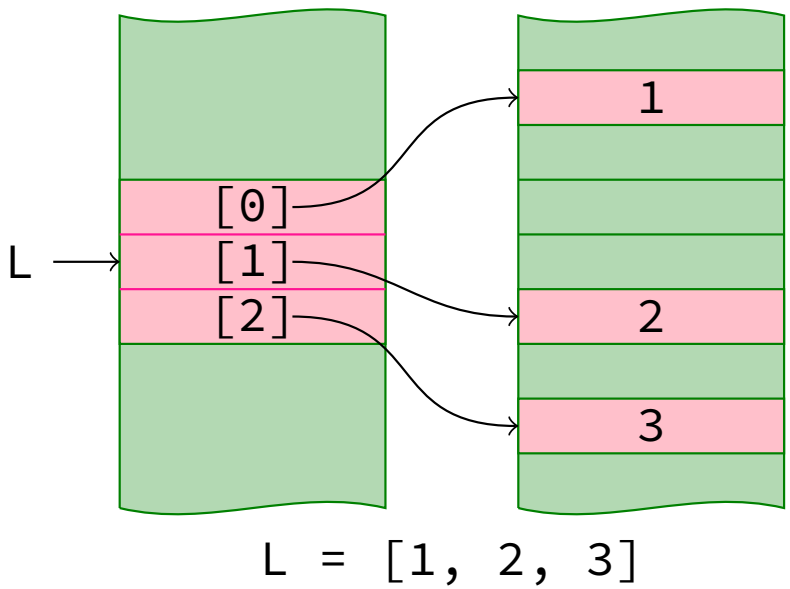
À la différence des variables, les listes Python sont stockées à un emplacement fixe, tout en conservant la capacité d'être modifiables. À ce titre elles sont qualifiées de mutables.

```
>>> L = [1, 2, 3]
>>> id(L)
4360984768
>>> L[0] = L[0] / 2
>>> L.append('a')
>>> L
[0.5, 2, 3, 'a']      # la valeur de L a été modifiée
>>> id(L)
4360984768            # son emplacement en mémoire n'a pas changé
```

Ceci est rendu possible car une liste Python se contente de stocker les identifiants des valeurs de chacune de ses cases.

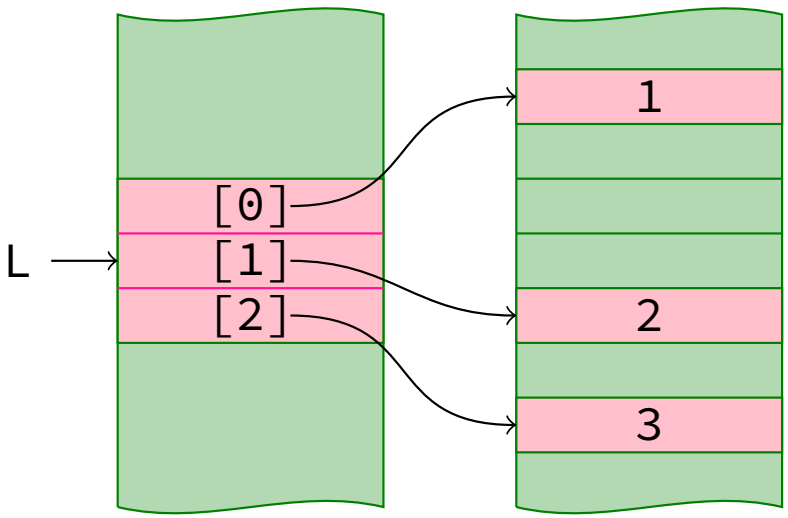
Listes

À la différence des variables, les listes Python sont stockées à un emplacement fixe, tout en conservant la capacité d'être modifiables. À ce titre elles sont qualifiées de mutables.

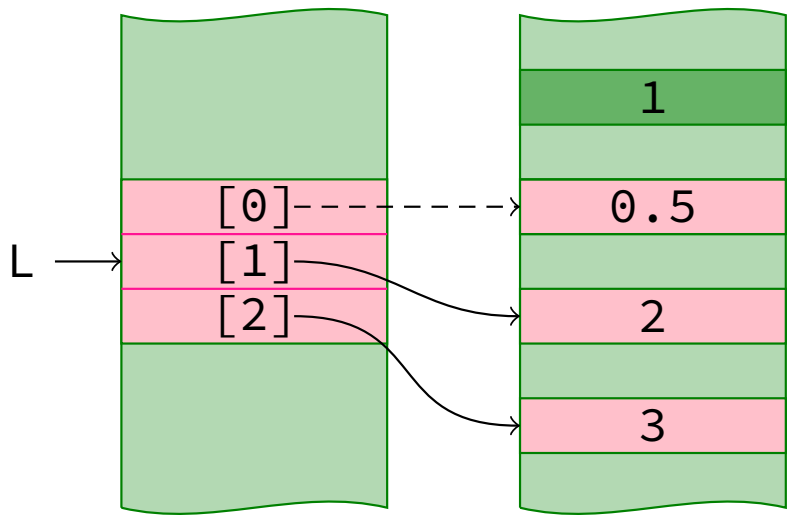


Listes

À la différence des variables, les listes Python sont stockées à un emplacement fixe, tout en conservant la capacité d'être modifiables. À ce titre elles sont qualifiées de mutables.



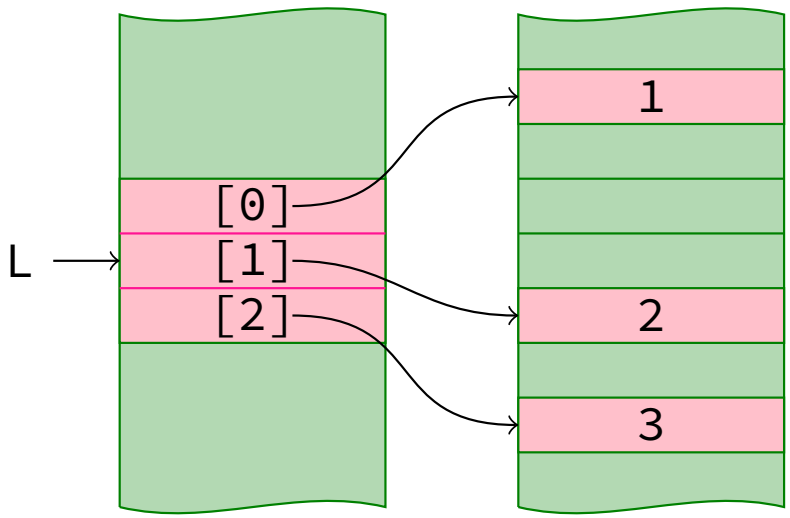
`L = [1, 2, 3]`



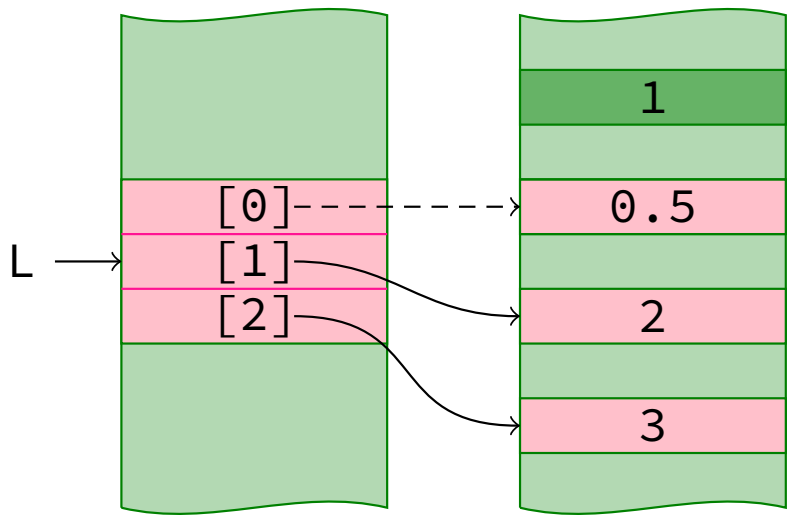
`L[0] = L[0] / 2`

Listes

À la différence des variables, les listes Python sont stockées à un emplacement fixe, tout en conservant la capacité d'être modifiables. À ce titre elles sont qualifiées de mutables.



`L = [1, 2, 3]`



`L[0] = L[0] / 2`

Interêt : modifier une case d'une liste Python se réalise à coût constant (indépendamment de la taille de la liste).

Création d'une liste

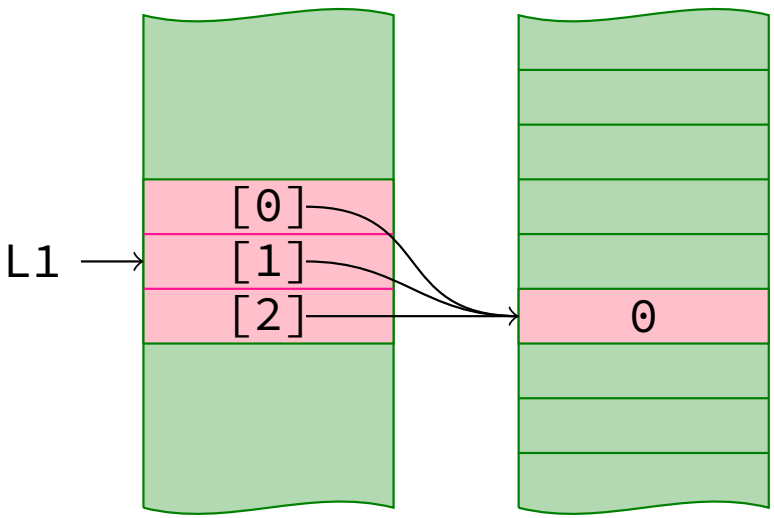
Il y a deux manières de créer une liste de n cases contenant une même valeur (ici `0`) :

- par duplication : `L1 = [0] * n`
- par compréhension : `L2 = [0 for i in range(n)]`

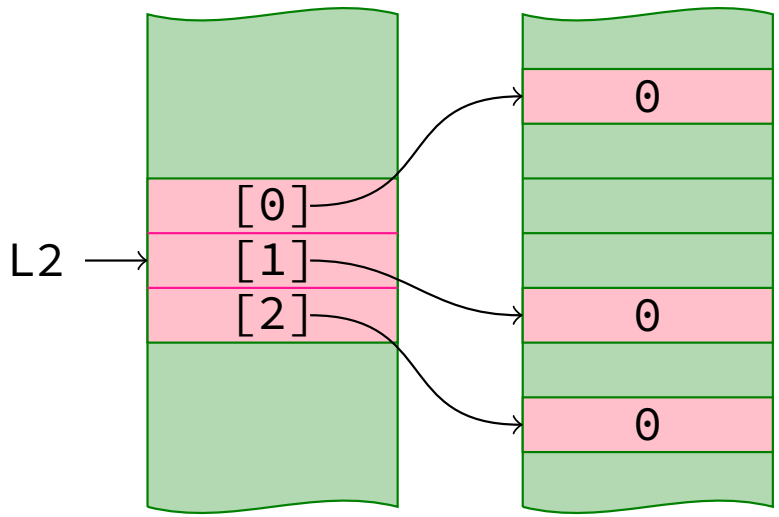
Création d'une liste

Il y a deux manières de créer une liste de n cases contenant une même valeur (ici `0`) :

- par duplication : `L1 = [0] * n`
- par compréhension : `L2 = [0 for i in range(n)]`



`L1 = [0] * 3`

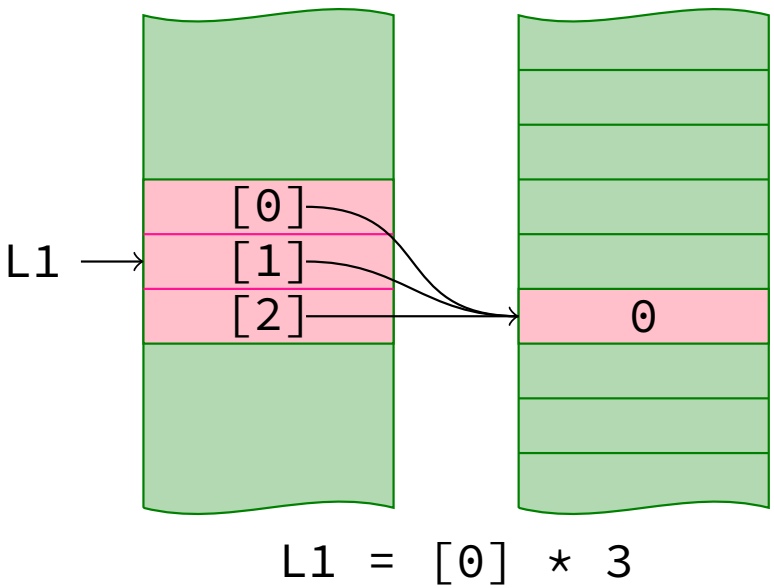


`L2 = [0 for i in range(3)]`

Création d'une liste

Il y a deux manières de créer une liste de n cases contenant une même valeur (ici `0`) :

- par duplication : `L1 = [0] * n`
- par compréhension : `L2 = [0 for i in range(n)]`



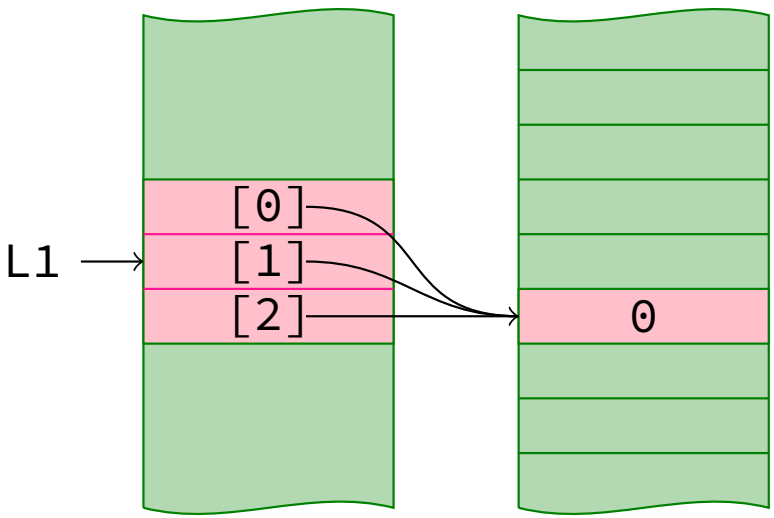

```
>>> id(L1[0])
4356169216
>>> id(L1[1])
4356169216
>>> id(L1[2])
4356169216
```

La création par duplication est-elle correcte ? Voyons ce qui se passe si on modifie une de ses cases avec l'instruction `L1[0] = L1[0] + 1`.

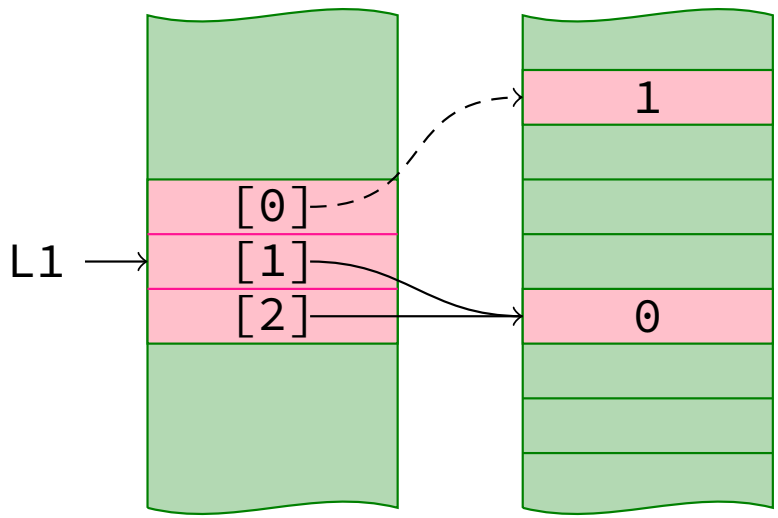
Création d'une liste

Il y a deux manières de créer une liste de n cases contenant une même valeur (ici `0`) :

- par duplication : `L1 = [0] * n`
- par compréhension : `L2 = [0 for i in range(n)]`



`L1 = [0] * 3`



`L1[0] = L1[0] + 1`

La création par duplication est-elle correcte ? Voyons ce qui se passe si on modifie une de ses cases avec l'instruction `L1[0] = L1[0] + 1`.
⟶ Ça marche car toute modification d'une variable est stockée et référencée dans un autre emplacement mémoire.

Création d'une liste bi-dimensionnelle

Créons un tableau 3 lignes et 2 colonnes par duplication :

```
>>> L = [[0] * 2] * 3
>>> L
[[0, 0], [0, 0], [0, 0]]
```

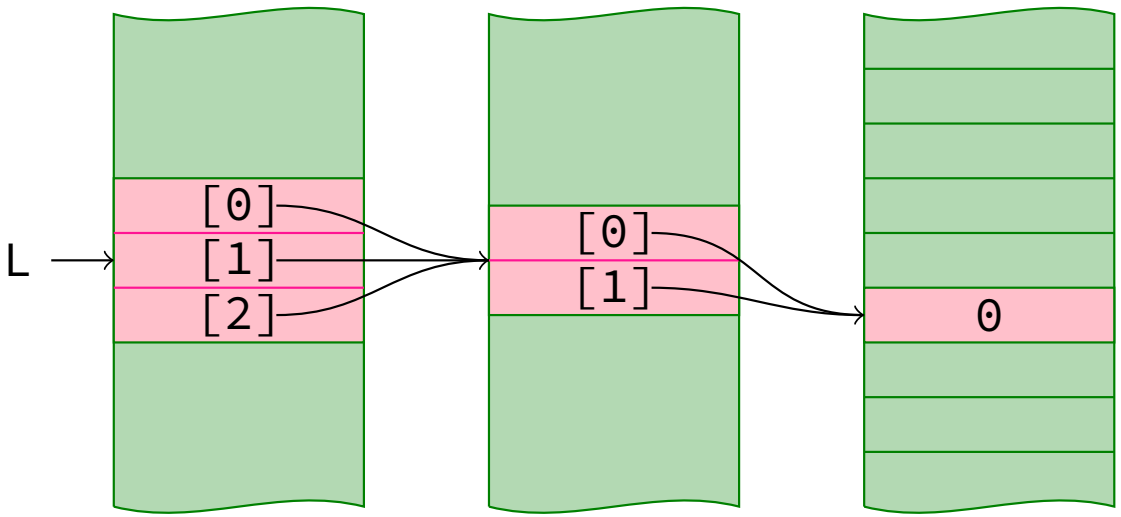
Ça semble marcher.

Création d'une liste bi-dimensionnelle

Créons un tableau 3 lignes et 2 colonnes par duplication :

```
>>> L = [[0] * 2] * 3
>>> L
[[0, 0], [0, 0], [0, 0]]
```

Ça semble marcher. Mais observons la construction en mémoire :

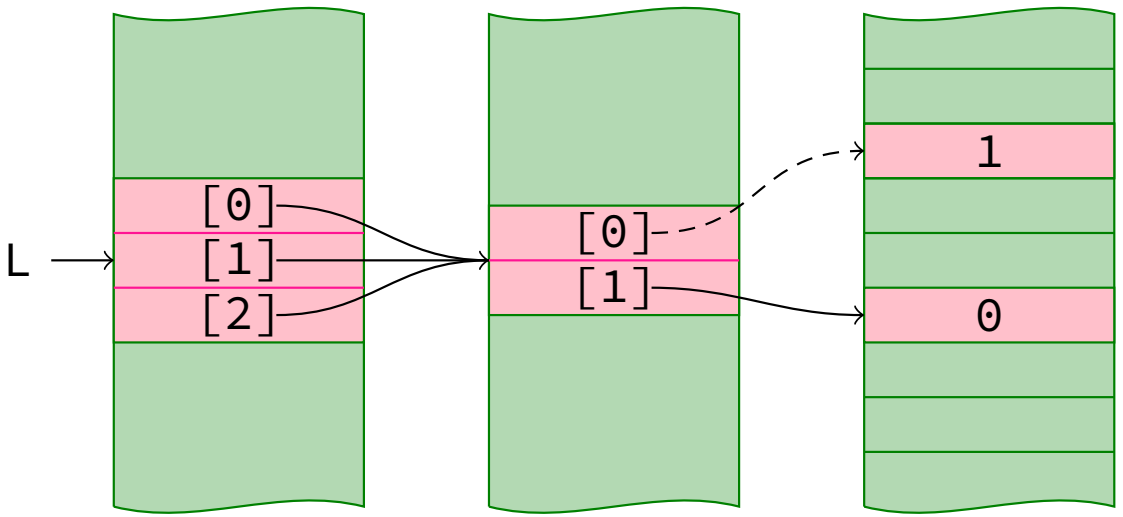


Création d'une liste bi-dimensionnelle

Créons un tableau 3 lignes et 2 colonnes par duplication :

```
>>> L = [[0] * 2] * 3
>>> L
[[0, 0], [0, 0], [0, 0]]
```

Ça semble marcher. Mais observons la construction en mémoire :




```
>>> L[0][0] = 1
>>> L
[[1, 0], [1, 0], [1, 0]]
```

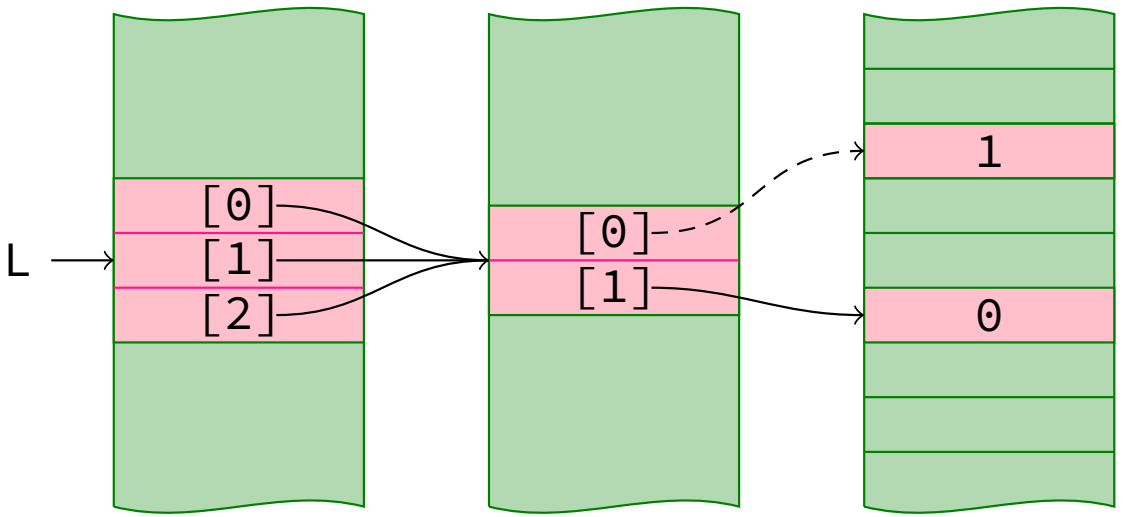
Modifier une case semble en modifier trois !

Création d'une liste bi-dimensionnelle

Créons un tableau 3 lignes et 2 colonnes par duplication :

```
>>> L = [[0] * 2] * 3
>>> L
[[0, 0], [0, 0], [0, 0]]
```

Ça semble marcher. Mais observons la construction en mémoire :




```
>>> L[0][0] = 1
>>> L
[[1, 0], [1, 0], [1, 0]]
```

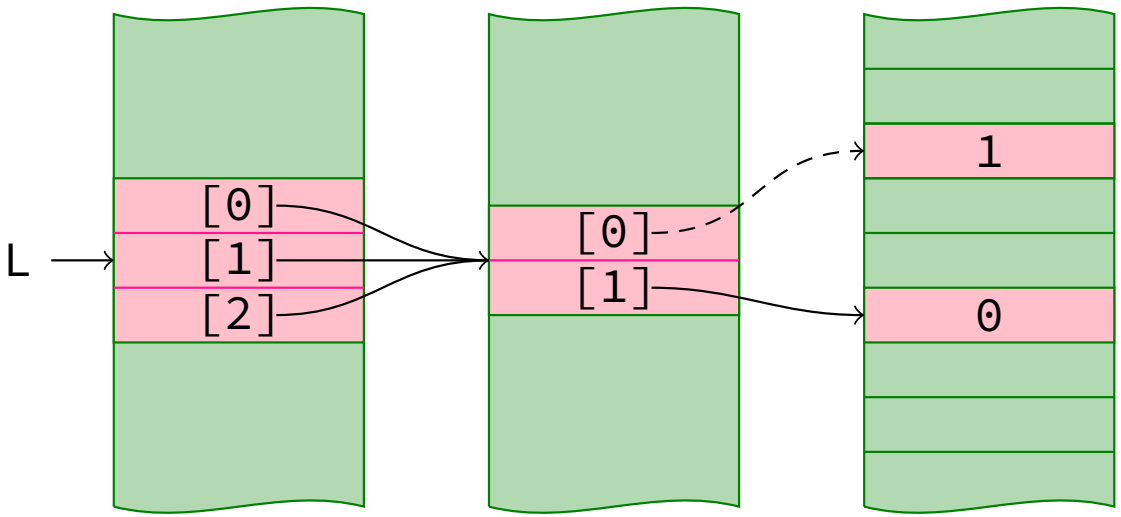
La duplication ne peut concerner que des objets immuables (= non mutables).

Création d'une liste bi-dimensionnelle

Créons un tableau 3 lignes et 2 colonnes par duplication :

```
>>> L = [[0] * 2] * 3
>>> L
[[0, 0], [0, 0], [0, 0]]
```

Ça semble marcher. Mais observons la construction en mémoire :




```
>>> L[0][0] = 1
>>> L
[[1, 0], [1, 0], [1, 0]]
```

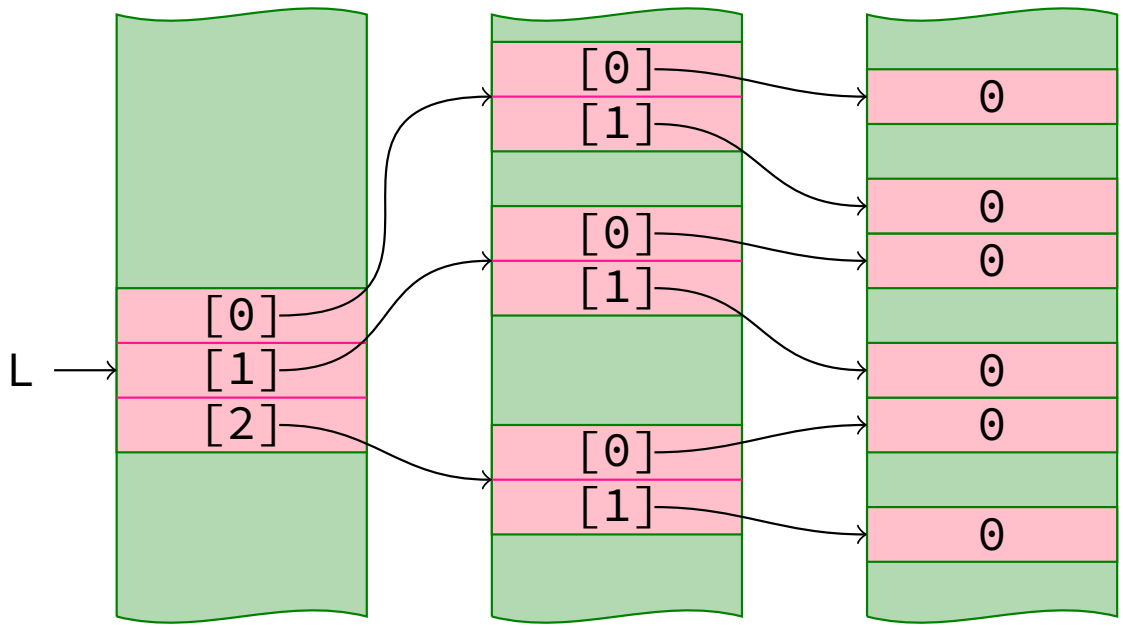
La duplication ne peut concerner que des objets immuables (= non mutables). La seule définition correcte pour créer une liste p lignes, q colonnes est par compréhension :

```
L = [[0 for j in range(q)] for i in range(p)]
```

Création d'une liste bi-dimensionnelle

La version correcte

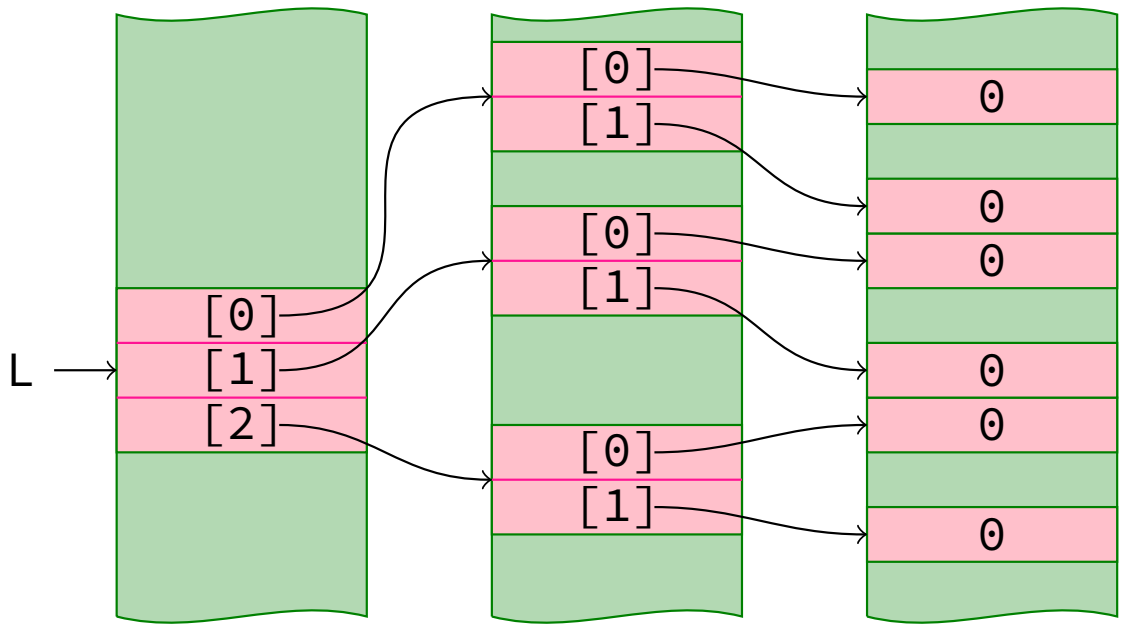
```
>>> L = [[0 for j in range (2)] for i in range(3)]
>>> L
[[0, 0], [0, 0], [0, 0]]
```



Création d'une liste bi-dimensionnelle

La version correcte

```
>>> L = [[0 for j in range (2)] for i in range(3)]
>>> L
[[0, 0], [0, 0], [0, 0]]
```

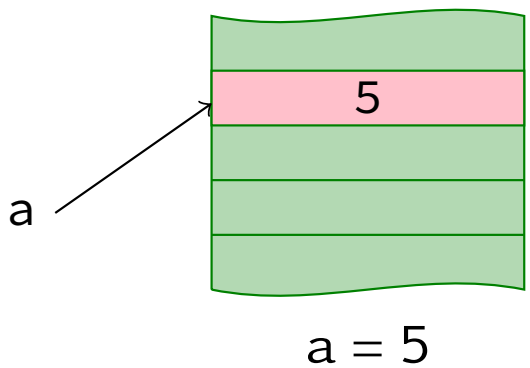



```
>>> L[0][0] = 1
>>> L
[[1, 0], [0, 0], [0, 0]]
```

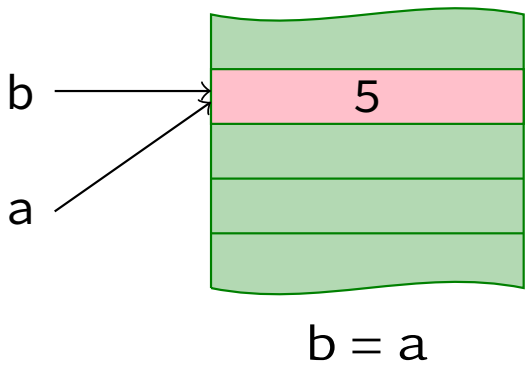
Duplication d'un objet mutable

Dupliquer un objet immuable ne pose pas de difficulté :

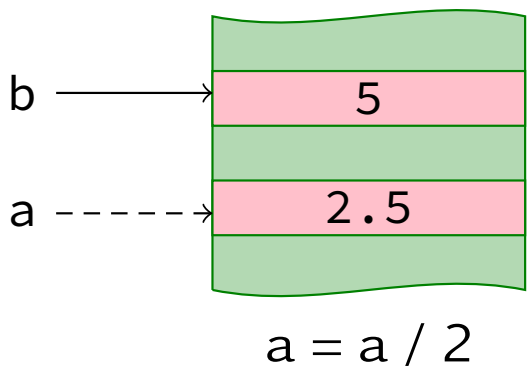
```
>>> a = 5
>>> b = a
>>> a = a / 2
```



a = 5



b = a

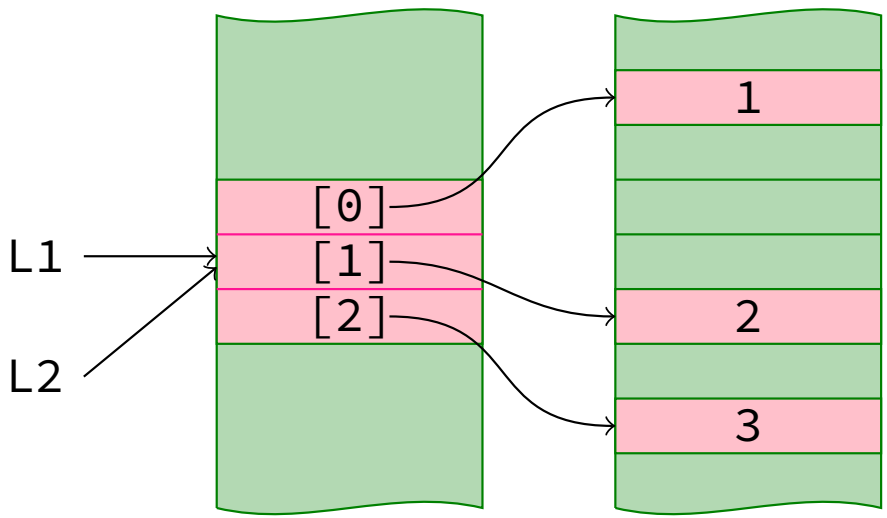


a = a / 2

Duplication d'un objet mutable

Mais l'équivalent pour un objet mutable ne donne pas le résultat souhaité :

```
>>> L1 = [1, 2, 3]
>>> L2 = L1
```

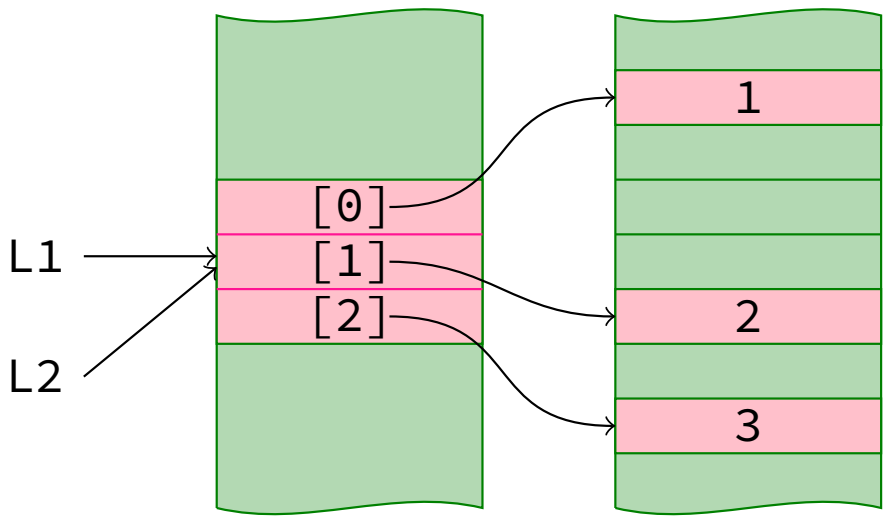


Toute modification de L1 se répercute sur L2.

Duplication d'un objet mutable

Mais l'équivalent pour un objet mutable ne donne pas le résultat souhaité :

```
>>> L1 = [1, 2, 3]
>>> L2 = L1
```



Toute modification de L1 se répercute sur L2.
Il n'y a pas de solution simple pour la duplication d'une liste, cependant

```
>>> L2 = L1.copy()
```

permet de dupliquer une liste uni-dimensionnelle.

Exercice 1

Rédiger une fonction `isSquare(n)` qui prend pour argument un entier naturel n et renvoie `True` si n est le carré d'un entier, et `False` sinon.

Exercice 1

Rédiger une fonction `isSquare(n)` qui prend pour argument un entier naturel n et renvoie `True` si n est le carré d'un entier, et `False` sinon.

Il faut trouver l'unique entier k tel que $k^2 \leqslant n < (k+1)^2$ puis déterminer si $n = k^2$.

Exercice 1

Rédiger une fonction `isSquare(n)` qui prend pour argument un entier naturel n et renvoie `True` si n est le carré d'un entier, et `False` sinon.

Il faut trouver l'unique entier k tel que $k^2 \leqslant n < (k+1)^2$ puis déterminer si $n = k^2$.
On recherche k par dichotomie.

Exercice 1

Rédiger une fonction `isSquare(n)` qui prend pour argument un entier naturel n et renvoie `True` si n est le carré d'un entier, et `False` sinon.

Il faut trouver l'unique entier k tel que $k^2 \leqslant n < (k+1)^2$ puis déterminer si $n = k^2$.
On recherche k par dichotomie.

```
def isqrt(n):
    a, b = 0, n + 1
    while a + 1 < b:
        k = (a + b) // 2
        if k * k <= n:
            a = k
        else:
            b = k
    return a

def isSquare(n):
    k = isqrt(n)
    return (k * k == n)
```

Exercice 2

Le prix d'entrée d'une manifestation est de 5€. Dans la file d'attente, chaque personne dispose d'un unique billet de 5€, 10€ ou 20€. La caisse du guichetier est initialement vide, et l'on souhaite savoir s'il sera possible de rendre la monnaie à tout le monde.
Rédiger une fonction `fondDeCaisse(L)` qui prend pour argument la liste des billets que possèdent les gens dans la file et renvoie un booléen indiquant s'il est possible de rendre la monnaie à tous les spectateurs.
Par exemple,

- pour `L=[5, 10, 5, 20]` cette fonction renverra `True` (le second client recevra un billet de 5€ et le quatrième, un billet de 10€ et un de 5€);
- pour `L=[5, 10, 10, 5]` la réponse sera `False` (il n'est pas possible de rendre la monnaie au troisième client).

Exercice 2

On simule le fond de caisse à l'aide de deux variables dénombrant le nombre de billets de 5€ et de 10€ dans la caisse.

Exercice 2

On simule le fond de caisse à l'aide de deux variables dénombrant le nombre de billets de 5€ et de 10€ dans la caisse.

- On peut rendre la monnaie sur 10€ lorsqu'on possède au moins un billet de 5€ ;

Exercice 2

On simule le fond de caisse à l'aide de deux variables dénombrant le nombre de billets de 5€ et de 10€ dans la caisse.

- On peut rendre la monnaie sur 10€ lorsqu'on possède au moins un billet de 5€ ;
- On peut rendre la monnaie sur 20€ lorsque :
  - on possède au moins un billet de 10€ et un de 5€
  - ou si on possède trois billets de 5€ au moins ;

Exercice 2

On simule le fond de caisse à l'aide de deux variables dénombrant le nombre de billets de 5€ et de 10€ dans la caisse.

- On peut rendre la monnaie sur 10€ lorsqu'on possède au moins un billet de 5€ ;
- On peut rendre la monnaie sur 20€ lorsque :
  - on possède au moins un billet de 10€ et un de 5€
  - ou si on possède trois billets de 5€ au moins ;
- si les deux solutions sont possibles, choisir la première.

Exercice 2

```
def fondDeCaisse(L):
    cinq, dix = 0, 0
    for x in L:
        if x == 5:
            cinq += 1
        elif x == 10 and cinq > 0:
            cinq -= 1
            dix += 1
        elif x == 20 and dix > 0 and cinq > 0:
            dix -= 1
            cinq -= 1
        elif x == 20 and cinq > 2:
            cinq -= 3
        else:
            return False
    return True
```