Gestion des listes en Python

Algorithmique

NSI1

8 octobre 2021

Opérations de base

Créer une liste

Créer une liste

Pour créer une variable de type `list` on peut utiliser les commandes suivantes :

Créer une liste

Pour créer une variable de type `list` on peut utiliser les commandes suivantes :

- `L = list()` crée une liste vide ;

Créer une liste

Pour créer une variable de type `list` on peut utiliser les commandes suivantes :

- `L = list()` crée une liste vide ;
- `L = []` fait la même chose ;

Créer une liste

Pour créer une variable de type `list` on peut utiliser les commandes suivantes :

- `L = list()` crée une liste vide ;
- `L = []` fait la même chose ;
- `L = ['a', 7, True]` crée la liste composée de 3 éléments.

Créer une liste

Pour créer une variable de type `list` on peut utiliser les commandes suivantes :

- `L = list()` crée une liste vide ;
- `L = []` fait la même chose ;
- `L = ['a', 7, True]` crée la liste composée de 3 éléments.

Une liste peut contenir des éléments de plusieurs types mais en pratique on évite cela.

Modifier un élément

Modifier un élément

Le type `list` est mutable, cela veut dire qu'on peut changer les éléments d'une liste sans changer la liste en elle-même (on précisera pourquoi et comment plus tard dans ce document).

Modifier un élément

Le type `list` est mutable, cela veut dire qu'on peut changer les éléments d'une liste sans changer la liste en elle-même (on précisera pourquoi et comment plus tard dans ce document).

Pour changer le deuxième élément d'une liste `L` qui vaut `[2, 3, 4, 1]` on écrira

Modifier un élément

Le type `list` est mutable, cela veut dire qu'on peut changer les éléments d'une liste sans changer la liste en elle-même (on précisera pourquoi et comment plus tard dans ce document).

Pour changer le deuxième élément d'une liste `L` qui vaut `[2, 3, 4, 1]` on écrira

```
L[1] = 10
```

Modifier un élément

Le type `list` est mutable, cela veut dire qu'on peut changer les éléments d'une liste sans changer la liste en elle-même (on précisera pourquoi et comment plus tard dans ce document).

Pour changer le deuxième élément d'une liste `L` qui vaut `[2, 3, 4, 1]` on écrira

```
L[1] = 10
```

et la liste aura la valeur `[2, 10, 4, 1]`.

Ajouter un élément à une liste

Ajouter un élément à une liste

`L.append(4)` ajoute la valeur 4 à la fin de la liste `L`.

Ajouter un élément à une liste

`L.append(4)` ajoute la valeur 4 à la fin de la liste `L`.

`L = L + [4]` a le même effet : on crée une « mini-liste » `[4]`, on concatène les 2 listes et on remet le résultat dans `L`.

Ajouter un élément à une liste

`L.append(4)` ajoute la valeur 4 à la fin de la liste `L`.

`L = L + [4]` a le même effet : on crée une « mini-liste » `[4]`, on concatène les 2 listes et on remet le résultat dans `L`.

En pratique la première méthode est la plus simple et aussi la plus rapide.

Retirer un élément à une position donnée

Retirer un élément à une position donnée

Si une liste `L` a pour valeur `[3 ,7, 1]` et qu'on veut supprimer son deuxième élément alors on écrit

Retirer un élément à une position donnée

Si une liste `L` a pour valeur `[3 ,7, 1]` et qu'on veut supprimer son deuxième élément alors on écrit

```
del L[1]
```

Retirer un élément à une position donnée

Si une liste `L` a pour valeur `[3 ,7, 1]` et qu'on veut supprimer son deuxième élément alors on écrit

```
del L[1]
```

Ensuite, `L` aura la valeur `[3,1]`.

Retirer une valeur précise

Retirer une valeur précise

Pour retirer une valeur qui appartient à une liste on procède ainsi :

Retirer une valeur précise

Pour retirer une valeur qui appartient à une liste on procède ainsi :

Si `L` a la valeur `[1, 2, 5, 4, 2, 3]` alors l'instruction

Retirer une valeur précise

Pour retirer une valeur qui appartient à une liste on procède ainsi :

Si `L` a la valeur `[1, 2, 5, 4, 2, 3]` alors l'instruction

```
L.remove(2)
```

Retirer une valeur précise

Pour retirer une valeur qui appartient à une liste on procède ainsi :

Si `L` a la valeur `[1, 2, 5, 4, 2, 3]` alors l'instruction

```
L.remove(2)
```

Supprime la première occurrence de `2` dans `L`.

Retirer une valeur précise

Pour retirer une valeur qui appartient à une liste on procède ainsi :

Si `L` a la valeur `[1, 2, 5, 4, 2, 3]` alors l'instruction

```
L.remove(2)
```

Supprime la première occurrence de `2` dans `L`.
Ainsi `L` a la valeur `[1, 5, 4, 2, 3]`.

Ajouter des listes

Ajouter des listes

On peut procéder de 2 manières :

Ajouter des listes

On peut procéder de 2 manières :

- `L.extend(M)` ajoute les éléments de la liste `M` à la fin de `L` ;

Ajouter des listes

On peut procéder de 2 manières :

- `L.extend(M)` ajoute les éléments de la liste `M` à la fin de `L` ;
- `L = L + M` crée une liste avec les éléments de `L` puis ceux de `L` et replace le résultat dans `L`.

Ajouter des listes

On peut procéder de 2 manières :

- `L.extend(M)` ajoute les éléments de la liste `M` à la fin de `L` ;
- `L = L + M` crée une liste avec les éléments de `L` puis ceux de `L` et replace le résultat dans `L` .

En pratique la première méthode est plus rapide.

Divers

Divers

`L.sort()` trie la liste dans l'ordre croissant.

Divers

`L.sort()` trie la liste dans l'ordre croissant.

`L.reverse()` met les éléments dans l'ordre inverse.

Divers

`L.sort()` trie la liste dans l'ordre croissant.

`L.reverse()` met les éléments dans l'ordre inverse.

Manipulations

Copier une liste (mauvaise méthode)

Copier une liste (mauvaise méthode)

```
L = [3, 5, 2]
M = L
M[0] = 2
print(M) # Affiche [2, 5, 2]
print(L) # Affiche [2, 5, 2] aussi. Aïe !
```

Copier une liste (mauvaise méthode)

```
L = [3, 5, 2]
M = L
M[0] = 2
print(M) # Affiche [2, 5, 2]
print(L) # Affiche [2, 5, 2] aussi. Aïe !
```

Ce comportement « étrange » vient du fait que le type `list` est mutable. Nous allons expliquer cela plus tard dans ce document.

Copier une liste (bonne méthode)

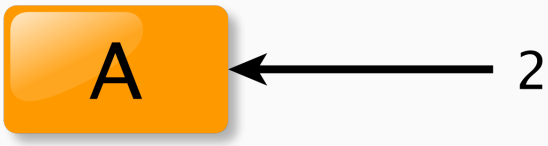
Copier une liste (bonne méthode)

```
L = [3, 5, 2]
M = L[:] # on copie tous les éléments de L dans M
M[0] = 2
print(M) # Affiche [2, 5, 2]
print(L) # Affiche [3, 5, 2] aussi. Ouf !
```

Mutabilité

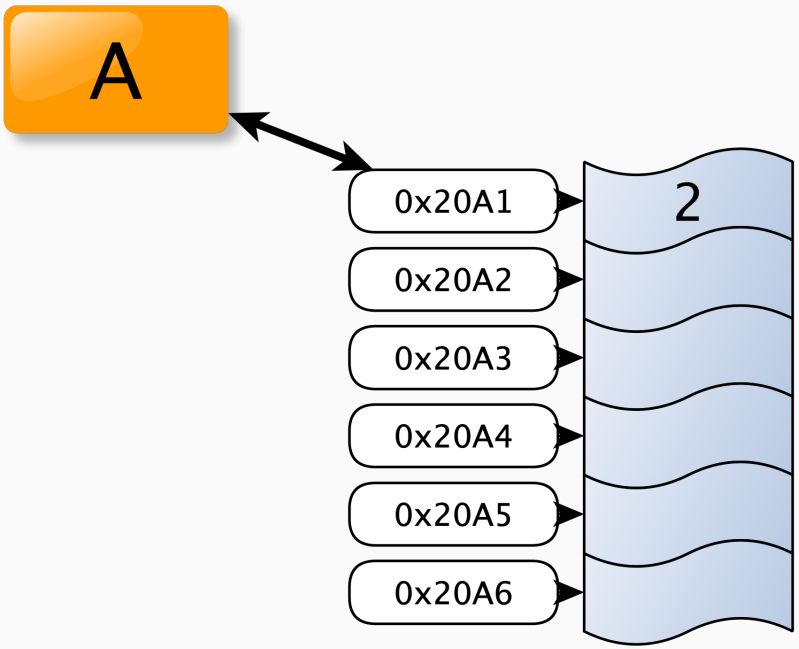
variables de type non-mutable

variables de type non-mutable



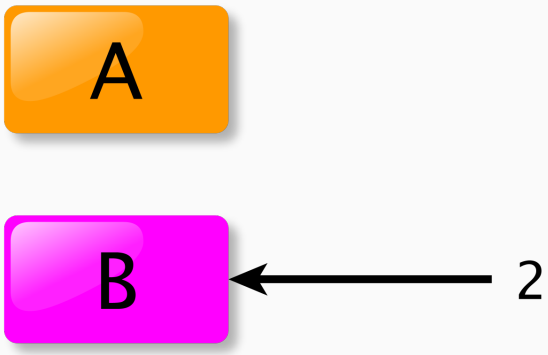
On affecte 2 à A.

variables de type non-mutable



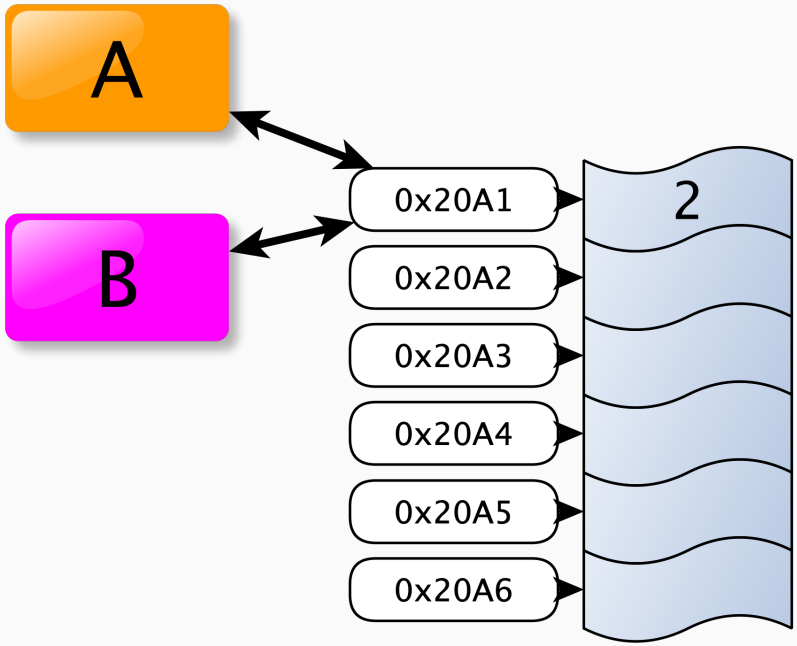
Une adresse mémoire est réservée.

variables de type non-mutable



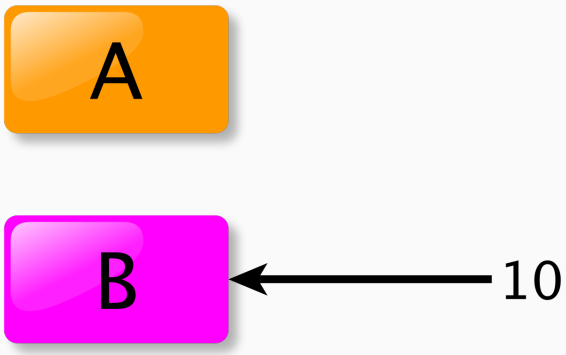
On affecte 2 à B.

variables de type non-mutable



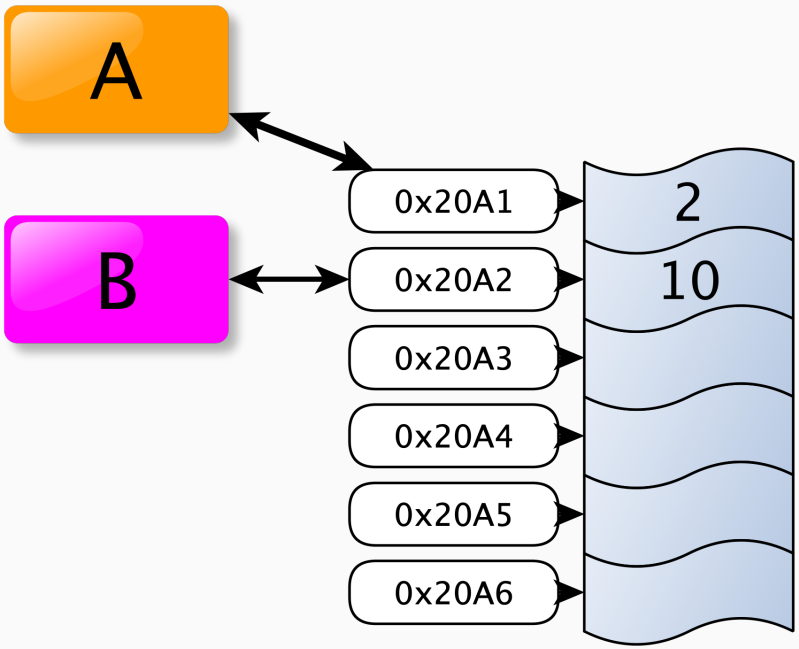
A et B pointent sur la même adresse.

variables de type non-mutable



On change la valeur de B.

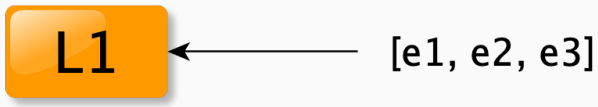
variables de type non-mutable



B pointe sur une autre adresse.

variables de type mutable

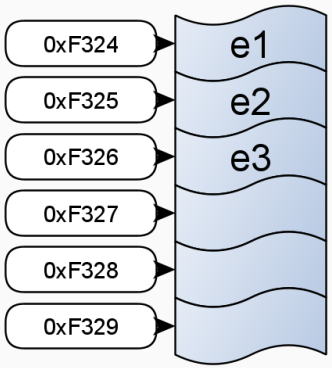
variables de type mutable



On affecte [e1, e2, e3] à L1.

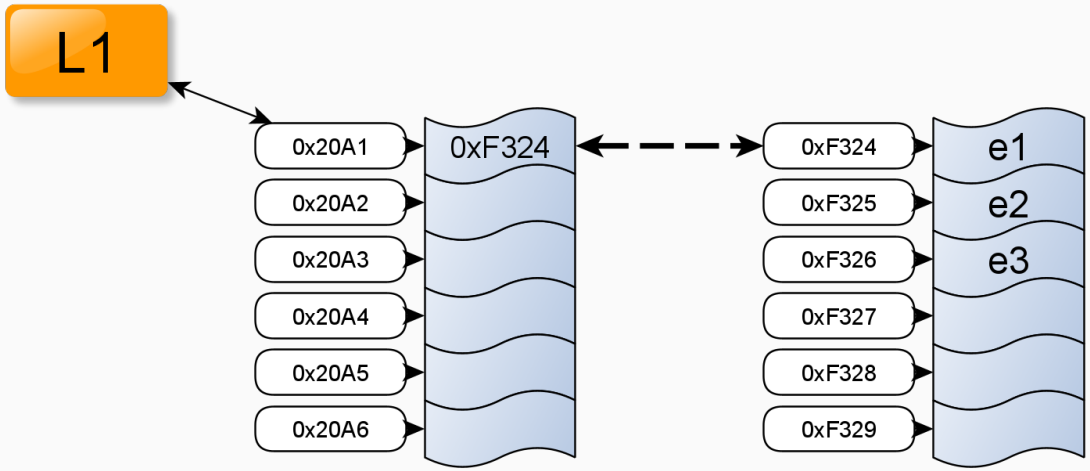
variables de type mutable

L1



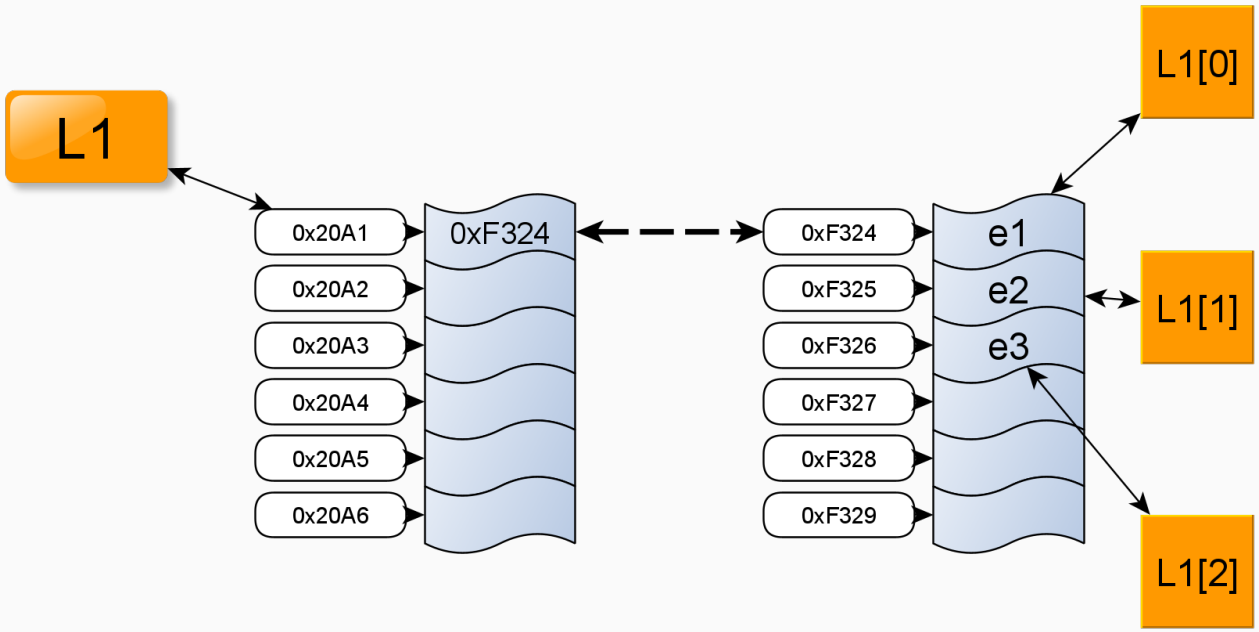
Les éléments sont mis en mémoire.

variables de type mutable

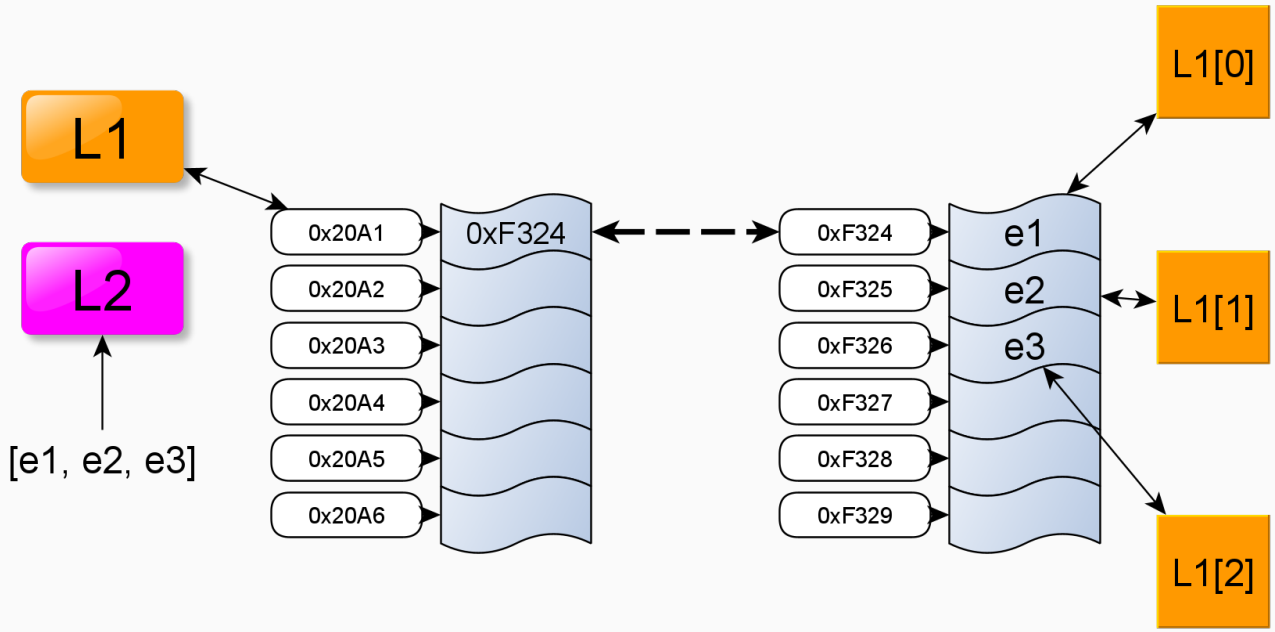


L1 contient l'adresse du début de la plage mémoire.

variables de type mutable

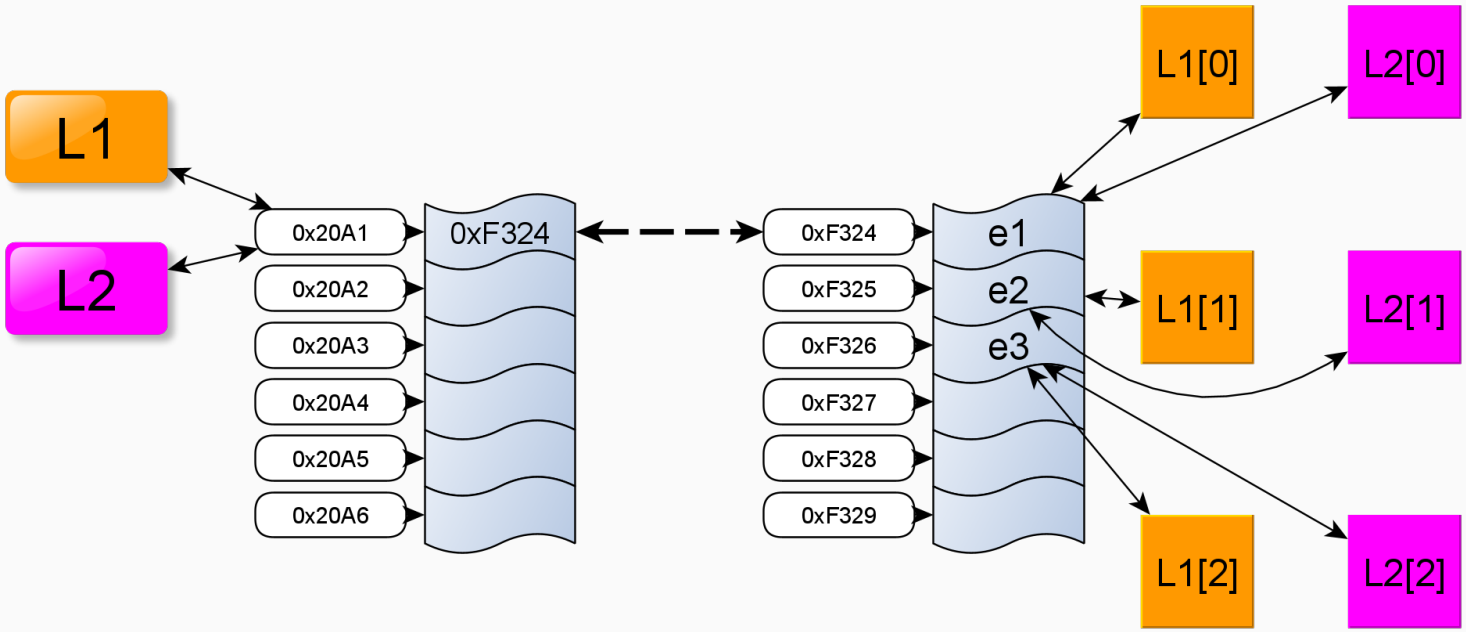


variables de type mutable



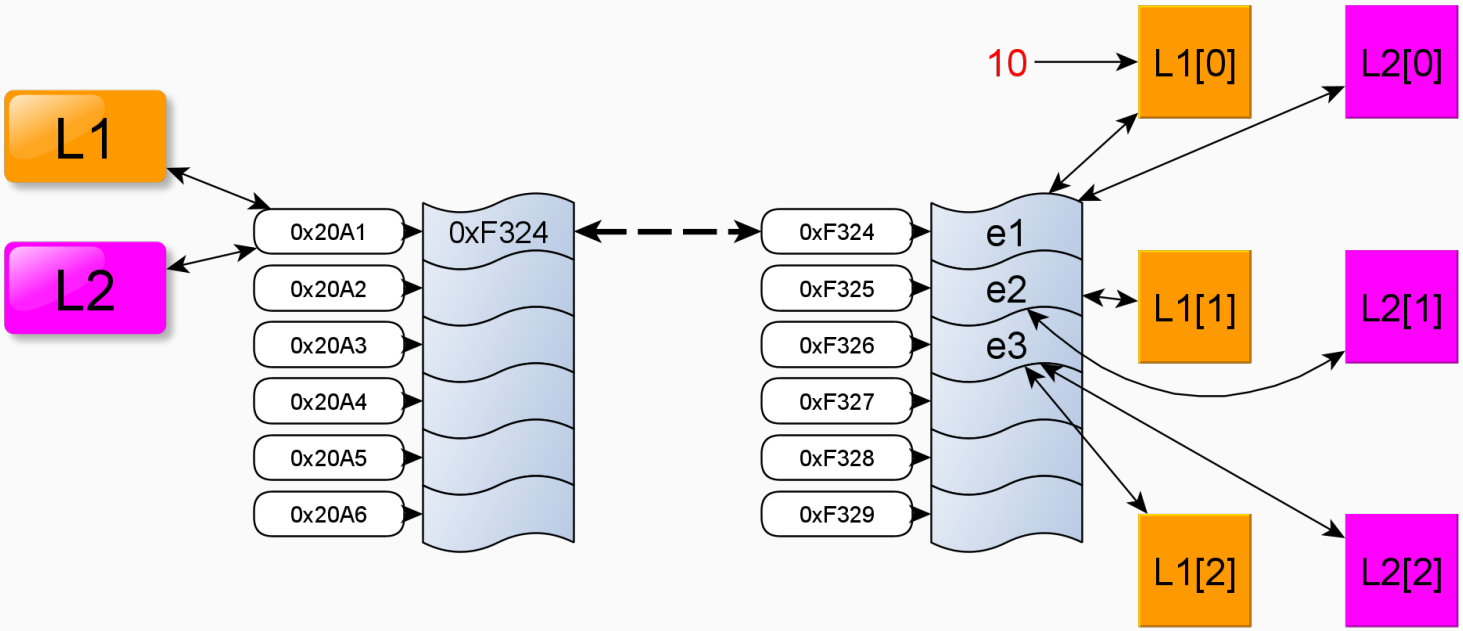
On affecte [e1, e2, e3] à L2.

variables de type mutable



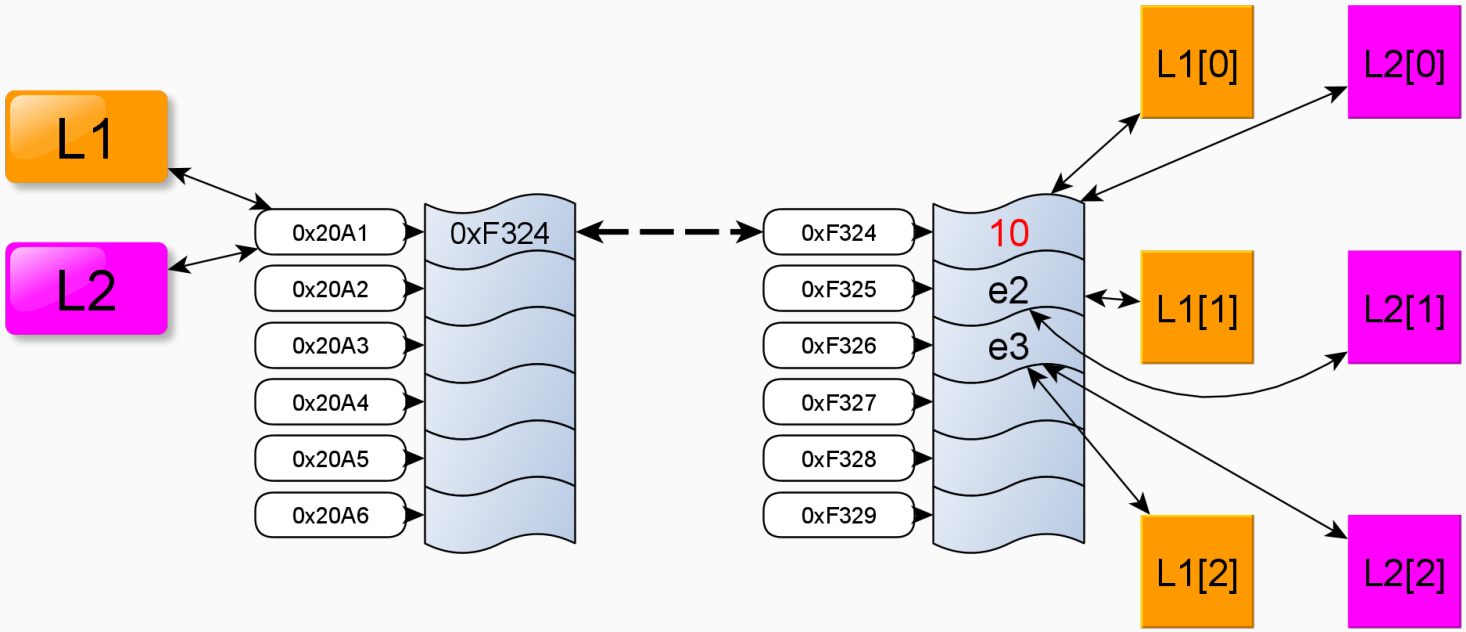
L2 contient la même adresse que A.

variables de type mutable



Si on affecte 10 à L1[0]

variables de type mutable



Alors L2[0] vaut 10 également.

Parcourir une liste

Parcours selon les indices

Parcours selon les indices

```
L = [54, 65, 123]
n = len(L)
for i in range(n):
    print(L[i])
```

Parcours selon les valeurs

Parcours selon les valeurs

```
L = [54, 65, 123]
for x in L:
    print(x)
```