Chap. 2 Types structurés Thème 2

Valeurs et références

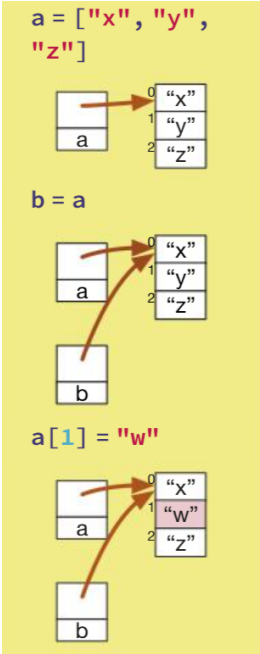
Sommaire du chapitre 2

I Alias

Lorsque l'on affecte une valeur d'un type construit à une variable, on affecte en réalité une **référence** à cette valeur. Lorsque l'on affecte cette variable à une autre variable, les deux référencent alors la *même* valeur. On dit que les deux variables sont un **alias** pour la même valeur. Les alias peuvent être source de confusion et d'erreurs lorsqu'ils référencent des valeurs muables, car la modification du contenu de cette valeur par une variable affecte la valeur de l'autre variable.

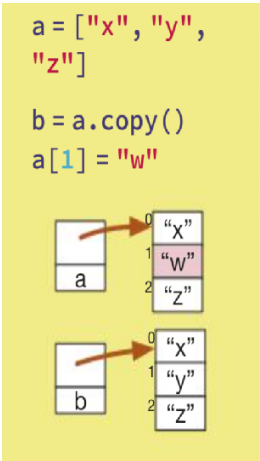
Dans l'exemple suivant, après la ligne 2, les variables `a` et `b` référencent le même tableau. Lorsque l'on change la valeur de l'élément `a[1]` (ligne 3), on constate que la valeur référencée par `b` a également changé (ligne 4).

```
1 a = ["x", "y", "z"]
2 b = a
3 a[1] = "w"
4 print(b)          # ['x', 'w', 'z']
```



Si l'on veut que `b` désigne une *copie* de `a`, on peut utiliser la méthode `copy()` des tableaux (qui est aussi disponible pour les dictionnaires) :

```
a = ["x", "y", "z"]
b = a.copy()      # copier le tableau a
a[1] = "w"
print(b)          # ['x', 'y', 'z']
```



Il faut prendre particulièrement garde aux alias lorsque l'on passe une valeur d'un type construit en paramètre à une fonction, car le paramètre, qui est une variable locale de la fonction, devient un alias pour la variable passé en argument.
Dans l'exemple suivant, le tableau `impairs` est, probablement involontairement, modifié par la fonction `tdouble` :

```
# passage d'un tableau en paramètre
def tdouble(t) :
    """ retourne un tableau dont les éléments sont le double de t """
    for i in range(len(t)) :
        t[i] = t[i] * 2
```

```
6       return t
7
8   impairs = [1, 3, 5, 7, 9]
9   doubles = tdouble(impairs)
10  print(doubles)     # [2, 6, 10, 14, 18]
11  print(impairs)     # [2, 6, 10, 14, 18]
12  # doubles et impairs sont le MEME tableau
```

Pour éviter ces problèmes, il faut clairement documenter la fonction pour indiquer si elle modifie ou non ses paramètres. Dans cet exemple, la documentation de la fonction `tdouble` devrait plutôt être libellé *« remplace les éléments de t par le double de leur valeur »*. Mais si l'intention est bien de retourner un nouveau tableau, il faut par exemple, effectuer une copie du paramètre avant la boucle en ajoutant la ligne suivante avant la ligne 4 :

```
t = t.copy()    # copier le tableau pour ne pas modifier l'original
```

II Egalité de valeurs ou de références : l'opérateur is

L'opérateur `==` teste l'égalité des *valeurs*, c'et-à-dire pour un type construit l'égalité des contenus. Python permet également de tester l'égalité des *références* avec l'opérateur `is`, et ainsi de savoir si deux références sont des alias de la même valeur :

```
1   a = [1, 2]
2   print(a == [1, 2], a is [1,2])   # True, False
3   b = a
4   print(a == b, a is b)    # True, True
5   b = a.copy()
6   print(a == b, a is b)    # True, False
```