Le problème de la copie des variables

Nous avons vu précédemment que les variables se copiaient de manière différente suivant leur type. Nous allons dans ce notebook aller explorer dans les entrailles de la machine l'implémentation des différents types de variable, pour expliquer ces différences.
Nous utiliserons l'instruction qui donne l' "identité" d'une variable : `id(variable)`. C'est en fait l'adresse mémoire de la variable. Cette adresse est renvoyée en décimal (base 10). Compléter le code ci-dessous pour obtenir l'adresse en hexadécimal, ce qui est plus standard.

```
[ ]  a = 3
     print(id(a))
```

La cellule suivante contient une fonction `adresse(variable)` qui renvoie l'adresse en hexadécimal, ainsi que le type, d'une variable. Cette fonction nous servira pour toute la suite du notebook. La variable est de type quelconque. Exécuter la cellule pour disposer de cette fonction dans la suite du notebook.

```
[ ]  def adresse(variable):
         """
         Renvoie l'adresse en hexadécimal et le type d'une variable
     quelconque
         """
         adr = hex(id(variable))
         typ = type(variable)
         return adr,typ
```

Copie et modification des entiers, flottants, chaînes de caractères.

Exécuter la cellule de code suivante. Changer la variable en un flottant, puis une chaîne de caractères.

```
[ ]  var_1 = 3
     var_2 = var_1    #TEST de la copie de variable
     (adr_1,typ_1) = adresse(var_1)
     (adr_2,typ_2) = adresse(var_2)
```

```
print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
```

Modifions la valeur de `var_2`

```
var_1 = 3
var_2 = var_1    #TEST de la copie de variable
(adr_1,typ_1) = adresse(var_1)
(adr_2,typ_2) = adresse(var_2)
print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
print("modification de la valeur de var_2")
var_2 = var_2 + 1
(adr_2,typ_2) = adresse(var_2)
print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
```

Compléter et **retenir** le paragraphe de cours suivant:
Lors d'une modification de la valeur d'un entier, d'un flottant ou d'une chaîne,...
Ces variables sont dites **immuables** (immutable in english) : on ne peut pas les "modifier", dans le sens où l'on ne peut pas modifier le contenu de leur adresse mémoire. A chaque modification dans l'exécution d'un programme, Python crée une nouvelle variable de même nom, à un autre emplacement mémoire. L'avantage est la sécurité (pas de modification involontaire de la valeur de la variable), l'inconvénient une certain inefficacité (perte de temps et d'espace mémoire).

Copie et modification des listes (dictionnaires, tuples, ...)

Reprenons le code précédent, avec une liste

```
var_1 = [1,2,3]
var_2 = var_1    #TEST de la copie de variable
(adr_1,typ_1) = adresse(var_1)
(adr_2,typ_2) = adresse(var_2)
print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
print("modification de la valeur de var_2")
var_2[2]= 5
print("var_1 ",var_1)
print("var_2 ",var_2)
(adr_2,typ_2) = adresse(var_2)
print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
```

Compléter et **retenir** le paragraphe de cours suivant:
Lors d'une modification de la valeur d'une liste...
Ces variables sont dites **muables** (mutable in english) : on change le contenu de leur adresse mémoire à chaque modification dans l'exécution d'un programme. Par rapport à une variable immuable, on gagne en efficaicté et on perd en sécurité.

Copie de listes bis

Observons le résultat de l'instruction `liste_2 = liste_1[:]`

```
[ ]  var_1 = [1,2,3]
     var_2 = var_1[:]    #TEST de la copie de variable
     (adr_1,typ_1) = adresse(var_1)
     (adr_2,typ_2) = adresse(var_2)
     print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
     print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
     print("modification de la valeur de var_2")
     var_2[2]= 5
     print("var_1 ",var_1)
     print("var_2 ",var_2)
```

Conclure:

Une deuxième méthode de copie

La méthode `var_2 = var_1.copy()` permet également de faire une copie "propre" de la liste.

```
[ ]  var_1 = [1,2,3]
     var_2 = var_1.copy()    #TEST de la copie de variable
     (adr_1,typ_1) = adresse(var_1)
     (adr_2,typ_2) = adresse(var_2)
     print("la variable var_1 est un ",typ_1," d'adresse ",adr_1)
     print("la variable var_2 est un ",typ_2," d'adresse ",adr_2)
     print("modification de la valeur de var_2")
     var_2[2]= 5
     print("var_1 ",var_1)
     print("var_2 ",var_2)
```

Passage des paramètres dans une fonction

La fonction ci-dessous est construite pour fonctionner avec des entiers, des flottants, des chaînes ou des listes.
Testez-là avec tous ces types de variables, et écrivez votre conclusion dans l'encadré ci-après. Ceci sera à retenir, en effet lors de l'écriture d'un programme (et donc du ou des projets) la manière dont sont passés les paramètres a une grande importance.

[ ]
```
from copy import copy
def double_ou_rien(variable,adres):
    """
    Est censé renvoyer la variable non modifiée
    A l'intérieur de la fonction:
        Un nombre est multiplié par deux
        Une chaine ou une liste est concaténée à elle-même
    """

    if type(variable) is list:
        variable.append("truc")
    else:
        variable = variable*2
    (adr_var,typ_var) = adresse(variable)
    if adr_var != adres:
        print("La variable a été recréée à l'intérieur de la
fonction :")
        print("\tSon type est ",typ_var)
        print("\tL'adresse locale (dans la fonction) est
",adr_var," alors que l'adresse dans le programme principal est
",adres)
    else:
        print("La variable n'a pas été recréée à l'intérieur de
la fonction")
        print("\tLa variable a pour adresse globale ",adres,\
              " aussi bien dans la fonction que dans le programme
principal")
    return

var = 1    #à tester avec entier, flottant, chaîne, liste
sauve_var = copy(var)
(adr_v,typ_v) = adresse(var)
print("la variable var est un ",typ_v," d'adresse ",adr_v)
double_ou_rien(var,adr_v)

print("\nRetour de fonction ")
if var == sauve_var :
    print("Pour une variable de type ",typ_v," la fonction n'a
pas modifié la valeur de la variable.")
    print("Avant ou après exécution, var = ",var)
else:
    print("Pour une variable de type ",typ_v," la fonction a
modifié la valeur de la variable.")
    print("Avant exécution, var = ",sauve_var)
```

```
        print("Après exécution var = ",var)
```

Lors du passage d'une liste en paramètre, on dit qu'il y a *effet de bord*.

Copie de matrices

Testons le résultat de l'instruction `matrice_2 = matrice_1[:]`

```
[ ]  matrice_1 = [[1,2,3],[10,20,30]]
     matrice_2 = matrice_1[:]     #TEST de la copie
     (adr_1,typ_1) = adresse(matrice_1)
     (adr_2,typ_2) = adresse(matrice_2)
     print("la variable matrice_1 est un ",typ_1," d'adresse ",adr_1)
     print("la variable matrice_2 est un ",typ_2," d'adresse ",adr_2)
     print("Adresses des lignes")
     (adr_10,typ_10) = adresse(matrice_1[0])
     (adr_11,typ_11) = adresse(matrice_1[1])
     (adr_20,typ_20) = adresse(matrice_2[0])
     (adr_21,typ_21) = adresse(matrice_2[1])
     print("adresse de la ligne 0 de matrice_1",adr_10)
     print("adresse de la ligne 1 de matrice_1",adr_11)
     print("adresse de la ligne 0 de matrice_2",adr_20)
     print("adresse de la ligne 1 de matrice_2",adr_21)
     print("essai de modification de la valeur de matrice_2 uniquement
     : ")
     matrice_2[1][0]= 35
     print("matrice_1 ",matrice_1)
     print("matrice_2 ",matrice_2)
```

Le résultat est-il celui attendu ?
Quel est le problème rencontré :

Testez avec la méthode `.copy()` comme ci-dessus pour les listes simples, et conclure :

```
[ ]  matrice_1 = [[1,2,3],[10,20,30]]
     matrice_2 = matrice_1.copy()
     (adr_1,typ_1) = adresse(matrice_1)
     (adr_2,typ_2) = adresse(matrice_2)
     print("la variable matrice_1 est un ",typ_1," d'adresse ",adr_1)
     print("la variable matrice_2 est un ",typ_2," d'adresse ",adr_2)
     print("Adresses des lignes")
     (adr_10,typ_10) = adresse(matrice_1[0])
     (adr_11,typ_11) = adresse(matrice_1[1])
```

```
(adr_20,typ_20) = adresse(matrice_2[0])
(adr_21,typ_21) = adresse(matrice_2[1])
print("adresse de la ligne 0 de matrice_1",adr_10)
print("adresse de la ligne 1 de matrice_1",adr_11)
print("adresse de la ligne 0 de matrice_2",adr_20)
print("adresse de la ligne 1 de matrice_2",adr_21)
print("essai de modification de la valeur de matrice_2 uniquement
: ")
matrice_2[1][0]= 35
print("matrice_1 ",matrice_1)
print("matrice_2 ",matrice_2)
```

Conclusion :
Ecrire un programme qui permet de faire une copie dite "profonde", c'est à dire dans laquelle la matrice 1 n'est pas modifiée si l'on modifie la matrice 2

[ ]

Il existe une méthode pour copier correctement les matrices : `.deepcopy()` Testez ci-dessous

[ ]
```
from copy import deepcopy
matrice_1 = [[1,2,3],[10,20,30]]
matrice_2 = deepcopy(matrice_1)
(adr_1,typ_1) = adresse(matrice_1)
(adr_2,typ_2) = adresse(matrice_2)
print("la variable matrice_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable matrice_2 est un ",typ_2," d'adresse ",adr_2)
print("Adresses des lignes")
(adr_10,typ_10) = adresse(matrice_1[0])
(adr_11,typ_11) = adresse(matrice_1[1])
(adr_20,typ_20) = adresse(matrice_2[0])
(adr_21,typ_21) = adresse(matrice_2[1])
print("adresse de la ligne 0 de matrice_1",adr_10)
print("adresse de la ligne 1 de matrice_1",adr_11)
print("adresse de la ligne 0 de matrice_2",adr_20)
print("adresse de la ligne 1 de matrice_2",adr_21)
print("modification de la valeur de matrice_2")
matrice_2[1][0]= 35
print("matrice_1 ",matrice_1)
print("matrice_2 ",matrice_2)
(adr_2,typ_2) = adresse(matrice_2)
print("la variable matrice_1 est un ",typ_1," d'adresse ",adr_1)
print("la variable matrice_2 est un ",typ_2," d'adresse ",adr_2)
```

[**Frederic Mandon**](mailto:frederic.mandon@ac-montpellier.fr), Lycée Jean Jaurès - Saint Clément de Rivière - France (2015-2019)