Variables (im)muables

December 11, 2020

1 Variables muables, variables immuables

Objectif du td : comprendre la différence de comportement des variables de type liste, par rapport aux entiers, chaînes etc..., lors du passage en argument dans une fonction.

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 la référence de la variable (l'adresse mémoire de la variable ne peut pas être connue, c'est l'interpréteur Python qui s'en charge). Cette référence 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 `reference(variable)` qui renvoie la référence 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 reference(variable):
    """
    Renvoie la référence en hexadécimal et le type d'une variable quelconque
    """
    ref = hex(id(variable))
    typ = type(variable)
    return ref, typ
```

1.1 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 = 1
var_2 = var_1    #TEST de la copie de variable
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
```

```
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
```

On peut remarquer que `var1`et `var2` désignent la même référence.
Modifions la valeur de `var_2`

[ ]:
```
var_1 = 1
var_2 = var_1    #TEST de la copie de variable
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
print("modification de la valeur de var_2")
var_2 = var_2 + 3
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
```

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 ce qu'elles désignent. A chaque modification de la variable dans l'exécution d'un programme, Python évalue l'expression à droite du signe =, puis crée une nouvelle variable de même nom, à un autre endroit (avec une autre référence). 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).
De nombreux langages de programmation encouragent l'utilisation de variables immuables. Le code est beaucoup plus solide et facile à entretenir, surtout lorsque plusieurs *fils d'exécution* (*thread*) en même temps. En effet, les threads sont des processus différents qui se partagent la mémoire, par exemple l'interface graphique d'un programme et le programme lui-même peuvent être deux threads. Comme ces threads se partagent la mémoire, l'accès à une information nécessaire à ces deux fils peut être très problématique si l'un des fils modifie la variable alors que l'autre fil suppose que cette valeur n'a pas été modifiée. Ce type d'accès à la mémoire est source de nombreuses failles de sécurité. Les variables qui ne sont pas immuables sont **muables** (**mutable**). Lors de la modification d'une variable, la référence est directement modifiée.

1.2 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
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
print("modification de la valeur de var2 (sans toucher à var1)")
var_2[2]= 5
(ref_1,typ_1) = reference(var_1)
```

```
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_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 référence à chaque modification dans l'exécution d'un programme. Par rapport à une variable immuable, on gagne en efficacité et on perd en sécurité.

1.3 Copie de listes bis

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

[ ]:
```
var_1 = [1,2,3]
var_2 = var1[:]    #TEST de la copie de variable
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
print("modification de la valeur de var2")
var_2[2]= 5
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
```

Conclure :

1.3.1 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
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
print("modification de la valeur de var2")
var_2[2]= 5
(ref_1,typ_1) = reference(var_1)
(ref_2,typ_2) = reference(var_2)
print("la variable var1 = ", var_1, " est un ",typ_1," de référence ", ref_1)
print("la variable var2 = ", var_2, "est un ",typ_2," de référence ", ref_2)
```

1.4 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,ref):
    """
    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")
    elif type(variable) is dict:
        variable["aleph"] = "infini"
    else:
        variable = variable*2
    (ref_var,typ_var) = reference(variable)
    if ref_var != ref:
        print("La variable a été recréée à l'intérieur de la fonction :")
        print("\tSon type est ",typ_var)
        print("\tLa référence locale (dans la fonction) est \t\t",ref_var)
        print(" \talors que la référence dans le programme principal est \t",␣
 ↪ref)
    else:
        print("La variable n'a pas été recréée à l'intérieur de la fonction")
        print("\tLa variable a pour adresse globale ",ref,\
              " aussi bien dans la fonction que dans le programme principal")
    return variable

var = 1   #à tester avec entier, flottant, chaîne, booléen liste, tuple,␣
 ↪dictionnaire
sauve_var = copy(var)
(ref_v,typ_v) = reference(var)
print("la variable var = ", var, " est un ",typ_v," de référence ",ref_v)
double_ou_rien(var,ref_v)

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

```
    print("\tPour une variable de type ",typ_v," la fonction a modifié la␣
↪valeur de la variable.")
    print("\tAvant exécution, var = ",sauve_var)
    print("\tAprès exécution var = ",var)
```

Compléter et **retenir** :
Les variables de type … sont des variables immuables
Les variables de type … sont des variables muables

Lors du passage d'une liste ou d'un dictionnaire en paramètre, on dit qu'il y a *effet de bord.* : la variable est modifiée par la fonction.

1.5 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
(ref_1,typ_1) = reference(matrice_1)
(ref_2,typ2) = reference(matrice_2)
print("la variable var1 = ", matrice_1, " est un ",typ1," de référence ", ref_1)
print("la variable var2 = ", matrice_2, "est un ",typ2," de référence ", ref_2)
print("Références des lignes")
(ref_10,typ10) = reference(matrice_1[0])
(ref_11,typ11) = reference(matrice_1[1])
(ref_20,typ20) = reference(matrice_2[0])
(ref_21,typ21) = reference(matrice_2[1])
print("Référence de la ligne 0 de matrice_1",ref_10)
print("Référence de la ligne 1 de matrice_1",ref_11)
print("Référence de la ligne 0 de matrice_2",ref_20)
print("Référence de la ligne 1 de matrice_2",ref_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()
(ref_1,typ_1) = reference(matrice_1)
(ref_2,typ2) = reference(matrice_2)
print("la variable var1 = ", matrice_1, " est un ",typ1," de référence ", ref_1)
print("la variable var2 = ", matrice_2, "est un ",typ2," de référence ", ref_2)
print("Références des lignes")
(ref_10,typ10) = reference(matrice_1[0])
```

```
(ref_11,typ11) = reference(matrice_1[1])
(ref_20,typ20) = reference(matrice_2[0])
(ref_21,typ21) = reference(matrice_2[1])
print("Référence de la ligne 0 de matrice_1",ref_10)
print("Référence de la ligne 1 de matrice_1",ref_11)
print("Référence de la ligne 0 de matrice_2",ref_20)
print("Référence de la ligne 1 de matrice_2",ref_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)
(ref_1,typ_1) = reference(matrice_1)
(ref_2,typ2) = reference(matrice_2)
print("la variable var1 = ", matrice_1, " est un ",typ1," de référence ", ref_1)
print("la variable var2 = ", matrice_2, "est un ",typ2," de référence ", ref_2)
print("Références des lignes")
(ref_10,typ10) = reference(matrice_1[0])
(ref_11,typ11) = reference(matrice_1[1])
(ref_20,typ20) = reference(matrice_2[0])
(ref_21,typ21) = reference(matrice_2[1])
print("Référence de la ligne 0 de matrice_1",ref_10)
print("Référence de la ligne 1 de matrice_1",ref_11)
print("Référence de la ligne 0 de matrice_2",ref_20)
print("Référence de la ligne 1 de matrice_2",ref_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)
(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@ac-montpellier.fr, Lycée Jean Jaurès - Saint Clément de Rivière - France (2015-2020)