W4-S5-C3-passage-par-reference

December 14, 2014

1 Passage d'arguments par référence

1.1 Complément - niveau intermédiaire

Entre le code qui appelle une fonction, et le code de la fonction elle-même:

```
In []: def ma_fonction(dans_fonction):
           print dans_fonction

       dans_appelant = 12
       ma_fonction(dans_appelant)
```

on peut se demander quelle est exactement la nature de la relation entre l'appelant et l'appelé, c'est-à-dire ici `dans_appelant` et `dans_fonction`.

C'est l'objet de ce complément.

1.1.1 Passage par valeur - passage par référence

Si vous avez appris d'autres langages de programmation comme C ou C++, on a pu vous parler de deux modes de passage de paramètres: * par valeur: cela signifie qu'on communique à la fonction, non pas l'entité dans l'appelant, mais seulement **sa valeur**; en clair, **une copie**; * par référence: cela signifie qu'on passe à la fonction une **référence** à l'argument dans l'appelant, donc essentiellement les deux codes **partagent** la même mémoire.

1.1.2 python fait du passage par référence

Certains langages comme Pascal - et C++ si on veut - proposent ces deux modes. En python, tous les passage de paramètres se font **par référence**.

Ce qui signifie qu'on peut voir le code ci-dessus comme étant - pour simplifier - équivalent à ceci

```
In []: dans_appelant = 12
       # ma_fonction (dans_appelant)
       # -> on entre dans la fonction
       dans_fonction = dans_appelant
       print dans_fonction
```

On peut le voir en instrumentant le code comme ceci (on rappelle que la fonction built-in `id` retourne l'adresse mémoire d'un objet)

```
In []: def ma_fonction(dans_fonction):
           print 'dans ma_fonction', id(dans_fonction)

       dans_appelant = 12
       print 'dans appelant', dans_appelant, id(dans_appelant)
       ma_fonction(dans_appelant)
```

1.1.3 Des références partagées

Et notamment, tout ce que l'on a vu la semaine passée sur les références partagées s'applique à l'identique.

```
In []: # on ne peut pas modifier un immuable dans une fonction
       def increment(n):
           n += 1

       compteur = 10
       increment(compteur)
       print compteur
```

```
In []: # on peut par contre ajouter dans une liste
       def insert(liste, valeur):
           liste.append(valeur)

       liste = range(3)
       insert(liste, 3)
       print liste
```

Pour cette raison, il est important de bien préciser, quand vous documentez une fonction, si elle fait des effets de bord sur ses arguments (c'est-à-dire qu'elle modifie ses arguments), ou si elle produit une copie. Rappelez-vous par exemple le cas de la méthode `sort` sur les listes, et de la fonction de commodité `sorted`, que nous avions vues en semaine 2.

De cette façon, on saura s'il faut ou non copier l'argument avant de le passer à votre fonction.