ITC – TD n°4

Applications de la récursivité

On s'intéresse dans ce problème à deux applications dans lesquelles la récursivité permet d'écrire des algorithmes de façon naturelle.

On placera dans le dossier de travail, à côté du fichier de code, le fichier de module `module_recursivite.py` et on effectuera l'import suivants en début de code :

```
import module_recursivite as rec
```

I. Le problème des copies de listes

On rappelle qu'en Python, utiliser l'opérateur d'affectation avec une liste revient à créer une nouvelle étiquette sur la même liste, autrement dit `l2 = l1` fait qu'une modification de `l2` modifiera `l1`, puisqu'il s'agit de la même liste.

1.1 – Si on exécute les lignes suivantes, que vaut `l1[0]` ?

```
In [1]: l1 = [9, 3, 2]
In [2]: l2 = l1
In [3]: l2[0] = -1
```

1.2 – Écrire une fonction `copie_liste` qui prend en paramètres une liste et renvoie une copie, au sens où elle règle le problème précédent, sans se préoccuper de la nature des éléments de la liste.

La question se complexifie quand on considère que les listes peuvent elle-mêmes contenir...des listes ! dans ce cas, par exemple `l1[0][0]` contient -1 après les instructions suivantes.

```
In [1]: l1 = [[9, 3], [6, 6]]
In [2]: l2 = copie_liste(l1)
In [3]: l2[0][0] = -1
```

1.3 – Proposer une fonction récursive `copie_profondeur` qui prend en paramètre une liste et renvoie une liste **réellement indépendante**, c'est-à-dire que ses éventuelles sous-listes sont elles-mêmes copiées de façon indépendante. Pour déterminer si une variable est une liste ou non, on pourra utiliser `type(var) == list`, qui sera `True` pour une liste et `False` sinon.

> Le module copy contient les fonctions `copy.copy` et `copy.deepcopy` qui font chacune ce travail de copie superficielle ou en profondeur.

II. Parcours de dossiers

On s'intéresse ici au parcours d'une arborescence de dossiers (ou répertoires) et de sous-dossiers sur le système de stockage d'un ordinateur. La représentation des données est la suivante :

- un dossier est représenté par une liste ;
- son élément en position 0 est le nom du dossier ;
- les autres éléments sont les éléments du dossier : soit des `str` (fichiers représentés par leur nom) soit des `list` (sous-dossier).

Par exemple, la liste

```
['dossier', ['ss-d1', 'f1.txt', 'f2.txt'],
            ['ss-d2', 'f3.txt', 'f4.txt', ['secret', 'fa.txt']],
            'f5.txt', 'f6.txt']
```

représente l'arborescence suivante :

```
* dossier
   | * ss-d1
   |    | f1.txt
   |    | f2.txt
   | * ss-d2
   |    | f3.txt
   |    | f4.txt
   |    | * secret
   |    |     | fa.txt
   | f5.txt
   | f6.txt
```

Dans toute la suite, on appellera « dossier » une liste Python représentant un dossier selon cette structure, et on appellera « fichier » un élément de ce dossier de type `str`.

> On pourra récupérer deux arborescences pour les tests avec les instructions `dos0 = rec.dossier_test(0)` et `dos1 = rec.dossier_test(1)`.

2.1 – Écrire une fonction `compte_fichiers` qui prend en paramètre un dossier et renvoie le nombre de fichiers contenus dans le dossier et tous ses sous-dossiers.

Tests : `compte_fichiers(dos0)` renvoie 10 et `compte_fichiers(dos1)` renvoie 39.

2.2 – Écrire une fonction `trouver_chemin` qui prend en paramètre un dossier et un nom de fichier, et renvoie dans une liste l'enchaînement des dossiers à suivre pour trouver le fichier en question. Dans l'exemple précédent, `trouver_chemin(dos, 'f2.txt')` renvoie `['dossier', 'ss-d1', 'f1.txt']`

Tests : `trouver_chemin(dos0, "nb_pkmn.dat")` renvoie
`['racine', 'Top secret', 'projet nouveau PKMN', 'nb_pkmn.dat']`
et `trouver_chemin(dos1, "option.py")` renvoie
`['python3', 'debpython', 'option.py']`

2.3 – Écrire une fonction `dossier_parent` qui prend en paramètre un dossier et deux noms de fichier, et renvoie le nom du sous-dossier le plus profond qui contient les deux fichiers demandés. Dans l'exemple précédent,
`dossier_parent(dos, 'f3.txt', 'fa.txt')` renvoie `'ss-d2'`

Tests : `dossier_parent(dos0, "nb_pkmn.dat", "photo nulle.jpg")` renvoie `'racine'`
et `dossier_parent(dos1, "interpreter.py", "version.cpython-38.pyc")` renvoie
`'debpython'`

2.4 – Écrire une fonction `affiche_dossier` qui affiche sous forme d'arborescence le contenu d'un dossier, de façon similaire à l'exemple ci-dessus.