QUIZ 17 - LES RÉFÉRENCES PARTAGÉES (4/5 points)

Références partagées (1)

On définit une liste comme étant

```
entier = 0
liste = [entier, entier, entier]
```

après quoi on fait

```
liste[0] = 1
```

Qu'obtient-on si on imprime `liste` à ce stade ? :

- `[0, 0, 0]`
- `[1, 0, 0]` ✔
- `[1, 1, 1]` ✖

EXPLANATION

`liste` ne contient pas de référence partagée dans ce scénario, l'affectation de `liste[0]` ne change pas les deux autres éléments de la liste.

Références partagées (2)

On considère le scénario suivant :

```
cellule = [0]
liste = [cellule, cellule, cellule]
# puis
liste[0][0] = 1
```

- [ ] `[[0], [0], [0]]`
- [ ] `[[1], [0], [0]]`
- [x] `[[1], [1], [1]]` ✔

EXPLANATION

Cette fois on modifie en place la liste référencée par `cellule`, qui est partagée par les trois éléments de liste, on est dans le cas d'une référence partagée, la modification affecte les 3 éléments de la liste.

Références partagées (3)

On considère la variante suivante:

```
cellule = [0]
liste = [cellule, cellule, cellule]
# puis
cellule[0] = 1
```

Qu'obtient-on comme valeur pour `liste` ? :

- [ ] `[[0], [0], [0]]`
- [ ] `[[1], [0], [0]]`
- [x] `[[1], [1], [1]]` ✔

EXPLANATION

À nouveau, on modifie ici en place la liste référencée par `cellule`, cette situation est exactement identique à celle de l'exercice 2.

On considère enfin la variante suivante:

```
cellule = [0]
liste = [cellule, cellule, cellule]
# puis
cellule = [1]
```

Qu'obtient-on comme valeur pour `liste` ? :

- [x] `[[0], [0], [0]]`
- [ ] `[[1], [0], [0]]`
- [ ] `[[1], [1], [1]]`

EXPLANATION

Cette fois, on change la valeur de `cellule`, mais cela n'affecte pas du tout `liste`.

Si vous avez encore des difficultés avec l'une de ces 4 premières questions, vous pouvez les exécuter telles quelles dans `http://pythontutor.com/visualize.html#mode=edit` afin de bien décortiquer les mécanismes en jeu.

Les limites de la copie

L'opérateur `*` sur les listes crée des références partagées, comme on peut le voir ici :

```
>>> liste = 3 * [ [0] ]
>>> liste
[[0], [0], [0]]
>>> liste[0][0] = 1
>>> liste
[[1], [1], [1]]
```

- il faut faire une *shallow copy*
- il faut faire une *deep copy*
- il faut construire la liste autrement ✔

EXPLANATION

Dans ce cas, une *shallow copy* ou une *deep copy* ne sera d'aucune aide, puisque cela va créer un nouvel objet, mais avec la même structure, donc les mêmes références partagées que l'objet original. La seule solution est de construire la liste autrement.

On pourrait faire par exemple, en extension:

```
liste = [[0], [0], [0]]
```

Ou alors avec une boucle `for` :

```
liste = []
for i in range(3):
    liste.append([0])
```

Ou encore avec une compréhension de liste:

```
liste = [[0] for i in range(3)]
```

Vous avez utilisé 3 essais sur 3

Mentions légales