TP 4 (programmation) - Listes et fonctions.

Comme d'habitude, vous devez rendre votre TP dans le délai indiqué. Lorsqu'un fichier contenant les doctests est fourni, vous devez le compléter. Sinon, vous devez écrire un programme par exercice, avec comme nom **exoX.py** (X est le numéro de l'exercice) et mettre en commentaires les tests que vous avez faits pour vérifier que votre programme fonctionne.

Exercice 1

Récupérez le programme `liste100000.py` sur la page du cours. Ce programme construit la liste des nombres entre 0 et 99 999 de deux manières différentes et affiche le temps d'exécution pour chacune des méthodes. Vous pouvez consulter la documentation du module `time` ici : `http://docs.python.org/3/library/time.html`

Expliquez l'énorme différence de temps entre les deux méthodes de construction d'une part et entre les deux méthodes de destruction d'autre part.

Exercice 2 (Salutations)

Complétez le fichier `exo2.py` fourni. **Ne changez surtout pas les doctests !** Ils vous indiquent le résultat attendu. Vous pouvez néanmoins rajouter vos propres tests dans le `main`.

Vous pourrez lancer votre programme soit normalement, soit uniquement les doctests ainsi :

```
$ python3 -m doctest -v exo2.py
```

1. Écrivez une fonction `bonjour` sans argument qui **affiche** le texte *bonjour*.
2. Écrivez une fonction `dis_bonjour` qui prend un entier `n` en argument et **affiche** `n` fois le texte *bonjour*, sur des lignes différentes.
3. Écrivez une fonction `renvoie_bonjour` qui prend un entier `n` et **renvoie** la chaîne composée de `n` fois le mot *bonjour*, séparés par des espaces. Attention, cette fonction n'affiche rien !

Exercice 3 (Fonctions en pagaille)

Complétez le fichier `exo3.py` fourni et testez vos fonctions :
— en vérifiant avec les doctests (attention, cela ne garantit pas que votre code est correct !),
— et en utilisant vos fonctions dans la partie `main` du programme.

Autant que possible, évitez de réécrire plusieurs fois un code qui fait la même chose. Les fonctions sont réutilisables d'une question à l'autre. Vous pouvez ajouter vos propres fonctions auxiliaires. Enfin, si un doctest est censé provoquer une `AssertionError`, vous pouvez utiliser le mot-clé `assert` pour la déclencher.

1. Écrivez une fonction `carre` qui prend un entier en argument et renvoie son carré.
2. Écrivez une fonction `liste_aleatoire` qui prend deux entiers `n` et `b` en arguments et renvoie une liste de taille `n` contenant des nombres entre 1 et `b` (compris) tirés au hasard.
3. Écrivez une fonction `liste_carres` qui prend en paramètre une liste `lst` et renvoie la liste des carrés des éléments de `lst`.
4. Écrivez une fonction `max_liste` qui prend une liste non vide en argument et renvoie le plus grand entier apparaissant dans la liste (vous n'avez pas le droit d'utiliser la fonction `max` de Python). Testez-la sur une liste aléatoire.
5. Écrivez une fonction `singletons` qui prend une liste d'entiers et renvoie la liste contenant les mêmes nombres mais au plus une fois et dans l'ordre de leur première apparition.

   Par exemple, `singletons([2, 1, 2, 1, 3, 2, 1, 4])` renverra la liste `[2, 1, 3, 4]`.

6. Écrivez une fonction `indice` qui prend une liste et un entier `n` et renvoie l'indice de la première occurrence de `n` dans la liste ou `-1` s'il n'y est pas.

7. Écrivez une fonction `nb_occurrences` qui prend un nombre `n` et une liste `lst` et renvoie le nombre de fois que l'entier `n` apparaît dans la liste `lst`. Par exemple,

```
>>> liste = [1, 1, 1, 2, 9, 4, 2, 2, 1, 1, 1, 9, 1, 2, 9, 4, 2, 2, 1]
>>> [nb_occurrences(1, liste), nb_occurrences(5, liste)]
[8, 0]
```

8. Écrivez une fonction `toutes_occurrences` qui prend une liste `lst` en paramètre et renvoie une liste de couples (élément, nombre de fois où il apparaît dans la liste) pour chaque élément différent de la liste. Un couple est simplement représenté par une liste de deux éléments.

   Par exemple :

```
>>> toutes_occurrences(liste)
[[1, 8], [2, 6], [9, 3], [4, 2]]
```

9. Écrivez une fonction `plus_frequent` qui prend une liste `lst` d'entiers non vide en argument et qui renvoie le nombre qui apparaît le plus de fois dans la liste. Si plusieurs nombres sont possibles, on renverra le plus grand.

   Par exemple, dans la liste `[1, 2, 1, 1, 2, 2, 3, 3, 4]`, les nombres 1 et 2 apparaissent tous les deux 3 fois. On renverra donc 2 qui est le plus grand des candidats.