TP 5 (programmation) - Listes.

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 (Premières listes)

(a) Dans votre fichier, déclarez la liste `lst` suivante : `[14, 7, 6, 12, 2, 3, 3, 10]` et affichez-la.

(b) Ensuite, écrivez le code qui modifie la liste de telle façon que son dernier élément soit divisé par 2 et réaffichez-la. Attention, votre programme doit continuer de fonctionner même si on change le nombre d'éléments stockés dans `lst`.

(c) Modifiez la liste de telle façon que l'on retranche 1 à tous ses éléments et réaffichez-la.

(d) Ensuite, affichez tous les éléments de la liste sur des lignes différentes. Écrivez une première version en utilisant un `for` et la fonction `range`, et une seconde version avec un `for` mais sans `range`.

(e) Puis affichez seulement les nombres pairs de cette liste (sans créer de nouvelle liste).

(f) Ensuite, affichez 10 fois chaque élément de la liste sur la même ligne séparés par des espaces. Vous n'avez pas le droit d'utiliser l'opérateur $*$ pour cette question.

Pensez à utiliser `print('...', end=' ')` pour ne pas revenir à la ligne après l'affichage.

(g) Affichez chaque élément autant de fois que sa valeur. Par exemple, 6 sera affiché 6 fois.

(h) Enfin, pour chaque élément dans la liste, affichez-le en lui ajoutant 1 s'il est pair, et -1 s'il est impair (facultatif : essayez d'écrire une version sans utiliser de `if`).

À la fin de l'exercice, votre programme devra afficher exactement :

```
Question 1
[14, 7, 6, 12, 2, 3, 3, 10]
Question 2
[14, 7, 6, 12, 2, 3, 3, 5]
Question 3
[13, 6, 5, 11, 1, 2, 2, 4]
Question 4
13
6
5
11
1
2
2
4
Question 5
6
2
2
4
```

```
Question 6
13  13  13  13  13  13  13  13  13  13
6  6  6  6  6  6  6  6  6  6
5  5  5  5  5  5  5  5  5  5
11  11  11  11  11  11  11  11  11  11
1  1  1  1  1  1  1  1  1  1
2  2  2  2  2  2  2  2  2  2
2  2  2  2  2  2  2  2  2  2
4  4  4  4  4  4  4  4  4  4
Question 7
13  13  13  13  13  13  13  13  13  13  13  13  13
6  6  6  6  6  6
5  5  5  5  5
11  11  11  11  11  11  11  11  11  11  11
1
2  2
2  2
4  4  4  4
Question 8
12 7 4 10 0 3 3 5
```

Exercice 2 (Les listes sont *modifiables*)

Voici une suite d'instructions exécutées au *top-level* :

```
>>> my_list = [1, 7, 19, 42]
>>> your_list = my_list
>>> your_list
>>> your_list[0]
>>> your_list[0] = 13
>>> your_list
>>> my_list
```

Répondez aux questions dans un fichier texte nommé `exo2.txt`.

(a) Que vont afficher les lignes 3, 4, 6 et 7 ?

(b) Expliquez ce qui s'est passé à la ligne 7.

(c) Que faut-il rajouter pour que `my_list` ne soit pas modifiée lorsque que l'on modifie `your_list` ?

Exercice 3 (Créer et modifier des listes)

Commencez par choisir au hasard un entier positif n entre 10 et 20. Pour rappel, la fonction `randrange(a, b)` du module `random` fournit un entier aléatoire entre a et b (exclus).

(a) Créez une liste `lst1` contenant n fois le nombre 1 en utilisant la fonction `append`.

(b) Créez une liste `lst2` contenant n fois le nombre 2 en utilisant l'opérateur $*$.

(c) Créez une liste `lst_random10` contenant n entiers aléatoires entre 0 et 10.

(d) Ensuite, construisez une liste `lst_random5` contenant les nombres de `lst_random10` qui sont strictement inférieurs à 5 et affichez-la avec la fonction `print`. N'oubliez pas que les listes sont modifiables : vérifiez que la liste initiale est restée la même !

(e) Construisez une liste `lst_random10_reverse` contenant les entiers de `lst_random10` dans l'ordre inverse et affichez-la (sans utiliser la fonction prédéfinie `reverse`).

(f) Construisez une liste `lst_random10_count` dont la i-ème case contient le nombre de fois que l'entier i apparaît dans `lst_random10`. Par exemple, avec `[4, 1, 7, 1, 0, 4, 1]`, on obtient `[1, 3, 0, 0, 2, 0, 0, 1, 0, 0, 0]`.

Exercice 4 (Dessiner une ville)

Pour cet exercice, vous pouvez réutiliser le fichier `dessin-init.py` du TP 3. Le but de l'exercice est de réaliser un programme dessinant une ville. Pour voir le résultat que l'on cherche à obtenir, récupérez l'exécutable `dessin-ville`. Vous pouvez lancer le programme en ligne de commande sous Linux (il faudra peut-être changer les permissions).

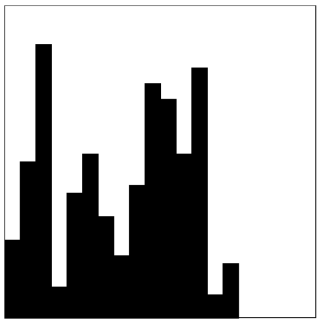

FIGURE 1 – hauteur

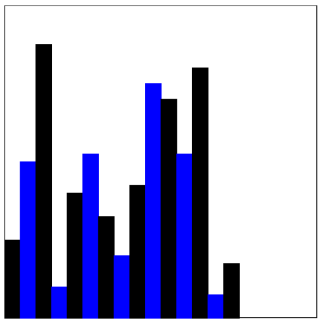

FIGURE 2 – couleur

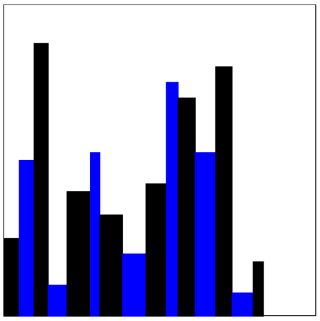

FIGURE 3 – largeur

(a) Déclarez la liste `lst` des hauteurs des immeubles dans la ville. Dans l'exemple :
`lst = [100, 200, 350, 40, 160, 210, 130, 80, 170, 300, 280, 210, 320, 30, 70]`

(b) Affichez une suite de rectangles de largeur fixe de 20 pixels et dont les hauteurs sont données par la liste `lst` (voir figure 1).

(c) Modifiez le programme pour que les couleurs des rectangles alternent entre le bleu et le noir. Pour ce faire, vous utiliserez une variable `couleur` qui vaudra soit `"blue"` soit `"black"` (voir figure 2).

(d) On souhaite désormais avoir des rectangles de largeur variable. Pour cela, créez la liste :
`lst_largeurs = [20, 19, 18, 24, 30, 12, 29, 30, 26, 15, 22, 26, 21, 27, 13]`

Modifiez votre programme pour qu'il affiche les rectangles dont les hauteurs sont données par la liste `lst` et les largeurs par `lst_largeurs`. Par exemple, le premier immeuble est de hauteur 100 et de largeur 20 (voir figure 3).

(e) Faites en sorte que la liste `lst_largeurs` soit créée en choisissant des valeurs aléatoires (entre 10 et 30, en utilisant `randrange`. N'oubliez pas de rajouter `from random import randrange` au début de votre programme). Affichez la liste `lst_largeurs` pour vérifier.

(f) Modifiez le programme pour que la largeur de la fenêtre corresponde exactement à la largeur de tous les immeubles.

(g) (facultatif) Modifiez votre programme pour qu'il affiche chaque rectangle avec un petit délai, en utilisant les fonctions `sleep` (du module `time`) et `mise_a_jour`.