Séance 5 : Listes, tuples et gestion mémoire

L1 – Université Côte d'Azur

Exercice 1 — Occurrence (⋆)

Pour chacune des fonctions, écrivez au moins trois tests avec `assert`.

1. En utilisant une boucle `for`, définissez une fonction `apparaît(x, L)` qui renvoie `True` si le nombre `x` apparaît dans la liste `L` et `False` sinon.
2. Comment auriez-vous fait avec le mot-clé `in` ?
3. Définissez la fonction `contient(L1, L2)` qui renvoie `True` si tous les nombres de la liste de nombres `L1` apparaissent dans `L2` et `False` sinon.
4. Définissez la fonction `commun(L1, L2)` qui renvoie le premier nombre de `L1` qui apparait aussi dans `L2`. Si un tel nombre n'existe pas, `commun(L1, L2)` renvoie `False`.

Exercice 2 — Produit scalaire (⋆)

Dans cet exercice, on identifie les tuples avec les coordonnées dans $\mathbb{R}^n$ muni de sa base orthonormale canonique.

1. Définissez une fonction `produit_scalaire(v1, v2)` qui prend en argument deux vecteurs représentés par des tuples de nombres de même longueur et qui renvoie leur produit scalaire. Par exemple, `produit_scalaire((1,2,3), (4,5,6))` renvoie 32 (= 4 + 10 + 18).
2. Déduisez-en une fonction `norme(v)` qui prend en argument un vecteur représenté par un tuple de nombres et qui renvoie sa norme. Par exemple, `norme((3,4))` renvoie `5.0`.

Exercice 3 — Produit externe (⋆)

Définissez une fonction `produit_externe(k, L)` qui prend en argument un nombre `k` et un vecteur $\vec{v}$ représenté par une **liste** de nombres, et qui renvoie la liste représentant le vecteur $k \cdot \vec{v}$, **sans modifier** `L`. Proposez une première solution avec la méthode `append` et une autre solution avec une liste définie par compréhension.

Exercice 4 — Substitution (⋆)

Définissez une *procédure* `remplace(x, y, L)` sans valeur de retour qui a pour effet de remplacer dans `L` tous les `x` par `y`. Par exemple, on aura au toplevel :

```
>>> L = [1, 2, 4, 1, 3]
>>> remplace(1, 5, L)
>>> L
[5, 2, 4, 5, 3]
```

Exercice 5 — Reconnaître une permutation (⋆⋆)

Définissez la fonction `est_permutation(L)` qui prend en argument une liste `L` de n entiers et qui renvoie `True` si `L` contient une et une seule fois tous les nombres de 0 à $n-1$. Par exemple, `est_permutation([0,2,1])` renvoie `True` mais `est_permutation([0,1,0])` et `est_permutation([1,3,2])` renvoient `False`. Cherchez d'abord une solution quelconque, puis ajoutez la contrainte de parcourir une seule fois `L` (complexité linéaire).

Exercice 6 — Crible d'Ératosthène (⋆⋆)

Écrivez une fonction `ératosthène(n)` qui renvoie la liste des nombres premiers inférieurs ou égaux `n` en appliquant l'algorithme du crible d'Ératosthène (demandez si nécessaire des explications à votre enseignant ou Wikipédia).

Exercice 7 — Les neurones de la lecture (⋆⋆–⋆ ⋆ ⋆)

1. Définissez une fonction `mélange(m)` qui renvoie un mot obtenu à partir de `m` en changeant aléatoirement l'ordre de ces lettres à l'exception de la première et de la dernière lettre. Par exemple, `mélange('cerise')` pourra renvoyer `'creise'` ou `'cierse'`. *Indication* : Vous pouvez utiliser `list` pour convertir une chaîne de caractères en liste et écrire vous-même la fonction inverse `liste_vers_chaîne(L)`. La fonction `random.shuffle` permet de mélanger les éléments d'une liste.
2. Définissez une fonction `liste_des_mots(m)` qui prend en argument un texte composé de mots et de ponctuation (`',.:; !?`) et qui renvoie la liste des mots et ponctuations qui constituent ce texte. Par exemple,

```
>>> liste_des_mots('Un: deux? Trois!')
['Un', ':', ' ', 'deux', '?', ' ', 'Trois', '!']
```

3. Définissez une fonction `mélange_texte(s)` qui renvoie ce même texte où chaque mot a été mélangé comme à la question 1.
4. Mélangez l'article 1 de la déclaration universelle des droits de l'homme. Confirmez-vous ce que rapportent certains sites webs sur la capacité du cerveau humain à remettre dans le bon ordre les lettres en lecture rapide ?

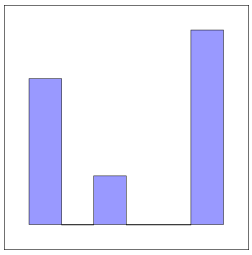
Exercice 8 — Statistiques (⋆ ⋆ ⋆)

Vous êtes libre pour gérer vous-même l'affichage (taille des marges, couleurs utilisés, etc) tant que le résultat est satisfaisant à vos yeux exigeants. Seule condition : le code doit fonctionner quelque soit la taille de la fenêtre.

1. Créer une fonction `effectif(L)` qui renvoie une liste de la forme `[(xmin,ymin), … ,(xmax,ymax)]`. Chaque couple `(x,y)` obtenu correspond à un élément `x` de `L` et à son nombre d'appartions `y` dans la liste `L`. Les valeurs `xmin` et `xmax` sont respectivement le minimum et le maximum de `L`.

```
>>> effectif([10,5,5,10,7,10,10,5])
[(5, 3), (6, 0), (7, 1), (8, 0), (9, 0), (10, 4)]
```

2. Avec Tk, faites un programme qui affiche l'histogramme correspondant à l'effectif. On prendra soin de n'afficher que les valeurs comprises entre le minimum et le maximum de `L` comme dans le graphisme ci-dessous correspondant à l'exemple précédant : `[10,5,5,10,7,10,10,5]`.



3. Écrivez une fonction `hasard(n)` qui simule `n` lancers de pièce et qui compte le nombre de piles.
4. Utilisez cette fonction pour générer une liste de 1000 valeurs aléatoires, puis, affichez son histogramme. Vous devez obtenir une courbe en cloche.