

Séance 6 : LISTES, GESTION MÉMOIRE ET EXCEPTIONS

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 sans boucle explicite mais 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 apparaît aussi dans `L2`. Si un tel nombre n'existe pas, `commun(L1, L2)` renvoie `False`.

Exercice 2 – Simulateur d'achat

On se donne une liste `stock` correspondant à un inventaire des produits d'un magasin.

```

1 >>> stock
2 [('Pommes', 10), ('Carottes', 5), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]
3 >>> maj_inventaire(stock, 'Carottes', 3)
4 >>> stock # il n'y a maintenant plus que 2 carottes !
5 [('Pommes', 10), ('Carottes', 2), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]

```

1. Écrivez une procédure `maj_inventaire(stock, produit, n)` qui **modifie** la liste `stock` en supprimant `n` occurrences du produit `produit`.
2. Modifier la fonction pour qu'elle lance `IndexError` si le produit n'existe pas et lance `ValueError` s'il n'y a pas assez de produits disponibles.
3. Écrivez un programme `achat(stock, produit, n)` qui fait appel à la fonction `maj_inventaire` et qui rattrape les éventuelles exceptions pour afficher un des trois messages suivant « Merci pour votre achat. », « Produit inexistant. » et « Produit en quantité insuffisante. »

```

1 >>> achat(stock, 'Pommes', 1)
2 Merci pour votre achat.
3 >>> achat(stock, 'Poivrons', 2048)
4 Produit en quantité insuffisante.
5 >>> achat(stock, 'Aubergines', 10)
6 Produit inexistant.

```

Exercice 3 – Crible d'Ératosthène

Écrivez une fonction `eratosthène(n)` qui renvoie la liste des nombres premiers inférieurs ou égaux `n` en appliquant l'algorithme du crible d'Ératosthène. On commence par créer un tableau dont les indices vont de 0 à `n` et dont toutes les cases sont à `True`. Lorsque l'on affecte `False` à la case d'indice `i`, on dit que l'on barre le nombre `i`. L'idée est de barrer tous les nombres inférieurs à `n` qui sont multiples d'un autre nombre ($\neq 1$) strictement plus petit ; à la fin de l'algorithme, les seuls nombres restants non barrés sont alors les nombres premiers.

Algorithme : On barre 0 et 1 qui ne sont pas premiers, puis pour chaque entier de 2 à \sqrt{n} , si `k` n'est pas barré, on barre tous les multiples de `k` (autre que `k`).

Exemple : Pour `n=100`, on barre les multiples de 2 (4, 6, 8, etc.), puis ceux de 3 (6, 9, 12, etc.), on ignore 4 qui est déjà barré, puis on recommence avec les multiples de 5, etc. On s'arrête après avoir fait les multiples de 10, car $10 \times 10 \geq 100$.

Exercice 4 – 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 5 – Les neurones de la lecture

- 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_chaine(L)`. La fonction `random.shuffle` permet de mélanger les éléments d'une liste.
- 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,

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

- Définissez une fonction `mélange_texte(s)` qui renvoie ce même texte où chaque mot a été mélangé comme à la question 1.
- 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?

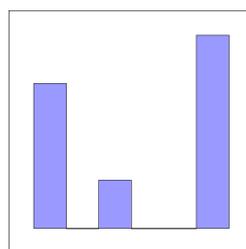
Exercice 6 – Statistiques

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.

- 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`.

```
1 >>> effectif([10,5,5,10,7,10,10,5])
2 [(5, 3), (6, 0), (7, 1), (8, 0), (9, 0), (10, 4)]
```

- 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]`.



- Écrivez une fonction `hasard(n)` qui simule n lancers de pièce et qui compte le nombre de piles.
- Utilisez cette fonction pour générer une liste de 1000 valeurs aléatoires correspondantes chacune à 500 lancers de pièces, puis, affichez son histogramme. Vous devez obtenir une courbe en cloche.

