Manipulation de listes et génération aléatoire

Devoir maison de préparation au projet Mastermind

Exercice 1 *Découverte du jeu du Mastermind* **(3 points)** Le *Mastermind* est un jeu de logique pour deux joueurs. Le **codificateur** choisit en secret une combinaison de 4 pions de couleur, parmi 6 couleurs (jaune, bleu, rouge, vert, orange, noir), et la cache ; il a le droit d'utiliser plusieurs fois la même couleur. Le **décodeur** doit retrouver cette combinaison, couleurs et ordre compris, en 10 propositions au plus.

Après chaque proposition, le codificateur répond avec de petites fiches :

— une **fiche noire** pour chaque pion de la bonne couleur placé au bon endroit ;

— une **fiche blanche** pour chaque pion d'une couleur présente dans le code, mais placé ailleurs ;

— rien pour les pions dont la couleur n'est pas dans le code.

La partie s'arrête quand le décodeur a reconstitué le code, ou quand il a épuisé ses 10 propositions.

1.5 pt **1.** Si un joueur propose la combinaison Jaune-Rouge-Bleu-Vert et que le code secret est Jaune-Bleu-Noir-Orange, combien de fiches noires et combien de fiches blanches le codificateur doit-il placer ? Justifiez votre réponse.

0.5 pt **2.** Le codificateur peut-il choisir une combinaison secrète contenant plusieurs fois la même couleur ? Justifiez votre réponse à l'aide du texte.

1 pt **3.** Dans la version classique du Mastermind avec 6 couleurs disponibles et 4 pions sans répétition, combien existe-t-il de combinaisons secrètes possibles différentes ? Justifiez votre raisonnement.

Exercice 2 *Conversion d'une liste en chaîne de caractères* **(2.5 points)** Écrivez en Python une fonction nommée `liste_vers_chaine(liste)` qui prend en paramètre une liste d'éléments (chaînes de caractères) et renvoie une chaîne de caractères obtenue en concaténant tous les éléments de la liste. **Contrainte :** vous ne devez **pas** utiliser la méthode `join()`.

Exemples

— `liste_vers_chaine(["J", "R", "V", "N"])` doit renvoyer `"JRVN"`.
— `liste_vers_chaine(["H", "U", "I", "L", "E"])` doit renvoyer `"HUILE"`.

Indications.

— Initialisez une variable chaîne vide : `resultat = ""`.
— Parcourez la liste avec une boucle `for`.
— À chaque itération, ajoutez l'élément courant à la chaîne avec l'opérateur `+`.

Question bonus : écrivez une version alternative de cette fonction en utilisant la méthode `join()` des chaînes de caractères. Documentez-vous sur cette méthode si nécessaire.

Exercice 3 *Compter les occurrences d'un élément* **(2.5 points)** Écrivez en Python une fonction nommée `compte_occurrences(element, liste)` qui prend en paramètres un élément et une liste, et renvoie le nombre de fois où cet élément apparaît dans la liste. **Contrainte :** vous ne devez **pas** utiliser la méthode `count()` des listes.

Exemples

— `compte_occurrences("J", ["J", "R", "J", "N"])` doit renvoyer `2`.
— `compte_occurrences("V", ["J", "R", "J", "N"])` doit renvoyer `0`.

— `compte_occurrences("R", ["R", "R", "R", "R"])` doit renvoyer `4`.

Indications.

— Initialisez un compteur à 0.
— Parcourez la liste avec une boucle `for`.
— Si l'élément courant est égal à `element`, incrémentez le compteur.

Question bonus : écrivez une version alternative de cette fonction en utilisant la méthode `count()` des listes. Vérifiez que les deux versions donnent les mêmes résultats.

Exercice 4 *Trouver la position d'un élément* **(3 points)** Écrivez en Python la fonction suivante, qui prend en paramètres un élément et une liste :

```
trouve_position(element, liste)
```

Elle renvoie :

— l'indice du **premier** élément trouvé s'il est présent dans la liste ;

— `-1` sinon.

Contrainte : vous ne devez **pas** utiliser la méthode `index()` des listes.

Exemples

— `trouve_position("J", ["J", "R", "V", "N"])` doit renvoyer `0`.
— `trouve_position("V", ["J", "R", "V", "N"])` doit renvoyer `2`.
— `trouve_position("O", ["J", "R", "V", "N"])` doit renvoyer `-1`.

Indications.

— Parcourez les indices de la liste avec une boucle `for` pour avoir à la fois l'indice et l'élément.
— Dès que vous trouvez l'élément recherché, renvoyez son indice avec `return`.
— Si la boucle se termine sans avoir trouvé l'élément, renvoyez `-1`.

Question bonus : la méthode `index()` des listes existe et permet de trouver l'indice d'un élément. Quelle différence majeure existe-t-il entre votre fonction et la méthode `index()` ? (Testez le comportement de `index()` quand l'élément n'est pas dans la liste.)

Exercice 5 *Copie de liste* **(3 points)** Lorsque nous travaillons avec des listes, il est parfois nécessaire de modifier une liste temporairement sans altérer la liste originale.

Considérons le code suivant :

```
1 liste_originale = ["A", "B", "C", "D"]
2 liste_modifiee = liste_originale
3 liste_modifiee[0] = "X"
4 print(liste_originale)
```

1 pt 1. Exécutez ce code. Qu'affiche `print(liste_originale)` ? Est-ce le résultat attendu ? Répondez en français et expliquez pourquoi ce phénomène se produit.

1.5 pt 2. Pour créer une véritable copie indépendante d'une liste, on utilise la notation suivante :

```
liste_copie = liste_originale[:]
```

Écrivez en Python le code ci-dessus modifié en utilisant cette notation pour que la modification de `liste_modifiee` n'affecte pas `liste_originale`.

0.5 pt

3. Vérifiez que votre code fonctionne : `liste_originale` doit rester inchangée après modification de `liste_modifiee`.

> **Pour information.** En Python, lorsqu'on écrit `liste2 = liste1`, on ne crée pas une nouvelle liste, mais seulement une nouvelle référence vers la même liste en mémoire. C'est pourquoi modifier `liste2` modifie aussi `liste1`. La notation `liste2 = liste1[:]` crée une véritable copie indépendante.

Exercice 6 *Génération aléatoire sans répétition* **(6 points)** Supposons que nous souhaitons sélectionner aléatoirement 4 éléments distincts parmi une liste d'éléments disponibles. Soit la liste suivante :

```
1 ELEMENTS = ["A", "B", "C", "D", "E", "F"]
```

3.5 pts

1. **Méthode manuelle (code Python).** Écrivez en Python un programme qui tire aléatoirement 4 éléments différents de la liste `ELEMENTS` en utilisant une boucle et en modifiant la liste au fur et à mesure. Vous devez utiliser :
   - la fonction `randint(a, b)` du module `random` qui renvoie un entier aléatoire entre `a` et `b` inclus ;
   - l'instruction `del liste[i]` qui supprime l'élément d'indice `i` dans `liste`.

> **Algorithme suggéré.**
> - Créez une copie de `ELEMENTS` pour ne pas modifier la liste originale (vous avez vu comment faire dans l'exercice précédent).
> - Créez une liste vide `selection = []`.
> - Répétez 4 fois : tirez un indice aléatoire, ajoutez l'élément correspondant à `selection`, puis supprimez-le de la copie.

2. **Méthode directe avec le module `random`.** Le module `random` de Python contient une fonction nommée `sample(population, k)` qui permet de sélectionner aléatoirement `k` éléments distincts dans la liste `population`.

0.5 pt

   a. Écrivez en Python la ligne de code qui permet d'importer uniquement la fonction `sample` depuis le module `random`.

1.5 pt

   b. Écrivez en Python une ligne de code qui utilise `sample` pour générer une liste de 4 éléments aléatoires distincts parmi `ELEMENTS` et stocke le résultat dans une variable nommée `selection`.

0.5 pt

   c. Testez votre code : exécutez-le plusieurs fois et vérifiez que vous obtenez bien à chaque fois 4 éléments différents choisis aléatoirement.

> **Pour aller plus loin (optionnel).** Le module `random` contient également une fonction `choices(population, k=n)` qui permet de sélectionner `n` éléments aléatoires dans `population` **avec** possibilité de répétition. Répondez en français : dans quelles situations pourrait-on préférer `sample` à `choices`, et inversement ?