et d'**Informatique**

Université de Strasbourg

Algorithmique et Programmation 1 : TP Noté Automne 2023

Durée : 1h30 ; tous documents autorisés ; aucune communication autorisée y compris avec une IA.

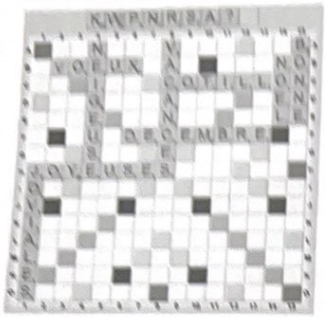
Consignes :

Dans ce TP, chaque question vous invite à concevoir et écrire une fonction en python.

1. Télécharger l'archive `tp_note.tar.gz`, l'extraire et aller dans le répertoire `tp_note` pour travailler.
2. **Vous écrirez votre code dans le fichier `tp.py`** sans modifier le code existant (les commentaires dans le code source l'indiquent). Les autres fichiers ne doivent pas être modifiés.
3. En fin d'épreuve, vous déposerez `tp.py` sur Moodle.
4. Vous pouvez tester chaque fonction écrite, en ouvrant le fichier `test.py`, en décommentant la fonction concernée, et en exécutant `test.py`.

1. Contexte

On veut écrire un jeu de scrabble. Le jeu est joué sur un plateau qui est une grille de 15x15 cases. Les 15 lignes du plateau sont identifiées par des lettres de A à O, les colonnes par un nombre de 1 à 15.



On considère ici qu'il n'y a qu'un seul joueur.

- Le joueur reçoit initialement 7 lettres tirées aléatoirement sac d'un qui en contient 100. Les 7 lettres sont posées sur un *support* (voir image).

- Il doit former un mot français existant avec les lettres reçues, et le poser sur la grille, soit horizontalement, soit verticalement.
- Au premier tour il peut poser son mot à n'importe quelle place sur le plateau du moment que le mot rentre dans la grille.
- Après avoir joué, les lettres utilisées sur le support sont remplacées par de nouvelles lettres tirées aléatoirement du sac, pour atteindre à nouveau 7 lettres (ou moins si le sac est vide).
- Le joueur peut à nouveau essayer de former un mot, en utilisant éventuellement les lettres des mots posés précédemment sur le plateau.
- Le jeu s'arrête quand le sac de lettres est vide et que le joueur ne possède plus de lettres.

La structure du déroulement du jeu vous est donné dans le fichier initial `tp.py` avec le programme principal (`main`) et la fonction `boucle_de_jeu`.

2. Initialisation et tirage (5 pts)

2.1 Initialisation du plateau (2 pts)

Ecrire une fonction `init_plateau` sans paramètre, qui créé une liste de listes représentant le plateau. La liste doit contenir 15 listes représentant les lignes. Chaque liste représentant une ligne contient 15 éléments, chaque élément étant un caractère espace (`' '`).

5 cases. Les 15 lignes du plateau sont identifiées par

155. Les 7 lettres sont posées sur un support (voir

le, soit horizontalement, soit verticalement.

ment que le mot rentre dans la grille.

les lettres tirées aléatoirement du sac, pour atteindre 7

s lettres des mots posés précédemment sur le plateau.

lettres.

programme principal (*main*) et la fonction

tant le plateau. La liste doit contenir 15 listes

ue élément étant un caractère espace (`' '`).

Tests Décommenter dans `test.py` la fonction `module_test.test_init_plateau()`.

2.2 Tirage des lettres (3 pts)

Cette fonction remplit le support avec des lettres tirées aléatoirement du sac.

Ecrire une fonction `tire_lettres` qui a deux paramètres : `support` et `sac`. La fonction ne retourne rien mais modifie les deux liste
et `sac`.

Tirer une lettre du sac consiste à

- choisir aléatoirement une position valide dans la liste `sac`, prendre la lettre à cette position,
- et l'ajouter à la liste `support`,
- supprimer la lettre tirée de `sac`.

La fonction répète cette opération jusqu'à ce que le support contienne 7 lettres ou que le sac soit vide.

Tests Décommenter dans `test.py` la fonction `module_test.test_tire_lettre()`.

3. Saisie utilisateur (4 pts)

Dans les trois questions suivantes, on traite la saisie de l'utilisateur afin de récupérer le sens et l'emplacement de pose du mot par l'utili

3.1 Convertir ligne (1 pts)

Ecrire une fonction `convertir_ligne` qui prend en paramètres une chaine `c` composée d'une lettre et rend le numéro de ligne dans la g
La lettre 'A' correspond à la ligne 0, 'B' correspond à la ligne 1, ..., 'O' correspond à la ligne 14.

On considère que le paramètre `c` est toujours valide (valeur entre 'A' et 'O') et il est inutile de le vérifier.

Tests Décommenter dans `test.py` la fonction `module_test.test_convertir_ligne()`.

3.2 Convertir colonne (1 pts)

Ecrire une fonction `convertir_colonne` qui prend en paramètres une chaine `c` représentant un nombre et qui rend un entier correspond
numéro de colonne dans la grille. La chaine '1' correspond à la ligne 0, '2' correspond à la colonne 1, ..., '15' correspond à la colonne 14.

On considère que le paramètre `c` est toujours valide et il est inutile de le vérifier.

Tests Décommenter dans `test.py` la fonction `module_test.test_convertir_colonne()`.

3.3 Analyse saisie (2 pts)

Ecrire une fonction `analyse_input` qui prend en paramètre une chaine de caractères décrivant le sens et la position où l'utilisateur va placer
son mot. La chaine saisie par l'utilisateur a la forme suivante :

- 1er caractère : est 'H' pour horizontal, 'V' pour vertical,
- 2e caractère : est une lettre désignant la ligne,
- 3e caractère ou 3e et 4e caractères : le numéro de colonne. Il faut utiliser les deux fonctions précédemment décrites.

La fonction rend un triplet `(sens,lig,col)` si la saisie utilisateur est correcte, où

- `sens` est 'H' ou 'V'
- `lig` et `col` sont des entiers entre 0 et 14.

La fonction rend un triplet vide `()` si la saisie est incorrecte.

La fonction doit donc tester la correction de la saisie utilisateur. On ne teste pas ici si le mot sort de la grille, ni que le mot recouvre un autre
mot déjà posé (testé dans une autre fonction).

Tests Décommenter dans `test.py` la fonction `module_test.test_analyse_input()`.

4. Jeu (11 pts)

La fonction `boucle_de_jeu` qui vous a été donnée (dans le fichier `tp.py`), après l'analyse de la saisie utilisateur, effectue trois actions :

- elle vérifie que le mot est jouable, c'est-à-dire qu'on peut poser le mot sur le plateau sans conflit avec les lettres déjà posées et sans sortir du plateau (4.3),
- elle vérifie que le joueur possède bien sur son support les lettres nécessaires (4.1),
- et enfin place le mot sur le plateau (4.2).

4.1 Vérifie support (3 pts)

4.1.1 Comptage (1 pts)

Ecrire une fonction `compte` qui prend deux arguments: un élément et une liste. La fonction retourne le nombre d'occurrences de l'élément dans la liste.

Tests Décommenter dans `test.py` la fonction `module_test.test_compte()`

4.1.2 Support Contient (2 pts)

Ecrire une fonction `support_contient` qui prend deux paramètres : `besoin` et `support` qui sont tous les deux des liste de lettres (liste de chaines de caractères).

La fonction rend `True` si toutes les lettres de `besoin` sont présentes dans `support`. Dans le cas contraire elle rend faux.

> Indication : On peut comparer le nombre d'occurences de chaque lettre dans `besoin` et vérifier qu'il y en a un nombre au moins aussi grand dans `support`.

Tests Décommenter dans `test.py` la fonction `module_test.test_support_contient()`.

4.2 Joue (3 pts)

Ecrire une fonction `joue` qui prend trois paramètres :

- `mot` : une chaine de caractères qui est le mot joué
- `position` : un triplet tel que renvoyé par `analyse_input`
- `plateau` : la grille de jeu

La fonction ne retourne rien mais modifie `plateau` en positionnant l'ensemble des lettres du mot dans la grille à l'aide de `position`. On place le mot en commençant avec la première lettre placée à la ligne et colonne, et dans le sens horizontal ou vertical indiquées dans `position`.

Toutes les vérifications de validité ont déjà été effectuées dans la fonction `verifie_jouable`.

Tests Décommenter dans `test.py` la fonction `module_test.test_joue()`.

4.3 Vérifie jouable (5 pts)

Écrire une fonction `verifie_jouable` qui prend trois paramètres :

- `mot` : une chaine de caractères qui est le mot joué
- `position` : un triplet tel que renvoyé par `analyse_input`
- `plateau` : la grille de jeu

La fonction retourne la liste `besoin` des lettres nécessaires pour pouvoir poser le mot sur le plateau. Si la pose est invalide elle retourne la liste vide.

L'algorithme de cette fonction est le suivant:

- on considère la première lettre `L` de `mot`,
- on examine le contenu de la case de grille aux coordonnées spécifiées dans `position`.
- si la case est `' '` (case vide), on ajoute la lettre `L` à `besoin`
- si la case n'est pas vide,
  - soit la lettre déjà sur le plateau est `L` (on ré-utilise une lettre d'un mot déjà posé), et il n'est pas nécessaire d'ajouter `L` à la liste `besoin`,
  - soit la lettre déjà sur le plateau n'est pas `L` et il y a un conflit : la pose est invalide, on affiche un message d'erreur (*Votre mot recouvre d'autres lettres !*), et on sort de la fonction en retournant la liste vide.
- on repète les étapes précédentes en considérant la deuxième lettre de `mot` et en examinant la case voisine de la précédente, en considérant le sens de pose (horizontal ou vertical). Si cette case voisine est hors du plateau, on affiche un message d'erreur (*Votre mo sort du plateau !*) et on sort de la fonction en retournant la liste vide.
- quand toutes les lettres du mot ont été examinées, on retourne la liste `besoin`.

Note : cet algorithme simplifié ne vérifie pas toutes les règles du jeu de Scrabble

Tests

Décommenter dans `test.py` la fonction `module_test.test_verifie_jouable()`.