université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

Introduction à l'informatique
L1

Devoir à la maison

1 Modalités

Date de rendu : Le devoir doit être rendu au plus tard le

jeudi 25 novembre 2021, 20h00

via le formulaire eCampus prévu à cet effet et disponible sur la page du cours.

Travail à rendre : vous ne devez déposer **que** le fichier `pendu.py` présent dans l'archive du devoir, une fois complété. Les autres fichiers (`corrige_pendu.py`, `mots.txt`) ne sont là que pour tester votre programme et n'ont pas à être rendus.

Barème : le devoir est sur 20 points. Le nombre indicatif de points est donné pour chaque question et reflète sa difficulté.

Plagiat : le devoir est **individuel**. Il est là pour vous préparer à l'examen. En cas de doute sérieux, les étudiant·e·s soupçonné·e·s de plagiat seront signalé·e·s à la commission disciplinaire de l'Université qui statuera sur le caractère avéré ou non du plagiat sur la base des documents rendus (cf. section 5.4 du règlement des études Premier Cycle).

Code Python : vous pouvez utiliser des constructions du langage Python non abordées en cours, uniquement si :

- elles sont compatibles avec la version de Python utilisée dans les salles de TP (c'est à dire Python 3.8)
- vous ne modifiez pas les lignes `import` se trouvant en début de fichier et n'en ajoutez pas de nouvelles

2 Description de l'archive

L'archive contient les fichiers suivants :

- Un fichier `mots.txt` contenant une liste de 59705 mots français, chacun sur une ligne. Les mots sont écrits en minuscules, en utilisant uniquement les lettres de `a` à `z` sans accent ni autre diacritique. Ainsi, les mots « école », « garçon » et « œuvre » sont écrits « `ecole` », « `garcon` » et « `oeuvre` ».
- Un fichier `pendu.py` que vous devez compléter aux endroits indiqués **et rendre**.
- Un fichier `corrige_pendu.py` contenant le code du corrigé, fonctionnel mais illisible (*i.e.* lire ce fichier ne vous aidera pas à trouver les réponses). Ce fichier est fourni pour éviter de rester bloqué sur une question sans pouvoir faire les suivantes. Par exemple, la question n°1 demande de définir une fonction `charge_mots(chemin)`. Si vous avez plus tard besoin d'appeler cette fonction mais que vous n'avez pas réussi à l'écrire, vous pouvez simplement appeler `corrige_pendu.charge_mots(chemin)` à la place.

3 Sujet

Le but de ce devoir est de réaliser un jeu de pendu en Python. Il est conseillé de lire intégralement l'énoncé et même de jouer quelques parties en utilisant le corrigé pour se donner une idée du jeu. Le corrigé peut être exécuté de la façon suivante :

```
$ python3 corrige_pendu.py
```

Au début de chaque partie, le programme demande au joueur de choisir une longueur de mot à deviner ou de presser « `q` » pour quitter (dans ce qui suit, le texte avec un fond gris représente les entrées de l'utilisateur) :

```
Saisir une longueur de mot ou q pour quitter: 5
```

Le programme va alors afficher le mot en remplaçant par un « . » les lettres inconnues. Puis il demande au joueur de saisir une lettre entre « `a` » et « `z` ». Après saisie de la lettre :

— si la lettre est dans le mot, le dessin du pendu n'est pas modifié

— sinon un bâton est ajouté au dessin du pendu

puis le programme redemande une lettre. La partie se termine soit quand le joueur a trouvé toutes les lettres, soit quand le nombre de tentatives atteint 15 (qui correspond au nombre de symboles du dessin du pendu). Une fois la partie terminée, le programme affiche un message indiquant le résultat (victoire ou échec) puis redemande au joueur de saisir une longueur de mot pour recommencer une partie ou `q` pour quitter. Voici un exemple de partie :

```
.....
Saisir une lettre entre a et z: e
  -
```

```
.....
Saisir une lettre entre a et z: a
  -
```

```
...a.
Saisir une lettre entre a et z: u
  --
```

```
...a.
Saisir une lettre entre a et z: o
  ---
```

```
...a.
Saisir une lettre entre a et z: i
  ---
```

```
.i.a.
Saisir une lettre entre a et z: p
```

```
   ---
  |
```

```
  .i.a.
  Saisir une lettre entre a et z: t
```

```
   ---
  |
```

```
  .ita.
  Saisir une lettre entre a et z: v
```

```
   ---
  |
```

```
  vita.
  Saisir une lettre entre a et z: l
```

```
   ---
  |
```

```
  Vous avez gagné !
  Saisir une longueur de mot ou q pour quitter: q
```

La sortie ci-dessous affiche une fin de partie perdante :

```
  .e.ut...e
Saisir une lettre entre a et z: w

  ---
 |   |
 o   |
/|\  |
/ \  |
     |
Vous avez perdu, le mot était: refutable
Saisir une longueur de mot ou q pour quitter:
```

Le jeu est relativement simpliste :

- en cours de partie, on ne peut pas quitter (sauf en appuyant sur CTRL-C) ;
- le programme n'empêche pas le joueur de saisir une lettredéjà saisie. Si c'est le cas, c'est un coup perdu (et le pendu gagne un symbole).

4 Question

0. (0.5 point) Remplacer les chaînes de caractères se trouvant en début de fichier pour définir votre nom, prénom et email (celui `@universite-paris-saclay.fr`, pas votre email personnel). Attention, il faut bien évidement mettre ces informations dans des chaînes de caractères.

1. (1.5 point) compléter la fonction `charge_mots(chemin)`. Cette dernière prend en argument le chemin d'un fichier texte et renvoie une paire $(t, lmax)$ où t est un tableau de chaînes de caractères représentant les mots du fichier et $lmax$ est la longueur du mot le plus long dans ce fichier.
2. (1.5 point) compléter la fonction `test_charge_mots()` afin que celle-ci charge le fichier `mots.txt` et effectue quatre tests distincts au moyen de `assert`. La longueur du plus long mot doit être 25, le nombre de mots doit être 59705 et deux mots (que vous choisissez) et dont vous repérez la ligne dans le fichier, doivent être à la bonne position dans le tableau.
3. (1.5 point) définir une fonction `mots_par_longueur(tab_mots, lmax)` qui prend en argument un tableau de chaînes de caractères `tab_mots` et la longueur du plus long mot de ce tableau et renvoie t, un tableau de tableaux tel que :
   — $t[0]$ vaut `[]` (un tableau vide) ;
   — Pour tout i compris entre 0 et $lmax$ (inclus), $t[i]$ est le tableau des mots de longueur i dans `tab_mots`.
   Par exemple si `tab_mots` vaut `['a', 'bonbon', 'code', 'dos', 'etre']` et `lmax` vaut 6 (car `bonbon` fait 6 lettres), alors `mots_par_longueur(tab_mots, lmax)` renvoie un tableau de la forme :

   ```
   [ [], ['a'], [], ['dos'], ['code', 'etre'], [], ['bonbon'] ]
   ```

   Ici, il n'y a pas de mots de longueur 0, 2 et 5, et il y a deux mots de longueur 4.
   Attention : on fera attention, lors de l'initialisation de t à faire en sorte que tous les tableaux intérieurs soit distincts (i.e. ne soient pas partagés en mémoire). De plus, on peut ajouter une valeur `v` à la fin d'un tableau `tab` en faisant :

   ```
   tab += [v]
   ```

4. (1 point) compléter la fonction `test_mots_par_longueur()` en utilisant l'exemple ci-dessus pour tester votre fonction au moyen d'`assert`s.
5. (1 point) compléter la fonction `choix_mot(tab_mots_long, l)` où `tab_mots_long` est un tableau de tableaux de chaînes de caractères tel que renvoyé par la fonction `mots_par_longueur` et `l` est une longueur de mot supposée valide et qui renvoie un mot aléatoirement choisi parmi ceux de longueur `l` dans `tab_mots`. Vous devez utiliser la fonction `randint` de Python.
6. (1 point) compléter la fonction `test_choix_mot()` afin d'effectuer au moins trois tests sur des longueurs différentes de mots.
   On appelle dans la suite un mot à deviner un *problème*.
7. (1 point) Un problème est un tableau de paires (l, n) où l est une lettre du mot à deviner et b un booléen valant `True` si la lettre l a été trouvée et `False` sinon. Compléter la fonction `init_probleme(mot)` qui renvoie un problème correspondant au mot `mot` pour lequel aucune lettre n'a été trouvée. Par exemple, pour le mot `'code'`, la fonction renvoie le problème

   ```
   [ ('c', False), ('o', False), ('d', False), ('e', False) ]
   ```

8. (1 point) Compléter la fonction `test_init_probleme()` afin d'effectuer au moins un test sur un mot de cinq lettres ou plus.
9. (1 point) Compléter la fonction `num_inconnues(probleme)` qui renvoie le nombre de lettres non trouvées dans le problème passé en argument.
10. (1 point) Compléter la fonction `test_num_inconnues()` afin d'effectueur au moins trois tests sur trois problèmes différents, dont au moins un avec toutes les lettres trouvées et un avec aucune lettre trouvée.
11. (1.5 point) Compléter la fonction `joue(probleme, l)` qui simule un essai avec la lettre `l` passée en argument sur le problème. Cette fonction renvoie **une copie** du problème passé en argument dans laquelle le booléen associé à la lettre `l` est mis à `True`, s'il valait `False` si la lettre `l` est présente dans le problème. Dans tous les autres cas, un problème identique à celui passé en argument est renvoyé.
12. (1 point) Compléter la fonction `test_joue()` afin que celle-ci effectue au moins trois tests de la fonction `joue`.
13. (1.5 point) Compléter la fonction `affiche_probleme(probleme)` qui étant donné un problème affiche dans la console la chaîne de caractères correspondant, i.e. celle où les lettres associées à `True` dans le problème sont affichées et où les lettres associées à `False` sont remplacées par un « . ». Par exemple, pour

le problème `[('c', True), ('o', False), ('d', True), ('e', False)]`, représentant le mot « code » dont la première et dernière lettre ont été devinées, la fonction affiche : `c.d.`

14. (1.5 point) Compléter la fonction `affiche_pendu(n)` qui dessine les `n` premiers traits du pendu. On supposera que `n` est inférieur ou égal à 15. On utilisera la variable globale `PENDU`, déjà définie dans le fichier, et qui est un tuple contenant 6 chaînes de 6 caractères chacune. On pourra utiliser l'algorithme suivant :
    — initialiser k à 0
    — pour chaque ligne i à afficher
    — — pour chaque caractère c la ligne `PENDU[`i`]`
        — — si c vaut `' '` (espace) alors l'afficher
            — sinon, si k est inférieur à n alors afficher c et incrémenter k
            — sinon afficher un espace
        — afficher un retour à la ligne

    On peut afficher une chaîne `s` sans retour à la ligne en utilisant la fonction `print(s, end='')`. De plus, on peut afficher un retour à la ligne en appellant la fonction `print()`.

15. (2.5 points) Compléter la fonction `partie(mot)` qui effectue une partie du jeu. Cette fonction prend en argument une chaîne de caractères représentant le mot à deviner. On pourra procéder de la façon suivante :
    — transformer le mot en problème
    — tant qu'il reste des lettres à deviner et que le pendu n'est pas complètement dessiné (15 traits) :
        — afficher le problème ;
        — demander une lettre (et répéter l'opération tant que le joueur n'entre pas une lettre valide) ;
        — faire un mouvement de jeu
        — si le nombre de lettres inconnues n'a pas diminuer, dessiner le pendu avec un trait de plus
    — si on a deviné le mot, afficher un message de victoire, sinon un message d'échec.