Table des matières

1 Introduction

OBJECTIF : Travailler avec :

1. les chaines de caractères,
2. les fichiers (en lecture),
3. les dictionnaires.

REMARQUE : La note ne valide pas seulement le résultat de votre programme, mais également son style :
choix approprié des noms de variables, présence de documentation pour les fonctions importantes, commentaires pertinents, découpage des fonctions en sous-fonctions...
Vérifiez ces points avant de demander à votre intervenant de valider votre code.

Définition :
Le Jeu du pendu : Le but du jeu du pendu est de découvrir un mot en devinant ses lettres. L'objectif de ce TP est de pouvoir jouer contre l'ordinateur. Pour cela, nous allons partir d'un fichier qui contient tous les mots du célèbre dictionnaire d'Émile Littré.

2 Préliminaire :dictionnaires

Les dictionnaires ressemblent aux tableaux mais les cases ne sont pas "numérotées" par des entiers. Par exemple :

```
jours = {
  'janvier': 31,
  'f\'evrier': 28,
  'f\'evrierB': 29,
  'mars':31,
  'avril':30,
  'mai':31,
  'juin':30,
  'juillet':31,
  'aout':31,
  'septembre':30,
  'octobre':31,
  'novembre':30,
  'decembre':31
}
```

Définition :
dictionnaire

L'accès à une case se fait de deux manières :

1. comme pour un tableau, avec jours["janvier"],
2. avec la méthode get : jours.get("janvier", ...).

```
>>> print(jours["january"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'january'
>>> print(jours.get("january", 0))
0
```

1. Écrivez une fonction nb_jours qui prend en argument une liste de noms de mois (des chaines de caractères) et renvoie le nombre de jours total de ces mois. Si un mois n'est pas valide, on ne compte pas de jours :

```
>>> print(nb_jours([]))
0
>>> print(nb_jours(['mars', 'avril']))
61
>>> print(nb_jours(['mars', 'april']))
31
>>> print(nb_jours(['avril', 'fevrier', 'novembre', 'octobre', 'aout', 'mai',
                    'juillet', 'janvier', 'juin', 'mars', 'septembre', 'decembre']))
365
```

Pour les questions qui suivent, vous pourrez utiliser le fichier littre.txt qui contient tous les mots du dictionnaire « Littré ». Ce fichier contient un mot par ligne et en voici un extrait :

```
ABAQUE
ABAS
ABASOURDI
ABASOURDIR
ABAT
ABATAGE
ABATANT
ABATELLEMENT
ABATIS
ABATTABLE
```

On peut récupérer la liste de toutes les lignes d'un fichier avec la fonction suivante :

```
def lire_mots(nom_fichiers):
    """fonction qui récupère la liste des mots dans un fichier

    préconditions
      - nom_fichier, de type chaine de caractère : nom du fichier contenant les mots
        (un par ligne)

    postconditions : liste de chaine de caractères
    """
    liste_mots = []       # le tableau qui contiendra les lignes
    f = open(nom_fichier, encoding="UTF-8") # on ouvre le fichier
```

```
    ligne = f.readline()      # une variable temporaire pour récupérer la ligne courante dans le fichier f
    while ligne != "":
        liste_mots.append(ligne.strip())    # on rajoute la ligne courante dans le tableau
        ligne = f.readline()                # on récupère la ligne suivante
    return liste_mots
```

2. Utilisez la fonction lire_mots (sans la modifier) pour trouver le nombre de mots dans le fichiers littre.txt. Ajoutez un commentaire dans votre fichier Python en indiquant le nombre de mots trouvés.
   État du jeu : Pendant une partie, nous représenterons l'état connu du joueur par une chaine de caractères. Sa taille sera le nombre de lettres du mot à trouver et les lettres inconnues seront remplacées par le caractère _.
   Remarque : pour éviter de tester les minuscules / majuscules, vous pouvez toujours convertir un caractère (ou une chaine de caractères) en majuscules avec

```
c = c.upper()
```

3. Écrivez la fonction "nouvel_etat" :

```
def nouvel_etat(mot, etat, c):
    """fonction qui renvoie le nouvel état après proposition d'une lettre c

    préconditions :
      - mot, de type chaine de caractères,
      - etat, de type chaine de caractères : les lettres inconnues sont
        représentées par des '_'. Cette chaine a la même longueur que le paramètre
        mot,
      - c, de type caractère : la lettre proposée.

    postconditions : chaine de caractère où les "_" correspondant au paramètre c ont été remplacés par c
    """
```

   Pour cela, le plus simple est de parcourir les caractères de la chaine mot, en reconstruisant l'état au fur et à mesure :

   - si le caractère est différent du caractère c passé en argument, on garde le caractère de *etat*et on continue
   - si le caractère est égal au caractère c passé en argument, on colle un c dans le nouvel état.

   Attention, comme le mot peut contenir plusieurs fois la même lettre, il ne faut pas s'arrêter à la première occurrence de c trouvée !

   L'exécution suivante :

```
mot = "CANARI"                              # mot secret
etat = "C___RI"                     # état actuel : le joueur a déjà trouvé les lettres "C", "R" et "I"
etat2 = nouvel_etat(mot, etat, "a")
print("mot :", etat2)                       # permet d'afficher le nouvel etat
```

   devra afficher

```
mot : CA_ARI
```

   PREMIERE version : On peut maintenant écrire une première version du jeu du pendu :

   - on commence par tirer un mot au hasard dans le dictionnaire,
   - on initialise une variable "etat" avec autant de "_" que de lettres dans le mot tiré,
   - tant que le nombre d'erreurs est plus petit que 8, on affiche l'état et on demande une lettre à l'utilisateur. La fonction nouvel_etat permet de comparer l'ancien état et le nouveau :

   - si les deux états sont identiques, c'est que la lettre proposée n'a rien changé : elle n'était pas dans le mot secret et il faut comptabiliser une erreur supplémentaire,
   - si les deux états sont différents, la lettre proposée a fait avancer le joueur : on peut continuer sans erreur supplémentaire. Dans ce cas, il ne faut pas oublier de mettre la variable *etat* à jour en lui affectant le nouvel état.

   Pour tirer un mot au hasard, il suffit de récupérer la liste de tous les mots et de tirer un numéro de case au hasard avec la fonction *randint*. N'oubliez pas de faire un

```
from random import randint

```

Voici un exemple de partie :

```
>>> pendu_version1("littre.txt")

_ _ _ _ _ _ _ _
Vous pouvez faire encore 8 erreurs.
Entrez une lettre, suivie d'un saut de ligne : e
Dommage...

_ _ _ _ _ _ _ _
Vous pouvez faire encore 7 erreurs.
Entrez une lettre, suivie d'un saut de ligne : a
Bravo !

_ A _ _ A _ _ _
Vous pouvez faire encore 7 erreurs.
Entrez une lettre, suivie d'un saut de ligne : l
Dommage...

_ A _ _ A _ _ _
Vous pouvez faire encore 6 erreurs.
Entrez une lettre, suivie d'un saut de ligne :

...
...

Perdu...
Le mot secret était 'RAMPANT'.

```

4. Écrivez la procédure pendu_version1. Cette procédure prend en argument le nombre maximal d'erreurs autorisées.
Note : il faudra bien entendu faire une boucle *while* et utiliser :

   - des *print* pour faire les affichages
   - un *input* pour récupérer la lettre proposée par le joueur :

   Deuxième version :gestion des accents
   La partie précédente ne prenait pas les accents en compte : si la lettre proposée est "e", il faut en fait vérifier si le mot contient des "é", "è", "ê" ou "ë". Par ailleurs, le français contient deux lettre doubles : le "æ" et le "œ. Lorsque la lettre proposée est "e", il faut donc aussi vérifier si le mot contient des "æ" ou "œ".
   Pour gérer cela, nous allons utiliser un dictionnaire qui associe aux caractères ASCII les caractères accentués correspondants.
   variantes =
   'A' : 'AÀÄÂÆ',
   'C' : 'CÇ',
   'E' : 'EÊÈÉËÆŒ',
   'I' : 'IÎÏ',
   'O' : 'OÔÖŒ',
   'U' : 'UÙÜÛ'
   Notez en particulier que les lettres doubles ont deux lettres correspondantes.
   Remarque : N'oubliez pas qu'il est possible de tester si un caractère apparait dans une chaine avec le mot clé in :

```
>>> "c" in "coco"
True
>>> "t" in "coco"
False
>>> "o" in "coco"
True
```

   Écrivez une fonction nouvel_etat_version2 qui prend les accents et lettres doubles en compte.
   Pour cela, il suffit de modifier la fonction nouvel_etat en modifiant le test qui vérifie si le ième caractère de mot est égal à c en un test qui vérifie si le ième caractère du mot apparait dans la chaine contenant toutes les variantes du c.
   Attention : certains caractères n'ont aucune variante et n'apparaissent donc pas dans le dictionnaire variantes. Il faudra utiliser la méthode variantes.get(...) pour gérer ces cas.
   L'exécution de :

```
mot = "DÉSŒUVRER"
etat = "D____VR_R"
```

```
etat2 = nouvel_etat(mot, etat, "e")
print("mot :", etat2)
```

devra afficher :
mot : DÉ_Œ_VRER

6. Programmez maintenant la procédure pendu_version2 similaire à pendu_version1, mais qui prend les accents et lettres doubles en compte.
   Troisième Version : mots encore disponibles :
   Nous allons améliorer la version précédente du jeu en ajoutant :
   - un affichage qui indique combien de mots du dictionnaire contiennent les lettres devinées,
   - un test qui vérifie que le joueur n'a pas déjà proposé une lettre.

```
>>> pendu_version3("littre.txt")

_ _ _ _ _ _ _ _
Vous pouvez faire encore 8 erreurs.
Il y a encore 11595 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : e
Bravo !

É _ _ _ _ _ _ E
Vous pouvez faire encore 8 erreurs.
Il y a encore 196 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : a
Bravo !

É _ _ _ _ A _ E
Vous pouvez faire encore 8 erreurs.
Il y a encore 46 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : l
Dommage...

É _ _ _ _ A _ E
Vous pouvez faire encore 7 erreurs.
Il y a encore 46 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : p
Bravo !

É _ _ _ P A _ E
Vous pouvez faire encore 7 erreurs.
Il y a encore 5 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : e
Vous avez déjà proposé cette lettre...

É _ _ _ P A _ E
Vous pouvez faire encore 7 erreurs.
Il y a encore 5 mots possibles...
Entrez une lettre, suivie d'un saut de ligne : e
Vous avez déjà proposé cette lettre...
```

7. Commencez par écrire la fonction :

```
def mots_longueur(liste_mots, n):
    """fonction qui renvoie la liste des chaines d'une longueur donnée dans une
    liste

    préconditions :
      - liste_mots, de type liste de chaines de caractères
      - n, de type entier

    postconditions : liste de chaines de caractères: tous les éléments de mots qui ont la
    longueur n
    """
```

Cette fonction vous permettra de savoir combien de mots sont possible au début d'une partie de pendu : il suffit de garder les mots de même taille que le mot secret.

8. Écrivez la procédure pendu_version3 et testez-la. N'oubliez pas de :

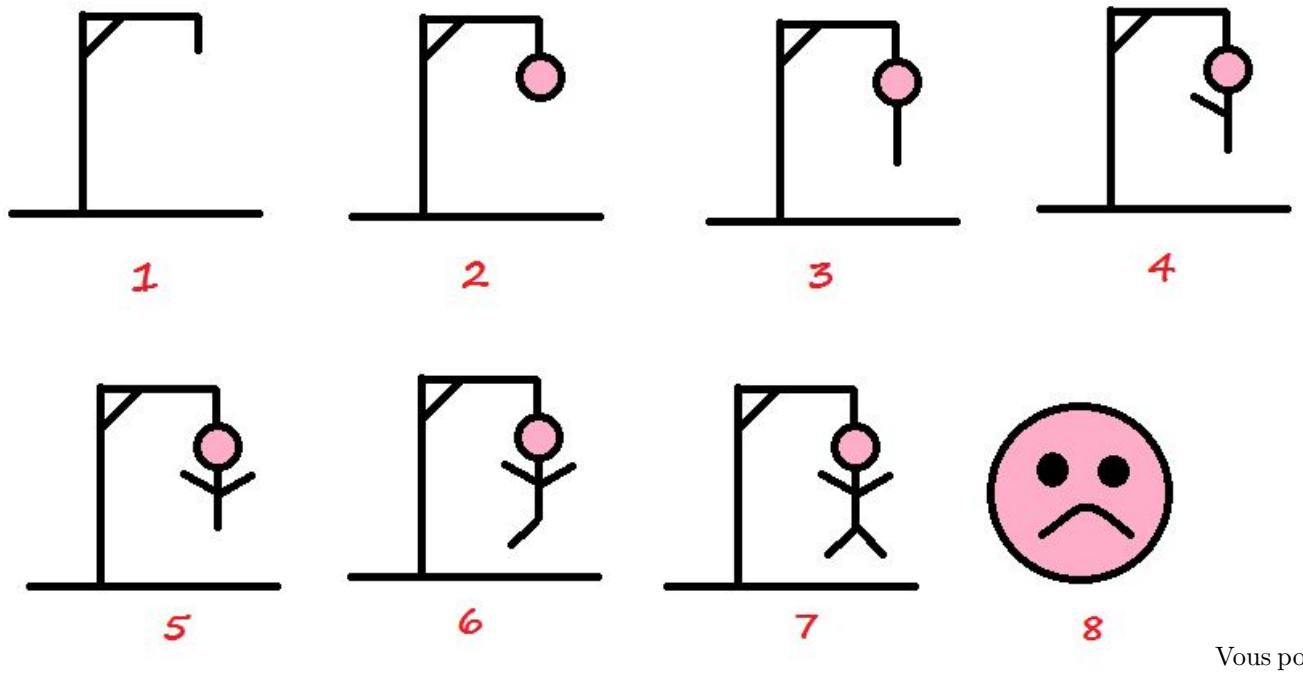
   - tester si les lettres ont déjà été proposées,

- afficher le nombre de mots possibles restant dans le dictionnaire.

Note : pour pouvoir compter le nombre de mots restants qui sont compatibles avec les lettres devinées, il est conseillé d'écrire des fonctions auxiliaires :

```
def mots_sans_lettre(liste_mots, c):
    """fonction qui renvoie la liste des éléments d'une liste qui ne contiennent pas un
    caractère donné

    préconditions :
      - liste_mots, de type liste de chaines de caractères
      - c, de type caractère

    postconditions : liste de chaine de caractères
    """
    ...


def mots_avec_lettre(liste_mots, c):
    """fonction qui renvoie la liste des éléments d'une liste qui
    contiennent un caractère donné

    préconditions :
      - liste_mots, de type liste de chaines de caractères
      - c, de type caractère

    postconditions : liste de chaine de caractères
    """
```

Ces fonctions vous permettront de mettre à jour la liste des mots possibles suivant que la lettre proposée était présente ou pas dans le mot secret. Aspect graphique :
Vous disposez des images :



Vous pourrez afficher ces images au fur et à mesure du jeu.