Chapitre 7

Jeux de lettres

7.1. Nouveaux thèmes abordés dans ce chapitre

- fichiers
- chaînes de caractères (string)
- ensembles (set)

7.2. Fichiers

On divise habituellement les fichiers en deux catégories.

Les fichiers binaires ne contiennent pas exclusivement du texte. Ils ne peuvent être « lus » que par des logiciels spécialisés. Une image JPEG, un fichier PDF, ou un MP3 sont des fichiers binaires.

Les fichiers textes sont « compréhensibles » par un humain : ils contiennent du texte (lettres, ponctuations, nombres, ...). Leur contenu est souvent organisé en lignes et on peut les lire avec des éditeurs de textes simples comme *NotePad++*, *WordPad*, etc.

Ici, nous allons exclusivement nous intéresser aux fichiers textes.

Pour jouer à des jeux de lettres, il nous faut d'abord constituer une liste de mots. On pourrait mettre tous ces mots dans une liste dans le corps du programme, mais il y a plusieurs inconvénients :

- on verra les mots en ouvrant le programme, ce qui gâchera le plaisir de les trouver ;
- si l'on a beaucoup de mots dans notre stock, cela nuira à la lecture du programme ;
- l'échange de données avec d'autres programmes (peut-être écrits dans d'autres langages) est tout simplement impossible, puisque ces données font partie du programme lui-même.

D'une manière générale, quand la quantité de données est importante, mieux vaut stocker ces dernières dans un fichier.

Un fichier est une ressource. L'accès aux ressources doit pouvoir être partagé entre les différents programmes qui peuvent en avoir besoin. Cela veut dire qu'il est nécessaire d'*acquérir* cette ressource avant de s'en servir, puis de la *libérer* après usage. Pour un fichier, cela se fait en l'ouvrant (`open`), puis en le fermant (`close`) :

```
# Ouverture d'un fichier en lecture:
fichier = open("/etc/passwd", "r")
...
# Fermeture du fichier
fichier.close()
```

Un fichier ouvert doit toujours être fermé après usage !

Le premier paramètre de la fonction `open` est le nom du fichier à manipuler, avec son chemin d'accès. Le second paramètre est le mode dans lequel on souhaite ouvrir le fichier. Le mode « r » (pour *read*) indique que nous allons ouvrir le fichier pour lire des données dedans. Tant qu'un fichier est ouvert, il ne peut pas être utilisé par un autre programme.

On peut aussi ouvrir un fichier pour écrire en utilisant le mode « w » (pour *write*) :

```
# Ouverture d'un fichier en écriture:
fichier = open("/etc/passwd", "w")
```

Mais attention ! Dans ce cas, si le fichier existe déjà, son contenu est écrasé ! Si vous voulez plutôt ajouter des données à la fin d'un fichier, il faut utiliser le mode « a » (pour *append*):

```
# Ouverture d'un fichier en ajout:
fichier = open("/etc/passwd", "a")
```

7.2.1. Lire dans un fichier

La lecture dans un fichier texte se fait de manière séquentielle, une ligne après l'autre (c'est la raison pour laquelle on parle de fichier à accès séquentiel). Il n'est pas possible de lire une ligne au milieu du fichier avant d'avoir lu toutes celles placées avant.

Il est donc nécessaire de connaître la structure du fichier pour pouvoir le lire correctement. Par exemple, dans les fichiers que nous allons utiliser, chaque ligne du fichier comportera un seul mot.

La méthode `readlines` lit toutes les lignes du fichier et les met dans une liste. Elle est utile si on souhaite lire tout le fichier d'un coup :

```
touteslelignes = fichier.readlines()
```

La méthode `readline` (au singulier) lit la ligne suivante dans le fichier.

```
ligne = fichier.readline()
```

Enfin, pour traiter l'une après l'autre toutes les lignes d'un fichier, on peut utiliser une boucle `for` :

```
for ligne in fichier:
    ...
```

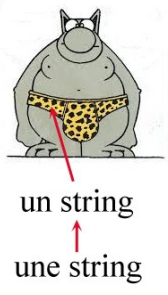
7.2.2. Écrire dans un fichier

Écrire dans un fichier se fait très simplement en utilisant la méthode `write()`.

```
fichier.write("ce qu'il faut écrire\n")
```

Les symboles **\n** indiquent que l'on saute à la ligne suivante.

Ces données sont enregistrées dans le fichier les unes à la suite des autres. Chaque nouvel appel de `write()` continue l'écriture à la suite de ce qui est déjà enregistré.



7.3. Chaînes de caractères

Les chaînes de caractères forment la classe 'str' (pour *string*). Une chaîne de caractères est une suite de symboles placées entre guillemets ("salut") ou entre apostrophes ('salut'). L'intérêt d'avoir ces deux délimiteurs est de pouvoir écrire des chaînes contenant des apostrophes ("aujourd'hui") ou des guillemets ('Pressez "enter" pour continuer').

Les méthodes de la classe 'str' sont nombreuses. Elles sont détaillées dans la documentation officielle de Python à cette adresse :

http://docs.python.org/dev/library/stdtypes.html#string-methods

Nous ne verrons ici que les plus utiles.

Prenons comme exemple de chaîne de caractères :

```
chaine = "ma Chaîne de Caractères"
```

7.3.1. Accès à des sous-chaînes

La fonction intégrée `len`, que l'on a déjà rencontrée, permet de connaître la longueur (*length* en anglais) d'une chaîne de caractères :

```
len(chaine)
```

donnera comme résultat 23. Les positions de la string `chaine` vont donc de 0 à 22.
On peut accéder à des lettres isolées de la chaîne ou à des sous-chaînes d'une manière analogue à celle utilisée avec les listes.

```
print(chaine[0])
```

donnera le premier caractère de la chaîne ; dans notre exemple '`m`'.

```
print(chaine[3:])
```

donne la sous-chaîne commençant à la troisième position, à savoir '`Chaîne de Caractères`'. (la numérotation des places commence à 0, comme pour les listes). Inversement,

```
print(chaine[:9])
```

donne la sous-chaîne se terminant à la huitième (!) position, à savoir '`ma Chaîne`'. En combinant les deux :

```
print(chaine[3:9])
```

on obtiendra la sous-chaîne '`Chaîne`'.
Il est possible de transformer une chaîne de caractères en une liste de « mots ». Il faut juste indiquer ce qui joue le rôle de séparateur. Par exemple, si le séparateur est l'espace,

```
chaine.split(' ')
```

produira comme résultat : `['ma', 'Chaîne', 'de', 'Caractères']`.
Pour avoir la liste des caractères de la chaîne, rien de plus facile :

```
list(chaine)
```

retournera la liste `['m', 'a', ' ', 'C', 'h', 'a', 'î', 'n', 'e', ' ', 'd', 'e', ' ', 'C', 'a', 'r', 'a', 'c', 't', 'è', 'r', 'e', 's']`.

On peut parcourir tous les caractères d'une chaîne avec l'instruction `for` :

```
for ch in chaine:
    print(ch,end=" ")
```

produira comme résultat : `m a  C h a î n e  d e  C a r a c t è r e s`
On peut savoir si une sous-chaîne se trouve dans une chaîne :

```
'de' in chaine
```

donnera `True`. Il est aussi possible de connaître la position où commence une sous-chaîne :

```
print(chaine.index('de'))
```

donnera 10, car « de » apparaît pour la première fois à la position numéro 10. Si la sous-chaîne

recherchée n'existe pas, le programme renvoie un message d'erreur :

```
print(chaine.index('u'))
```

produit ValueError: substring not found

7.3.2. Mises en forme

Pour tout écrire en majuscules :

```
print(chaine.upper())
```

donnera comme résultat : 'MA CHAÎNE DE CARACTÈRES'
Pour tout écrire en minuscules :

```
print(chaine.lower())
```

donnera comme résultat : 'ma chaîne de caractères'
Pour écrire la première lettre en majuscules et les autres en minuscules :

```
print(chaine.capitalize())
```

donnera comme résultat : 'Ma chaîne de caractères'

7.3.3. Manipulations des strings

Attention ! Contrairement aux listes, les chaînes de caractères ne sont pas modifiables. Ainsi, une affectation du type :

```
chaine[0]="T"
```

donnera le message d'erreur : TypeError: 'str' object does not support item assignment
On peut cependant faire des opérations sur des chaînes de caractères. On peut par exemple enlever des caractères qui se trouvent en début ou en fin de chaîne avec la méthode `strip()` :

```
print('grand-maman'.strip('-agmn'))
```

donnera comme résultat : 'rand'
Si la fonction `strip()` n'a pas d'argument, on enlèvera par défaut des espaces. Si on veut enlever des caractères en début de chaîne, il existe la méthode `lstrip()` ; si on veut enlever des caractères en fin de chaîne on utilisera la méthode `rstrip()` :

```
print('grand-maman'.lstrip('-agmn'))
```

donnera comme résultat 'rand-maman', tandis que

```
print('grand-maman'.rstrip('-agmn'))
```

donnera comme résultat 'grand'

On peut concaténer deux chaînes avec l'opérateur + :

```
chaine1 = "ma Chaîne"
chaine2 = " de Caractères"
chaine = chaine1 + chaine2
print(chaine)
```

donnera 'ma Chaîne de Caractères'.

On peut aussi répéter une chaîne de caractères plusieurs fois :

```
chaine = "salut "
chaine = chaine*4
print(chaine)
```

donnera : 'salut salut salut salut '

7.4. Le pendu - règles du jeu

Le jeu du pendu consiste à retrouver un mot le plus vite possible (avant que le dessin du pendu soit terminé) en proposant des lettres. Si la lettre appartient au mot, elle est écrite aux bons emplacements, sinon on continue de dessiner le pendu.

Nous allons commencer par une version non graphique où il s'agira de deviner un mot avec le moins d'essais possible. Nous verrons une version graphique au chapitre 8.

Exemple de partie

```
------
Entrez une lettre ou '?' pour abandonner : E
------
Entrez une lettre ou '?' pour abandonner : U
U---U-
Entrez une lettre ou '?' pour abandonner : A
U-A-U-
Entrez une lettre ou '?' pour abandonner : N
U-ANU-
Entrez une lettre ou '?' pour abandonner : S
U-ANUS
Entrez une lettre ou '?' pour abandonner : R
URANUS
Bravo ! Le mot URANUS a été trouvé en 6 coups
```

Remarque : les lettres en gras ont été entrées au clavier par le joueur.

7.5. Code du programme

pendu.py

```
# Le jeu du pendu

from random import choice

fichier = open("liste_mots.txt", "r")
liste_mots = fichier.readlines()    # met tous les mots du fichiers dans une liste
mot = choice(liste_mots)            # prend au hasard un mot dans la liste
mot = mot.rstrip()                  # supprime le caractère "saut à la ligne"
fichier.close()

mot_devine = "-" * len(mot)
print(mot_devine)
nbr_essais = 0

while mot_devine != mot:
    lettre = input("Entrez une lettre ou '?' pour abandonner : ")
    lettre = lettre[0]  # évite des erreurs si un mot est entré au lieu d'une lettre
    if lettre == '?':
        print('Le mot était',mot)
        break
    lettre = lettre.upper()
    for i in range(len(mot)):
        if lettre == mot[i]:
            mot_devine = mot_devine[:i] + lettre + mot_devine[i+1:]
    print(mot_devine)
    nbr_essais += 1

if mot == mot_devine:
    print('Bravo ! Le mot',mot,'a été trouvé en',nbr_essais,'coups')
```

7.6. Analyse du programme

```
from random import choice
```

La fonction `choice` permet de choisir un élément au hasard dans une liste. Ici, il s'agira de choisir un mot.

```
fichier = open("liste_mots.txt", "r")
liste_mots = fichier.readlines()    # met tous les mots du fichiers dans une liste
mot = choice(liste_mots)            # prend au hasard un mot dans la liste
mot = mot.rstrip()                  # supprime le caractère "saut à la ligne"
fichier.close()
```

Tous les mots se trouvent dans le fichier `liste_mots.txt`. Ce fichier doit se trouver au même niveau que le programme.

```
mot_devine = "-" * len(mot)
print(mot_devine)
nbr_essais = 0
```

Au début, on écrit une suite de tirets pour indiquer la longueur du mot. Ces tirets seront remplacés par des lettres au fur et à mesure que le joueur les découvrira.

```
while mot_devine != mot:
```

La boucle principale se répète tant que l'on n'a pas découvert le mot.

```
    lettre = input("Entrez une lettre ou '?' pour abandonner : ")
    lettre = lettre[0]  # évite des erreurs si un mot est entré au lieu d'une lettre
    if lettre == '?':
        print('Le mot était',mot)
        break
```

On demande une lettre au joueur. Il peut en fait entrer n'importe quoi, mais cela ne le fera pas avancer dans sa quête... S'il entre un point d'interrogation, alors on donnera le mot à trouver et on sortira de la boucle par l'instruction `break`.

```
    lettre = lettre.upper()
```

Comme dans le fichier contenant notre stock de mots, tous les mots sont écrits en majuscules, on s'assure que la lettre entrée est bien une majuscule.

```
    for i in range(len(mot)):
        if lettre == mot[i]:
            mot_devine = mot_devine[:i] + lettre + mot_devine[i+1:]
    print(mot_devine)
    nbr_essais += 1
```

La variable `i` parcourt toutes les positions du mot. Si la lettre proposée se trouve à la position i, alors on remplace le tiret par cette lettre (3ème ligne du bout de code ci-dessus). On écrit ensuite le nouveau résultat intermédiaire et on incrémente le nombre d'essais.

```
if mot == mot_devine:
    print('Bravo ! Le mot',mot,'a été trouvé en',nbr_essais,'coups')
```

Après être sorti de la boucle, on écrit un petit message de félicitation si le bon mot a été découvert.

Exercice 7.1

Imaginons que la fonction `choice` n'existe pas. Comment choisir un mot au hasard dans la liste `liste_mots` ?

Exercice 7.2

Réutilisez le fichier `liste_mots.txt` pour un autre jeu qui consistera à retrouver un mot, l'ordinateur ayant écrit ses lettres par ordre alphabétique. Par exemple :

```
AEIIMMNPRT
Entrez l'anagramme : imprimante
Bravo !
```

7.7. Ensembles

Les ensembles en Python sont les mêmes ensembles qu'en mathématiques. On peut leur appliquer les mêmes opérations : union, intersection, différence symétrique, différence. On peut aussi tester si un ensemble est inclus dans un autre.

Voici un programme qui résume toutes ces opérations.

ensembles.py

```
ens1 = set([1, 2, 3, 4, 5, 6])
ens2 = set([5, 6, 7, 8])
ens3 = set([2, 4, 6])

# union
print(ens1 | ens2)
# intersection
print(ens1 & ens2)
# différence symétrique
print(ens1 ^ ens2)
# différence
print(ens1 - ens2)

# inclusion
print(ens2.issubset(ens1))
print(ens3.issubset(ens1))

# transformation en une liste
liste1 = list(ens1)
print(liste1)
```

Voici le résultat de ce programme :

```
{1, 2, 3, 4, 5, 6, 7, 8}
{5, 6}
{1, 2, 3, 4, 7, 8}
{1, 2, 3, 4}
False
True
[1, 2, 3, 4, 5, 6]
```

Remarquez que dans un ensemble, chaque élément n'apparaît qu'une seule fois.

Notons enfin que l'on peut facilement convertir une chaîne de caractères en un ensemble :

```
set('carnaval')
```

retournera l'ensemble : `{'l', 'n', 'a', 'c', 'v', 'r'}`

Exercice 7.3

Écrivez un programme qui remplace les voyelles d'un texte (éventuellement accentuées) par une étoile. Par exemple « `Il était une fois` » deviendra « `*l *t**t *n* f**s` ».

Exercice 7.4

Pour toutes les questions ci-dessous, utilisez le fichier dico.txt.

dico.txt

a) Calculez le pourcentage de mots français où la seule voyelle est le « e » (il peut y en avoir plusieurs dans le mot).
Par exemple : exemple, telle, égrener.

b) Calculez le pourcentage de mots français où deux lettres identiques se suivent.
Par exemple : femme, panne, créer.

c) Affichez les mots de moins de 10 lettres ayant comme lettre centrale un « w ». Le mot doit avoir un nombre impair de lettres.
Par exemple : edelweiss, kiwis.

d) Affichez la liste des mots contenant la suite de lettres « dile ».
Par exemple : crocodile, prédilection.

e) Affichez les mots de moins de 5 lettres qui commencent et finissent par la même lettre.
Par exemple : aura, croc, dard.

f) Affichez tous les mots palindromes.
Par exemple : serres, kayak, ressasser.

g) Affichez tous les mots anacycliques : un mot lu de droite à gauche donne un autre mot.
Par exemple : les – sel, bons – snob.

h) Affichez tous les mots composés de deux séquences de lettres qui se répètent.
Par exemple : papa, chercher.

Exercice 7.5

Pour toutes les questions ci-dessous, utilisez le fichier dico.txt.

Dans le jeu du pendu, les mots les plus difficiles à trouver sont ceux qui ont peu de voyelles et des consonnes rares. Voici les lettres de la plus fréquente à la moins fréquente, en français :

Lettres courantes : `E A I S T N R U L O D M P`
Lettres rares : `C V Q G B F J H Z X Y K W`

dico.txt

a) Affichez la liste des mots n'ayant que des voyelles et des lettres rares.

b) Affichez la liste des mots de plus de 15 lettres n'ayant que des lettres courantes.

c) Affichez la liste des mots n'ayant aucune des lettres `E A I S T N R U`.

d) Affichez la liste des mots ayant exactement deux voyelles et plus de 9 caractères (tiret compris). Par exemple : transports, check-list.

e) Affichez la liste des mots commençant par au moins 4 consonnes consécutives (tiret non autorisé).

Exercice 7.6

L'argot javanais, apparu en France dans la dernière moitié du 19ème siècle, consiste à intercaler dans les mots la syllabe « av » entre une consonne et une voyelle. Nous n'utiliserons ici que les règles simplifiées (*) :

- On rajoute « av » après chaque consonne (ou groupe de consonnes comme par exemple ch, cl, ph, tr,…) d'un mot.

- Si le mot commence par une voyelle, on ajoute « av » devant cette voyelle.
- On ne rajoute jamais « av » après la consonne finale d'un mot.

Écrivez un programme qui traduit un texte français en javanais. Par exemple,

```
Je vais acheter des allumettes au supermarché.
```

deviendra

```
Jave vavais avachavetaver daves avallavumavettaves avau savupavermavarchavé.
```

(*) Pour les règles complètes, voir https://fr.wikipedia.org/wiki/Javanais_(argot)

7.8. Le mot le plus long – règles du jeu

Les règles sont élémentaires : on entre une série de lettres et l'ordinateur essaie de trouver le plus long mot possible avec ces lettres. Il peut y avoir plusieurs fois la même lettre dans la série.

Exemple de partie

```
entrez votre tirage : urtefaaijg
10 lettres
Le script a mis 0.184 s pour trouver des mots de 9 lettres
['fatiguera', 'jaugerait']
```

Exercice 7.7

Nous allons utiliser les ensembles dans l'implémentation du jeu « Le mot le plus long ». Mais avant cela, nous aurons besoin d'un lexique ordonné, que nous allons élaborer à partir du fichier texte « dico.txt », qui contient une liste de 323'467 mots classés par ordre alphabétique.

Utilisez « dico.txt » pour faire un dictionnaire (au sens Python du terme, voir chapitre 6), où la clé sera la longueur du mot, et où la valeur sera la liste alphabétique des mots de cette longueur. Si des mots contiennent un ou des tirets, il faudra supprimer les tirets ! Ainsi, le mot « porte-avions » deviendra « porteavions ». Idem avec les apostrophes.

7.9. Code du programme

motlepluslong.py

```python
# Le mot le plus long

from time import time

def dictionnaire_ordonne():
    # on lit le fichier et on range les mots alphabétiquement selon leur longueur
    fichier = open("dico.txt", "r")
    dict_ord = {}
    for longueur in range(26):
        dict_ord[longueur+1] = []
    mot = fichier.readline()
    while mot != '':
        mot = mot.strip('\n')
        if '-' in mot:
            mot = mot.replace('-','')
        if "'" in mot:
            mot = mot.replace("'",'')
        dict_ord[len(mot)].append(mot)
        mot = fichier.readline()
    fichier.close()
    return dict_ord

def lettres_multiples_ok(mot,tirage):
    # teste si chaque lettre figure suffisamment de fois dans le tirage
    for lettre in mot:
        if lettre in tirage:
            tirage.remove(lettre)
```

```
        else:
            return False
    return True

def trouver_plus_long_mot(dico, tirage):
    longueur_mot = len(tirage)
    solution= []
    set_tirage = set(tirage)
    while longueur_mot > 0:
        for mot in dico[longueur_mot]:
            if set(mot).issubset(set_tirage):
                # les lettres du mot sont un sous-ensemble du tirage
                tirage_test = list(tirage)
                if lettres_multiples_ok(mot,tirage_test):
                    solution.append(mot)
        if solution != [] or longueur_mot==1:
            return solution, longueur_mot
        longueur_mot -= 1


dico = dictionnaire_ordonne()
jouer = True
while jouer:
    tirage = input('Entrez votre tirage : ').lower()
    print(len(tirage),'lettres')
    t0 = time()
    solution, longueur = trouver_plus_long_mot(dico, tirage)
    if solution == []:
        print('Pas de mot trouvé !')
    else:
        t1 = time()-t0
        print('Le script a mis','{0:.3f}'.format(t1),
              's pour trouver des mots de',longueur,'lettres')
        print(solution)
    rejouer = input('Rejouer ? (o/n) : ')
    jouer = (rejouer == "o")
```

7.10. Analyse du programme

```
from time import time
```

C'est dans le module `time` que se trouve la fonction... `time()`, qui renvoie un nombre réel correspondant au nombre de secondes écoulées depuis le 1er janvier 1970 à 00:00:00. Nous allons utiliser cette fonction pour chronométrer la durée de la recherche du mot le plus long ; nous verrons comment un peu plus loin.

```
def dictionnaire_ordonne():
    # on lit le fichier et on range les mots alphabétiquement selon leur longueur
    fichier = open("dico.txt", "r")
    dict_ord = {}
    for longueur in range(25):
        dict_ord[longueur+1] = []
```

La fonction `dictionnaire_ordonne` était l'objet de l'exercice 7.5. Le dictionnaire est « découpé » en 25 listes, la liste k comprenant tous les mots de k lettres classés alphabétiquement, avec k allant de 1 à 25. Rappelons que le plus long mot de la langue française comporte 25 lettres : « anticonstitutionnellement ».

```
    mot = fichier.readline()
    while mot != '':
        mot = mot.strip('\n')
        if '-' in mot:
            mot = mot.replace('-','')
        if "'" in mot:
            mot = mot.replace("'",'')
        dict_ord[len(mot)].append(mot)
        mot = fichier.readline()
```

On lit tous les mots du fichier, ligne par ligne (il y a un mot par ligne dans le fichier) ; on supprime le caractère saut de ligne :

```
        mot = mot.strip('\n')
```

et les tirets qui se trouveraient dans le mot (idem pour les apostrophes)

```
            mot = mot.replace('-','')
```

puis on ajoute à la liste correspondante ce mot.

```
        dict_ord[len(mot)].append(mot)
```

Le fichier est finalement fermé et le dictionnaire ordonné est retourné.

```
    fichier.close()
    return dict_ord
```

La fonction `lettres_multiples_ok()` teste si chaque lettre figure suffisamment de fois dans le tirage. L'idée est simple : les lettres du mot sont enlevées une à une de la liste tirage. Si la lettre courante du mot n'est pas dans la liste tirage, le mot ne peut pas être écrit et la fonction renvoie `False`.

```
def lettres_multiples_ok(mot,tirage):
    # teste si chaque lettre figure suffisamment de fois dans le tirage
    for lettre in mot:
        if lettre in tirage:
            tirage.remove(lettre)
        else:
            return False
    return True
```

La fonction `trouver_plus_long_mot()` parcourt le dictionnaire ordonné, en commençant par la liste des plus longs mots possibles.

```
def trouver_plus_long_mot(dico, tirage):
    longueur_mot = len(tirage)
    solution= []
    set_tirage = set(tirage)
    while longueur_mot > 0:
        for mot in dico[longueur_mot]:
```

La ligne qui suit fait un prétraitement : avant de voir si les lettres sont en suffisance, on regarde d'abord si toutes les lettres du mot sont dans le tirage.

```
            if set(mot).issubset(set_tirage):
                # les lettres du mot sont un sous-ensemble du tirage
                tirage_test = list(tirage)
                if lettres_multiples_ok(mot,tirage_test):
                    solution.append(mot)
```

Si une solution a été trouvée, on la retourne. Sinon, on essaie avec la liste des mots ayant une lettre de moins.

```
        if solution != [] or longueur_mot==1:
            return solution, longueur_mot
        longueur_mot -= 1
```

Le programme principal ne présente pas de difficultés.

```
dico = dictionnaire_ordonne()
jouer = True
while jouer:
    tirage = input('Entrez votre tirage : ').lower()
    print(len(tirage),'lettres')
```

```
    t0 = time()
    solution, longueur = trouver_plus_long_mot(dico, tirage)
    if solution == []:
        print('Pas de mot trouvé !')
    else:
        t1 = time()-t0
        print('Le script a mis','{0:.3f}'.format(t1),
              's pour trouver des mots de',longueur,'lettres')
        print(solution)
    rejouer = input('Rejouer ? (o/n) : ')
    jouer = (rejouer == "o")
```

Pour chronométrer le temps de calcul, on enregistre « l'heure » dans la variable `t0` avant de lancer la recherche du mot le plus long, puis, juste après la recherche, on enregistre la différence entre « la nouvelle heure » et `t0`.

```
        t1 = time()-t0
```

Exercice 7.8

Est-il possible de se passer des ensembles dans le programme du § 7.9 ? Essayez de supprimer du programme la ligne :

```
        if set(mot).issubset(set_tirage)
```

ainsi que les lignes ayant un rapport avec ce prétraitement.
Votre programme est-il plus rapide ? Pour le savoir, testez-le avec la fonction `time()`.

Exercice 7.9

Modifiez le programme du § 7.9 pour qu'il écrive toutes les anagrammes des mots d'une longueur donnée par l'utilisateur.
Par exemple :

```
Longueur des mots : 5
abats : ['basat', 'batas']
abeti : ['batie', 'beait']
abime : ['amibe', 'iambe']
...
zoner : ['ornez']
```

Exercice 7.10

Au Scrabble (version française), les valeurs des lettres sont les suivantes :

• A, E, I, L, N, O, R, S, T, U	1 point
• D, G, M	2 points
• B, C, P	3 points
• F, H, V	4 points
• J, Q	8 points
• K, W, X, Y, Z	10 points

Modifiez le programme du § 7.9 pour qu'il écrive les mots qui rapporteraient le plus de points au Scrabble, en utilisant les lettres données par l'utilisateur.
Par exemple :

```
Entrez votre tirage : deefirrsv
['feviers', 'fevrier', 'fievres'] : 13 points au Scrabble
```

Exercice 7.11

La **saisie intuitive** est une technologie conçue afin de simplifier la saisie de texte sur les claviers téléphoniques.

Là où une combinaison de touches correspond, dans le système traditionnel, à un et un seul mot, le système de saisie intuitive doit faire face à un problème : la pluralité des combinaisons de lettres correspondant à une même saisie. Par exemple, le nombre 7243 correspond à $4 \times 3 \times 3 \times 3 = 108$ combinaisons de lettres possibles. Mais seules quelques-unes correspondront (peut-être) à un mot français.

Écrivez un programme permettant de trouver tous les mots correspondant à un nombre donné.

```
Entrez le code votre mot (p. ex. 665587783) : 7243
Le script a mis 0.015 s pour trouver :
['page', 'paie', 'rage', 'raid', 'raie', 'sage', 'scie']
```

Exercice 7.12

Le jeu « Motus » repose sur la recherche de mots d'un nombre fixé de lettres (entre 7 et 10). Un candidat propose un mot qui doit contenir le bon nombre de lettres et être correctement orthographié, sinon il est refusé. Le mot apparaît alors sur une grille : les lettres bien placées sont colorées en rouge, les lettres présentes mais mal placées sont en jaune.

C	A	S	T	O	R
C	I	N	E	M	A
C	Y	P	R	E	S
C	I	T	R	O	N

Une lettre ne peut être colorée au maximum que le nombre de fois que cette lettre apparaît dans le mot. Par exemple, si le mot cherché est CABLES et que l'on propose CABANE, on aura la coloration :

C	A	B	A	N	E

et non pas

C	A	B	A	N	E

car il n'y a qu'un « A » dans CABLES.

Mots non valables

- les noms propres ;
- les mots mal orthographiés ;
- les mots composés (chewing-gum, week-end, ...) ;
- les verbes conjugués (seuls les infinitifs et les participes passés et présents sont acceptés) ;
- les mots qui ne commencent pas par la première lettre indiquée ;
- les mots qui ne respectent pas le nombre de lettres donné.

Votre programme devra :

- prendre un mot au hasard dans le fichier « dico.txt ». Ce mot devra avoir entre 7 et 10 lettres. Contrairement aux règles originales, on permettra les formes conjuguées ;
- indiquer le nombre de lettres du mot à trouver et donner la première lettre ;
- écrire le numéro de l'essai ;
- écrire sous la proposition du joueur :
  - une lettre majuscule si elle est bien placée ;
  - une lettre minuscule si elle est mal placée ;
  - un point si la lettre n'est pas dans le mot, ou si elle a moins d'occurrences ;
  - un message d'erreur si le mot est non valide.

Exemples de partie

```
Vous cherchez un mot de 7 lettres. Vous avez 6 essais.
  O......
1 orageux
  OrA.e..
2 opaline
  O.A.I.e
3 otaries
  OTARIES
Bravo !

Vous cherchez un mot de 8 lettres. Vous avez 6 essais.
  F.......
1 fouinant
  FOUi.a.t
2 febriles
  Fe..i..s
3 fouettes
  FOUet..s
4 foutasse
Pas dans le dictionnaire
5 foutaise
  FOUTAISE
Bravo !

Vous cherchez un mot de 7 lettres. Vous avez 6 essais.
  T......
1 trousse
  T.o.s..
2 passeur
Première lettre non valide
3 traceur
  T.aC...
4 tamisees
Longueur non valide
5 tapisse
  TA..s..
6 tapoter
  TA.o...
Désolé, la réponse était TANCONS
```

Exercice 7.13

Deux joueurs vont s'affronter en lançant plusieurs fois de suite une pièce de monnaie. Avant la partie, chacun des joueurs choisit une séquence de trois résultats (chaque résultat étant Pile ou Face). Puis, on lance la pièce de monnaie autant de fois qu'il le faut (en notant à chaque fois le résultat) jusqu'à ce qu'une des deux séquences apparaisse.

Par exemple, avant de commencer à lancer la pièce, le joueur 1 choisit la séquence *PFP* (Pile-Face-Pile) et le joueur 2 choisit la séquence *FPP*. On lance ensuite la pièce plusieurs fois de suite et voici les issues obtenues successivement :

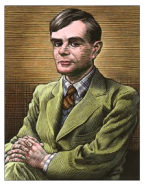
F P F F P F F ***F P P***

Cette partie a duré 10 lancers et la première des deux séquences sortie est celle du joueur 2, qui est alors déclaré vainqueur.

A priori, les deux joueurs devraient avoir la même chance de gagner… en tout cas, c'est ce que l'intuition nous laisse penser. Ce n'est pourtant pas le cas. Si vous connaissez la séquence choisie par l'adversaire, vous pourrez définir la vôtre et gagner plus souvent que votre adversaire.

En simulant des milliers de parties, trouvez la meilleure séquence du joueur 2 à opposer à chacune des 8 séquences possibles du joueur 1.

Déterminez ensuite la stratégie gagnante.



7.11. Le défi Turing

Avec vos connaissances actuelles en Python, vous pourrez résoudre plus ou moins facilement presque tous les exercices du défi Turing : www.apprendre-en-ligne.net/turing/

Le défi Turing est une série d'énigmes mathématiques qui pourront difficilement être résolues sans un programme informatique. **Attention !** Votre programme devra trouver la réponse **en moins d'une minute !**

256 problèmes à résoudre ! Vous avez du grain à moudre...

7.12. Ce que vous avez appris dans ce chapitre

- Vous avez vu comment stocker et lire des données dans un fichier texte.
- Une chaîne de caractères (*string*) est une suite ordonnée de caractères (lettres, chiffres, symboles). Elle n'est pas modifiable, mais on peut quand même la manipuler grâce aux nombreuses méthodes de la classe 'str'.
- Vous avez vu que l'on peut manipuler des ensembles (*set*) comme en mathématiques. Les ensembles sont parfois pratiques, notamment pour éliminer des doublons.