Le mot le plus long

La seconde partie du jeu « Des chiffres et des lettres » est le « Le mot le plus long ». Il s'agit simplement de trouver le mot le plus grand à partir d'un tirage de lettres. Pour savoir si un mot est valide, on va utiliser une longue liste des mots français.

Activité 1 (Chercher un mot)**.**

Objectifs : vérifier si un mot donné est bien un mot de la langue française. On va extraire ces mots d'un fichier afin de créer un répertoire qui contient tous les mots admissibles.

Préalable. Il faut récupérer un des deux fichiers suivants qui contient des mots français :

- `repertoire_francais_simple.txt` qui contient 20 239 mots les plus courants (de **ABAISSER** à **ZYGOTE**),
- ou `repertoire_francais_tout.txt` qui contient une liste complète de 131 896 mots français de **A** à **ZYGOMATIQUES**.

Ces fichiers sont disponibles ici :

github.com/exo7math/python2-exo7

Les mots de ce fichier sont classés par ordre alphabétique. Ils sont en majuscules, sans accents : les seuls caractères autorisés sont les lettres **A** à **Z**. Des rappels pour lire/écrire un fichier texte sont donnés juste après cette activité.

1. **Lecture du répertoire.** Programme une fonction `lire_repertoire(fichier)` qui lit un fichier texte (pour nous un des deux fichiers `repertoire.txt`) et renvoie la (longue) liste de tous ces mots. On appelle « répertoire » la liste de tous ces mots français. Vérifie que tu obtiens le nombre de mots annoncé.

   Dans la suite on veut décider si un mot donné est un mot français ou pas. On dit qu'un mot est valide s'il apparaît dans notre répertoire. Par exemple **COUCOU** est français car il apparaît dans le répertoire (c'est le mot numéro 4 822 dans la liste simple et le numéro 27 374 dans la liste complète, en commençant la numérotation à 0). Par contre **BOLOSS** n'y est pas et n'est donc pas considéré comme un mot français.

   Il s'agit donc de décider si un mot donné est présent ou pas dans une liste. `Python` saurait faire cela très bien, à l'aide de l'opérateur `in` ou de la méthode `index()`. Ici on va programmer deux fonctions.

2. **Recherche séquentielle.**

   Programme une fonction `recherche_basique(mot,liste)` qui renvoie le rang du mot dans la liste s'il est présent et `None` sinon. Vérifie que ta fonction est correcte en comparant avec la commande `liste.index(mot)`.

   Méthode imposée. Utilise une boucle « tant que » qui parcourt un par un les éléments de la liste et les compare au mot cherché. Tu as seulement le droit d'effectuer des tests d'égalité entre deux chaînes de caractères.

Questions. Le mot **SERPENT** est-il dans notre répertoire, si oui à quelle place ? Et le mot **PYTHON** ?

Analyse. Si on cherche le rang d'un mot français alors, en moyenne, il faut parcourir la moitié de la liste avant de le trouver. Il faut donc effectuer en moyenne 10 000 tests dans le cas de la liste simple, ou environ 60 000 dans le cas de la liste complète. Si un mot n'appartient pas à la liste, on le sait seulement après avoir testé tous les éléments.

3. **Recherche dichotomique.**

   On va profiter du fait que la liste soit ordonnée pour faire la recherche beaucoup plus rapidement. Programme une fonction `recherche_dichotomie(mot,liste)` qui renvoie de nouveau le rang du mot dans la liste s'il est présent et `None` sinon.

   Algorithme de la dichotomie. Pour comprendre l'idée, voir le principe de la dichotomie dans le chapitre « Dérivée – Zéros de fonctions ».

   Algorithme.
   - — Entrée : un mot à trouver et une liste de mots.
     — Sortie : le rang du mot trouvé dans la liste ou `None` en cas d'échec. (Le rang d'une liste commence à 0.)
   - $a \leftarrow 0$
   - $b \leftarrow n-1$ où n est la longueur de la liste.
   - Tant que $b \geqslant a$, faire :
     — $k \leftarrow (a+b)//2$
     — Si `mot` égal `liste[k]` alors renvoyer k.
     — Si `mot` vient après `liste[k]` dans l'ordre alphabétique alors faire $a \leftarrow k+1$,
     — sinon faire $b \leftarrow k-1$.
   - Renvoyer `None` (c'est le cas uniquement si aucun mot n'a été trouvé dans la boucle précédente).

   Ordre alphabétique. `Python` connaît l'ordre alphabétique ! Par exemple « `'INDIC' < 'INFO'` » vaut « Vrai ». On peut comparer deux chaînes de caractères :
   - `mot1 == mot2` vaut « Vrai » si les chaînes sont égales,
   - `mot1 < mot2` vaut « Vrai » si `mot1` est avant `mot2` dans l'ordre alphabétique,
   - `mot1 > mot2` vaut « Vrai » si `mot1` est après `mot2` dans l'ordre alphabétique.

   Question. Est-ce que tu obtiens les mêmes résultats qu'auparavant pour les mots **SERPENT** et **PYTHON** ? (La réponse doit être « oui » !)

   Analyse. Pour chercher un mot, on divise la liste en deux à chaque étape. Donc en k étapes on trouve un mot dans une liste de longueur 2^k. Ainsi si on a une liste de longueur n alors il faut environ $\log_2(n)$ étapes. Le nombre d'étapes devant être un entier c'est en fait $k = E(\log_2(n)) + 1$ (où $E(x)$ désigne la partie entière, qui est la commande `floor(x)` de `Python`). Si on compare avec la recherche séquentielle, l'amélioration est frappante :

Taille de la liste	Nb. max. d'étapes - Séquentielle	Nb. max. d'étapes - Dichotomie
n	n	$E(\log_2(n))+1$
Rép. simple $n = 20\,239$	20 239	15
Rép. complet $n = 131\,896$	131 896	18

Cours 1 (Les fichiers)**.**
Voici un très bref rappel des instructions `Python` pour lire et écrire un fichier texte.
Lire un fichier. Ici on ouvre un fichier en lecture et on affiche à l'écran chaque ligne.

```
fic = open("mon_fichier.txt","r")

for ligne in fic:
    print(ligne.strip())

fic.close()
```

La commande `ligne.strip()` renvoie la chaîne de caractères de la ligne sans les espaces de début et de fin, ni le saut de ligne.

Écrire un fichier. Voici comment écrire un fichier. Ici chaque ligne écrite est de la forme : `Ligne numéro x`.

```
fic = open("mon_fichier.txt","w")

for i in range(100):
        ligne = "Ligne numéro " + str(i) + "\n"
        fic.write(ligne)

fic.close()
```

Cours 2 (Dictionnaire)**.**
Un dictionnaire est un peu comme une liste, mais les éléments ne sont pas indexés par des entiers mais par une « clé ». Un ***dictionnaire*** est donc un ensemble de couples clé/valeur : à une ***clé*** est associée une ***valeur***.

Exemple : un dictionnaire identifiant/mot de passe.

- Voici l'exemple d'un dictionnaire `dico` qui stocke des identifiants et des mots de passe :
  `dico = {'jean':'rev1789', 'adele':'azerty', 'jasmine':'c3por2d2'}`
- Par exemple `'adele'` a pour mot de passe `'azerty'`. On obtient le mot de passe comme on accéderait à un élément d'une liste par l'instruction :
  `dico['adele']` qui vaut `'azerty'`.
- Pour ajouter une entrée on écrit :
  `dico['lola'] = 'abcdef'`
- Pour modifier une entrée :
  `dico['adele'] = 'vwxyz'`
- Maintenant la commande `print(dico)` affiche :
  `{'jean':'rev1789', 'adele':'vwxyz', 'jasmine':'c3por2d2', 'lola':'abcdef'}`
- Le parcours d'un dictionnaire se fait par une boucle « pour ». Par exemple, la boucle suivante affiche l'identifiant et le login de tous les éléments du dictionnaire :

  ```
  for prenom in dico:
      print(prenom + " a pour mot de passe " + dico[prenom])
  ```

- Attention : il n'y a pas d'ordre dans un dictionnaire. Tu ne contrôles pas dans quel ordre les éléments sont traités.

Commandes principales.

- Définir un dictionnaire `dico = {cle1:valeur1, cle2:valeur2,...}`
- Récupérer une valeur : `dico[cle]`
- Ajouter une valeur : `dico[new_cle] = valeur`
- Modifier une valeur : `dico[cle] = new_valeur`
- Taille du dictionnaire : `len(dico)`
- Parcourir un dictionnaire : `for cle in dico:` et dans la boucle on accède aux valeurs par `dico[cle]`
- Tester si une clé existe : `if cle in dico:`
- Dictionnaire vide : `dico = {}`, utile pour initialiser un dictionnaire dans le but de le remplir ensuite.

Des commandes un peu moins utiles :

- Liste des clés : `dico.keys()`
- Liste des valeurs : `dico.values()`
- On peut récupérer les clés et les valeurs pour les utiliser dans une boucle :

```
for cle,valeur in dico.items():
    print("Clé :", cle, " Valeur :", valeur)
```

Activité 2 (Dictionnaire avec Python)**.**

Objectifs : s'initier à la manipulation des dictionnaires en `Python`.

1. On reprend le dictionnaire des identifiants/mots de passe :
```
dico = {'jean':'rev1789', 'adele':'vwxyz', 'jasmine':'c3por2d2',
                          'lola':'abcdef'}
```
   (a) Ajoute `'angela'` avec le mot de passe `'qwerty'`. Affiche alors le dictionnaire et sa longueur.
   (b) Programme une fonction `affiche_mot_de_passe(prenom)` qui affiche soit « untel a pour mot de passe ... » ou bien « untel n'a pas de mot passe ».
2. Voici les prénoms/âges des enfants d'une classe.

Clé (prénom)	Valeur (âge)
`'zack'`	8
`'paul1'`	5
`'eva'`	7
`'paul2'`	6
`'zoe'`	7

   Note que l'on peut avoir deux valeurs identiques, mais pas deux clés identiques (d'où `'paul1'` et `'paul2'`).
   (a) Définis un nouveau dictionnaire `dico` qui correspond à ces données. Les valeurs sont ici des entiers.
   (b) Programme une boucle qui calcule la somme des âges, puis calcule la moyenne des âges.
3. Voici les notes d'un élève par matière :

Clé (matière)	Valeur (liste de notes)
`'maths'`	`[13,15]`
`'anglais'`	`[16,12,14]`
`'sport'`	`[17]`

(a) Pars d'un dictionnaire `notes = {}` vide, puis complète matière par matière le dictionnaire. Les valeurs sont ici des listes d'entiers.

(b) Introduis une nouvelle matière `'python'` avec les notes 18 et 17.

(c) Ajoute la note de 16 en `'maths'`. (Il s'agit juste d'ajouter un élément à la liste `dico['maths']`.)

Activité 3 (Anagrammes).

Objectifs : trouver tous les anagrammes de la langue française.

- Deux mots sont des ***anagrammes*** s'ils ont les mêmes lettres, mais dans des ordres différents. Par exemple **CRIME** et **MERCI** ou bien **PRIERES** et **RESPIRE**.
- L'***indice*** d'un mot est la suite ordonnée de ses lettres. Par exemple l'indice du mot **KAYAK** est **AAKKY**.
- Deux mots forment des anagramme exactement lorsqu'ils ont des indices identiques. Par exemple **CRIME** et **MERCI** ont bien le même indice **CEIMR**.

1. Programme une fonction `calculer_indice(mot)` qui renvoie l'indice du mot. Par exemple `calculer_indice("KAYAK")` renvoie `"AAKKY"`.
   Indications minimalistes. Tu peux utiliser `join()`, `list()`, `sort()` dans le désordre.
2. Programme une fonction `sont_anagrammes(mot1,mot2)` qui teste si les deux mots donnés sont des anagrammes. Avec **CHIEN** et **NICHE** la fonction renvoie « Vrai ».
3. Programme une fonction `dictionnaire_indices_mots(liste)` qui à partir d'une liste de mots construit un dictionnaire ; dans ce dictionnaire les clés sont les indices, et à chaque indice est associé les mots de la liste correspondant.
   Exemple. Voici une liste de mots :
   `['CRIME', 'COUCOU', 'PRIERES', 'MERCI', 'RESPIRE', 'REPRISE']`
   et voici le dictionnaire renvoyé par la fonction :

   ```
   {
   'CEIMR': ['CRIME', 'MERCI'],
   'CCOOUU': ['COUCOU'],
   'EEIPRRS': ['PRIERES', 'RESPIRE', 'REPRISE']
   }
   ```

   Ainsi dans notre dictionnaire les mots sont regroupés par anagrammes et indexés par l'indice. Note que chaque valeur associée à un indice n'est pas un mot mais une liste de mots.
   Méthode.
   - Partir d'un dictionnaire vide `dico = {}`.
   - Pour chaque mot de la liste :
     — calculer l'indice du mot,
     — si cet indice n'existe pas déjà dans le dictionnaire, alors ajouter une nouvelle entrée : `dico[indice] = [mot]`,
     — si l'indice existe déjà, alors ajouter le mot à la liste existante : `dico[indice].append(mot)`.
4. À partir du répertoire (simple ou complet) des mots de la langue française, dresse la liste de tous les anagrammes possibles. Combien trouves-tu de classes d'anagrammes en tout ? Quel est l'anagramme supplémentaire à **PRIERES**, **RESPIRE**, **REPRISE** ? Trouve les anagrammes de **CESAR**.
5. *Bonus.* Programme une fonction `fichier_indice_mots(fichier_in,fichier_out)` qui à partir du fichier de tous les mots français (`fichier_in`), écrit dans un fichier (`fichier_out`) les indices et les mots correspondants. Voici un extrait du fichier obtenu :

```
CEHO : ECHO
CEHOP : CHOPE POCHE
CEHOPR : CHOPER PROCHE
CEHOPRS : PROCHES
CEHOPS : POCHES
```

Cours 3 (Le mot le plus long : règle du jeu)**.**
Le jeu « Le mot le plus long » est très simple. On tire un certain nombre de lettres au hasard (de 7 à 10 lettres). Il s'agit de trouver un mot français ayant le plus de lettres possibles.
Par exemple avec les lettres [**G**, **E**, **A**, **T**, **G**, **A**, **N**], on peut former des mots de 5 lettres **AGENT**, **GAGNE**, **ETANG**, des mots de 6 lettres comme **GAGENT**, et des mots de 7 lettres comme **TANGAGE**.
Les lettres sont choisies au hasard, à l'aide d'un tirage sans remise. Certaines lettres (en particulier les voyelles) sont plus présentes que d'autres. Voici une constitution possible avec 59 plaques :

- Voyelles **A,E,I,O,U** : 5 plaques chacune,
- Consonnes fréquentes **B,C,D,F,G,H,L,M,N,P,R,S,T** : 2 plaques chacune,
- Consonnes rares **K,J,Q,V,W,X,Y,Z** : 1 plaque chacune.

Cours 4 (Le mot le plus long : stratégie)**.**
Une mauvaise stratégie. Commençons par l'idée qui vient en premier pour trouver le mot le plus long. Imaginons que l'on a un tirage de 10 lettres. Notre idée est la suivante, on génère toutes les combinaisons de mots possibles à partir de ces 10 lettres, puis on teste un par un s'ils sont présents dans le répertoire des mots français. Quel est le problème ? Il y a $10! = 10 \times 9 \times 8 \times \cdots \times 1 = 3\,628\,800$ combinaisons possibles (10 choix pour la première lettre, 9 choix pour la seconde…). En ensuite il faut tester si chaque mot est dans le répertoire, ce qui demande au moins 10 tests d'égalité (voir la première activité). Donc en tout plus de 30 millions de tests d'égalité ce qui est beaucoup, même pour `Python` !

Une bonne stratégie. Il vaut mieux partir des mots connus que des mots possibles, car il y en a moins. Voici le principe :

- Génération du dictionnaire (à faire une fois pour toute, voir l'activité 3) :
  — on part de la liste des mots existants en français,
  — pour chaque mot on calcule son indice,
  — on obtient un dictionnaire indexé par les indices et qui contient tous les mots français possibles associés.
- Trouver le mot le plus long à partir d'une suite de lettres :
  — on commence par ordonner ces lettres,
  — puis voir si cela correspond à un indice de notre dictionnaire,
  — si c'est non il n'y a pas de mot,
  — si c'est oui la valeur associée à l'indice donne le ou les mots français.

Exemple. Avec le tirage de lettres **N**, **E**, **C**, **O**, **I**, on ordonne les lettres pour obtenir le candidat indice **CEINO**. On cherche dans le dictionnaire des indices/mots si cet indice existe, et ici on trouve qu'il existe et qu'il est associé au mot **ICONE**.

Analyse. La génération du dictionnaire n'est faite qu'une seule fois et nécessite 20 000 (ou 130 000) étapes. À chaque tirage, chercher est très simple, il s'agit juste de chercher si un indice existe dans la liste des 20 000 (ou 130 000) indices. Si la liste est ordonnée, on a vu que cela se fait en moins de 20 étapes. C'est donc quasi-immédiat.

Une difficulté. Prenons le tirage **X**, **E**, **C**, **S**, **I**, le candidat indice est **CEISX**, mais il n'est associé à aucun mot français. Il faut donc chercher des mots moins longs, par exemple sans le **X**, les lettres restantes forment l'indice **CEIS** qui est l'indice du mot **SCIE**. Conclusion : pour un tirage donné il faut aussi considérer tous les sous-tirages possibles.

Activité 4 (Le mot le plus long)**.**

Objectifs : gagner à tous les coups au jeu « Le mot le plus long »

1. Programme une fonction `tirage_lettres(n)` qui renvoie une liste de n lettres tirées au hasard.
2. Programme une fonction `liste_binaire(n)` qui renvoie toutes les listes possibles formées de n zéros et uns. Par exemple avec $n = 4$, la fonction renvoie la liste :
   `[ [1,1,1,1], [1,1,1,0], [1,1,0,1],..., [0,0,1,0], [0,0,0,1], [0,0,0,0] ]`
   Indications. Tu peux utiliser l'écriture binaire des entiers. Pour k variant de 0 à $2^n - 1$:
   - calculer l'écriture binaire de k (exemple `bin(7)` renvoie `'0b111'`),
   - supprimer le préfixe `'0b'` et transformer le reste en une liste d'entier (ex. on obtient `[1,1,1]`),
   - rajouter éventuellement des zéros pour obtenir la longueur souhaitée (ex. avec $n = 5$, on obtient `[0,0,1,1,1]`).
3. Reprends et modifie la fonction précédente pour obtenir une fonction `indices_depuis_tirage(tirage)` qui à partir d'une liste de lettres forme tous les indices possibles.
   Par exemple avec le tirage `['A','B','C','L']` de $n = 4$ lettres, la fonction renvoie les $15 = 2^n - 1$ indices possibles (qui correspondent à tous les sous-tirages possibles) :
   `['ABCL','ABC','ABL','ACL','BCL','AB','AC','BC','AL','BL','CL','A','B','C','L']`
   Indication. Une liste de 0 et de 1 indique quelles lettres du tirage on conserve : on garde les lettres correspondant aux 1. Avec les lettres de départ `['A','B','C','L']` :
   - `[1,1,1,1]` donne l'indice `'ABCL'` (on garde toutes les lettres),
   - `[1,1,1,0]` donne l'indice `'ABC'`,
   - `[1,1,0,1]` donne l'indice `'ABL'`,
   - ...
   - `[0,0,0,1]` donne l'indice `'L'`.

   Il est important d'avoir ordonné les lettres du tirage par ordre alphabétique pour obtenir des indices valides.
4. Programme une fonction `mot_le_plus_long(tirage,dico)` qui permet de gagner au jeu. `tirage` est une liste de lettres et `dico` est un dictionnaire qui associe à des indices les mots français correspondant (c'est le dictionnaire de l'activité 3).

   Indications.
   - Calcule la liste des indices possibles à partir du tirage (c'est la question précédente).
   - Ne conserve que les indices valides. Ce sont ceux qui apparaissent dans le dictionnaire (utilise le test `if indice in dico:`).
   - À partir des indices valides, trouve les mots français réalisables (ils sont directement donnés par `dico[indice]`).

   Exemple. Avec notre tirage `['A','B','C','L']`, on trouve par exemple :
   - indice valide `'ABC'`, mot associé `'BAC'`,
   - indice valide `'ABL'`, mot associé `'BAL'`,

- indice valide `'ACL'`, mots associés `'CAL'` et `'LAC'`.

Comme aucun mot n'est associé à l'indice `'ABCL'`, il n'y a pas de mots français de longueur 4 avec ce tirage, les meilleurs mots possibles sont donc de longueur 3.

Voici des exemples à traiter :

- Trouve des mots avec le tirage de 7 lettres `['Z','M','O','N','U','E','G']`.
- Trouve des mots avec le tirage de 8 lettres `['H','O','I','P','E','U','C','R']`.
- Trouve des mots avec le tirage de 9 lettres `['H','A','S','T','I','D','O','I','T']`.
- Trouve des mots avec le tirage de 10 lettres `['E','T','N','V','E','U','Z','O','V','N']`.