# Séance 8 : Ensembles, dictionnaires et fichiers texte

L1 – Université Côte d'Azur

Récupérez le fichiers `dictionnaire.txt` sur la page du cours (en faisant `enregistrez-sous` et non copier-coller).

https://upinfo.univ-cotedazur.fr/~obaldellon/L1/bi1/tp8/dictionnaire.txt

puis placez-le dans votre répertoire de travail, il servira pour tout le TP. Ce fichier contient les mots du dictionnaire à raison d'un par ligne.

**Exercice 1** — Ensembles et dictionnaires

Dans cette exercice, `chemin` et `chemin_bilan` sont des chaînes de caractères indiquant le nom (ou le chemin) d'un fichier texte dans le répertoire courant. Typiquement, on pourra choisir `"dictionnaire.txt"` comme exemple pour `chemin`.

1. Écrire une fonction `ensemble_de_lettres(chemin)` qui renvoie l'ensemble des lettres contenu dans le fichier donnée en paramètre. On ne comptera pas les ponctuations comme étant des lettres Combien y a-t-il de lettres distinctes dans le fichier `dictionnaire.txt` ?
2. Écrire une fonction `statistique_lettre(chemin)` qui renvoie le dictionnaire indiquant pour chaque lettre (les clés) son nombre d'occurences.

```
>>> dico = statistique_lettre("toto.txt")
>>> dico['e'] # la lettre « e » apparaît 123 fois dans le fichiers toto.txt
123
```

3. Écrire une fonction `bilan(chemin, chemin_bilan)` qui écrit dans le fichier dont le nom est fournis par la chaîne `chemin_bilan` les deux phrases suivantes où on aura remplacé les « ... » par les bonnes valeurs

   La lettre la plus présente dans le fichier ... est ...
   La seconde lettre la plus présente dans le fichier ... est ...

**Exercice 2** — Utilisation du dictionnaire

À partir de cet exercice, on appelera `dico` la liste (de type `list` et non `dict`) de tous les mots du dictionnaire.

1. Écrivez une fonction `extraire_dictionnaire()` qui lit le fichier `dictionnaire.txt` et renvoie une liste dont les éléments sont les mots du dictionnaire. Sauvegarder le résultat de cette fonction dans une variable `dico`.
2. Faites afficher les vingt premiers mots de `dico`, un par ligne, en donnant leur indice dans `dico` et leur longueur.

```
0 abaissable 10
1 abaissante 10
2 abaissée 8
...
19 abarticulation 14
```

3. Calculez **sans l'afficher** la liste `L` des longueurs des mots du dictionnaire. Affichez `L[:3]`, vous devez trouver `[10,10,8]`
4. À l'aide d'une boucle `for`, calculez `longueur_max(dict)` la longueur du plus long mot de `dict`. Vérifiez en comparant `longeur_max(dico)` à `max(L)`.
5. Affichez le ou les mots les plus longs du dictionnaire. Vous devriez en trouver trois.

**Exercice 3** — Le jeu du pendu

Le jeu du pendu est un jeu entre deux joueurs, appellés ci-dessous *Alice* et *Bob*. Au début, Alice choisit un mot secret $m$ et le publie en ne divulgant que sa première et dernière lettre, les autres étant remplacées par la symbole « _ ». À chaque tour, Bob propose une lettre : si elle fait partie du mot, Alice complète l'affichage en l'ajoutant autant de fois qu'elle apparaît, sinon Bob perd une vie. L'objectif est pour Bob de deviner le mot sans perdre toute ses vies.
**Exemple de partie** ou Alice est joué par l'ordinateur et perd face à Bob (l'humain)

```
il te reste 5 vies
e___t

Quelle lettre demandes-tu, humain ? e
Bien joué !
e__et

Quelle lettre demandes-tu, humain ? m
Raté… Il te reste 4 vies
e__et

Quelle lettre demandes-tu, humain ? f
Bien joué !
effet
GAGNÉ ! On rejoue ?
```

Dans votre répertoire, créez un fichier `pendu.py` Au début du fichier, commencez par créer la liste `dico` à partir du fichier. `dictionnaire.txt` comme dans l'exercice 2.

1. Écrivez une fonction `mot_au_hasard()` sans arguments qui renvoie un mot tiré au hasard. Vous devrez utiliser la variable globale `dico` et la fonction `randint` du module `random` (si vous ne vous souvenez plus comment fonctionne `randint`, faites `help(randint)`. Testez votre fonction en affichant dix mots tirés au hasard
2. Écrivez une fonction `cache_lettre(mot)`, qui renvoie une chaîne de caractère obtenue en remplaçant dans `mot` toutes les lettres (sauf la première et la dernière) par le symbole : `_`. Par exemple `cache_lettre('inspiration')` renverra `'i_________n'`.
3. Écrivez une fonction `ajoute_lettre(mot,secret,lettre)` qui prends trois arguments :
   - `secret` est un mot (une chaîne de caractère)
   - `mot` est une version incomplète de la chaîne `secret`
   - `lettre` est une des 26 lettres de l'alphabet.

   Si `lettre` est un caractère de `secret`, la fonction `ajoute_lettre` doit renvoyer une version complétée de `mot` auquel on a ajouté la lettre « `lettre` », sinon, on renvoie la chaîne `mot` sans modification. Dans cette question, on suppose que les mots ne contiennent pas d'accent.

```
>>> ajoute_lettre('e_s__g__m__t','enseignement','e')
'e_se_g_eme_t'
>>> ajoute_lettre('e_s__g__m__t','enseignement','z')
'e_s__g__m__t'
```

4. Écrire une fonction `pluriel` qui prend un nombre et un mot et qui renvoie la chaîne correspondante en accordant au pluriel si nécessaire.

```
>>> (pluriel(3,"chien") , pluriel(1,"chat"))
('3 chiens', '1 chat')
```

5. Il est temps de voir comment nous allons utiliser ces fonctions. La fonction principale sera la suivante :

```python
def nouvelle_partie(vies) :
    secret = mot_au_hasard() # le mot a deviner
    mot = cache_lettre(secret) # le mot partiellement decouvert
    print('il te reste' , vies , 'vies')
    while vies > 0 and '_' in mot :
        print(mot)
        c = input('Quelle lettre demandes-tu, humain ? ')
        ancien_mot=mot
        mot=ajoute_lettre(mot,secret,c)
        if mot != ancien_mot :
            print('Bien joué !')
        else :
            vies = vies-1
            print('Raté… Il te reste ' + pluriel(vies,'vie')+'.' )
    if vies == 0 :
        print('PENDU! Le mot secret était' , secret + '. On rejoue ?')
    else :
        print(mot)
        print('GAGNÉ ! On rejoue ?')
```

Copier la et testez la.

6. On va maintenant gérer correctement les caractères accentués en utilisant une fonction.

```python
from unicodedata import normalize
def sans_accents(s):
    return normalize('NFKD',s).encode('ascii' , 'ignore').decode('ascii')
```

Cette fonction permet de renvoyer une chaîne obtenue en otant les cédilles et les accents en les remplaçant par leur version ASCII. Modifiez votre fonction `ajoute_lettre` pour qu'elle gère correctement les accents.

```
>>> ajoute_lettre('d___f___nç____nt','déréférençassent','e')
'dé_éfé_enç___ent'
```

**Exercice 4** — Record battu !

Modifier votre programme précédant pour qu'il ajoute votre score (nombre de vies restantes) dans un fichier `score.txt`. Le fichier devra se présenter sous cette forme :

```
Vies restantes (meilleur score)
================================
6 : Olivier le 19/02/2021 à 12:30:1
5 : Sandrine le 23/03/2021 à 11:11:11
1 : Pierre le 15/02/2021 à 09:03:51
```

Lorsque le joueur gagnera on lui demandera son nom, on lira le fichier `score.txt`, on stockera les anciens résultats dans une liste que l'on mettra à jour avec le nouveau score puis on réecrira le tout dans le fichier initial. Si le fichier n'existe pas, il faudra le créer. On fera attention à ce que les scores soient toujours classés par ordre décroissant. On pourra s'aider de la fonction suivante pour afficher la date.

```python
from time import localtime
def date():
    t=localtime()
    heure=f"{t.tm_hour:0>2}:{t.tm_min:0>2}:{t.tm_sec:0>2}"
    jour =f"{t.tm_mday:0>2}/{t.tm_mon:0>2}/{t.tm_year:0>2}"
    return 'le ' + jour + ' à ' + heure
```

**Exercice 5** — L'ordinateur sauve sa peau

On veut maintenant échanger les rôles, et faire chercher le mot secret à l'ordinateur. On commence par implémenter une première stratégie utilisant du hasard, puis on étudie une stratégie plus efficace.

1. Écrivez une fonction `candidats(début,fin,longueur)` qui prend en argument deux caractères `début` et `fin` et un entier `longueur` et qui renvoie la liste de tous les mots du dictionnaire qui commencent par `début`, finissent par `fin`, et sont de longueur `longueur`. Par exemple :

```
>>> candidat('h','s',6)
['habeas', 'hachis', 'haggis', 'hermès', 'herpès', 'hiatus', 'hormis', 'hybris']
```

2. Écrivez une fonction `choix_lettre(mot,possibilités)` qui prend en argument un mot partiellement révélé et une liste de mots `possibilités` et qui renvoie au hasard une lettre qui apparait dans au moins un mot de `possibilités` à une position où on n'a encore rien révélé. Par exemple, si `s = 'h___is'` et `possibilités = ['hachis', 'haggis', 'hormis', 'hybris']`, un appel à `choix_lettre(mot,possibilités)` renverra une lettre au hasard parmi `achgormybr`.
3. Écrivez une fonction `filtre_lettre(lettre,possibilités)` qui prend en argument un caractère `lettre` et une liste de mots `possibilités` et qui renvoie la sous-liste des mots qui ne contiennent pas `lettre`, sauf éventuellement comme première ou dernière lettre. Par exemple,

```
>>> filtre_lettre('e',['avion','état','route','été','enquête'])
['avion', 'état', 'route', 'été']
```

4. Écrivez une fonction `est_compatible(mot,secret)` qui prend en arguments deux chaînes de caractères correspondant à un mot incomplet (`mot`) et un mot complet `secret`, et qui renvoie `True` si `mot` peut se compléter en `secret`. Par exemple,

```
>>> est_compatible('h___is','hachis')
True
>>> est_compatible('h___is','hermès')
False
>>> est_compatible('he___s','hermès')
False
```

5. Écrivez une fonction `joue_chercheur(vies)` qui démarre une partie où vous devez faire deviner un mot à l'ordinateur. L'ordinateur donnera des informations sur l'état d'avancement de ses réflexions. On aura par exemple

```
Quel est ton indice de départ, humain? h____s
J'hésite entre : habeas hachis haggis hermès herpès hiatus hormis hybris
Je demande le a. Nouvel indice? h____s
Il me reste 4 vies.
J'hésite entre : hermès herpès hormis hybris
Je demande le i. Nouvel indice? h___is
J'hésite entre : hormis hybris
Je demande le y. Nouvel indice? hy__is
J'ai trouvé : hybris.
```

6. Améliorez la fonction `choix_lettre(s,L)` pour accroître les chances de gagner de l'ordinateur.
   *Indication* Une possibilité est de s'inspirer de la dichotomie : on peut penser que le choix de la lettre `c` sera un bon choix si il permet de garder autant de candidats qu'il permet d'en éliminer, autrement dit si le nombre de mots de `possibilité` qui contiennent `c` est le plus proche possible de la moitié de la longueur de `possibilité`. Ainsi quelque soit la réponse, on éliminera environ la moitié des possibilités.
   Par exemple, si `mot = 'h_____s'` et `L = ['habeas', 'hachis', 'haggis', 'hermès', 'herpès', 'hiatus', 'hormis', 'hybris']`, le choix de `'i'` permet de garder 5 candidats et d'en éliminer 3, tandis que le choix de `'y'` permet de garder 1 candidats et d'en éliminer 7 ; le choix de `i` est donc meilleur que le choix de `y`. Mais le choix optimal, sur cet exemple, est `'a'`, car il permet de garder 4 candidats et d'en éliminer tout autant.
   Vous pouvez cependant réfléchir à d'autres stratégies (maximiser les chances de réduire le nombre de lettres à deviner, stratégie « petit joueur » pour prendre le moins de risque possible, etc), et comparer expérimentalement l'efficacité de ces stratégies.