Séance 6 : Exceptions et fichiers textes

L1 – Université Côte d'Azur

Exercice 1 — Nombre de définitions (⋆⋆)

Définissez une fonction `nombre_définitions(s)` qui renvoie le nombre de fonctions définies dans le fichier Python nommé `s`. Sauvez votre fonction dans un fichier `tp6-exo1.py` et testez au toplevel. *Indication* : dans un programme Python, une définition de fonction est une ligne qui commence par `def`.

```
>>> nombre_définitions('tp6-exo1.py')
1
```

Exercice 2 — Simulateur d'achat (⋆⋆)

On se donne une liste `inv` correspondant à un inventaire des produits d'un magasin.

```
>>> inv
[('Pommes', 10), ('Carottes', 5), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]
>>> maj_inventaire(inv,'Carottes',3)
>>> inv # il n'y a maintenant plus que 2 carottes !
[('Pommes', 10), ('Carottes', 2), ('Radis', 23), ('Lentilles', 534), ('Poivrons', 12)]
```

1. Écrivez une procédure `maj_inventaire(inv,produit,n)` qui **modifie** la liste `inv` en supprimant `n` occurences du produit `produit`.
2. Modifier la fonction pour qu'elle lance `IndexError` si le produit n'existe pas et lance `ValueError` s'il n'y a pas assez de produits disponibles.
3. Écrivez un programme `achat(inv,produit,n)` qui fait appel à la fonction `maj_inventaire` et qui rattrape les éventuelles exceptions pour afficher un des trois messages suivant « `Merci pour votre achat.`, `Produit inexistant.`, `Produit en quantité insuffisante.` »

```
>>> achat(inv,'Pommes',1)
Merci pour votre achat.
>>> achat(inv,'Poivrons',2048)
Produit en quantité insuffisante.
>>> achat(inv,'Aubergines',10)
Produit inexistant.
```

Exercice 3 — Utilisation du dictionnaire (⋆)

1. 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/py/tp6/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.
2. 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`.

```python
def extraire_dictionnaire():
    s="dictionnaire.txt"
    f_in = open(s, 'r', encoding = 'utf-8')
    L=[]
    for ligne in f_in:
        mot=ligne[0:-1] # le dernier symbole correspond au symbole '\n'
        L.append(mot)
    return L

dico=extraire_dictionnaire()
```

3. 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
```

```python
for i in range(20) :
    print( i , dico[i] , len(dico[i]) )
```

4. Calculez **sans l'afficher** la liste `L` des longueurs des mots du dictionnaire. Affichez `L[:3]`, vous devez trouver `[10,10,8]`

Première méthode en bouclant sur les indices.

```python
n=len(dico)
L=[]
for i in range(n) :
    L.append(len(dico[i]))
```

Seconde méthode en bouclant directement sur les mots.

```python
n=len(dico)
L=[]
for mot in dico :
    L.append(len(mot))
```

Troisième méthode avec une compréhension.

```python
[ len(mot) for mot in dico ]
```

5. 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)`.

```python
def longueur_max(dico):
    l_max=0
    for i in range(len(dico)):
        if len(dico[i])>l_max:
            l_max=len(dico[i])
    return l_max
```

Avec une boucle `for` plus pythonesque.

```python
def longueur_max(dico):
    l_max=0
    for mot in dico:
        if len(mot)>l_max:
            l_max=len(mot)
    return l_max
```

6. Affichez le ou les mots les plus longs du dictionnaire. Vous devriez en trouver trois.

```
l_max=longueur_max(dico)
for m in dico :
    if len(m) == l_max :
        print(m)
```

anticonstitutionnellement désoxyribonucléoprotéique oligoasthénotératospermie

Exercice 4 — 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

```
def mot_au_hasard():
    # dico est une variable globale
    n=len(dico)
    aléa=randint(0,n-1)
    return dico[aléa]
```

Si on veut être sûr de ne jamais tirer de mots de deux ou une lettre, on peut rajouter un test et un appel récursif.

```
def mot_au_hasard():
    # dico est une variable globale
    n=len(dico)
    aléa=randint(0,n-1)
    if len(dico[aléa]) >=3:
        return dico[aléa]
    else:
        return mot_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'`.

```
def cache_lettre(mot):
    n=len(mot)
    assert(n>=3)
    res=mot[0]
    for i in range(1,n-1):
        res = res + '_'
    return res + mot[n-1]
```

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'
```

```
def ajoute_lettre(mot,secret,lettre):
    n=len(mot)
    assert(n==len(secret))
    res=""
    for i in range(n):
        if secret[i]==lettre:
            res = res+secret[i]
        else:
            res = res+mot[i]
    return res
```

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')
```

```
def pluriel(k,mot):
    if k==0 or k==1:
        return str(k) + ' ' + mot
    else:
        return str(k) + ' ' + mot + 's'
```

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'
```

```python
def ajoute_lettre(mot,secret,lettre):
    n=len(mot)
    assert(n==len(secret))
    res=""
    for i in range(n):
        if asciize(secret[i])==lettre:
            res = res+secret[i]
        else:
            res = res+mot[i]
    return res
```

Exercice 5 — 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
```

La fonction `découpe` transforme une chaîne de la forme `'6 : Olivier le 19/02/2021 à 12:30:1'` en un couple contenant un entier et une chaîne : `(6,'Olivier le 19/02/2021 à 12:30:1')`

```python
def découpe(ligne):
    k = ligne.index(" : ")
    return (int(ligne[:k]),ligne[k+3:])
```

La fonction ajouter permet d'insérer notre couple au bonne endroit parmis une liste de couple déja triéé. On commence a ajouter les scores qui sont supérieurs, puis notre nouveau score puis les scores inférieurs. La liste obtenue est donc toujours triéée.

```python
def ajoute(nouvelle_ligne,liste):
    (score,chaîne)=nouvelle_ligne
    rés=[]
    i=0
    # On ajoute les scores supérieurs
    while i<len(liste) and liste[i][0]>score:
        rés.append(liste[i])
        i=i+1
    # On ajoute le nouveau score
    rés.append(nouvelle_ligne)
    # On ajoute les scores inférieurs
    while i<len(liste):
        rés.append(liste[i])
        i=i+1
    return rés
```

La fonction pour afficher les meilleurs scores (en supposant que le fichier existe déjà)

```python
def afficher_score():
    assert(os.path.isfile('score.txt'))
    f_in = open('score.txt', 'r', encoding = 'utf-8')
    contenu=f_in.read()
    f_in.close
    print(contenu)
```

```python
import os.path

def ajout_score(vies,nom):
    if os.path.isfile('score.txt'):# On vérifie si le fichier existe déjà.
        f_in = open('score.txt', 'r', encoding = 'utf-8')
        liste=f_in.readlines()
        f_in.close()
        liste=liste[2:] # on supprime les deux première ligne
        liste= [ découpe(ligne) for ligne in liste ]
    else:
        liste=[]
    nouvelle_ligne = (vies, nom+' '+date()+'\n')
    liste = ajoute(nouvelle_ligne,liste)
    f_out = open('score.txt', 'w', encoding = 'utf-8')
    f_out.write('Vies restantes (meilleur score)\n')
    f_out.write('================================\n')
    for couple in liste:
        (v,c)=couple
        f_out.write(str(v) + ' : ' + c)
    f_out.close()
```

Exercice 6 — 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']
```

Version classique

```python
def candidats(deb,fin,l) :
    L=[]
    for m in dico:
        if len(m) == l and m[0] == deb and m[-1] == fin:
            L.append(m)
    return L
```

Version plus courte

```python
def candidats(deb,fin,l) :
    return [ m for m in dico if len(m) == l and m[0] == deb and m[-1] == fin]
```

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`.

```python
def choix_lettre(s,L):
    lettres = []
    l = len(s)
    for m in L :
        assert(len(m) == l)
        for i in range(l) :
            if s[i] == '_' :
                lettres.append(asciize(m[i]))
    x = randint(0, len(lettres) - 1)
    return lettres[x]
```

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']
```

```python
def filtre_lettre(c,L) :
    F=[]
    for m in L:
        if c not in asciize(m[1:-1]):
            F.append(m)
    return F

# Version plus courte
def filtre_lettre(c,L) :
    return [m for m in L if c not in asciize(m[1:-1])]
```

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
```

```python
def est_compatible(s,m) :
    if len(s) != len(m) :
        return False
    for i in range(len(s)) :
        if s[i] != '_' and s[i] != m[i] :
            return False
        if s[i] == '_' and m[i] != '_' and m[i] in s[1:-1] :
            return False
    return True
```

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.
```

```python
def joue_chercheur(vies) :
    s = input('Quel est ton indice, humain? ')
    L = candidats(s[0],s[-1],len(s))

    while vies > 0 and len(L) > 1 :

        print("J'hésite entre" , L)
        lettre = choix_lettre(s,L)
        print('Je demande le ' , lettre + '.')

        while True :
            s2 = input('Nouvel indice? ')
            if est_compatible(s,s2) : break
            print('erreur de saisie. Ancien indice :',s)

        if s2 == s :
            # rate
            L = filtre_lettre(lettre,L)
            vies = vies - 1
            print('Il me reste' , vies , 'vie' + ('s' if vies > 2 else '') + '.')
        else :
            # bonne lettre
            s = s2
            L = [m for m in L if est_compatible(s,m)]

    if len(L) == 1 :
        print("J'ai trouvé :" , L[0])
    elif len(L) == 0 :
        print("Tricheur!")
    else :
        print("J'ai perdu, tu avais bien choisi ton mot...")
```

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.