TP #6 — Fichiers

Manipulation de fichiers textes en Python

On donne ci-dessous une documentation succincte de quelques fonctions Python de manipulation de fichiers textes (et de chaînes de caractères). Dans le cadre du programme, vous devez être en mesure d'utiliser ces fonctions si une documentation est fournie.
La documentation ci-dessous étant très courte, elle ne reflète pas forcément les bonnes pratiques de manipulation de fichiers en Python ; pour plus de détails, vous pouvez consulter la documentation officielle à ce sujet.

— `open` :

  — Pour *s* une chaîne de caractères donnant le chemin (relatif ou absolu) d'un fichier, `open(s, "r")` construit et renvoie un objet permettant de *lire* (`"r"` : *read*) le contenu du fichier *s*, en utilisant les fonctions ci-dessous.
  — `open(s, "w")` fait de même, pour des opérations en *écriture* (`"w"` : *write*). **Attention** : cette ouverture en écriture **supprime le contenu actuel du fichier *s* s'il existe**. Si l'on souhaite éviter ceci, on peut utiliser `"a"` (*append*) à la place.
  — En fonction de l'encodage du fichier texte et de votre environnement, il peut être nécessaire d'ajouter une option particulière, par exemple : `open(s, "r", encoding="utf-8")`. Ceci semble être le cas sur certaines (?) des machines du lycée.

— `read` :

  — Pour *f* un objet tel que renvoyé par un appel en lecture à `open`, `f.read(c)` lit *au plus c* caractères dans le fichier correspondant. On peut également omettre l'argument pour lire l'intégralité (restante) du fichier, avec `f.read()`. Attention : une fois lus, ces caractères ne peuvent pas être relus, et si l'on souhaite y avoir à nouveau accès il faut nous-même les sauvegarder quelque part. Il faut se représenter *read* comme une fonction qui « consomme » le texte contenu dans *f*.
  Exemple :

```
In [100]: f.read(2)
Out[100]: 'MA'
In [101]: f.read(6)
Out[101]: 'RCEL P'
```

— `readline` :

  — Similaire à *read*, mais consomme et renvoie une « ligne » de texte (se terminant à un caractère `'\n'` de retour à la ligne).
  Exemple :
  ```
  In [102]: f.readline()
  Out[102]: 'À LA RECHERCHE DU TEMPS PERDU\n'
  ```

— `readlines` :

  — Renvoie une `list` de toutes les lignes (au même sens que *readline*) du fichier.

— `write` :

  — Pour f un objet tel que renvoyé par un appel en écriture à `open`, et s une chaîne de caractères, `f.write(s)` écrit s à la fin du fichier et renvoie le nombre de caractères écrits.

— `close` :

  — Pour f un objet tel que renvoyé par un appel à `open`, `f.close()` ferme le fichier correspondant. Il est particulièrement important de faire cet appel après avoir effectué des écritures (s'il est oublié, il est fort probable que des écritures seront perdues).

Exemple global :

```
In [103]: f = open("ohai.txt","w")
In [104]: f.write("OHAI\n")
Out[104]: 5
In [105]: f.write("hewwo\n")
Out[105]: 7
In [106]: f.close()
In [107]: f = open("ohai.txt","r")
In [108]: f.readlines()
Out[108]: ['OHAI\n', 'hewwo\n']
In [109]: f.read()
Out[109]: ''
```

On rappelle également l'existence d'une fonction `split` qui, pour une chaîne de caractères s et un caractère c est telle que `s.split(c)` renvoie la liste des sous-chaînes de s séparées par c :

```
In [110]: "Il fait beau aujourd'hui".split(' ')
Out[110]: ['Il', 'fait', 'beau', "aujourd'hui"]
```

Il existe aussi une variante d'appel particulièrement pratique : si aucun argument n'est donné pour *split*, la séparation se fait pour les espaces, les tabulations, et les caractères de retour à la ligne ; les chaînes vides sont également supprimées de la liste renvoyée :

```
In [111]: "Il fait\nbeau aujourd'hui".split()
Out[111]: ['Il', 'fait', 'beau', "aujourd'hui"]
```

Cette dernière variante de *split* est (très probablement) la plus utile pour l'exercice ci-dessous.

Exercice 1. *Génération pseudo-aléatoire de texte*

Basé sur une idée de Jean-Christophe Filliâtre

L'objectif de cet exercice est de générer des phrases aléatoires, après un « entraînement » préalable sur des textes existants.

Le principe est le suivant :

— *Entraînement*. Soit s une chaîne de caractères contenant un texte en français (ou toute autre langue), on découpe s en mots (séparés par des espaces, mais en gardant la ponctuation), et pour tout triplet de mots a, b, c consécutifs dans s, on mémorise le fait que c vient après a et b. On fait cela « avec multiplicité », c'est à dire que si c apparaît plusieurs fois comme successeur de a, b dans s, on le mémorise plusieurs fois également.

— *Génération*. On tire aléatoirement et uniformément deux mots consécutifs de s dont le premier commence par une majuscule ; on les concatène pour former un embryon de phrase, et tant que celui-ci ne se termine pas par un point on le prolonge par un mot tiré uniformément (en prenant en compte la multiplicité, *cf.* ci-dessus) parmi les successeurs des deux derniers.

  Exemple :

  — Les deux premiers mots sont « La » et « fille ». On tire uniformément un successeur parmi les mots suivant le couple (*La*, *fille*) dans le texte ; on obtient « de ».
  — On tire uniformément un successeur parmi les mots suivant le couple (*fille*, *de*) dans le texte ; on obtient « Minos »
  — etc.
  — On s'arrête lorsqu'on obtient « La fille de Minos et de la nuit. », qui se termine par un point.

On peut ainsi générer des phrases plus ou moins étranges, à la grammaire plus ou moins correcte :

— « Il veut toujours ménager la chèvre savante. »
— « Sa lumière avait détruit le bureau du télégraphe. »
— « Il s'élève sans effort; il s'abaisse comme s'il se fût abstenu. »
— « Si rares qu'ils devinssent, ces moments-là rien d'autre à la taille courte et légèrement jaunie déjà, nous marchions tous les trois sans bruit: Augustin avait à dépouiller son âme comme des objets de bronze sous les inscriptions qu'y avait ajoutées au crayon un des petits coups de baguette, en regardant du côté de Bourges, je reçus la deuxième lettre du grand Meaulnes; et dès

qu'il n'était heureux de la nuit les feuilles des pivoines; la terre gelée, un chien se mit à déboutonner cette pièce mystérieuse d'un costume qui n'était pas digne du docteur du Boulbon. »

— « D'ailleurs, comme je ne connaissais personne mieux que Mme Verdurin était une condition suffisante. »

— « Je revois ce lieu, qui devait lui apporter un chapeau pour la première vague de cette journée de fin d'août que j'aperçus, tournant au vent de cet avis, sans l'avouer. »

1. Écrivez une fonction `collect_words(s)`, dont le paramètre est une chaîne de caractère *s* donnant le chemin (absolu ou relatif) d'un fichier **texte**, et qui construit une liste ordonnée de tous les mots (séparés par des espaces, ou tabulations, ou caractères de retour à la ligne) du fichier de chemin *s*.
2. Testez votre fonction, par exemple avec un ou plusieurs romans disponibles sur le site du projet Gutenberg.
3. Écrivez une fonction `train(wl)` qui effectue la phase d'entraînement décrite ci-dessus, pour une liste de mots *wl* telle que construite avec un (ou plusieurs) appels à *collect_words*.
   C'est à vous de choisir un ou plusieurs type de données appropriés pour représenter le résultat de cet entraînement, mais n'hésitez pas à me solliciter si vous avez un doute sur votre approche.
4. Écrivez une fonction `sample` qui effectue la phase de génération décrite ci-dessus.
   Encore une fois, c'est à vous de choisir comment vous y prendre (de déterminer les arguments, leur type, éventuellement comment les construire *via* des fonctions intermédiaires...)
   Conseil : vous pouvez utiliser la fonction *isupper*, qui indique si la chaîne de caractères à laquelle elle est attachée est **intégralement** en lettres majuscules ou non :
   ```
   In [200]: s = "MARTRE"
   In [201]: s.isupper()
   Out[201]: True
   In [202]: s = "Martre"
   In [203]: s.isupper()
   Out[203@: False
   ```
5. Testez !!

Exercice 2. *Représentation textuelle de graphes*

Un *graphe* est un ensemble de *sommets* reliés par des *arêtes* ou *arcs*. Nous définirons plus longuement ces objets et leurs différentes représentations possibles dans les cours à venir, mais pour l'heure l'objectif va être d'implémenter deux petites fonctions utilitaires qui pourront être pratiques pour les TPs à venir. Le but de ces fonctions est de convertir un graphe représenté sous forme d'un fichier texte en

un graphe représenté par ce que nous appellerons une *liste de listes d'adjacences*, et *vice-versa*.
Plus précisément :

— Les sommets des graphes manipulés sont des entiers entre 0 et $S-1$, avec S le nombre de sommets du graphe.
— Pour chaque sommet, on stocke ses *voisins* dans une *liste d'adjacence* représentée par une `list` (mais on pourrait aussi bien utiliser un `dict`). Les voisins d'un sommet u sont les sommets v tels qu'il existe une arête $u-v$ dans un graphe *non-orienté*, ou un arc $u \to v$ (une arête orientée) dans un graphe *orienté*.
Dans un graphe non-orienté, on ne distingue donc pas l'arête $u-v$ de l'arête $v-u$, mais on devra bien trouver *à la fois* u dans la liste d'adjacence de v et v dans celle de u.
— On stocke toutes les listes d'adjacence dans un tableau représenté par une `list` : la liste d'adjacence du sommet u se trouve simplement à l'index u.

On propose le format (très simple) suivant pour décrire de tels graphes dans un format texte :

— La première ligne est ou bien `UNDIRECTED` ou bien `DIRECTED`, en fonction de si le graphe est non-orienté ou orienté.
— La seconde ligne est `S = n` avec n un entier (écrit en base 10) correspondant au nombre de sommets du graphe.
— Les lignes suivantes sont de la forme `u -- v` pour un graphe non-orienté, ou `u -> v` pour un graphe orienté (cette différence est purement cosmétique, comme vous devriez vous en rendre compte lors de l'implémentation), avec u et v des identifiants de sommet. Chacune de ces lignes indique la présence d'une arête ou d'un arc entre u et v, et dans le cas d'un graphe non-orienté une même arête n'est indiquée qu'une unique fois.

On donne ci-dessous une représentation d'un graphe sous forme de liste de listes d'adjacence, puis en représentation textuelle s'il est non-orienté, puis enfin en représentation textuelle s'il est orienté.

```
[[1, 2], [0, 4], [0, 3, 4], [2], [1, 2]]
```

```
UNDIRECTED
S = 5
0 -- 1
0 -- 2
1 -- 4
2 -- 3
2 -- 4
```

```
DIRECTED
S = 5
0 -> 1
0 -> 2
1 -> 0
1 -> 4
2 -> 0
2 -> 3
2 -> 4
3 -> 2
4 -> 1
4 -> 2
```

Remarque. Aucune de ces représentations n'est unique, puisque l'ordre dans lequel les voisins (ou les arêtes, ou les arcs) sont énumérés ne change pas le graphe lui même (en tant qu'objet formel). Dans le cas de sommets représentés par des entiers, on peut facilement (si l'on souhaite) rendre ces représentations uniques en imposant un ordre particulier pour cette énumération (celui de ces exemples, disons) ; on peut alors facilement tester si deux graphes sont littéralement égaux, mais décider l'égalité *à renommage près* des sommets (qui est souvent plus pertinente) n'est pas simple…

1. Écrivez une fonction `readg(p)`, où *p* est le chemin d'un fichier représentant un graphe avec la représentation textuelle décrite ci-dessus, et qui renvoie une représentation du même graphe sous forme de liste de listes d'adjacence. Rappel : si *s* est une chaîne de caractères donnant l'écriture d'un entier en base 10, `int(s)` renvoie une représentation de ce même nombre dans le type `int` ; à l'inverse, `str(i)` renvoie une chaîne de caractères représentant l'écriture en base 10 de l'entier *i*.
2. Testez votre fonction.
3. Écrivez une fonction `writeg(G, undir, p)` qui écrit dans le fichier de chemin *p* une représentation textuelle du graphe de liste de listes d'adjacence *G*, dont le booléen *undir* indique s'il est ou non non-orienté.
4. Testez votre fonction.
5. Écrivez une fonction `check_dir(G) -> bool` qui renvoie `True` si le graphe représenté par la liste de listes d'adjacence *G* est *nécessairement* orienté, et `False` sinon.
6. Testez votre fonction.
7. Quelle peut être l'utilité d'une telle fonction, dans le contexte de *writeg* ?
8. Modifiez votre fonction *writeg* en conséquence. (Si vous voulez en profiter pour tester des choses complètement et résolument hors programme, essayez d'utiliser (à bon escient) `raise ValueError` dans votre fonction modifiée.)