Manipulation de fichiers ; graphiques élémentaires avec `matplotlib`

D'après un TP de Stéphane Gonnord

Buts du TP

- Apprendre à lire ligne-à-ligne le contenu d'un fichier texte ; extraire les différents champs d'une ligne via des séparateurs. Apprendre à écrire des données simples dans un fichier texte.
- Réaliser des graphiques simples avec la bibliothèque `matplotlib`.

1 Rappels sur les manipulations de fichiers textes

Bloc-Note 1 *Lecture / écriture de fichiers textes*

En Python, l'accès à un fichier texte se fait par l'intermédiaire d'un descripteur de fichier créé à l'aide de la primitive `open(nom,mode)`. Une fois que les manipulations sont terminées, il faut bien penser à fermer le descripteur de fichier avec `f.close()`.

Lors de l'ouverture d'un fichier on précise l'un des trois modes d'accès : lecture `'r'`, écriture `'w'` ou ajout `'a'`. Attention, pour l'ouverture en mode écriture, les modes `'w'` ou `'a'` créent le fichier s'il n'existe pas mais le mode `'w'` écrase le fichier s'il existe déjà.

Pour bien manipuler un fichier texte, il faut d'abord connaître la façon dont il est structuré!!!

Fonctions	Rôle
`f = open('nom_fichier.txt','w')`	accès en écriture avec création d'un nouveau fichier
`f = open('nom_fichier.txt','r')`	accès à un fichier existant en mode lecture
`f = open('nom_fichier.txt','a')`	accès en écriture à un fichier existant en mode ajout
`f.write('texte')`	ajout de 'texte' dans le fichier
`f.writelines(liste)`	ajout d'une liste de lignes dans un fichier
`f.read()`	lecture de tout le fichier
`f.read(8)`	lecture des 8 premiers caractères du fichier
`f.readline()`	lecture de la ligne courante du fichier
`f.readlines()`	liste de toutes les lignes à partir de la position courante
`f.tell()`	position courante du curseur
`f.seek(16)`	place le curseur sur le caractère en position 16
`for ligne in f`	itération sur les lignes du fichier
`f.close()`	fermeture du fichier

Lecture de tout le fichier

```
1 f = open('fichier.txt','r')
2 data = f.read()
3 f.close()
```

Lecture ligne par ligne

```
1 f = open('fichier.txt','r')
2 for ligne in f:
3     #traitement sur la ligne
4 f.close()
```

Lecture ligne par ligne

```
f = open('fichier.txt','r')
ligne = f.readline()
while ligne != '':
    #traitement sur la ligne
    ligne = f.readline()
f.close()
```

Capture dans une liste de toutes les lignes

```
f = open('fichier.txt','r')
listlignes = f.readlines()
f.close()
```

Écriture

```
f = open('fichier.txt','w')
f.write('Une ligne qui ecrase tout\
     n')
f.close()
```

Ajout à la fin

```
f = open('fichier.txt','a')
f.write('Ligne de plus\n')
f.close()
```

EXERCICE 1 *Dans le dossier du TP du jour, placer les fichiers suivants :*
`cadeau.py`, `dico.txt`, `premiers-1000.txt`, `admissibles.txt`, `scores.txt`, `departements.txt`, `base_exp.txt`, `Dosage.txt`, `jocondeBW.jpg`

2 Lecture dans un fichier texte

EXERCICE 2 *Le fichier* `dico.txt` *contient des mots de la langue française rangés un par ligne et dans l'ordre alphabétique.*

1. *Écrire une fonction qui retourne le nombre total de mots contenus dans le fichier.*
2. *Écrire une fonction qui retourne le nombre de mots commençant par la même lettre que son prénom. Attention certains mots du fichier commencent par une majuscule. La méthode* `lower()` *des chaînes de caractères sera bien utile dans ce cas.*

   ```
   >>> 'A'.lower()
   'a'
   ```

3. *Écrire une fonction qui retourne le nombre de mots se terminant par la même lettre que son prénom.*

   Dans un fichier texte, le saut de ligne est marqué par un caractère invisible spécifique à la plateforme : `'\n'` *sous Linux ou* `'\r\n'` *sous Windows.*

   Pour supprimer le caractère de fin de ligne, on utilise l'une des méthodes `strip` *ou* `rstrip`.

   ```
   >>> f = open('dico.txt')
   >>> p = f.readline()
   >>> p
   'a\n'
   >>> p.rstrip()
   'a'
   ```

EXERCICE 3 *Le fichier* `premiers-1000.txt` *contient les* 1000 *plus petits entiers premiers rangés un par ligne et dans l'ordre croissant. Écrire une fonction qui lit ce fichier et retourne la somme des* 1000 *plus petits entiers premiers.*

EXERCICE 4 *Le fichier* `scores.txt` *contient les scores (entre* 0 *et* 1000*) réalisées lors de* 500 *parties d'un jeu video par* 26 *joueurs distincts de noms compris entre 'A' et 'Z'. Le fichier est mal formaté, chaque ligne contient un nombre aléatoire d'espaces puis la séquence* `'nom,score\n'`.

```
U,432
          C,424
  X,52
```

Pour chaque ligne, on supprime le caractère de fin de ligne puis on découpe selon un caractère de séparation en une liste de champs avec la méthode `split`*, dont la réciproque est la méthode* `join`*.*

```
>>> '          C,424\n'.strip()
'C,424'
>> 'C,424'.split(',')
['C', '424']
>>> ','.join(['C', '424'])
'C,424'
```

1. *Écrire une fonction* `extraire_scores(fichier)` *qui prend en argument le nom du fichier de scores et qui retourne un tableau avec les scores cumulés des* 26 *joueurs de 'A' à 'Z'. La fonction* `ord` *sera bien utile.*

   ```
   >>> ord('B') - ord('A')
   1
   >>> extraire_scores('scores.txt')
   [10404, 11664, 13091, ... , 10825]
   ```

2. *Modifier la fonction précédente en une fonction* `extraire_moyennes(fichier)` *qui retourne les scores moyens par partie des* 26 *joueurs.*

   ```
   >>> extraire_moyennes('scores.txt')
   [520.2, 432.0, 503.5,..., 470.6521739130435]
   ```

EXERCICE 5 *Le fichier*[1] `admissibles.txt` *contient* 5398 *lignes de la forme :*

```
1234  TOURNESOL Tryphon    Admissible   2
```

ou bien :

```
43210 HADDOCK Archibald    Eliminé      -
```

(Des tabulations séparent les quatre champs ; le nom et le prénom séparés par un espace sont dans un même champ.)

Écrire une séquence d'instructions qui lit ce fichier, détermine le nombre d'admissibles et le nombre de candidats attribués à chaque série d'oral (il y en a 4).

3 Écriture dans un fichier texte

EXERCICE 6 *On trouvera dans le fichier* `cadeau.py` *un programme testant la primalité des entiers. Constituer un fichier texte*[2] *contenant exactement tous les entiers majorés par* 10^4 *qui sont premiers.*

EXERCICE 7 *Constituer quatre fichiers regroupant les noms et prénoms des admissibles de chaque série des mines (cf. exercice 5).*

EXERCICE 8 *Si* `c` *est une chaîne de caractères,* `c.upper()` *renvoie une chaîne où toutes les minuscules de* `c` *ont été remplacées par des majuscules :*

```
>>> 'Hop'.upper()
'HOP'
```

Écrire une suite d'instructions copiant un fichier en changeant toutes les minuscules en majuscules.

On pourra appliquer ceci au fichier `.py` contenant ces instructions ! Vous aurez alors par exemple le fichier de travail `tp_fichier.py` recopié dans un fichier du genre `tp_fichier_majuscules.py`.

4 Quelques graphiques avec `matplotlib`

On commence avec un exemple simple en représentant le graphe de la fonction cosinus, avec quelques fioritures.

EXERCICE 9 *Taper les lignes suivantes dans le fichier python en cours, puis l'exécuter :*

1. Il s'agit des candidats au concours MP des mines 2013
2. De chemin relatif 'fichiers-sortie\premiers-10000.txt'

```
import matplotlib.pyplot as plt
import numpy as np


t = np.linspace(-np.pi,3*np.pi,1000)
y = np.cos(t)

plt.plot(t,y, label="cosinus")

plt.savefig('cosinus.pdf')
plt.show()
```

Observer les valeurs de t et y. Réfléchir à la signification de chaque ligne.

Note : il est d'usage de regrouper tous les `import ...` en début de fichier.

EXERCICE 10 *Dans les commandes de l'exercice précédent, insérer* progressivement *(avant le* `savefig`*) les lignes :*

```
plt.grid()
plt.axhline(color='black')
plt.axvline(color='black')
ys = np.sin(t)
plt.plot(t,ys, label="sinus")

plt.legend(loc='upper left')
```

Observer le résultat produit par chaque commande.

EXERCICE 11 *Essayer de produire un fichier* `pdf` *dont le contenu ressemble à celui donné en annexe.*

EXERCICE 12 *Le fichier* `Dosage.txt` *rassemble les différentes valeurs du pH mesuré lors du titrage d'un volume* $V_0 = 20$ *mL d'une solution ascorbique de concentration* C_a *inconnue diluée 5 fois par une solution de soude de concentration* $C_b = 1{,}011.10^{-1}\ mol.L^{-1}$.

1. *Ouvrir le fichier et examiner sa structure.*
2. *Construire la liste V des différents volumes de solution de soude versés et la liste pH des valeurs de pH correspondantes.*
3. *Créer la liste Der des valeurs de* $\frac{dpH}{dV}$ *à l'aide de la formule de dérivation numérique suivante :*

$$\frac{dpH}{dV}(V_i) = \frac{pH(V_{i+1}) - pH(V_{i-1})}{V_{i+1} - V_{i-1}}$$

4. *Tracer le graphique du pH et de sa dérivée en fonction du volume. Si* `x` *désigne la liste des valeurs de l'axe des abscisses,* `y1` *et* `y2` *les deux listes des ordonnées correspondantes, une méthode pour tracer un graphique avec deux axes des ordonnées est la suivante :*

```
fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.plot(x, y1, 'g-')
ax2.plot(x, y2, 'b-')

ax1.set_xlabel('X data')
ax1.set_ylabel('Y1 data', color='g')
ax2.set_ylabel('Y2 data', color='b')

plt.show()
```

5. *Déterminer le ou les volumes équivalents en utilisant la liste* `Der`*. Analyser le graphique. Conclure.*
6. *En déduire la concentration* C_a *de la solution d'acide ascorbique.*

5 Recherche dans un fichier

EXERCICE 13 Projet Euler 99

Comparer deux nombres écrits en notation exponentielle comme 2^{11} *et* 3^7 *n'est pas difficile, puisque n'importe quel calculateur confirmera que* $2^{11}2048 < 3^7 = 2187$.

Néanmoins, vérifier que $632382^{518061} > 519432^{525806}$ *est plus difficile car ces deux nombres contiennent plus de* 3 *millions de chiffres.*

Le fichier `base_exp.txt` *contient* 1000 *lignes avec une paire (base, exposant) par ligne. Les deux premières lignes représentent les deux nombres donnés en exemple.*

Déterminer le numéro de ligne contenant le nombre le plus grand.

EXERCICE 14 *Dans le cadre de sa réforme territoriale, l'empereur a décidé de diviser l'empire galactique en* 100 000 *départements. Les noms de chaque département sont constitués de cinq lettres majuscules choisies aléatoirement suivies du numéro de tirage. Les noms ont été stockés dans le fichier* `departements.txt`*, à raison de un nom par ligne.*

1. *Écrire une fonction* `extraire_departements(fichier)` *qui prend en paramètre le nom d'un fichier de noms de départements et qui retourne un tableau contenant ces noms classés dans l'ordre alphabétique.*

2. *Écrire une fonction* `creer_numero_departements(fichier, tab)` *qui prend en paramètres un nom de fichier et un tableau de noms de départements dans l'ordre alphabétique et qui crée un fichier dont chaque ligne contient un numéro de département (la numérotation commence à* 1*) et un nom séparés par une virgule :*

   ```
   1,AAABL54099
   2,AAAGC94645
   ```

   Pour trier sur place dans l'ordre croissant un tableau, on utilise la méthode `sort`.

   ```
   In [1]: t = ['AA2', 'AA1', 'AB3']

   In [2]: t.sort()

   In [3]: t
   Out[3]: ['AA1', 'AA2', 'AB3']
   ```

3. *Ecrire une fonction* `recherche_seq_numero_departements(t, nom)` *qui recherche par balayage le numéro d'un département à partir de son nom, dans un tableau de nom ordonné dans l'ordre alphabétique.*

4. *Ecrire une fonction* `recherche_dicho_numero_departements(t, nom)` *qui recherche par dichotomie le numéro d'un département à partir de son nom, dans un tableau de nom ordonné dans l'ordre alphabétique.*

   Estimer en fonction du nombre de départements, l'ordre de grandeur des complexités temporelles des fonctions des fonctions `recherche_seq_numero_departements` *et* `recherche_dicho_numero_departements` *pour la recherche du numéro du dernier département dans le même tableau ordonné ?*

   On pourra vérifier expérimentalement ces estimations en chronométrant les temps d'exécution des fonctions à l'aide de la fonction `chrono` *contenue dans* `cadeau.py`.

   ```
   In [111]: chrono(recherche_seq_numero_departements)(t,'ZZZYN56793')
   Valeur de retour de recherche_seq_numero_departements : 100000 en 0.019629 secondes

   In [112]: chrono(recherche_dicho_numero_departements)(t,'ZZZYN56793')
   Valeur de retour de recherche_dicho_numero_departements : 100000 en 0.000019 secondes
   ```

6 Encore des fichiers

EXERCICE 15 *Reprendre l'exercice 6 en mettant dix nombres premiers (séparés par des tabulations par exemple, ou des virgules...) par ligne.*

EXERCICE 16 *À l'aide du fichier créé dans l'exercice 15, calculer la somme des nombres premiers majorés par* 10000*... en seulement trois lignes, dont une pour ouvrir le fichier, et une autre pour le fermer !*

EXERCICE 17 *Représenter le « graphe discret » des* 11 *premiers termes de la suite u de premier terme* $u_0 = -8,43$ *et vérifiant la relation de récurrence* $u_{n+1} = -\frac{1}{2}u_n + 63$ *(cf. annexes).*

EXERCICE 18 d'après un exercice du site `http://www.france-ioi.org/`

1. *Créer un fichier* `points.txt` *contenant les coordonnées des* 101^2 *points à coordonnées entières dont l'abscisse et l'ordonnée appartiennent à l'intervalle* $[\![0;100]\!]$. *Chaque couple de coordonnées est enregistré sur une ligne au format* `100\t40\n` *pour le point de coordonnées* $(100,40)$ *par exemple. De plus les coordonnées doivent être stockées dans un ordre aléatoire. On utilisera la fonction* `shuffle` *du module* `random` *qui mélange sur place un tableau.*

```
>>> t = [k for k in range(10)]
>>> t
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> random.shuffle(t)
>>> t
[4, 0, 9, 6, 1, 5, 7, 2, 3, 8]
```

2. *Écrire une fonction* `extraire_points(fichier, n)` *qui prend en argument le nom d'un fichier de points (situé dans le même répertoire) et un nombre de lignes et qui retourne un tableau avec les coordonnées des points enregistrés sur les* `n` *premières lignes du fichier.*

3. *Compléter le code de la fonction* `crible(points)` *qui crée un tableau à deux dimensions (une matrice* 101×101*) nommé* `etat` *tel que* `etat[y][x]` *vaut* `True` *si le points de coordonnées* $(x;y)$ *appartient au tableau de coordonnées de points* `points` *et* `False` *sinon.*

```
def crible(points):
    etat = [[False]*101 for k in range(101)]
    for p in points:
        ...
    return etat
```

4. *Écrire une fonction* `nb_milieux(points)` *qui prend en argument un tableau* `points` *de coordonnées de points et qui retourne le nombre de points qui sont le milieu de deux autres points contenus dans* `points`.

   Pour repérer les points possibles et ne pas compter deux fois un milieu, on initialise un tableau de crible `etat` *avec la fonction* `crible` *et on change la valeur booléenne de* `etat[y][x]` *si le point de coordonnées* (x,y) *est un milieu déjà visité.*

```
>>> pts = extraire_points('points.txt', 500)
>>> len(pts)
500
>>> pts[:5]
[(24, 14), (4, 76), (78, 94), (30, 55), (37, 9)]
>>> nb_milieux(pts)
1522
```

EXERCICE 19 *Le fichier* `jocondeBW.jpg` *peut être manipulé par Python via la classe* `Image` *du e module* `PIL` *: on l'importe d'abord via* `from PIl import Image` *puis on crée un objet associé à ce fichier via la commande* `Mona = Image.open('jocondeBW.jpg')`

L'objet `Mona` *contient entre autres une méthode* `histogram`. *Observer le résultat renvoyé par cette méthode... et représenter l'histogramme de la Joconde, ainsi que l'histogramme cumulé (ce qui induit deux normalisations différentes de l'axe des y).*

On sauvera également l'histogramme dans un fichier `csv` *« human readable ».*

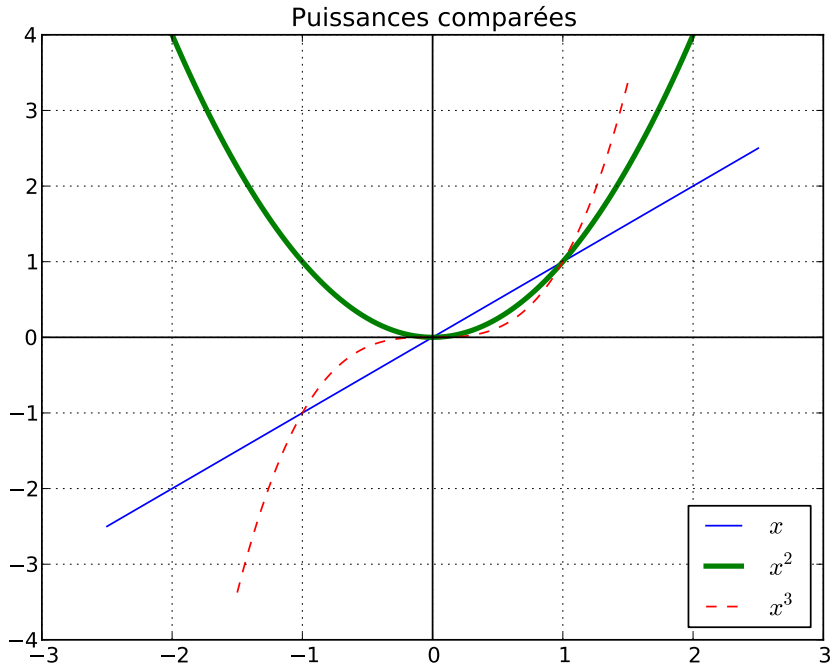
7 Besoin d'indications ?

- Exercice 5. On peut créer un tableau `series` de quatre entiers représentant le nombre d'admissibles aux différentes séries (en faisant attention au décalage : `series[i]` contient le nombre d'admissibles à la série $i+1$). Pour incrémenter le compteur correspondant à la série présente dans la chaîne `ch`, on pourra par exemple exécuter : `series[ int(ch) - 1] += 1` Cette même chaîne aura probablement été obtenue via quelque chose comme `ligne.strip().split('\t')[3]` !
- Exercice 7. On peut créer un tableau de 4 fichiers en une seule étape, via :
  `fichiers = [open('admissibles-serie-'+str(i+1)+'.txt','w') for i in range(4)]`
- Exercice 11. Les mots clés suivants pourront servir, pour les fioritures :

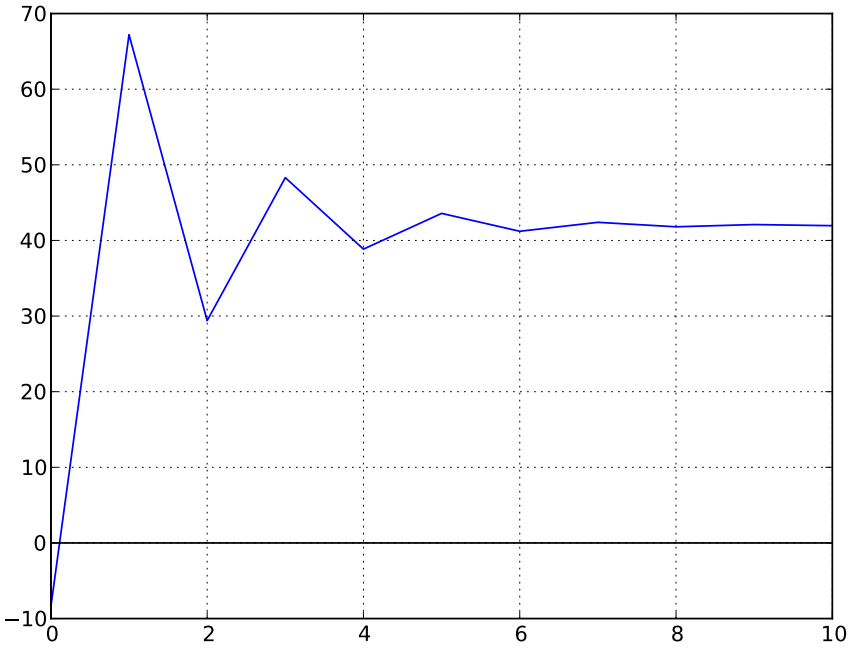
  `title;  linewidth=;  '--'; 'lower right'`

8 Annexe : quelques jolis graphiques

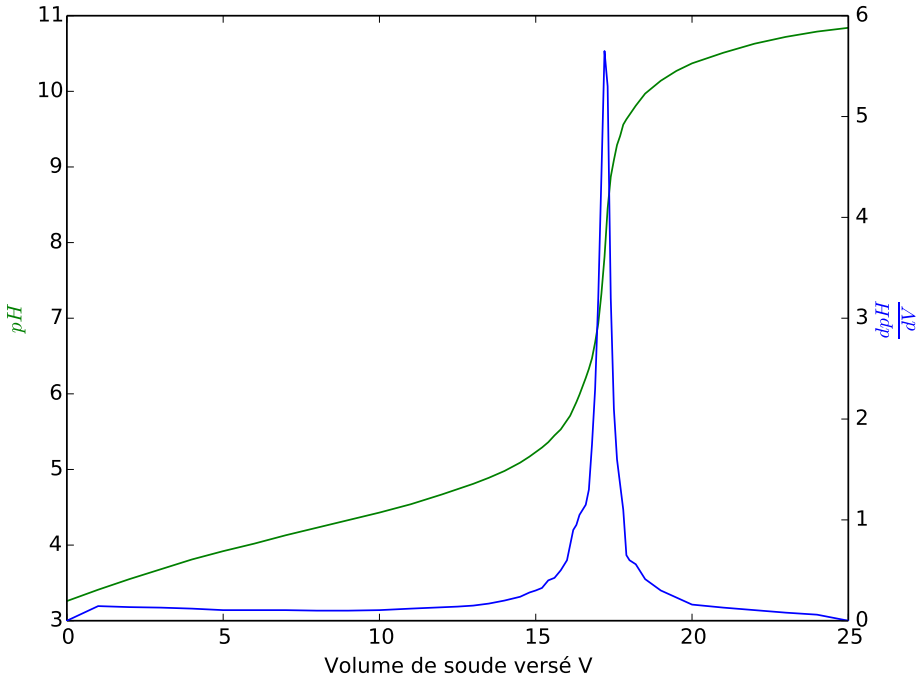
Exercice 11



Exercice 17



Exercice 12



Exercice 19