Corrigé du TP 07 Fichiers et Graphiques avec `matplotlib`

Chenevois-Jouhet-Junier-Rebout

1 Outils

Imports des bibliothèques/modules nécessaires :

```
import math, random
import numpy as np                #module avec les array qui sont des tableaux optimisés
import matplotlib.pyplot as plt   #module avec des fonctions graphiques
import sys                        #accès au système (sortie standard  sys.sdtout, entrée standard sys.stdin)
import time                       #module avec des fonctions de mesure de temps
```

Pour mesurer le temps d'exécution d'un appel de fonction on utilisera la fonction `time` ou la fonction `perf_counter` du module `time`.

`time.time()` mesure le nombre de secondes écoulés depuis Epoch, l'origine des temps informatiques, fixée le 01/01/1970 à 00:00:00 en temps UTC.

`time.perf_counter()` retourne le temps écoulé depuis le premier appel de `time.perf_counter()`. C'est un outil de mesure plus précis, pour mesurer un temps plus court.

```
"""
In [28]: debut = time.perf_counter() ; t = [i for i in range(10**7)] ; duree = time.perf_counter() - debut

In [29]: duree
Out[29]: 0.6903144909997536
"""
```

Voici une fonction permettant d'exécuter la fonction qui lui est passée en argument et d'afficher le temps d'exécution.

```
def chrono(fonction):
    '''Fonction de chronométrage de la fonction passée en paramètre
    Pour exécuter et chronométrer f(*args) saisir chrono(f)(*args) '''

    def fonction2(*args):
        chrono_debut = time.perf_counter()
        rep = fonction(*args)
        chrono_fin = time.perf_counter()
        print("Temps d'exécution de {:s} : {:1.6f} secondes".format(\
        fonction.__name__, chrono_fin - chrono_debut))
        return rep

    return fonction2

"""
In [33]: def f(n):
    ...:     return [i for i in range(n)]
    ...:
In [34]: t = chrono(f)(10**7)
Temps d'exécution de f : 0.583690 secondes
"""
```

2 Lecture d'un fichier texte

2.1 Exercice 2

```
def nb_ligne(fichier):
    c = 0
    f = open(fichier, 'r')
    for ligne in f:
        c += 1
    f.close()
    return c

"""
In [7]: nb_ligne('dico.txt')
Out[7]: 336530
"""

def matchFirstCharPrenom(fichier, prenom):
    firstChar = prenom.lower()[0]
    c = 0
    f = open(fichier, 'r')
    for ligne in f:
        if ligne.lower()[0] == firstChar:
            c += 1
    f.close()
    return c

def matchLastCharPrenom(fichier, prenom):
    lastChar = prenom.lower()[-1]
    c = 0
    f = open(fichier, 'r')
    for ligne in f:
        if ligne.rstrip().lower()[-1] == lastChar:
            c += 1
    f.close()
    return c

"""
>>> matchFirstCharPrenom('dico.txt','frédéric')
12485

>>> matchLastCharPrenom('dico.txt','frédéric')
179
"""
```

2.2 Exercice 3

```
def somme_premiers_fichier(fichier):
    '''retourne la somme des entiers premiers successifs stockés dans
    le fichier dont le fichier d'accès est passé en paramètre'''
    mon_fichier = open(fichier,'r') # 'r'  pour lecture
    s = 0
    for L in mon_fichier:
        #penser à nettoyer la ligne (enlever les caractères de fin de ligne)
        s += int(L.rstrip())
    mon_fichier.close()
    return s

"""
```

```
In [1]: somme_premiers_fichier('premiers-1000.txt')
Out[1]: 3682913
"""
```

2.3 Exercice 4

```
def creer_fichier_scores(nom, nbscores):
    """Crée un fichier de scores réalisés lors de parties d'un jeu video
    par 26 joueurs possibles de nom compris entre 'A' et 'Z'.
    Format d'une ligne 'nom,score\n'"""
    joueurs = [chr(ord('A') + k) for k in range(26)]
    f = open(nom, 'w')
    for k in range(nbscores):
        f.write(random.randint(0,12)*' ' +
        joueurs[random.randint(0, 25)]+','+ str(random.randint(0, 1000)) + '\n')
    f.close()

def extraire_scores(fichier):
    f = open(fichier, 'r')
    score_joueur = [0]*26
    for ligne in f:
        joueur, score = ligne.strip().split(',')
        score_joueur[ord(joueur) - ord('A')] += int(score)
    return score_joueur

def extraire_moyennes(fichier):
    f = open(fichier, 'r')
    score_joueur = [0]*26
    nbpartie_joueur = [0]*26
    moyenne_joueur = [0]*26
    for ligne in f:
        joueur, score = ligne.strip().split(',')
        delta  = ord(joueur) - ord('A')
        score_joueur[delta] += int(score)
        nbpartie_joueur[delta] += 1
    for k in range(26):
        moyenne_joueur[k] = score_joueur[k]/nbpartie_joueur[k]
    return moyenne_joueur
```

2.4 Exercice 5

```
def admissibles(fichier):
    '''ouvre un fichier d'admissibilité et retourne une liste de 5 éléments
    constituée du  toral  d'admissibles et des admissibles par série d'oral'''
    mon_fichier = open(fichier,'r') # 'r' pour lecture
    decompte = [0]*5
    for ligne in mon_fichier:
        ligne = ligne.rstrip() #nettoyage des fins de ligne
        #on récupère les champs séparés par des tabulations dans une liste
        champs = ligne.split('\t')
        if champs[-1]!='-':
            #incrémentation du total d'admissibles
            decompte[0] += 1
            #puis de la série d'oral correspondante
            decompte[int(champs[-1])] += 1
    mon_fichier.close()
    return decompte
```

```
"""
In [2]: admissibles('admissibles.txt')
Out[2]: [1718, 417, 417, 410, 474]
"""
```

3 Écriture dans un fichier texte

3.1 Exercice 6

```
def est_premier(n):
    if n <= 1:
        return False
    for d in range(2, int(math.sqrt(n))+1):
            if n%d == 0:
                return False
    return True

def premiers_fichier(fichier,n):
    '''Ecrit tous les entiers premiers majorés par n>=2
    dans un fichier texte'''
    mon_fichier = open(fichier,'w')
    assert n>=2 #n doit etre supérieur ou égal à 2
    entier = 2
    while entier < n:
        if est_premier(entier):
            mon_fichier.write(str(entier)+'\n')
        entier += 1
    mon_fichier.close()
"""
In [6]: premiers_fichier('premiers.txt', 10**4)
"""
```

3.2 Exercice 7

```
def tri_admissibles(fichier):
    '''A partir d'un fichier d'admissibles, crée quatre fichiers regroupant les
    noms et prenoms des  admissibles selon leur série à l'oral. '''
    #la fonction os.path.join permet de construire une chemin d'accès avec le séparateur
    #de fichier propre au système '\' pour Windows ou '/' pour Linux

    fichier_source = open(fichier,'r')
    #liste avec les descripteurs de fichiers ouverts en écriture (un par série)
    oral = [open(os.path.join('fichiers-sortie','admissible%s.txt'%i),
    'w') for i in range(1, 5)]
    for ligne in fichier_source:
        ligne = ligne.rstrip()
        champs = ligne.split('\t')
        admissible = champs[-1]
        if admissible != '-':
            oral[int(admissible)-1].write(champs[-3]+'\n')
    for fichier in oral:
        fichier.close()
    fichier_source.close()

"""
```

```
In [9]: tri_admissibles('fichiers-entree/admissibles.txt')
"""
```

3.3 Exercice 8

```
def upper_fichier(fichier):
    '''Transforme en majuscules toutes les caractères d'un fichier'''
    racine, extension = os.path.splitext(fichier)
    mon_fichier = open(fichier,'r')
    #ouverture du fichier qui contiendra le texte en majuscules
    fichier_majuscule = open(os.path.join('fichiers-sortie',
    racine+'-upper'+extension),'w')
    tampon = mon_fichier.read()
    tampon = tampon.upper()
    fichier_majuscule.write(tampon)
    mon_fichier.close()
    fichier_majuscule.close()

"""
In [13]: upper_fichier('correcTP7Fichiers.py')
"""
```

4 Quelques graphiques avec `matplotlib`

4.1 `matplotlib` est une immense bibliothèque, où trouver de l'aide ?

- Documentation officielle sur `matplotlib.pyplot` : http://matplotlib.org/api/pyplot_summary.html
- Un tutoriel en anglais sur `numpy` et `scipy` (bibliothèque de calcul scientifique plus vaste incluant `numpy`) : http://wiki.scipy.org/Tentative_NumPy_Tutorial
- Un tutoriel en français sur `matplotlib.pyplot` : http://www.loria.fr/~rougier/teaching/matplotlib/
- Les scipy lecture, des tutoriel en anglais sur `numpy`, `scipy` et `matplotlib` : http://scipy-lectures.github.io/.

4.2 Exercices 9 et 10

```
def exo9():
    import matplotlib.pyplot as plt
    import numpy as np


    def exo10():
        """Code de l'exo 10"""
        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')

    #tableau de 1000 valeurs prises entre -pi et 3*pi compris avec un pas régulier
    t = np.linspace(-np.pi,3*np.pi,1000)
    print('type de la variable t :' ,type(t), '\nMéthodes de l\'objet : ',dir(t))
    #t est un array numpy type particulier de tableau
    #on utilise la fonction cos du module numpy car elle est vectorialisée
    #on peut l'appliquer à un tableau (elle s'applique à chaque élément)
    y = np.cos(t)
    plt.plot(t,y, label='cosinus')
```

```
    #on peut commenter la ligne suivante pour ne pas l'exécuter
    exo10()
    #toujours appeler savefig avant show sinon sauvegarde vide !!!
    plt.savefig('cosinus-sinus.pdf')
    plt.show()

"""
In [22]: exo9()
"""
```

- Type de la variable `t` : `<class 'numpy.ndarray'>`
- Méthodes de l'objet : `['T', '__abs__', '__add__', '__and__', '__array__', ..., 'var', 'view']`

4.3 Exercice 11

```
def exo11(xmin,xmax,ymin,ymax):
    '''Courbes des fonctions x->x, x->x^2 et x->x^3 dans la fenetre
    [xmin,xmax]x[ymin,ymax]'''
    import matplotlib.pyplot as plt
    import numpy as np
    import os.path
    plt.axis([xmin,xmax,ymin,ymax])
    t = np.linspace(xmin,xmax,1000)
    style = ['-','-','--']
    couleur = ['red','green','blue']
    largeur=[2,4,1]
    for i in range(3):
        plt.plot(t,t**(i+1), linewidth=largeur[i], linestyle=style[i],
        color=couleur[i])
    plt.axhline(color='black')
    plt.axvline(color='black')
    plt.legend([r'$x$',r'$x^2$',r'$x^3$'],loc='lower right')
    plt.title('Puissances comparées')
    plt.savefig('puissances_comparees.pdf')
    plt.show()
```

4.4 Exercice 12

```
def exo12():
    import matplotlib.pyplot as plt
    V, pH, Der = [],[],[0]
    veq = []

    # Lecture du fichier et "construction" des listes V et pH

    monfichier = open('Dosage.txt','r')

    monfichier.readline()
    for l in monfichier :
        v,ph = l.rstrip().split('\t')
        V.append(float(v))
        pH.append(float(ph))

    monfichier.close()

    # Calcul de la dérivée et "construction" de la liste Der

    for i in range (1, len(V)-1):
```

```
        Der.append((pH[i+1]-pH[i-1])/(V[i+1]-V[i-1]))
    Der.append(0)

    # Tracé des graphiques

    fig, ax1 = plt.subplots()
    ax2  =  ax1.twinx()
    ax1.plot(V, pH, 'g-')
    ax2.plot(V, Der,'b-')
    ax1.set_xlabel('Volume de soude versé V')
    ax1.set_ylabel('$pH$', color='g')
    ax2.set_ylabel(r"$\frac{dpH}{dV}$", color='b', fontsize = 16)

    plt.savefig("Fig dosage.pdf")
    plt.show()

    # Détermination des volumes équivalents "théoriques" par recherche des maxima locaux

    for i in range(1,len(Der)-1):
        if Der[i] > Der[i-1] and Der[i] > Der[i+1]:
            veq.append(V[i])

    if len(veq) == 0 :
        print ("Pas de volume équivalent")
    elif len(veq) == 1 :
        print("Un seul volume équivalent : Véq = ",veq[0],"mL")
    else :
        print("Les volumes équivalents sont :\n")
        for i in range(len(veq)):
            print("Véq(",i+1,") = ",veq[i],"mL")
```

L'analyse du graphique montre qu'il n'y a qu'une seule équivalence, les autres maxima locaux correspondant à des artéfacts

Véq = 17,2 mL

Relation à l'équivalence $\mathrm{Ca}/5\,V0 = Cb\mathrm{Véq}$ d'où $\mathrm{Ca} = (5Cb\mathrm{Véq})/\mathrm{V0} = 0{,}435$ mol/L

5 Recherche dans un fichier

5.1 Exercice 13

```
def projet99():
    from math import log
    parse = lambda ligne : list(map(int, ligne.rstrip().split(',')))
    f = open('base_exp.txt','r')
    t = list(map(parse,  f.readlines()))
    f.close()
    maxi = float('-inf')
    n = 1
    for base, exp in t:
        m = exp * log(base)
        if exp * log(base) > maxi:
            nmaxi = n
            maxi = m
        n += 1
    return nmaxi
    """
In [3]: projet99()
Out[3]: 709
```

```
"""
```

5.2 Exercice 14

```
def creer_departements(fichier, n):
    '''Cree un fichier texte contenant n noms de départements
    de format *****numero_ligne où * est une lettre majuscule.'''
    debut = ord('A')
    fin = ord('Z')
    f = open(fichier, 'w')
    for k in range(1, n+1):
        f.write(''.join(chr(random.randint(debut, fin)) for j in range(5)) + \
        str(k) + '\n')
    f.close()

def extraire_departements(fichier):
    '''Extrait dans un tableau les noms de départements contenus sur
    chaque ligne du fichier de format nom_departement et trie le tableau
    dans l'ordre croissant.'''
    f = open(fichier, 'r')
    t = [ligne.rstrip() for ligne in f]
    t.sort()
    return t

def creer_numero_departements(fichier, tab):
    '''Recopie les noms de fichier du tableau tab  sont dans l'ordre croissant,
    dans un fichier en formatant chaque ligne ainsi : numero (de 1 ),
     nom_departement'''
    g = open(fichier, 'w')
    for k in range(len(tab)):
        g.write( str(k+1) +  ',' + tab[k] + '\n')
    g.close()

def recherche_dicho_numero_departements(t, nom_dep):
    '''Recherche dichotomique d'un numéro de département dans un tableau de noms
    de départements triés dans l'ordre croissant.
    Retourne -1 si le département n'existe pas.'''
    debut = 0
    fin = len(t) - 1
    while debut <= fin:
        med = (debut + fin)//2
        target = t[med]
        if target == nom_dep:
            return med + 1
        elif target < nom_dep:
            debut = med + 1
        else:
            fin = med - 1
    return -1

def recherche_seq_numero_departements(t, nom_dep):
    '''Recherche séquentielle d'un numéro de département dans un tableau de noms
    de départements triés dans l'ordre croissant.
    Retourne -1 si le département n'existe pas.'''
    for k, element in enumerate(t):
        if element == nom_dep:
            return k + 1
    return -1
```

Dans **le pire des cas** (recherche de l'élément en dernière position dans le tableau ordonné), la complexité d'un algorithme de

recherche séquentielle (ou par balayage) est **linéaire**, de l'ordre de la taille du tableau.

La complexité d'un algorithme de **recherche dichotomique** est **logarithmique**, de l'ordre du logarithme de la taille du tableau, quelle que soit la position de l'élément recherché

6 Encore des fichiers

6.1 Exercice 15

```
def premiers_par_ligne(fichier, n, m):
    '''Crée un fichier texte puis écrit tous les entiers premiers majorés par
    n>=2 avec m entiers premiers par ligne (par exemple m =10)'''
    mon_fichier = open(fichier,'w')
    assert n>=2 #n doit etre supérieur à 2
    nbpremier = 0
    entier = 2
    while entier<n:
        if est_premier(entier):
            mon_fichier.write(str(entier)+'\t')
            nbpremier += 1
            #si m divise entier on a  rempli une ligne avec m  premiers
            if nbpremier%m == 0:
                mon_fichier.write('\n')
        entier += 1
    mon_fichier.close()

"""
In[32]:premiers_par_ligne('premiers-1000-10ligne.txt',1000,10)
"""
```

6.2 Exercice 16

```
def somme_premiers_fichier2(fichier):
    """retourne la somme des 1000 plus petits entiers premiers contenus
    dans un fichiert passé en paramètre.
    Une ligne peut contenir plusieurs entiers premiers  séparés par des '\t'"""
    f = open(fichier,'r')
    tabprem = [int(n) for ligne in f for n in ligne.rstrip().split('\t')]
    f.close()
    return sum(tabprem)

"""
In [40]: somme_premiers_fichier2('premiers-1000.txt')
Out[40]: 5736396
"""
```

6.3 Exercice 17

```
def exo17():
    '''Graphe discret des 11 premiers termes de la suite u définie par
    u(0)=-8.43 et u(n+1)=-1/2*u(n)+63'''
    import matplotlib.pyplot as plt
    import numpy as np
    import os.path
    #liste des indices
```

```
    n = np.arange(0,11)
    #calcul de la liste des termes
    u = [-8.43]
    for i in range(10):
        u.append(-0.5*u[-1]+63)
    plt.plot(n,u,linestyle='--',color='red',marker='o')
    plt.axhline(color='black')
    #la suite converge vers 42
    plt.axhline(42,color='blue')
    #chaine précédée d'un r pour raw, ainsi les caractères spéciaux de python
    #ne sont pas interprétés, le code Latex pour l'affichage mathématique l'est
    plt.legend([r'$u_{n+1}=-\frac{1}{2}u_{n}+63$ et $u_{0}=-8.43$'],
    loc='lower right')
    plt.title('Evolution d\'une suite')
    plt.savefig('exo17-suite.pdf')
    plt.show()
```

6.4 Exercice 18

```
def creer_fichier_points(nom):
    '''Crée un fichier texte avec les coordonnées de n points à coordonnées entières
    .Chaque ligne est de la forme : 100\t40\n pour un point de coordonnées (100,40).
    0<= x <= 100 et 0<= y <= 100.
    '''
    f = open(nom,'w')
    points = [(x, y) for x in range(101) for y in range(101)]
    random.shuffle(points)
    for p in points:
        f.write(str(p[0])+'\t'+str(p[1])+'\n')
    f.close()

def extraire_points(fichier, n):
    '''Retourne un tableau avec les coordonnées des points enregistrés sur les
    n premières lignes du fichier'''
    f = open(fichier)
    pts = []
    for k in range(n):
        ligne = f.readline()
        coord = tuple(map(int, ligne.split('\t')))
        pts.append(coord)
    f.close()
    return pts

def crible(points):
    '''Retourne un crible d'un tableau etat de dimensions 101x101
    avec t[y][x] = True si le point de coordonnées (x,y) appartient
    au tableau points et t[y][x] = False sinon'''
    N = 101
    etat = [[False]*N for k in range(N)]
    for p in points:
        x, y = p
        etat[y][x] = True
    return etat

def nb_milieux(points):
    '''Retourne le nombre de points qui sont milieu de deux autres points
    dont les coordonnées sont contenues dans le tableau points'''
    nbmilieux = 0
    nbpoints = len(points)
```

```
    etat = crible(points)
    #decompte des milieux, boucles sur les couples de points
    for k in range(nbpoints):
       xa, ya = points[k]
       for j in range(k+1, nbpoints):
          xb, yb = points[j]
          if (xa + xb)%2 == 0 and (ya + yb)%2 == 0:
             xm, ym = (xa + xb)//2, (ya + yb)//2
             if etat[ym][xm]:
                nbmilieux += 1
                etat[ym][xm] = True
    return nbmilieux
```

6.5 Exercice 19

```
def exo19(fichier):
    '''Distribution des valeurs des pixels d'une image en niveaux de gris'''
    from PIL import Image
    import matplotlib.pyplot as plt
    import numpy as np
    Mona = Image.open(fichier)
    #affichage du mode (ici 'L' pour niveaux de gris) et du format de l'image
    #ey de la taille de l'image (Largeur,Hauteur)
    print(Mona.mode,Mona.format,Mona.size)
    L,H = Mona.size
    histo = Mona.histogram()
    histocumul = [histo[0]]
    #il y a 256 nuances de niveau de gris len(histo) == 256
    for i in range(1, 256):
        histocumul.append(histocumul[-1]+histo[i])
    #normalisation des histogrammes
    max1,max2 = list(map(max,(histo,histocumul)))
    histonorm = [float(f)/max1 for f in histo]
    histocumulnorm = [float(g)/max2 for g in histocumul]
    #Graphique
    plt.axis([0,256,0,1])
    plt.plot(np.arange(256),histonorm,color='green')
    plt.plot(np.arange(256),histocumulnorm,color='blue')
    plt.ylabel(u'Distribution, somme normalisée')
    plt.xlabel(u'Niveaux de gris')
    plt.title(u'Distribution des pixels pour la Joconde')
    plt.savefig('DistributionPixelsJoconde.pdf')
    #plt.show()
    return histo, histocumul
```