Chapitre 4 : Lecture d'un fichier et premiers algorithmes d'application

I) Lecture d'un fichier

1) Chemin d'accès

Quand on ferme Python après l'exécution d'un programme, aucune des variables n'est sauvegardée. L'enregistrement dans des fichiers externes peut être un bon mieux de conserver des données, pour les exploiter ultérieurement ou avec un autre logiciel.

Si on crée un jeu par exemple, le score ou le niveau du joueur doit être conservé pour qu'il ne reparte pas de zéro.

Pour éditer un fichier en Python, on utilise la fonction **open()**.

Cette fonction prend 2 paramètres en entrée :

- le premier est le **chemin d'accès** du fichier dans l'arborescence,

- le second est le **type d'ouverture**.

Les deux paramètres sont déclarés entre guillemets (simples ou doubles), comme une chaîne de caractères.

Prop : On distingue chemin **relatif** et chemin **absolu**.

- Le chemin relatif concerne le dossier dans lequel se trouve votre fichier exécutable Python (suffixe .py). C'est le dossier de travail : si on ne précise rien de plus (exemple : « **'blabla.txt'** »), Python va opérer sur un fichier « blabla.txt » dans le même dossier que le programme lui-même.
- Le chemin absolu est celui de l'arborescence systématique de votre système de fichiers (exemple pour le même fichier : « **C:\Users\lucdu\OneDrive - Lycée Gustave Eiffel Dijon\Cours Luc\cours info TSI\ blabla.txt** »). Le chemin absolu permet d'accéder et d'opérer sur n'importe quel fichier présent dans la mémoire.

Rq : En général, (quitte à copier-coller) les fichiers, on enregistrera nos fichiers dans le même répertoire que python pour ne pas avoir à préciser le chemin complet.

2) Ouverture d'un fichier

prop : La fonction **open(*nom_du_fichier*)** seule ne réalise rien de visible.

```
In [63]: fichier=open('blabla.txt')

In [64]: print(fichier)
<_io.TextIOWrapper name='blabla.txt' mode='r' encoding='cp1252'>
```

On constate cependant que la fonction **print()** ne permet pas de lire le contenu du fichier, mais des renseignements sur le fichier et son encodage. Il est donc nécessaire d'affecter à une variable le résultat de cette ouverture.

Prop : Le second paramètre de la fonction **open()** précise le mode d'ouverture :

- **'r'** pour *read*, soit en lecture seule,

- **'w'** pour *write*, soit en écriture : si le fichier n'existe pas, Python le crée. A chaque ouverture de ce type, le contenu du fichier est écrasé !

- **'a'** pour append, soit en écriture en mode ajout : si le fichier n'existe pas, Python le crée.

Si on ne précise rien, Python ouvre le fichier en lecture seule. Cela sera très souvent suffisant.

Rq : Comme tout élément ouvert, il faut refermer un fichier avec Python après utilisation. Si on ne le fait pas, certains problèmes peuvent survenir (dans Python ou dans le système d'exploitation).

La fonction **close()** réalise cette opération.

Dans spyder, vous avez accès en haut à gauche au chemin d'accès de vos programmes. Il suffit alors d'ouvrir le document et d'y coller les différents documents (texte, csv, jpeg…) que vous voulez traiter en python. Notons que pour certains types de fichier il sera nécessaire de commencer par importer le module adéquat. (import csv par exemple pour traiter les fichiers .csv)

Rq : Lors de l'ouverture d'un fichier, seuls les caractères apparaissent, les mises en forme ne sont (à priori) pas conservées. Cela nécessitera un peu de travail pour reconstituer des tableaux par exemple (on le verra plus tard).

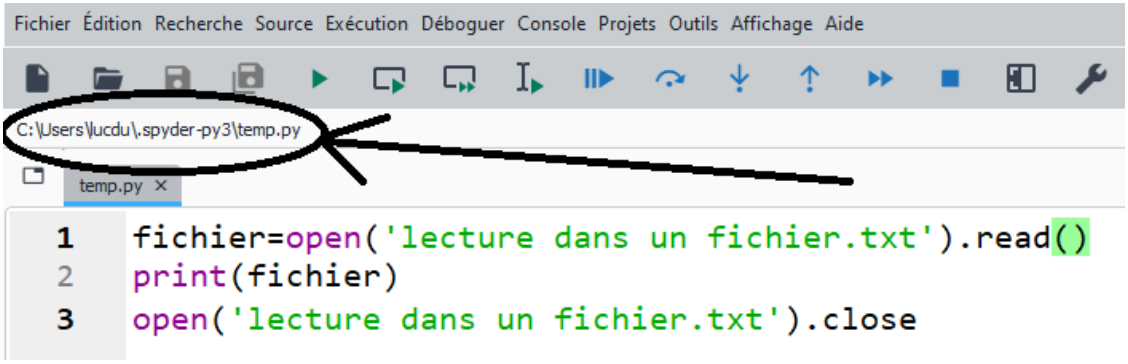
Pour le moment, nous travaillerons avec des fichiers au format .txt

Def : L'instruction « .read() » à la suite de l'instruction « open('nom_du_fichier') » permet de lire et d'afficher dans la console le contenu du fichier nommé.

rq : Si on ne veut afficher que les n premiers caractères, on peut utiliser « .read(n) ».

Exemple :

J'ai créé un fichier au format txt intitulé « lecture dans un fichier » et je l'ai enregistré le document contenant mon fichier python ouvert (adresse entourée ci-dessous)



Cette instruction va afficher dans la console de spyder le contenu de mon fichier.

II) Recherche des mots dans un texte

1) syntaxe

La manipulation de chaînes de caractères possède des points communs avec celle des listes, et quelques différences :

- la création d'une chaîne de caractères se fait entre guillemets,
- « + » entre deux chaînes de caractères réalise une concaténation
- les éléments d'une chaîne sont indicés de 0 à len(chaîne)-1,
- on peut accéder au i-ème élément d'une chaîne a avec $a[i]$ mais pas le modifier directement,
- on peut extraire une partie d'une chaîne avec $a[i:j]$ (i inclus, j exclus),
- $a[:i]$ renvoie la partie de a située avant l'indice i (exclus), $a[i:]$ renvoie la partie de a située après l'indice i (inclus).

Exemple :

```
7   a='salut'
8   b=' à tous'
9   c=a+b
10  print(c) # affiche : salut à tous
11  d='a'+'b'
12  print(d) #affiche : ab
13  e=2*b
14  print(e) #affiche : à tous à tous
15  f=a[2:5]
16  print(f) #affiche : lut
17  g=b[:4]
18  print(g) #affiche :  à t
```

2) recherche d'un mot dans un texte.

L'objectif de cette partie est d'écrire un programme capable de chercher un mot dans un texte, (et éventuellement de compter combien de fois ce mot apparait).

a) Ecrire une fonction nommée « trouve » qui a 3 entrées : le nom du fichier texte dans lequel on cherchera le mot, la chaine de caractère (le mot) à trouver ainsi que la position de départ dans le texte complet et qui renvoie 1 si le mot cherché commence effectivement bien à la position donnée et qui renvoie 0 sinon.

b) A l'aide de la fonction précédente et d'une boucle pour, écrire un programme qui possède en entrée le nom d'un fichier texte, ainsi qu'une chaine de caractère (un mot) à trouver et qui renvoie en sortie le nombre de fois que ce mot apparait dans le texte.

```
1   def trouve(texte,mot,rang):
2       fichier=open(texte).read()
3       rep=0
4       if fichier[rang:rang+len(mot)]==mot:
5           rep=1
6       open(texte).close
7       return(rep)
8
9   print(trouve("lecture dans un fichier.txt","bonjour",0))
10
11  def comptemot(texte,mot):
12      fichier=open(texte).read()
13      s=0
14      for k in range(0,len(fichier)-1):
15          s=s+trouve(texte,mot,k)
16      open(texte).close
17      print(mot, " apparait ",s, " fois ")
18
19  print(comptemot("lecture dans un fichier.txt","lo"))
```

Réponse :

Voici ce que j'ai enregistré dans mon fichier .txt et la réponse obtenue sur spyder.

bonjour à tous les TSI1
hello
hello
hello

```
In [1]: runfile('C:/Users/lucdu/.spyder-py3/temp.py', wdir='C:/Users/lucdu/.spyder-py3')
1
lo  apparait  3  fois
None
```

Rq : le « 1 » est le résultat du premier print (pour tester la fonction trouve). Le « none » à la fin signale que la seconde fonction ne renvoie aucune valeur (elle ne comporte pas de return). Elle affiche simplement le message écrit.

3) Complexité

Def : On appelle complexité temporelle d'un algorithme la donnée d'une information sur le temps d'exécution de l'algorithme, en fonction de la taille n des données à traiter. L'objectif sera de diminuer au maximum la complexité d'un algorithme pour le rendre plus efficace.

Rq : On se place dans le cas idéal, et la complexité temporelle d'un programme (qui est propre au programme) ne dépendra donc ni de l'ordinateur utilisé, ni du logiciel de programmation employé. Bien qu'en réalité ces éléments aient un impact sur le temps réel nécessaire pour réaliser l'algorithme.

Def : On peut considérer comme opérations élémentaires les opérations suivantes :

+ ; - ; * ; / ; // ; % ; < ; > ; == ; != ; input ; open ; print ; close ; = (affectation)

Prop : Déterminer la complexité temporelle d'un programme revient à donner une estimation (dans le pire des cas) du nombre d'opérations élémentaires qu'il devra traiter. On ne cherchera pas à déterminer le nombre exact d'opérations élémentaires, mais une bonne estimation de celui-ci.

Def : (Notation de Landau)

Etant données deux suites numériques (u_n) et (v_n) ne s'annulant pas. On dit que (u_n) est dominée par (v_n) lorsque $n \to +\infty$ et on note $u_n = O(v_n)$ si $\lim\limits_{n\to+\infty} \frac{u_n}{v_n} < +\infty$

Exemple : $n + 1 = O(n^2)$ car $\lim\limits_{n\to+\infty} \frac{n+1}{n^2} = 0 < +\infty$

$2n = O(n)$ car $\lim\limits_{n\to+\infty} \frac{2n}{n} = 2 < +\infty$

Rq : être dominé par ne signifie donc pas être inférieur à…

Def : Etant donné un algorithme, on note $T(n)$ sa complexité temporelle. On dit que sa complexité temporelle est :

- constante lorsque $T(n) = O(1)$
- logarithmique lorsque $T(n) = O(\ln(n))$
- linéaire lorsque $T(n) = O(n)$
- quasi linéaire lorsque $T(n) = O(n \times \ln(n))$

- quadratique lorsque $T(n) = O(n^2)$
- polynomial lorsque $T(n) = O(n^p)$ pour un entier $p \geq 2$
- exponentielle lorsque $T(n) = O(a^n)$ pour un réel $a > 1$

Exemples :

a)
```
1   def suite(n):
2       for k in range(0,n+1):
3           print(k)
```

Cet algorithme qui affiche les entiers de 0 à n réalisera $n + 1$ opérations élémentaires (les $n + 1$ affichages). Sa complexité temporelle est donc linéaire et $T(n) = O(n)$

b)
```
1   def suite(n):
2       for k in range(0,n+1):
3           for i in range (0,n+1):
4               print(i,k)
```

Cet algorithme qui affiche tous les couples d'entiers compris entre 0 et n réalisera $(n + 1)^2$ opérations élémentaires (on réalise $n + 1$ affichage pour chaque passage dans la première boucle, et il y a $n + 1$ passages dans la première boucle). Ainsi $T(n) = O(n^2)$

c) L'algorithme de recherche de mots dans un texte vu en 2) a une complexité linéaire (par rapport à la taille totale du texte). En effet si on note m la taille du mot cherché et n la taille du texte complet, alors la complexité de la fonction trouve est en $O(m + 4) = O(m)$ puisqu'on compare simplement deux chaines de m caractères plus 4 opérations au maximum (ouverture et fermeture du fichier, et 1 ou 2 affectations). Puis on appelle n fois cette fonction dans la fonction « comptemot » et pour chaque appel, on réalise en plus une somme et une affectation. Enfin, on réalise en plus une affectation ($s = 0$) et 1 ouverture et 1 fermeture de fichier. On a donc une complexité en $O(m \times n + 2n + 3) = O\big(n(m + 2)\big) = O(n)$ (puisque $m + 2$ est constant et petit)

III) Représentation des données

On peut créer des courbes, graphs et histogrammes (entre autres) en python.
Comme pour tracer n'importe quel graphique en python, il est nécessaire d'importer le module « matplotlib »
L'instruction « import matplotlib.pyplot as graphe1 » permet de définir l'objet (nommé ici graphe1) comme étant un graphique dont on pourra ensuite préciser les paramètres ensuite.

Pour créer un histogramme en python, il faut créer 2 listes. La première sera la liste des valeurs indiquées en abscisse, ou étiquettes (qui peuvent être numériques ou non). La seconde sera la liste des valeurs correspondant (dans le même ordre) aux étiquettes de la première liste. Ces valeurs seront indiquées en ordonnées.

L'instruction «ax.bar(liste_étiquettes, liste_valeurs) » va alors créer l'histogramme correspondant aux étiquettes et aux valeurs indiquées dans les deux listes mentionnées.

On peut également ajouter un titre au graphique, un nom pour chaque axe etc…

Rq : ne pas oublier de finir par l'instruction « nom_du_graphique.show() » pour afficher le graphique crée.

Exemple :

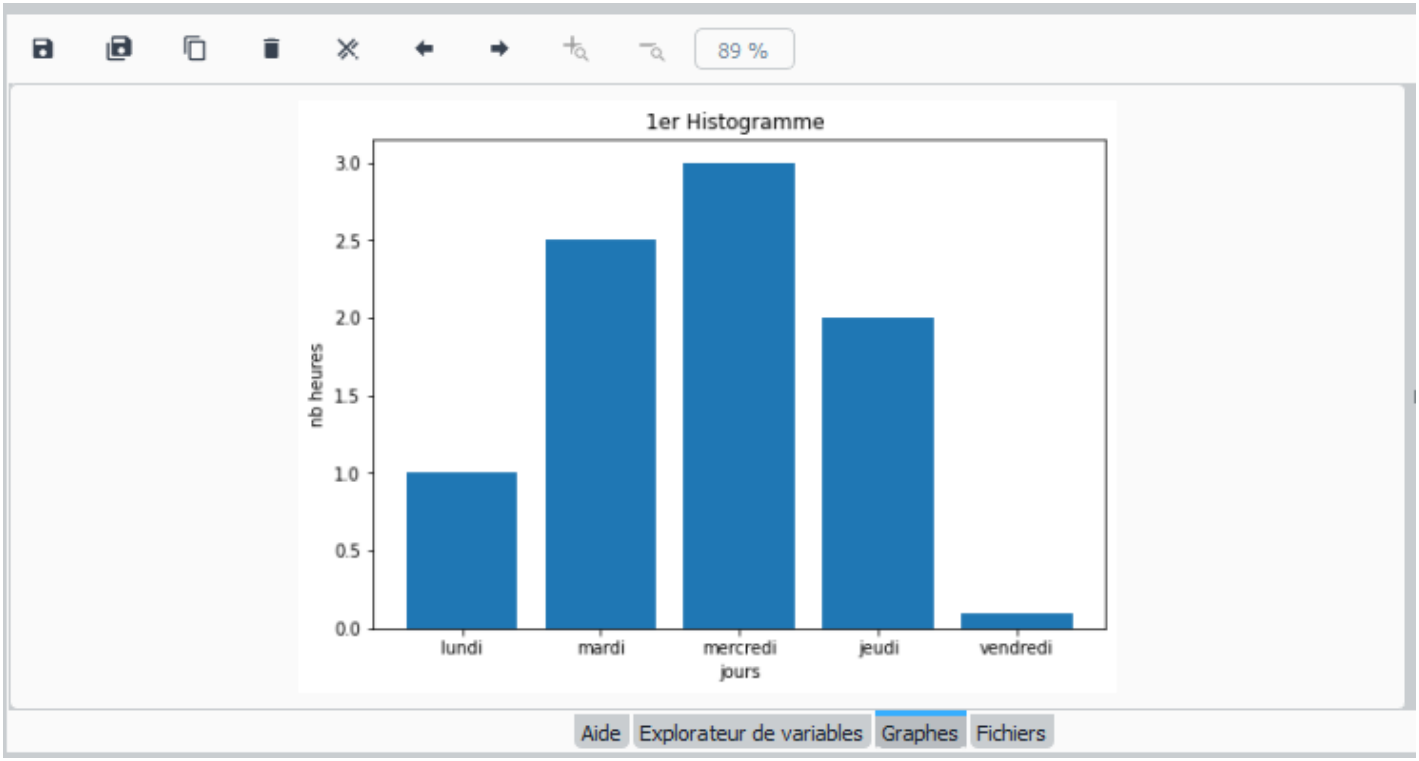
```
import matplotlib.pyplot as graphe1  # Module pour tracer les graphiques

# Préparation de la figure
fig = graphe1.figure()  #ouvre la fenetre graphique
ax = fig.add_axes([0, 0, 1, 1]) #ajoute des axes et règle la taille

L=['lundi', 'mardi', 'mercredi','jeudi','vendredi']
V=[1,2.5,3,2,0.1]

# Affichage des données
ax.bar(L,V) # instruction créant un histogramme
graphe1.title("1er Histogramme ")  # Titre du graphique
graphe1.ylabel('nb heures')  # Titre de l'axe des ordonnées
graphe1.xlabel('jours') # Titre de l'axe des abscisses
graphe1.show()  # Affichage du graphique
```

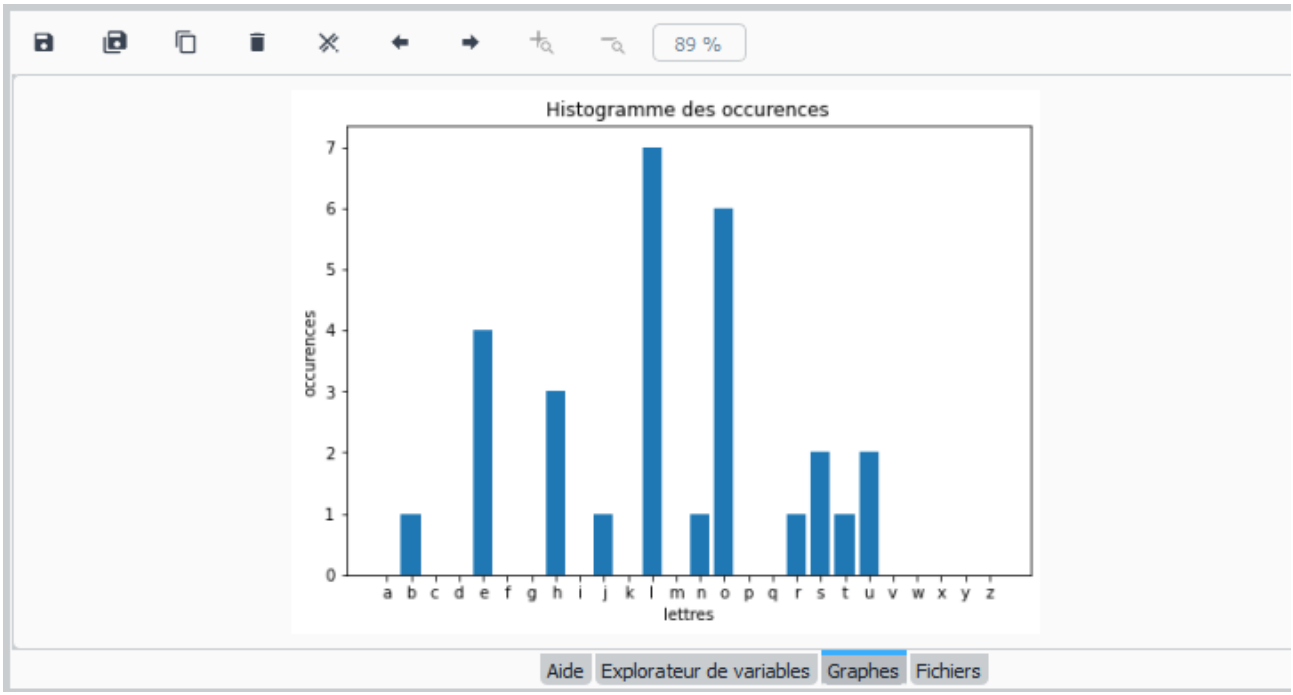
Donnera :



Exemple 2 : Si on ajoute l'instruction `return(s)` à la fin de la fonction « comptemot » vu précédemment, on peut par exemple établir un histogramme faisant apparaitre le nombre d'occurrences de chaque lettre dans le texte étudié.

```
24  import matplotlib.pyplot as graphe1  # Module pour tracer les graphiques
25
26  # Préparation de la figure
27  fig = graphe1.figure()  #ouvre la fenetre graphique
28  ax = fig.add_axes([0, 0, 1, 1]) #ajoute des axes et règle la taille
29
30  L=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s
31  valeurs = [comptemot("lecture dans un fichier.txt",k) for k in L]
32
33  # Affichage des données
34  ax.bar(L, valeurs) # instruction créant un histogramme
35  graphe1.title("Histogramme des occurences")  # Titre du graphique
36  graphe1.ylabel('occurences')  # Titre de l'axe des ordonnées
37  graphe1.xlabel('lettres') # Titre de l'axe des abscisses
38  graphe1.show()  # Affichage du graphique
```

Pour obtenir ceci :



Exercices :

1) Ecrire une fonction qui a pour entrée un mot (ou un texte) et qui renvoie le nombre de fois que la lettre « e » apparait dans le mot (ou le texte). On pourrait également afficher la fréquence d'apparition de la lettre « e » dans le mot (ou le texte)

2) Créer l'histogramme indiquant le nombre d'heures de cours que vous avez chaque jour.

3) Que permet de faire l'algorithme suivant ? Montrer qu'il a une complexité temporelle logarithmique par rapport à S.

```
1  n=0
2  exposant=5
3  S=10**exposant
4  a=3
5  while a**n<S:
6      n=n+1
7  print(n)
```