10. Lecture et écriture de fichiers

10.1. Pré-requis : répertoire de travail et chemins

On rappelle qu'un script Python est avant tout un *fichier* stocké quelque part sur votre ordinateur. Ce fichier se trouve dans un *répertoire* (plus couramment appelé *dossier*) que vous pouvez parcourir avec n'importe quel *explorateur de fichiers*, par exemple celui fourni par défaut par votre système d'exploitation ou encore celui intégré à Spyder.

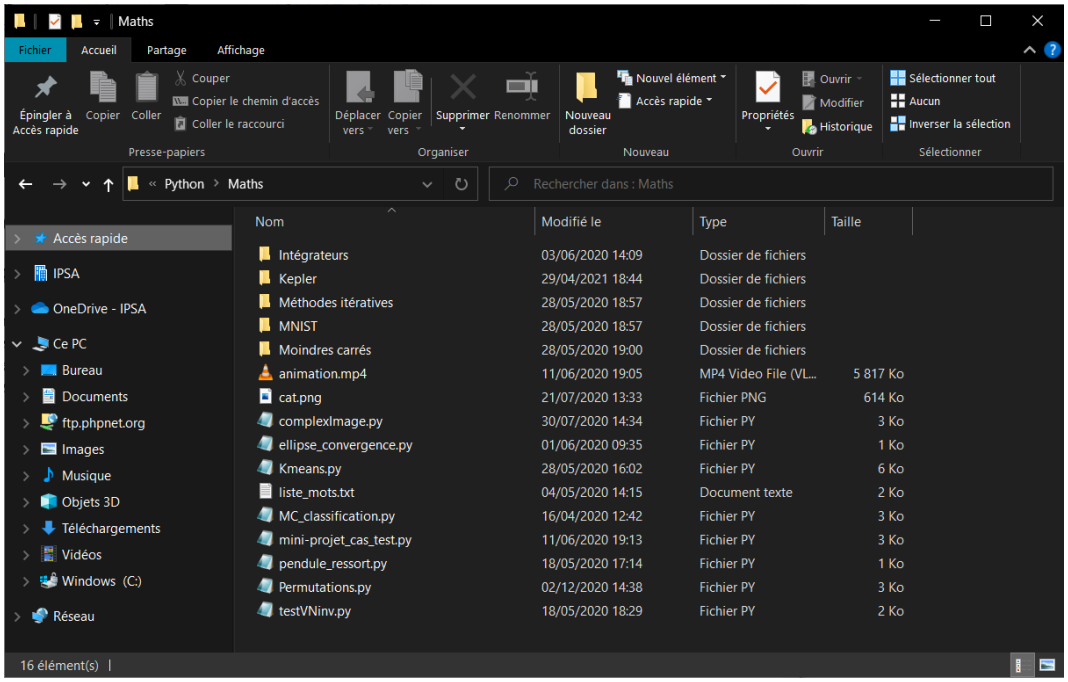


FIGURE 10.1. – Vue sur un dossier via l'explorateur de fichiers de Windows 10.

Remarque. La suite de cette section est rédigée pour un système Windows, mais la plupart des notions présentées se retrouvent de manière identique ou similaire sur d'autres systèmes.

Chaque répertoire ou fichier est localisé sur l'un des disques durs de votre ordinateur, et ces derniers sont en général désignés par les lettres `C:`, `D:`, `E:`, etc. On identifie alors un élément (fichier ou répertoire) de manière unique sur votre ordinateur en donnant son **chemin absolu** : il s'agit d'une chaîne de caractères commençant par la lettre du disque dur contenant cet élément, puis la liste des noms de tous les répertoires qu'il faut parcourir (dans l'ordre) pour arriver jusqu'à l'élément voulu et enfin le nom de l'élément, le tout séparé par des *antislashes* « \ ». Par exemple :

```
C:\Users\DamienBouloc\Python\Maths\cat.png
```

Pour se familiariser avec la notion de chemin, passons à un peu de pratique. Dans la barre de recherche Windows, recherchez le programme *Anaconda Powershell Prompt (Anaconda3)* et lancez-le (celui-ci a normalement été installé avec la suite Anaconda). [1] Une fenêtre semblable à celle de la figure 10.2 devrait s'ouvrir, avec un curseur clignotant vous indiquant que cette fenêtre est prête à recevoir des commandes écrites de votre part. On appelle d'ailleurs ce programme une *invite de commande* (ou *prompt* en anglais).

1. Si vous êtes sous Linux ou Mac, ouvrez un *terminal* et suivez les mêmes étapes.

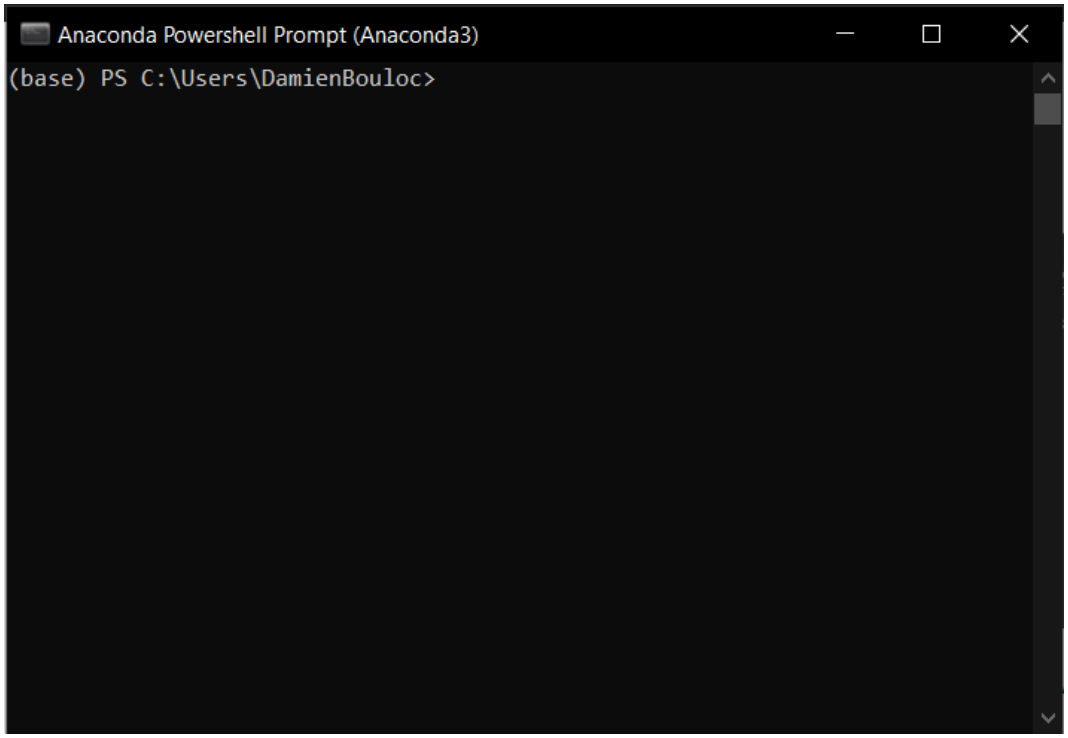


FIGURE 10.2. – Une capture de la fenêtre *Anaconda Powershell Prompt*.

Cette fenêtre fait apparaître l'indication `(base)`, la mention `PS` acronyme de *Powershell*, et enfin un chemin absolu. Il s'agit du chemin absolu du répertoire dans lequel l'invite de commande est en train de travailler actuellement, on l'appelle le *répertoire de travail courant*. Saisissez la commande `ls` (contraction de *list*) et validez avec la touche *Entrée*. Vous devriez voir apparaître de multiples lignes similaires à celles ci-dessous (mais pas exactement les mêmes) :

```
d-r---        02/03/2022     16:33                Desktop
d-r---        02/03/2022     11:44                Documents
d-r---        03/03/2022     15:57                Downloads
-a----        11/06/2020     17:05         127709 animation.mp4
-a----        26/04/2021     17:42          36808 Figure_1.png
-a----        26/04/2021     17:44          33954 Figure_2.png
```

Chacune de ces lignes correspond à un élément présent dans le répertoire de travail courant, avec sa date de dernière modification, sa taille et son nom. Chacun de ces éléments peut être soit un répertoire lui-aussi (dans ce cas la ligne correspondante commence par la lettre `d` pour *directory*), soit un fichier. Dans l'exemple ci-dessus, `Desktop`, `Documents` et `Downloads` sont des répertoires, tandis que `animation.mp4`, `Figure_1.png` et `Figure_2.png` sont des fichiers (respectivement une vidéo et deux images). Le contenu que vous voyez listé dans votre invite de commande est exactement le même que celui que vous verriez apparaître si vous parcouriez le répertoire de travail courant avec un explorateur de fichiers.

Vous pouvez également faire suivre la commande `ls` du nom d'un répertoire présent dans votre répertoire de travail courant, pour lister le contenu de ce nouveau répertoire. Par exemple en suivant toujours l'exemple ci-dessus (qui nous servira de fil rouge dans cette section), je peux saisir la commande

```
ls Desktop
```

et j'obtiens alors les lignes suivantes :

```
d-----        27/10/2021     13:36                kepler
d-----        28/02/2022     12:06                Nouveau dossier
-a----        23/01/2021     15:31         235900 Attestation.pdf
-a----        05/05/2021     17:26         185940 exos_moodle.pdf
```

Si je saisis plutôt la commande

```
ls Documents
```

alors j'obtiens sur mon ordinateur l'affichage suivant :

```
d-----        07/10/2021     15:40                Git
d-----        04/02/2021     11:01                MATLAB
-a----        07/10/2021     15:40         165926 coord3d.zip
```

Ces instructions vous permettent de deviner petit à petit que, sur mon ordinateur, mon disque dur `C:` est organisé suivant l'*arborescence* schématisée sur la figure 10.3.

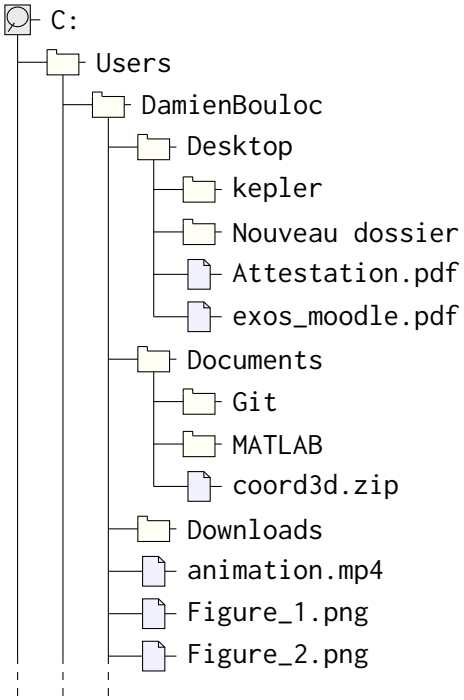


FIGURE 10.3. – Arborescence du disque dur utilisé dans les exemples précédents.

De là on en déduit que, par exemple, le dossier `kepler` a pour chemin absolu :

```
C:\Users\DamienBouloc\Desktop\kepler
```

Toutefois, plus l'arborescence de votre disque dur est complexe et profonde, plus les chemins absolus peuvent devenir longs et peu pratiques à utiliser. Il est alors fréquent d'utiliser des **chemins relatifs**. Leur fonctionnement est similaire à celui des chemins absolus, sauf qu'au lieu de partir d'une lettre désignant un disque dur, on part du caractère `.` qui désigne le répertoire de travail courant. Dans notre exemple, le répertoire de travail courant est le dossier `DamienBouloc`. Dans ce contexte le dossier `kepler` peut alors être désigné par le chemin relatif :

```
.\Desktop\kepler
```

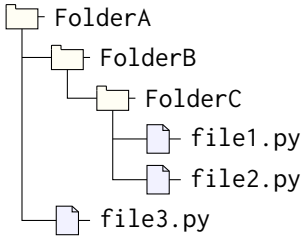
Dans un chemin relatif, on peut aussi utiliser l'identifiant `..` qui désigne le répertoire parent de l'élément qu'il suit. Toujours dans l'exemple où le répertoire de travail courant est `DamienBouloc`, le chemin relatif `..` désigne le dossier `Users`, tandis que le chemin relatif `..\..` désigne la racine `C:`.

Attention. Il faut bien comprendre que, contrairement au *chemin absolu*, le *chemin relatif* d'un élément dépend du répertoire de travail courant dans lequel on est en train d'opérer. Dans l'arborescence de la figure 10.3, on a par exemple la situation suivante :

Répertoire de travail courant	**Chemin relatif du fichier `Attestation.pdf`**
`C:\Users\DamienBouloc`	`.\Desktop\Attestation.pdf`
`C:\Users\DamienBouloc\Desktop`	`.\Attestation.pdf`
`C:\Users\DamienBouloc\Documents`	`..\Desktop\Attestation.pdf`
`C:\Users\DamienBouloc\Documents\Git`	`..\..\Desktop\Attestation.pdf`

Exercice 10.1. Entrainez-vous à naviguer au sein de l'arborescence de votre ordinateur uniquement via l'invite de commande *Anaconda Powershell Prompt* (ou équivalent). En plus de la commande `ls` qui liste le contenu d'un répertoire, vous pourrez utiliser la commande `cd` (pour *change directory*) qui permet de désigner un répertoire qui deviendra le nouveau répertoire de travail courant.

Exercice 10.2. On considère l'arborescence ci-dessous.



Donner les chemins relatifs des fichiers `file1.py`, `file2.py` et `file3.py` :

1. lorsque le répertoire de travail courant est `folderA`,
2. lorsque le répertoire de travail courant est `folderB`,
3. lorsque le répertoire de travail courant est `folderC`.

Lorsqu'un script Python est interprété, cela se fait aussi via une invite de commande qui possède alors son propre répertoire de travail courant. Comme l'IDE Spyder s'occupe tout seul de cette tâche à votre place, vous ne voyez pas l'invite de commande en question. Toutefois vous pouvez retrouver le répertoire de travail courant :

— soit en regardant le champ dédié situé en haut à droite de votre fenêtre Spyder,

— soit en appelant la fonction `getcwd()` du module `os` qui renvoie le répertoire de travail courant (*current working directory*) sous forme d'une chaîne de caractères.

Si vous n'avez pas changé les paramètres par défaut de Spyder, alors celui-ci devrait automatiquement changer de répertoire de travail courant lorsque vous exécutez un fichier Python, de sorte à se placer dans le répertoire qui contient votre script Python. Ainsi tant que vous passez par Spyder, vous pouvez toujours considérer que le répertoire de travail courant d'un fichier Python est le répertoire qui contient ce fichier.

Jusque là, vous n'aviez pas à vous soucier du répertoire de travail courant utilisé lorsque vous exécutiez vos scripts Python. Mais maintenant celui-ci va avoir une importance car nous allons apprendre à ouvrir des fichiers existants ou en créer des nouveau à l'aide de Python, et il est donc important de comprendre « où » ces fichiers seront recherchés ou créés lorsque vous exécuterez vos scripts.

10.2. Ouverture de fichiers avec Python

Pour ouvrir un fichier avec Python, on appelle la fonction open en lui donnant les deux arguments suivants :

— le chemin (absolu ou relatif) du fichier que l'on souhaite ouvrir,

— le *mode d'ouverture* de ce fichier.

Ce dernier paramètre est une chaîne de caractères à choisir parmi plusieurs valeurs prédéfinies. Dans ce cours, nous nous intéresserons uniquement aux trois valeur suivantes.

Option	Signification	Description
`"r"`	*read*	Ouvre le fichier en mode **lecture** seulement. Il sera possible de récupérer le contenu du fichier pour le traiter, mais le fichier ne sera pas modifiable. Si le fichier n'existe pas, une erreur est déclenchée.
`"w"`	*write*	Ouvre le fichier en mode **écriture** (avec remplacement). On pourra alors écrire dans le fichier, mais pas lire son contenu. Si le fichier n'existe pas, il est créé. S'il existe déjà, son contenu est complètement effacé.
`"a"`	*append*	Ouvre le fichier en mode **écriture** (avec ajout à la fin). On pourra alors écrire dans le fichier, mais pas lire son contenu. Si le fichier n'existe pas, il est créé. S'il existe déjà, son contenu n'est pas effacé, les nouvelles données seront ajoutées à la suite du contenu déjà présent.

Cette fonction renvoie un *objet-fichier* : c'est un objet qui garde temporairement la main sur votre fichier pour le lire ou y écrire dedans (selon le mode choisi), comme le ferait un éditeur de texte classique. Une fois que vous avez terminé de travailler avec le fichier, il vous faut penser à le libérer en appelant la méthode `close` depuis votre objet fichier. Autrement, vous risquez d'obtenir des messages d'erreur si vous essayez de déplacer ou modifier votre fichier alors que celui-ci est encore ouvert par l'objet-fichier dans votre programme Python.

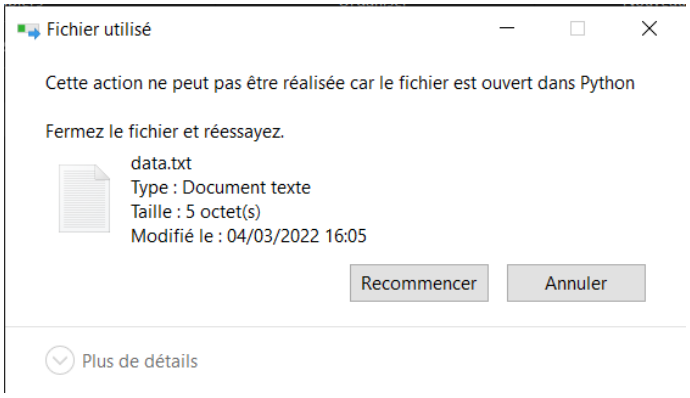


FIGURE 10.4. – Exemple de message d'erreur apparaissant lorsqu'on essaie de déplacer un fichier encore utilisé par un programme Python.

Remarque. S'il peut être tentant d'utiliser des chemins absolus dans vos programmes Python, nous vous conseillons de plutôt toujours placer les fichiers que vous voulez traiter dans le même répertoire que votre script (ou dans un de ses sous-répertoires) et d'utiliser des chemins **relatifs**. De cette manière, votre code sera

portable (c'est-à-dire que si vous déplacez le dossier contenant votre script et les fichiers utilisés sur un autre ordinateur, tout devrait continuer à fonctionner), mais aussi plus sûr : en ne manipulant que des fichiers dans le répertoire de travail local de votre script, vous ne prenez pas le risque d'effacer ou modifier par mégarde des fichiers importants de votre disque dur.

10.2.1. Écriture de fichier

L'écriture de fichier est l'opération la plus facile à effectuer. Une fois un fichier ouvert en mode écriture (`w` ou `a`), il suffit d'appeler la méthode `write` à partir de l'objet-fichier, en lui passant en argument la chaîne de caractères que l'on veut rajouter à la suite du fichier. On peut appeler la méthode `write` autant de fois que l'on veut tant que le fichier est toujours ouvert.

À tester. Vérifier que le programme suivant crée un fichier `output.txt` dans le répertoire de travail courant, et analyser son contenu.

```
f = open("output.txt", "w")

f.write("Un")
f.write("Deux")
f.write("3")

f.close()
```

Que se passe-t-il si vous essayez d'appeler la méthode `write` *après* avoir fermé le fichier ?

Attention. Il faut retenir que la méthode `write` n'ajoute pas de saut de ligne automatiquement. Si vous voulez revenir à la ligne, c'est à vous d'insérer explicitement le caractère spécial `"\n"` aux endroit opportuns dans votre fichier.

Exercice 10.3. Des fichiers très pertinents.

1. Créez un fichier `numbers.txt` qui contiendra les nombres de 1 à 100 écrits en chiffres, un par ligne.
2. Dans un dossier nommé `inutile`, créez 30 fichiers `file01.txt`, `file02.txt`, ..., `file30.txt` contenant tous uniquement la chaîne de caractères « RIEN ».

10.2.2. Lecture de fichiers

Une fois un fichier ouvert en mode lecture, on peut appeler à partir de l'objet-fichier les méthodes suivantes.

— `read(n)` : renvoie les n caractères suivants du fichier (ou moins s'il reste moins de n caractères à lire),

— `readline()` : renvoie la ligne suivante du fichier, c'est-à-dire que les caractères sont lus jusqu'au prochain retour à la ligne (inclus) ou jusqu'à la fin du fichier le cas échéant.

Si l'on précise que ces méthodes renvoient les caractères ou lignes *suivantes* du fichier, c'est parce qu'il faut imaginer que l'objet-fichier possède un *curseur* qui se déplace dans le fichier. Lorsqu'on ouvre un fichier en mode lecture, ce curseur se trouve initialement au tout début du fichier. Ensuite, ce curseur avance au fur et à mesure qu'on lit le texte (par exemple avec `read` ou `readline`). Cela signifie notamment qu'en général,

deux appels successifs à `read` (ou à `readline`) ne renvoient **pas** le même résultat. Une fois que le curseur a atteint la fin du fichier, tout appel à `read` ou `readline` renvoie la chaîne de caractères vide.

À tester. Téléchargez le fichier `exemple.txt` fourni avec ce support, contenant uniquement les trois lignes suivantes :

```
ABCD
EFG
HIJKL
```

Dans le même répertoire de travail, testez ensuite les deux codes ci-dessous (séparément) et observez le contenu des variables qu'ils créent.

```
f = open("exemple.txt", "r")

a = f.read(6)
b = f.read(6)
c = f.read(6)
d = f.read(6)

f.close()
```

```
f = open("exemple.txt", "r")

a = f.readline()
b = f.readline()
c = f.readline()
d = f.readline()

f.close()
```

Attention. Notez bien que lorsqu'on lit une ligne complète d'un fichier, alors celle-ci se termine en général par le caractère de fin de ligne `"\n"` (sauf s'il s'agit de la dernière ligne du fichier). Lorsqu'on traite le contenu d'un fichier, il est fréquent de commencer par retirer ce caractère indésirable en fin de chaîne. On revoie pour cela le lecteur à la méthode `rstrip` du chapitre sur les chaînes de caractères.

Exercice 10.4. Le fichier `alice_chapter1.txt` contient l'intégralité du premier chapitre de *Alice's Adventures in Wonderland* de Lewis Carroll. Parcourir ce fichier caractère par caractère de sorte à compter :

- le nombre de caractères qu'il contient (sans distinction : lettres, espaces, retours à la lignes, etc.),
- le nombre d'occurrences de la lettre « e » (majuscule ou minuscule),
- le nombre d'occurrences de la lettre « a » (majuscule ou minuscule).

Toutefois, quand on souhaite lire un fichier ligne par ligne dans sa totalité, le plus simple est encore d'itérer directement sur l'objet-fichier à l'aide de l'opérateur `for`.

À tester. Dans le même répertoire de travail que précédemment, observez l'affichage produit par le programme suivant.

```
f = open("exemple.txt", "r")

n = 0
for line in f:
    n = n + 1
    print(f"Ligne {n}: '{line}'")

f.close()
```

Remarque. Il existe aussi la méthode `readlines` qui renvoie directement une liste contenant toutes les lignes d'un fichier, mais nous **déconseillons** son usage. En effet, vous pourriez rencontrer des fichiers suffisamment volumineux pour ne pas pouvoir être chargés entièrement dans la mémoire vive. La bonne pratique lorsqu'on analyse ou crée des fichiers est de procéder caractère par caractère, ou éventuellement ligne par ligne, afin de réduire les besoins en mémoire de votre programme. Dans les exercices de ce chapitre, contentez-vous donc d'utiliser les méthode `read` et `readline` (sans le `s` final) ou l'opérateur `for`.

Exercice 10.5. Le fichier `impayes.csv` contient une liste d'informations sur des clients devant encore de l'argent aux entreprises *Factix*. Chaque ligne donne, séparés par des virgules et dans cet ordre : la civilité du client (M. ou Mme.), son prénom, son nom, le montant dû, et la date depuis laquelle le montant est dû.

Écrire un programme qui lit ce fichier ligne par ligne, de sorte à afficher dans la console des phrases de la forme : « *M. Franck Ofone doit 281.63 euros depuis le 23/03/2021* ».

Remarque. Le fichier utilisé dans l'exercice précédent est un fichier au format CSV (*Comma-separated values* : valeurs séparées par des virgules). C'est un format simple à générer ou à traiter qui s'avère pratique lorsqu'on veut stocker des données sous forme de « tableau ». Il est donc assez répandu, bien que défini de manière imprécise (certains logiciels séparent les données par des virgules et d'autres par des points-virgules, le séparateur décimal peut varier d'une langue à l'autre, …). Vous aurez d'ailleurs peut-être déjà remarqué qu'un fichier CSV peut être ouvert avec un logiciel tableur comme Microsoft Excel. Il existe aussi des modules Python (typiquement le module `csv`) pour manipuler plus aisément ces fichiers. Toutefois c'est un bon exercice en première année que de s'entraîner à manipuler ce format sans ces modules, en utilisant uniquement les fonctionnalités élémentaires de lecture/écriture de fichiers fournies par Python.

Exercices supplémentaires

Exercice 10.6. Copie de fichiers. Écrire une fonction `copy_file(path_in, path_out)` qui ouvre le fichier de chemin `path_in` et le recopie caractère par caractère dans le fichier de chemin `path_out`. Vous penserez à fermer les deux fichiers une fois la copie terminée.

L'astuce ici est que rien ne vous interdit d'avoir deux fichiers ouverts en même temps, l'un en lecture et l'autre en écriture.

Exercice 10.7. Analyse de fréquences. Reprendre le fichier `alice_chapter1.txt`, et créer cette fois un dictionnaire dont les clés sont les lettres minuscules de l'alphabet, et dont les valeurs associées sont le nombre d'occurrence de chacune de ces lettres dans le fichier.

Bonus : représenter ensuite ces résultats sous forme d'histogramme à l'aide de la fonction `bar` du module `matplotlib.pyplot`.

Exercice 10.8. Lettres de relance. On se base à nouveau sur la liste de clients du fichier `impayes.csv`. Le fichier `modele_relance.txt` contient un modèle de lettre de relance que l'on veut envoyer à chacun de ces clients. Écrire un programme qui traite le fichier `impayes.csv` ligne par ligne, et qui pour chacun des clients de ce fichier, crée dans le dossier `mails` un nouveau fichier `relance_nom_prenom.txt` contenant la lettre de relance qui sera envoyée au client. Cette lettre de relance sera créée à partir du modèle évoqué précédemment, en injectant les bonnes données dans les champs suivants :

0. la civilité du client,

1. le prénom du client,
2. le nom du client, transformé en lettres majuscules,
3. le montant dû (attention : il faut fournir un flottant),
4. la date depuis laquelle le montant est dû,
5. le montant de l'éventuelle pénalité, égal à 10% du montant dû (attention : il faut fournir un flottant).

À retenir

— Comprendre les notions de chemins absolus et relatifs.

— Savoir identifier le chemin relatif d'un fichier par rapport à un répertoire de travail fixé dans une arborescence donnée.

— Savoir ouvrir et fermer un fichier à l'aide de Python en choisissant le mode adapté.

— Savoir écrire des données dans un fichier.

— Savoir lire un fichier à l'aide des méthodes `read` et `readlines` et d'une boucle `for`.

— Savoir analyser et/ou créer des fichiers à l'aide d'algorithmes simples.