TP 25
Fichiers texte

I Quelques notions sur le système de fichiers

I.1 L'arborescence des fichiers

Les fichiers sur l'ordinateur permettent de stocker des informations et de les récupérer quand on en a besoin. Dans un programme Python, il est courant de vouloir accéder à un fichier pour en lire le contenu, ou bien pour écrire dedans et sauvegarder des données.

Ils ont tous un nom et sont rangés dans des dossiers. Chaque dossier peut donc contenir plusieurs fichiers mais aussi d'autres dossiers. Cela se représente naturellement par un arbre. Chaque fichier possède un **chemin d'accès** dans lequel le caractère `/` indique que ce qui suit se trouve dans le dossier de ce qui précède. Le chemin d'accès est en général donné relativement à un dossier de base dans lequel on travaille.

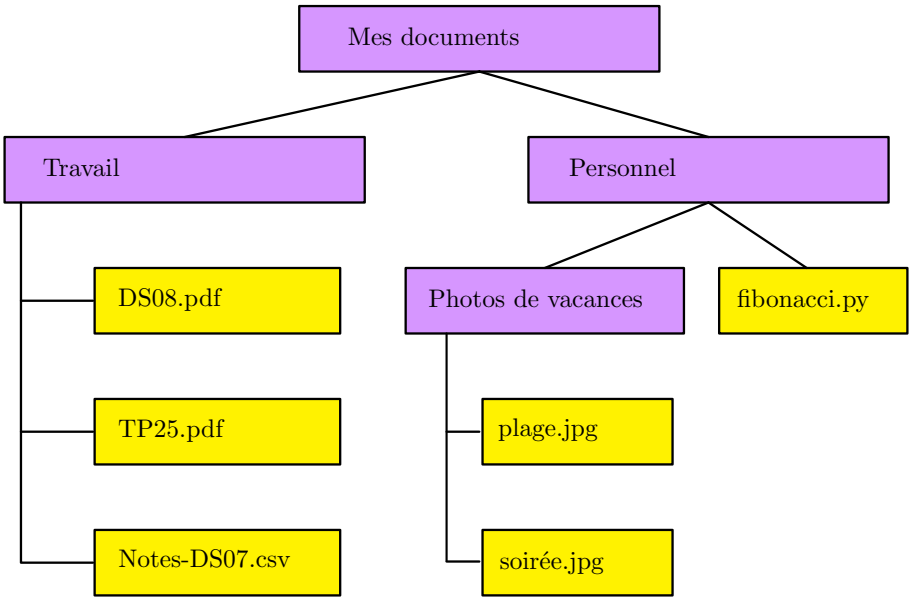

Figure 1. – Ici, travaillant à partir du dossier `Mes documents`, le fichier `plage.jpg` est à l'adresse relative `Personnel/Photos de vacances/plage.jpg`. Le dossier `Travail` contient trois fichiers ; le dossier `Personnel` contient un dossier (contenant lui-même deux fichiers) et un fichier.

Avertissement

Pour bien démarrer le TP, il est impératif de bien vérifier que le programme Python comprenne, que les chemins d'accès au fichier sont donnés relativement au dossier dans lequel se trouve le TP. C'est la raison pour laquelle il est important de décompresser en entier le fichier `materiel.zip`, avec le fichier `.py` et ses données jointes dans un même dossier.

Dans le logiciel Spyder, le nom du « répertoire de travail » (le dossier dans lequel le programme Python pense être, et relativement auquel les fichiers seront cherchés) est affiché en haut à droite. Un clic droit dans l'onglet du fichier puis l'option « Définir le répertoire de travail de la console » permet de configurer le répertoire de travail comme étant bien celui contenant le fichier `.py` sur lequel on est en train de travailler. Avec le logiciel Pyzo, c'est un clic droit dans la console interactive puis « Définir le répertoire courant en accord avec le fichier ouvert dans l'éditeur ». Éventuellement, cette configuration est automatique si on exécute tout le fichier d'un seul coup.

Si les petits tests dans le programme s'exécutent correctement, le TP peut démarrer.

I.2 Le processus de lecture et écriture de fichier

Bien qu'un fichier texte soit constitué, comme son nom l'indique, de texte, on ne peut pas le lire exactement comme les chaines de caractères en accédant à tout moment au i-ème caractère. Les problèmes suivants se posent :

- Le fichier peut être stocké sur le disque dur, ou sur une clé USB, ou sur un ordinateur auquel on accède à travers le réseau ; dans tous ces cas chaque accès au fichier demande un certain effort au matériel, et il n'est pas raisonnable d'écrire une boucle qui demande à lire chaque caractère un par un. Cela serait beaucoup trop lent et userait le matériel.
- On ne peut pas non plus toujours le stocker en mémoire d'un seul coup avant de le lire, car il peut être trop gros.

En fait, lors de la lecture d'un fichier, un mécanisme automatique et largement caché à l'utilisateur se met en place et le fichier est chargé en mémoire par blocs. Penser par exemple à une vidéo YouTube de plusieurs heures, dont le fichier original occupe plusieurs giga-octets : on n'a pas besoin de la télécharger en entier avant de la regarder, mais pendant qu'on lit un morceau alors la suite se télécharge en arrière-plan. Le terme technique est *mise en mémoire tampon* (en anglais *buffer*), de la mémoire temporaire pour stocker un bout de fichier.

De même en écriture, il serait très inefficace de demander au disquer dur d'écrire uns par uns les caractères, et on ne veut pas non plus stocker le fichier en mémoire puis l'écrire d'un seul coup à la fin. Les données à écrire sont donc chargées dans une « file d'attente » et celle-ci est automatiquement vidée régulièrement par blocs. Le terme technique est *flush* comme dans… tirer la chasse d'eau.

En résumé on peut retenir que l'utilisateur n'a pas toujours la garantie que sa commande d'écriture a effectivement bien écrit les données sur le disque dur, même si elle s'est correctement terminée. Pour que tout se passe bien il y a un procesuss d'**ouverture** et de **fermeture** de fichier.

À n'importe quel moment le système d'exploitation sait si un fichier est ouvert quelque part ou non. C'est par exemple cela qui va provoquer des avertissements ou des erreurs si on débranche une clé USB alors qu'un fichier dessus est encore considéré comme ouvert… Cela sert aussi à interdire à un autre programme de modifier le fichier pendant qu'on est en train de le lire (ce serait embêtant !).

II Les fichiers en Python

II.1 Ouverture

La fonction `open(nom, mode)` sert à ouvrir un fichier en donnant son nom ou son chemin d'accès. Elle renvoie une variable de type `file` qui représente un fichier ouvert. La fonction prend deux arguments : le nom du fichier (une chaine de caractères), et le mode d'ouverture. Les modes possibles sont listés ci-dessous :

- `"r"` (*read*) : Lecture seule. Permet de lire du contenu dans le fichier.
- `"w"` (*write*) : Écriture dans le fichier. Si le fichier n'existe pas, il est créé comme un fichier vide. Sinon, il est d'abord effacé, pour être remplacé par ce qu'on va écrire. **Ce mode est dangereux** : avant de l'utiliser, vérifiez bien que le nom de fichier est correct, et correspond bien à un fichier qu'on peut se permettre d'effacer !
- `"a"` (*append*) : Écriture dans le fichier en mode ajout. Si le fichier n'existe pas, il est créé comme un fichier vide ; mais sinon, tout ce qui va être écrit va s'ajouter à la fin du fichier déjà existant, sans rien effacer des données précédentes.

Une fois que le fichier a été ouvert avec `f = open(nom, mode)`, la méthode `f.close()` ferme le fichier. Il n'est alors plus possible de lire ou d'écrire dedans. Comme expliqué dans la partie I.2, en écriture, cette commande vide éventuellement la « file d'attente » et termine d'écrire tout ce qui devait l'être.

> *Remarque.* Les fichiers ont souvent des **droits d'utilisation**, typiquement des fichiers de configuration du système peuvent être lus mais ne peuvent pas être modifiés par un utilisateur quelconque. Les modes d'ouverture doivent donc être en accord avec les droits du fichier. Par exemple l'ouverture d'un fichier en mode écriture échouera si le fichier a seulement les droits de lecture, et même si on n'écrit finalement rien dedans ! Bien qu'il existe aussi un mode de lecture et écriture simultané, on ouvre le fichier dans le mode qui donne *le moins* de droits possible par rapport à ses besoins.

Exercice 1
Tester et observer le fichier de démarrage. Exécutez-le plusieurs fois et observez les fichiers produits.

II.2 Lecture

Une fois qu'un fichier a été ouvert en mode lecture `"r"` et représenté par une variable `f`, on peut lire les lignes unes par unes simplement avec une boucle `for s in f`, où `s` sera une chaine de caractères :

```
>>> for s in f: s
'Première ligne du fichier de test.\n'
'Deuxième ligne.\n'
'Troisième et dernière ligne.\n'
```

Chaque ligne se termine par un caractère spécial noté `"\n"` qui sert à marquer la fin de la ligne (*newline*). C'est un caractère à part entière, bien qu'il ne corresponde pas directement à une lettre, tout comme les espaces : les fichiers textes sont enregistrés comme une suite de caractères consécutifs et c'est le `"\n"` qui indique au logiciel de passer à la ligne !

Un petit problème ennuyeux se pose si on veut comparer la ligne avec du texte car elle se termine par ce caractère. La méthode `s.strip()` permet de retirer le caractère `"\n"` final, ainsi qu'éventuellement des espaces en trop au début ou à la fin de ligne :

```
>>> s = "      un mot   \n"
>>> s
'      un mot   \n'
>>> s.strip()
'un mot'
```

Dans cet exemple on pourra donc effectuer des comparaisons telles que `s.strip() == "un mot"`, sans se soucier du caractère de fin de ligne.

Exercice 2
Le fichier joint `noms.txt` contient la liste des prénoms des élèves de la classe, chacun sur une seule ligne.

1. D'abord, écrire un programme qui ouvre et ferme le fichier et contient une boucle qui affiche uns par uns les noms.
2. Améliorer le programme pour qu'il affiche `Bonjour [nom] et bienvenue en BCPST1B !` (où `[nom]` sera remplacé par le prénom de l'élève).

Exercice 3
Le fichier joint `dictionnaire.txt` contient un dictionnaire de plus de 600 000 mots pour la correction orthographique.

Écrire une fonction `cherche(m)` qui prend en argument un mot `m` et qui renvoie `True` si le mot `m` apparait dans le fichier, `False` sinon.

Exercice 4
Le fichier joint `pi.txt` contient un million de décimales du nombre π, formatées en lignes de 100 décimales.

Écrire un programme qui compte combien de fois chaque chiffre apparait, renvoyant une liste `C` de taille 10 initalement remplie de zéros, où `C[i]` contiendra le nombre de fois que le chiffre `i` apparait dans les décimales.

Attention, étant donné un caractère `x` de la chaine (qui représente bien un nombre, mais est de type `str`), on a besoin de la fonction `int(x)` pour le convertir en un nombre.

Remarque. Il existe d'autres fonctions pour lire un fichier texte : `f.read()` (lit tout le fichier et renvoie une seule chaine de caractères), `f.readline()` (lit une seule ligne, renvoie une chaine de caractères), `f.readlines()` (lit toutes les lignes et renvoie son résultat sous forme d'une liste de chaines de caractères).

Chacune de ces fonctions déplace un curseur dans le fichier, de sorte que des appels successifs (à `f.readline()` par exemple) vont avancer dans le fichiers. À la fin, il n'y a plus rien à lire, et même avec la syntaxe précédente, une deuxième boucle successive `for s in f` ne produira aucun effet car le curseur est à la fin ! La commande `f.seek(0)` permet de remettre le curseur au début du fichier, et alors on peut le lire à nouveau.

II.3 Écriture

Une fois qu'un fichier a été ouvert dans un mode `"w"` ou `"a"` permettant l'écriture, et représenté par une variable `f`, il y a deux façons principales d'écrire dedans :

- `f.write(s)` : écrit la chaine de caractères `s` dans le fichier. Pour sauter des lignes, il est nécessaire d'écrire manuellement des caractères `"\n"` dans `s` : des appels successifs à `f.write()` ne passent pas automatiquement à la ligne. Comme la fonction prend uniquement en argument une chaine de caractère, si on veut écrire des nombres il faut d'abord formater les chaines comme dans l'annexe III.1.
- `print(x, y, ..., file=f)` : la bonne vieille fonction `print` à laquelle on dit d'écrire dans le fichier `f` plutôt que sur l'écran. On peut alors écrire tout ce qu'on écrit d'habitude avec `print` (des chaines de caractères, des nombres, des listes, etc). Comme d'habitude, les appels successifs passent à la ligne, sauf si on rajoute l'argument `end=""` (« remplacer le caractère `"\n"` automatiquement aujouté à la fin par la chaine vide `""` »).

Exercice 5
Il est courant d'avoir un programme qui fait un calcul et de vouloir enregistrer le résultat pour ne pas le perdre. Choisissons par exemple, au hasard, un programme qui calcule les termes successifs d'une suite de récurrence d'ordre 2… Écrire une fonction `fibonacci(n)` qui ouvre un fichier nommé `fibonacci.txt` pour écriture, et écrit dedans uns par uns les termes de la suite de Fibonacci.

À la fin on doit obtenir un fichier qui ressemble à ceci :

```
0
1
1
2
3
5
8
13
...
```

N'hésitez pas à tester avec $n = 100$ et vous pouvez enregistrer le fichier et le garder en souvenir !

Exercice 6
On veut étudier des séries de 10 lancers de dés, et conserver les résultats obtenus. Écrire un programme qui ouvre un fichier nommé `dés.txt` en mode `"a"`, simule 10 fois de suite un lancer de dé, et écrit le résultat dans le fichier. Idéalement, les résultats sont séparés par un espace, et il faut ajouter un nouvelle ligne une seule fois à la fin.

Si on lance plusieurs fois le programme et que tout se passe bien, les séries de lancers s'accumulent. Le fichier devra ressembler à :

```
2 6 5 3 6 6 1 1 3 5
6 3 2 3 2 4 6 1 3 3
1 3 1 4 6 5 6 6 5 1
...
```

II.4 Lire d'un côté et écrire de l'autre

Texte simple On propose la situation suivante. La procédure Parcoursup se termine et la direction du lycée Hoche reçoit la liste des élèves qui ont été admis et ont accepté de venir. Elle souhaite donc envoyer un message de bienvenue personnalisé à chacun. Mais il n'est pas question de copier-coller des messages, ni de tout ré-écrire à la main. À la place, on veut écrire un programme Python qui, d'un côté lit la liste des élèves, de l'autre produit des fichiers prêts à envoyer…

Exercice 7
Écrire une fonction `bienvenue(n, nom)` qui prend en argument un nombre et un prénom, qui crée une variable contenant le nom du fichier sous la forme `bienvenue-[n].txt` (en remplaçant `[n]` par le nombre) puis ouvre ce fichier en mode `"w"` et écrit dedans un texte de bienvenue personnalisé. On pourra choisir librement le texte, par exemple

```
Bonjour [nom],
Félicitations pour votre admission au lycée Hoche. Nous vous souhaitons la bienvenue dans
la classe de BCPST1B et espérons que vous vous y plairez.
Le Lycée Hoche
```

Pour tester la fonction il faut par exemple essayer `bienvenue(1, "L.-C. Lefèvre")` et observer si un fichier a bien été créé. Si c'est le cas on peut passer à la deuxième partie.

Comme plusieurs élèves de la classe peuvent porter le même prénom, il est délicat de vouloir utiliser le prénom pour former le nom de fichier. D'où le besoin de numéroter les élèves. Idéalement, les fichiers produits devraient tous se trouver dans un même dossier `lettres` et les numéros devraient tous être alignés sur deux caractères (en complétant avec un zéro), voir l'annexe III.1.

Exercice 8
Écrire une programme qui ouvre en lecture le fichier `noms.txt` contenant la liste de tous les élèves de la classe, qui itère sur les lignes pour récupérer les prénoms uns par uns (en comptant au passage les prénoms dans une variable `n`) et appelle la fonction précédente pour générer le fichier de bienvenue.

Puis lancer le programme et observer. Si tout se passe bien, de nombreux fichiers doivent être créés dans le dossier `lettres`.

Avec un fichier CSV Une autre grosse source d'amélioration consiste à charger non pas une simple liste des élèves mais tout un fichier CSV contenant plus d'informations sur chaque élève (nom, prénom, date de naissance, lycée d'origine, …). Un fichier CSV s'ouvre au départ comme un fichier texte `f = open(nom, "r")` mais ensuite il faut créer un objet `g` « lecteur » pour itérer dessus en fournissant unes à unes les lignes avec la syntaxe `for x in g` ; plus précisément la commande `g = csv.DictReader(f)` utilise l'en-tête du fichier CSV pour nommer les colonnes et l'itération fournit les lignes unes par unes en tant que dictionnaires. Pour la liste des élèves cela donne le schéma de programme suivant :

```
import csv
f = open("liste.csv", "r")
g = csv.DictReader(f)
for x in g:
    ...
    x["nom"], x["prenom"], x["etablissement_origine"], ...
    ...
f.close()
```

Exercice 9
Reprendre le programme de l'exercice précédent mais avec un fichier CSV :

- La fonction `bienvenue(n, ...)` prend par exemple en argument des noms, prénoms, établissements d'origine, et crée une lettre personnalisée à partir de toutes ces informations. Les fichiers seront écrits dans le dossier `lettres-CSV`.
- Le programme principal utilise le fichier CSV de liste des élèves `liste.csv` et appelle la fonction `bienvenue(n, ...)` sur chaque entrée.

Générer du PDF Pour produire un beau fichier PDF, il faudrait passer par un langage de programmation permettant de mettre en forme un document tout en donnant du contenu. C'est bien trop compliqué pour ce TP…

Un programme `lettres.py` est fourni dans le matériel du TP. Le programme Python écrit un fichier dans un autre langage appelé LaTeX, qui est très utilisé pour produire des documents scientifiques de haute qualité (les

feuilles d'exercices, sujets de DS et DM, mais pas ce TP). Tester le programme en exécutant tout le fichier d'un coup, attendre, et admirer le résultat.

III Annexes

III.1 Le formatage des chaines de caractère

Le problème du formatage consiste à insérer dans une chaine de caractère des mots, des nombres, et parfois de régler certains paramètres comme le nombre de caractères affichés ou le nombre de chiffres significatifs.

Une syntaxe moderne en Python consiste à précéder la chaine de caractère directement par la lettre `f` (on parle de *f-string* : chaine formatée). Ensuite, lorsque la chaine va contenir la syntaxe `{nom:code}` la variable nommée `nom` va être remplacée par sa valeur, éventuellement formatées avec le code de formatage `code`. Quelques exemples :

```
>>> nom = "L.-C. Lefèvre"
>>> print(f"Bonjour {nom}")
Bonjour L.-C. Lefèvre
>>> x = 15
>>> print(f"x = {x:4d}")
x =   15
>>> print(f"x = {x:04d}")
x = 0015
```

Pour les nombres entiers, le code `d` permet de l'afficher comme un nombre décimal ; précédé d'un nombre, il indique la place en nombre de caractères que va occuper l'écriture du nombre. Précédé aussi d'un zéro, il force l'affichage d'un zéro pour remplir l'espace. On peut aussi précéder encore d'un `+` pour forcer l'affichage du symbole `+` pour les nombres positifs ; cela est pratique pour bien aligner en colonne des nombres positifs et négatifs, observer l'alignement :

```
>>> for x in (-123, -8, 13, 256): print(f"{x:+04d}")
-123
-008
+013
+256
```

Pour les nombres à virgule flottante, il existe diverses options pour régler le nombre de chiffres significatifs, l'affichage en décimal ou bien en notation exponentielle, etc. C'est un mini-langage en soi. Se référer à une documentation.

III.2 Le mot-clé `with`

Lorsqu'on manipule des fichiers on trouve souvent le code suivant :

```
with open(nom, mode) as f:
    ...
    # bloc d'instructions
    # opérations sur le fichiers
    ...
...
```

La première ligne est à peu près équivalente à notre `f = open(nom, mode)`. Mais les différences avec la syntaxe précédente sont :

- Ce qui suit est un *bloc d'instructions*, exactement comme avec les `if`, les boucles, etc.
- Le fichier est automatiquement fermé à la fin du bloc d'instructions…
- … même si une erreur pendant l'exécution se produit.

Au contraire sans le mot-clé `with`, si une erreur se produit après avoir ouvert le fichier alors le programme va s'arrêter et donc risque de ne pas aller jusqu'à l'instruction de fermeture du fichier !

On parle de **gestionnaire de contexte** — le mot-clé déclenche une sorte de veille sur le bon déroulement des opérations qui sont bien délimitées dans le bloc d'instructions. Il semble que le mot-clé `with` soit une bonne pratique recommandée en Python, mais pas adoptée en BCPST…

III.3 Le module `os`

Le module `os` (*Operating System*, système d'exploitation) contient de nombreuses fonctions pour gérer tout ce que l'on pourrait faire manuellement en se baladant dans l'explorateur de fichiers, et d'obtenir des informations sur le système. Quelques exemples :

- `os.getcwd()` : donne le nom du dossier dans lequel on travaille.
- `os.listdir()` : renvoie la *liste* de tous les dossiers et fichiers situés dans le dossier dans lequel on travaille.
- `os.mkdir(nom)` : crée un dossier avec le nom donné.
- `os.remove(nom)` : supprime le fichier de nom donné.
- `os.rename(nom_avant, nom_après)` : renomme un fichier.

Pour faire des tests dans ce TP, cela peut être intéressant d'écrire dans le programme le code qui efface les fichiers créés, pour pouvoir facilement recommencer. On peut aussi vouloir écrire un programme qui traite automatiquement tous les fichiers dans un dossier donné.