On découvre

- les objets fichiers
- les différents modes d'ouverture d'un fichier
- la fermeture d'un fichier

Vous utilisez des fichiers tous les jours sans y penser : les fichiers `.py` qui contiennent vos programmes, les fichiers `.pdf` qui contiennent les consignes, les fichiers `.mp4` qui contiennent les vidéos en ligne, les fichiers `.md` des git, les fichiers `.png` des copies d'écran…

Dans ce cours nous nous intéresserons uniquement aux fichiers texte, qui contiennent… du texte, et qui sont structurés par ligne.

1 Syntaxe générale

Nous allons voir comment manipuler le contenu d'un fichier via des instructions Python, pour par exemple :

- ajouter une ligne en fin de fichier
- ne garder que les lignes d'un fichier qui satisfont une certaine condition
- décider si un fichier contient uniquement des lignes telles que…

Pour faire ces manipulations on a besoin :

- du nom du fichier
- de créer à partir de ce nom un "objet fichier" qui va fournir des fonctions permettant de manipuler le contenu du fichier
- et pour cela de préciser avec quel mode d'action on va manipuler le contenu du fichier : actions de lecture et/ou d'écriture ?

La syntaxe et l'utilisation type sont les suivantes :

```
f = open(<nom_fichier>, <mode>)
... manipulations via f ...
f.close()
```

c'est à dire qu'on ouvre le fichier, on manipule son contenu puis on le ferme.

2 Ouverture d'un fichier : fonction `open`

Le nom du fichier est de type `str`.

Attention ! Vous souhaitez ouvrir le fichier `cours.md`, mais c'est le nom `'cours.md'` qu'il faut passer à `open`.

Le mode d'ouverture du fichier est aussi de type `str`. Le paramètre peut prendre les valeurs suivantes :

Mode lecture d'un fichier texte : `'r'` ou `'rt'` (le `'t'` pour "texte" est optionnel car utilisé par défaut). Dans ce cas on ne peut que lire le contenu du fichier. Dans l'UE on utilisera essentiellement les méthodes `readline` et `readlines`.

Mode écriture d'un fichier texte: `'w'` ou `'wt'` ou `'a'` ou `'x'` (le `'t'` pour "texte" est optionnel car utilisé par défaut). Dans ce cas il n'est possible que d'écrire dans le fichier. Dans l'UE on utilisera essentiellement les méthodes `write` et `writelines`.

Mode lecture et écriture : mode permettant à la fois la lecture et l'écriture d'informations dans le fichier. On n'utilisera pas ce mode dans l'UE, on se limitera à soit lire, soit écrire.

On écrira par exemple :

```
>>> f = open('cours.md', 'w')
```

Quel que soit le mode d'ouverture la fonction `open` renvoie un *file object*, un objet de type `TextIOWrapper` (on ne vous demande pas de retenir le nom de ce type !) :


```
>>> type(f)
<class '_io.TextIOWrapper'>
>>> open('cours2.md', 'r')
<_io.TextIOWrapper name='cours2.md' mode='r' encoding='UTF-8'>
```

Nous verrons dans les prochains chapitres le fonctionnement des méthodes précédemment citées.

3 Fermeture d'un fichier : fonction `close`

La fonction `open` permet de lier un objet fichier au contenu du fichier ouvert, via une sorte de canal. Pour que les manipulations sur le contenu du fichier soient effectives (par ex : ajouter une ligne au fichier), il est nécessaire de fermer proprement ce canal quand les manipulations sont terminées. Sinon les conséquences des manipulations sont perdues.

Essayons d'écrire 2 lignes dans un fichier :

```
>>> f = open('liste_etud.txt', 'w')
>>> f.writelines(['LATUILE;Anatole\n', 'DUBOUCHON;TomTom\n'])
```

Si à ce moment vous éditez le contenu du fichier `liste_etud.txt`, vous n'y verrez pas trace des lignes qu'on vient d'y écrire. Il faut attendre la fermeture par `close()`.

```
>>> f.close()
```

L'édition du fichier montre que les 2 lignes ont été ajoutées.

Après la fermeture du canal, il n'est plus possible d'accéder au contenu du fichier via l'objet fichier :

```
>>> f.writelines(['DUBOUCHON;Nana\n'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

3.1 Forme syntaxique `with`

Python offre une structure syntaxique permettant d'omettre la commande explicite de fermeture (ce qui évite le classique oubli de la fermeture du fichier, et la perte de données consécutive).

Cette forme débute par le mot-clé `with` et utilise le mot-clé `as` :

```
# instr avant
with open(<nom_fichier>, <mode>) as f:
   # traitement sur f
# instr après
```

L'identificateur placé après le mot-clé `as` est le nom donné à l'objet fichier renvoyé par la fonction `open`. À la fin de ce bloc le fichier est automatiquement fermé, sans qu'il soit nécessaire d'invoquer la méthode `close`.

De cette façon, les lignes précédentes sont équivalentes aux suivantes :

```
# instr avant
f = open(<nom_fichier>, <mode>)
# traitement sur f
f.close()
# instr après
```

4 Memento

- La commande `open` crée un objet fichier qui permet de manipuler le contenu d'un fichier
- Il faut préciser le mode d'ouverture pour manipuler le contenu du fichier en lecture ou en écriture
- La fermeture du fichier par `close` est nécessaire pour ne pas perdre de données

On découvre comment lire les lignes d'un fichier texte.

On considère par la suite le fichier `timoleon.txt` qui contient uniquement les trois lignes suivantes:

```
Timoleon est un homme politique grec
ayant vécu au IVème siècle av. JC.
Il est connu pour avoir recolonisé la Sicile.
```

Dans l'UE on verra uniquement comment lire des lignes entières (par exemple `"Il est connu pour avoir recolonisé la Sicile."` mais pas `"Sicile"`).

1 Lecture d'une ligne : fonction `readline`

Quand l'objet fichier est créé par une ouverture en mode lecture (`'r'`) :

```
>>> entree = open('timoleon.txt', 'r')
```

alors on peut appeler sur cet objet la méthode `readline` :

```
>>> entree.readline()
'Timoleon est un homme politique grec\n'
>>> entree.readline()
'ayant vécu au IVème siècle av. JC.\n'
```

On constate que le premier appel à `readline` renvoie la première ligne du fichier, terminée par le marqueur de fin de ligne `\n`. Le second appel renvoie la deuxième ligne du fichier.

Ces 2 appels montrent que l'objet fichier possède un *état* qui est la *position courante* du canal de lecture dans le fichier. Cette position :

- indique initialement le tout début du fichier (la première ligne)
- change après chaque lecture de données
- indique alors la position de la prochaine lecture.

La méthode `readline` renvoie donc la ligne qui suit la position courante dans le fichier, terminée ici par le marqueur de fin de ligne.

Comme on peut s'y attendre, un troisième appel renvoie la troisième ligne :

```
>>> entree.readline()
'Il est connu pour avoir recolonisé la Sicile.'
```

On constate l'absence du marqueur de fin de ligne : la personne ou le code qui a créé le fichier n'a pas terminé la saisie de cette dernière ligne par un retour chariot.

Fin de fichier

Une fois les trois lignes du fichier lues, la position courante est en fin du fichier. Que se passe-t-il si nous invoquons encore la méthode `readline` ?

```
>>> entree.readline()
''
```

Lorsque `readline` renvoie une chaîne vide, c'est qu'on est arrivé à la fin du fichier. C'est une caractéristique qu'il est possible d'exploiter en programmation.

Il ne faut pas confondre la chaîne vide renvoyée par `readline` et une ligne vide contenue dans un fichier texte.

En effet, si l'objet fichier pointe vers une ligne vide du fichier, la méthode `readline` renvoie une chaîne de caractères contenant le seul marqueur de fin de ligne : `'\n'`, chaîne qui n'est donc pas vide.

2 Lecture de toutes les lignes : fonction `readlines`

La méthode `readlines` renvoie la liste de toutes les lignes restant à lire à partir de la position courante.

Si l'objet fichier vient d'être ouvert, cela revient à lire toutes les lignes du fichier.

```
>>> entree.close() # on ferme pour rouvrir
>>> entree = open('timoleon.txt','r')
>>> lignes_lues = entree.readlines()
['Timoleon est un homme politique grec\n', [...], 'Il est connu pour avoir recolonisé la Sicile.']
```

En fin de lecture la position courante est en fin de fichier mais le fichier n'est pas fermé.

```
>>> entree.readline()
''
```

NB La lecture de toutes les lignes n'est possible que si le contenu du fichier tient dans la mémoire de l'ordinateur (ce qui sera toujours le cas dans cette UE).

3 Itération sur les lignes d'un fichier

Les objets fichier créés par `open` sont de type `TextIOWrapper`.

Ce type a la particularité d'être **itérable**, mais pas **indexable**.

Rappel Jusqu'à présent nous avons vu les types liste et chaîne de caractère, qui sont itérables (on peut écrire : `for car in "azerty": ...`) et indexables par un indice de type entier (on peut écrire : `"azerty"[0] == "a"`).

Un objet fichier est itérable, ce qui permet d'itérer en lecture sur ses lignes pour mimer un appel à `readlines`:

```
>>> lignes = []
>>> with open('timoleon.txt','r') as entree:
       for ligne in entree:
           lignes.append(ligne)
```

Mais il n'est pas indexable…

```
>>> lignes = []
>>> with open('timoleon.txt','r') as entree:
       for i in range(3):
           lignes.append(entree[i])
[...]
TypeError: '_io.TextIOWrapper' object is not subscriptable
```

On ne peut pas non plus calculer sa longueur par la fonction `len`:

```
>>> with open('timoleon.txt', 'r') as f:
       for i in range(len(f)):
           [...]

Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
TypeError: object of type '_io.TextIOWrapper' has no len()
```

Le fait que les objets fichiers ne soient pas traités comme des séquences peut surprendre : on constate pourtant que le fichier `'timoleon.txt'` a une première ligne, puis une deuxième ligne, puis une troisième, et contient en tout 3 lignes. Pour mieux comprendre : d'une part l'ouverture d'un fichier en lecture ne déclenche pas la lecture de tout le fichier (et donc on ne connaît pas a priori le nombre de lignes), d'autre part on pourrait lire le fichier caractères par caractères.

4 Ouverture / lecture impossible

Si le fichier qu'on veut ouvrir n'existe pas une exception `FileNotFoundError` est déclenchée.

```
>>> entree = open('nexistepas.txt', 'r')
[...]
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'nexistepas.txt'
```

Il est impossible de lire (quelle que soit la méthode) dans un fichier ouvert en écriture.

```
with open('nouveau.txt', 'w') as sortie:
    lignes_lues = sortie.readlines()
[...]
io.UnsupportedOperation: not readable
```

5 Memento

- En lecture, l'objet fichier contient la position de la ligne courante dans le fichier
- La méthode `readline` renvoie la ligne courante
- La méthode `readlines` renvoie la liste des lignes qui suivent la position courante
- On peut itérer sur les lignes d'un fichier, mais les lignes ne sont pas indexables

CC BY

On découvre

- comment ouvrir un fichier texte en écriture,
- comment écrire des chaînes de caractères dans un fichier.

On verra uniquement comment ajouter du texte en fin de fichier ou au début d'un fichier dont on vient de supprimer tout le contenu ou qu'on vient de créer.

1 Modes d'ouverture et position initiale

Comme avec le mode lecture, un fichier ouvert en écriture possède une position courante qui indique la position de la prochaine écriture. On présente 3 modes d'ouverture en écriture, qui diffèrent par la position courante initiale.

1.1 Ouverture en mode ajout à un fichier existant (ou pas)

Le mode *ajout* (*append*) ouvre un fichier existant en écriture afin d'ajouter de nouvelles données. Toute nouvelle opération d'écriture se fait à la suite du texte existant avant l'ouverture.

Si on reprend le fichier `timoleon.txt` , on pourra ajouter du texte en fin de fichier en commençant par :

```
>>> sortie = open('timoleon.txt', 'a')
```

Si on utilise ce mode avec le nom d'un fichier qui n'existe pas, un nouveau fichier à ce nom est créé.

1.2 Ouverture en mode écriture dans un nouveau fichier (ou pas !!!)

On considère un fichier `memo_python.md` qui contient un nombre non négligeable de lignes. L'instruction

```
>>> sortie = open('memo_python.md', 'w')
```

provoque la **perte immédiate** des données que contenait le fichier[1], sans même attendre la fermeture par `close`. Puis la position courante est le début du fichier (qui est vide…).

On utilise ce mode pour partir d'un fichier vide, que ce fichier existe déjà ou non. L'instruction

```
>>> sortie = open('devoir1.py', 'w')
```

crée sur le disque le fichier `devoir1.py`, qui n'existait pas auparavant.

1.3 Ouverture en mode écriture dans un nouveau fichier

Python propose un mode particulier d'ouverture en écriture, qui vérifie préalablement l'existence ou non du fichier qu'on veut ouvrir. C'est le mode `x`.

En supposant que le fichier `existe.bien` existe bien, on aura:

```
>>> sortie = open('existe.bien', 'x')
...
FileExistsError: [Errno 17] File exists: 'existe.bien'
```

Si le fichier n'existe pas, il est créé sur le disque.

2 Écriture de données

2.1 Écrire des chaînes de caractères

Il n'y a pas d'équivalent à la méthode `readline`. La méthode `write` permet d'écrire n'importe quelle chaîne de caractères dans un fichier, à la position courante et **sans ajouter de caractère de fin de ligne**. La méthode `write` renvoie le nombre de caractères écrits.

[1] Oups ! On aurait peut-être dû l'écrire avant ?

CC BY

```
>>> sortie = open('timoleon.txt', 'a')
>>> sortie.write('Tu savais ça, toi ?\n')
20
>>> sortie.write('Non.' ) # pas de \n en fin de chaîne
4
>>> sortie.write(' Tu crois que c\'est au programme ?\n')
35
>>> sortie.close()
```

La position courante initiale étant la fin du fichier (mode `'a'`), le fichier `timoleon.txt` contient maintenant (noter l'absence de saut de ligne après `Non.`):

```
Timoleon est un homme politique grec
ayant vécu au IVème siècle av. JC.
Il est connu pour avoir recolonisé la Sicile.
Tu savais ça, toi ?
Non. Tu crois que c'est au programme ?
```

2.2 Écrire des lignes

La méthode `writelines` est une méthode analogue à `readlines` : elle permet d'écrire dans un fichier une liste de chaînes de caractères, **sans ajouter de caractère de fin de ligne**.

Si chacune de ces chaînes se termine par un marqueur de fin de ligne, le nombre de lignes écrites dans le fichier est égal à la longueur de la liste.

Considérons les instructions suivantes :

```
>>> sortie = open('timoleon.txt', 'w')
>>> sortie.writelines(['Timoleon est un homme politique grec\n', [...], 'Il est connu pour avoir recolonis
>>> sortie.close()
```

L'ouverture en mode `'w'` efface le contenu du fichier `timoleon.txt`. La position courante est le début du fichier vide, dans lequel on écrit les 3 lignes qui permettent de revenir au contenu du fichier initial.

3 Erreurs en écriture

Il est impossible d'utiliser les méthodes `write` et `writelines` sur un fichier qui serait ouvert en lecture :

```
>>> f = open('timoleon.txt', 'r')
>>> f.write("hé !")
[...]
io.UnsupportedOperation: not writable
```

Il est de même impossible d'itérer sur les lignes d'un fichier ouvert en écriture :

```
>>> f.close()
>>> f = open('timoleon.txt', 'w')
>>> for ligne in f:
...     pass
[...]
io.UnsupportedOperation: not readable
```

4 Memento

- Il existe différents modes d'ouverture en écriture : `a` pour `append`, `x` pour prudence, et `w` (warning ? à manier avec précaution en tout cas) pour écrire dans un fichier vide
- La méthode `write` écrit la chaîne passée en paramètre à la position courante
- La méthode `writelines` écrit une liste de chaînes
- Dans les 2 cas les marqueurs de fin de ligne doivent être explicités.

CC BY