Pratique de Python

Retrouvez tous les codes sources des exemples sur le dépôt GitHub du livre `Python-sans-detour`.

1

Ouvrir et écrire des fichiers

INSTALLATION NÉCESSAIRE

Dans ce chapitre, nous aurons besoin des modules complémentaires suivants :

- `openpyxl`
- `pandas`
- `python-docx`
- `wxpython`

1.1. Organisation des fichiers dans un ordinateur

La grande majorité des utilisateurs accède à leurs fichiers en utilisant le gestionnaire de fichiers graphique de leur système d'exploitation (macOS, Windows, raspbian, etc.). Lorsque l'on clique dans des fenêtres successives pour accéder aux fichiers, on ne se préoccupe pas de la fenêtre initiale, l'important est de trouver le fichier. Parfois même, on utilise l'aide contextuelle ou bien l'historique des fichiers pour accéder à l'élément souhaité. Dans le gestionnaire de fichiers, il est possible de retrouver la localisation du fichier en affichant la barre d'adresse. Par exemple, le fichier `exemple1.py` est situé pour raspbian dans `/home/lberger/Documents/Livre_Python` ; pour Windows, dans `c:\users\lberger\documents\livre_python` et pour macOS (utilisez le raccourci pour afficher la barre d'adresse dans le Finder), dans `/users/lberger/documents/livre_python`).

La localisation du fichier suivie du nom du fichier définissent le chemin complet pour accéder au fichier, `/home/lberger/Documents/Livre_Python/exemple1.py`, `/users/lberger/documents/livre_python/exemple1.py` et `C:\users\lberger\documents\livre_python\exemple1.py` pour respectivement raspbian, macOS et Windows.

Le chemin complet est une chaîne de caractères. Chaque dossier dans cette chaîne est séparée par un / sous Linux ou bien un \ sous Windows. L'origine du chemin est / pour Linux et on l'appelle la racine. Pour Windows, c'est le nom du disque (appelé aussi

volume) suivi du symbole : (deux points) et d'une barre oblique inversée \ (appelée aussi antislash ou backslash) soit par exemple `C:\` où la lettre C est le nom du disque.

Le chemin complet permet de localiser un fichier par rapport à la racine. Il existe aussi la notion de chemin relatif. Un chemin relatif ne part plus de la racine mais d'un emplacement par défaut. Cet emplacement par défaut dépend du contexte du programme. Si le programme est lancé dans un terminal alors le chemin relatif sera donné par rapport au répertoire par défaut du terminal. Si le programme est lancé à partir de la console Python, le chemin sera donné par rapport au chemin où est installé Python.

En programmation, quel que soit le langage informatique choisi, il faut préférer donner le chemin complet pour accéder au fichier qu'on souhaite lire ou écrire.

SYNTAXE POUR LES CHEMINS RELATIFS

Pour définir un chemin relatif, on utilise le symbole `.` (point) pour désigner le répertoire courant et les symboles `..` (point point) pour désigner le répertoire de niveau supérieur.

Par exemple dans la configuration macOS, si le répertoire par défaut est `/users/lberger/Documents/livre_python` alors on peut trouver le fichier `exemple2.py` dans deux répertoires :

`./exemple2.py`
(chemin absolu : `/users/lberger/Documents/livre_python/exemple2.py`)

`../exemple_python/exemple2.py`
(chemin absolu : `/users/lberger/Documents/exemple_python/exemple2.py`)

Sous Windows, il n'y a pas de différence entre majuscules et minuscules pour les noms de fichiers. Ce n'est pas le cas sous Linux, les noms de fichiers sont sensibles à la casse.

Quand on débute en programmation, il est toujours préférable d'utiliser les chemins absolus pour éviter l'erreur `File Not Found`. Avec un peu d'expérience, vous n'aurez aucun problème à passer aux chemins relatifs et à bien vérifier l'orthographe des noms de fichiers lorsqu'une erreur `File Not Found` se produira.

Pour obtenir le répertoire par défaut en Python, il faut importer le module `os` et appeler la fonction `os.getcwd()`.

```
import os

print("répertoire par défaut : ", os.getcwd())
```

Figure 1.1 : *PCManFM : Gnome (raspbian)*

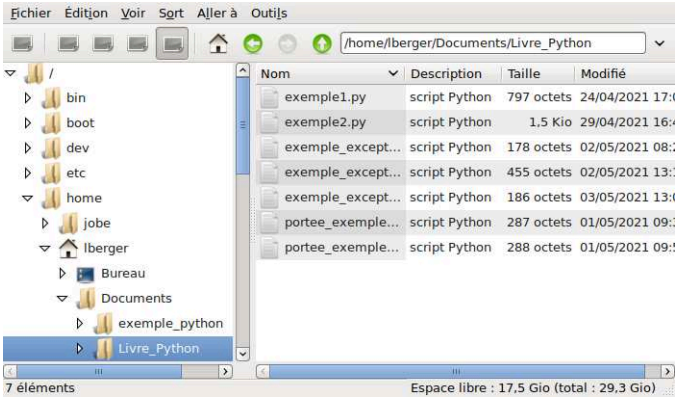


Figure 1.2 : *Finder : macOS(Big Sur)*

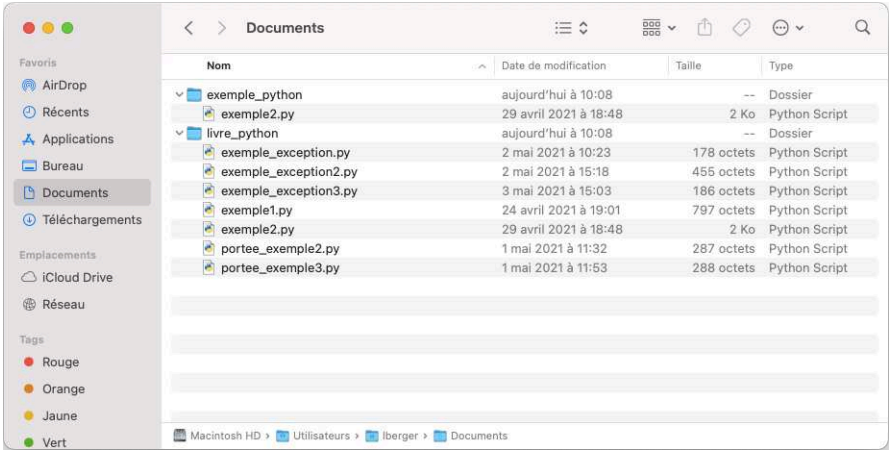
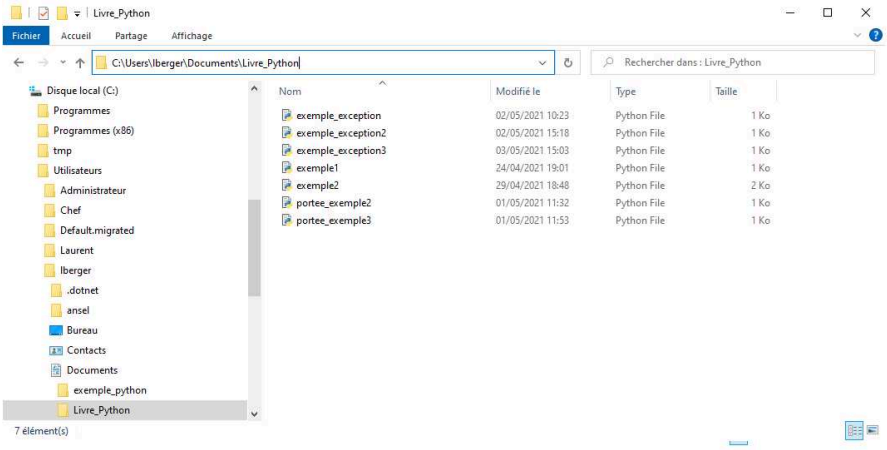


Figure 1.3 : Windows : Explorateur de fichier



La principale différence entre Windows et Linux dans la syntaxe des noms de fichiers est l'usage de \ pour Windows et du / pour Linux. Dans Python, vous pouvez utiliser le / pour Windows. Si vous ne souhaitez pas utiliser le / sous Windows alors vous devez faire précéder la chaîne de caractères définissant le chemin d'un symbole r ou R, par exemple `chemin = r"c:\users\lberger\documents\livre_python\exemple1.py"`.

EXTENSION DU NOM DE FICHIER, SIGNATURE

Un fichier contient des données structurées. Ces données sont structurées selon un certain format. Ce format peut dépendre du logiciel propriétaire (Word ou Excel) ou open-source (Libreoffice) ou bien suivre les recommandations d'un document du type Request For Comment pour la plupart des formats disponibles sur Internet. Pour reconnaître le format des données, les systèmes d'exploitation existants utilisent différentes techniques. Pour Windows, c'est l'extension du nom de fichier, pour macOS c'est la signature MIME et pour Linux c'est le contenu du fichier.

L'extension du nom de fichier est composée des derniers caractères du nom précédés par un point. S'il y a plusieurs points dans le nom, seul le dernier compte. L'extension du nom de fichier est utilisé par le système Windows et elle est souvent cachée par l'Explorateur de fichiers (nom du gestionnaire de fichiers de Windows). On la retrouve souvent sur les autres systèmes d'exploitation où, là, elle n'est pas utilisée.

Quelques exemples d'extensions :

- `.docx` : extension utilisée par Microsoft Office Word depuis la version 2007 ;
- `.xlsx` : extension utilisée par Microsoft Office Excel depuis la version 2007 ;
- `.odt` : extension utilisée par LibreOffice ou OpenOffice pour les documents textes ;
- `.ods` : extension utilisée par LibreOffice ou OpenOffice pour les tableurs ;
- `.py` : extension utilisée pour les scripts Python.

1.2. Lire et écrire un fichier

Codes sources dans le dossier `chapitre_fichier/fichier` 

Toute opération de lecture ou écriture dans un fichier doit être précédée par l'appel à la fonction open permettant d'ouvrir ce fichier. Le premier argument de open doit être le chemin complet ou relatif du fichier. Il est toujours préférable de donner le chemin complet. Le second argument est nommé *mode*. Le mode définit d'abord si le fichier doit être ouvert en lecture, écriture, lecture et écriture ou bien ajout et ensuite quel est le type de fichier soit binaire ou texte.

Tableau 1.1 : *Paramètres*

Mode	Type d'ouverture
'r'	ouverture en lecture
'w'	création d'un nouveau fichier et ouverture en écriture (destruction de l'ancien s'il existe déjà)
'x'	création du fichier, lance une exception si le fichier existe déjà
'a'	ouverture en ajout, création si le fichier n'existe pas
'b'	mode binaire
't'	mode texte
'+'	ouverture pour mise à jour

FICHIER BINAIRE OU TEXTE

- Le résultat de la lecture d'un fichier en mode `text` est du type `str` et les caractères sont décodés selon le paramètre nommé `enconding` ; la valeur par défaut est `locale.getpreferredencoding(False)` soit cp1252 sur Windows et UTF-8 sur macOS et Linux.
- Le résultat de la lecture d'un fichier en mode binaire est du type Bytes et les caractères sont lus sans décodage.

Exemple 1.1 : *fichier_ex1.py*

Le programme suivant ouvre le fichier `essai.txt` dans le répertoire `tmp` en lecture en mode `text` ❶ en appelant la fonction `open` de Python. Le contenu du fichier est lu en utilisant la méthode `read`. Le résultat de la méthode `read` est une chaîne de caractères affectée à la variable nommée `contenu_fichier` ❷ (le fichier est ouvert en mode `text`). Le fichier est ensuite fermé ❸ en utilisant la méthode `close`. Le traitement du fichier est un simple affichage de son contenu ❹.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Premier exemple de lecture de fichier
sans gestion des erreurs
"""
f = open('/tmp/essai.txt', 'rt') ❶
contenu_fichier = f.read() ❷
f.close() ❸
print(contenu_fichier) ❹
```

Exemple 1.2 : *fichier_ex2.py*

Dans l'Exemple 1.1, on n' a pas géré les erreurs et cela n'est pas recommandé lorsque l'on veut avoir un programme fiable. La fonction `open` peut générer une exception lorsque le chemin du fichier ne peut être ouvert (chemin incorrect ou bien le fichier est déjà ouvert dans une autre application) ; la fonction `read` peut aussi générer une exception lorsque les données contenues dans le fichier ne peuvent être décodées.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Second exemple de lecture de fichier
```