**Tâche 1 : Obtenir la liste de toutes les fonctions permettant d’interagir avec des données externes.**

**À noter** : J’ai sélectionné seulement les paramètres que j’ai trouvés pertinents.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Librairie | Nom de la fonction | Principaux arguments | Exemple | Mode de chargement | Commentaires |
| native | open (name, mode) | name (string) : indiquant le nom du fichier à ouvrir  mode (string) : indiquant comment le fichier sera ouvert | open(‘file.txt’, ‘r+’) | Lecture et/ou écriture | La lecture ou l’écriture dépend des valeurs du mode.  ‘r’ : lecture seulement (mode par défaut)  ‘w’ : écriture  ‘x’ : création du fichier s’il n’existe pas déjà  ‘a’ : ouvre en écriture et ajoute à la fin du fichier les éléments à ajouter  ‘b’ : mode binaire  ‘t‘ : mode texte (par défaut)  ‘+’ : ouvre en modification (lecture et écriture)  ‘w+b’ : ouvre et vide le fichier pour un accès en écriture binaire  ‘r+b’ : ouvre le fichier sans le vider pour un accès en lecture binaire  (The Python Software Foundation, 2020) |
| numpy | np.loadtxt (fname, dtype, delimiter)  (Pour fichier texte) | fname (string) : fichier à lire  dtype : type du tableau résultant (le type par défaut est float). Ce paramètre est facultatif  delimiter (string) : utilisé pour séparer les valeurs. Le délimiteur par défaut est l’espace. | np.loadtxt("fichier.txt", dtype=float, delimeter=',')  ou  np.loadtxt("fichier.txt ") | Lecture | Il retourne les données contenues dans le fichier texte qu’on a.  (The SciPy community, 2020) |
| numpy | np.savetxt (fname, array)  (Pour fichier texte) | fname (string) : nom du fichier dans lequel on veut sauvegarder notre tableau  array : tableau à mettre dans le tableau  (The SciPy community, 2020) | np.savetxt("fichier.txt", myArray) | Écriture |  |
| numpy | numpy.load(file, mmap\_mode=None, allow\_pickle=False)  (Pour fichier binaire) | file (string) : fichier à lire  mmap\_mode (char) : le mode d’ouverture. Si rien n’est fourni, alors par défaut le mode est lecture. (facultatif)  allow\_pickle (bool) : permet de charger des tableaux d’objets dans des fichiers npy. (facultatif) | np.load ("fichier.npy", "w") | Lecture | Par défaut, allow\_pickle est à false pour des raisons de sécurité. Il faut changer sa valeur car s’il est désactivé, on ne pourra pas charger de tableaux.  L’instance retournée doit être fermée pour éviter les fuites de descripteurs de fichiers.  (The SciPy community, 2020) |
| numpy | numpy.save (file, arr, allow\_pickle=True)  (Pour fichier binaire) | file (string) : fichier dans lequel les données doivent être sauvegardées.  arr : le tableau de données à sauvegarder  allow\_pickle (bool) : par défaut prend la valeur true. | np.save (fic, x)  avec fic la variable qui reçoit l’ouverture du fichier, et x le tableau à sauvegarder | Écriture | Si le fichier est une chaîne de caractères ou un chemin d’accès, une extension .npy sera ajoutée au nom du fichier s’il n’en a pas déjà.  (The SciPy communit, 2020) |
| os | os.open(file/path, flags, mode) | file (string) : fichier qu’on souhaite ouvrir  flags : spécifie des fonctions qui permettent de savoir comment l’ouverture se fera  mode : permet de savoir si on est en mode texte ou mode binaire | os.open (fichier, os.O\_RDONLY) |  | Les différents flags que nous avons sont :   * Sur Windows et Unix   os.O\_RDONLY: ouverture en lecture seulement;  os.O\_WRONLY: ouverture en écriture seulement;  os.O\_RDWR: ouverture en lecture et en écriture;  os.O\_APPEND: ouverture en écriture pour ajouter du contenu à la fin du fichier;  os.O\_CREAT: création d’un fichier s’il n’existe pas.  os.O\_EXCL: affiche une erreur si l’on crée un fichier qui existe déjà;  os.O\_TRUNC: troncature la taille du fichier à 0.   * Sur Unix uniquement :   os.O\_DSYNC  os.O\_RSYNC  os.O\_SYNC  os.O\_NDELAY  os.O\_NONBLOCK  os.O\_NOCTTY  os.O\_CLOEXEC   * Sur Windows uniquement :   os.O\_BINARY  os.O\_NOINHERIT  os.O\_SHORT\_LIVED  os.O\_TEMPORARY  os.O\_RANDOM  os.O\_SEQUENTIAL  os.O\_TEXT  On peut combiner plusieurs flags en utilisant l’opérateur | |
| utils | utils.load\_data() |  | file=utils.load\_data() | Lecture | J’ai trouvé ceci, mais il y avait plusieurs exemples de paramètres, et parfois il n’y avait pas de paramètres donc je ne sais pas trop.  Voir |
| io | io.open(file, mode) | file (string) : indiquant le nom du fichier à ouvrir  mode (string) : indiquant comment le fichier sera ouvert | io.open (‘fichier’, ‘r+b’) | Lecture et/ou écriture | La lecture ou l’écriture dépend des valeurs du mode.  ‘r’ : lecture seulement (mode par défaut)  ‘w’ : écriture  ‘x’ : création du fichier s’il n’existe pas déjà  ‘a’ : ouvre en écriture et ajoute à la fin du fichier les éléments à ajouter  ‘b’ : mode binaire  ‘t‘ : mode texte (par défaut)  ‘+’ : ouvre en modification (lecture et écriture)  ‘w+b’ : ouvre et vide le fichier pour un accès en écriture binaire  ‘r+b’ : ouvre le fichier sans le vider pour un accès en lecture binaire  (The Python Software Foundation, 2020) |
| json | json.load(file, parse\_float, parse\_int, parse\_constant, object\_pairs\_hook,) | file (string) : un fichier texte ou binaire supportant le .read() et contenant un document JSON  parse\_float : si spécifiée, sera appelée pour chaque nombre réel JSON à décoder sous forme d’une chaîne de caractères.  parse\_int : si spécifiée, sera appelée pour chaque nombre entier JSON à décoder sous forme d’une chaîne de caractères.  parse\_constant, si spécifiée, sera appelée avec l’une des chaînes de caractères suivantes : '-Infinity', 'Infinity' ou 'NaN' | json.load(fichier) | Lecture | Si les données à désérialiser ne sont pas un document JSON valide, une JSONDecodeError sera levée.  Pour le parse\_float / parse\_int, par défaut, elle est équivalente à float/int (num\_str). Cela peut servir à utiliser un autre type de données ou un autre analyseur pour les nombres réels/entiers JSON.  Le parse\_constant peut servir à lever une exception si des nombres JSON invalides sont rencontrés. |
| sklearn | sklearn.datasets.load\_files(container\_path, \*, description=None, categories=None, load\_content=True, shuffle=True, encoding=None, decode\_error='strict', random\_state=0) | container\_path (string ou unicode) :  Chemin d’accès au dossier principal contenant un sous-dossier par catégorie ;  Les paramètres suivants sont facultatifs :  description (string ou unicode) : paragraphe qui décrit les caractéristiques de l’ensemble des données : sa source, sa référence…), default = None ;  categories (ensemble de string) : indique quelles catégories il faut charger, default = None;  load\_content (bool) : indique si on doit charger ou non le contenu des différents fichiers, default = True;  shuffle (bool) : indique si l’on peut mélanger toutes les données ou pas, default = True;  encoding (string) : permet de décoder les fichiers texte en unicode (si load\_content est True);  decode\_error (string) : peut prendre l’une des valeurs (‘strict’, ‘ignore’, ‘replace’) : instruction qui indique quoi faire si une séquence d’octets est donnée pour analyser si les fichiers contiennent des caractères non donnés;  random\_state(int) : détermine une génération aléatoire de nombres pour le mélange de données. | sklearn.datasets.load\_files ("C:\Users\Utilisateur…") | Lecture |  |
| PIL | PIL.Image.open(filename, mode) | filename (string): le nom de l’image qu’on veut ouvrir ;  mode (char) : si spécifié, il devrait être ’r’ | Image.open(r"image.png") | Lecture | Ici, il faudrait préciser qu’on importe Image de PIL (from PIL import Image)  (Je n’ai pas trouvé un exemple dans lequel on utilise PIL dans la syntaxe de la fonction. Tous ceux que j’ai trouvés étaient comme l’exemple fourni) |
| torchvision | torch.load(f, map\_location=None, pickle\_module=<module 'pickle' from '/opt/conda/lib/python3.6/pickle.py'>, \*\*pickle\_load\_args) |  |  |  |  |
| pickle | pickle.load(file) | file (string) : le fichier | pickle.load(file) | Lecture |  |
| csv | open (name, mode, newline=’ ’) | newline : est utilisé parce que le csvfile est un fichier (csvfile est la variable qui reçoit l’ouverture du fichier).  (Python Software Foundation, 2020) | open(‘file.csv’, ‘w’, newline=’ ’) | Lecture et/ou écriture | Il utilise la fonction native open de Python  La lecture ou l’écriture dépend des valeurs du mode.  ‘r’ : lecture seulement (mode par défaut)  ‘w’ : écriture  ‘x’ : création du fichier s’il n’existe pas déjà  ‘a’ : ouvre en écriture et ajoute à la fin du fichier les éléments à ajouter  ‘b’ : mode binaire  ‘t‘ : mode texte (par défaut)  ‘+’ : ouvre en modification (lecture et écriture)  ‘w+b’ : ouvre et vide le fichier pour un accès en écriture binaire  ‘r+b’ : ouvre le fichier sans le vider pour un accès en lecture binaire  (The Python Software Foundation, 2020) |

# Références

* Python Software Foundation. (2020, 06 18). *7. Les entrées/sorties*. Retrieved from Python: https://docs.python.org/fr/3.6/tutorial/inputoutput.html
* Python Software Foundation. (2020, juin 24). *csv — Lecture et écriture de fichiers CSV*. Retrieved from Python: https://docs.python.org/fr/3/library/csv.html?highlight=csv#module-csv
* The Python Software Foundation. (2020, juin 24). *Fonctions natives*. Retrieved from Python: https://docs.python.org/fr/3/library/functions.html#open
* The SciPy communit. (2020, mai 24). *numpy.save*. Retrieved from Numpy: https://numpy.org/doc/stable/reference/generated/numpy.save.html#numpy.save
* The SciPy community. (2020, mai 24). *numpy.load*. Retrieved from Numpy: https://numpy.org/doc/stable/reference/generated/numpy.load.html#numpy.load
* The SciPy community. (2020, mai 24). *numpy.loadtxt*. Retrieved from Numpy: https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html#numpy.loadtxt
* The SciPy community. (2020, mai 24). *numpy.savetxt*. Retrieved from Numpy: https://numpy.org/doc/stable/reference/generated/numpy.savetxt.html#numpy.savetxt

Pour la librairie utils, voir <https://www.programcreek.com/python/example/83230/utils.load_data>