Manipulation des fichiers textes en Python

1. 1. Principes de lecture de fichier textes

1.1. 1.1 Premier exemple : ouvrir et lire la totalité d'un fichier

Copiez-collez le fichier `zoo.txt` dans le même dossier que ce notebook, puis déclenchez les deux cellules suivantes :

```
file  = open("zoo.txt","r",encoding="utf8")
```

```
file
```

La première cellule ouvre le fichier `zoo.txt` en mode lecture seule, comme l'indique le second argument `r` (pour *read*) de la fonction `open()`. Remarquez que le fichier n'est pas encore lu, mais simplement **ouvert** (un peu comme lorsqu'on ouvre un livre, mais qu'on ne l'a pas encore lu). Le **curseur de lecture** est prêt à lire le premier caractère du fichier, et lira les caractères avec l'**encodage** `utf8`. L'instruction `open("zoo.txt", "r",encoding="utf8")` suppose que le fichier `zoo.txt` est dans le répertoire depuis lequel l'interpréteur Python a été lancé. Si ce n'est pas le cas, il faut préciser le chemin d'accès au fichier, soit de manière absolue soit de manière relative.

La seconde cellule nous confirme que la **variable** `file` contient la référence vers la zone de la mémoire vive dans laquelle est stockée le fichier.

```
lignes = file.readlines()
lignes
```

On lit ici la totalité du fichier grâce à la **méthode** `.readlines()`. Cette méthode renvoie un tableau stocké dans la varaible `lignes` où chaque élément est une ligne du fichier texte. Le **curseur de lecture** est déplacé à la fin de la dernière ligne du fichier.

```
file.readlines()
```

Si on tente de relire une fois de plus les données, la méthode `.readlines()` renvoie une liste vide, car le **curseur de lecture** ne trouve plus rien.

```
file.close()
```

La ligne précédente ferme le fichier, plus précisément elle ne renvoie rien mais modifie l'état de l'objet `file` en fichier fermé.

```
file.readlines()
```

Il est donc impossible de lire de nouveau des informations.

Exercice :

1) Quel est le type des éléments du tableau renvoyé par la méthode `.readlines()` ?

2) Ajouter à l'aide de `Notepad++` les animaux `perroquets` et `ornithorynque` à la suite de `zoo.txt`, puis relancer l'ouverture, la lecture et la fermeture du fichier.

3) Lors de la lecture du fichier, apparaissent les caractères `\n`. A quoi correspondent-ils ?

2. 1.2 Lire un fichier ligne par ligne

Il ne vous a pas échappé que l'objet renvoyé par la méthode `.readlines()` renvoie une liste. Il est donc possible d'**itérer** sur cette liste, en utilisant une boucle `for` :

```
file  = open("zoo.txt","r",encoding="utf8")
lines = file.readlines()
for n, line in enumerate(lines) :
    print(f"Ligne {n} : {line}")
file.close()
```

2.1. 1.3 Simplification d'écriture avec le gestionnaire de contexte `with`

Il est nécessaire de faire une remarque importante :

Il est impératif de fermer un fichier dès qu'on en a fini avec lui !

En effet, si le fichier reste ouvert en mémoire vive, il est possible qu'un appel du système d'exploitation modifie la zone mémoire liée à ce fichier, ce qui risque de ne pas enregistrer les changements(au mieux), voire de le rendre illisible (au pire - c'est ce qui arrivait souvent sur les clés USB quand elles étaient mal éjectées).

Pire que ça, si une exception (= erreur) se produit avant l'appel à la méthode `.close()`, le fichier demeurera ouvert en mémoire et... advienne que pourra...

Il est bien entendu possible de prévoir un tel problème avec un bloc `try : ... finally : ...`, de la manière suivante :

```
try :
    file  = open("zoo.txt","r",encoding="utf8")
    print(file.readlines())

finally :
    file.close()
```

Cependant ce type de structure est très lourd à mettre en place, et heureusement les développeurs de Python ont prévu un mécanisme plus simple : le **gestionnaire de contexte** `with`.

```
with open("zoo.txt","r",encoding="utf8") as file :
    print(file.readlines())
```

L'utilisation de `with` garanti la bonne fermeture du fichier, même dans le cas d'exceptions.

2.2. 1.4 Autres méthodes de lecture

En plus de la méthode `.readlines()`, il existe aussi d'autres possibilités pour lire un fichier texte : - la méthode `.read` :

```
with open("zoo.txt","r",encoding="utf8") as file :
    contenu = file.read()
contenu
```

Cette méthode lit l'intégralité d'un fichier comme une unique chaine de caractères.

- la méthode `.readline()`

```
with open("zoo.txt","r",encoding="utf8") as file :
    contenu = file.readline()
contenu
```

Cette méthode lit le fichier ligne par ligne, en déplacant le **curseur de lecture** d'une ligne à chaque appel.

3. 2. Ecriture dans un fichier texte

2.1 Création d'un fichier texte

La méthode d'écriture dans un fichier texte est sensiblement la même que pour la lecture :

```
with open("monPremierFichierTexte.txt", "w", encoding = "utf8") as f :
    f.write("J'écris dans un fichier texte.\n")
```

```
    f.write("J'écris une deuxième ligne dans un fichier texte.")
    f.write("J'écris la suite de la deuxième ligne.")
```

Ici, le gestionnaire de contexte `with` ouvre le fichier en écriture seule, ce qui est signalé par le deuxième paramètre `w` (comme *write*). Le fichier n'existant pas au départ, il est automatiquement créé par Python. Le **curseur d'écriture** est placé au début du fichier.Puis l'appel de la méthode `.write()` écrit la chaine de caractère passée en argument dans le fichier, et place le **curseur d'écriture** à la fin de la dernière chaine de caractère écrite. Notez qu'il n'y a pas de saut de ligne dans le fichier entre les chaines 2 et 3, pusisqu'on n'a pas explicitement demandé un saut à l'aide du caractère spécial `\n`.

```
with open("monPremierFichierTexte.txt", "w", encoding = "utf8") as f :
    f.write("Je réécris dans un fichier")
```

Après exécution de la cellule ci-dessus, on constate l'effacement des données précédentes du fichier. En effet, si le fichier existe déjà, l'ouvertur avec le paramètre `w` va écraser l'ancien contenu, le **curseur d'écriture** étant placé au début du fichier.

3.1. 2.2 Ajouter des chaines à un fichier préexistant

Pour ajouter des chaines de caractères à un fichier préexistant, il faut utiliser la fonction `open()` de la mlanière suivante :

```
with open("monPremierFichierTexte.txt", "a", encoding = "utf8") as f :
    f.write("\n")
    f.write("Maintenant, j'ajoute à mon fichier précédent.")
```

3.2. 2.3 Ajouter et lire des données non textuelles

Comment faire pour ajouter des données numériques à un fichier texte ?

```
with open("monPremierFichierTexte.txt", "a", encoding = "utf8") as f :
    for i in range(100):
        f.write(i)
```

On constate que l'erreur donnée est `TypeError: write() argument must be str, not int`. Il est donc impossible d'ajouter directement des données numériques, il faut d'abord les caster en type `str` :

```
with open("monPremierFichierTexte.txt", "a", encoding = "utf8") as f :
    for i in range(100):
        f.write(f"{i}")
```

Il devient alors possible de passer dans un fichier texte le contenu de variables, en utilisant directement les `f-string` :

```
dico = {"Vache" : "Meuh",
           "Chien" : "Wouf",
           "Chat" : "Miaou",
           "Poule" : "Cot-cot",
           "Ane" : "Hi-Han",
           "Cochon" : "Gruik",
           "Dindon" : "Leon"
           }
with open("sauvDico.txt","w",encoding="utf8") as file:
    file.write(f"{dico}")
```

Pour lire ces données, il suffit alors de relire l'intégralité du fichier et de caster à l'aide de la fonction *built-in* `eval()` :

```
with open("sauvDico.txt","r",encoding="utf8") as file:
    textdic = file.read()
newDico = eval(textdic)
print(type(newDico))
print(f"Le chien fait {newDico['Chien']}")
```

3. Exercices

0.3. 3.1 Exercice 1

Le fichier `Notes.txt` contient les notes reçues par des élèves à un examen de Python.

1. Ecrire une fonction `ouvreNote()` qui ouvre le fichier, et renvoie la liste des notes sous la forme d'une liste d'entiers.
2. Ecrire une fonction `calculeMoyenne()` qui calcule la moyenne du groupe d'élève
3. Ecrire une fonction `admis()` qui crée un nouveau fichier `listeAdmis.txt`, avec en face de chaque note la mention *admis* si la note est supérieure ou égale à 10, ou la mention *recalé* si la note est strictement inférieure à 10.

0.4. 3.2 Exercice 2

Le **Zen de Python** est un ensemble de 19 principes qui influencent le design du langage de programmation Python, et sont utiles pour comprendre et utiliser le langage.

Écrit et publié sur la liste de discussion de Python en juin 1999 par Tim Peters, le Zen de Python a été ensuite publié comme le Python Enhancement Proposal **(PEP) numéro 20**, et est aussi présent sur le site web officiel de Python, en anglais. Il est aussi inclus comme *Easter egg* dans la distribution de l'interpréteur Python, et apparait quand on tape la commande `import this`. (extrait de wikipedia)

Sa traduction en français est donnée dans la cellule suivante :

```
zen = ["Préfère :\n",
       "la beauté à la laideur,\n",
       "l'explicite à l'implicite,\n",
       "le simple au complexe\n",
      "et le complexe au compliqué,\n",
      "le déroulé à l'imbriqué,\n",
       "l'aéré au compact.\n",
        "Prends en compte la lisibilité.\n",
"Les cas particuliers ne le sont jamais assez pour violer les règles.\n",
"Mais, à la pureté, privilégie l'aspect pratique.\n",
"Ne passe pas les erreurs sous silence,\n",
"... ou bâillonne-les explicitement.\n",
"Face à l'ambiguïté, à deviner ne te laisse pas aller.\n",
"Sache qu'il ne devrait [y] avoir qu'une et une seule façon de procéder,\n",
"même si, de prime abord, elle n'est pas évidente, à moins d'être Néerlandais.\n",
"Mieux vaut maintenant que jamais.\n",
"Cependant jamais est souvent mieux qu'immédiatement.\n",
"Si l'implémentation s'explique difficilement, c'est une mauvaise idée.\n",
"Si l'implémentation s'explique aisément, c'est peut-être une bonne idée.\n",
"Les espaces de nommage ! Sacrée bonne idée ! Faisons plus de trucs comme ça."]
```

1. Ecrire une fonction `ecritZen()` qui écrit le contenu de la variable `zen` dans le fichier `zenPython.txt`
2. Ecrire une fonction `inverseZen()` qui lit le contenu du fichier `zenPython.txt`, et **renvoie une chaine de caractère contenant les phrases dans l'ordre inverse**.

0.5. 3.3 Exercice 3

Créer une fonction qui ouvre le fichier `codeJedi.txt`, et qui remplace le mot *Jedi* par le mot *Sith*, les négations *ne* et *pas* étant ajoutées aux bonnes positions, sauf pour la ligne 2 où *pour défendre et protéger* est supprimé.

```
with open("codeJedi.txt", "r", encoding="utf8") as f :
    toto = f.readlines()
```

```
toto
```

```
['Les Jedi sont les gardiens de la paix dans la galaxie.\n',
 'Les Jedi utilisent leurs pouvoirs pour défendre et protéger.\n',
 'Les Jedi respectent la vie, en toutes ses formes.\n',
 'Les Jedi servent autrui plutôt que les dominer, pour le bien de la galaxie.\n',
 "Les Jedi cherchent à s'améliorer à travers le savoir et l'instruction."]
```

```
for ligne in toto :
    print(ligne.split(" "))
```

```
['Les', 'Jedi', 'sont', 'les', 'gardiens', 'de', 'la', 'paix', 'dans', 'la', 'galaxie.\n']
['Les', 'Jedi', 'utilisent', 'leurs', 'pouvoirs', 'pour', 'défendre', 'et', 'protéger.\n']
['Les', 'Jedi', 'respectent', 'la', 'vie,', 'en', 'toutes', 'ses', 'formes.\n']
['Les', 'Jedi', 'servent', 'autrui', 'plutôt', 'que', 'les', 'dominer,', 'pour', 'le', 'bien', 'de', 'la',
'galaxie.\n']
['Les', 'Jedi', 'cherchent', 'à', "s'améliorer", 'à', 'travers', 'le', 'savoir', 'et', "l'instruction."]
```

```
toto
```

```
['Les Jedi sont les gardiens de la paix dans la galaxie.\n',
 'Les Jedi utilisent leurs pouvoirs pour défendre et protéger.\n',
 'Les Jedi respectent la vie, en toutes ses formes.\n',
 'Les Jedi servent autrui plutôt que les dominer, pour le bien de la galaxie.\n',
 "Les Jedi cherchent à s'améliorer à travers le savoir et l'instruction."]
```