Langages de script (Python)
CMTP n° 4 : Fichiers et exceptions

Nous allons mettre en œuvre les structures de données vues lors des séances précédentes pour manipuler des fichiers. Cependant, il faut savoir comment Python traite les exceptions. Vous devez traiter les exceptions lancées lors des lectures ou écritures des fichiers.

I) Exceptions

En Python, les exceptions sont des erreurs détectées durant l'exécution d'un programme. Les exceptions peuvent être traitées à l'intérieur du programme par un gestionnaire d'exceptions. Voir https://docs.python.org/3/tutorial/errors.html pour plus de détails.

```
>>> #exceptions are errors raised during the execution of a statement
>>> int("trois")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    int("trois")
ValueError: invalid literal for int() with base 10: 'trois'
>>> 3/0
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    3/0
ZeroDivisionError: division by zero
>>> "hello"[0] = "H"
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    "hello"[0] = "H"
TypeError: 'str' object does not support item assignment
>>> #exceptions can be handled by try ... except ... statements
>>> try:
...     x = int(input("enter a number\n"))
...     print(x)
... except ValueError:
...     print("That was no valid number.")
...
enter a number
trois
That was no valid number.
>>> #exceptions handlers may have multiple except clauses
>>> try:
...     print("give me two numbers\n")
...     x = int(input("x = "))
...     y = int(input("y (choose different from 0) = "))
...     print("the euclidean quotient is ",x//y)
... except ValueError as err:
...     # an exception is an object that can used in an except clause
...     print("That was not a number")
...     print(err)
... except ZeroDivisionError:
...     print("y was required nonzero")
... finally:
...     # a clause finally is optional,
...     # if present it will run whether or not an exception has been produced
```

```
...     print("the end")
...
give me two numbers

x = 3
y (choose different from 0) = 0
y was required nonzero
the end
>>> raise ZeroDivisionError  #exceptions can be raised by the raise instruction
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError
```

Exercice 1 : Devine le nombre

Écrire un script (programme exécutable) qui choisit aléatoirement un entier n entre 0 et 100 ; demander à l'utilisateur de deviner le nombre. Le programme doit avoir le comportement suivant :

- si l'utilisateur rentre un nombre strictement plus grand (resp. strictement plus petit) que n, le programme affiche `"Nombre␣trop␣grand."` (resp. `"Nombre␣trop␣petit."`) et attend une nouvelle proposition de l'utilisateur ;
- si l'utilisateur devine le nombre, le programme affiche `"Bravo␣!"` et termine ;
- si l'utilisateur appuie sur une touche d'interruption (normalement `Control-C`) le programme termine après avoir affiché le numéro qui devait être deviné ;
- si l'utilisateur rentre une chaîne de caractères qui ne correspond pas à un nombre entier, le programme affiche `"Proposez␣uniquement␣des␣nombres␣entiers"` et attend une nouvelle entrée de l'utilisateur.

Nous aimons la politesse, dans tous les cas possibles le programme doit afficher `"Au␣revoir"` avant de terminer.

II) Fichiers

L'accès aux fichiers en Python se fait à travers des *fileObject*. Les fonctions principales pour traiter ces objets sont :

- `open(filepath, mode)` renvoie un `fileObject` correspondant au fichier de chemin `filepath` avec les différents modes :
  - `'r'` ouvre en lecture (par défaut)
  - `'w'` ouvre en écriture : le fichier est crée s'il n'existe pas, s'il existe il sera écrasé
  - `'a'` ouvre en écriture, ajoutant à la fin du fichier s'il existe
  - `'x'` ouvre en écriture exclusive, en échouant si le fichier existe déjà
  - `'+'` ouvre en modification (lecture et écriture)
  - `'t'` mode fichier texte (par défaut)
  - `'b'` mode fichier binaire
- `fileObject.read()` lit tout le contenu du fichier et le renvoie sous forme d'une chaîne de caractères
- `fileObject.readline()` lit la ligne suivant le curseur dans le fichier et la renvoie sous forme d'une chaîne de caractères, terminé par un `"\n"`. *Astuce :* vous pouvez utilisez la méthode `.rstrip()` des chaînes des caractères pour supprimer le `"\n"` à la fin d'une `str`.
- *à noter :* les fichiers en Python sont aussi des itérables, qui retournent une ligne (lu avec `.readline()`) à la fois. Donc vous pouvez itérer sur les lignes d'un fichier de la façon suivante :

```
f = open("toto.txt")
for line in f:
    print(line.rstrip()) # rstrip avoids printing *two* \n here
```

- `fileObject.readlines()` lit tout le contenu du fichier et renvoie les lignes sous forme d'une liste de chaînes de caractères
- `fileObject.write(str)` écrit une chaîne dans le fichier

- `fileObject.writelines(iterable)` écrit un itérable dans le fichier, en mettant bout à bout les éléments
- `fileObject.close()` ferme le fichier et libère les ressources qu'il utilise.

Pour plus de détails :

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files.

La fermeture d'un fichier doit être assuré aussi en cas de levée d'une exception, avec des blocs `try ... finally` :

```
>>> try:
...     f = open('workfile')
...
... finally:
...     f.close()
>>> # We can check that the file has been closed.
>>> f.closed
True
```

Le mot-clé `with` introduit un *context manager* qui garantit la fermeture automatique du fichier une fois que le bloc de code qu'il contient sera terminé (avec ou sans exception, comme dans le cas de `finally`) :

```
>>> with open('workfile') as f:
...     read_data = f.read()
...
>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

Exercice 2 : Fréquence des mots

Écrire un programme qui prend en entrée le nom d'un fichier texte `fichier.txt` et affiche les cinq mots le plus utilisés dans `fichier.txt` avec le nombre de leur occurrences, sans faire distinction entre lettres majuscules et minuscules, et sans prendre en compte la ponctuation ou les retours à la ligne. Nous considérons "mot" une chaîne de caractères dont tous les caractères sont des lettres alphabétiques (voir la fonction `isalpha()` de la librairie standard de Python). Par exemple sur le fichier `proust.txt` à télécharger de moodle le programme doit afficher :

```
de 7774
la 3913
et 3853
à 3615
que 3123
```

(*Suggestion : construire une dictionnaire dont les clés sont les mots du fichier en minuscule et les valeurs sont le nombre de leur occurrences. Attention, assurez vous que la construction du dictionnaire est linéaire dans la taille du fichier. Pour traiter les arguments passés à un script Python par une ligne de commande, regardez la documentation de* `sys.argv`*. Pour traiter les chaînes de caractères rappelez vous des fonctions du module* `str`*, comme par exemple* `str.split()`*,* `str.strip()`*,* `str.lower()`).

Exercice 3 : Fréquence des mots 2

Modifier le programme de l'exercice 2 pour que, lorsque on l'appel avec en premier paramètre le nom d'un fichier texte `fichier.txt` et en deuxième paramètre un entier `n`, le programme affiche les cinq mots le plus utilisés dans `fichier.txt` *ayant une longueur strictement supérieure à* `n`. Par exemple sur le fichier `proust.txt` et l'entier `5` le programme doit afficher :

```
encore 330
odette 256
verdurin 252
jamais 247
quelque 237
```

Exercice 4 : Spell checker

Télécharger depuis le `moodle` du cours le fichier `dico_light.txt` contenant une version réduite des notes du dictionnaire de la langue française. Écrire un script qui prend en entrée le nom d'un dictionnaire (notamment `dico_light.txt`) et le nom d'un fichier texte `texte.txt` et qui crée un nouveau fichier `texte_avec_erreurs.txt` où tous les mots de `texte.txt` qui n'apparaissent pas dans le dictionnaire sont entourés par `**`. Par exemple si `texte.txt` contient :

```
ceci est un texte
edité par un italien qui parle
plutot mal le français, desolé !
attention aux erreurs à coté de la ponctuatio.
```

Le programme doit créer un fichier texte contenant :

```
ceci est un texte
**edité** par un italien qui parle
**plutot** mal le français, **desolé** !
attention aux erreurs à coté de la **ponctuatio.**
```

Attention à ne pas faire de différence entre lettres majuscules et minuscules et à prendre en compte la ponctuation. Vous pouvez supposer que le mots sont séparés au plus par un espace ou un changement de ligne.

Plusieurs modules sont fournit dans la bibliothèque standard de Python pour accéder et manipuler le système des fichiers, par exemple :

- `os` fournit une façon portable d'utiliser les fonctionnalités dépendantes du système d'exploitation ;
- `os.path` fournit une façon portable pour manipuler les chemins de fichiers ;
- `shutil` propose des opérations de haut niveau sur les fichiers, comme par exemple copier, déplacer, supprimer, etc.

Exercice 5 : Modifications récentes

Écrire un script `recent_files.py` qui prend en entrée un chemin d'accès `path` à un répertoire et affiche la liste des dix fichiers (s'il y en a) qui ont été modifiés le plus récemment contenus (directement) dans le répertoire `path` ensemble à la date de leur dernière modification.
(*Suggestion : vous pouvez utiliser* `os.path.getmtime(path)` *pour obtenir le nombre de secondes écoulés depuis la dernière modification depuis epoch. Le module* `time` *vous fournit les fonctions pour convertir cette quantité dans une date lisible. Aussi les fonctions suivantes peuvent vous être utiles :* `os.listdir(path)`, `os.path.isfile(path)` *et* `os.path.isdir(path)`.)

Exercice 6 : Modifications récentes (récursif)

Modifier le script de l'exercice 5 afin que le programme recherche les fichiers dans le répertoire dont le chemin `path` est passé en paramètre et, récursivement, dans tous les sous-répertoires ayant `path` comme préfixe. Le script affichera toujours les dix fichiers qui ont été modifiés le plus récemment ensemble avec la date de leur dernier changement en format lisible. Astuce : utilisez l'itérateur `os.walk()`.