Echauffement

Echauffement

Écrivez un script qui demande à l'utilisateur une lettre. Si l'utilisateur a entré plus qu'une lettre, le programme lui demandera de recommencer.

Une fois que l'utilisateur à entrer 6 lettres, le programme affichera les 6 lettres ensembles, comme si elle formait un mot.

Fichier

Ouvrir et Fermer

Lorsque vous voulez accéder à un fichier, vous avez besoin de l'ouvrir.

Pour cela vous devez affecter le fichier à une variable comme suit:

```
my_file = open("mon_fichier.txt")
```

Ouvrir et Fermer

Et donc comme vous avez ouvert le fichier avec la fonction `open`, vous allez devoir aussi le fermer:

```
my_file = open("mon_fichier.txt")
…
my_file.close()
```

C’est entre l’ouverture et la fermeture de votre code que le traitement du fichier ce passera.

Lire un Fichier

Fichier et Mode

Quand vous ouvrez un fichier, vous devez choisir un mode d'accès. Il en existe deux principaux, la lecture ("r") et l'écriture ("w").
Le mode d'accès à votre fichier se décrit par une lettre que vous placez après le nom de votre fichier dans la fonction `open`. Le mode lecture se décrit par la lettre "r" pour *read*.

```
my_file = open("mon_fichier.txt", "r")
```

Fichier et Mode

Le mode par défaut étant la lecture vous ne devez donc pas le préciser quand vous ouvrez le fichier, mais par contre tout les autre modes doivent l'être.

```
my_file = open("mon_fichier.txt", "w")
```

Ici le fichier est ouvert en mode écriture symbolisé par la lettre "w" pour *write*.

Lecture

Dans le mode lecture("r"), il y plusieurs façon d'accéder au contenu du fichier.

la méthode `read` par exemple reverra tout le contenu (non-lu) du fichier.

```
my_file = open("mon_fichier.txt")
data = my_file.read()
my_file.close()
```

Ici à la fin de ce script la variable `data` contiendra l’entièreté du contenu du fichier sous forme de *string*

A vous de jouer

Écrivez un script qui prend le contenu du fichier "file_1.txt" et l’affiche à l’écran.

N’oubliez pas de fermer le fichier après l’avoir lu.

Lignes

Ligne

Une autre méthode pour lire un fichier est d'utiliser la méthode `readline`, cette méthode vous renverra la prochaine ligne non-lue du fichier.

```
my_file = open("mon_fichier.txt")
data = my_file.readline()
my_file.close()
```

Ici la variable `data` contiendra la première ligne du fichier (toujours sous forme de string).
Si il ne reste plus de ligne à lire `readline` renverra `None` à la place

Lignes

La troisième méthode que l'on verra pour lire un fichier est d'utiliser la méthode `readlines`, cette méthode vous renverra toutes les lignes non-lues du fichier, sous forme de liste de *string*.

```
my_file = open("mon_fichier.txt")
data = my_file.readlines()
my_file.close()
```

Ici la variable `data` contiendra toutes les lignes du fichier sous forme de `list` de `str`.

A vous de jouer

Le fichier "file_2.txt"contient des lignes avec des nombres.

Écrivez un script qui va lire toutes les lignes du fichier "file_2.txt" et affichera le plus petit et le plus grand nombre.

N’oubliez pas de fermer le fichier après l’avoir lu.

Ecrire dans un Fichier

Write

Quand vous accédez à un fichier en mode écriture ("w"), vous pouvez écrire dans le fichier (logique).

Contrairement au mode lecture, qui entrerait en erreur si vous lui demander de lire un fichier qui n’existe pas, écrire dans un fichier non-existant, le crée.

Par contre écrire dans un fichier existant en écrase son contenu.

Write

```
my_file = open("mon_fichier.txt", "w")
my_file.close()
```

Ces lignes créent le fichier "mon_fichier.txt" si il n’existe pas et en efface son contenu si il existe.

Write

Pour écrire dans un fichier vous aurez besoin de la méthode `write`, qui va écrire dans fichier ce que vous lui passez en paramètre.

```
my_file = open("mon_fichier.txt", "w")
my_file.write("test")
my_file.close()
```

Dans cette exemple le fichier "mon_fichier.txt" contiendra "test" à la fin de l'exécution de ces lignes.

Petit test

Utiliser plusieurs fois la méthode write à la suite écrit dans le fichier à la suite de ce que vous avez écrit depuis son ouverture.

Reprenez l’exercice d’échauffement, et en plus de l'exécution du code actuel, faite en sorte qu’à chaque fois qu’une lettre est choisie, elle soit écrite dans fichier (nom du fichier à votre discrétion).
Une fois que le programme a fini de tourner, allez voir le contenu du fichier.
Que constatez-vous?

Petit test

En effet les lettres qui ont été enregistrée sont colée les une aux autres sans passage à la ligne.

Il vous faudra donc passer un retour à la ligne ("\n") en plus de votre ligne si vous voulez écrire sur des lignes différentes.

"A"??

Encore un petit test

Reprenez votre test précédent, mais à la place du "w", écrivez un "a". Lancer le programme au moins deux fois.

Que pouvez-vous déduire de ce nouveau mode "a"?

Ecrire autrement

Le mode d’écriture ajout ou *append* ("a") se comporte comme le mode d’écriture standard sauf qu’il n'efface pas le contenu du fichier avant d’écrire à l’intérieur.

Cela peut-être pratique lorsque l’on veut conservé ce qui a été écrit dans les précédentes exécution du code.

Et pour finir

Exercice - 1

Reprenez le fichier "file_2.txt"

Ecrivez un programme qui va écrire dans un autre fichier les nombres du fichier "file_2.txt" qui sont divisibles par 7.
rappel pour voir si x est divisible par y , on peut tester `x % y == 0`

Attention à bien fermer les fichiers après utilisation.

Exercice - 2

Écrivez un script qui pose une question à l'utilisateur. (Par exemple, on peut lui demander son âge).

Stockez les réponses dans un fichier (peu importe le nom). Une ligne égale une réponse.

Important: les réponses précédemment données doivent être conservées.

N'oubliez de fermer le fichier après usage.

Exercice - 3

Créer un fichier avec trois questions (une ligne par question).

Ensuite écrivez un script qui va poser ces trois questions (en allant les chercher dans le fichier). Le script va écrire dans un autre fichier la question suivi de la réponse.
ex:

Fichier questions:

```
Quel âge as-tu?
Qu'as-tu mangé à midi?
Aimes-tu les chats?
```

Fichier réponses:

```
Quel âge as-tu? 38
Qu'as-tu mangé à midi? des pâtes
Aimes-tu les chats? OUI
```

Split

Split

Quand vous avez une *string*, vous pouvez utiliser la méthode `split` pour la diviser plusieurs sous *string*.

La méthode split prend un `str` en paramètre et vous renverra une list composé de sous-liste équivalent à votre *string* divisé à chaque fois que la *string* en paramètre apparaît.

(plus d’explication ci-après)

Split

Si j’ai la chaine de caractère "Ceci est un test" et que vous utilisez split de la manière suivante.

```
test = "ceci est un test"
test_split = test.split("e")
```

A la fin de l'exécution de ses deux lignes, la variable `split_test` contiendra la list suivante.

```
["c", "ci ", "st un t", "st"]
```

On notera que les lettres ayant servie de “séparation” ont disparues.

Split et Fichier

La méthode split est souvent utilisé pour fractionner les lignes en sous éléments.

Par exemple si on a un fichiers avec des coordonnées (x, y). le fichier pourrait se présenter comme suit:

```
1, 4
2, 5
3, 6
```

Dans ce cas de figure un `split(",")` pourra séparer les différent partie de la coordonnées afin de mieux les traitées.

Exercice - 4

Dans le fichier file_3.txt des lignes avec 5 nombres séparé pas un espace (“ ”).

Ecrivez un programme qui va aller additionner tout les nombres d’une ligne et aller l’écrire dans un autre fichier.

Chaque résultat sera séparé dans le fichier par un tiret (“-”).

Exercice - 5

Dans le fichier file_5.txt il y a une ligne qui contient l’état d’une liste dont chaque élément de la liste étant séparé par un espace. Récupérez les éléments stockés et mettez les dans une liste, ensuite afficher la liste.

Le programme va demander à l'utilisateur là où il veut insérer sa donnée et ensuite quelle donnée il veut insérer.

Une fois que la donnée est insérée sauver l’état de la liste dans le fichier file_5.txt (en une ligne, chaque élément séparé par un espace).

(cette page a été corrigée par Mihai)