File Input and Output

In this section we will learn to read and write to text files, handle file paths and rename a set of files.

Python has built in tools to work with files

Writing Output to a File

Create a new file in the current directory using the open() function, give it a title and open it in write mode

The using the .writelines() function we can write text into the new file and then close it

The new file will appear in the same folder as this program script and will contain the text passed to it.

In [1]:
```
output_file = open("hello.txt", "w")
output_file.writelines("Just some practice text.")
output_file.close()
```

One way to collect what you plan to write is in a list.

Run the code below and take a look at the file, what happened?

In [2]:
```
lines_to_write = [
    "This is my file",
    "\nIt may be just a text file",
    "\nBut it is more mine than most"
]

output_file = open("hello.txt", "w")
output_file.writelines(lines_to_write)
output_file.close()
```

Opening a file write mode ("w") any existing content in the file is deleted.

In order to retain previous content use the append mode ("a")

In [3]:
```
output_file = open("hello.txt", "a")
output_file.writelines("\nI don't want to start again each time")
output_file.close()
```

Reading from a file

We use the file read mode ("r") to read a files contents as input

Note that each line in the files is stored as an element in a list

In [4]:
```
1 input_file = open("hello.txt", "r")
2 print(input_file.readlines())
3 input_file.close()
```

```
['This is my file\n', 'It may be just a text file\n', 'But it is more min
e than most\n', "I don't want to start again each time"]
```

We can loop over the elements in a list and print them.

In [5]:
```
1 input_file = open("hello.txt", "r")
2
3 for line in input_file.readlines():
4     print(line)
5
6 input_file.close()
```

```
This is my file

It may be just a text file

But it is more mine than most

I don't want to start again each time
```

We can avoid the unwanted space between the new lines by adding an empty string to the end of each element in the list when we print.

In [6]:
```
1 input_file = open("hello.txt", "r")
2
3 for line in input_file.readlines():
4     print(line, end = "")
5
6 input_file.close()
```

```
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time
```

We are also able to use a while loop, because the .readline() function will return an empty string when it reaches the end of the file, so we can use that as a condition for our while loop.

In [7]:
```
input_file = open("hello.txt", "r")

line = input_file.readline()
while line != "":
    print(line, end = "")
    line = input_file.readline()

input_file.close()
```

```
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time
```

If we try and loop through the file and print it a second time, what happend?

This is because the position of where the file is read from does not return to the beginning unless you close and reopen the file.

In [8]:
```
input_file = open("hello.txt", "r")

print("First try:")
for line in input_file.readlines():
    print(line, end = "")

print("\n\nSecond try:")
for line in input_file.readlines():
    print(line, end = "")

input_file.close()
```

```
First try:
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time

Second try:
```

On way to solve this issue is to first store what is returned from the .readlines() function into a variable. Another way is by using the .seek() function, which we'll see later.

In [9]:
```
input_file = open("hello.txt", "r")

lines_to_store = input_file.readlines()

print("First try:")
for line in lines_to_store:
    print(line, end = "")

print("\n\nSecond try:")
for line in lines_to_store:
    print(line, end = "")

input_file.close()
```

```
First try:
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time

Second try:
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time
```

The with keyword

The with keyword makes you code more easy to read and understand. Also it atomatically closes the files for you without having to write the .close function.

In [10]:
```
with open("hello.txt", "r") as input_file:
    for line in input_file.readlines():
        print(line, end = "")
```

```
This is my file
It may be just a text file
But it is more mine than most
I don't want to start again each time
```

The with keyword also lets you open multiple files at once

In [11]:
```
with open("hello.txt", "r") as source_file, open("goodbye.txt", "w") a
    for line in source_file.readlines():
        destination_file.write(line)
```

The .seek() Function

You can use the .seek() function to go to a specific part of the file

In [12]:
```
input_file = open("hello.txt", "r")
print("Line 0 (first line):", input_file.readline())

input_file.seek(0) # Jump back to beginning
print("Line 0 again:", input_file.readline())
print("Line 1:", input_file.readline())

input_file.seek(8) # Jump to character at index 8
print("Line 0 (starting at 8th index):", input_file.readline())

input_file.close()
```

```
Line 0 (first line): This is my file

Line 0 again: This is my file

Line 1: It may be just a text file

Line 0 (starting at 8th index): my file
```

Working with Paths in Python

In order to work with files in different directories than the one your script is saved in, you'll need to work with paths.

The os Module

This module contains many functions related to the operating system, it also helps you work with file paths and structures. Many of it's functions are similar to the terminal commands we convered earlier (such as rmdir() and mkdir())

In [13]:
```
import os

os.mkdir("challenge_directory")

# The above line creates a new
# directory in the directory your script is saved
```

In [18]:
```
import os

# Go to the directory in which you
# want to create a new directory, type pwd and copy the output
path = "/Users/pmatthen/Documents/Programming/valley_bootcamp/File Inp
os.mkdir(os.path.join(path, "My Directory"))

# While you could concatenate the two strings together,
# the os.join() function ensure the correct
# number of forward slashes are added.
```

To remove a directory, you can use the os.rmdir() function.

In [19]:
```
import os

path = "/Users/pmatthen/Documents/Programming/valley_bootcamp/File Inp
os.rmdir(os.path.join(path, "My Directory"))
```

Modifying Files

In this example we will use

1. The os.listdir() function to get a list of file names in the specified directory.

2. The string method .endswith() to check the files extension

3. The os.rename) function to rename each file

Note: Ensure you have downloaded the files for this section and shifted them to where you script will be written. Again, use the pwd command in the terminal to be able to copy the path.

In [22]:
```
import os

path = "/Users/pmatthen/Documents/Programming/valley_bootcamp/File Inp

for file_name in os.listdir(path):
    if file_name.lower().endswith(".jpg"):
        full_path = os.path.join(path, file_name)
        new_file_name = full_path[:-4] + "_backup.jpg"
        os.rename(full_path, new_file_name)
```