Python, Day 8: Files

Andrew Bydlon

January 22, 2019

Why are we interested in files?

Up until now, all of our data has been stored in RAM. The advantage of this is that our data is stored in the fastest memory possible. The disadvantage of this is that all of the data disappears once the program terminates.

Therefore, it seems reasonable that we would want to store data outside of RAM. This is is colloquially referred to as **File Handling**.

Syntax (Opening Files)

PythonObject = open(filename, mode)

Modes

There are a few possibilities for modes for opening files:

1. There are 2 primary types of files accessible in python: text files, and binary files.
   - Text files are sequences of standard characters.Examples include .txt files, .py files, and .html files.
   - Binary files are the raw data read directly by memory. Examples include .mp3, .jpg, and .doc files.
2. In addition, there are 3 ways to open a file:
   - read mode: As it sounds, makes no modifications to the file.
   - write mode: Destroys the data of the file as it exists, but allows writing new data to the file. If it doesn't exist, it creates the file.
   - append mode: Adds new data to an existing file. If it doesn't exist, it creates the file.

Some examples

Example

```
> > > File1 = open("MyFile.txt", "rt")
# Python defaults to text files. So this is equivalent to
> > > File1 = open("MyFile.txt", "r")
> > > File2 = open("MyOtherFile.txt", "w")
> > > File3 = open("MyThirdFile.txt", "a")
> > > File4 = open("MyBinary.dat", "rb")
# All of the above use relative paths (stored in the project folder). You can also use Absolute paths: Linux and Windows respectively.
> > > File5 = open("~/Documents/README.md", "w")
> > > File6 = open("C:\\Users\\andrew\\Documents\\file.md", "w")
> > > import os
> > > os.path.join("~", "Documents", "File.txt")
'~/Documents/File.txt'
```

Closing Files

When you finish with a file, or want to open it in a different mode, it is best practice to close the file.This is done with the close function:

Syntax

MyFile.close()

A simple example of why this is necessary is as follows:

1. f = open("MyFile.txt", "w"): Start the file fresh
2. f.close()
3. f = open("MyFile.txt", "r"): Read off the data from the file

Available method functions

When you open a file as above, it is opened with type _io.TextIOWrapper. As with all other types we have studied, this has many method functions associated to it:

- read(num): Reads num-many characters from the file, as a string. If no [num] is given, it reads the whole file!
- readline(num): Reads the first num characters (or all) a single line, as a string. If executed in succession, reads the next line
- readlines(num): Reads whole lines totalling up to num many bytes, and returns a list. If no num is given, it reads the whole file.
- write(str): Writes str to the file, and returns number of characters written.
- seek(offset, origin): Moves the pointer to the given offset from origin.Origin defaults to 0 if omitted.
- tell(): Returns pointer position.

Example

Example

```
> > > MyFile = open("MyFile.txt", "w")
> > > MyFile.write("Line 1\n")
> > > MyFile.write("Line 2\n")
> > > MyFile.write("Line 3\n")
> > > MyFile.close()
> > > MyFile = open("MyFile.txt", "r")
> > > MyFile.read()
'Line 1\n Line 2\nLine 3\n'
> > > MyFile.read()
''

> > > MyFile.seek(0)
> > > print(MyFile.read())
Line1
Line2
Line3
```

Checking existence of a file

You can check whether a file exists with the os.path.isfile() command.Then you can open a file in read mode if it exists.

Example

```
> > > os.path.isfile("/home/andrew/Documents/MyFile.txt")
True
> > > os.path.isfile("/home/andrew/Documents/MyNoFile.txt")
False
```

Then we can continue by reading the file.

Reading a file

Example

```
> > > MyFile = open("/home/andrew/Documents/MyFile.txt", "r")
> > > print(MyFile.read(5))
Hello
> > > print(MyFile.read(10)
, World!

> > > print(MyFile.read())
My name is Andrew!
> > > print(MyFile.read())
```

Different Reading Method

Example

```
> > > MyFile.seek(0)
> > > print(MyFile.readline())
Hello, World!

> > > print(MyFile.readline())
My name is Andrew!
```

With the readline() function, we read the entire line of MyFile.

Append Mode

Example

```
> > > MyFile = open("/home/andrew/Documents/MyFile.txt", "a")
> > > MyFile.write("I am a Mathematician")
20
> > > MyFile.write(" specializing in algebra.")
25
> > > MyFile.close()
My name is Andrew!
> > > MyFile = open("/home/andrew/Documents/MyFile.txt", "r")
> > > print(MyFile.read())
Hello, World!
My Name is Andrew!
I am a Mathematician specializing in algebra.
```

Other Notes

Note that certain commands don't work in certain modes.

1. read mode: access to read, readline, readlines, seek, tell

2. write/append mode: access to write, seek, tell

When you seek in write mode, you can specify where you are writing new data.It overwrites the data that is currently in that position. Append mode, on the other hand, will only write text to the end of the file, independent of seek position.

Assignment 13

Modify your (or my) submission from Class 5 Part 1 as follows: Instead of taking the list in python, ask the user for a file. The file will have 2 lines; the first principal investments separated by spaces, the second time in years separated by spaces. Write the list of annuities to a new file of the users choosing.

As a hint, you can use the .readline() command to get the lines of data from the file. This will give you a string. Then .strip("\n") to remove the end line command. Finally .split() will give a list, and MyList = list(map(float,MyList)) returns a list of floating point values.

Upload the file when you finish.