Python: File Input/Output

Computer Science ICS20

Opening Files

Up until now, we have been getting all of our external input from the standard keyboard input and all of our output has been to the standard console output.

But before you can read or write to a file, you have to open it using the **open()** function. This function returns a **file** object which can then be used to call other functions that will act on the file.

So to open a file we issue the following command:

```
file = open("myFile.txt")
```

For this command to successfully return an opened file object, the file specified as a parameter must actually exist in the same directory as the Python program file. If it does not, then we need to specify the path to the file, something like this:

```
file = open("data files/text/myFile.txt")
```

The above examples will open the requested file for reading only, which is the default access mode, but that can change. In fact, if you specify a second parameter after the filename, it is interpreted as the access mode. Here is a list of the different access modes:

Mode	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The *file* Object Attributes

As mentioned earlier, after you have opened a file you will have a file object. This object has a few attributes that may be useful:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Note that these are attributes and not functions. (There are no round brackets!) You can think of attributes as variables associated with the file.

Closing files

After we are finished with a file it should be closed using the close() function. Here is an example of a program that opens a file, prints out its name using the file.name attribute and lastly, closes the file.

```
file = open("myFile.txt")

print("The name of the file is " + file.name)
file.close()
```

The close() function of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Writing to Files

The write() function writes a string to an open file. Python strings can have binary data as well as text. The write() function does not add a newline character ('\n') to the end of the file. Here is an example of writing to a file:

```
file = open("myFile.txt", "w")
file.write("Python is a great programming language!")
file.close()
```

The above function would create a new file called myFile.txt, write the given string to it and then close it. If you opened this file using a text editor it would have the following content:

```
Python is a great programming language!
```

Reading Files

The read() function reads data from an open file. Keep in mind that Python strings can have binary data as well as text. Here is an example of reading from the file that was created above:

```
file = open(“myFile.txt”, “r”)
str = file.read(15)# read 15 characters (or bytes) from the file
print(“The string that was read from the file is: “ + str)
file.close()
```

The above program would print:

```
The string that was read from the file is: Python is a gre
```

If you try to read more data from the above file, it would continue reading from where it left off since the last read() call. So the following additional code:

```
str = file.read(5) # read 5 characters from the file
print(str)
```

The above code would print:

```
at pr
```

which are the next 5 characters from the file.

File Position

As we saw from the previous section the read() function reads some number of bytes from the file and leaves the file position at the last byte read. If you want to know what the current file position is then the **tell()** function will return that.
If you want to go to a certain position in the file, then you can use the **seek()** function. The following program shows these functions in use:

```
file = open("myFile.txt", "r+")
str = file.read(10);
print("Read String is : ", str)

# Check current position
position = file.tell();
print(Current file position : ", position)

# Reposition pointer at the beginning once again
position = file.seek(0, 0);
str = file.read(10);
print("Again read String is : ", str)
# Close opened file
fo.close()
```

The output of the above program is:

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Some other file input/output functions that you may find useful are listed below:

Function Name	Description
rename(current_file_name, new_file_name)	The ***rename()*** function allows you to rename files. It takes two arguments, the current filename and the new filename.
remove(file_name)	You can use the ***remove()*** function to delete files by supplying the name of the file to be deleted as the argument.
mkdir("newdir")	You can use the ***mkdir()*** *function* to create directories in the current directory. The name of the new directory must be specified.
chdir("newdir")	You can use the ***chdir()*** function to change the current directory. The **chdir**() function takes an argument, which is the name of the directory that you want to make the current directory.
getcwd()	The ***getcwd()*** function displays the current working directory.
rmdir()	The ***rmdir()*** method deletes the directory, which is passed as an argument in the method. Before removing a directory, all the contents in it should be removed.

Exercises

1. Write a Python program that generates 100 random numbers between 0 and 100 inclusive, and writes them to a file so that there are 10 numbers per line separated by a single space. Name the file 100Numbers.txt. Check your file after your program has created it by opening it with any text editor.
2. Write a Python program that reads the file that you created in question 1, and outputs the average, mean, maximum and minimum values in the file.