**Writing to a File (File Output)**

Suppose you want to create a new file and write to it. A file can be opened and written to using the open() and write() commands.

f = open('blah.txt', 'w') # opens blah.txt

The open() statement creates a file called "blah.txt" and returns a file handle which is stored in the variable f. If the file exists already, it will be erased and overwritten. If the file is new, it will be opened and given a filename and a file handle.

The variable f can be any name:

myFile = open('blah.txt', 'w') # opens blah.txt

The ***filename*** in this case is ‘blah.txt’. The ***file*** ***mode*** is given by the letter w which stands for “write”. As we will soon see, you can also read the file. The **file** **handle** is named myFile. It is our way of referring to the file.

Now that we've opened the file, we write to it and then close it:

myFile = open('blah.txt', 'w') # opens blah.txt

myFile.write('blah blah blah') # writes to the file

myFile.close()

The first line opens and creates the file on your hard drive. The second line writes some gibberish to the file. The third line closes it. The last line is often forgotten, but it is important! If you do not close the file then you cannot access it.

Once you run the program, Replit will create a new file named “blah.txt”. You will see this file in a tab under the blue main.py tab to the left of your screen in Replit.

If you click on the “blah.txt” tab, it will open the file. It should contain one line:

blah blah blah

A file is not finished with until it is closed. To emphasize the importance of closing a file, try this:

myFile = open('blah.txt', 'w') # opens blah.txt

myFile.write("Yeeeeeaaaaaah!!") # writes to the file

It should run no problem. Now locate "Blah.txt" in the directory again and open it... oh oh!

Now go back and add the line that closes the file:

myFile = open('blah.txt', 'w')

myFile.write("Yeeeeeaaaaaah!!")

myFile.close() # don't forget to close it!

Run it again and go back to the directory and open the file. Ah!

By the way: notice that it erased the old version of the file that had "blah blah blah" on it. Every time you open a file, you are erasing the old version of that file.

**Writing Numbers to a File**

You cannot write numbers to a file, only text:

f = open('blah.txt', 'w') # opens blah.txt

x = 24

f.write(x) # error! - can't write a number

f.close()

To write numbers, they must be converted to a string first:

f = open('blah.txt', 'w') # opens blah.txt

x = 24

f.write(str(x)) # ok

f.close()

**Appending to a File**

Every time you write to a file, the old file is erased. Sometimes you want to *add* to a file but not write over it. To do this, open the file in the ***append*** ***mode***. The word “append” means “*to attach to the end”*.

f = open('blah.txt', 'a') # opens blah.txt in append mode

f.write("hello") # adds this to the file

f.close()

Try running this program 3 times (hit the play button 3 times) and then opening the text file. What do you notice?

**The Newline Character**

You might wonder how we create a new line in our text file. The answer is by inserting a special character called the **newline** character. This character is actually two characters together, a ‘backslash’ and the letter ‘n’. Here is an example of its use in a regular print statement:

print ("hello\nhello" )

This will print:

hello

hello

The newline character is one of many special characters called ***escape*** ***sequences***. Another useful-ish escape sequences is the tab:

\t # tab

print ("hello \t hello" )

These must be inserted in a string – they have no life outside of one. For example, this does not work:

print ("hello", \n) # error!

But this would be ok:

print ("hello", "\n") # ok

What if you want to print a backslash? For example:

print ("This backslash (\n) escape sequence creates a new line.")

To actually print the sequence \n, you use the double backslash!

print ("This backslash (\\n) escape sequence creates a new line.")

Back to our file writing, the print() function automatically inserts a newline character at the end of every print statement, so this:

print("hello")

print("hello")

print()

will output this:

hello

hello

As you should know by now, if you don’t want each print statement on a new line, you have to tell the print function to end without the newline character:

print("hello", end ="" ) # don’t add a new line

print("hello")

print()

The file write() function works in the opposite way. If you don’t say anything, it will write each new statement along the same line. You have to insert the newline character any place you want a new line.

f = open('blah.txt ', 'a') # opens blah.txt in append mode

f.write("hello") # doesn’t add a new line after “hello”

f.write("goodbye\n") # adds a new line after “goodbye”

f.close()

If you run this code 3 times it will write “hellogoodbye” on 3 separate lines.

**Key Terms: file name, file mode, file handle, output, append, open, close, escape sequence, newline character.**

**Exercise**

1. Write a program that asks the user for a first name, last name, address and phone number. Have the program save that information on separate lines in a text file named “**address.txt**”. Put the sequence in a FOR loop that repeats three times. Don’t forget to close the file.

Note: The text file should look like this :

Michael

Jordan

23 Bulls Way

234-2343

Steve

Nash

10 Lakers Blvd

345-3453

etc.