

Reading and Writing to a File

by Sophia



WHAT'S COVERED

In this lesson, we'll learn how to read and write to files. Specifically, this lesson covers:

1. Reading and Writing to Files

1. Reading and Writing to Files

In the previous lesson we started with a variable called `fout` (file out) and used the `open()` function to open and create a file using the 'w' mode. When we use the `open()` function, we have a few different modes that could be set to open the files:

EXAMPLE

```
fout = open('output.txt', 'r')
```

- By default, 'r' is used for reading a file only. In this mode nothing can be written to the file. If we do not specify a mode (second parameter), it is assumed that read mode ('r') is used.

EXAMPLE

```
fout = open('output.txt', 'w')
```

- The 'w' mode is used for writing to a file. It creates a file if it does not exist yet. If it does exist, the file is cleared out completely.

EXAMPLE

```
fout = open('output.txt', 'a')
```

- The 'a' mode is used to append to an existing file if it already exists. If the file doesn't exist, it will create the file.



KEY CONCEPT

So, what is the big difference between the write ('w') and append ('a') modes? Remember, when using write ('w') mode, if there is a current file with information included in it, the write mode will clear it out completely. The append mode will only add a string to a file; it will not delete anything if something does currently exist in a file.

So, when would you use one over the other? The append ('a') mode could typically be used in a log file since you would want to keep the existing information and add to it. The write ('w') mode can be used in a daily report with specific data that is placed in the file that reflects the current information. You wouldn't want to have yesterday's data as part of a daily report.

Writing to a File

When we open a file object in write or append mode, we can use its `.write()` method to write or append data to a text file. With either mode, we pass one string as an argument similar to the `print()` function. Unlike the `print()` function, we must convert

other data types such as the `int` and `float` types to the `str` type before we write them to the file. In addition, we can include a newline character to end each line in the text file.

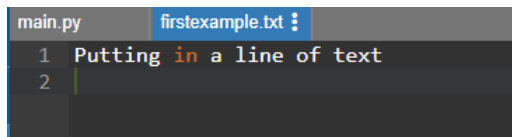
Using the 'w' Writing Mode

This first example uses write ('w') mode to create a new file and write one line to it.

↪ EXAMPLE

```
fout = open('firstexample.txt', 'w')
fout.write('Putting in a line of text\n')
fout.close()
fout = open('firstexample.txt', 'w')
fout.write('Putting in a line of text\n')
fout.close()
fout = open('firstexample.txt', 'w')
fout.write('Putting in a line of text\n')
fout.close()
```

Each time we run this example, our `firstexample.txt` file will just have the output file always looking like the following (remember, we need to move to the `firstexample.txt` file to see this output):



```
main.py  firstexample.txt :
1  Putting in a line of text
2
```



Directions: In your `main.py` file, try adding the previous code and running the program. Visit the new file created called `firstexample.txt` to see the output. Try running it a few times to notice the output is the same.

This is because each time we open the file, we are opening it in write ('w') mode. This means that each time the file is opened, it creates a new file and overwrites all of the old content.

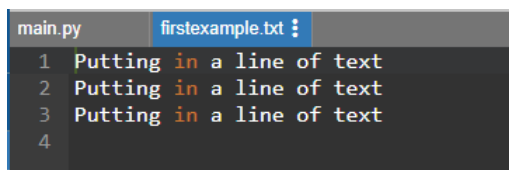
Using the 'a' Appending Mode

If we want to, we can use the append ('a') mode to append a line to the file.

↪ EXAMPLE

```
fout = open('firstexample.txt', 'a')
fout.write('Putting in a line of text\n')
fout.close()
```

Each time the file is opened and written to, a new line is added. If we run it three times, the results will look like this.



```
main.py  firstexample.txt :
1  Putting in a line of text
2  Putting in a line of text
3  Putting in a line of text
4
```



Directions: Now move back to your `main.py` file and add the append code. Run the program again. As before, visit the `firstexample.txt` file to see the output. Notice you see a new appended line each time you run the `main.py` file.

Reading From a File

After we open a file and create a file object, we can use three different methods to read data from the file. These include:

- **.read() method:** As we saw in the prior lesson, the `.read()` method reads the entire file and returns the entire contents as a string.
- **.readlines() method:** The `.readlines()` method reads the entire file and returns the results as a list.
- **.readline() method:** The `.readline()` method reads one line up to a newline character and returns a string.

Let's start with a new file as a foundation that has four lines of text.

EXAMPLE

```
fout = open('myfile.txt', 'w')
fout.write('My first line of text\n')
fout.write('My second line of text\n')
fout.write('My third line of text\n')
fout.write('My fourth line of text\n')
fout.close()
```



Directions: Move back to your `main.py` file and add the code above. Then run the program.

Once we create the new file `myfile.txt` and add the four lines of text, we can see that it did create the lines in the `myfile.txt` file.

```
main.py  firstexample.txt  myfile.txt
1 My first line of text
2 My second line of text
3 My third line of text
4 My fourth line of text
5
```

No Method Example

In the following code example, the variable `fin` (file in) will open our `myfile.txt` file in read-only mode ('r'); using a variable line, we loop through each line of file without having to use the `.readline()` method directly. The `for` loop moves through and prints each line of the file to the console.

EXAMPLE

```
fin = open('myfile.txt', 'r')

for line in fin:
    print(line)
print()

fin.close()
```

If we run this code as is, this will be the output.

My first line of text

My second line of text

```
My third line of text
```

```
My fourth line of text
```

Notice the extra lines. That is because when we read each line, the string includes the newline character and by default, the `print()` function always includes a newline character at the end of each output.

With the next code example, since each line already ends with a newline character, we set the `end` argument of the `print()` function to an empty string. Now, the `print()` function doesn't add a second newline character to each line. Here, the code doesn't call any of the read methods that we listed above. That's because the `for` loop automatically calls the `.readline()` method to get the next line in the file.



KEY CONCEPT

`for` loops inherently use the `.readline()` method when reading from files. The `for` loop will read up to and including a newline character and return the string.

EXAMPLE

```
fin = open('myfile.txt', 'r')

for line in fin:
    print(line, end="")
print()

fin.close()
```



TRY IT

Directions: In your `main.py` file, add this example code and run the program. You should see the output as follows.

```
My first line of text
My second line of text
My third line of text
My fourth line of text
```

Using the `.read()` Method

The following code calls the `.read()` method to read the entire file and returns its contents as a string. It doesn't matter how many lines are read, as they will all be read into a single string including the newline characters. Then, it prints that string to the console. Again, we will be using our `myfile.txt` in this example.

EXAMPLE

```
fin = open('myfile.txt', 'r')

contents = fin.read()
print(contents)

fin.close()
```



TRY IT

Directions: In your `main.py` file, add this example code and run the program. You should see the same output again.

```
My first line of text
My second line of text
My third line of text
My fourth line of text
```

Using the `.readlines()` Method

With the `.readlines()` method, we can use this method to read the entire file and return its contents as a list. In our example, it then prints the first two items in the list to the console.

↗ EXAMPLE

```
fin = open('myfile.txt', 'r')
#the following statement will read all the lines from the fin file handler and return the lines as a list.
lines = fin.readlines()
print(lines[0], end="")
print(lines[1])
```

```
fin.close()
```



TRY IT

Directions: Add the example code with the `.readlines()` method and run the program again. You should see the first two lines as output.

```
My first line of text
My second line of text
```

Using the `.readlines()` Method With `for` Loop

We could also use a `for` loop to output each line from the list as well.

↗ EXAMPLE

```
fin = open('myfile.txt', 'r')
#the following statement will read all the lines from the fin file handler and return the lines as a list.
lines = fin.readlines()
for line in lines:
    print(line, end="")
```

```
fin.close()
```



TRY IT

Directions: Add the modified example code with the `for` loop and run the program again. You should see all lines again as output.

```
My first line of text
My second line of text
My third line of text
My fourth line of text
```

Using the `.readline()` Method, Reading in Only What Is Needed

This example uses the `.readline()` method to read the first two lines of our `myfile.txt` file and prints them to the console. This is similar to the prior example, except that it doesn't read the entire file at once. Instead, it reads the first line and prints it to the

console. Then, it reads the second line and prints it to the console.

⇨ EXAMPLE

```
fin = open('myfile.txt', 'r')

line1 = fin.readline();
print(line1, end="")
line2 = fin.readline();
print(line2)
fin = open('myfile.txt', 'r')
```

```
My first line of text
My second line of text
```



Directions: Try this code using the `.readline()` method and run the program. You should only see the first two lines of our `myfile.txt` file as output. But this time the whole file was not read in. Only what we asked for.

KEY CONCEPT

If we have a small file, all of the read methods work equally well. However, for a large file, you'll probably want to use the `.readline()` method since it only reads one line into memory at a time. Conversely, the `.read()` and `.readlines()` methods read the entire file into memory at once, which could have a negative impact on performance if the computer doesn't have enough memory to store the entire file.

Because it's common for a program to store its data in a list, we need to understand how to write a list to a text file and how to read a list from a text file.

Write a List to a Text File

In the next two examples, we will be writing and reading from a file.

In the example below, we are writing two names in a list that will be stored in a new file called `names.txt`.

⇨ EXAMPLE

```
names = ["Sophia", "Python"]
fout = open('names.txt', 'w')
for m in names:
    fout.write(m + "\n")
fout.close()
```

A `for` loop uses the `.write()` method of the file object to write each name in the list to the file. To make that work, this code adds a newline character to the end of the string for each element.

KEY CONCEPT

Using a `\n` (newline character) adds a newline character to the position that it is inserted in the string. In our example, we added it to the end of the list element so not only does this trigger the newline, adding the next element to the subsequent line, it also adds a whitespace character to the end of the string. So, if we were counting the characters of the string, we would see the count of the visible characters plus the `\n` character. For example, if we were to count the output string of “Sophia”, the count would be 7.

So Remember, a `/n` (newline character) adds 1 character to a string.



Directions: Try adding the list to a new file called `names.txt`. Run the program, then see if the text shows up in the `names.txt` file as below.

```
main.py firstexample.txt myfile.txt names.txt
1 Sophia
2 Python
3
```

Reading Data From a Text File

To read from that text file, a `for` loop gets the string for each line in the `names.txt` file. Then, the `.replace()` method of the string replaces the newline character with an empty string. The **`.replace()` method** replaces a specific phrase in the string with another specified phrase. The `.replace()` method takes in two required parameters, the first being the phrase to look for while the second parameter is a phrase to replace the old phrase in the string. After that, this code appends the string to the `names` list. When the `for` loop finishes, a `print()` function prints the elements of the list to the console.

EXAMPLE

```
names=[]
fin = open('names.txt','r')
for line in fin:
    line = line.replace("\n", "")
    names.append(line)
print(names)
```



Directions: Try changing the code to the example above and running the program. You should get the following added (appended) to the empty list from the `names.txt` text file.

```
['Sophia', 'Python']
```

Writing Integers and Floats

Writing to a file works the same as we have been doing it: with the 'w' (write) or 'a' (append) modes of the `.open()` method. The only exception is that integers or floats must be converted to a string before the data can be written to the file.

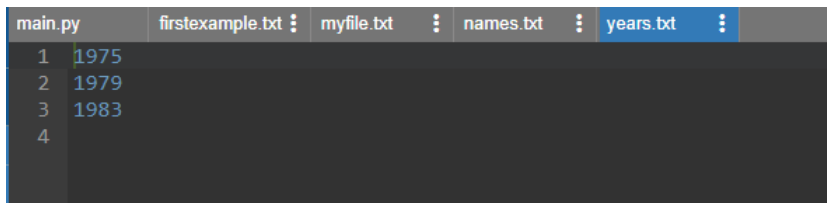
Let's try an example of integers. In this example, it will be years that need to be converted to strings. First, we have a list of integers that we define. Then, the variable `fout` (file out) uses the `open()` function to open a file and create the file `years.txt` using the 'w' mode.

EXAMPLE

```
years = [1975, 1979, 1983]
fout = open('years.txt','w')
for year in years:
    fout.write(str(year) + "\n")
fout.close()
```



Directions: Try adding the code above to create a new file called `years.txt`. Run the program, then see if the years as strings show up in the `years.txt` file as below.



```
main.py firstexample.txt myfile.txt names.txt years.txt
1 1975
2 1979
3 1983
4
```

The reading example works the same as before, except that the year string must be converted to an integer.

EXAMPLE

```
years=[]
fin = open('years.txt','r')
for line in fin:
    line = line.replace("\n", "")
    years.append(int(line))
print(years)
```



Directions: Try changing the code to the example above and running the program. You should get the following added (appended) to the empty list from the `years.txt` text file.

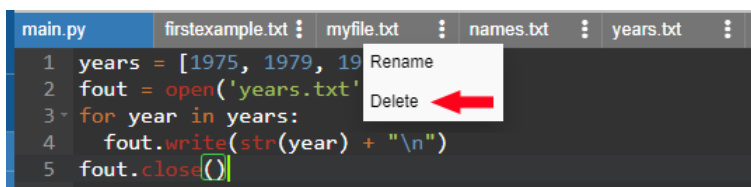
```
[1975, 1979, 1983]
```

Deleting a File

Deleting an existing file is simple in this IDE. In the event you wish to delete a local file.

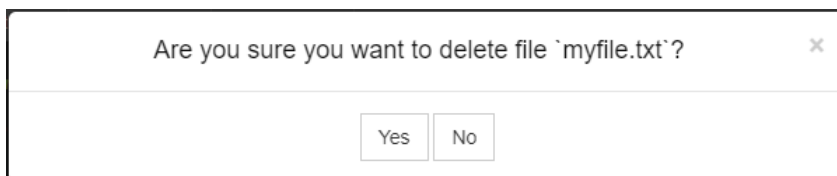


Step 1: Currently we have four text files and the main.py file. On the file that you wish to delete, in our case let's delete myfile.txt, select the three vertical dots next to the filename in the menu at the top.

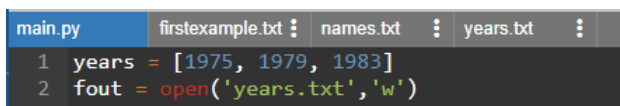


Step 2: Select the "Delete" option.

Step 3: In the confirmation window, select "Yes" (I want to delete file 'myfile.txt')



Now we can see that there are only three text files and the main.py file.





Directions: Now try and delete a file(s) from your project.

TERMS TO KNOW

.readlines()

The `.readlines()` method reads the entire file and returns the results as a list.

.readline()

The `.readline()` method reads one line up to a newline character and returns a string.

.replace()

The `.replace()` method replaces a specific phrase in the string with another specified phrase.

SUMMARY

In this lesson, we had a chance to **read and write to files**. We started out getting an understanding of the mode options of the `open()` function that allows us to read-only, write, and append after opening a file. Also, after we opened a file and created a file object, we used each of the three different methods to read data from a file. Lastly, we learned how to write back to a file and how to delete a file using the `.remove()` method.

Best of luck in your learning!

Source: THIS CONTENT AND SUPPLEMENTAL MATERIAL HAS BEEN ADAPTED FROM “PYTHON FOR EVERYBODY” BY DR. CHARLES R. SEVERANCE ACCESS FOR FREE AT www.py4e.com/html3/ LICENSE: [CREATIVE COMMONS ATTRIBUTION 3.0 UNPORTED](https://creativecommons.org/licenses/by/3.0/).

TERMS TO KNOW

.readline()

The `.readline()` method reads one line up to a newline character and returns a string.

.readlines()

The `.readlines()` method reads the entire file and returns the results as a list.

.replace()

The `.replace()` method replaces a specific phrase in the string with another specified phrase.