# Practice - Reading & Writing Text Files

Python can read and write binary files (images, video, audio, etc.,) but we will be concentrating on text files which are python’s default.

When practicing progranning file processing do also make use of web learning resources such as as [realpython’s - guide to reading and writing files](https://realpython.com/read-write-files-python/) , [w3schools file handling](https://www.w3schools.com/python/python_file_handling.asp) and of course the documentation at [python.org](https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files).

## Writing Text Files

### write()

Here is some sample code to create and write a string of text to a file. Write a Python script containing this code and save it to a convenient location in your file system. The file this code will create will be written to the same folder so make sure you know where that is.

with open('myfile1.txt', 'w') as my\_f:  
 my\_f.write('some text to my file')

Now run the script and look in the folder where you saved it. There should be a new file there called ’ myfile.txt’. Open it and read the contents. It should be ‘some text to my file’. Change the code and run it again to see the changes.

**Warning:** Opening a file for write will **DELETE** the contents of the file if it already exists.

Let’s write more text to our file:

with open('myfile1.txt', 'w') as my\_f:  
 my\_f.write('some text to my file')  
 my\_f.write('1 more text added to my file')  
 my\_f.write('2 more text added to my file')  
 my\_f.write('3 more text added to my file')  
 my\_f.write('4 more text added to my file')

Run the code and open the file created - everything is on one line!

To write the individual strings to separate lines it is necessary to add an explicit newline character to each string. The newline character is ‘’.

with open('myfile1.txt', 'w') as my\_f:  
 my\_f.write('some text to my file\n')  
 my\_f.write('1 more text added to my file\n')  
 my\_f.write('2 more text added to my file\n')  
 my\_f.write('3 more text added to my file\n')  
 my\_f.write('4 more text added to my file\n')

Now the file contains several lines of text - as required.

### writelines()

Writes a list of strings to the file.

Let’s try appending some additional lines to our file using the *’ append’* option.

with open('myfile1.txt', 'a') as my\_f:  
 list\_of\_str = ['5 moretext\n', '6 moretext\n', '7 moretext\n', '8 moretext\n']  
 my\_f.writelines(list\_of\_str)

Open the file and read it to confirm the code executed correctly.

## Reading Text Files

### read()

Reads the whole file into a single string.

Create this script in the same folder as the text file we just created:

with open('myfile1.txt', 'r') as my\_f:  
 contents = my\_f.read()  
 print('Contents is an object of type:', type(contents))  
 print(contents)

### readlines()

Reads the whole file, a line at a time, into a list. Actually it’s much more memory efficient than that as it creates a type of iterator object that gives us access to any line.

with open('myfile1.txt', 'r') as my\_f:  
 contents = my\_f.readlines()  
 print(contents[6]) # print the 5th line of the file  
 for each\_line in contents:  
 print(each\_line)

### for each\_line in file:

There is also a convenient ‘pythonic’ way of reading a file a line at a time using a for loop:

with open('myfile1.txt', 'r') as my\_f:  
 for each\_line in my\_f:  
 print('variable eachline is an object of type:', type(each\_line))  
 print(each\_line)

## Summary Table

This table summarises the most used file methods for reading and writing text files:

| Method | What it does |
| --- | --- |
| read() | reads the whole file all at once regardless of how big it is |
| readline() | reads a single line |
| readlines() | reads the whole file into a list every line of the file is an item in the list |
| write() | writes a string to the file |
| writelines() | writes a list of strings to the file |

## Reading & Writing Unicode

The files we have been creating, writing to and reading from are encoded in a way that is specific the OS we are using. This is convenient but if we intend the files to be universally iterchangeable we need to be specific about how the file is encoded. Unless there is a special reason not to, this means unicode.

Most text files intended for general purposed use are encoded using Unicode ‘utf-8’. When reading and writing utf-8 files we need to specify encoding = 'utf-8' option in the file open statement. Here is an example:

# Read the whole file  
with open('../corpora/244-0.txt', encoding='utf-8') as f:  
 contents = f.read()

Open the above file in python, print out the first few dozen lines. Do you recognise the text?