# Table of Contents

# Introduction

This document will introduce reading data from files. This process can be handy if you want Python to read and display contents of a file, or process data contained in a file. It is also possible to write data to files (which works similarly to reading data), but that will not be covered in this document.

# Video Links

The following videos cover this topic:

1. File I/O Part 1 (6:27)
    - Reads and displays a text file
2. File I/O Part 2 (6:37)
    - Reads a file containing a sequence of numbers and processes them line by line

# 1 General File I/O Process

Generally, the process by which you read data from a file consists of three steps:

1. Opening the file
2. Reading the contents of the file
3. Closing the file

# 2 Opening Files

For Python to open a file, two pieces of data are required: the filename and the file mode.

With the filename, Python will look for a file with the same name relative to the current directory. Typically, this is the directory that the Python file with your code is located in.

For the file mode, there are 3 main modes when opening a file: Read (r), write (w), and append (a).

| | |
|---|---|
| `>>> filename = "data.txt"`<br>`>>> filemode = "r"`<br>`>>> file = open(filename, filemode)`<br>`>>> filename2 = "data/items.txt"`<br>`>>> file2 = open(filename, filemode)`<br><br>`>>> type(file)`<br>`<class '_io.TextIOWrapper'>` | This will open the file `data.txt`, located in the current directory in "read" mode. The second file will be the file `items.txt` located in the `data` directory, which should be a subdirectory in the current directory.<br><br>The object returned by open is of type TextIOWrapper. This type of object contains methods that assist in extracting the file |

| | |
|---|---|
| | contents, some of which will be covered in the next section. |

# 3 Reading Data from File

You will access data from the file object by calling methods on that object. The following sections will compare some of the most common methods. For these examples, the sample text file `example.txt` will contain these 7 lines:

```
spoon
flowers
sofa
rubber duck
helmet
tomato
picture frame
```

## read()

This method will return the entire contents of the file in a single string object.

| | |
|---|---|
| `>>> file = open('example.txt', 'r')`<br>`>>> text = file.read()`<br>`>>> print(text)`<br>`spoon`<br>`flowers`<br>`sofa`<br>`rubber duck`<br>`helmet` | The entire contents of the file are bound as a single string to the variable `text`.<br><br>By printing `text`, we can see the entire contents of the file.<br><br>Using the `repr` function to provide a "printable string" of the string object, we can see specifically |

| | |
|---|---|
| `tomato`<br>`picture frame`<br>`>>> print(repr(text))`<br>`'spoon\nflowers\nsofa\nrubber`<br>`duck\nhelmet\ntomato\npicture frame\n'` | which characters are present in the string. The character sequence `\n` represents a single newline character. |

## readline()

The readline method reads a single line from the file. When reading a single line, the file object's position in the file will be advanced, so repeatedly calling readline will yield subsequent lines in the file.

| | |
|---|---|
| `>>> file = open('example.txt', 'r')`<br>`>>> text = file.readline()`<br>`>>> print(text)`<br>`spoon`<br>`>>> text2 = file.readline()`<br>`>>> print(repr(text2))`<br>`'flowers\n'`<br>`>>> text3 = file.readlines()`<br>`>>> print(len(text))`<br>`5` | The first call to `readline` returns the first line in the file, `spoon`.<br><br>The second call to `readline` returns the second line in the file, `flowers`. Notice that the call to `readline` also reads the newline character from the file, as we can see when using the `repr` method.<br><br>The third call to readline returns the third line of the file, `'sofa\n'`. Don't forget that the newline character adds to the length of the string. |

## readlines()

The readlines method is similar to the readline method, except that it reads all the lines of the file at once, placing them into a list. This method is useful for when you want to both read the entire file at once, and obtain a data type that you can loop through line by line.

| | |
|---|---|
| `>>> file = open('example.txt', 'r')`<br>`>>> text = file.readlines()`<br>`>>> print(text)`<br>`['spoon\n', 'flowers\n', 'sofa\n',`<br>`'rubber duck\n', 'helmet\n', 'tomato\n',`<br>`'picture frame\n']`<br>`>>> for line in text:`<br>`...  print(line)`<br>`...`<br>`spoon`<br><br>`flowers`<br><br>`sofa`<br><br>`rubber duck`<br><br>`helmet`<br><br>`tomato`<br><br>`picture frame` | Like a cross between read and readline, the `readlines` method returns a list of each line in the file, newline characters intact. We are able to loop through this list. Note that when printing the lines, they are more widely spaced than usual. This is because the print method by default adds a newline character, and there is another newline character in the string. |

# 4 Closing Files

Just as files are opened, they can be closed. Closing files allows other programs on your computer to access the same file. It's good practice to close files right after you are done interacting with them, typically right after the last call to one of the file's methods.

```
>>> file = open('example.txt', 'r')
>>> text = file.readlines()
>>> file.close()
>>> print(text)
['spoon\n', 'flowers\n', 'sofa\n',
'rubber duck\n', 'helmet\n', 'tomato\n',
'picture frame\n']
>>> for line in text:
...     print(len(line))
...
6
8
5
12
7
7
14
```

Notice here that we close the file on the 3rd line, right after the call to `readlines`.

After the call to `readlines`, there is a `list` object containing all the contents of the file, which the variable `text` is bound to.. As we are interacting with this `list` object and not the file itself, it's safe to close the file at this point.