Reading from & Writing to Files

This document is written to be used with the Reading Files assignment, but can be applied to other situations as well. Note that the files mentioned are specific to this assignment and you will need to use different names if you are using different files.

Reading a file

First, be sure the file you want to read is a .txt file and is in the same directory as your .py file. As we start working with files, your repos will have .txt files pre-loaded in them. As we progress, you may need to upload your own files.

To open and read the `temps.txt` file, be sure you have the correct assignment open and then enter the following into PyCharm:

```
with open('temps.txt') as file_object:    # opens the file and assigns it to file_object
    contents = file_object.read()       # reads the entire file and assigns it to contents
print(contents)
```

Note that the keyword `with` closes the file after it has been read. This helps avoid improperly closed files.

Add another print statement after `print(contents)` and run the program. Notice that when you print the contents of the file, you also print a blank line at the end. That happens because the read() method returns an empty string at the end of the file. You can remove it by "stripping" the whitespace from the right side of the text file with the `rstrip()` method.

Replace `print(contents)` with `print(contents.rstrip())` and run the program again.

Reading the file line by line

Often, you will want to read each line of the file individually. You can do that by opening it and then using a `for` loop to cycle through each line.

```
with open('temps.txt') as file_object:    # opens the file and assigns it to file_object
    for line in file_object:              # loops through each line
        print(line)
```

Note that you now have blank lines after each line from the file. You can remove them in the printout using `rstrip()` again: `print(line.rstrip())`

Creating a list of lines

Lists are far more convenient to work with that the raw file, which is only available while the file is open. To create a list of lines, you can use a method called `readlines()` that automatically reads each line and appends them to a list.

```
with open('temps.txt') as file_object:    # opens the file and assigns it to file_object
    line_list = file_object.readlines()       # reads each line and appends it to a list

print(line_list)
```

Note that each element has the `\n` newline escape character at the end. This is what the `rstrip()` method was removing before printing in the examples above. If we want to work with the data, it's better to remove those characters permanently! We can do that by cycling through the list and stripping them off each element - and then reassigning the stripped element back to itself:

```
with open('temps.txt') as file_object:
    line_list = file_object.readlines()

list_length = len(line_list)    # determines the length of the list
for i in range(list_length):
    line_list[i] = line_list[i].rstrip()    # removes the newline whitespace

print(line_list)
```

Now the list has each entry as it was in the original file. You also have all of the list tools at your disposal.

What if we want to find the average temperature in October? How could you use the tools you know to find the average of the values in the list? (Hint: you've done this before! See the "Average x" assignment!)

Be careful - pay attention to data types as you do this! Use `print(type(x))` to determine the type of an element `x` if you aren't sure!

At the end, round your answer to the hundredths place. You can use the `round(x,y)` function, where the first argument is the variable you want to round and the second argument is the number of decimal places you want.

Writing to Files

Writing to an empty file

First, decide what you will name your file. Then, open the file in write-mode by using the argument `'w'` in `open()`. The `'w'` indicates that we are opening the file in "write" mode; other modes are "read" `'r'`, "append" `'a'` and read-an-write `'r+'`. The default mode is "read-only" which is why we didn't use these arguments when we learned how to read files.

```
with open('sample.txt', 'w') as file_object:
    file_object.write('Is it lunch time yet?')
```

Be cautious about opening existing files in write mode because Python will erase the file before returning the file object.

If you type the code above into a Python file and run it, you should see a new file called `sample.txt` appear in the file list for your repo. Open it and you should see the sentence you wrote to it.

You can write multiple lines as long as the file is open:

```
with open('sample.txt', 'w') as file_object:
    file_object.write('Is it lunch time yet?')
    file_object.write("I'm hungry!")    # Note that double quotes are needed to use an apostrophe in the string!
```

Note that by running this code, you will erase the previous contents of `sample.txt`.

If you look at `sample.txt` you will see this:

```
Is it lunch time yet?I'm hungry!
```

The two lines are stuck together. If you want them to be on separate lines, you will need to add a newline "escape character" `\n` to the end of the previous line. For the sake of completeness, it's best to end each line with one.

```
with open('sample.txt', 'w') as file_object:
    file_object.write('Is it lunch time yet?\n')
    file_object.write("I'm hungry!\n")
```

You should now see the output on separate lines:

```
Is it lunch time yet?
I'm hungry!
```

Appending to a file

Opening a file in "write" mode erases the file and writes new text to it. If you want to *add* text to a file, you will need to **append** using `'a'`.

```
with open('sample.txt', 'a') as file_object:
    file_object.write('A cup of tea would be nice too.\n')
    file_object.write('A walk would be good too.\n')
```

Note that we still use the `write()` method; the only difference is the use of `'a'` instead of `'w'`.