Wednesday: Writing a File With Operator

Writing a file

Writing a file in python is pretty straight forward,

```
handle = open("text-write.txt", "w")
```

Here we change the mode to `"w"` to mean we want to write on the file. We need to be careful when we do this because when we use the `"w"` on an existing file it will overwrite all the content in that file.

```
handle = open("text-write.txt", "w")

handle.write("Hello Moringa")
handle.close()
```

When we run this on our console we see a new file `text-write.txt` has been created in our project folder and the text `"Hello Moringa"` is in it.

With operator

Python has a `with` operator that can simplify how we read and write files. The operator creates a **context manager** that automatically closes the file when you are done.

```
with open("test.txt","r") as handle:
    data = handle.read()
    print(data)
```

Here we open the file, but we do not provide a closing operation. All the functionality we want to do with the data from the file, is done in the `with` block.

Lorem File

We will now see how to manipulate data from a lorem ipsum text file.

Create a new Folder name it `Lorem`. Inside the folder create two files `readfiles.py` and `test_readfiles.py`

Create another file `test.txt` , then follow this [link (https://www.lipsum.com/)](https://www.lipsum.com/) to create a lorem ipsum text then copy it into the file

First, Let us first breakdown the different behaviors we want our application to perform.

1. We would like to read the file

2. We would like to see how many times a single word is being used.
3. We would like to see how many lines we have in our text files
4. We would like to see the longest word in the text file.

Setup Test Class

test_readfiles.py

```
import unittest
import readfiles

class TestReadFiles(unittest.TestCase):
    """
    Class to test the  functions on the  readfiles module

    Args:
        unittest.TestCase: Class from the unittest module to create unit tests
    """
if __name__ == "__main__":
    unittest.main()
```

First we setup the test class. Import `unittest` to create our test cases and `readfiles` modules that will be holding our code.

Read a file

test_readfiles.py

```
    def test_get_data(self):
        """
        Test case to confirm that we are getting data from the file
        """
        with open("test.txt","r") as handle:
            data = handle.read()
            self.assertEqual(data,readfiles.read_file("test.txt"))
```

We are testing to see if we are actually getting data from the text file. And comparing it with our `readfiles.readfile()` function that returns data from the file.

Run the code to confirm the test fails. Then write the code to pass the test.

readfiles.py

```
text_file = "test.txt"
def read_file(text_file):
    """
    Function that reads a text file and returns the data from the text file
        """
    with open(text_file,"r") as handle:
        data = handle.read()
        return data
```

Wrong File Path

So what if by mere chance we passed in a wrong file path. Let us test for this.

```
def test_nonfile(self):
        """
        Test to confirm that an exeption is raised when a wrong file is inputted
        """
        self.assertEqual(None,readfiles.read_file("tests.txt"))
```

Here we now pass in a wrong file name and we expect to get a None response from now calling the `readfiles.read_file()` function.

When we run this test we get a `FileNotFoundError` lets handle this error.

```
def read_file(text_file):
    """
    Function that reads a text file and returns the data from the text file

    Raises:
        FileNotFoundError:If it cannot the file
    """

    try:
        with open(text_file,"r") as handle:

            data = handle.read()
            return data
    except FileNotFoundError:

        return None
```

Here we nest the `with` operator inside a `try` block and we catch the `FileNotFoundError` exception we return `None`.

Lorem File

Try and implement the remaining features with your pair.Make sure you follow the Red-Green-Refactor when writing tests.

Features

1. We would like to see how many times a single word is being used.
2. We would like to see how many lines we have in our text files
3. We would like to see the longest word in the text file.