CS-UY 1114: Lab 7

Functions and File I/O

You must get checked out by your lab CA prior to leaving early. **If you leave without being checked out, you will receive 0 credits for the lab.**

Restrictions

The Python structures that you use in this lab should be restricted to those you have **learned in lecture so far (topics learned in Rephactor may only be used if they have also been taught in lecture)**. Please check with your teaching assistants in case you are unsure whether something is or is not allowed!

Create a new python file for each of the following problems.

Your files should be named *lab[num]_q[num].py* similar to homework naming conventions.

Problem 1: *Reading With Numbers*

Often times when a word is misspelled, our brain can still make sense of what we're reading. This phenomenon is unofficially recognized as Typoglycemia. For this problem, you will test the limits of Typoglycemia by prompting the user for text and changing the user's input to swap some characters with numbers L1K3 TH15 (like this).

Part A: Defining the Helper Function

First define the `numberify(word)` function. This function accepts a string message and returns an *uppercase version* of this message. Specific characters will be swapped out for numbers as so:

```
A → 4
E → 3
I → 1
S → 5
T → 7
O → 0
```

```
"""
Returns a numberified version of the passed in string word
"""
def numberify(word):
    # Code here
```

Part B: Prompting for User Input

In the `main()`, prompt the user for a message to numberify. A word is numberified *only if* the word has a length greater than 3. You may assume each word will be separated by a single whitespace. You must use `numberify()` in your solution.

The following are examples of possible outputs:

```
Please enter a message to numberify: This message serves to prove how our minds can do amazing things
Your numberified string is: 7H15 M3554G3 53RV35 TO PR0V3 HOW OUR M1ND5 CAN DO 4M4Z1NG 7H1NG5
```

```
Please enter a message to numberify: In the beginning it was hard but your mind is reading it automatically with out even thinking about it
Your numberified string is: IN THE B3G1NN1NG IT WAS H4RD BUT Y0UR M1ND IS R34D1NG IT 4U70M471C4LLY W17H OUT 3V3N 7H1NK1NG
```

```
4B0U7 IT
```

Problem 2: *Files Files Files*

Part A: *Decreasing Numbers*

Write a function named `decreasing_numbers` with two parameters, `filename` and `n`, that should write all the decreasing integers from n to 1 into a file (given by filename) in the format of exactly one integer on each line.

```
def decreasing_numbers(filename, n):
```

For example, a call to decreasing_numbers('numbers.txt', 5) would generate a text file named `numbers.txt` in the following format:

```
5
4
3
2
1
```

Here is a main function you can use to test:

```
def main():
    decreasing_numbers("numbers.txt", 5)
```

Part B: *Square Them*

In a **new file**, write a function named `squared_numbers` with two parameters, `filename and outFile`, that should read from a file (given by filename) that has exactly one integer on each line, and write to another file `outFile` the squared of those integers. This written file should be in the same format of one integer per line.

You should use the file that you created in Part A to test your code. Make sure to check

that it exists first!

```
def squared_numbers(filename, outFile):
```

For example, a call to `square_numbers('numbers.txt', 'num_squared.txt')` would generate a text file named `num_squared.txt` with the following contents:

```
25
16
9
4
1
```

Here is a main function you can use to test.

```
def main():
    squared_numbers("numbers.txt", "squaredNumbers.txt")
```

Problem 3: *Roman Code (On a File!)*

Read from the given file `roman_code_msg.txt`. Using your solution from Lab 6 Question 2: Roman Code, write the encoded message to a file named `secret_msg.txt`.

After execution, `secret_msg.txt` should look as follows:

```
Good luck on your midterms! You got this!
```

Use the following main code block to test your code:

```
def main():
    ROMAN_FILE = "roman_code_msg.txt"
    ROMAN_DECODED_FILE = "secret_msg.txt"
```

```
    decode_roman_file(ROMAN_FILE, ROMAN_DECODED_FILE)
```

Hints

- Take note of how the function `decode_roman_file()` is called in the `main()`. This should help give you an idea of what function you need to define.
- You should not have to modify your implementation for Roman Code. Calling `decode_entire_msg()` should be sufficient.
- Recall that `\n` characters exist at the end of each line in `roman_code_msg.txt`. These characters will offset the message decoding if not removed (or `strip()`-ped) first.
- Be careful to not accidentally strip whitespaces since removing them will offset the message decoding as well

Problem 4: *Alphabet Soup*

Create a file named `alphabet.txt` that will contain all letters of the alphabet as so:

```
a
b
c
...
y
z
```

File contents have been shorthanded for convenience.

`alphabet.txt` could also be a file that already exists with an incomplete alphabet. Below is *one* example of how an incomplete `alphabet.txt` could look like:

```
a
b
c
```

In this case, you want to continue the alphabet from where the file last left off.

Part A: *Reading the Last Line of a File*

To help with the case of handling a pre-existing `alphabet.txt`, one of the tasks we'd need to do is read the very last line with a character in the file. Implement the following function:

```
def get_last_char(filename):
    """
    Grabs the last alphabetic character of a pre-existing file
with content

    :param filename: name of the file to grab the last
character from
    :return: char present at the end of the character or None
otherwise
    """
```

Be sure to still consider the possibility that the function is called on a file which does not exist in your current folder.

Part B: *Creating or Appending the Alphabet*

Implement the function as shown below. Use `get_last_char()` in your implementation when handling the case of a pre-existing `alphabet.txt` file.

```
def alphabet_soup(filename):
    """
    Creates or continues an alphabet in file `alphabet.txt`.

    :param filename: name of the file to create or continue the
alphabet in
    """
```

You may use the following `main()` to test your code:

```
def main():
    ALPHABET_FILE = "alphabet.txt"

    alphabet_soup(ALPHABET_FILE)
```

Be sure to modify `alphabet.txt` as necessary to ensure your program can continue the alphabet from a pre-existing file.

Problem 5: *Evil E's*

For some reason, you have a lot of issues with the letter `E`. In fact, you despise the letter so much you wish to remove it from every single file you read. You are tasked with reading a file and removing all instances of `E` (both uppercase and lowercase).

First, implement the following helper function:

```
def remove_Es(msg):
    """
    Removes all instances of E (upper and lowercase) from a str

    :param msg: str to have Es removed from
    :return: str of msg with Es removed
    """
```

Part A: *Writing to a New File*

For the first approach to this problem, create a new file where its contents are the same as the file provided `evil_es_msg.txt`. Be sure to use `remove_Es()` in your implementation:

```
def remove_Es_new_file(filename):
    """
    Creates a new file where all instances of E (upper and
lowercase) from filename are removed

    :param filename: str of file to be read
    :return: bool True if operation was successful, False if
unsuccessful
    """
```

Your new file should have the following content:

```
Lorm ipsum dolor sit amt, conscttur adipiscing lit, sd do
iusmod
tmpor incididunt ut labor t dolor magna aliqua.
Ut nim ad minim vniam, quis nostrud xrcitation
ullamco laboris nisi ut aliquip x a commodo consquat.
Duis aut irur dolor in rprhndrit in voluptat
vlit ss cillum dolor u fugiat nulla pariatur.
xcptur sint occacat cupidatat non proidnt,
sunt in culpa qui officia dsrunt mollit anim id
st laborum
```

Part B: *Writing to the Same File*

For the second approach, implement a function similar to the one from Part A except you must overwrite the same file you are reading from:

```
def remove_Es_same_file(filename):
    """
    Updates filename file where all instances of E (upper and
lowercase) are removed

    :param filename: str of file to be read
    :return: bool True if operation was successful, False if
unsuccessful
    """
```

You may find it useful to create a copy of `evil_es_msg.txt` so you may still test your Part A with the original message.

You may test your program with the following `main()` block:

```
def main():
    EVIL_ES_MSG = "evil_es_msg.txt"
    EVIL_ES_COPY = "evil_es_copy.txt"   # A copy of the og
message for testing purposes

    remove_Es_new_file(EVIL_ES_MSG)
    remove_Es_same_file(EVIL_ES_COPY)
```

Hints:

- Overwriting a file you are reading from can initially seem tricky depending on how you typically write to files. However, you can break this process into 2 steps where you first read the entire file and *then* overwrite the file.