CS-UY 1114: Lab 9

Exceptions & File/IO

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. 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: Exceptions Galore

Write the outputs of the following code snippets by hand.

(a) Write the output for when `words.txt` doesn't exist and for when it does exist

```
def calc_val():
    val = 0

    for num in range(5):
        try:
            words_file = open("words.txt", "r")
            val += 5
            words_file.close()
        except FileNotFoundError:
            print("File cannot be found.")
            val += 10
        finally:
            val += 5
            print("Val:", val)

calc_val()
```

(b) The contents of `nums.txt` is as follows:

```
5
3
4
4
3
1
```

```
2
2
```

The contents of `nums2.txt` is as follows:

```
1
2
3
4
```

(b) Assume both files exist for the following:

```
def calc_val2():
    # Opening the files
    try:
        nums_file = open("nums.txt", "r")
    except FileNotFoundError:
        print("nums.txt cannot be found.")
        return False

    try:
        nums2_file = open("nums2.txt", "r")
    except FileNotFoundError:
        print("nums2.txt cannot be found.")
        return False

    for line in nums2_file:
        line = line.strip()
        nums_file.readline()
        nums_line = nums_file.readline().strip()

        print(int(line) * int(nums_line))

    nums_file.close()
    nums2_file.close()

    return True

was_operation_successful = calc_val2()
print(was_operation_successful)
```

Problem 2: *Files Files Files*

Part A: *Consecutive Numbers*

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

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

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

```
1
2
3
4
5
```

Here is a main function you can use to test:

```
def main():
    consecutive_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:

```
1
4
9
16
25
```

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 8 Question 3: 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 suffice.
- 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 Es

For some reason, you have a lot of issue 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.