Severance Chapter 11 Coding Assignment

General Instructions

My expectations for your work on coding assignment exercises will grow as we progress through the course. In addition to applying any new programming techniques that have been covered in the current chapter, I will be expecting you to follow all of the good programming practices that we have adopted in the preceding weeks. Here is a quick summary of good practices that we have covered so far:

- Include a Python Docstring that describes the intent of the program.
- Place your highest-level code in a function named *main*.
- Include a final line of code in the program that executes the *main* function.
- Follow all PEP-8 Python coding style guidelines enforced by the PyCharm Editor. For example, place two blank lines between the code making up a function and the code surrounding that function.
- Choose names for your variables that are properly descriptive.
- Define CONSTANT_VALUES and use them in place of *magic numbers*.
- Always use f-strings for string interpolation and number formatting.
- When processing items from Python lists and tuples, unpack the values into variables with meaningful variable names to avoid using indexed expressions in your code.
- Close all files before the conclusion of the program.
- Remember that your program should behave reasonably when it is not given any input. This might be the result of the user pressing enter at a console prompt. Or, it might be the result of the user providing a an input file that is empty.
- Model your solution after the code that I demonstrate in the tutorial videos.
- Make sure that your test input/output matches the sample provided.
- Create a sub-directory named *data* within your PyCharm project to hold data files.
- Remember to submit all data files with your PyCharm project – including the files that were provided as starter files to this assignment.
- All functions that are not *main()* should have descriptive, action-oriented names.
- All functions should be of reasonable size.
- All functions should have high *cohesion*, and low *coupling*.
- Remember to test your program thoroughly before submitting your work.
- Your code must pass all relevant test cases. Make sure that it passes tests at the boundaries created by *if, else*, and *elif* conditions in your program (boundary value tests).
- Use of the *break* statement is allowed but not encouraged.
- Use of the *continue* statement is forbidden.
- Regular expression patterns should be expressed as Python *raw strings.*

Exercise 1 (Required)
Create a program named *find_whales.py*. It should be modeled after the program that I demonstrated in the tutorial (*my_uncle.py*). Your program should be different in the following respects:

1. Your program should use re.search() to find and count the occurrences of the word *whale* (the target word) in a variety of contexts.

2. Your program should be tested with the following files that are provided as starter files:

    a. `whale.txt` (The text of the novel *Moby Dick*)
    b. empty_file.txt

3. Your program should count occurrences of the target word in the following contexts. Please note that this is one more context than was included in the tutorial program:

    a. Anywhere on the line
    b. At the beginning of the line
    c. At the end of the line
    d. At the beginning or at the end of the line
    e. At the beginning and at the end of the line

When opening the input file, make sure to specify UTF-8 encoding for the text. Otherwise, some characters in the text will not be readable by your program and cause a program interruption when testing. Your open statement should resemble the one below:

- `infile = open(infile_path_and_name, 'r', encoding='utf-8')`

Please make sure that your regular expression patterns match with the target word regardless of case (upper, lower, mixed).

Please make sure that your regular expression patterns are implemented with Python *raw strings*.

When running a test with empty file input, you should expect the following input/output on your console:

```
Please enter the input filename: empty_file.txt

0 lines in file contain target string.
0 lines in file contain target string at the beginning.
0 lines in file contain target string at the end.
0 lines in file contain target string at the beginning or at the end.
0 lines in file contain target string at the beginning and at the end.
```

When running a test with typical file input, you should expect the following input/output on your console:

```
Please enter the input filename: whale.txt

1621 lines in file contain target string.
155 lines in file contain target string at the beginning.
42 lines in file contain target string at the end.
195 lines in file contain target string at the beginning or at the end.
2 lines in file contain target string at the beginning and at the end.
```

Exercise 2 (Required)
Create a program named *find_zipcodes.py*. It should be modeled after the program that I demonstrated in the tutorial (*find_telephones.py*). Your program should be different in the following respects:

1. Your program should use re.findall() to find and extract the occurrences of two different zip code formats:

    a. 99999 (traditional 5-digit zip code)
    b. 99999-9999 (more modern zip+4 format)

2. Your program should be tested with the following files that are provided as starter files:

    a. `empty_file.txt`
    b. `zipcode.txt`

The program should find and extract all occurrences of zip codes that match either format. The extracted zip codes should be printed in a list as shown below in the sample output.

Please be aware that the order in which your regex pattern checks for these zip code patterns is significant. Your regex pattern should check for the longer pattern first and the shorter pattern second. If your regex pattern checks for these patterns in the opposite order, you will get wrong results.

When running a test with empty file input, you should expect the following input/output on your console:

```
Please enter the input filename: empty_file.txt

No zip codes were found in the input file.
```

When running a test with typical file input, you should expect the following input/output on your console:

```
Please enter the input filename: zipcode.txt

The following zip codes were found in the input file:
    08033
    60025-2252
    55555
    44444-1212
    77777
    66666-1234
```

Exercise 3 (Required)
Create a library module named *my_better_password_module.py*. Start by copying the library module that was created in the tutorial (*my_password_module.py*).

Your module will be different from the module created in the tutorial in following respects:

1. The *validate_password()* function in your version of this module should check the candidate password for the following errors. Please note that this list includes checking for 4 additional errors that were not checked in the tutorial version:

    a. Password must be at least 6 characters long.
    b. Password must include at least one upper-case letter (A-Z).
    c. Password must include at least one lower-case letter (a-z).
    d. Password must include at least one digit (0-9).
    e. Password must include at least one special character (!@#$%^&*).
    f. Password may not contain the word "opensesame" in any case.
    g. Password may not contain the word "password" in any case.
    h. Password may not contain the word "secret" in any case.
    i. Password may not contain a space.

Please make sure that your regular expression patterns match with target word regardless of case (upper, lower, mixed).

Please make sure that your regular expression patterns are implemented with Python *raw strings*.

As part of your work on this exercise, you need to extend the unit test code that has been placed in the *main()* function of the module. Model your test code after the tests that were demonstrated in the tutorial. This includes the use of the unit testing functions available in *is430_unit_test_helpers.py.*

Be sure to include additional unit test cases for the 4 rules rule that you are adding in this exercise. Also, check that these new rules have not broken any of the unit test cases that were passing when you finished the tutorial.

When you run the unit tests for this module, you should expect the following output on your console:

```
Unit testing output...

Test case 1: password meets all criteria
Passed

Test case 2: password too short
Passed

Test case 3: password missing upper case letter
Passed

Test case 4: password missing lower-case letter
Passed

Test case 5: password missing special character
Passed

Test case 6: password contains word opensesame
Passed

Test case 7: password missing multiple of the criteria
Passed

Test case 8: password missing digit
Passed

Test case 9: password contains word password
Passed

Test case 10: password contains word secret
Passed

Test case 11: password contains a space
Passed
```

Exercise 4 (Required)
Create a program named *usable_select_new_password.py*. It should be modeled after the program that I demonstrated in the tutorial (*select_new_password.py*). Your program should be different in the following respects:

1. Your program should be more usable than the tutorial version in that it will report errors and re-prompt the user to enter the new password.

2. Your program should be more usable than the tutorial version in that it will allow the user to opt out of providing a new password by just pressing the *Enter* key.

When running a test in which you provide empty input, you should expect the following input/output on your console:

```
Please enter a candidate password (<Enter> to cancel):
Password change has been canceled.
```

When running a test in which you provide typical input, you should expect the following input/output on your console:

```
Please enter a candidate password (<Enter> to cancel): _ _

This was not an acceptable choice.
Please correct the following problems:
    Password must be at least 6 characters long.
    Password must include at least one upper-case letter (A-Z).
    Password must include at least one lower-case letter (a-z).
    Password must include at least one digit (0-9).
    Password must include at least one special character (!@#$%^&*).
    Password may not contain a space.

Please enter a candidate password (<Enter> to cancel): SeCrEt PaSSwOrD

This was not an acceptable choice.
Please correct the following problems:
    Password must include at least one digit (0-9).
    Password must include at least one special character (!@#$%^&*).
    Password may not contain the word "password" in any case.
    Password may not contain the word "secret" in any case.
    Password may not contain a space.

Please enter a candidate password (<Enter> to cancel): iSchool45&
Your new password has been accepted.
```

Exercise 5 (Optional Challenge Exercise)
Create a library module named *my_best_password_module.py*. Start by copying the library module that was created in Exercise 3 (*my_better_password_module.py*).

Your module will be different from the module created in Exercise 3 in following respects:

1. The *validate_password()* function in your version of this module should check the candidate password for an additional error condition:

    a. Password must not be on the list of common passwords.

2. The list of common passwords should be based upon the contents of the following starter file:

    a. `top_100_most_common_passwords.txt`

    This file contains the top 100 most common passwords as presented in the following Wikipedia article:
    https://en.wikipedia.org/wiki/Wikipedia:10,000_most_common_passwords

    As you might imagine, this list includes offensive words. If you might be triggered by these words, and you still wish to do this exercise, please contact me for a list that does not contain offensive words.

    Please note that this list is presented in all lower case. So, before searching this list, the password text must be shifted to lower case as well.

    While this type of feature might best be implemented based on an updatable file of values, feel free to implement this list of disallowed passwords in your program as a hard-coded data structure (list, set, or dictionary) with literal values. My solution to this exercise uses such a hard-coded structure.

Be sure to include an additional unit test case for the rule that you are adding in this exercise. Also, check that this new feature has not broken any of the unit test cases that were passing when you finished Exercise 3.

When you run the unit tests for this module, you should expect the following output on your console:

```
Unit testing output...

Test case 1: password meets all criteria
Passed

Test case 2: password too short
Passed

Test case 3: password missing upper case letter
Passed

Test case 4: password missing lower-case letter
Passed

Test case 5: password missing special character
Passed

Test case 6: password contains word opensesame
Passed

Test case 7: password missing multiple of the criteria
Passed

Test case 8: password missing digit
Passed

Test case 9: password contains word password
Passed

Test case 10: password contains word secret
Passed

Test case 11: password contains a space
Passed

Test case 12: password is on the list of common passwords
Passed
```

Tools
Use PyCharm to create and test all Python programs.

Submission Method
Follow the process that I demonstrated in the tutorial video on submitting your work. This involves:

- Locating the properly named directory associated with your project in the file system.
- Compressing that directory into a single .ZIP file using a utility program.
- Submitting the properly named zip file to the submission activity for this assignment.

File and Directory Naming
Please name your Python program files as instructed in each exercise. Please use the following naming scheme for naming your PyCharm project:

```
surname_givenname_exercises_severance_chapter_11
```

If this were my own project, I would name my PyCharm project as follows:

```
trainor_kevin_exercises_severance_chapter_11
```

Use a zip utility to create one zip file that contain the PyCharm project directory. The zip file should be named according to the following scheme:

```
surname_givenname_exercises_severance_chapter_11.zip
```

If this were my own project, I would name the zip file as follows:

```
trainor_kevin_exercises_severance_chapter_11.zip
```

Due By
Please submit this assignment by the date and time shown in the Weekly Schedule.

Last Revised
2022-03-13