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.
- Always use the *with* statement as a context manager for files to assure that 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 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 (Regular)
Create a program named *find_whales.py*. It should be modeled after the program that I demonstrated in the tutorial (*find_uncles.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:

- `With open(infile_name, 'r', encoding='utf-8') as infile:`

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:

```
Count lines that contain the string "whale" in the input file.

Please enter the input filename: data/empty_file.txt

Processed 0 lines from file data/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:

```
Count lines that contain the string "whale" in the input file.

Please enter the input filename: data/whale.txt

Processed 22,333 lines from file data/whale.txt
1,621 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 (Regular)
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:

```
Extract matches to 2 zipcode patterns from input file.

Please enter the input filename: data/empty_file.txt

Processed 0 lines from data/empty_file.txt

No zip codes were found.
```

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

```
Extract matches to 2 zipcode patterns from input file.

Please enter the input filename: data/zipcode.txt

Processed 8 lines from data/zipcode.txt

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

Exercise 3 (Regular)
Create a library module named *user_authentication_common.py*. Start by copying the library module that was created in the tutorial. It has been provided in the starter files for this assignment (*user_authentication_common.py*).

Continue by copying the corresponding test module. It has also been provided in the starter files for this assignment (*test_user_authentication_common.py*).

Generally, your job in this exercise is to complete the code in *user_authentication_common.py* to implement the further requirements listed below. As part of that task, you will also need to add test cases to *test_user_authentication_common.py* to test the code that you have added for each of the new requirements.

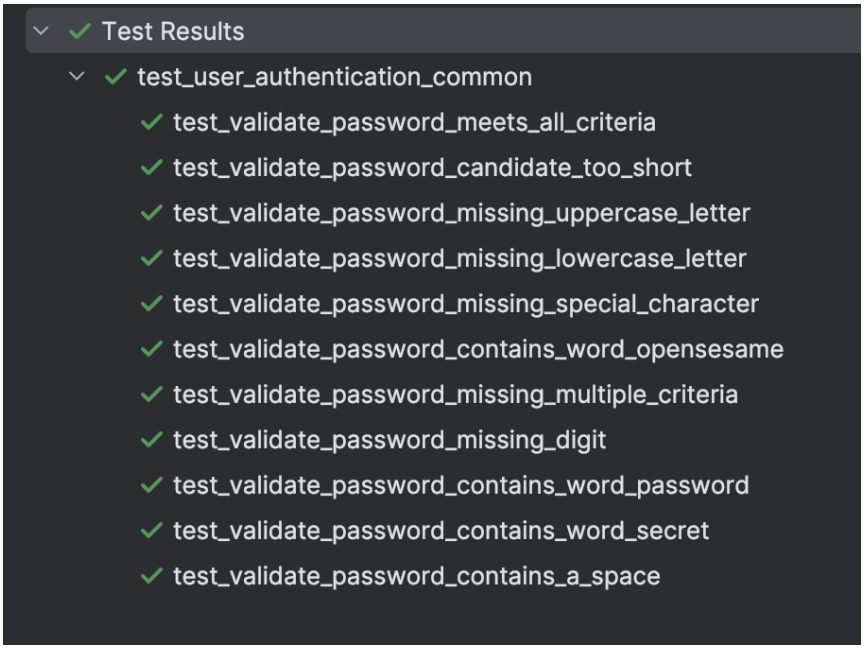
PLEASE NOTE: Adding new code to *user_authentication_common.py* can be expected to break existing tests in *test_user_authentication_common.py.* The best way to proceed is to add the code for new requirements going one requirement at a time. Then, repair any test code that may have broken. Then, add new test code to explicitly test the code you added to implement the new requirement. When, you get all your code to calm down, you are ready to proceed to implementing the next new requirement. If you don't follow this step-by-step approach (adding new code, then fixing test code, then adding new test code), the process will take substantially longer.

The following is a list of new validation requirements that must be added to the *validate_password()* function in *user_authentication_common.py*:

a. Password must include at least one digit (0-9).
b. Password may not contain the word "password" in any case.
c. Password may not contain the word "secret" in any case.
d. 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*.

When you run all test cases in *test_user_authentication_common.py* and get passing results, the PyCharm test results output panel should resemble the following:



Exercise 4 (Challenge)
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 will import the new version of *user_authentication_common.py* that you created in the earlier exercise, so it should implement the 4 new validation critera.

2. 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.

3. 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.

4. Note that your program should continue to re-prompt the user until the user has either:

    - entered a valid password, or
    - signaled that they wish to quit by pressing just the *Enter* key.

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

```
      You have requested to change your password.

      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:

```
You have requested to change your password.

Please enter a candidate password (<Enter> to cancel): aaaa
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 digit (0-9).
      Password must include at least one special character (!@#$%^&*).

Please enter a candidate password (<Enter> to cancel):
a123PASSWORDsecret OpenSesame
Please correct the following problems:
      Password must include at least one special character (!@#$%^&*).
      Password may not contain the word "opensesame" in any case.
      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): &Ab12345
Your new password has been accepted.
```

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
2025-10-08