**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 single-line comment with name of program file.
- Include a single-line comment 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.
- Output printed by the program (both prompts and results) should be polite and descriptive.
- Choose names for your variables that are properly descriptive.
- Choose names for your functions that are properly descriptive.
- Close all files before the conclusion of the program.
- Model your solution after the code that I demonstrate in the tutorial videos.
- 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).
- String used in regular expression patterns should be coded as Python *raw strings*.

**Exercise 1**

Create a program named *find_whales.* This program reads a file containing the text of the novel *Moby Dick*. This file is included with the starter files for this assignment and it is named:

- whale.txt

When opening the input file, make sure to include the parameter that specifies 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. It is a good practice to include this encoding parameters on all of the files that you open. UTF-8 encoding is supported by most text editors and it is very popular encoding scheme for many documents created in the U.S. Your open statement should resemble the one below:

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

The program will use the re.search() function to count occurrences of the following target character string in several different contexts:

- whale

The program should count and report the number of times that the target character string occurs in the following contexts:

- Occurs anywhere within the text line
- Occurs at the beginning of the text line
- Occurs at the end of the text line
- Occurs either at the beginning or at the end of the text line
- Occurs both at the beginning and at the end of the text line

Please be sure to shift the text to lowercase before searching in order to detect the target string regardless of how it may be capitalized.

Please be sure to code all strings used as regular expression patterns using Python *raw strings*.

When coding and testing this program, follow the approach that I take in the Part 1 tutorial video.

When this program is run, the console session should look like this:

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

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**

Create a program named *find_zipcodes.* This program reads a file containing zip codes embedded in surrounding text. This file is included with the starter files for this assignment and it is named:

- zipcode.txt

Your program will analyze the input file one line at a time using the re.findall() function to extract and print occurrences of zip code strings that match the following patterns

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

The program should find and extract all occurrences of zip codes that match either format. The extracted zip codes should be printed in a simple list – one zip code per line.

Please be sure to code all strings used as regular expression patterns using Python *raw strings*.

When coding and testing this program, follow the approach that I take in the Part 2 tutorial video.

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 this program is run, the console session should look like this:

```
Extract matches to 2 different zip code patterns from input file.

Please enter the input filename: zipcode.txt
08033
60025-2252
55555
44444-1212
77777
66666-1234
```

**Exercise 3 (part 1 of 2)**
Please note that Exercise 3 requires that you create 2 programs:

- my_password_module
- use_my_password_module

*my_password_module* is a container for password validation functions that will be used by other programs. The module should contain the following validation function:

- validate_password()

The code in *validate_password()* should include all code demonstrated in the Part 3 tutorial video plus additional code that implements the following requirements:

1. **Password may not contain the word "password" in any case.**
2. **Password may not contain the word "secret" in any case.**

Please be sure to code all strings used as regular expression patterns using Python *raw strings*.

*my_password_module* should also include a *main()* function that will contain unit testing code. Your code should include all unit test code demonstrated in the Part 3 tutorial video plus enough additional unit test cases to fully test the additional requirements introduced above.

Since your version of *my_password_module* is an enhancement of the code demonstrated in the Part 3 tutorial video, you should use the unit testing code from that video as your starting point. When you add further unit testing code, use my approach from the tutorial video as your model.

When this module is run by itself (not when it is imported), the console session should look like this:

```
Unit testing output...

Test case 1: password too short
passed

Test case 2: password meets all criteria
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 missing multiple of the criteria
passed

Test case 7: password contains word password
passed

Test case 8: password contains word secret
passed
```

**Exercise 3 (part 2 of 2)**
Please note that Exercise 3 requires that you create 2 programs:

- my_password_module
- use_my_password_module

*use_my_password_module* is an interactive test tool that demonstrates module importing. Your version of this program should be exactly the same as the version that I demonstrated in the Part 3 tutorial video. The difference in your experience will be that your test will be using the enhanced version *my_password_module*.

When coding and testing this program, you should use the code that I demonstrated in the Part 3 tutorial video. When testing, please be sure that the unit test code from the imported module does not run in this context.

When this program is run, console sessions should look like this:

```
Please enter a candidate password: Aa$456
This password is valid.


Please enter a candidate password: 12345
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 special character (!@#$%^&*).


Please enter a candidate password: @SecretPassword
Password may not contain the word "password" in any case.
Password may not contain the word "secret" in any case.
```

**Tools**
Use PyCharm to create and test both 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 project:

YourLastName_YourFirstName_exercises_severance_chapter_11

When you have compressed your project directory into a .ZIP file, it should have the following name structure:

YourLastName_YourFirstName_exercises_severance_chapter_11.zip

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