Zelle 3e Chapter 7 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 lecture 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).

No Tutorial Videos for This Chapter

There are no tutorial videos for this coding assignment. Instead, each of the exercises is based upon one of the programs from the supplemental demonstration project. Programs from the supplemental demonstration project are discussed in the *Beyond the Textbook* lecture video for this chapter.

Exercise 1 (Regular)
A foot race has been held for a large group of children. In keeping with modern thinking regarding children's competitions, every participant will receive a ribbon. The following table indicates which ribbon the participant should receive based up the place number in which they finished.

Place	Ribbon
1	Blue
2	Red
3	Orange
4	Gold
5	Green
6	Purple
>6	White

Create a program named *distribute_race_ribbons.py*. This program will be used by race organizers when distributing the ribbons. Each time the program is run, it will prompt the user for an integer representing the place in which the child finished. Then, it will print the color of the ribbon that the child has earned.

Design your program such that the code that looks up the ribbon earned is in a separate function named *determine_ribbon*. For this program, you can trust the user to enter an integer. Nevertheless, you need to check for inputs that are proper integers but do not represent proper finishing places. In this case, the function should return an error message instead of the description of the race award.

When creating this program, you may use the sample program *decisions_25_lookup_in_function* as a model.

The following is an example of expected input/output on the console from a successful user interaction:

```
Please enter place finished (1, 2, 3...): 5

Ribbon Awarded: Green
```

The following is an example of expected input/output on the console from a user interaction the results in an error:

```
Please enter place finished (1, 2, 3...): -9

Ribbon Awarded: ERROR - Place must be greater than zero.
```

Exercise 2 (Regular)
A state motor vehicle agency needs to calculate annual registration fees for vehicles registered in the state. Fees are based upon the vehicle type (car or truck) and the vehicle weight.

Create a python program named *calculate_registration_fees.py*.

Within that program, create a Python function that will be used in the vehicle registration system. The function should be named *determine_annual_registration_fee*. It should accept two input parameters: vehicle-type and weight. It should return the annual fee as a float value. The annual fee will be based upon the following table:

Vehicle Type	Weight	Annual Fee
Car	< 3000	125.00
Car	>= 3000	200.00
Truck	< 4000	250.00
Truck	>= 4000	350.00

In normal circumstances, any calls made to the determine_annual_registration_fee function should be for a car or a truck. Nevertheless, the code should check the vehicle-type for unexpected values. If an unexpected value is detected, the code should raise a *ValueError* with a descriptive message.

PLEASE NOTE: This exercise introduces a new approach to testing. The *main()* function will be dedicated to test code. While this is not how professional programmers typically test their code, it is a useful next step for students who are learning to test functions like *determine_annual_registration_fee().*

Each line of code in *main()* will implement a different test case. For each test case that passes, the program should print the Boolean value *True*. For each test case that fails, the program should print the Boolean value *False*. When choosing your test cases, remember to consider the need for boundary value testing.

When creating this program, you may use the sample program *decisions_35_nested_in_function* as a model.

The following is an example of expected output on the console from a test with normal test cases (no unexpected vehicle types):

```
True
True
True
True
```

```
True
True
True
True
```

The following is an example of expected output on the console from a test case with an unexpected vehicle-type:

```
Traceback (most recent call last):

<- Several lines omitted because of space considerations ->

ValueError: "motorcycle" is not a recognized vehicle-type.
```

Exercise 3 (Regular)
Create a program named *detect_input_error.py.* This program will be based upon the demonstration program *decisions_40_try.py.* Feel free to copy the sample program to provide a starting point for your code.

Your program should be different from the sample program in the following respects:

1. Instead of only prompting the user for 1 integer, your program should use a for/in loop to prompt the user for an integer 5 times.

2. Your program should print polite messages to the user at the start of the program so that the user knows how many integers they will be prompted for and what kind of output to expect.

Make sure that your program catches any bad input, prints the appropriate error message, and makes an immediate graceful exit.

The following is an example of expected input/output on the console from a test in which proper integer values are entered by the user:

```
This program prompts you for 5 integers.
Valid integer inputs are echoed back to the user.
Invalid inputs cause an error message and a graceful exit.

Please enter an integer: 11
You have entered the integer 11.

Please enter an integer: 22
You have entered the integer 22.

Please enter an integer: 33
You have entered the integer 33.

Please enter an integer: 44
You have entered the integer 44.

Please enter an integer: 55
You have entered the integer 55.

Thanks for playing.
```

The following is an example of expected input/output on the console from a test in which an improper value is entered by the user:

```
This program prompts you for 5 integers.
Valid integer inputs are echoed back to the user.
Invalid inputs cause an error message and a graceful exit.

Please enter an integer: 111
You have entered the integer 111.

Please enter an integer: 222
You have entered the integer 222.

Please enter an integer: hi mom

An integer was expected. You entered "hi mom".
The program is ending. Please run it again with proper input.
```

Exercise 4 (Regular)
Create a program named *find_highest_from_file.py*. This program will be based upon the sample program *decisions_50_find_lowest*.py. Feel free to copy the sample program to provide a starting point for your code.

Your program will be different from the sample program in the following respects:

1. Your program will be finding the highest value.

PLEASE NOTE: In past semesters, there has been some confusing regarding the comparison value to use when searching for the highest value in the file. The comparison value should be a very low integer. To do this, import the *maxsize* constant from the *sys* package. Then, turn that value negative by placing a minus sign immediately before *maxsize*. Your code that creates this very low value might look like this:

```
very_low_int_value = -maxsize
```

The following is an example of expected input/output on the console from a test using the *integer_values.txt* input file:

```
Please enter the name of the file containing integer values: integer_values.txt

The input file contained 1000 entries.
The highest value was 99975.
```

The following is an example of expected input/output on the console from a test using the *empty_file.txt* input file:

```
Please enter the name of the file containing integer values: empty_file.txt

The input file was empty. No values could be analyzed.
```

Exercise 5 (Challenge)
Create a program named *check_new_panamax_limits.py*. This program will be based upon the sample program *decisions_80_validation_using_function_and_messages.py*. Feel free to copy the sample program to provide a starting point for your code.

Your program will be different from the sample program in the following respects:

1. Your program will evaluate candidate vessels that wish to transit the Panama Canal using the General characteristics New Panamax limits as shown in the Wikipedia article at https://en.wikipedia.org/wiki/Panamax

2. Your program should allow for floating point values to be entered for each of the characteristics.

The following is an example of expected input/output on the console from a test using all valid input values:

```
This program checks candidate vessels for compliance with the new Panamax limits.
Please enter the tonnage of the vessel in DWT: 90000
Please enter the length of the vessel in feet: 801
Please enter the beam of the vessel in feet: 85
Please enter the height of the vessel in feet: 125
Please enter the draft of the vessel in feet: 40
Please enter the capacity of the vessel in TEU: 8000

This vessel is eligible to transit the Panama Canal.
```

The following is an example of expected input/output on the console from a test using all invalid input values:

```
This program checks candidate vessels for compliance with the new Panamax limits.
Please enter the tonnage of the vessel in DWT: 120001
Please enter the length of the vessel in feet: 1202
Please enter the beam of the vessel in feet: 169
Please enter the height of the vessel in feet: 191
Please enter the draft of the vessel in feet: 51
Please enter the capacity of the vessel in TEU: 14001

This vessel is NOT ELIGIBLE to transit the Panama Canal for the following reasons:
Tonnage must not be greater than 120,000 DWT.
Length must not be greater than 1,201 feet.
Beam must not be greater than 168 feet.
Height must not be greater than 190 feet.
Draft must not be greater than 50 feet.
Capacity must not be greater than 14,000 TEU.
```

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_zelle_3e_chapter_07

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

trainor_kevin_exercises_zelle_3e_chapter_07

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_zelle_3e_chapter_07.zip

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

trainor_kevin_exercises_zelle_3e_chapter_07.zip

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

Last Revised
2024-02-10