

Zelle 4e 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 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*
- Your finished code must be refactored to meet all good program design practices covered in this course.

- Your refactored code must be retested to demonstrate that refactoring has not altered program functionality.

Exercise 1 (Regular)

Create a program named *going_to_boston.py*. It should be modeled after the program that I demonstrated in the tutorial (*beat_that.py*). Your program should be different in the following respects:

1. Your program will simulate the playing of the dice game Going to Boston. For game rules, see <https://funattic.com/dice-games/>.
2. Your program will simulate a game for 2 players with the number of rounds chosen by the user. Establish a minimum number of rounds that the user can request (0) and a maximum number of rounds that they may request (1000). Document these using Python constants. Use these constant values when validating user entries associated with desired number of rounds.
3. If the user signals that they don't want to play any rounds by just pressing enter, or by entering 0 for rounds, do not simulate game play. Instead, make sure that game play does not take place and that an appropriate message is printed. See sample test output below.
4. When simulating game play, make all 3 of the rolls of the dice as specified by the rules and choose highest values from the dice rolled. Do not shortcut this process. The automated version of this game should simulate the way that human players would play the game with actual dice. For game rules, see <https://funattic.com/dice-games/>.
5. When no rounds have been requested, do not report game play statistics. Instead, report that no rounds were requested.
6. When reporting game results, take care to print text that is grammatically correct. So, print "Player A has won 1 round." rather than "Player A has won 1 rounds.". See sample test output below.
7. When you have succeeded in demonstrating that your code passes all relevant tests, then look for and exploit any refactoring opportunities. These opportunities should include:
 - a. Eliminating duplicate code.
 - b. Eliminating dead code (code that is no longer used or has been commented out).
 - c. Renaming variables, parameters, and functions.
 - d. Improving the interfaces of functions (parameters in, return values out)
 - e. Any changes that would make your code more readable, more testable, or more maintainable.

In this exercise, you should do extensive manual unit testing as documented below. Automated unit testing for *going_to_boston.py* will be conducted in Exercise 2.

When running a test in which the user requests no rounds by pressing <Enter>, you should expect the following input/output on your console:

```
Welcome to Going to Boston.
```

```
Please enter the number of rounds to be played (<Enter> to stop):  
No rounds have been requested.
```

```
Come play again soon!
```

When running a test in which the user requests no rounds by entering 0, you should expect the following input/output on your console:

```
Welcome to Going to Boston.
```

```
Please enter the number of rounds to be played (<Enter> to stop): 0  
No rounds have been requested.
```

```
Come play again soon!
```

When running a test in which the user enters unexpected values for the number of rounds to be played, you should expect the following input/output on your console:

Welcome to Going to Boston.

Please enter the number of rounds to be played (<Enter> to stop): hi mom
An integer was expected. You entered hi mom.
Please enter the number of rounds to be played (<Enter> to stop): 2.2
An integer was expected. You entered 2.2.
Please enter the number of rounds to be played (<Enter> to stop): -1
A value between 0 and 1000 was expected. You entered -1.
Please enter the number of rounds to be played (<Enter> to stop): 1001
A value between 0 and 1000 was expected. You entered 1001.
Please enter the number of rounds to be played (<Enter> to stop): 2

Playing Round 1:
Player A's turn...
Player A rolls [2, 2, 3] and keeps 3.
Player A rolls [5, 1] and keeps 5.
Player A rolls and keeps 3.
==>Player A's turn score is 11
Player B's turn...
Player B rolls [1, 5, 6] and keeps 6.
Player B rolls [1, 4] and keeps 4.
Player B rolls and keeps 3.
==>Player B's turn score is 13
Player B wins the round!

Playing Round 2:
Player A's turn...
Player A rolls [3, 2, 3] and keeps 3.
Player A rolls [4, 3] and keeps 4.
Player A rolls and keeps 2.
==>Player A's turn score is 9
Player B's turn...
Player B rolls [6, 2, 3] and keeps 6.
Player B rolls [3, 2] and keeps 3.
Player B rolls and keeps 4.
==>Player B's turn score is 13
Player B wins the round!

Game Results:
Player A has won 0 rounds.
Player B has won 2 rounds.
Player B wins the game!

Come play again soon!

The following console input/output depicts a typical run that includes a tie for a round:

Welcome to Going to Boston.

Please enter the number of rounds to be played (<Enter> to stop): 2

Playing Round 1:

Player A's turn...

Player A rolls [4, 6, 1] and keeps 6.

Player A rolls [6, 2] and keeps 6.

Player A rolls and keeps 4.

=>Player A's turn score is 16

Player B's turn...

Player B rolls [1, 5, 1] and keeps 5.

Player B rolls [3, 5] and keeps 5.

Player B rolls and keeps 6.

=>Player B's turn score is 16

The players tie the round.

Playing Round 2:

Player A's turn...

Player A rolls [5, 2, 6] and keeps 6.

Player A rolls [6, 4] and keeps 6.

Player A rolls and keeps 4.

=>Player A's turn score is 16

Player B's turn...

Player B rolls [3, 1, 1] and keeps 3.

Player B rolls [2, 3] and keeps 3.

Player B rolls and keeps 6.

=>Player B's turn score is 12

Player A wins the round!

Game Results:

Player A has won 1 round.

Player B has won 0 rounds.

Player A wins the game!

Come play again soon!

The following console input/output depicts a typical run that includes a tie for the game:

Welcome to Going to Boston.

Please enter the number of rounds to be played (<Enter> to stop): 2

Playing Round 1:

Player A's turn...

Player A rolls [3, 6, 6] and keeps 6.

Player A rolls [3, 6] and keeps 6.

Player A rolls and keeps 6.

=>Player A's turn score is 18

Player B's turn...

Player B rolls [3, 3, 6] and keeps 6.

Player B rolls [1, 5] and keeps 5.

Player B rolls and keeps 4.

=>Player B's turn score is 15

Player A wins the round!

Playing Round 2:

Player A's turn...

Player A rolls [2, 5, 4] and keeps 5.

Player A rolls [2, 4] and keeps 4.

Player A rolls and keeps 1.

=>Player A's turn score is 10

Player B's turn...

Player B rolls [4, 5, 2] and keeps 5.

Player B rolls [5, 5] and keeps 5.

Player B rolls and keeps 4.

=>Player B's turn score is 14

Player B wins the round!

Game Results:

Player A has won 1 round.

Player B has won 1 round.

The players tie the game.

Come play again soon!

The following console input/output depicts a typical run that does not include ties for either a round or the game:

Welcome to Going to Boston.

Please enter the number of rounds to be played (<Enter> to stop): 3

Playing Round 1:

Player A's turn...

Player A rolls [2, 1, 1] and keeps 2.

Player A rolls [2, 2] and keeps 2.

Player A rolls and keeps 5.

Player B's turn...

Player B rolls [2, 2, 3] and keeps 3.

Player B rolls [4, 1] and keeps 4.

Player B rolls and keeps 6.

Player B wins the round!

Playing Round 2:

Player A's turn...

Player A rolls [2, 5, 3] and keeps 5.

Player A rolls [6, 2] and keeps 6.

Player A rolls and keeps 5.

Player B's turn...

Player B rolls [2, 2, 5] and keeps 5.

Player B rolls [6, 5] and keeps 6.

Player B rolls and keeps 1.

Player A wins the round!

Playing Round 3:

Player A's turn...

Player A rolls [5, 4, 6] and keeps 6.

Player A rolls [4, 5] and keeps 5.

Player A rolls and keeps 6.

Player B's turn...

Player B rolls [3, 1, 2] and keeps 3.

Player B rolls [3, 2] and keeps 3.

Player B rolls and keeps 2.

Player A wins the round!

Game Results:

Player A has won 2 rounds.

Player B has won 1 round.
Player A wins the game!

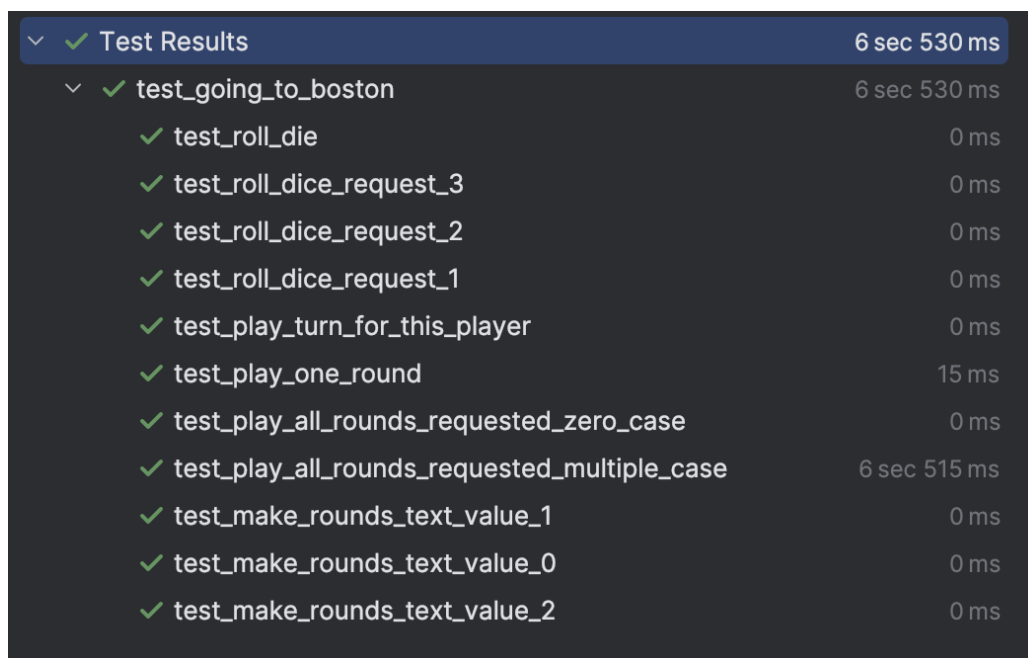
Come play again soon!

Exercise 2 (Challenge)

In this exercise, create a full set of automated unit tests for *going_to_boston.py* using Pytest. Using the approach that I demonstrated in the tutorial, start testing functions in *going_to_boston.py* at the bottom of your program and continue by testing functions progressively towards the top of your program until you reach functions that don't lend themselves to automated unit testing using our beginner Pytest techniques. Functions that don't lend themselves to beginner Pytest techniques include:

- The function that gets the desired number of rounds from the user.
- The function that prints the report.

When running your Pytest unit tests for *going_to_boston.py*, you should see output similar to the following:



The screenshot shows the Pytest Test Results window. The root node is 'Test Results' with a green checkmark and a duration of '6 sec 530 ms'. It contains a single test suite 'test_going_to_boston' with a green checkmark and a duration of '6 sec 530 ms'. This suite contains 13 individual tests, all marked with green checkmarks. The tests are: 'test_roll_die' (0 ms), 'test_roll_dice_request_3' (0 ms), 'test_roll_dice_request_2' (0 ms), 'test_roll_dice_request_1' (0 ms), 'test_play_turn_for_this_player' (0 ms), 'test_play_one_round' (15 ms), 'test_play_all_rounds_requested_zero_case' (0 ms), 'test_play_all_rounds_requested_multiple_case' (6 sec 515 ms), 'test_make_rounds_text_value_1' (0 ms), 'test_make_rounds_text_value_0' (0 ms), and 'test_make_rounds_text_value_2' (0 ms).

✓ Test Results	6 sec 530 ms
✓ test_going_to_boston	6 sec 530 ms
✓ test_roll_die	0 ms
✓ test_roll_dice_request_3	0 ms
✓ test_roll_dice_request_2	0 ms
✓ test_roll_dice_request_1	0 ms
✓ test_play_turn_for_this_player	0 ms
✓ test_play_one_round	15 ms
✓ test_play_all_rounds_requested_zero_case	0 ms
✓ test_play_all_rounds_requested_multiple_case	6 sec 515 ms
✓ test_make_rounds_text_value_1	0 ms
✓ test_make_rounds_text_value_0	0 ms
✓ test_make_rounds_text_value_2	0 ms

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_givename_exercises_zelle_4e_chapter_11

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

trainor_kevin_exercises_zelle_4e_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_givename_exercises_zelle_4e_chapter_11.zip

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

trainor_kevin_exercises_zelle_4e_chapter_4e.zip

Due By

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

Last Revised

2025-10-13