**Zelle 3e Chapter 9 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.
- Your code should be factored such that there is a function in your program for each part of the problem.
- Each function should contain code relating to the same thing – it should have *high cohesion*.
- Functions should know as little as possible about the workings of other functions – they should have *low coupling*.
- 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).
- Your finished code must be refactored to meet all of the good program design practices covered in this course.
- Your refactored code must be retested to demonstrate that refactoring has not altered program functionality.

**Assignment Overview**

The goal of this assignment is to build a simulation of a popular children's dice game named *Beat That. Beat That* is an educational game in which children learn strategic thinking and the concept of place value. See https://www.activityvillage.co.uk/beat-that

While the rules of the game can be flexible, we will be playing a basic version in which 2 players (Player A and Player B) are rolling 2 dice in a 5-round game.

Game play is based on rounds. In each round, players take a turn in which they roll 2 dice. After rolling the dice, the player decides how to use the two dice values to form a 2-digit number. The goal is to arrange the digits so that they form the highest number possible. Since you are developing your computer programming skills rather than your number literacy skills, your program will automatically make the choice for the player – choosing the highest of the two possible combinations as the player's number choice for that round. In choosing for the player in this manner you are simulating the play of expert player of *Beat That* rather than simulating the play of a novice.

At the close of the round, your program will compare players' number choices for that round to determine the round scoring. The player with the highest number choice wins the round. There are 3 possible scoring outcomes of a round:

- The Players Tie this Round
- Player A Wins this Round
- Player B Wins this Round

The program will print a message to announce the outcome of the round.

The player who wins a round scores 1 point in the overall game. The player who loses the round does not score a point in the overall game. If the players are tied for the round, then neither player scores a point in the overall game.

When 5 rounds have been played, the program will determine the outcome of the overall game based upon points earned. There are 3 possible outcomes of a game:

- The Players Tie the Game
- Player A Wins the Game
- Player B Wins the Game

The program will print a message to announce the outcome of the game.

While developing our *Beat That* simulation game, you are going to use a design and implementation strategy based upon Agile Software Development Principles. Consequently, you will not attempt to design, code, and test the entire finished *Beat That* five-round simulation as a single programming activity. Instead, you are going to conduct your work in two parts (called sprints in Agile terminology). These two sprints will correspond to the two exercises that make up this coding assignment. The following is a summary of the goal of each of these exercises (sprints):

1. Create a program that simulates a single-round of the *Beat That* game.

2. Create a program that simulates a full five-round *Beat That* game.

Choosing the single-round version of the game as the target of your first sprint allows you to generate a program that has real user value. While playing one round will not be as satisfying as playing a five-round game, playing one round does yield some player satisfaction. Also, it allows you to build and test the most challenging part of the software first. For both of these reasons, this approach is consistent with Agile Software Development values and practices.

While you are working on Exercise 1, you will NOT be creating a version of Exercise 1 that is a formal Python module that could be imported into the code that you create in Exercise 2. Instead, you should borrow as much code as possible from Exercise 1 to create the code for Exercise 2. It is probably helpful to think of the finished code in Exercise 2 as a more mature version of the code in Exercise 1.

Remember the approach that I took during the tutorial video when I developed the Coin Toss game simulation using two steps. I took care to build a function in the first step that had as many of the features that I would need in the second step as possible. Admittedly, the code for my first step had a few more features than it absolutely needed. Yet, this allowed me to take most of the code from my first program and reuse it in my second program without having to add more features to that reused code.

Please see the following section for details on the 2 exercises that make up this assignment.

**Exercise 1**
Create a program named *beat_that_single_round.* Based upon the description above, the program should implement the functionality for playing a single round of *Beat That*.

When coding and testing this program, follow the approach that I take in the first part of the tutorial video where I design and build the program that simulates one round of the Coin Toss game.

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

```
Playing round 1 of Beat That...
Player A rolls (4, 6)
Player A chooses 64
Player B rolls (4, 3)
Player B chooses 43
Player A Wins This Round!
Player A score is 1 and Player B score is 0

Playing round 1 of Beat That...
Player A rolls (2, 6)
Player A chooses 62
Player B rolls (6, 3)
Player B chooses 63
Player B Wins This Round!
Player A score is 0 and Player B score is 1

Playing round 1 of Beat That...
Player A rolls (5, 4)
Player A chooses 54
Player B rolls (5, 4)
Player B chooses 54
Players Tie This Round.
Player A score is 0 and Player B score is 0
```

After getting your program to produce the exact results expected, remember to do all refactoring needed to make this program readable, understandable, and maintainable. These refactoring steps might include:

- Renaming variables to make their names more descriptive
- Renaming functions to make their names more descriptive
- Eliminating unused variables
- Eliminated unused functions
- Eliminating unused code within functions
- Renaming formal parameters in function headers to make their names more easily distinguishable from the names of the variables in the calling code

- Eliminating parameters being passed to functions that are no longer needed in the body of that function
- Eliminating return values passed back by the function that are no longer needed in the calling code.

Remember that **refactoring is the final design step that happens after initial coding**. Your initial program design need not be perfect. By contrast, after the final refactoring step, your code should be in a state that you would be proud to pass on to a teammate and happy to receive from a teammate.

Finally, remember to test your refactored code. A refactoring is only complete after testing has demonstrated that your changes have not altered the behavior of the program code.

**Exercise 2**

Create a program named *beat_that_five_round_game*. Based upon the description above, the program should implement the functionality for playing a 5-round game of Beat That.

When coding and testing this program, follow the approach that I take in the second part of the tutorial video where I design and build a program that simulates a multi-round Coin Toss game.

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

```
Playing round 1 of Beat That...
Player A rolls (2, 1)
Player A chooses 21
Player B rolls (5, 2)
Player B chooses 52
Player B Wins This Round!

Playing round 2 of Beat That...
Player A rolls (2, 2)
Player A chooses 22
Player B rolls (1, 5)
Player B chooses 51
Player B Wins This Round!

Playing round 3 of Beat That...
Player A rolls (4, 2)
Player A chooses 42
Player B rolls (6, 1)
Player B chooses 61
Player B Wins This Round!

Playing round 4 of Beat That...
Player A rolls (6, 4)
Player A chooses 64
Player B rolls (4, 2)
Player B chooses 42
Player A Wins This Round!

Playing round 5 of Beat That...
Player A rolls (2, 5)
Player A chooses 52
Player B rolls (1, 5)
Player B chooses 51
Player A Wins This Round!
```

```
Game score:
Player A has won 2 rounds.
Player B has won 3 rounds.
Player B Wins This Game!
```

Before finishing your work on Exercise 2. **Please remember to apply the same refactoring and retesting steps described in Exercise 1 (above).**

**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_zelle_3e_chapter_09

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

YourLastName_YourFirstName_exercises_zelle_3e_chapter_09.zip

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