Programming on Purpose (CSC 120)
Winter 2024

Project 0 – Dice Games

Due: 1/11/2024

In this project, you will implement a series of functions, loosely connected by the theme of "dice games". The main purpose of this project is to help you remember your Python skills from the intro class.

Starter Code

You will write all of your code in the file `dice_games.py`. Download a starter version of this file from Nexus.

Open the file in VS Code and have a look at it.

The starter code does not provide any stubs for the functions that you have to write. But it does give you examples of how these functions will be called that you can use for testing. Notice that while you are welcome to add additional code for your own testing, you should not modify the function calls that are provided. They should work as given.

1 Simulating die rolls

Write a function `die_roll` that simulates a die roll by returning a random number that represents the die's face value after rolling it (i.e. the number shown on the top of the die). The function takes one parameter that indicates the number of sides of a die. That is, for a typical six-sided die, we would call this function with parameter value 6 and it should return a number between 1 and 6.

2 A weighted die

Write a function `weighted_die` that simulates a weighted die. Like the `die_roll` function it should take one parameter to indicate the number of sides of the die. The function should then return a randomly chosen number between 1 and the number of sides. However, the highest two numbers should be twice as likely to occur as the other numbers. For example, while the function call `die_roll(6)` should return the numbers 1 through 6 with equal probability ($\frac{1}{6}$). The function call `weighted_die(6)` should return the numbers 1 through 4 with probability $\frac{1}{8}$ and the numbers 5 and 6 with probability $\frac{2}{8}$.

3 Dice inspection

Write a function `inspect_die` to examine whether a given die roll function is biased or not. The function should take two parameters (as shown in the example function calls given in the starter code): 1) the name of a function that simulates a die roll, such as `die_roll` or `weighted_die`, and 2) a number representing how many sides the die has.

The function should then "roll the die" 10,000 times and compute the average value of those die rolls. In addition, it should record how often each value was rolled. Finally, it should print out the average and the counts for each value. For example, the function call `inspect_die(weighted_die, 6)` should produce output similar to this:

```
Average of rolled values: 4.0
1 : 1228
2 : 1234
3 : 1243
4 : 1253
5 : 2550
6 : 2492
```

The exact counts may differ slightly but the counts for 5 and 6 should be roughly twice the counts for the other numbers. Please use the same wording and formatting as shown in the example above.

Note: The function you are asked to write here, will take a function name as a parameter. This may be something that you have not encountered before. In Python, functions can be passed as parameter values like any other kind of object (ints, strings, floats, etc.). The parameter name referring to a function can then be used like a function name. Here is an example of a function that takes a die specification (i.e. a function that simulates a die roll and the number of sides of the die), rolls the die twice, and returns both values.

```
def roll_twice(die_fn, sides):
    value1 = die_fn(sides)
    value2 = die_fn(sides)
    return (value1, value2)
```

We could call this function like this `roll_twice(die_roll, 6)` or like this `roll_twice(weighted_die, 16)`.

4 Dice battle

Write a function `simple_dice_battle` that let's a human player play a **very** simple dice game against the computer. The game goes as follows: The human player rolls the die. Then the computer rolls the die. Whoever rolls the higher value wins.

The function should start by printing out the following statement: *Your turn to roll the die. Hit enter when you are ready.* Then it should wait for the user to hit enter. (Hint: Use the built-in `input` function.) Once the user has hit enter, the function should roll the die (a fair, six-sided die) and print out the result. Then, it should print *My turn.*, roll the die again, and print out the result of that second die roll. Finally, it should evaluate who has won or if it's a tie.

Here are three examples of a user's interaction with this program might look like:

- ```
  Your turn to roll the die. Hit enter when you are ready.
  3
  My turn.
  3
  It's a tie.
  ```
  ```

- ```
  Your turn to roll the die. Hit enter when you are ready.
  6
  My turn.
  3
  Boo! You win.
  ```
- ```
  Your turn to roll the die. Hit enter when you are ready.
  5
  My turn.
  6
  Yay! I win.
  ```

Please use exactly the wording and formatting shown in the examples.

5 Best of three

Write a function `best_of_three_dice_battle` that has the human player and the computer player take turns rolling the die three times. Every time the human and the computer each have rolled once, the winner is determined. After three rounds of this, whoever had more wins is declared the overall winner.

Here is an example interaction. Again, use exactly the wording and formatting shown below. The only thing that should differ are the numbers and the very last line that announces the winner. Use the same phrases for announcing the winner as above (i.e., "It's a tie.", "Boo! You win.", or "Yay! I win.").

```
Your turn to roll the die. Hit enter when you are ready.
1
My turn.
5
Your turn to roll the die. Hit enter when you are ready.
4
My turn.
2
Your turn to roll the die. Hit enter when you are ready.
2
My turn.
4
Your rolls: [1, 4, 2]
My rolls: [5, 2, 4]
Yay! I win.
```

6 Time to refactor

At this point, save a copy of your code in a file called `dice_games_v0.py`. (You will upload this file to Gradescope.)

Then continue to work in the file `dice_games.py`.

Refactor your code. That is, improve your implementation without changing the functionality. In particular, check for the following issues. But if you notice additional ways to improve your code, also implement those.

- Look at the code you wrote for `simple_dice_battle` and `best_of_three_dice_battle`. Most likely, the implementations of these two functions contain bits of code that are very similar to each other.

Could you define some additional helper functions to take care of those bits of code, so that the functions `simple_dice_battle` and `best_of_three_dice_battle` just need to contain function calls to those helper functions? Your goal is to reduce the amount of repeated code and to make `simple_dice_battle` and `best_of_three_dice_battle` more concise and easier to understand.

- Are you using informative variable and function names?
- Does the organization of your code help make the logic of your implementation clear?
- Is your code well documented and sufficiently commented?

  Start by reading the section on "Commenting vs. Documenting Code" in "Documenting Python Code: A Complete Guide" by James Mertz. (Link also available on Nexus.)

  To be well *documented* the Python file/module should have a header docstring that describes the purpose of the program and gives the author (Later you will add some additional information this docstring.) and each function definition should be accompanied by a docstring that describes what the function does and explains its parameters and return value. Read the formatting guidelines for one-line and multi-line docstrings in PEP 257 (Docstring Conventions). (Link also available on Nexus.)

  Add *comments* where necessary to clarify the logic of your implementation. Be careful to not over-comment. You expect the readers of your comments to be yourself or other Python programmers. So you do not need to comment every line and your comments should not just be English translations of the Python. E.g. the comment "Assigning 6 to the variable 'sides'." does not add additional information to the line `sides = 6`. Read the section on "Comments" in PEP 8 (Style Guide for Python Code). (Link also available on Nexus.)
- Is your formatting consistent and neat? Are you using empty lines consistently to separate function definitions from each other? Are you using blank characters (e.g. around parentheses and operators) consistently? Are your lines short enough that they fit comfortably on one line, even if the editor window is not full screen? (Traditionally, this is interpreted as meaning that no line should have more than 80 characters. With today's screens, I am willing to allow for 100.)

Reflect on the main changes you made during this refactoring step and write up your thoughts (100–300 words). Save your reflection document as a pdf. (You will upload this pdf file to Gradescope.) You don't have to document every change you made. Summarize the biggest changes and describe how they improved your code.

7 A driver function for the two dice games

Write a function `die_battle_driver` that should exhibit the following behavior:

1. Greets the human player by printing "Let's play dice!".
2. Ask the player which version of the dice game they want to play, the single trial version (`simple_dice_battle`) or the best-of-three version (`best_of_three_dice_battle`).
3. Depending on the player's answer, play the corresponding dice game.
4. Ask the player whether they want to play again. If so, go back to the previous step. If no, end the program after printing the message "OK. That was fun! Let's play again some time."

Here is an example interaction.

```
Let's play dice!

How do you want to play?
```

```
1) single trial
2) best of three
Type 1 or 2:  1

Your turn to roll the die. Hit enter when you are ready.
5
My turn.
1
Boo! You win.
would you like to play again? (Y/N)  Y

Your turn to roll the die. Hit enter when you are ready.
6
My turn.
5
Boo! You win.
would you like to play again? (Y/N)  N
OK. That was fun! Let's play again some time.
```

8 Getting Ready to Submit

1. Re-read your code and check again for ways in which it could be improved.
2. Include an acknowledgment of any resources that helped you complete this assignment. You do not need to cite me, the textbook, or the official Python documentation. Please do cite any other people or sources.
3. Include the honor code affirmation at the top of each Java file you have written or edited:

```
I affirm that I have carried out my academic endeavors
with full academic honesty. [Your Name]
```

9 How to submit

Submit the project by uploading the Python files `dice_games.py` and `dice_games_v0.py` as well as your reflection document (pdf) to Gradescope.