Project Title: Geo_Trivia

1. Introduction

- The trivia game features a dictionary and list to handle the questions and answers. It also has four functions with their respective for loops and/or conditional if/else statements. Within the loops, we are using various methods such as append, get, and upper and lower among others. Finally, a while loop asks the player if they want to play again.

2. Design and Implementation

- When I envisioned this trivia game, I wanted to welcome a player, have them register, give them trivia category options, and calculate a final score out of the categories they chose. As I began to write the code, I realized that I needed to break the project into understandable tasks and codable pieces. I kept my code to simple functions and loops. The more advanced components include a nested loop and getting the game to play again.
- I started with a list of simple geography questions that I wanted to answer. I initially used a dictionary to store the key(question), value(answer) pair. [INSERT SCREEN SHOT] I then set up the basic pieces that I wanted to code, playing the game, checking the answers, showing the score, and asking the player to play again.
- In the play the game function, I call the questions using a for loop to call the key in the dictionary and I then call the answer within the initial for loop. This was very challenging, and I reviewed the nested loop section of our course in addition to online resources to help me get this code working. In the early versions of my code, I printed the question and asked for user input to answer and then check the answer. This was fine for a question of two, but if I wanted to do 10 or 100 questions, that would be a lot of lines of code with the potential to make many mistakes, which I did and returned many errors. I then split this single function in two, a play_game and check_response function to run the code and check the answers. I then added the upper and lower methods to ensure that any question response would be accepted but then modified that again after looking

through resources to include a list of responses so that players had options from answers and to make the answer input process simpler.

- I created a score function after I had a couple of questions answered and printed out the score. Again, this was limited, and I then realized that I would need to create a function to check answers and give a point for correct responses and then I would calculate the correct attempts within the play game function as I created my variables there. I faced another challenge in coding the option for the player to try again. I attempted using the try again code we used in class, but ultimately modified it and then researched to find out I run it at the end of the program.
- I modified the way the code handles questions and checks the responses to accommodate an innumerable set of questions. This was critical to making the code more concise.
- Another modification was made to calculate the score after much research and finding the best way to find the best way to define a correct attempts variable.

3. Conclusions

- I learned a lot coding this program. What I was seeing in my head didn't quite fit in with how the code would be executed and that ultimately was ok as I learned more about how to make this code run. In the end, it turned out to be simpler than I envisioned, and it is therefore easier to grow from here. In hindsight, I would create outlines that handle the simplest tasks and work from there.
- In the future, I would create a separate file to handle the questions and answers. As this project grows and there are multiple categories, I will be most efficient to import the questions and answers from a separate file in the same package.
- A second feature to add is tabulating the score based on the categories played.
- I could also add a multiple player feature that saves scores and returns the highest score at the end of the turn.