What are some different error types and why did you choose the specific ones you chose for fizzbuzz_adv and amortization?

- I used both ValueError and TypeError for both of the respective functions. For fizzbuzz_adv I was looking for n needing to be an integer and also that n could not be a non-negative number. For the amortization function, I noticed that another student created a function inside of the the amortization to check different input values for principal, monthly payment, annual rate and filename. I started by validating that filename was a string type, but then I also wrote up a ValueError identification for the other arguments as well. I was having a couple of problems with pytests, which forced to research both Piazza and the zoom office hours to try and find answers. I also extrapolated these checks out to functions, so that I could just call them inside of amortization.

What is the value of currying a function like add_filename_extension? How did you use this function in amortization?

- I believe the value of this curried function is it is compact and handles the default values of filename extensions within it. I was not able to figure out how to use add_filename_extension(), but instead used add_txt() to change the filename appropriately within the amortization() function and pass the pytests accordingly. I will need to practice currying more because this definitely took me a very long time to figure out how to adjust add_filename_extension() to accomplish the same things as the original function along with the lambda functions. Curried functions are sequential in nature, and each function only uses one argument, so it should be easier to perform debugging on a curried function.

What other changes did you make to amortization beyond the requirements of this assignment?

- I did not change a lot to the amortization function outside of the requirements laid out within the assignment. I was doing one of the checks inside the while loop, but moved that out, which is substantially cleaner. I think because I grouped things together it is easier to follow the logic of the function now and it is not as much to go through the function. The more compact you can make your functions then the easier it is to debug something if things go wrong.

Why might we want to use a function as an argument for another function?

- By nesting functions in this way then it lays out a sequential connection, which can make tracing a flow of something easier. It is easier to reuse functions in different places when we use a function as an argument for another function. I definitely can follow things throughout my code substantially easier when I decompose actions into multiple functions rather than one big function with lots of different commands and arguments. This also allows me to breakdown the different pieces of what needs to get done into many different individuals functions so then I can comment on that function and understand exactly what it is doing.

Describe one thing that you learned while working on this lesson that stood out as useful or interesting.

- I love using pytests, so much because there is so much to the assignments at times that I take an initial stab at things and then I can work through the

pytests to see where things are going wrong. I learned how to isolate pytests and I just ran all the pytests and would come across a problem. I would then work through that problem and look for the next failing pytest and do the same thing over and over. This made getting to the finish line more realistic. I believe it would have been very daunting to figure things without them. I really like using checks for values and types and I will probably use much more of that in my coding. I think this can then filter things off better. My data scientists and engineers that have reported to me and currently do report to me definitely talk about error handling and it makes sense how elegant error handling can help users out so much.