1. Write test case, write code to make it pass, write test case, write code to pass, ... write final test case, write code to make it pass.
2. I disagree with both points. I believe that it is better to write code thinking of the algorithm that needs to be written rather than, “hmm, how can I do something so I can pass the next test case.” I feel that this method would lead to worse code because you are doing the bare minimum that needs to be done. I feel that it would be better to think about how you can pass all possible scenarios rather than how can you pass the next one? Going about it in steps isn’t bad, but when you do the least possible work to get it to pass it just seems like a waste of time.
3. The advantages seem to be working through it a little at a time so you aren’t overwhelmed by the entire problem all at once, but this fails on some things like the second time we used it in this assignment. Other than just hardcoding in the return every time you have to actually think through an algorithm that will work for most if not all of them. It wasn’t something that could really be done with TDD because it would take forever, each time just adding another test case for the next number and then adding that into the method until you finally give up and decide it’s not worth the work, I’m going to actually write an algorithm for this. Another thing is you rely on the test cases so much that if you mess one of those up you are trying to figure out what is wrong with your code and in reality it is the test itself that is wrong.

(For this lab I didn’t commit every time we updated because everything was done on Daniel Beckmann’s computer, so we committed on his repository every time, I have the final version on mine.)