**Summary**

1. Describe your unit testing approach for each of the three features.
   1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
      1. My tests were checked against the requirements that were specified in the assignments and code. For example, I validated that the appointment dates cannot be in the past, among many other examples. These tests helped to make the code perform within expectations.
   2. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage
      1. The tests covered extreme and standard cases. Utilizing things such as input validation, I helped to ensure that all things were checked. You High coverage suggests that the most potential errors can be identified and dealt with accordingly.
2. Describe your experience writing the JUnit tests.
   1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
      1. There are several things that I employed to make sure that my code was technically sound such as using `assertEquals` to verify outcomes and `assertThrows` to deal with error handling in the case of failure. In my `AppointmentTest` class, i used the line `assertThrows(IllegalArgumentException.class, () -> new Appointment(null, date, desc))` which made invalid inputs be denied.
   2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
      1. I avoided using unnecessarily complex testing to help my code be more efficient. By doing this, I created only the needed objects and used straightfoward validation. For example, I only constructed a single `Contact` object per test and then validated that. This helps to streamline tests and make them more efficient.

**Reflection**

1. Testing Techniques
   1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
      1. In some instances I used things like boundary value analysis. For example, I tested IDs legnth limits and equivalence partioning by using valid and invalid phone numbers and/or dates.
   2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
      1. I think something I should have used but did not was mutation testing. I didn't alter the code to see if tests catch the changes, which would have proven useful as they can uncover subtle problems with your code that otherwise may not come to light.
   3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
      1. Things like boundary and negative testing are widely used. It helps to ensure that your code works at the most basic level. Mutation testing is used in more advanced/complex projects that require more vigorous testing. This could be in places where data is very sensitive and needs to be well protected.

**Mindset**

1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.
   1. I remained cautious by questioning assumptions and double-checking straightfoward conditions. I had to make sure to check even the most mundane things, such as a single invalid date or long ID because it could balloon in to a bigger problem if left unchecked.
2. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.
   1. I think the easiest way that I avoided bias was by testing code as if it was written by someone else. I removed myself completely from the project, almost as if I was a consultant or superior reviewing the code. This helped me to be objective and catch errors that I otherwise perhaps wouldn’t have caught.
3. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.
   1. Being disciplined is one of if not the most important things that you can do /be as a software developer. By not cutting corners, I make sure that any issues that appear are dealt with swiftly and efficiently. This helps to not allow the errors to “fester” and possibly get released into productio. I plan to keep doing unit tests and regularly review my code in intervals as its written.