So, in the unit testing project, I'm going to add a new class. And call it DemeritPoints CalculatorTests.

First I'm going to apply the TestFixture attribute here, and now I'm going to generate all the test stops. So, method, CalculateDemeritPoints,

What is the first scenario? Well, this function gets an integer which determines the speed of the car.

What if we give it a negative number? It doesn't make sense.

So I expect this method to throw an argument out of range exception, so speed is Negative, it should ThrowArgument out of range, exception.

Another test method. CalculateDemeritPoints What if the speed is zero? The car is stopped. You should get 0 right? So SpeedIsZero we should ReturnZero. Another scenario is the speed is greater than zero, but less than the speed limit. Again, you should get zero, because as far as the driver drives under the speed limit, they won't get any demerit points. So, CalculateDemeritPoints, SpeedIsLessThanSpeed Limit, again, returns 0 Now, what if the driver is driving exactly at the speed limit? Again, you should get zero again, right? So I see a pattern, you're writing three tests and all these tests are testing that the function returns 0, I would rather we combine these scenarios into one test with multiple test cases. So, we can change the name of this method to something more generic, SpeedIsLessThan OrEqualToSpeedLimit.

Now here we need to add a parameter. that is the speed, and we define multiple test cases. So TestCase, when speed is 0, we should get 0 points, when the speed is less than the speed limit like 64, again, you should get 0, and this exactly, is the speed limit we should still get zero, then we can delete. This test method. Now, what if the driver is over this speed limit, but less than 5 kilometers an hour. For example, what if you're driving at 66 kilometers an hour. Again, they shouldn't get any points. They would get a point for every 5 kilometers of the speed limit. So I'm going to add another test case here, set the speed to 66.

And again we should get 0. But the name of this function no longer represents all these test cases. So I want to make it even more generic. So, let's change the scenario to WhenCalled ReturnDemeritPoints. Now we have a more generic name for our tests, let's add a few more test cases. So the other test case is five kilometers above the speed limit. So, if the speed is 70, then we should get 1 point, and just for peace of mind, I want to make sure that the speed is 75, then you should get 2 points.

If our function correctly calculates DemeritPoints, for these two test cases, it's likely that it's going to work if the speed is 80, 85, 90, and so on. But what if the speed is ridiculously large number, like 500 kilometers an hour, or 50,000 kilometers an hour, it doesn't make sense. In this case, I expect this function to throw an argument out of range exception. So similar to the first test, I'm going to duplicate this let's make an assumption here. You can assume that most cars don't go faster than 300 kilometers an hour, so if SpeedIsOver 300, again it should throw an argument out of range exception.

So now that we have various test cases for this function, let's go ahead and implement these tests.

First, I'm going to create calculator object, so calculator, DemeritPointsCalculator.

Now we act (?) calculator, .Calculate, and give it a negative number.

Again you want to use simple numbers, so minus 1, is better than minus 10, because otherwise it creates confusion. What is this minus 10, is it a special number in this domain? No we just care about a negative number here. Now in this case, because this method is going to throw an exception, we need to wrap it with a lambda expression, and run it as part of making our Assertion. So, Assert That we add a lambda expression, and here we call this method. Okay? Now, we Assert that this method Throws, look here we don't have argument out of range exception, so we need to use Exception property. And set it's type, TypeOf ArgumentOutOfRangeException.

Let's run this test, Command and T and Command and R

Okay the test passed, beautiful.

Now this second test case is very similar to the first one.

So instead of having two separate methods, you can parameterize this test method, and add multiple test cases. So I'm going to define a parameter called speed, and here, we're going to have two test cases. So TestCase, 1 is minus 1, and the other is 301.

Now to keep our code clean, we should change this scenario to something more generic, speed is out of range. Okay? And finally we replace minus 1 with speed. Then we don't need this method anymore. Let's run the test again. Okay, our test failed. So, here's our test method you can see, we have two test cases the first one which was for minus one succeeded, but the second one which was but (?) 301 failed.

Look at the details, so we expect that an argument out of range exception, but we got no exception thrown from this method. So this is telling me that there is something wrong in the implementation of this method. Let's go back. Look, nowhere here, we're checking for the upper limit. So I'm going to change this to if speed is less than 0, or greater than 300, we're going to throw an argument out of range exception. Now to make the code more readable, I would rather to replace this hard coded magic number with a constant. So here we can define the private constant, integer MaxSpeed, and replace 300 with MaxSpeed. Now, back to our test method, let's run this one more time, Okay, this time you can see most test cases successfully passed. Beautiful. Now, all these test cases, so once again I'm going to create calculator object, new DemeritPointsCalculator, call Calculate function for calculate method, give it the speed get the result, which is in this case points. And Assert that

points is EqualTo what? Well, we need to add another parameter here. That is the expected result. So integer expected result.

These numbers that we're supplying using our test case attribute,

these numbers represent expected result. I'm going to use the argument here, expectedResult, Now, let's run all these tests, All of them are

passing, beautiful. So we have a total of eight unit tests, for this class.