Alright, so we're ready to start our unit testing journey. From this lecture onwards, in every lecture I'm going to show you one or two techniques, you're going to start with simple examples and then these examples will gradually get more complex, are you ready? Let's get started.

So TestNinja project, open the fundamentals folder, and look at this math class. The first test you want to write is for this add method here. Very simple method, with only one line of code. How many tests do we need here?

Well, look at how many execution paths we have in this method. You have only a single execution path, you don't have any conditional statements, you don't have e and l's (?) or switch (?) case. So when testing this method, I'm going to supply to our arguments like one and two, and verify that this method returns three. If that works, then I don't need to test it again, with another set of arguments, okay? So only a single test is sufficient for the add method. Now, back to our unit testing project, I'm going to add a new class, and call this math tests.

First, I should apply the test fixture attribute here, otherwise

NUnit test runner is not going to run the in this test. Now we create a public void method, and use the convention I told you before, so the method on the test is Add. Now what scenario are we testing here?

In this case we are dealing with a single scenario. So I'm going to use a generic term. Like WhenCalled.

So the add method, when called, what should it do?

It should return the sum of argument.

Okay? Now we need to decorate this method with test attribute.

So here we have three parts. Arrange, act, and assert. In the arrange we create a new instance, of the math class, make sure to use the math class defined in TestNinja.fundamentals.

Not the one defined in the system main space that is part of .net,

framework. Okay? Now we add so math.add, we give it two arguments but what arguments should we use here? It's best to use simple values like 1 and 2, as opposed to some magic random numbers like 952. Because when someone looks at this code, they wonder, what is the meaning of 952?

Does this number have a special meaning in the domain in the application, is this a constant defined somewhere else, is it the idea of a record in the database, so we don't want to create such confusions for others.

And we use simple values, 1 and 2.

We get the result, and then we need to verify that result is equal to 3. So assert that result, is equal to 3.

Let's run this test. So, if you're using Rider or ReSharper, your shortcut is command and T, and Command and R on Mac, or Control T and Control R on Windows. And if you're using Visual Studio without ReSharper, you need to run all the tests. Unfortunately I don't remember the shortcut for that, so you need to look that up yourself. So, let's run this test. Okay, our test passed, beautiful.