Alright, before we get started, I want to show you a shortcut to generate a test method quickly. This shortcut only works if you're using ReSharper in Visual Studio or if you're using the Rider ID. So, in Rider you can go to your preferences dialogue box and search for live templates. So, here we have this concept called Life Templates, which is like code snippets, in Visual Studio, with this we can quickly generate some code.

So here I'm going to add a new live template, give it an abbreviation, I'm going to use tm, as a short for test method. And for the template text I'm going to add the basic test method. So, all our test methods have the test attribute. We have public void, the name of the method, the scenario

we're testing, and the ExpectedBehavior.

Okay, now I want to convert these parts into parameters that I can set at run time. So, we simply surround them with $ signs like this and you can see their color changes so this indicates that each of these is a parameter. Okay?

Now here we have this warning, no applicable context yet. Click Define, so this is basically where we specify where we can use this live template. We don't have C# in this list, so I use other. Now save, okay, so, back in our unit testing project. I'm going to create a new test class, FizzBuzzTests. First I'm going to decorate this with TestFixture. Now I want to use the life template that I just created. So if you go to the code menu on the top, you can see this menu item, Insert Live Template, note the shortcut, on Mac it's command and J. So, if you press Command and J here, you can see the live templates that are applicable in this context. So here we have TM. You can simply press enter. And this automatically generates this stuff for our test method. And now we can set our parameters. So this first parameter is the name of the method we're testing. In this case GetOutput, now we press tab, we get to the second parameter of the scenario, InputIsDivisible by 3And5. Tab. And here we need to specify the expected behavior. So it should Return FizzBuzz. So, we can see live

templates help you write tests faster. So this is the first test scenario, before I implement this, I would rather generate the test stubs for all different scenarios. So the other scenario is where the input is divisible by 3 only. So once again, command and J, test method, GetOutput, InputIsDivisibleBy3 Only. And it should ReturnFizz. Another scenario, once

a (?) GetOutput, Input IsDivisibleBy5Only this should ReturnBuzz, and the last scenario. GetOutput. InputIsNot DivisibleBy3Or5. Should ReturnSameNumber.

So these are all the scenarios we need to test, now let's implement them one by one. So, I'm going to start with the first one. Back to our solution explorer, if you look at the implementation of this FizzBuzz class.

You can see that this method is defined as a static,

So we don't have to create an instance of this class in our test method. In other words, we don't have an arranged part, we simply act.

So FizzBuzz. Output. You want a number that is divisible by 3 and 5.

So I'm gong to go with 15. Get the result. And then Assert that result, is equal to, FizzBuzz. Now when testing strings, I told you its better not to be specific, especially when you're dealing with

error messages. Because a simple change can easily break your tests.

But in this case, I want to check for the exact match, because that's the purpose of this GetOutput function, we want to

make sure it's returning exactly this string.

Now the second scenario. To save time, I'm going to copy these two lines, paste them here. This time I'm going to use 3 as the argument to this method.and Assert that result is Fizz.

One more time, Copy, raise it here. This time, the input should be 5, and the result should be Buzz. And finally, the last test case.

Once again, copy, paste, I'm going to use a simple number like 1 that is not divisible by 3 or 5. And Assert that result is 1.

Now let's run all the tests in this class. So Command and T and command and R.

So you can see all test cases passed successfully. Beautiful.