One of the common questions I get is:

"Mosh, I don’t know what to test!" So, in this lecture I'm going to give you a simple recipe that will guide you when writing tests.

A lot of times when you don't know what to test, that's probably because you're dealing with poorly written code. When your methods have more than 10 lines of code, of course you don't know what to test and how! Even I wouldn't have a clue! That's why I say that unit testing and clean coding go hand in hand. If your code is clean and properly structured, it's easier to test. Now, assuming that your code is clean, as a rule of thumb, you should test the outcome of a function. In programming, we have two types of functions. queries and commands.

Query functions return some value. They don't necessarily have to query a database! This just means their role is to return some value.

So, for testing a query function, you should write a test and verify that your function is returning the right value. Now, you might have multiple execution paths in that function. In that case, you should test all the execution paths and make sure that each execution path results in the right value. A command method performs an action. This often involves changing the state of an object in memory, and/or writing to a database, or calling a web service or sending a message to a message queue and so on. These are all commands, they're making a change in the system. Such functions may return a value as well. For example, the method that creates a new order and writes it to a database may return an order object that may includes a unique ID. To test this method, similar to query methods, you should test the outcome of this method. If the outcome is to change the state of an object in memory, you should test that the given object is now in the right state. If the outcome is to talk about an external resource like a database or web service, you should verify that the class on the test is making the right call to these external dependencies. We'll look at dealing with external dependencies in the next section. So, have a basic recipe about what to test and how. Now, let's take a look at what not to test:

You should never test language features. For example, if you have a simple C# class that is mainly a property bag, you should not test these properties. You should not set them and then read their value to make sure the properties are working. Because this way, you're testing C# language features! The same is true when you're dealing with third-party code. So, if you're using a third-party library, like any framework, you should not write tests around their methods. You should assume they're properly tested. and you should only test your code.