GENERAL ASSEMBLY

UNIT TESTS

Zeke Abuhoff

Lead iOS Instructor, General Assembly

UNIT TESTS

LEARNING OBJECTIVES

+ Explain a unit test’s purpose
+ Create a unit test target
+ Write a unit test
+ Write test assertions

TESTING

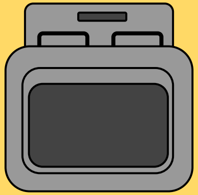
You’re in the market for a new gas stove. Which brand would you prefer?

Wonderflame

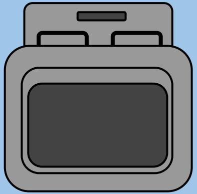
Every part tested!

Thorough safety!

Thorough quality!



Cookmaster



We made it right.

We’re pretty sure.

Buy it!

TESTING

Testing lets us know our code better, confirm our assumptions and catch mistakes before they reach users.

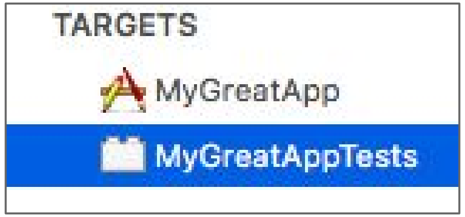
While manual testing can do the job sometimes, **automated testing** is faster and more precise. When we write tests in code that test code, we call them **unit tests**.

Unit tests should be:

Focused (a test should test one thing)	Clear (the code should be short and easy to read)
Consistent (repeating a test should yield the same result)	Quick (running tests should be temptingly easy)

XCTEST

Working in Xcode, we use a framework called XCTest when we want to make unit tests.



When you create a new testing target, Xcode will start you off with an XCTest file that features a subclass of XCTestCase.

```
class MyGreatAppTests: XCTestCase { }
```

XCTEST

In a test file, you add a test by creating a new method in your XCTestCase subclass whose name begins with the word “test.” When Xcode builds your project, it will see that this method is intended as a test and add a test-running icon beside it.

```
    func testAddingToDoItem() { }
```

When you run a test, Xcode builds your project but only runs the code referenced by the test. Note: **you are not running the entire app**. Do not expect to see login screens, user profiles and whatever else make up the full experience of the software you’re developing. These tests are for isolating and evaluating pieces of code.

UNIT TESTS

XCTASSERT

To make a test pass or fail, we call **XCTAssert()**.

This method takes a boolean expression.

```
    XCTAssert(todos != nil)
```

If the value of the boolean expression is true, the test code continues. If the test finishes running without failures, it passes.

If the value of the boolean expression is false, the test registers a failure at that line. The test keeps running, but its outcome is firmly decided.

UNIT TESTS

SETUP AND TEARDOWN

XCTestCase has methods named setup() and teardown() that you can override to tweak test behavior.

```
override func setup() { }
override func teardown() { }
```

All these methods do is run before each test (setup) and after each test (teardown). Overriding them is a convenient way to establish the context of the test without repeating the same lines over and over.

UTILITY CODE

In addition to housing test methods, test files are also regular swift files. That means you can write new code in them. You can add classes, functions - whatever. Just keep in mind that **test code is not part of your app**. Anything you write in test files should be dedicated to testing.

```
func setUpExampleToDoList() {
    todos = [ToDo(“breathe”), ToDo(“sleep”)]
}

func testAddingToDoItem() {
    setUpExampleToDoList()
    todos.append(ToDo(“eat”))
    XCTAssert(todos.count == 3)
}
```

TESTING

Remember, tests should be:

Focused (a test should test one thing)	**Clear** (the code should be short and easy to read)
Consistent (repeating a test should yield the same result)	**Quick** (running tests should be temptingly easy)

Practice:

1) In a new test target, write tests for the array members count, append(), remove(at:) and sort().
2) Discuss with a partner why you wrote these tests the way you did.