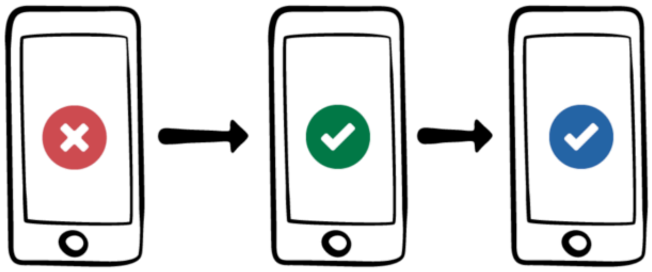
BEGINNING iOS UNIT & UI TESTING



HANDS-ON CHALLENGES

Beginning iOS Unit and UI Testing

Joshua Greene

2 Test Basics - Challenge

By Joshua Greene

Every unit test has three basic sections: **given**, **when** and **then**.

- **Given** is the initial state or setup before the test can be performed.
- **When** is the action or behavior that's being tested.
- **Then** is verification that the expected behavior occurred or value was returned.

Each section may contain zero, one or more lines of code.

It's often useful to add `// given`, `// when` and `// then` comments to complex test methods. This helps explain, "What exactly is this test doing?"

For example, here's how we would have annotated `testFirstPancakeHouseHasExpectedValues()` when it was the *only* test:

```
func testFirstPancakeHouseHasExpectedValues() {

  // given
  collection = PancakeHouseCollection()

  // when
  let bundle = Bundle(for: PancakeHouseCollectionTests.self)
  collection.loadPancakeHouses("test_pancake_houses",
                               in: bundle)

  // then
  let pancakeHouse = collection[0]

  XCTAssertEqual(pancakeHouse.name, "name 1")
  XCTAssertEqual(pancakeHouse.details, "details 1")
  XCTAssertEqual(pancakeHouse.photo,
                 UIImage(named: "pancake\(plistIndex)"))
}
```

In the demo, however, we refactored out `verifyPancakeHouseHasExpectedValues(index:)` to eliminate duplicate code after we wrote the second test method.

Do the given, when and then sections still exist? You bet! They've just been moved:

```
override func setUp() {
  super.setUp()
  // given
  collection = PancakeHouseCollection()

  // when
  collection.loadPancakeHouses("test_pancake_houses", in: bundle)
}

func verifyPancakeHouseHasExpectedValues(index: Int) {

  // then
  let pancakeHouse = collection[index]
  let plistIndex = index + 1

  XCTAssertEqual(pancakeHouse.name, "name \(plistIndex)")
  XCTAssertEqual(pancakeHouse.details, "details \(plistIndex)")
  XCTAssertEqual(pancakeHouse.photo,
                 UIImage(named: "pancake\(plistIndex)"))
}
```

Whether or not you explicitly annotate these sections (some developers do and others don't), it's useful to consider them whenever you're writing unit tests.

In the following challenge, and throughout the rest of this video series, you'll frequently see `// given`, `// when` and `// then` annotations for complex test methods to make them a little easier to understand. :]

Challenge

Write a unit test that verifies a new pancake house can be added using this method on `PancakeHouseCollection`:

```
public func addPancakeHouse(_ pancakeHouse: PancakeHouse)
```

For the challenge solution, keep scrolling beyond the hints!

Hints

You can create a new `PancakeHouse` using a dictionary like this one:

```
let dict: [String: Any] = ["name": "Test Pancake House",
                           "priceGuide": 1,
                           "rating": 1,
                           "details": "Test"]
```

You'll then need to call `addPancakeHouse(_:)` and assert that `collection._pancakeHouses` contains the new pancake house.

Challenge solution

Here's a sample solution:

```
func testCanAddPancakeHouse() {

  // given
  let dict: [String: Any] = ["name": "Test Pancake House",
                             "priceGuide": 1,
                             "rating": 1,
                             "details": "Test"]

  let pancakeHouse = PancakeHouse(dictionary: dict)!

  // when
  collection.addPancakeHouse(pancakeHouse)

  // then
  XCTAssertTrue(collection._pancakeHouses.contains(pancakeHouse))
}
```

Über challenge

Write another unit test for this method on `PancakeHouseCollection`:

```
public func removePancakeHouse(_ pancakeHouse: PancakeHouse) throws
```

You can test removing a pancake house by grabbing the first one, calling `removePancakeHouse(_:)` and asserting that the collection doesn't contain it.

Since this method `throws`, use `try!` when you call it.

For a sample solution, check out the completed challenge in the resources for this video.