# Getting Started with Unit Testing

## Overview

The principal objective of this lab is to get you to think about what is a suitable range of tests for some existing code.

You will be given a prewritten CustomDate object, which is just a wrapper around three integers for day, month, and year. Your task is to write tests to verify the functionality of the CustomDate object. You will need to think of a good range of both positive and negative test cases, to ensure that the functionality you write both accepts the positive cases and rejects the negatives.

## Source folders

* <LAB\_HOME>\labs\testing
* <LAB\_HOME>\solutions\testing

## Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Defining simple test cases
2. Going further with test cases
3. Testing more complex code
4. Additional suggestions

## Getting started with the Student code

Go to the *Student* folder, which contains 4 files. CustomDate.js defines a prewritten CustomDate object, which is already complete. Here's a quick description:

* The constructor performs simple validation via the logic in the setDay(), setMonth(), and setYear() functions. These functions throw an exception if the date is out of range in any way.
* There are also getter functions to get the day, month, and year. The purpose of the getter and setter functions is to encapsulate the actual values held in the object (the actual values are named \_day, \_month, and \_year). Later in the course, we'll see a better way to encapsulate fields in an object, so that the fields really are private like in a real C# or Java class.
* The equals() function compares the current CustomDate object with another one, and returns true or false accordingly.
* The offsetByDays() function takes an integer parameter, and returns a new CustomDate object that is offset by the specified number of days.
* The toString() function returns a textual representation of the current CustomDate object.

There are 3 other files in the *Student* folder:

* CustomDateTest.html is a simple HTML page to host your tests and to display their outcomes. This file is similar to what you saw during the chapter, and is already complete.
* CustomDateTest.js is a JavaScript file where you will define all your tests. We've set up the boilerplate code as per the chapter. You'll spend most of your time in this lab adding tests in here.
* TestLib.js contains the test library functions that we discussed during the chapter, i.e. runTests(), assert(), assertEquals(), fail(), and displayMessage(). You'll use these functions to formulate your tests.

## Exercise 1: Defining simple test cases

Open CustomDateTest.js and add the following code:

* Define a setup function that creates a new CustomDate object, e.g. containing your date of birth. This object will be used in all the subsequent tests that you write in this lab.
* Define a function to test that the getter functions in the CustomDate object return the correct values. From a purist point of view you should define 3 separate test functions here, but in practice you might prefer to put all the tests into a single function. Save your work and open CustomDateTest.html in a browser. It should display a series of green messages indicating all the tests so far have passed.
* Back in CustomDateTest.js, define a function to test that the setter functions in CustomDate set the values correctly. You should specify valid values for the day, month, and year at this stage – i.e. you're testing positive conditions here. Save your work and re-run the tests to verify positive outcomes. As an aside, notice how quick and easy it is to run the tests, and to retain previous tests – these are important ingredients in a useable unit testing framework.
* Back in CustomDateTest.js, define functions to test that the equals() function returns true for two dates that are the same, and false if two dates are different. Run these tests to verify the code.

## Exercise 2: Going further with test cases

Add some additional tests in CustomDateTest.js, as follows:

* Test that the setter functions in the CustomDate object detect invalid dates. In other words, if you try to set an invalid day or month, the setter functions should throw an exception.
* Test that the equals() function returns false if you pass in an invalid parameter (i.e. a non-CustomDate object). You'll find that the equals() function actually fails this test – we've deliberately implemented equals() incorrectly so that you can detect this deficiency via testing. Improve equals() so that it passes your test.

## Exercise 3: Testing more complex code

In this exercise you'll write tests to test the offsetByDays() function. You might want to take a quick look at this function in CustomDate.js to remind yourself how it works. We've actually embedded some deliberate bugs in here too, which you have to weed out with your tests.

Open CustomDateTest.js and write tests such as the following:

* If you call offsetByDays() with a value of 0 for the days parameter, the function should return the same date value.
* If you call offsetByDays() with a value of 1 for the days parameter, the function should return the date of the following day.
* If you call offsetByDays() with a value of -1 for the days parameter, the function should return the date of the previous day.
* If you call offsetByDays() with a relatively large value for the days parameter, the function should return a date in another month. Likewise for subtracting a large number of days.
* If you call offsetByDays() with a very large value for the days parameter, the function should return a date in another year. Likewise for subtracting a very large number of days
* As an interesting "edge" condition (where things can often go wrong), if you call offsetByDays() with the date of New Year's Eve and a days parameter of 1, it should return the date of New Year's Day.
* If you subtract 1 day from New Year's Day, you should get New Year's Eve of the previous year.

**Exercise 4 (If time permits): Additional suggestions**

* At the moment, assertEquals() takes 3 parameters, where the first parameter is a description. This parameter is superfluous – assertEquals() really only needs 2 parameters (i.e. the expected value and the actual value). If the values don't match up, throw an error showing the expected and actual values.
* Think about some additional functions that you could implement in the CustomDate object.
* Consider how you can test these new functions.