COMP3006 – Full-Stack Development

Workshop 5 – Model Answer

Autumn 2020

This set of model answers are intended to show possible ways of solving the unit test exercises in Workshop 5 – there are others. As stated in the workshop sheet, I have developed these answers using jQuery. If you would like feedback on a vanilla JavaScript solution, please ask during the lab session.

1. The first question requires you to test that a function returns the correct string. The function is called **sayHello**, takes a single string argument, and returns the string **"Hello *argument*"**, where *argument* should be replaced with a person's name.

   To test the function, you need to create a suite, and within it a test. The test should:

   (a) create a test case (by defining a variable in which to hold a name to pass to the function);

   (b) call the function;

   (c) check that the result is correct.

   The following code does this:

```
suite("Hello World suite", function() {

    test("Test function returns correct response", function() {
        // Set up the data for the test and call the function.
        let name = "David";
        let response = sayHello(name);

        // Check that the response was as expected.
        chai.assert.equal("Hello David", response, "Incorrect function response");
    });

});
```

   To get this test to pass you need to implement the **sayHello** function. Add the following to your application code:

```
function sayHello(name) {
    return "Hello " + name;
}
```

2. The next exercise displays the message generated in the previous question in a paragraph element in the page using a name entered in a text field. Here is the test:

```
    test("Test input and button click populates message", function() {
        // Set up the form data and press the button.
        let name = "David";
        $("#name").val(name);
        clickElement($("#btn"));

        // Check that the message was populated correctly.
        let msg = $("#msg").html();
        chai.assert.equal("Hello David", msg, "Incorrect message in page");
    });
```

The test sets a value in the input and clicks the button using the **clickElement** function provided for you in the template for this workshop. It then extracts the HTML from the **msg** element and checks that it has the correct value.

One of the important principles of unit testing is that the system state should be restored at the end of a test – this is why you're able to run the tests in any order. This test alters the state of the program by inserting a value into the **name** field and updating the **msg** field. To clean up after the test, add a **teardown** method to remove these values ready for the next test:

```
    teardown(function() {
        $("#name").val("");
        $("#msg").html("");
    });
```

This will be run following every test, so you are assured that the page will always have been reset.

All of the following code snippet has been seen above, but so that you can see how it all fits together here is the complete test suite for the first two exercises:

```
suite("Hello World suite", function() {

    teardown(function() {
        $("#name").val("");
        $("#msg").html("");
    });

    test("Test function returns correct response", function() {
        // Set up the data for the test and call the function.
        let name = "David";
        let response = sayHello(name);

        // Check that the response was as expected.
        chai.assert.equal("Hello David", response, "Incorrect function response");
    });

    test("Test input and button click populates message", function() {
        // Set up the form data and press the button.
        let name = "David";
        $("#name").val(name);
        clickElement($("#btn"));

        // Check that the message was populated correctly.
        let msg = $("#msg").html();
        chai.assert.equal("Hello David", msg, "Incorrect message in page");
    });

});

suite("Calculator suite", function() {
```

3. This part of the workshop requires you to write unit tests to test the components of a simple calculator. The first part requires you to test the **sum** function – you will want to start a new test suite, as follows:

```
suite("Calculator suite", function() {

});
```

Within the suite, add a unit test as follows:

```
    test("Sum test", function() {
        // Define the correct totals.
        let totals = [3, 6, 9, 12, 15];
        // Loop over each test case and assert that the result is correct.
        for (i=0; i<self.x.length; i++) {
            let result = sum(self.x[i], self.y[i]);
            chai.assert.equal(result, totals[i], "Incorrect sum result");
        }
    });
```

The test refers to two attributes on the test – **self.x** and **self.y**. These contain test data that will be used in multiple tests, and need to be set up. We use the **suiteSetup** method to do this:

```
    suiteSetup(function() {
        self.x = [1, 2, 3, 4, 5];
        self.y = [2, 4, 6, 8, 10];
    });
```

That completes the test for the first part – you will need to implement the application code to get the tests to pass. Add the following to the application JS:

```
function sum(a, b) {
    return a + b;
}
```

4. This question requires you to repeat the steps from the previous exercise to enable the multiplication feature. First, add another unit test to check that the **product** function works correctly:

```
    test("Multiply test", function() {
        // Define the correct totals.
        let totals = [2, 8, 18, 32, 50];

        // Loop over each test case and assert that the result is correct.
        for (i=0; i<self.x.length; i++) {
            let result = product(self.x[i], self.y[i]);
            chai.assert.equal(result, totals[i], "Incorrect product result");
        }
    });
```

   This test defines its own totals, as the ones in the previous test are not applicable, but relies on the **self.x** and **self.y** inputs as before.

   Running the test will fail until you implement the **product** function, which is done by adding the following to the application code:

```
function product(a, b) {
    return a * b;
}
```

   The test should now pass.

5. Finally, you are asked to unit test the user interface. You will test three aspects: the look and feel of the results paragraph; the functionality of the **sum** button; and the functionality of the **multiply** button.

   Begin by adding the following test:

```
    test("Result display test", function() {
        // Obtain a handle on the results paragraph.
        let result = $("#result");

        // Check the background colour.
        chai.assert.equal(rgb2hex(result.css("background-color")), "#ffffff", "Wrong col");

        // Check the font is correct.
        chai.assert.equal(result.css("font-size"), "20px", "Wrong font size");
    });
```

And modify the application code by adding the following to the style block:

```
#result { background-color: #FFF; font-size: 20px; }
```

The next step is to test the button clicks. Beginning with the **sum** button, add the following test:

```
    test("Sum button test", function() {
        // Assign some values to the form.
        $("#number1").val(2);
        $("#number2").val(4);

        // Click the button.
        clickElement($("#btnAdd"));

        // Confirm that the correct result is shown in the page.
        let result = $("#result").html();
        chai.assert.equal(result, "6", "Incorrect sum shown in page");
    });
```

This test adds values to the two form inputs, clicks the button using the **clickElement** function provided in the workshop template, and confirms that the **result** paragraph has the correct result in it. You only need to test with one value, as you've already checked that the **sum** function itself works. Here we're just confirming that the UI processes its results correctly. Note that it doesn't reset the form data – we will take care of that shortly with a **teardown** method.

To get the test to pass, add an appropriate event handler to the button, such as the following:

```
    $("#btnAdd").click(function() {
        let num1 = parseInt($("#number1").val());
        let num2 = parseInt($("#number2").val());
        let result = sum(num1, num2);
        $("#result").html(result);
    });
```

Then repeat the same process for the **multiply** button, beginning with the test:

```
    test("Multiply button test", function() {
        // Assign some values to the form.
        $("#number1").val(2);
        $("#number2").val(4);

        // Click the button.
        clickElement($("#btnMul"));

        // Confirm that the correct result is shown in the page.
        let result = $("#result").html();
        chai.assert.equal(result, "8", "Incorrect product shown in page");
    });
```

Implement the event handler code so that the test passes:

```
    $("#btnMul").click(function() {
        let num1 = parseInt($("#number1").val());
        let num2 = parseInt($("#number2").val());
        let result = product(num1, num2);
        $("#result").html(result);
    });
```

Finally, implement the **teardown** method. Whereas we set up the test data using **suiteSetup**, we want the form data and result to be rest after each test – that means we need to use **teardown**. Add the following to the suite:

```
    teardown(function() {
        $("#number1").val("");
        $("#number2").val("");
        $("#result").html("");
    });
```

Again, all of the code has been provided above, but here is the complete suite so that you can see how it all fits together:

```
suite("Calculator suite", function() {

    suiteSetup(function() {
        self.x = [1, 2, 3, 4, 5];
        self.y = [2, 4, 6, 8, 10];
    });

    teardown(function() {
        $("#number1").val("");
        $("#number2").val("");
        $("#result").html("");
    });

    test("Sum test", function() {
        // Define the correct totals.
        let totals = [3, 6, 9, 12, 15];

        // Loop over each test case and assert that the result is correct.
        for (i=0; i<self.x.length; i++) {
            let result = sum(self.x[i], self.y[i]);
            chai.assert.equal(result, totals[i], "Incorrect sum result");
        }
    });

    test("Multiply test", function() {
        // Define the correct totals.
        let totals = [2, 8, 18, 32, 50];

        // Loop over each test case and assert that the result is correct.
        for (i=0; i<self.x.length; i++) {
            let result = product(self.x[i], self.y[i]);
            chai.assert.equal(result, totals[i], "Incorrect product result");
        }
    });

    test("Result display test", function() {
        // Obtain a handle on the results paragraph.
        let result = $("#result");

        // Check the background colour.
        chai.assert.equal(rgb2hex(result.css("background-color")), "#ffffff", "Wrong col");

        // Check the font is correct.
        chai.assert.equal(result.css("font-size"), "20px", "Wrong font size");
    });

    test("Sum button test", function() {
        // Assign some values to the form.
        $("#number1").val(2);
        $("#number2").val(4);

        // Click the button.
        clickElement($("#btnAdd"));

        // Confirm that the correct result is shown in the page.
        let result = $("#result").html();
        chai.assert.equal(result, "6", "Incorrect sum shown in page");
    });

    test("Multiply button test", function() {
        // Assign some values to the form.
        $("#number1").val(2);
        $("#number2").val(4);

        // Click the button.
        clickElement($("#btnMul"));

        // Confirm that the correct result is shown in the page.
        let result = $("#result").html();
        chai.assert.equal(result, "8", "Incorrect product shown in page");
    });

});
```