COMP3006 – Full-Stack Development

Workshop 2 – Model Answer

Autumn 2020

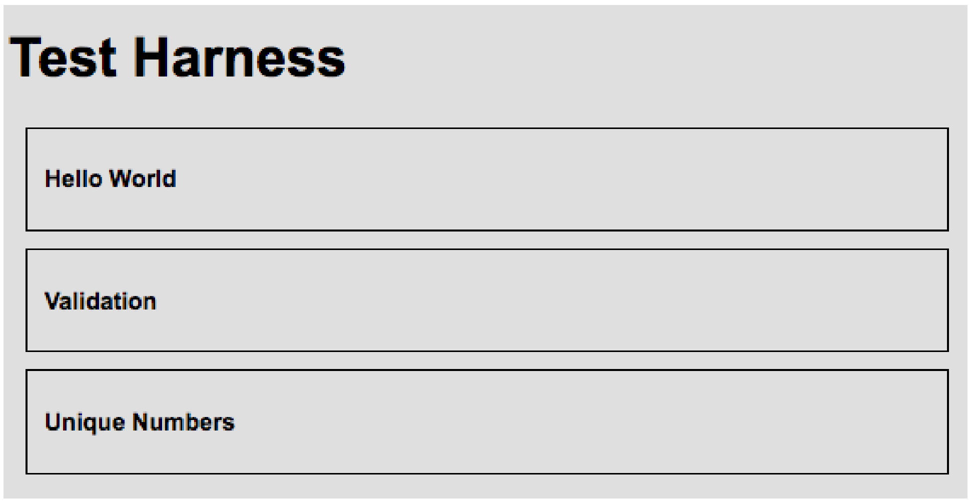
This set of model answers are intended to show possible ways of solving the unit test exercises in Workshop 2 – there are others. As stated in the workshop sheet, I have developed these answers using jQuery. If you would like feedback on a vanilla JavaScript solution, please ask during the lab session. To remind you, I have not provided a solution for exercises where you had the full code in the lab sheet (Exercise 2, in this case), and I am not providing a model answer for the mini project or extension activities. Please attempt the exercises before looking at the answers.

1. This week's workshop required you to construct some small JavaScript programs. You need a simple interface, so the first exercise required you to construct a simple test harness using HTML that you can display in the browser. Here is the code needed to do this – as always, there are multiple ways of doing this, so having a different solution to mine is no problem as long as it works.

   Here is the HTML and CSS for my version of the test harness.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Test Harness</title>
        <style>
            body { background-color: #DFDFDF; font-family: Arial; }
            section { border: 1px #000 solid; margin: 10px; padding: 10px; }
            section h1 { font-size: 14px; }
        </style>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></scr
        <script src="javascript.js"></script>
    </head>
    <body>
        <h1>Test Harness</h1>
        <section>
            <h1>Hello World</h1>
        </section>
        <section>
            <h1>Validation</h1>
        </section>
        <section>
            <h1>Unique Numbers</h1>
        </section>
    </body>
</html>
```

   The result should look something like this – a page heading, and three sections each of which has its own heading:


Test Harness

Hello World

Validation

Unique Numbers

2. Full code provided in workshop sheet.

3. The second exercise is about validating numbers – we want to know if the number entered by the user is less than ten, or greater than or equal to ten. The first step is to add some HTML so that we add the input for the user to type in, the button to initiate the check, and the instructions. Modify the validation section of your test harness as follows:

```
        <section>
            <h1>Validation</h1>
            <p>Please enter a number: <input type="number" id="number" /></p>
            <button id="checkNumber">Check Number</button>
            <div id="numberMessage" class="message hidden">
                <p id="numberMessageText"></p>
            </div>
        </section>
```

You are going to need some CSS styles for the message. Modify the CSS section of your test harness document as follows:

```
        <style>
            body { background-color: #DFDFDF; font-family: Arial; }
            section { border: 1px #000 solid; margin: 10px; padding: 10px; }
            section h1 { font-size: 14px; }

            .message { width: 100%; padding: 10%px; margin-top: 10px; }
            .hidden { display: none; }
            .greater { background-color: #FF9999; border: 1px #CC3300 solid; }
            .less { background-color: #CCffCC; border: 1px #009933 solid; }
        </style>
```

Finally you can implement the behaviour of the program. The whole program code will live within the event handler of the button, so in your JavaScript code add such an event handler. You should add this alongside the existing code in your window onload event handler – all of the button handlers can go into there. The page has been designed so that they do not conflict with each other, for example by specifying different button IDs.

The `checkNumber` event handler does the following:

- Extract the number from the input field (using the **val** method).
- Remove any spurious classes form the `numberMessage` field (it shouldn't have either of the **greater** or **less** classes, but it might if you've pressed the button twice without reloading the page – so we're just tidying up here).
- Check if the number is less than ten – add the corresponding message and class.

- Remove the **hidden** class so that the message is visible in the browser.

This process is implemented as follows:

```
$(function() {
    $("#checkNumber").click(function() {
        let number = $("#number").val();

        $("#numberMessage").removeClass("greater less");

        if (number < 10) {
            $("#numberMessageText").html("Less than ten");
            $("#numberMessage").addClass("less");
        } else {
            $("#numberMessageText").html("Greater than or equal to ten");
            $("#numberMessage").addClass("greater");
        }

        $("#numberMessage").removeClass("hidden");
    });
});
```

4. This exercise is a variation on the theme of the other questions. This time you need an input that takes a comma separated sequence of numbers. As such you need an input of type **text** and not **number**, or you won't be able to have the commas. Add the following to the unique numbers section:

```
        <section>
            <h1>Unique Numbers</h1>
            <p>Please enter some numbers, separated by commas:
                                 <input type="text" id="numbers" /></p>
            <button id="listUnique">List unique numbers</button>
            <p id="getUniqueNumbers"></p>
        </section>
```

There is no CSS for this question, so you can now move straight on to the JavaScript. It's possible to do this with the in-built JavaScript **set** functionality, but I would rather you use loops and an array to do it manually – the point of the exercise is to practice this. I've provided the solution for that approach, and have included the set approach in next week's workshop.

The process for extracting the set and displaying it in the page is as follows:

- Extract the numbers entered using the **split** method to split on each comma.
- Initialise an empty array to contain the unique values.
- For each number the user entered, if that number does not exist in the `uniqueNumbers` array (check with the **includes** method) add it using the **push** method.
- Append each unique number to the `uniqueNumbers` element in the document. Add a space after each for readability.

These steps are implemented as follows:

```
$(function() {
    $("#listUnique").click(function() {
        let numbers = $("#numbers").val().split(",");

        let uniqueNumbers = [];

        for (i=0; i<numbers.length; i++) {
            if (!uniqueNumbers.includes(numbers[i])) {
                uniqueNumbers.push(numbers[i]);
            }
        }

        for (i=0; i<uniqueNumbers.length; i++) {
            $("#uniqueNumbers").append(uniqueNumbers[i] + " ");
        }
    });
});
```