COMP3006 – Full-Stack Development

Workshop 3 – 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, 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. Here the aim is to define an object and print it out in the console. You could do this entirely in the console, or wrap it in a simple HTML page. Here is the JavaScript required, in either case:

```
// Define the object.
let obj = {
    forename: "David",
    surname: "Walker",
    dob: "1970-01-01"    // Unix clock started on this date, not my
};                       // actual D.O.B...

// Print out the whole object.
console.log(obj);

// Print out each of the attributes and their values.
console.log(obj["forename"]);
console.log(obj["surname"]);
console.log(obj["dob"]);

// Could also do the same with a loop over attribute values.
for (var property in obj) {
    if (obj.hasOwnProperty(property)) {
        console.log(property + ": " + obj[property]);
    }
}
```

2. Most of this question was covered in last week's model answers. Here I'll show you the new JavaScript code. You need to have a function that takes an array and constructs a set from it. Here is an example of how that is done, along with some code that calls the function:

```
function unique(arr) {
    return Array.from(new Set(arr));
}

$(function() {
    let arr = [1, 1, 2, 5, 3, 2, 1, 4, 7, 6, 3];
    let theSet = unique(arr);
    console.log(arr);
});
```

The `Array.from` method converts the output from the `Set` into a JavaScript array.

3. This exercise requires you to initialise a date object to the current system time, and display a time-appropriate greeting in a webpage. So, first you will need a webpage in which to display the message:

```
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scr
        <script src="dates.js"></script>
    </head>
    <body>
        <h1>Current Time</h1>
        <p id="message"></p>
    </body>
</html>
```

You will see that this HTML links to two external JavaScript files – one links to jQuery and the other contains the bespoke JavaScript code written for this exercise. This file looks like this:

```
function timeGreeting() {
    let now = new Date();

    if (now.getHours() < 12) {
        return "Good morning";
    } else if (now.getHours() < 18) {
        return "Good afteroon";
    }
    return "Good evening";
}

$(function() {
    let msg = timeGreeting();
    $("#message").html(msg);
});
```

The file contains two functions. The first is `timeGreeting`, which initialises a date object (with `new Date()`) and then uses the `getHours()` method within an if-statement to determine which message should be returned.

The second function runs when the page loads, and does two things. First it calls the `timeGreeting` method, assigning the result to a variable called `msg`, and then it adds the message to the *message* paragraph defined in the HTML.

4. This exercise has three parts to it: creating the form, creating the data structures, and creating the event handlers for the buttons. We will begin by creating the form:

```
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scr
        <script src="people.js"></script>
        <link rel="stylesheet" href="people.css">
    </head>
    <body>
        <h1>Lecturers</h1>
        <table id="lec_table">
            <tr><th>Name</th><th>Email</th><th>Room Number</th></tr>
        </table>
        <h2>New lecturer</h2>
        <p>Name: <input id="lec_name" /></p>
        <p>Email: <input id="lec_email" /></p>
        <p>Room number: <input id="lec_room" /></p>
        <button id="add_lec">Add</button>
        <h1>Students</h1>
        <h2>New lecturer</h2>
        <table id="stu_table">
            <tr><th>Name</th><th>Email</th><th>Student Number</th></tr>
        </table>
        <p>Name: <input id="stu_name" /></p>
        <p>Email: <input id="stu_email" /></p>
        <p>Student number: <input id="stu_id" /></p>
        <button id="add_stu">Add</button>
    </body>
</html>
```

By now there should be nothing particularly exciting about this code. The head section links to the relevant external files (two JavaScript sources and a CSS file – there isn't much in the CSS but I've added a bit to make the tables more attractive). The body section contains the forms, tables and buttons.

Here is the CSS, which is largely based on the example I demonstrated in Lecture 1:

```
table { border-collapse: collapse; }
td, th { border: 1px #000 solid; padding: 5px; }
tr:nth-child(odd) { background-color: #dfdfdf; }
th { background-color: #afafaf; color: #fff; }
```

The next thing to do is start the JavaScript portion of the exercise. I would start by defining the data structures. As I said in the lab sheet, you can do this with either classes or using one of the function patterns described in the lecture. Here I'll give an example of using classes and function constructors. First, the class approach:

```
class Person {
    constructor(name, email) {
        this.name = name;
        this.email = email;
    }
}

class Lecturer extends Person {
    constructor(name, email, room) {
        super(name, email);
        this.room = room;
    }
}

class Student extends Person {
    constructor(name, email, studentId) {
        super(name, email);
        this.studentId = studentId;
    }
}
```

Alternatively, you could write a function constructor:

```
function Person(name, email) {
    this.name = name;
    this.email = email;
}

function Lecturer(name, email, room) {
    Person.call(this, name, email);
    this.room = room;
}

Lecturer.prototype = Object.create(Person.prototype);

function Student(name, email, studentId) {
    Person.call(this, name, email);
    this.studentId = studentId;
}

Student.prototype = Object.create(Person.prototype);
```

Having defined the data structures, whichever approach you have taken, you need to think about how to add the data to a table. There are various approaches you might take to do this – I have created a function that adds a row to a specified table. Here is the code for this:

```
function addToTable(obj, tableId) {
    let row = "<tr><td>" + obj.name + "</td><td>" + obj.email + "</td><td>";
    if (obj instanceof Lecturer) {
        row += obj.room + "</td></tr>";
    } else {
        row += obj.studentId + "</td></tr>";
    }

    $(tableId).append(row);
}
```

The ID of the table (in this case it will be either `#lec_table` or `#stu_table`, referring back to the HTML. The `instanceof` operator is used to work out if the object you've passed is a `Lecturer` or a `Student`, so that the right data is appended.

The final step is to enable the button event handlers. Each button has an event handler, which adds the data in the inputs to the table having created an object. Here is the code:

```
$(function(){
    $("#add_lec").click(function() {
        // Extract the lecturer details from the form.
        let name = $("#lec_name").val();
        let email = $("#lec_email").val();
        let room = $("#lec_room").val();

        // Instantiate a new lecturer object.
        let lecturer = new Lecturer(name, email, room);

        // Add the new lecturer to the lecturer table.
        addToTable(lecturer, "#lec_table");

        // Clear the fields.
        $("#lec_name").val("");
        $("#lec_email").val("");
        $("#lec_room").val("");
    });

    $("#add_stu").click(function() {
        // Extract the student details from the form.
        let name = $("#stu_name").val();
        let email = $("#stu_email").val();
        let studentId = $("#stu_id").val();

        // Instantiate a new student object.
        let student = new Student(name, email, studentId);

        // Add the new student to the student table.
        addToTable(student, "#stu_table");

        // Clear the fields.
        $("#stu_name").val("");
        $("#stu_email").val("");
        $("#stu_id").val("");
    });
});
```

The two event handlers are largely the same, acting on their respective fields, creating the appropriate objects, and modifying the appropriate parts of the UI. Each handler does the following:

- Extract the values from the inputs and assign them to variables.
- Instantiate the appropriate object using the data from the inputs.
- Use the `addToTable` function to add the data to the appropriate table.
- Clear the field values so the user can enter their next data.