Mathsoc Programming Workshop: Week 1

Introduction

For each Program, you have been provided with some incomplete code to get you started, and working through the exercises should tell you how to develop this into a fully working interactive web page. If at any stage you get stuck you are advised to ask others, the workshop helpers, or look for relevant materials online.

In this workshop, you will be learning primarily through examples and hands on experience, as it is the author's opinion that this is the only real way to learn programming, however, you will find the exercises easier if you also spend some time to look some tutorials and references for HTML and JavaScript online. The following resources are particularly useful:

- `http://www.w3schools.com/js/default.asp`
- `http://www.w3schools.com/html/default.asp`
- `http://www.w3schools.com/jquery/default.asp`

Getting the code and using c9.io

In order to complete this workshop, you will first need to create an account at `https://c9.io`, an online development environment which will allow you to edit and preview your programs in your web browser.

Once you have created your account, you should login and click on *Create a new workspace*. Here you should fill in the *Clone from Git or Mercurial URL* field as `https://github.com/twright/mathsoc-programming-week1` which will allow you to get a copy of the workshop code, and under *Choose a template* select HTML5. Now you should click *Create workspace*, and once this is done, you will be ready to start coding.

Program 1: Click The Button

The goal of these exercises is to make a basic interactive webpage, with a button which actually does something.

For these exercises you will need to edit the files `click-the-button.html` and `click-the-button.js`.

1. Have a look at `click-the-button.html` and `click-the-button.js` and try to understand the code. This is a basic example of a static web with a HTML file which determines the content of the the page, and which is linked to a JavaScript file, which at the moment is empty, but will soon allow us to start programming and add interactive functionality to the page.

   You should preview the page by opening `click-the-button.html` and selecting *Preview* → *Live Preview File* from the menu.

   You should try changing the text to something more interesting, and seeing how this changes are reflected in the preview of the page.

2. Your next task is to edit `click-the-button.html` to add a clickable button.

   You can do this by adding the code

   ```
   <button id="the-button">
       Click The Button
   </button>
   ```

   inside the second paragraph (`<p>`) tag. Now when you preview the page, it should contain your newly added button.

This code specifies a button with text `Click The Button`. Importantly, we say `id="the-button"` which sets the ID of the button. This will allow our JavaScript code to find the button when we want to add interactive functionality.

3. Now we have have added a button to the page it would be nice if it actually did something.

   To do this we need to start editing the `click-the-button.js` file. We will make this change in two stages.

   (a) First we must define a function determining what action we want to take when the button is clicked.

   Functions specify smaller, reusable programs from which we can build larger programs (functions can be mathematical functions like sin and cos which take some arguments and produce a result, but they may also perform actions like changing the content of the page, or asking the user a question).

   For example, in JavaScript we can write a function to compute the function $f(x) = x^2$ like,

   ```
   function f(x) {
       return x*x;
   }
   ```

   or a function to RickRoll the user like

   ```
   function rickRoll() {
       window.location.href = "https://goo.gl/faEUwX";
   }
   ```

   For this part of the exercise you should create a function called `sayHello` containing the code:

   ```
   alert("Hello World!");
   ```

   (b) From the previous part of the question, you should now have a function `sayHello`. But how can we actually get it to run when the button is clicked?

   For this we need to add the following code to the bottom of `click-the-button.js`:

   ```
   $("#the-button").click(sayHello);
   ```

   The first bit of this code `$("#the-button")` finds the button in the page via the ID we gave it and `.click(sayHello)` attaches the function `sayHello` as the action to take when it is clicked.

   You should now preview the page and check that is works if expected. If so, congratulations, you have now created your first interactive website.

Program 2: Rock Paper Scissors

The goal of these exercises to make a fully functioning game of Rock Paper Scissors.

For these exercises you will need to edit the files `rock-paper-scissors.html` and `rock-paper-scissors.js`.

1. You should take a look at the file `rock-paper-scissors.html` which contains the start of our rock paper scissors game.

   This page uses a form with HTML radio selection boxes which allow the user to select one of a list of options. Each has a text label, telling the user what it does, and a value, which is what our program will see as the user's choice. Notice that the ID of the form is set to `rock-paper-scissors`; this will be useful when our JavaScript program needs to interact with the form.

```
<form id="rock-paper-scissors">
    <label>
        <input type="radio" name="choice" value="rock" />
        Rock
    </label>
    <label>
        <input type="radio" name="choice" value="paper" />
        Paper
    </label>
</form>
```

However it seems that our game is missing a key option, that is, the ability for the user to select *Scissors*. You should look at the existing code, and modify it to add a *Scissors* option with label `Scissors` and value `scissors`.

2. You should now add a button, which will later allow the user to play the game. This button should have ID `play` and text `Play`.

   Note: For this exercise you just need to add the button to the page. How to actually make it do something will be explained in the next exercise.

3. Now we will take the next small step, and implement a simplified version of Rock Paper Scissors, when the user wins if they select Rock, and loses if they select anything else.

   For this we need to use an `if` statement, which allows the program to do different things depending on some condition. An example of this is shown in the following function

   ```
   function whatIsYourFavouriteMathematicalConstant(s) {
       if (s == "pi") {
           alert("Good choice");
       } else if (s == "e") {
           alert("Even better choice");
       } else if (s == "tau") {
           window.location.href = "https://goo.gl/faEUwX";
       } else {
           alert("You did not pick pi, e, or tau.");
       }
   }
   ```

   You will also need to get the value the user has selected in the form. You can do this using the code

   ```
   $("#rock-paper-scissors input[name='choice']:checked").val()
   ```

   (you should compare this code with the form in `rock-paper-scissors.html` and try to figure out how it works).

   Armed with this knowledge you should be able to implement this simplified version of Rock Paper Scissors in two steps.

   (a) Write a function called `playRockPaperScissors` which runs the code `alert("You Win!")` if the user has selected `rock` and `alert("You Lose!")` otherwise.

   (b) Attach the `playRockPaperScissors` function you have written to run when the `Play` button is clicked.

   You should now test your game to check it works as expected.

4. The next step is to implement the full logic of the Rock Paper Scissors game, but for the moment, with an opponent whom always selects Paper (but we should make sure the program is general enough to handle whatever the opponent picks, as we will be making the opponent smarter in the next exercise).

   You should replace your `playRockPaperScissors` function with the following code

```
function playRockPaperScissors() {
    var computerChoice = "paper";
    var playerChoice = $("#rock-paper-scissors input[name='choice']:checked"
        ).val();
    var outcome = rockPaperScissorsCompare(playerChoice, computerChoice);

    if (outcome == 1) {
        alert("You win!");
    } else if (outcome == -1) {
        alert("You lose!");
    } else {
        alert("Draw!");
    }
}
```

   You should read this code and try and understand how it works. Note that this code uses a new feature of JavaScript, variables, which allows us to store values we have computed, and give them names, which we can then use to refer to them later in our code. The code relies upon another function which we have not yet implemented called `rockPaperScissorsCompare` which takes two arguments which we will call `x` and `y`, and returns 1 if `x` beats `y`, -1 if `y` beats `x`, and 0 if it is a draw.

   Your task is now to implement this function `rockPaperScissorsCompare(x, y)`. Once you have done this, you should now be able to play Rock Paper Scissors, albeit against a very predictable opponent.

5. Now you should be ready to extend this program to a proper game of Rock Paper Scissors, with an opponent which picks its choice randomly.

   You should do this in two steps.

   (a) In JavaScript, you can store multiple values in a single variable using arrays like

```
var xs = [1, "cat", 3.14159265];
```

   (think of this like a vector). Once we have this array, we can get the value at index `xs[i]` (note: arrays are indexed from 0) and the length of the array like `xs.length` (so in our example, `xs[1] == "cat"` and `xs.length == 3`).

   In JavaScript, the function `Math.random()` returns a random number between 0 and 1 and the function `Math.floor(x)` returns the smallest integer less than x.

   Using this knowledge, you should be equipped to write a function `randomChoice(xs)` which takes an array `xs` and returns a random element of this.

   (b) Use the function `randomChoice(xs)` you have created to modify the `playRockPaperScissors` function such that the computer opponent randomly selects its choice as one of `rock`, `paper`, or `scissors`.

   You should now test your program to check it works as expected.

6. As a final exercise, you could build on the previous exercise to implement a new game, Rock Paper Scissors Lizard Spock: `http://www.samkass.com/theories/RPSSL.html`.

   Whilst this may sound like more of the same, it is worth giving some thought as to how you could simplify your code for the previous exercise, to make sure it does not become unwieldy once more

options are added. Specifically, if you have simply implemented your `rockPaperScissorsCompare` function using nested if statements, once we have two extra options, this code will be very long indeed. Instead, you should store the order of what beats what in an array like

```
var rockPaperScissorsWinners = [["rock", "scissors"],
    ["scissors", "paper"], ["paper", "rock"]];
```

and use this to write a much simpler version of `rockPaperScissorsCompare`. In doing this, it may be useful to know that you can test whether an element `x` is in an array `xs` using the code `xs.indexOf(x) > -1`.

Next Week

In next week's workshop, we will put the the interactive programming skills we have developed this week to use learning how to create 2D graphics in the browser and implement an interactive online drawing program.