Computing GCSE Coursework

Thomas Bass
Candidate 4869
Centre 52423
OCR A452 Practical Investication

Word Count: 915

Made with LaTeX

2017

Summary

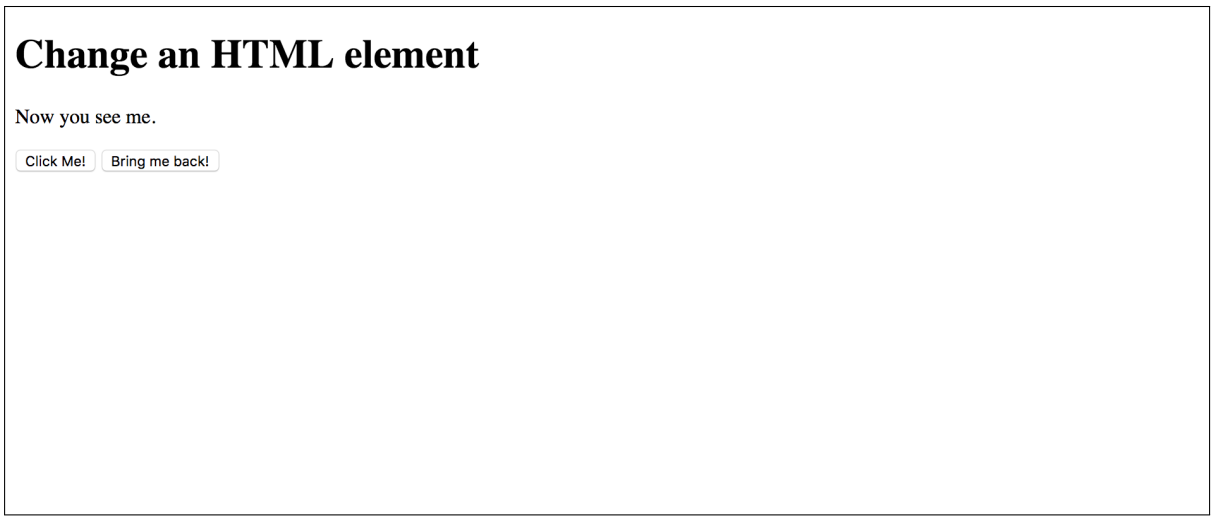
1 Task 1

Often, a web designer wants a change to happen when a user clicks on a screen object or moves the mouse over it. JavaScript can make changes to the HTML elements.
Enter and run this script:

```
<!DOCTYPE html>
<html>
<body>
<h1>Change an HTML element</h1>
<p id="msg">Now you see me.</p>
<button type="button"
onclick="document.getElementById('msg').innerHTML = 'Gone!'">
Click Me!</button>
<button type="button"
onclick="document.getElementById('msg').innerHTML = 'Back again!'">
Bring me back!</button>
</body>
</html>
```

1.1 Explain how you ran this script:

This script was copied into a HTML document, and opened into a web browser.
When the script ran it gave the following output:



The web page took the code entered, ran it, and produced this output.

1.2 Explain what each line of the script does:

1.2.1 The HTML Code:

```
01      <!DOCTYPE html>
02      <html>
03      <body>
04      <h1>Change an HTML element</h1>
05      <p id="msg">Now you see me.</p>
06      <button type="button"
07      onclick="document.getElementById('msg').innerHTML = 'Gone!'">
08      Click Me!</button>
09      <button type="button"
10      onclick="document.getElementById('msg').innerHTML = 'Back again!'">
11      Bring me back!</button>
12      </body>
13      </html>
```

1.2.2 Commentry:

```
01      This declares that the document is a HTML document
02      This opens the <html> tag, and shows that the code enclosed
        is HTML code
03      This opens the <body> tag, and shows that the code enclosed
        is placed in the body of the document
04      This opens the <h1> tag, showing that the text enclosed
        ("Change an HTML Element") is placed in the highest header, and closes
        the tag
05      This opens a <p> tag, showing that the text enclosed ("Now you see me.")
        is paragraph text, and it has the ID "msg", and then the tag is closed
06      This opens a <button> tag, showing that the information enclosed is
        a button, and has type "button"
07      This declares that following an onclick event (when the button is clicked),
        the program will execute a Javascript function to find the elements with
        the ID of "msg" (the body text in line 05) and change its HTML code to "Gone".
08      This line provides the text of the button ("Click Me!") and closes the
        <button> tag
09      This opens a <button> tag, showing that the information enclosed is
        a button, and has type "button"
10      This declares that following an onclick event (when the button is clicked),
        the program will execute a Javascript function to find the elements with
        the ID of "msg" (the body text in line 05) and change its
        HTML code to "Back Again!".
11      This line provides the text of the button ("Bring me back!") and closes the
        <button> tag
12      This closes the <body> tag
13      This closes the <body> tag and ends the document
```

2 Task 2

As is the case with most programming languages, in JavaScript you can use arrays in order to store multiple values under the same identifier. For example, an array of products can be set up as below for use on an ecommerce web site.

```
var products = ["Printer","Tablet","Router"];
```

2.1 Set up an array to include the items shown above, plus a few extras of your choice.

2.1.1 Products:

Printer, Tablet, Router, Network Switch, Monitor, Keyboard, Mouse, 500GB Hard Drive, ATX Motherboard, Memory Card, Flash Drive, Network Switch, Bluetooth Adaptor, Modem, Wireless Speaker, 256GB SSD.

2.1.2 Array:

```
var products = [”Printer”,”Tablet”,”Router”,”Network Switch”,”Monitor”,
”Keyboard”,”Mouse”,”500GB Hard Drive”,”ATX Motherboard”,”Memory Card”,
”Flash Drive”,”Network Switch”,”Bluetooth Adaptor”,”Modem”,
”Wireless Speaker”, ”256GB SSD”];
```

=16 items

2.1.3 Array in code editor:

```
26      var products = ["Printer","Tablet","Router","Network Switch","Monitor",
27      "Keyboard","Mouse","500GB Hard Drive","ATX Motherboard","Memory Card",
28      "Flash Drive","Network Switch","Bluetooth Adaptor","Modem","Wireless Speaker", "256GB SSD"];
```

2.2 Write a script that:

2.2.1 Outputs the items in alphabetical order

```
25  <script>
26    var products = ["Printer","Tablet","Router","Network Switch","Monitor",
27    "Keyboard","Mouse","500GB Hard Drive","ATX Motherboard","Memory Card",
28    "Flash Drive","Network Switch","Bluetooth Adaptor","Modem","Wireless Speaker", "256GB SSD"];
29    function sortProducts(){
30      temp = products.sort();
31      document.write(temp.join('<br/>'));
32    }
33    sortProducts();
34  </script>
```

This code snippet is written in JS embedded in an HTML document. Line 25 opens the `<script>` tag, which indicates to the HTML code that the following is JS script. Lines 26 to 28 then declares `products` to be a global array with the contents as described in part 2.1. Line 29 declares `sortProducts()` to be a function, and opens the code for the function. Line 30 declares `temp` to be the array `products`, sorted alphabetically with the `.sort` function built into JS. Line 31 then outputs the `temp` array, joined by a `<br>` line break, onto the document body with the `.write` function. Line 32 closes the code for the `sortProducts()` function, and Line 33 calls the `sortProducts()` function its self. Line 34 then closes the `<script>` tag, indicating the end of this JS script.

2.2.2 Counts the number of items in the array.

```
36    <script>
37      function printLength(){
38        document.write(products.length);
39      }
40      printLength()
41    </script>
```

This code snippet is again written in JS, and embedded in the same HTML document as in 2.2.1. Line 36 opens the `script` tag to declare the following as JS code. Line 37 then declares `printLength()` to be a function, and opens the code for the function. Line 38 outputs the length of the array with the built-in `.length` function, and writes it to the body with `.write`. The code is then closed in line 39, and the `printLength()` function is called in line 40. Line 41 then closes the `<script>` tag to show the end of the JS script.

2.2.3 Full code:

```
1   <!doctype html>
2   <html>
3   <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width">
6   <title>JS Bin</title>
7   </head>
8   <body>
9     <style>
10      body{
11        background-color: #ececec;
12      }
13      h3 {
14        font-family: arial;
15        color: #3581df;
16      }
17      body {
18        font-family: arial;
19        color: #39ccd2;
20      }
21    </style>
22
23    <h3>Alphabetical:</h3>
24
25    <script>
26      var products = ["Printer","Tablet","Router","Network Switch","Monitor",
27      "Keyboard","Mouse","500GB Hard Drive","ATX Motherboard","Memory Card",
28      "Flash Drive","Network Switch","Bluetooth Adaptor","Modem","Wireless Speaker", "256GB SSD"];
29      function sortProducts(){
30        temp = products.sort();
31        document.write(temp.join('<br/>'));
32      }
33      sortProducts();
34    </script>
35    <h3>Length of Array:</h3>
36    <script>
37      function printLength(){
38        document.write(products.length);
39      }
40      printLength()
41    </script>
42  </body>
43  </html>
```

2.2.4 Output:

Alphabetical:

256GB SSD
500GB Hard Drive
ATX Motherboard
Bluetooth Adaptor
Flash Drive
Keyboard
Memory Card
Modem
Monitor
Mouse
Network Switch
Network Switch
Printer
Router
Tablet
Wireless Speaker

Length of Array:

16

3 Task 3

3.1 (i) Make a list of assets that will be required in order to produce this display.

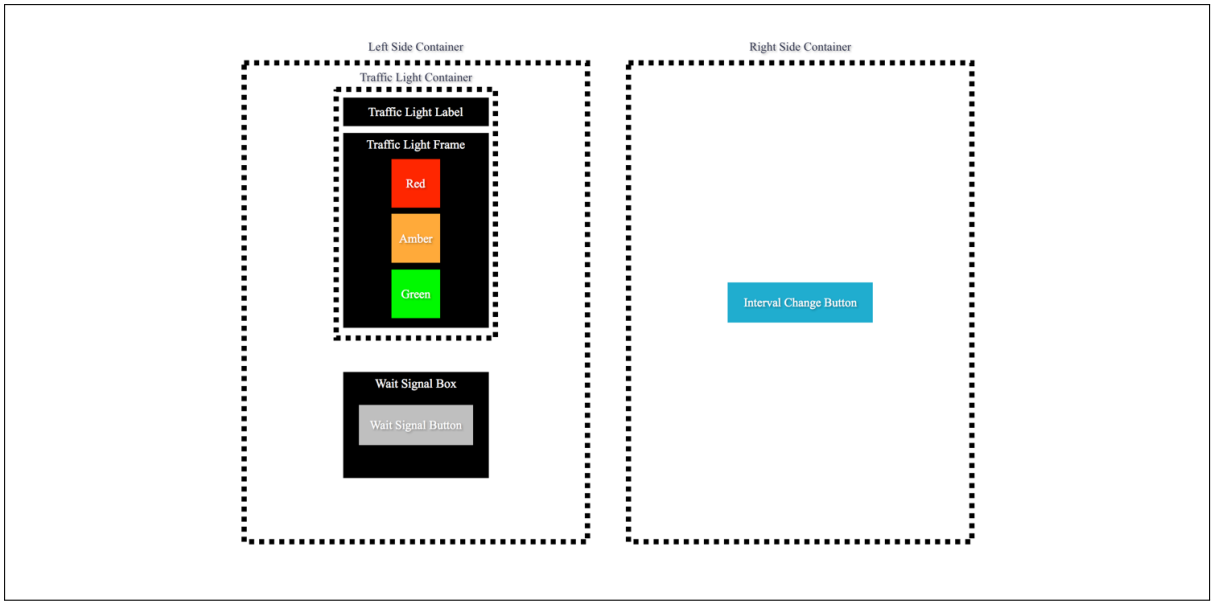
Asset Name	Asset Type	Asset Description	Asset Properties
Left side container	FlexBox Container	Layout container for left hand side	Invisible
Traffic light container	FlexBox Container	Layout container for traffic lights	Invisible
Traffic light frame	FlexBox Container	The frame for the traffic lights	Black with White border
Red traffic light	FlexBox Item	The red traffic light	Dimmed Red (#49180d), ID = "red"
Amber traffic light	FlexBox Item	The amber traffic light	Dimmed Amber (#8f7414), ID = "amber"
Green traffic light	FlexBox Item	The green traffic light	Dimmed Green (#0d4913), ID = "green"
Traffic Light Label	Button	A label for the traffic light, and to help FlexBox spacing	No OnClick event, grey
Wait signal box	FlexBox Container	The frame for the pedestrian wait button	Black
Wait signal button	Button	The button to stop the lights for a pedestrian	Black
Right side container	FlexBox Container	The container for the interval change button and text	Invisible
Interval change button	Button	Button to call the interval time change dialogue box	#1fadcf

3.2 (ii) Describe and explain where the assets will be best located.

For this task, as it is a small, single page website, all the scripts and styles will be embedded into the HTML document. The objects will be arranged in `<div>` structures, and items will be inside FlexBoxes.

FlexBoxes are a CSS Layout style, where flexible boxes contain items. This means that the objects are dynamically aligned, and no positioning is fixed or absolute. This makes it usable on mobile devices as well as a range of screen sizes. It is also adaptable, and easy to adjust. My research primerially came from the W3Schools page on FlexBoxes: http://www.w3schools.com/css/css3_flexbox.asp

The layout will be in two columns, with the traffic lights and wait button on the left, and the button to change the interval time on the right. The layout is shown in this image:



3.3 (iii) Describe the structure of an array that could be used to handle the traffic light sequence.

I will be using FlexBoxes for my traffic lights instead of images, so it would not be appropriate to store the FlexBox items in an array, as they are structured in a `<div>` arrangement. However, as I will be using JS style changes to change the colours of the traffic lights, it is easier to store these in an array. Here are the colours I will use:

Colour Number	Colour Name	Hex Value
0	Red On	#FF0000
1	Red Off	#49180D
2	Amber On	#FDA428
3	Amber Off	#8F7414
4	Green On	#00AF1B
5	Green Off	#0D4913

Put into an array, this would be:

```
var CustomColor = ["#ff0000", "#49180d", "#fda428", "#8f7414", "#00af1b", "#0d4913"]
```

For simplicity, the Red On and Amber On have been substituted for their HTML default colours, `red` and `orange`

```
202     var CustomColor = ["red", "#49180d", "orange", "#8f7414", "#00af1b", "#0d4913"]
```

3.4 (iv) Write a script that uses the array described in part 3.3 to produce an animation of a set of traffic lights such that the lights change in the standard sequence each time the button is clicked.

3.4.1 Traffic Light Sequence

Firstly, I worked out the sequence of the traffic lights. The sequence is as follows:

Iteration	Red	Amber	Green
0	On	Off	Off
1	On	On	Off
2	Off	Off	On
3	Off	On	Off

I then programmed this into the JS Functions:

```
4   function lightchange(){
5         if (lightstate == 0){
6           lightstate0();
7           lightstate = 1;
8         }
9         else if (lightstate == 1){
10          lightstate1();
11          lightstate = 2;
12        }
13        else if (lightstate == 2){
14          lightstate2()
15          lightstate = 3;
16        }
17        else if (lightstate == 3){
18          lightstate3()
19          lightstate = 0;
20        }
21      }
22      setTimeout(function(){lightchange()}, window.intervalTime)
23    function lightstate0(){
24      document.getElementById("red").style.backgroundColor = CustomColor[0];
25      document.getElementById("amber").style.backgroundColor = CustomColor[3];
26      document.getElementById("green").style.backgroundColor = CustomColor[5];
27    }
28    function lightstate1(){
29      document.getElementById("red").style.backgroundColor = CustomColor[0];
30      document.getElementById("amber").style.backgroundColor = CustomColor[2];
31      document.getElementById("green").style.backgroundColor = CustomColor[5];
32    }
33    function lightstate2(){
34      document.getElementById("red").style.backgroundColor = CustomColor[1];
35      document.getElementById("amber").style.backgroundColor = CustomColor[3];
36      document.getElementById("green").style.backgroundColor = CustomColor[4];
37    }
38    function lightstate3(){
39      document.getElementById("red").style.backgroundColor = CustomColor[1];
40      document.getElementById("amber").style.backgroundColor = CustomColor[2];
41      document.getElementById("green").style.backgroundColor = CustomColor[5];
42    }
43    lightchange();
```

In this code, Line 4 declares `lightchange()` as a function. Lines 5-8 are the if statement for changing the lights to Iteration 0, as the function `lightstate0()`. Lines 9-12 are the if statement for switching to Iteration 1 (`lightstate1()`), and lines 13-16 and 17-20 are for Iteration 2 (`lightstate2()`) and Iteration 3 (`lightstate3()`) respectively. Line 22 then calls `lightchange()` in a timeout loop, as specified by the global variable of `intervalTime`.

Lines 23-26 code for the lights to change to the Stop state (Iteration 0) when `lightstate0()` is called. Lines 28-32 code for the change to Iteration 1 for `lightstate1()`, lines 33-37 for Iteration 2 for `lightstate2()`, and lines 38-42 for the change to Iteration 3 with `lightstate3()`. These all use the `document.GetElementById` function, which addresses each element by their `id` tag as defined in the HTML.

3.4.2 HTML Layout

For the layout, I used `div` hierarchies to lay out the elements as seen in the layout plan above. `<div>` tags allow elements to be grouped and addressed in CSS. I researched this on the W3Schools website: https://www.w3schools.com/tags/tag_div.asp . This is the layout structure:

```
3   <div class="container">
4       <ul>
5         <button id="button" class="button" onclick="">Traffic Light</button>
6         <div class="panel">
7           <ul style="list-style-type:none">
8             <li id="red" class="flexItem red"></li>
9             <li id="amber" class="flexItem amber"></li>
10            <li id="green" class="flexItem green"></li>
11          </ul>
12        </div>
13        <br>
14        <div class="StopButton">
15          <p id="stopInfo" class="stopInfo">Push button <br> Wait for signal</p>
16          <button id="wait" class="wait" onclick="">...WAIT...</button><br>
17          <button id="stop" class="stop" onclick="stopLights()">Stop Button</button>
18        </div>
19      </ul>
20    </div>
21    <div class="inputContainer">
22      <ul>
23        <p class="buttonTitle">Click to change interval</p>
24        <button class="inputButton" onclick="inputPrompt()">Change Interval  (milliseconds)</button>
25      </ul>
26    </div>
```

The two containers, Left side container and Right side container, are opened in Line 3 and Line 21 with the classes "container" and "inputContainer" respectively.

Inside the Left side container, an unordered list (`<ul>`) to contain the Traffic Light Label (id = "button"), the `<div>` for the Traffic light frame (id = "panel") and the Wait signal box (id = "StopButton"). Inside the Traffic light frame, is another `<ul>` for the 3 lights, with id = "red" "amber" and "green", and classes of "flexItem". Inside the Wait signal box, is the paragraph text (id = "stopInfo") for the label, the wait light (id = "wait") and the wait button (id = "stop").

Inside the Right side container is the `<ul>` for the label (id = "buttonTitle") and the input button (id = "inputButton").

3.4.3 Styling and Layout

Task 3 only requires a button to change Iterations, so the automatic code, interval change, and wait button elements will be ignored until they are needed in Task 4. As FlexBox requires on CSS, the styling will also be embedded. Here is the standard, un-styled code for the traffic lights:

Body and #container

```
10      body {
11        display: -webkit-flex;
12        display: flex;
13        -webkit-flex-wrap: wrap;
14        flex-wrap: wrap;
15        -webkit-align-content: center;
16        align-content: center;
17        padding: 0;
18        margin: 0;
19        min-height: 100vH;
20      }
21      #container>ul{
22        list-style: none;
23        padding: 0px;
24      }
25      #container {
26        display: -webkit-flex;
27        display: flex;
28        -webkit-flex-wrap: wrap;
29        flex-wrap: wrap;
30        -webkit-align-content: center;
31        align-content: center;
32        margin: auto;
33      }
```

The body styles for the WebKit Flex Wrap in Lines 11-15, content is aligned centre in Line 16, and Lines 17-20 code for no padding or margins, and 100% width. #container>ul codes for the `<ul>` inside the container. It codes for no list style, and 100% width. The #container codes for the WebKit Flex Align and automatic margins and alignment.

#Panel, #FlexItem and #Button

```
#panel>ul{
  list-style: none;
  padding: 0px;
}
#panel {
  display: -webkit-flex;
  display: flex;
  -webkit-flex-wrap: wrap;
  flex-wrap: wrap;
  -webkit-align-content: center;
  align-content: center;
  margin: auto;
  background-color: lightgrey;
}
#flexItem {
background-color: red;
width: 100px;
height: 100px;
margin: 10px;
display: flex;
-webkit-flex-wrap: wrap;
flex-wrap: wrap;
-webkit-align-content: center;
align-content: center;
margin: 10px;
}
#button1 {
  background-color: gray;
  color: white;
  font-size: 40px;
  border-radius: 10px;
  border: 0;
  -webkit-flex-wrap: wrap;
  flex-wrap: wrap;
  -webkit-align-content: center;
  align-content: center;
  -webkit-transition-duration: 0.4s;
  transition-duration: 0.4s;
  cursor: pointer;
  border-style: solid;
  margin-bottom: 8px;
}
#button1:hover{
  background-color: #009100;
}
```

4 Task 4

5 Task 5

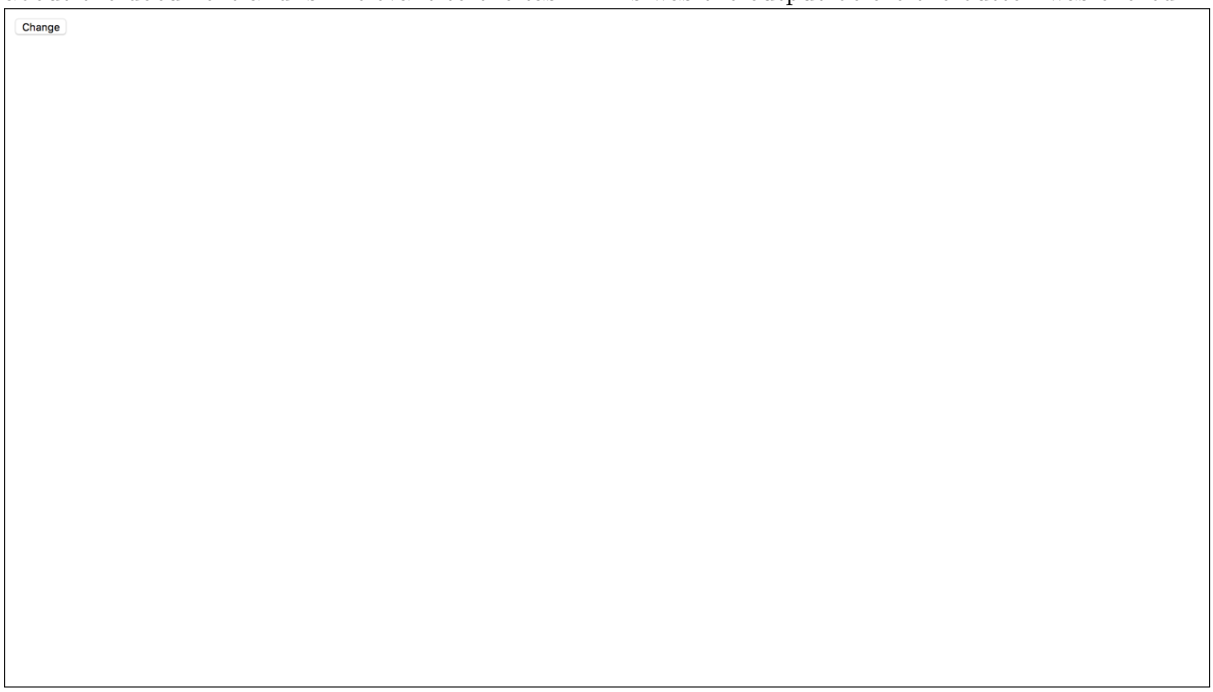
Scripts can be embedded in the HTML of web pages or saved externally as script files. Discuss the benefits and drawbacks of each approach.

5.1 Embedded JS Scripts

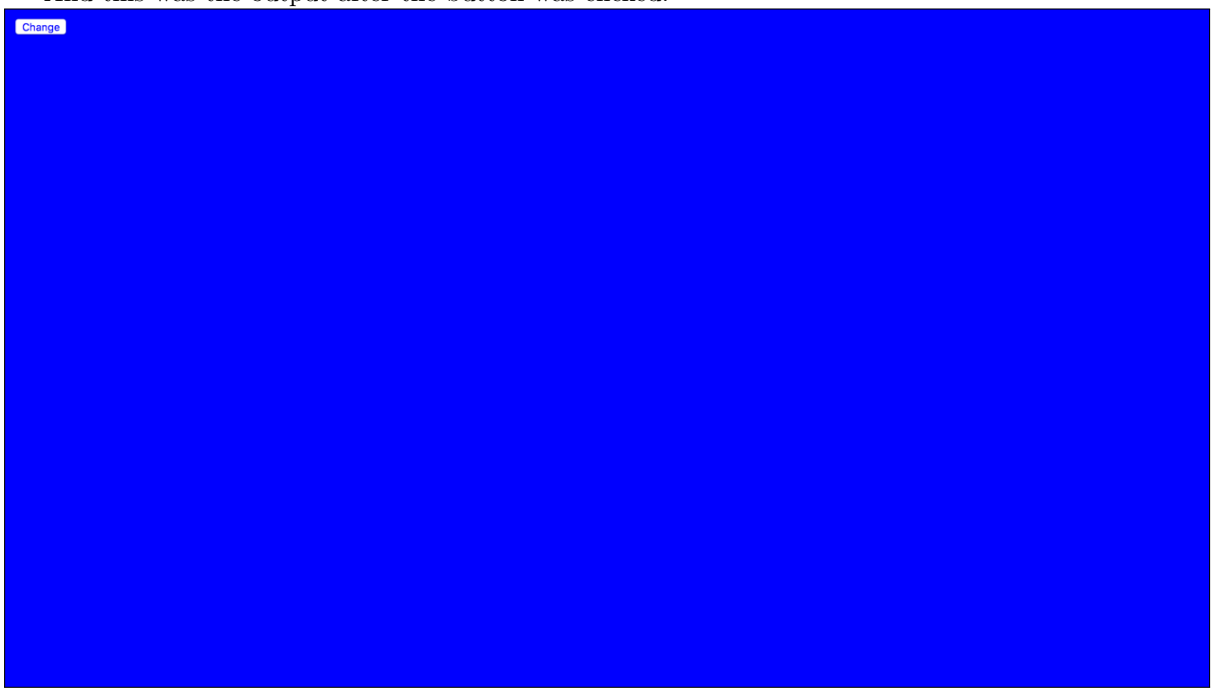
JavaScript functions can easily be embedded into an HTML document, like I've done in tasks 3 and 4. In HTML, the `<script>` tag opens the area for JS code, but must be closed afterwards. For example, a button can have the `onclick()` event for `change()`, which will call the `change()` function in the JavaScript area. For a demo, I used jsbin.com, a website where you can write and run HTML, JS and CSS code embedded or in a separate area. I entered the following demo code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <button id="button" onclick="change()">Change</button>
  <script>
    function change(){
      document.getElementById("button").style.color = "#0000ff";
      document.body.style.background = "#0000ff"
    }
  </script>
</body>
</html>
```

On the click of the button with id of `button`, the `change()` function will be called from the `<script>` area. The function changes the color of the element with ID of `button` to blue and changes the body background to blue. The first lines inside the `<head>` tag and with the `<meta>` tags is just information about the document and is irrelevant to the task. This was the output before the button was clicked:



And this was the output after the button was clicked:



This clearly show that the embedded script can easily change the style of HTML elements. This could be adapted to change position, size, and any other attribute.

5.2 External JS Scripts

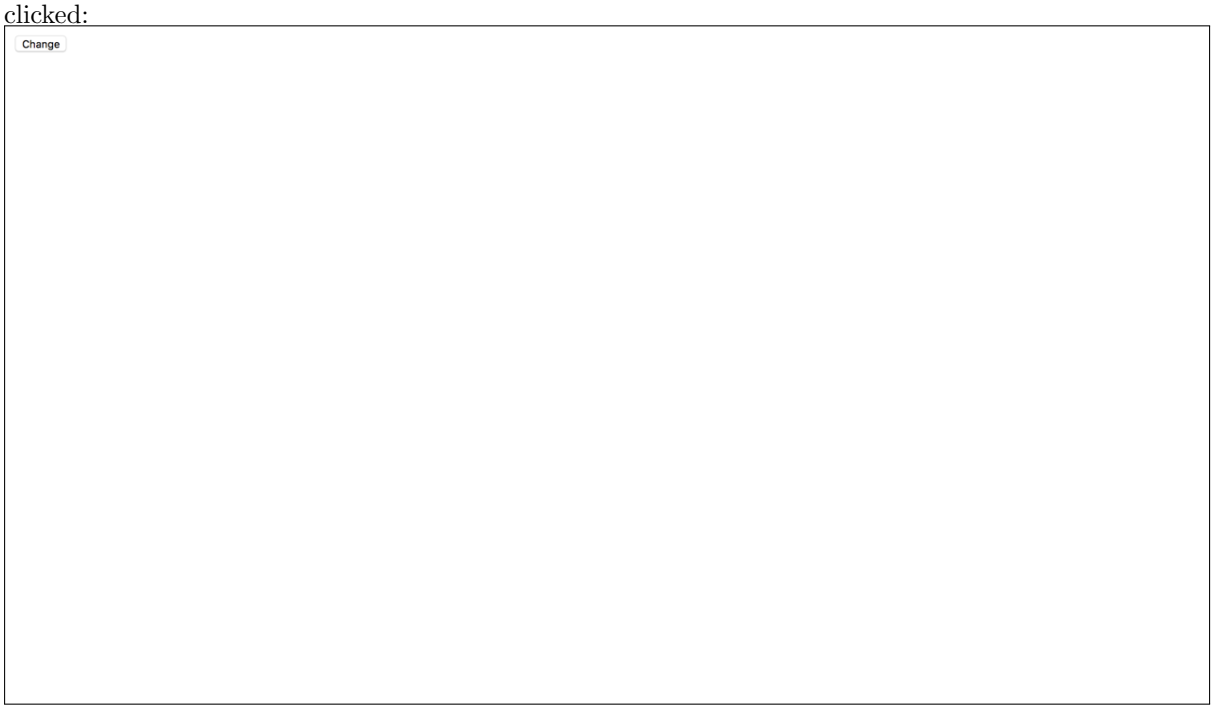
The same effect as in 3.1 can be made with externally saved scripts. Alongside the HTML file (normally named `index.html`) the JS file can be saved (normally named `script.js`. In this scenario, the CSS styling is also saved externally, rather than embedded as the `<style>` tag, normally as `style.css`. I adapted the demo code in 3.1 to show this.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <button id="button" onclick="change()">Change</button>
</body>
</html>
```

This is the HTML code, which is the same as before in 3.1, but with the `<script>` code removed. This document is now entirely HTML. The JS was moved to the external file:

```
function change(){
      document.getElementById("button").style.color = "#0000ff";
      document.body.style.background = "#0000ff"
    }
```

This shows the `change()` function moved to the external file. This program will operate in the exact same way, as the files are both saved in the same location. Here is the output before the button was clicked:



And this is the output after the button was clicked:

As you can see, this produces the exact same results. So what are the benefits and drawbacks of each approach?

The embedded script approach is more suitable to small projects, such as a personal portfolio or small website. Having it all saved to the same file can make it a lot easier to manage, as well as preserving consistency and ensuring no failures. However, for larger tasks like a web app or large website, external scripts are often more applicable.

With an external script, pre-made code can be referenced from the HTML document and used elsewhere. This limits the size of the HTML document and makes it a lot tidier and easier to have consistent indentation. Websites like www.dynamicdrive.com offer scripts for both JS and CSS that can be embedded into the code easily. These off-the-shelf snippets are useful in small-scale webpages and can be embedded or saved externally, but are normally embedded.

I prefer to use embedded scripts, as I often code small-scale projects and it is easier to have it all in the same document. However, if a project gets to large, I will move my JS and CSS to an external file for consistency, and to reduce the time it takes for a page to load.