Computing GCSE Coursework

Thomas Bass
Candidate 4869
Centre 52423
OCR A452 Practical Investication

Word Count: 915

Made with LaTeX

2017

Summary

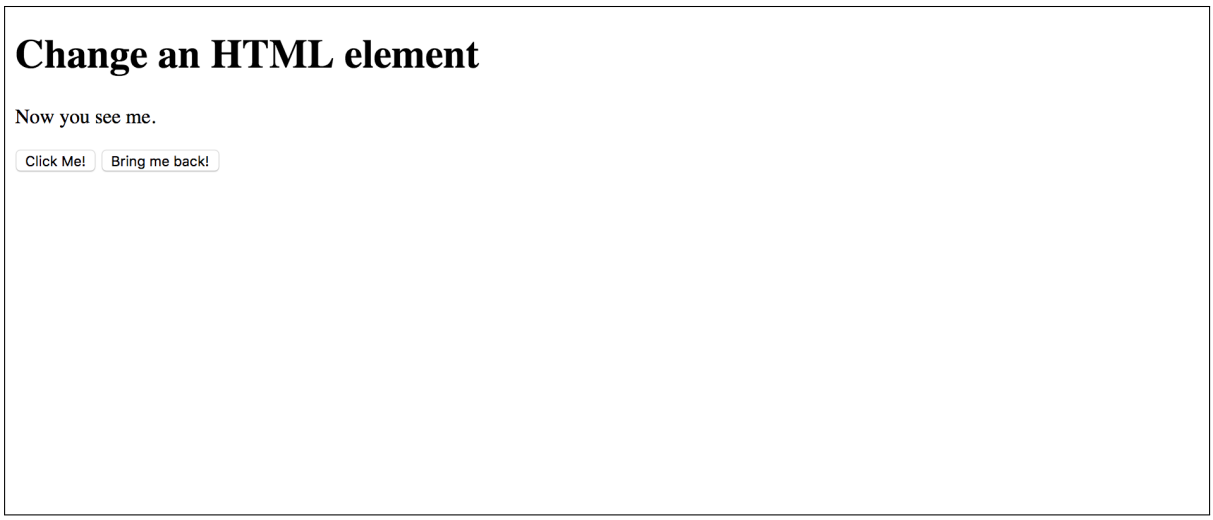
1 Task 1

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 5

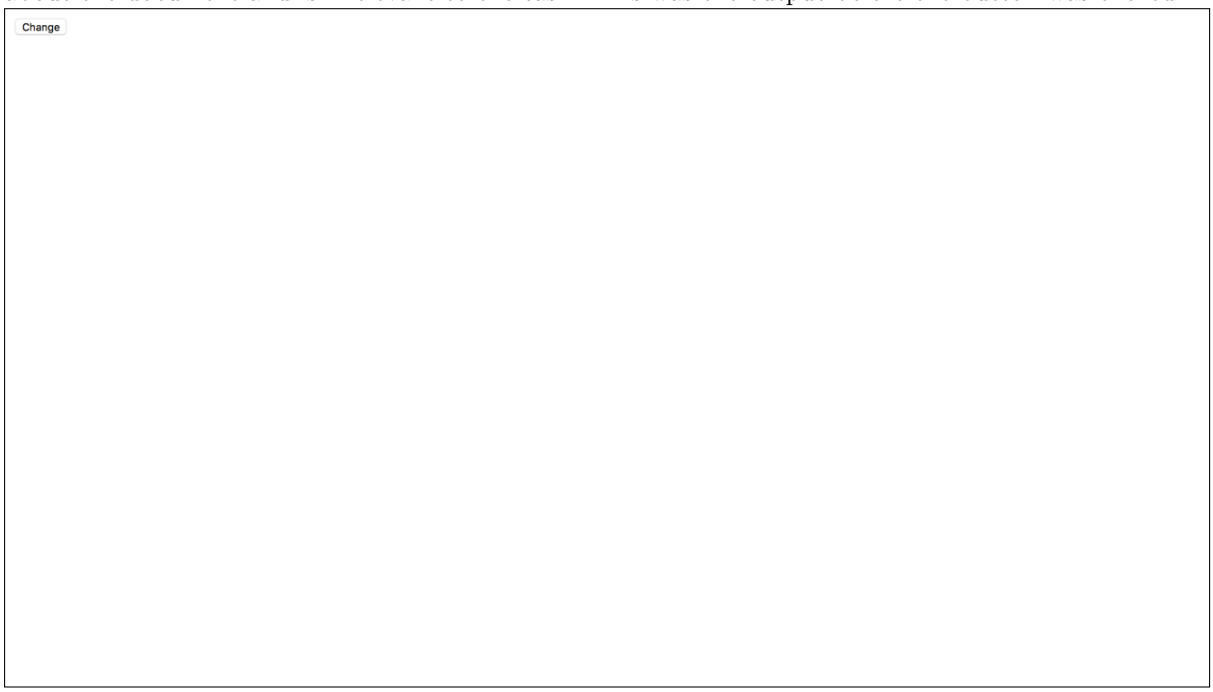
Scripts can be embedded in the HTML of web pages or saved externally as script files. Discuss the benefits and drawbacks of each approach.

3.1 Embedded JS Scripts

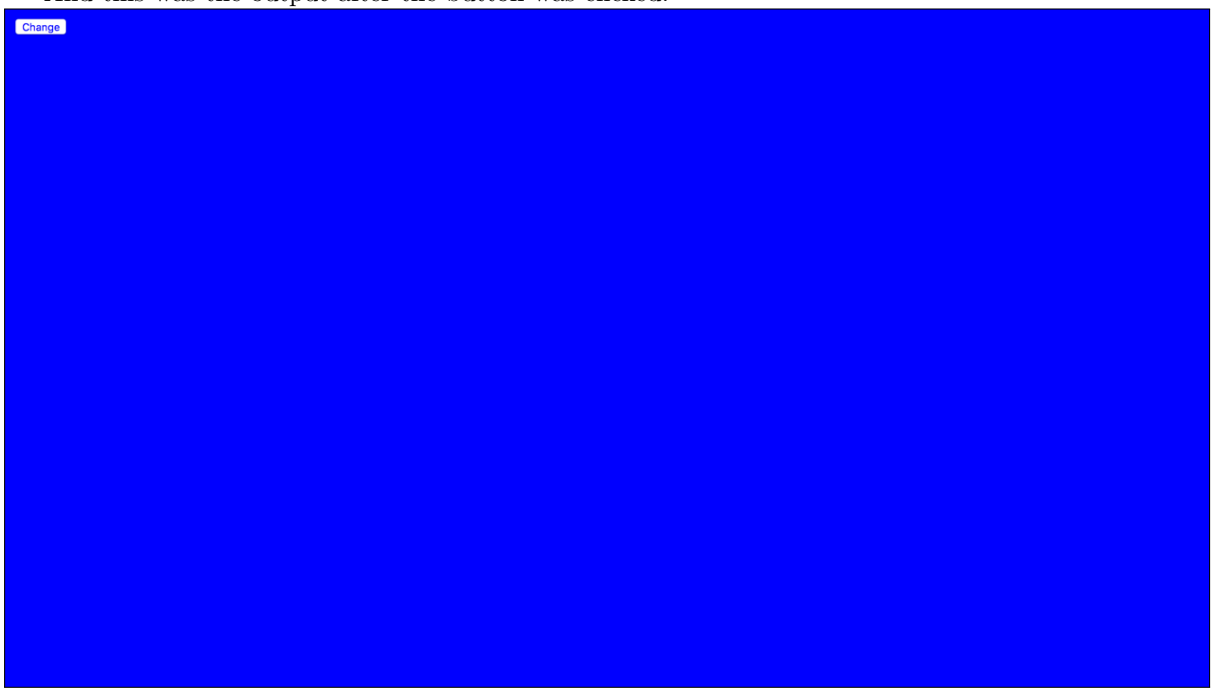
JavaScript functions can easily be embedded into an HTML document, like I've done in tasks 3 and 4. In HTML, the `<script>` tag opens the area for JS code, but must be closed afterwards. For example, a button can have the `onclick()` event for `change()`, which will call the `change()` function in the 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.

3.2 External JS Scripts

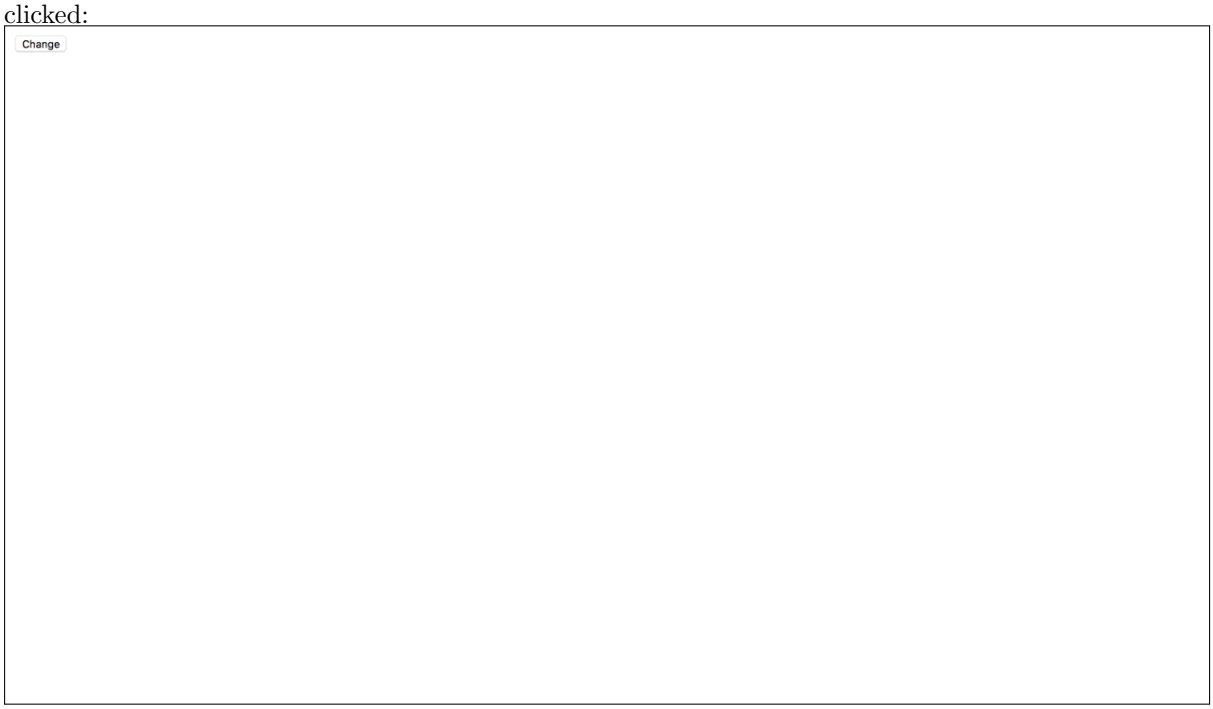
The same effect as in 3.1 can be made with externally saved scripts. Alongside the HTML file (normally named `index.html`) the JS file can be saved (normally named `script.js`. In this scenario, the CSS styling is also saved externally, rather than embedded as the `<style>` tag, normally as `style.css`.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