Getting Started With JavaScript

Setup

- In your websites folder, create a new folder called *test-js* or similar.
- In Visual Studio Code, open the *test-js* folder (`File > Open Folder`)
- Create a new HTML file in this folder called *index.html*
- Create a new CSS file and link it to *index.html*. The CSS file can be at the root of the project, or in a sub-folder, the choice is yours.
- Add the following HTML code to the `body` of *index.html*. (Note the *id* attributes on each element. These are similar to *class* attributes, but the value of each must be unique on the page.)

```
<h1 id="page-heading">This is a static page</h1>
<button id="button1">Click Me!</button>
```

- Add the following CSS to your stylesheet:

```
body {
    font-family: arial, helvetica, sans-serif;
}
h1 {
    background-color: #002b36;
    color: #93a1a1;
    padding: 1em;
}
button {
    display: inline-block;
    margin: 0.5em 0;
    padding: 0.5em 1em;
    font-size: 1em;
    cursor: pointer;
    box-shadow: none;
    border-radius: 0;
    border-width: 0;
    text-decoration: none;
    line-height: 1.2;
    background-color: #268bd2;
    color: #fff;
}
```

- Preview your page in your browser to make sure everything is linked up correctly.

Adding JavaScript to the Page

In *index.html*, just before the closing *body* tag, add a script tag:

```
        <!-- Heading and button here... -->

        <script>
            // Add your JavaScript code here
        </script>
    </body>
</html>
```

When the browser is *parsing* the page, it will load and execute the code found within this tag.

To see this in action, add this JavaScript code **inside** the *script* tag:

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

In your browser, reload the page and you should see a dialog box displayed.

A Simple Event Listener

Now we will do something a bit more interesting so delete the code you added to the *script* tag earlier (but leave the script tags in place!)

Add the following code to the *script* tag, including the comments (lines starting with //). The comments describe what the lines following them are doing.

```
// 1. Get a reference to the button element
let myButton = document.getElementById('button1');

// 2. Define some behaviour that should happen when the button is clicked
function handleClick() {
    alert('The button has been clicked!');
}

// 3. Link the behaviour defined in handleClick to "clicks" on the button
myButton.addEventListener('click', handleClick);
```

Reload the page in your browser. When you click the button, a dialog should be displayed.

Code breakdown:

1. This is a *variable assignment*. Variables are places we can store things within our code. Here, we are storing a *reference* to the button element from our HTML in a variable named *myButton*. For JavaScript to modify or interact with an element on the page, it needs a *reference* to it. `document.getElementById` is one way that we can look up elements in the page and get a reference to them.
2. This is a *function definition*. Essentially, a piece of code that can be run on demand. In this case, the function simply opens an alert dialog.
3. This is an *event* handler. Much of our JavaScript runs in response to *events* which occur in the browser. E.g. Clicking things, scrolling, typing, hovering, etc.

Some DOM Manipulation

The *DOM* is a model of the HTML document, in a format that JavaScript and other technologies can work with.

Manipulating the DOM is all about adding, removing or modifying elements within the HTML document.

Change the *handleClick* function so it looks like this:

```
function handleClick() {
    // 2.1 Get a reference to the h1 element
    let myHeading = document.getElementById('page-heading');
    // 2.2 Change the text within the h1 element
    myHeading.textContent = 'Now I am Dynamic!!!';
}
```

Reload the page in your browser. When you click the button, the text in the *h1* should change.

- The first line of code (2.1) gets a reference to the *h1* element and stores it in a variable, just like we saw earlier with the button. We want to modify the *h1*, so we need a reference to it.
- The second line of code (2.2) assigns a new value to the element's *textContent* property. Once we change this property via JavaScript, the browser updates the screen accordingly.

In addition to manipulating the elements on the page, we can also manipulate their styles.

Update your *handleClick* function to look like this:

```
function handleClick() {
    // 2.1 Get a reference to the h1 element
    let myHeading = document.getElementById('page-heading');
    // 2.2 Change the text within the h1 element
    myHeading.textContent = 'Now I am Dynamic!!!';
    // 2.3 Change the element's background colour
    myHeading.style.backgroundColor = '#586e75';
}
```

Preview the page in your browser and click the button.

Code organisation

Our JavaScript programs can grow to be very large and complex. Writing all of the JavaScript in a script tag within the HTML is not sustainable (and it makes it impossible to re-use the code).

Instead, the JavaScript code is normally stored in an external file, which is linked to the HTML page (similar to how we store our CSS).

Inside the *test-js* folder, create a new folder called *scripts*. Inside this folder, create a new file called *main.js*.

Copy all of the code from within the script tags in *index.html* (not including the script tags) and paste it into *main.js*.

In *index.html*, replace the *script* tag with this:

```
<script src="scripts/main.js"></script>
```

Save your files and reload the page in your browser. Is it still working?

If not, check that the folder name and file name used in the *src* attribute match the actual file and folder names.

Debugging

All modern browsers come with built-in developer tools, that allow us investigate and debug our JavaScript code (and our CSS). We will be using the *console* element of the developer tools extensively.

To activate the console in the browser:

- Chrome: `Ctrl+Shift+J` (Windows), `Command+Option+J` (Mac)
- Firefox: `Ctrl+Shift+K` (Windows), `Command+Option+K` (Mac)

A new panel will be displayed either below or beside the page, depending on your browser. *NB: You can change the position of the panel via its toolbar*

The *console*:

- Lists JavaScript errors that have occurred on the page
- Displays debug messages that have been set in the code
- Run arbitrary snippets of JavaScript code and see the results
- And lots more...

Debug messages

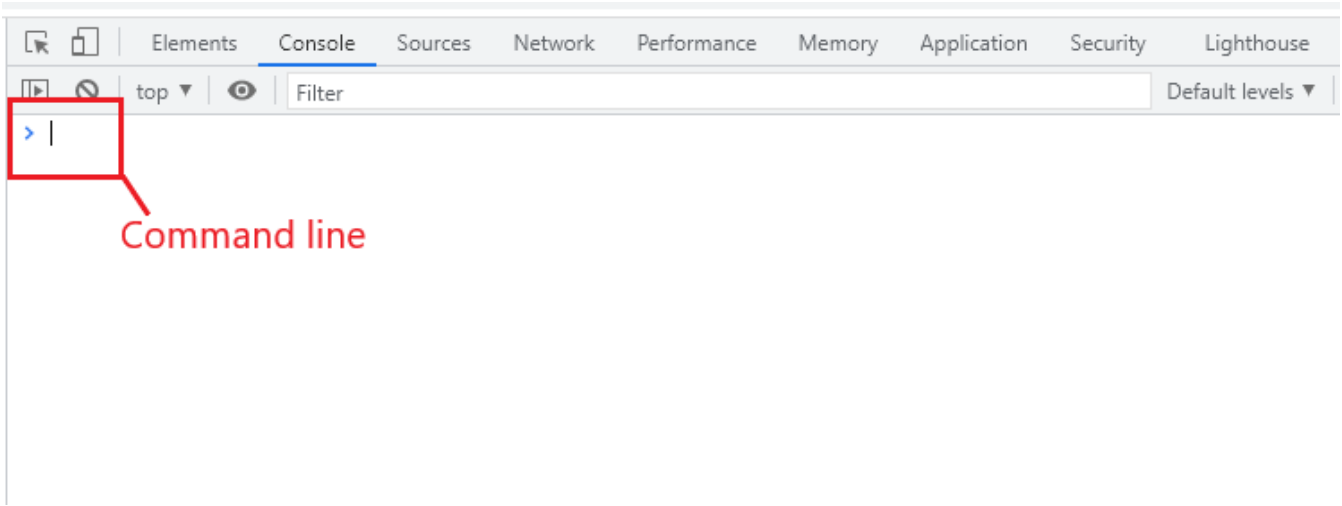
Open *main.js* and add some `console.log()` statements to the *handleClick* function:

```
function handleClick() {
    console.log('The button has been clicked'); // add this
    let myHeading = document.getElementById('page-heading');
    console.log('Before:', myHeading.textContent); // add this
    myHeading.textContent = 'Now I am Dynamic!!!';
    console.log('After:', myHeading.textContent); // add this
    myHeading.style.backgroundColor = '#586e75';
}
```

Reload the page in your browser. Open the *console* and then click the button. You should see three messages displayed.

Running JavaScript Code

The console also has a *command line* where we can enter JavaScript code directly and it will be run within the context of the page.



Try typing `2 + 2` on the command line and then press enter/return. You should see `4` displayed.

Now try typing this, pressing enter/return when done:

```
document.getElementById('button1').textContent
```

You should see *Click Me!* displayed.

More about the developer tools

- Firefox: https://developer.mozilla.org/en-US/docs/Tools
- Chrome: https://developer.chrome.com/docs/devtools/overview/