Session 8

Recap

In the previous session we covered the following:

- Testing: Katas

DOM Manipulation

Before we deep dive into more full blown applications, it's important to understand the DOM (Document Object Model) and how JavaScript can be used to manipulate it (by this I mean accessing it, changing element content, moving items around etc).

As mentioned in the early sessions, there is a keyword, pre-existing object, the `document` that is available to you when you write JavaScript.

This can be something as simple as creating a `script` tag and simply adding lines as such:

```
<script type="text/javascript">
    console.dir(document); // this will show us everything that is contained inside the
'document'
</script>
```

The DOM can be imagines as a tree with a sequence of nodes that hang off it:


dom-tree

If you open the Console window in DevTools, you'll see when you hover over the logged message, it highlights the whole screen. The `document` represents the whole web page.

If we say add a `div` to the HTML page:

```
<div id="content">
</div>
```

We now have an `element`, a div `element` that we can now try to target using it's ID attribute. To target the `div` using the `document` object, we can use the built-in `getElementById` method which looks for an returns an element that has the corresponding ID:

```
<script type="text/javascript">
    console.log(document.getElementById("content"));
</script>
```

We can alternatively use the `querySelectorAll` to select elements that match a given criteria, such as find all 'divs':

```
<script type="text/javascript">
    const allDivs = document.querySelectorAll("div");
    console.log(allDivs);
</script>
```

If we had say a div that had a class, we can use the `querySelector` function:

```
<div class="container">
    Hello
</div>

<script type="text/javascript">
    const divWithClass = document.querySelector(".container");
</script>
```

The majority of elements on a HTML page have a set of pre-defined events that can be triggered, for example, let's take an input field:

```
<input type="text" name="username" class="formField" />
```

This input field can have the following events (not all are listed as you can find them on the MDN site):

- onclick
- oninput
- onblur
- onchange
- etc...

```
    <head>
        <script type="text/javascript" defer>
            const echoOutput = (event) => {
                echoBlock.textContent = event.target.value;
            }

            const input = document.querySelector('input'); // fetch the only input on
the page
            const echoBlock = document.getElementById('echo');

            input.addEventListener('input', echoOutput);
        </script>
    </head>
<body>
    <input type="text" placeholder="Enter text ..." name="name" />
    <p id="echo"></p>
</body>
```

When we add an event listener, we omit the `on` from the word as the first argument. The second argument is a function that we want to call whenever this event occurs. Here the `echoOutput` function is called each time there is a value entered into the input field.

You will have noticed an `event` parameter is provided to the `echoOutput` function. This argument is always provided as an argument whenever an event is triggered.

Here the `event` is used to retrieve the current value from the element that has triggered this event, referred to as the as `target`.

Try out the above snippet and see for yourself!

What if we wanted to handle events based on a button click? Using our previous example (or tweaking it should I say):

```
<input type="text" placeholder="Enter text ..." name="name" />
<button>Click me to echo</button>

<p id="echo"></p>

<script type="text/javascript">
    const echoOutput = (event) => {
        if (input.value === '') {
            alert('Please enter a value');
        }
        else {
            echoBlock.textContent = input.value;
        }
    }

    const btn = document.querySelector('button');
    const input = document.querySelector('input');
    const echoBlock = document.getElementById('echo');

    btn.addEventListener('click', echoOutput);

</script>
```

We can also manipulate how the web page looks in terms of styling (for the scope of this bootcamp I won't be delving too much into CSS).

To add say a class to an element, we can do this:

```
<head>

    <style>
        .error {
            border: 1px solid red;
        }

        .text {
            color: white;
            background-color: black;
            padding: 6px;
        }
    </style>
```

```
</head>

<body>
<input type="text" placeholder="Enter text ..." name="name" />
<button>Click me to echo</button>

<p id="echo"></p>

<script type="text/javascript">
    const echoOutput = (event) => {
        if (input.value === '') {
            input.classList.add('error');
            alert('Please enter a value');
        }
        else {
            input.classList.remove('error');
            echoBlock.textContent = input.value;

            echoBlock.classList.add('text');
        }
    }

    const btn = document.querySelector('button');
    const input = document.querySelector('input');
    const echoBlock = document.getElementById('echo');

    btn.addEventListener('click', echoOutput);

</script>
</body>
```

Now to get more comfortable with manipulating the DOM, we'll tackle some challenges.

Challenge: Rock Paper Scissors

Convert your rock, paper, scissors game to use images and click events. Also display the score of who won and increment it on each new game.

Challenge: Speed Typing

In this challenge we'll be making a speed typing game that asks the user to type a random word within a time period to gain points.