JavaScript Introduction

JavaScript is a *client-side* scripting language. This means that it can alter webpage content without the web-browser having to reload or direct elsewhere.

If you wished to create a browser-based platformer game -- something like Super Mario Bros -- JavaScript could be a good option. By downloading all of the logic and assets to the visitor's computer, the game can then utilise the client's hardware. In this way, each frame is rendered without having to fetch any new information from the web server, making for a smoother gaming experience.

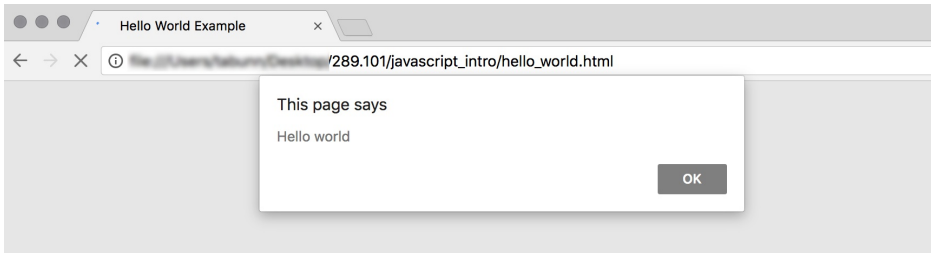
To store something like a world-wide high scores table, one would rely on a *server-side* language (like PHP, Python, or Ruby). You may use client- and server-side languages in combination with one another; popular techniques include AJAX and REST.

Hello World

Create a new directory in your 289.101 repository named "javascript_intro". Within this, create a "hello_world.html" file and add the following code to it:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Hello World Example</title>
</head>
<body>

  <h1>My First JavaScript File</h1>

  <script>
    alert('Hello world');
  </script>

</body>
</html>
```

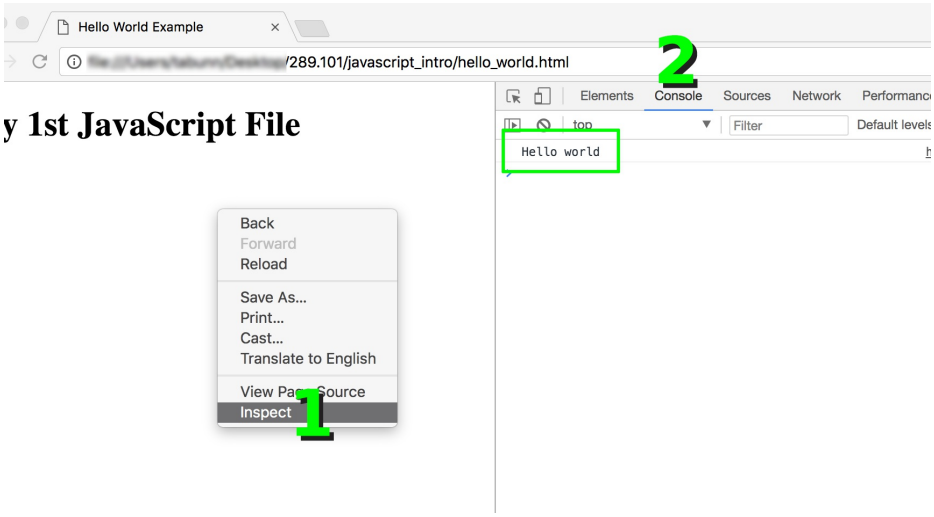
Test this in your web browser. The above code will trigger an alert (displaying "Hello World") to pop-up over the webpage.



Rather than using an alert, change the script so that it outputs to the console instead:

```
...

<script>
  console.log('Hello world');
</script>

...
```

If you save and refresh your browser, there will be no alert displaying the text. Instead, you will need to open your browser's developer tools to view the console. In Chrome, you can right-click anywhere on the webpage and select "Inspect" (or "Inspect Element" for Firefox). This will reveal the developer tool panel from which you can select the "Console" tab. It's best to leave this panel open for the rest of this tutorial.

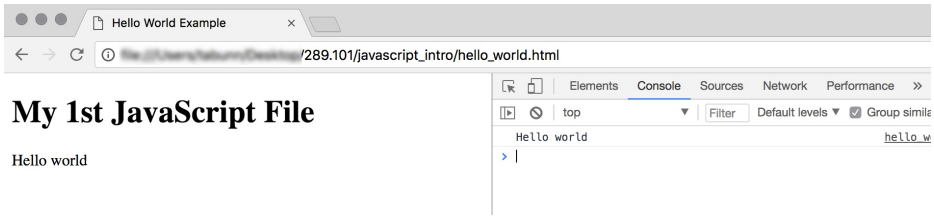


The `console.log()` method is great for debugging purposes as you can hide your messages from general users (who would never look at the developer panel). Most web browsers include some form of developer tools similar to these.

Now try the `document.write()` method, which writes the output directly into the HTML:

```
...

<script>
```

```
  console.log('Hello world');
  document.write('Hello world');
</script>

...
```



As you are, effectively, inserting HTML code here, you may include some formatting:

```
...

<script>
  console.log('Hello world');
  document.write('<h1>Hello World</h1>');
</script>

...
```

You can include as many script tags as you like and place them anywhere in the `head` or `body` elements of your document. However, as you will soon find out, some placements are better than others.

Comments

You can comment using two methods in JavaScript -- multi-line and single-line. Try an example of each in your code:

```
...

<script>

  /*
  all of this is ignored
  console.log('Hello world');
  */

  // display Hello world in large bold text
  document.write('<h1>Hello World</h1>');
```

```
</script>

...
```

Test this, noting how the console output has disappeared.

Variables

Variables are placeholders for information. Kind of like when you use letters in algebra to represent a value. In fact, variables in JavaScript look quite similar. Try this example:

```
  ...

  x = 1

  console.log(x)     // displays 1

  console.log(x + 2) // displays 3

</script>

...
```

While the above code works fine, you should add a few features: namely, the `var` keyword in front of the variable declaration; and a semi-colon after each statement. Amend your existing code, as this is the more 'correct' way of doing things:

```
  ...

  var x = 1;

  console.log(x);     // displays 1

  console.log(x + 2); // displays 3

</script>

...
```

When naming variables, however, you are not limited to a single letter. You may use any combination of alpha-numeric characters, provided the name does not contain any spaces or begin with a number. The following are all valid variable names:

- `var x = 1;`
- `var car = 1;`
- `var cars = 1;`
- `var redcars = 1;`
- `var red_cars = 1;`
- `var redCars = 1;`
- `var redCars1 = 1;`

Operators

You have just seen how JavaScript can carry out numeric addition operations. It can also 'add together' text (string data) by means of *concatenation* (chaining together). Try out the following code:

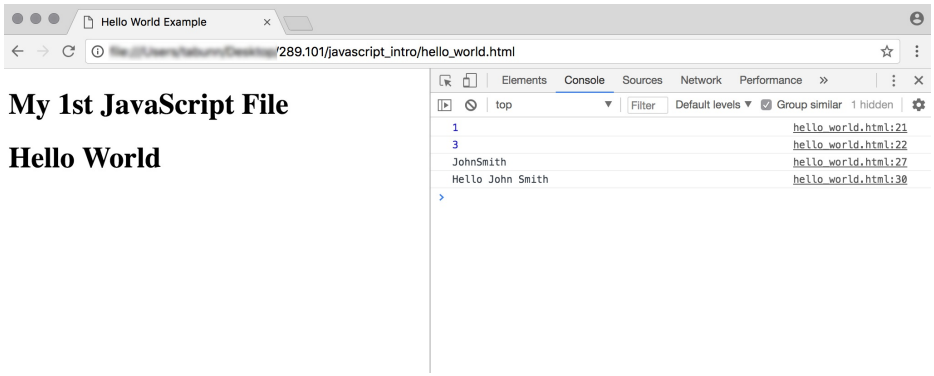
```
    ...

    var surname = "Smith";
    var firstname = 'John';

    console.log(firstname + surname); // displays JohnSmith

  </script>

  ...
```

Note that strings appear in quotation marks (and that you can use double- or singe-quotes).

To add space characters to the console output, you must explicitly include them:

```
    ...

    console.log(firstname + surname); // displays JohnSmith

    // displays: Hello John Smith
    console.log('Hello ' + firstname + ' ' + surname);

  </script>
```

```
...
```



The addition operator is the only operator that works with text; the others are exclusively for numeric operations. Try out subtract (`-`), divide (`/`), and multiply (`*`):

```
 ...

  console.log(10 - 2); // displays 8
  console.log(10 / 2); // displays 5
  console.log(10 * 2); // displays 20

</script>

...
```

The `=` sign is also a type of operator -- namely an *assignment* operator. The assignment operator can also combine with arithmetic operators to save you time. For example:

```
  ...

  var x = 1;
  console.log(x); // displays 1

  x = x + 2;
  console.log(x); // displays 3
</script>

...
```

-- can be rewritten as:

```
    ...

    var x = 1;
    console.log(x); // displays 1

    x += 2;
    console.log(x); // displays 3
  </script>

  ...
```

If you wish to in/decrement by one, use ++ or `--`:

```
    ...

    var x = 1;
    console.log(x); // displays 1

    x += 2;
    console.log(x); // displays 3

    x ++;
    console.log(x); // displays 4

  </script>

  ...
```

Order of operations is as you'd expect, with brackets overriding everything else (remember BEDMAS?):

```
    ...

    console.log(1 + 2 * 3);   // displays 7
    console.log((1 + 2) * 3); // displays 9

  </script>

  ...
```

The HTML DOM

Up until this point we have primarily dealt with console output -- which is not very useful for our website visitors. To get JavaScript interacting with the elements of your page, you will want to harness the *Document Object Model* (DOM).

You can think of HTML as a tree-like structure comprised of a series of child and parent elements. You can address and manipulate your webpage by controlling these various elements via the DOM.

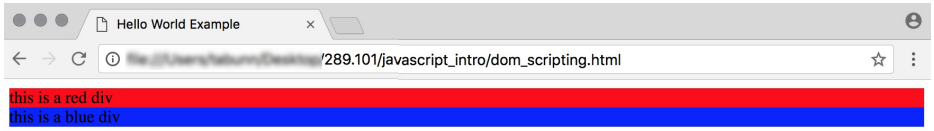
Create a new HTML document alongside your hello_world file, named "dom_scripting.html". Add the following code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>DOM Scripting Example</title>
</head>
<body>

  <div id="red">
    this is a red div
  </div>

</body>
</html>
```

The div has been assigned an id attribute equal to `red`. Recall that, unlike class attributes, `id`'s are restricted to a single instance per page. Also, they are addressed using a `#` (rather than a `.`) in CSS. Add some internal CSS, and another `div`:

```
  ...

  <style>
    #red {
      background-color: red;
    }

    #blue {
      background-color: blue;
    }
  </style>

</head>
<body>
```

```
  <div id="red">
    this is a red div
  </div>

  <div id="blue">
    this is a blue div
  </div>

...
```



document.getElementById()

As id attributes appear just once in a webpage (but not necessarily an entire website), they can be used as unique identifiers to reference elements in the DOM. To begin, here is some script that **doesn't work**:

```
...
<body>

  <script>
    var reddiv = document.getElementById('red');
    reddiv.style.backgroundColor = 'green';
  </script>

  <div id="red">
    this is a red div
  </div>

  <div id="blue">
    this is a blue div
  </div>

...
```

In the above code, the JavaScript is attempting to change the background colour of the `"red"` div to green. However, this **does not work because the script is placed above the div element** it is trying to reference. To fix this problem, you can place the script just before the closing `body` tag, thus ensuring the relevant HTML is loaded *before* the script that addresses it:

```
...
<body>

  <div id="red">
    this is a red div
  </div>

  <div id="blue">
    this is a blue div
  </div>

  <script>
    var reddiv = document.getElementById('red');
    reddiv.style.backgroundColor = 'green';
  </script>

</body>
</html>
```

The first `div` element now appears green.

You can refine things further and abbreviate the script, as the variable line is not entirely necessary:

```
  ...

  <script>
    document.getElementById('red').style.backgroundColor = 'green';
  </script>

</body>
</html>
```

The *style* object allows one to manipulate the CSS properties with which you are already familiar -- although in JS, the properties use lower-camelCase rather than hyphens, i.e. in JavaScript, it's `marginTop` as opposed to the CSS `margin-top`. You can find a list of these `style` properties here:

https://developer.mozilla.org/docs/Web/CSS/CSS_Properties_Reference

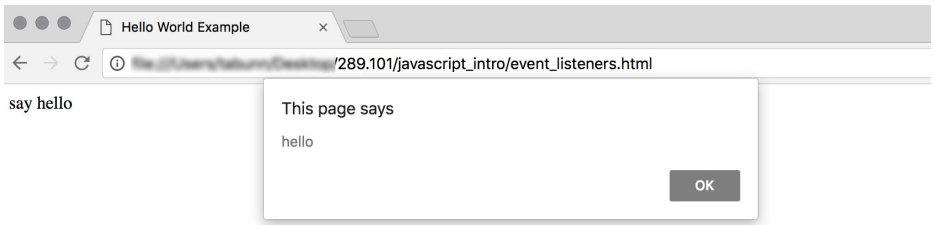
Event Listeners

Specific events trigger event listeners. For example a click; or a rollover; a specific key input; etc.

Create a new HTML document named "event_listeners.html"; adding the following code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Event Listeners Example</title>
</head>
<body>

 <div id="hello_btn">
    say hello
  </div>

  <script>
    document.getElementById('hello_btn').addEventListener('click', () => {
      alert('hello');
    });
  </script>

</body>
</html>
```

Test this by clicking the "say hello" text:

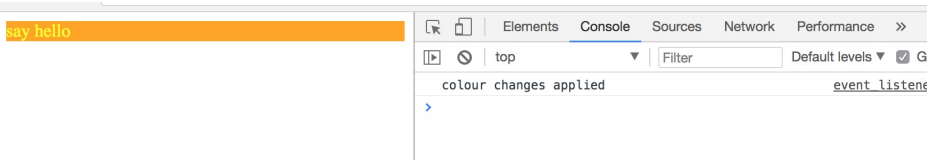


You are not limited to single actions for every listener. Add a few more lines to change the appearance of your button:

```
...

<div id="hello_btn">
  say hello
</div>

<script>
  document.getElementById('hello_btn').addEventListener('click', () => {
```

```
    document.getElementById('hello_btn').style.color = 'yellow';
    document.getElementById('hello_btn').style.backgroundColor = 'orange';
    console.log('colour changes applied');
  });
</script>

...
```

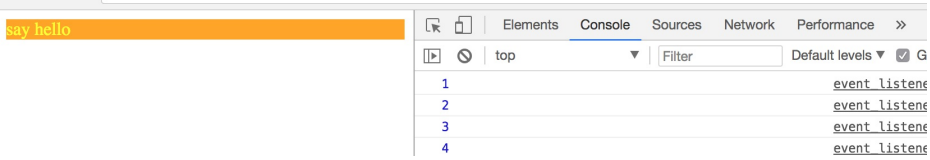
When you click "say hello" you'll see the following result:



Now adjust the script so that a unique number is logged with each click:

```
...

<script>
  var x = 1;

  document.getElementById('hello_btn').addEventListener('click', () => {
    document.getElementById('hello_btn').style.color = 'yellow';
    document.getElementById('hello_btn').style.backgroundColor = 'orange';
    console.log(x);
    x ++;
  });
</script>

...
```

Each time you click "say hello" the number logged is incremented by one:



Identikit Task

Grab the lesson files from Stream; extract them; then move the "identikit" folder into to your working repository. You need to complete the identikit task by making all of the buttons operational. Firstly, though, you'll need a quick introduction to CSS positioning.

CSS Positioning

Your lecturer/tutor will begin with a quick overview of CSS positioning. Just sit back and as watch he/she moves some colourful div's about using the code below:

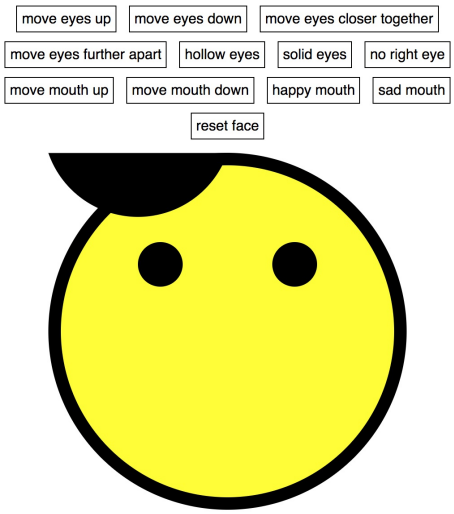
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Absolute Positioning Example</title>

  <style>
      body { margin: 0 }

      #red {
        background-color: red;
        height: 250px;
        width: 250px;
        position: relative; /* necessary to position child elements */
      }
      #yellow {
        background-color: yellow;
        height: 80px;
        width: 80px;
        position: absolute;
        left: 50px;
        top: 20px;
        z-index: 1; /* yellow div to render above purple */
      }
      #purple {
        background-color: purple;
        height: 30px;
        width: 190px;
        position: absolute;
        left: 10px;
        top: 50px;
      }
      #blue {
        background-color: blue;
        height: 50px;
```

```
        width: 50px;
        position: fixed; /* positioned relative to the viewport */
        right: 50px;
        bottom: 10px;
      }
  </style>

</head>
<body>

  <div id="red">
    <div id="yellow"></div>
    <div id="purple"></div>
  </div>
  <div id="blue"></div>

</body>
</html>
```

If you miss the lesson or need more information on absolute/relative positioning, refer to the relevant Mozilla documentation:

https://developer.mozilla.org/Learn/CSS/CSS_layout/Positioning

Next, your lecturer/tutor will provide a quick overview of the task. The "move eyes up" has already been coded for you.



Note that you will need to do some research to find out how to swap an image `src` using JavaScript (in order to get the "hollow eyes" and "sad mouth" working).

Further Reading

For a refresher (or if you missed class), you can refer to the relevant online documentation:

JavaScript introduction:

- https://developer.mozilla.org/Learn/JavaScript/First_steps/What_is_JavaScript
- https://developer.mozilla.org/Learn/JavaScript/First_steps/Variables
- https://developer.mozilla.org/Learn/JavaScript/First_steps/Math
- https://developer.mozilla.org/docs/Web/API/Document_Object_Model/Examples

end