JAVASCRIPT

EXPLORING JAVASCRIPT

To fully unlock the potential interactivity of a webpage, developers make extensive use of the JavaScript programming language. Using JavaScript, developers can change an HTML element's content, modify CSS styles, change attributes, and more. This is basically the editing of the Document Object Model. The web page reacts to the web users actions.

JavaScript is a programming language where developers specify instructions the browser must perform to accomplish a specific task. That makes JavaScript unique is that the instructions are executed within the browser, a process referred to by developers as client-side processing.

In other words JavaScript in a browser does have to be executed via a web server. It can be launched on your desktop (a html file) and the JavaScript within the HTML file will execute. I don't recommend this approach. Everything you develop should be tested by rendering it through the file being requested via a http request using a web browser. Upload your work to your web server and rendered it by going to the URL.

JavaScript is a dynamic, loosely typed, prototype-based programming language which is used in many different environments. As well as being the prevalent client-side programming language of the web, it's also used to write plugins for IDEs, in PDF files and as a basis for other platforms and higher abstractions.

It is also being used as a server side (compiles on the web server) language. Node.js is an example. Server-side JavaScript (SSJS) refers to JavaScript that runs on server-side and is therefore is not downloaded to the browser. JavaScript is used just like you'd use any language on the server, e.g. PHP/C# .NET. Typically to handle HTTP requests and generate content. JavaScript is predominantly used on the client-side.

Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics.

TRENDS

A web browser is made up of many different components, all working together to deliver you a fast and efficient web browsing experience. One such element is the JavaScript engine; a component responsible for compiling your JavaScript code into the native instructions your computer can run.

Support for JavaScript became a standard feature in every browser after 1995, with each vendor building an engine of their own to support it. Over the years these engines have evolved, been replaced, renamed and forked into other engines. Keeping track of all these versions is difficult…

By the mid-2000's JavaScript was standardised and prolific but code execution was still slow. A race for speed began in 2008 with a slew of new engines. What followed after 2008 were a succession of creative and innovative speed improvements in JavaScript engine design, and a race to build the fastest browser.

Despite this confusing proliferation of names, what all of these engines have in common is that they parse and execute JavaScript. What then is the difference between all these engines? **Well these days they compete for speed.** So unless your building a web-based real-time shooter. The details of speed are not that relevant at this moment. Just note that JavaScript is support in all major browsers, including mobiles. Although **they used different JavaScript engines to execute the code.**

GETTING STARTED WITH JAVASCRIPT

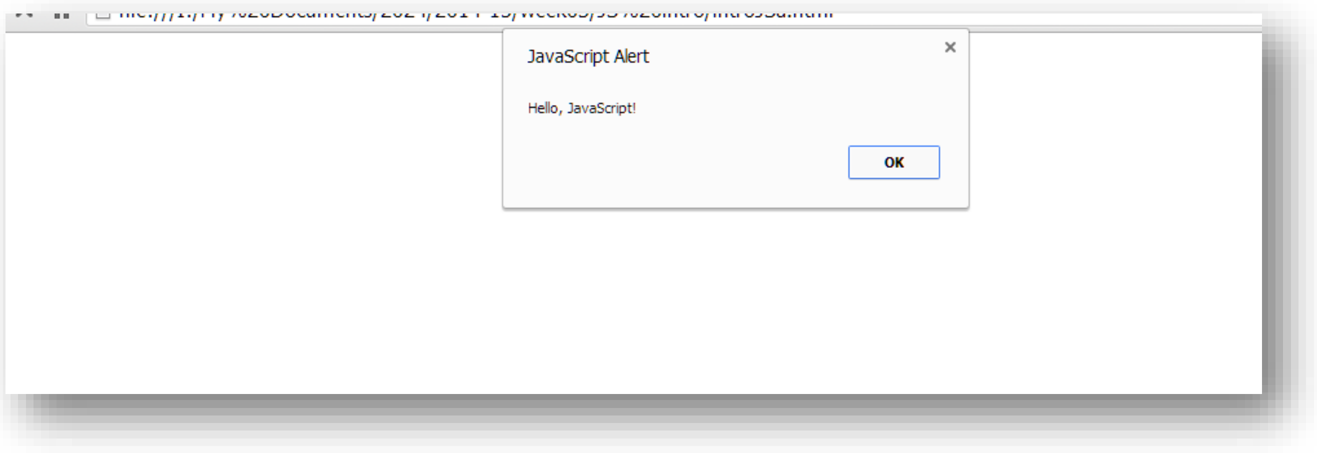
To place JavaScript programming instructions within an HTML file, developers normally place the <script> and </script> tag pair within the head section of the HTML file. The following HTML file, mark-up, places a single JavaScript statement that displays a dialog box with the message "Hello, JavaScript!" This is shown in

```
<!DOCTYPE html>
<html>
<head>
<script>
alert('Hello, JavaScript!');
</script>
</head>
<body>
</body>
</html>
```

In this case, when the browser loads the HTML file, the browser (not the Server) will execute the JavaScript statement

```
      alert('Hello, JavaScript!');
```

this directs it to display the dialog box.


JavaScript Alert

Hello, JavaScript!

In the previous example, the HTML file placed a <script> and </script> tag pair within the document's head section. The <script> tag is HTML 5 format. As you examine older JavaScript code within HTML files, you will encounter <script> tags in the following form:

```
<script type="text/javascript">
// code here
</script>
```

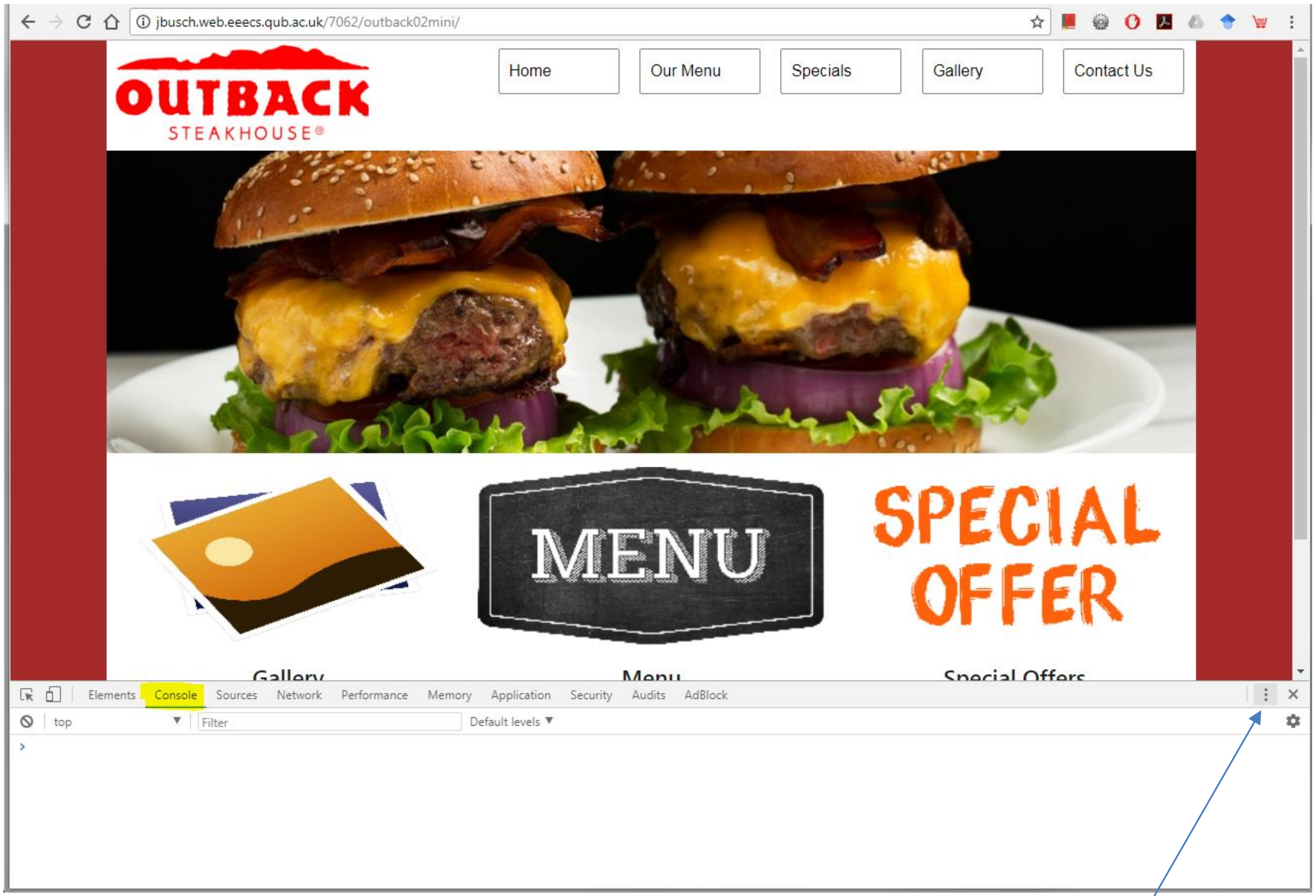
In this case, the type="text/javascript" attribute tells the browser that the script that follows will be JavaScript. Although this format still works, HTML5 has simplified it, making JavaScript the default scripting language. See the first example above which does not use the 'type' attribute. So the element can be just…

```
<script>
  alert("Hello World");
</script>
```

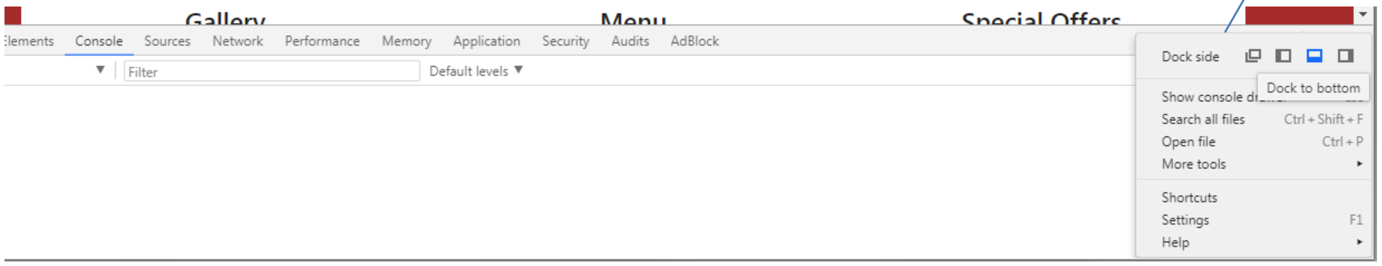
CONSOLE PANEL

Most modern browsers have a console panel. Chrome, IE and FireFox include in their Developer Tools environment. In Windows that can be opened up using F12 key.

You need to then display the 'Console' panel. Below is it open in Chrome…



Use the Settings button to move the Panel to the bottom of the browser…



The Console panel allows you to output JavaScript data and also displays any errors from the JavaScript runtime or the rendering runtime of the DOM (CSS and HTML).

Try to use the console.log() JavaScript method rather than the alert() method. The alert() causes the runtime to stop the processing of the web page.

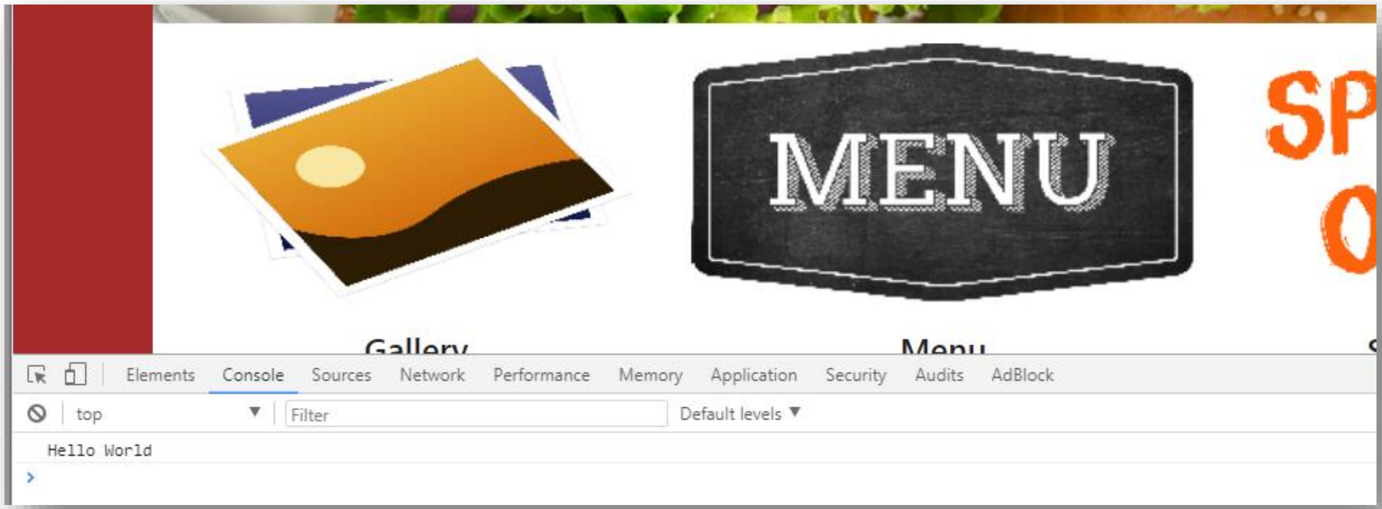
Open a web page source file e.g. outback02mini/index.html (week 04 zip file) and add the following JavaScript to the <head> element…

```
<script>

   console.log("Hello World");

</script>
```

Complete script below..

```
<!DOCTYPE html>
<html>

    <head>
        <title>Outback Steakhouse</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
         <link rel="stylesheet"
href="https://gitcdn.link/repo/Chalarangelo/mini.css/master/dist/mini-
default.min.css"/>
        <link rel="stylesheet" href="styles/styles.css">

        <script>
            console.log("Hello World");

        </script>
    </head>

    <body>
```

Save and upload it to your web server. View the Console panel to see…



```
Hello World
```

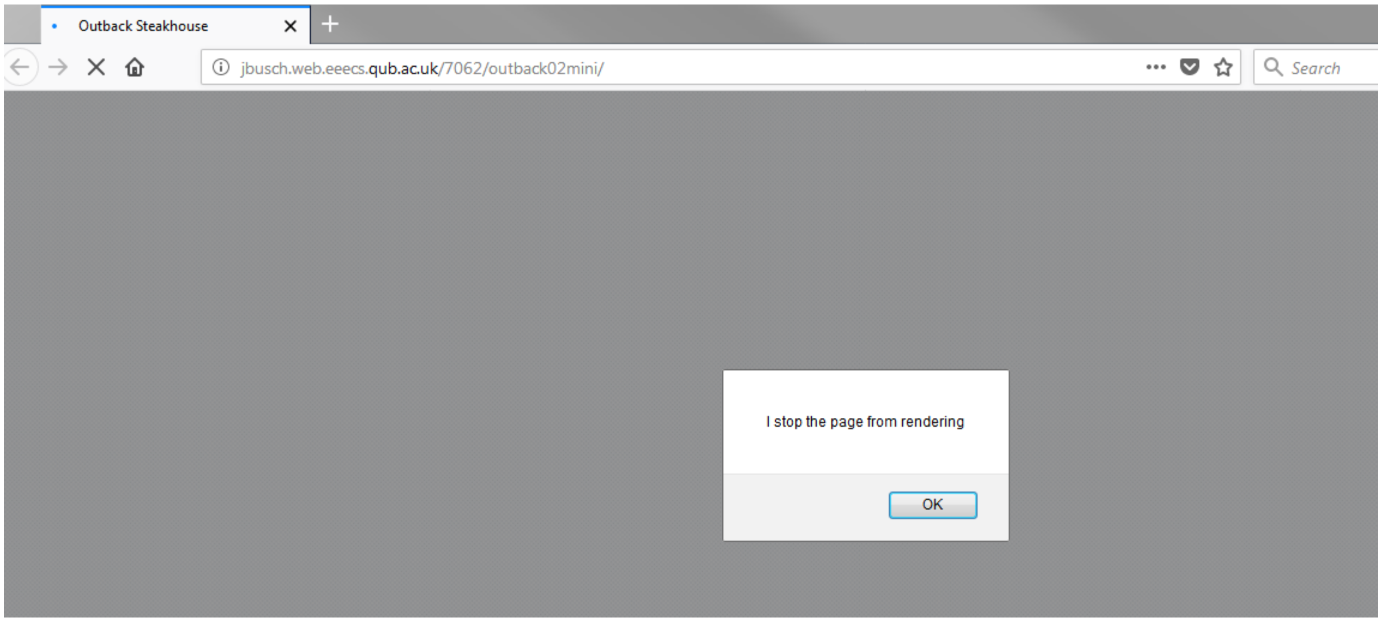
JAVASCRIPT FROM TOP TO BOTTOM

It is very important to understand when JavaScript is loaded onto a web page. If the <script> tags are in the <head> element, then the JavaScript will execute before any of the HTML tags within the <body> tag will render. Meaning the alert box will appear before any rendering of the pages content.

In outback02mini/index.html add the JS code…

```
<!DOCTYPE html>
<html>

    <head>
        <title>Outback Steakhouse</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
         <link rel="stylesheet"
href="https://gitcdn.link/repo/Chalarangelo/mini.css/master/dist/mini-
default.min.css"/>
        <link rel="stylesheet" href="styles/styles.css">

        <script>
            alert("I stop the page from rendering");
            console.log("Hello World");

        </script>
    </head>

    <body>
```

Save and upload. Use another browser to go and request the web page. This is due to caching issues, below is a render of the web page http://jbusch.web.eeecs.qub.ac.uk/7062/outback02mini in FireFox


Outback Steakhouse
jbusch.web.eeecs.qub.ac.uk/7062/outback02mini/
Search
I stop the page from rendering
OK

You now need to press the OK button to execute the rest of the JavaScript and the MARKUP.

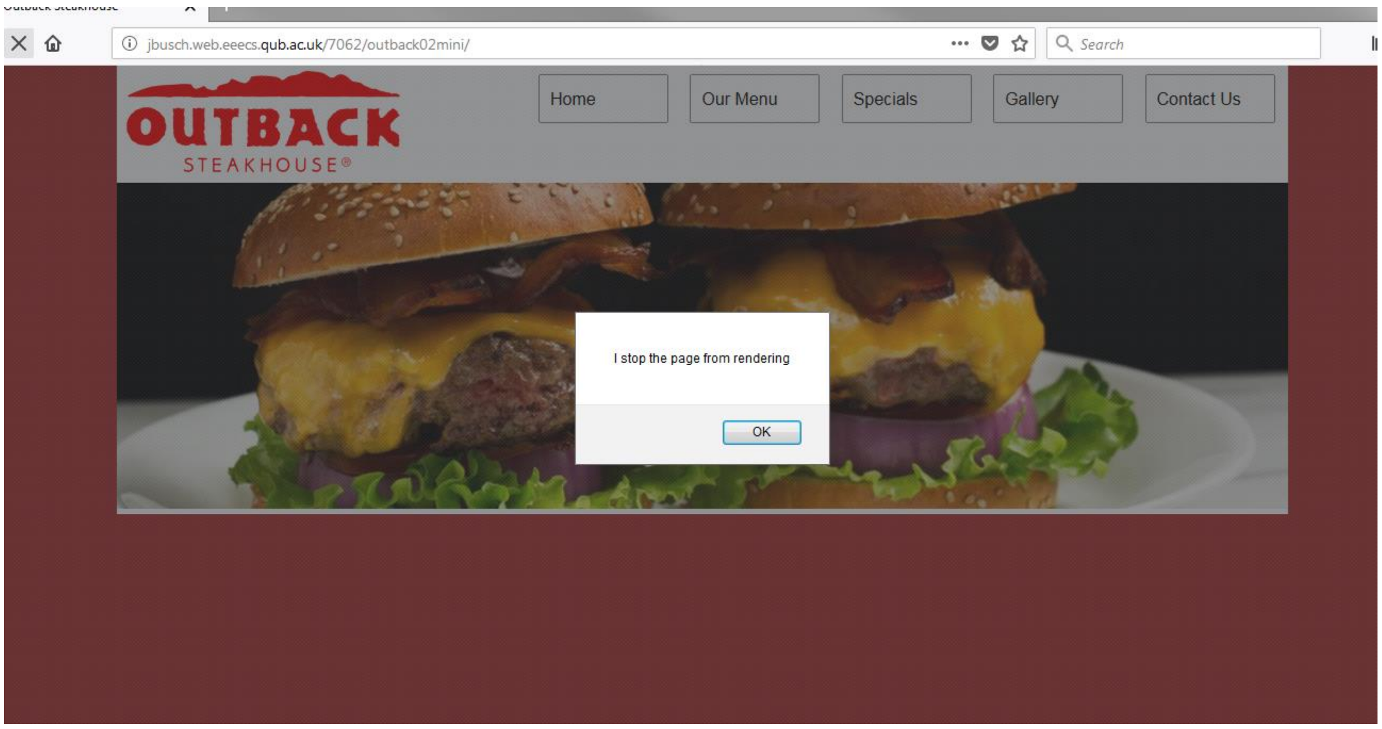
Because JavaScript is a client-side scripting language primarily used for web sites it position in the mark-up (HTML) can be placed anywhere. For example you can place the <script> tags within the <body> tags.

Move the <script> element code to…

```
            </div>
        </div>

        <!-- front image -->
        <div class="row">
                <div class="col-md-12 col-lg-12 nomargin">
                <img src="img/second.jpg" alt="outback image of food" class="full"/>
                </div>
        </div>

        <script>

            alert("I stop the page from rendering");
            console.log("Hello World");

        </script>

        <!-- panels -->

        <div class="row">
                <div class="col-sm-12 col-md-12 col-lg-4 back">
                      <img src="img/gallery.png" />
                      <h3>Gallery</h3>
                  <p>Within our gallery you can view some of the foods we serve and
see what the atmosphere within our restaurant is
```

Save and upload your file.

Render the file again but you need to close the FireFox browser and then navigate to the /outback02mini/index.html file…



I stop the page from rendering

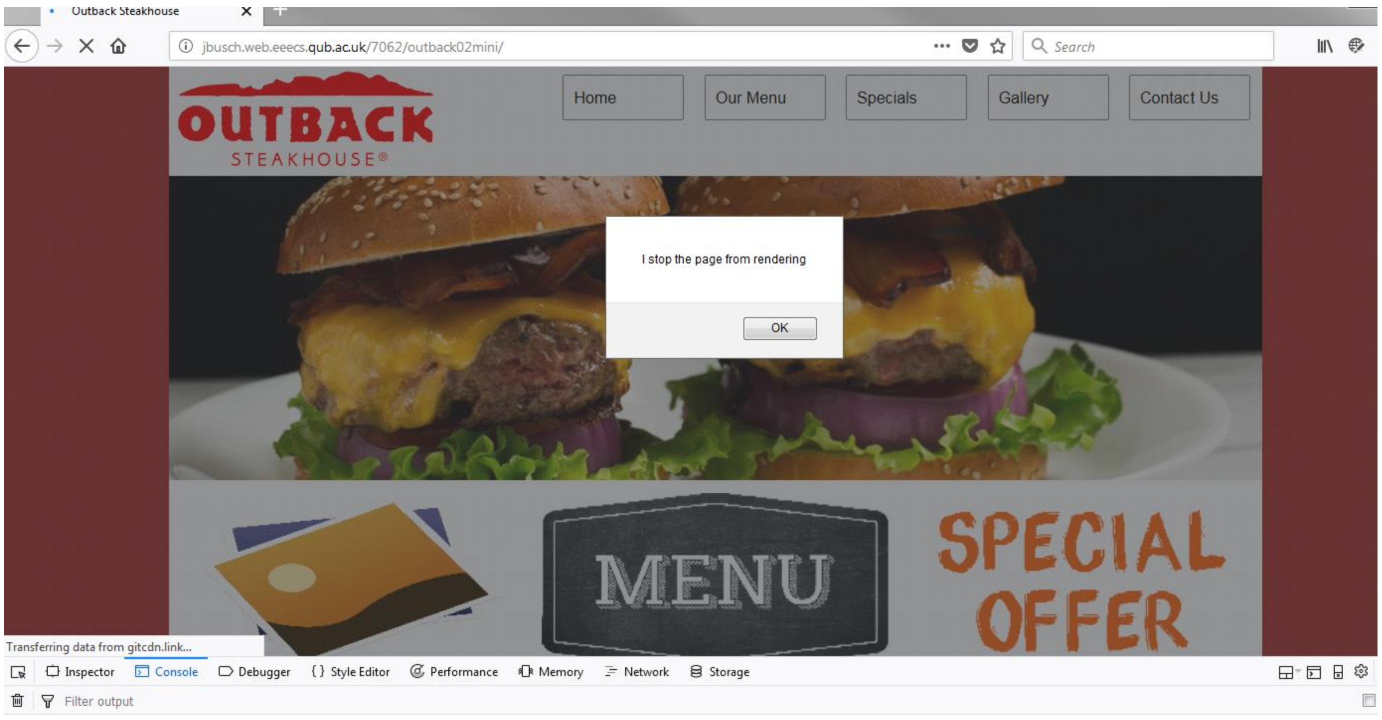
From the above render you can see the half of the web page has loaded until it is stopped by the JavaScript alert() method.

Finally JavaScript and be placed at the very bottom of the HTML mark-up. One reason why this is done is to make sure that all elements (HTML tags) are loaded before the browser executes the JavaScript. See introJSd.html, notice that the <script> tags are still inside the <html> tags.

Move the <script> element code to…

```
                        <p>View our menu, we offer a range of delicious meals and
foods. We offer various meats ranging from chicken to beef.</p>
                         <a href="menu.php" class="button">Our Menu</a>
                    </div>
                    <div class="col-sm-12  col-md-12 col-lg-4 back">
                                <img src="img/special.png" />
                                <h3>Special Offers</h3>
                         <p>At Outback Steakhouse we have different special offers
weekly, go see whats on this week!</p>
                          <a href="special.php" class="button">Specials</a>
                    </div>
            </div>
        </div>


    </body>
        <script>

            alert("I stop the page from rendering");
            console.log("Hello World");

        </script>
</html>
```

Save and upload your file. Render the file again but you need to close the FireFox browser and then navigate to the /outback02mini/index.html file…



I stop the page from rendering

OK

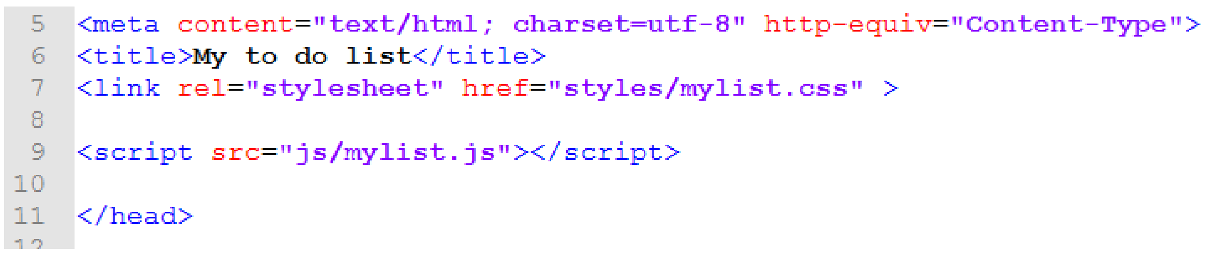
Transferring data from gitcdn.link...

NOTICE that the 'Hello Worls' did not execute until you press the OK button.

This basic understanding of the top down execution during runtime is crucial if you want to use JavaScript frameworks for development better web page interactivity.

EXTERNALLY LINKED JAVASCRIPT

Another popular method of adding JavaScript to a .html file is to externally link it…



```
 5  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
 6  <title>My to do list</title>
 7  <link rel="stylesheet" href="styles/mylist.css" >
 8
 9  <script src="js/mylist.js"></script>
10
11  </head>
```

Line 9 is asking the html document to load in the JavaScript that is held within the mylist.js file within the folder ‘js’.

This works similar to the way an image or CSS file is external linked using <img> or <link> tags.

Another advantage to this is that any JavaScript file that is stored on the web potentially can be referenced…

```
 <head>
    <title>jquery</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js">
    </script>

</head>
```

Meaning that the actual file is not located in your web server but on another web server. The top <script> element loads in the JQuery JavaScript library to make clientside coding a little be easier and more powerful.

JAVASCRIPT SYNTAX

You are not expected to understand the full 100% JavaScript code base. But familiarity with the syntax is essential.

Understanding statements, variable naming, whitespace, and other basic JavaScript syntax.

A simple variable declaration

```
<script>
var foo = 'hello world';
console.log(foo);
</script>
```

Whitespace has no meaning outside of quotation marks

```
var foo =         'hello world';
```

Parentheses indicate precedence

```
<script>

var cal = 3 * 3 + 5;    // returns 14; multiplication happens first
var cal2 = 4 * (5 + 5);  // returns 40; addition happens first

console.log(cal);
console.log(cal2);

</script>
```

Tabs enhance readability, but have no special meaning

```
<script>

             console.log('hello');

<script>
```

USING JAVASCRIPT OPERATORS

When programming, you often need to perform arithmetic operations, such as addition, subtraction, and multiplication. To help you perform such operations, JavaScript provides the set of operators defined below. The following HTML mark-up, demonstrates the use of several JavaScript arithmetic operators:

```
<!DOCTYPE html>
      <html>
      <head>
      <script>
            console.log(3+4);
            console.log(3*4);
            console.log3/4);
      </script>
      </head>
<body>
</body>
</html>
```

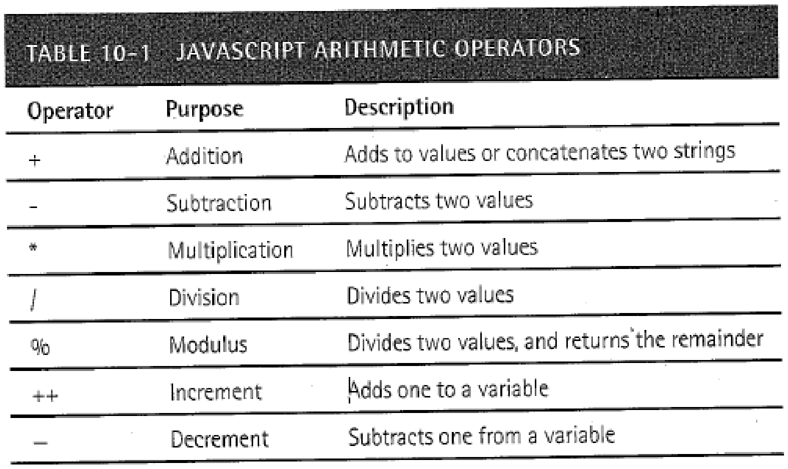


TABLE 10-1 JAVASCRIPT ARITHMETIC OPERATORS

Operator	Purpose	Description
+	Addition	Adds to values or concatenates two strings
-	Subtraction	Subtracts two values
*	Multiplication	Multiplies two values
/	Division	Divides two values
%	Modulus	Divides two values, and returns the remainder
++	Increment	Adds one to a variable
–	Decrement	Subtracts one from a variable

In this case, the file uses three separate alert dialog boxes to display the math results in the console panel on the browser.

STORING DATA IN JAVASCRIPT VARIABLES

As you write program instructions, even for simple tasks, it is common to need to temporarily store values that your instructions will later use. Programmers store such temporary values using named locations in memory called variables. In the simplest sense, a variable is the name of a storage location in memory. In other programming languages, you have to provide specifics about the memory location, such as the size of the value the location will hold as well as the type of value, such as a number, date, or sequence of characters. JavaScript, in contrast, is a "loosely typed" programming language, which means you simply need to specify a variable name, but not a related data type. To declare a JavaScript variable, use the var keyword followed by a variable name. A keyword is a word reserved by a programming language that has special meaning to the language and cannot be used for a variable name. The following statement creates a variable named message:

```
var message;
```

When you declare a variable, choose a name for the variable that describes the content it will store. Avoid names such as A, B, or C because they do not give another programmer who is reading your code any information about the value the variable stores. In contrast, names such as Username, OrderDate, or GraphicsFile are much more meaningful.

When you declare variables within JavaScript, the names of several variables can be specified within one statement:

```
var Username, OrderDate, GraphicsFile;
```

Or you can individually declare each variable:

```
var Username;
var OrderDate;
var GraphicsFile;
```

JavaScript is a "forgiving" programming language. If you forget to declare a variable, JavaScript creates one for you when it encounters the first occurrence of an unknown name. As a best practice, however, declare each variable before using it.

ASSIGNING A VALUE TO A VARIABLE

You should have seen this all before in previous modules.

As you have learned, a variable defines a storage location in memory for storing values as a program's instructions execute. To create a variable within JavaScript, use the `var` keyword to specify the variable's name.

After you declare a variable, use the equal sign, which is the JavaScript assignment operator, to assign a value to the variable:

```
var Age;
Age = 21;
```

In this case, the code used two statements to declare and then to initialize the variable Age. Both operations also can be performed in one statement, as shown here:

```
var Age = 21;
```

When you assign a value to a variable, the type of value you assign may differ. The following statements assign an integer, a floating-point, and a character string value to different variables:

```
var Age = 21;
var Salary= 35000.00;
var Name = 'Jane Doe';
```

As you can see, the variable Age is assigned the integer value 21. An integer value is essentially a counting number. The following statement assigns the value 35000.00 to the variable Salary. Developers refer to numbers with a decimal point as a floating-point number. Finally, the last statement assigns a name, contained in single quotes, to the variable Name. Programmers refer to characters contained between quotes as a string. The following HTML markup, assigns values to three variables and then displays each variable's value using an alert dialog box:

```
<!DOCTYPE html>
<html>
 <head>
      <script>
       var Age = 21;
       var Salary= 35000.00;
       var Name= 'Jane Doe';
       console.log (Age);
       console.log (Salary);
       console.log (Name);
      </script>
 </head>
<body>
 ...
</body>
</html>
```

Within the code, Age, Salary, and Name are variables which contain a value. To reference the value a variable stores, you simply use the variable name. That's why there are no single quotes around the names in the console.log() function. The file uses the assignment operator to assign values to each variable. After you assign a value to a variable, when you use the variable's name within your code, JavaScript retrieves the variable's value from memory and substitutes that value for the name.

SINGLE QUOTES VERSUS DOUBLE QUOTES

A character string is a sequence of characters enclosed within quotes. In JavaScript, you can use either single or double quotes to group the characters.

However, you must use the quotes consistently; that is, if you start with a single quote, you must end with a single quote:

```
var Firstname = 'Kris';
var Lastname "Jamsa";
```

So…

```
var Lastname 'Jamsa";
```

Would throw an error.

WHAT WOULD I USE?

I would try to use single quotes. The reason will not become clear until we start using server-side scripting. PHP can be use to write/print HTML and JavaScript. Using a single quote will make this process easier to understand.

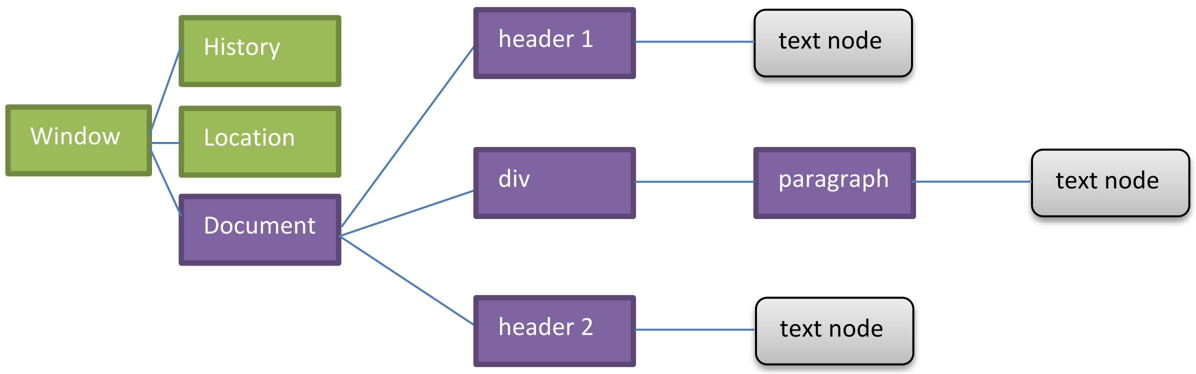
DOCUMENT OBJECT MODEL

The Document Object Model, normally abbreviated to DOM, is the API through which JavaScript interacts with content within a website. JavaScript and the DOM are usually seen as a single entity since JavaScript is most commonly used for this purpose (interacting with content on the web). The DOM API is used to access, traverse and manipulate HTML .

Write and render a simple HTML like…

```
<!DOCTYPE html>
<html>
 <head>

 </head>
<body>

 <h1>My First Page</h1>

 <div>
      <p>Hello world.</p>
</div>

 <h2>Footer</h2>

</body>
</html>
```

The browser rendering engine will build the DOM like…



The window object serves as the global object, you access it by just typing "window". It's within this object that all of your JavaScript code is executed. Like all objects it has properties and methods.

A property is a variable stored under an object. All variables created on a web-page automatically become properties of the window object.

A method is a function stored under an object. Since all functions are stored under (at least) the window object they can all be referred to as 'methods'.

The DOM creates a hierarchy corresponding to the structure of each web document. This hierarchy is made up of nodes. There are several different types of DOM nodes, the most important are 'Element', 'Text' and 'Document'.

An 'Element' node represents an element within a page. So if you have a paragraph element ('<p>') then it can be accessed through the DOM as a node.

A 'Text' node represents all text (within elements) within a page. So if your paragraph has a bit of text in it can be directly accessed through the DOM.

The 'Document' node represents the entire document. (It's the root-node of the DOM hierarchy/tree).

Also note that element attributes are DOM nodes themselves.

Each layout engine has a slightly different implementation of the DOM standard. For example, the Firefox web browser, which uses the Gecko layout engine, has quite a good implementation but Internet Explorer, which uses the Trident layout engine is known for its buggy and incomplete implementation. This is the reason why layouts of webpages look fine on one browser but when rendered through another browser a completely different.

You can directly access the objects within the DOM (HTML tags or their content) using JavaScript. Let use this HTML mark-up…

```
<!DOCTYPE html>
<html>
 <head>

 </head>
<body>


 <div id='mycontent'>
      <p>Hello world.</p>
 </div>
```

```
</body>
</html>
```

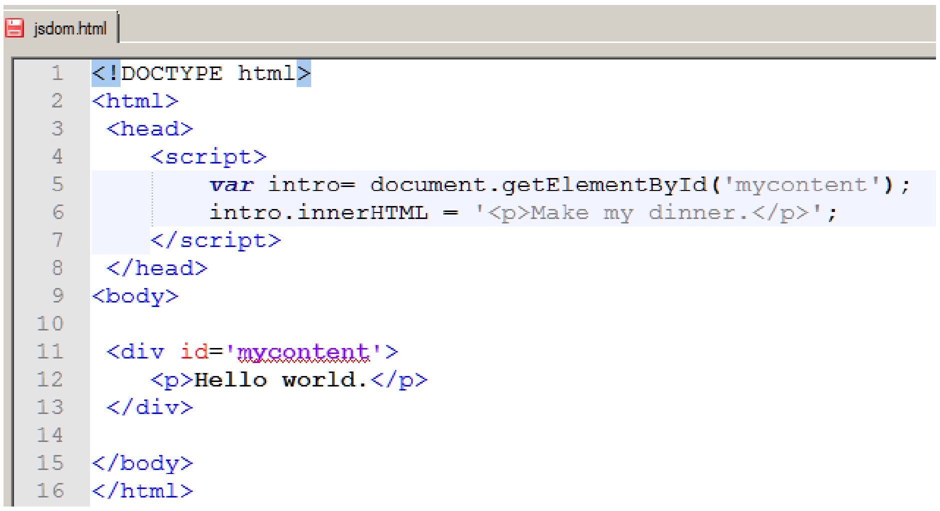
In this first example we're going to access our paragraph by using the 'getElementById' DOM method:

```
var intro= document.getElementById('mycontent');
```

The JavaScript variable 'intro' is now a reference to the DOM node. We can do a number of things with this node, - we can query its content and attributes, and can manipulate any aspect of it. We can remove it, clone it or move it to other parts of the DOM tree. For example changes its text node content from 'Hello World.' to 'Make my dinner.' …

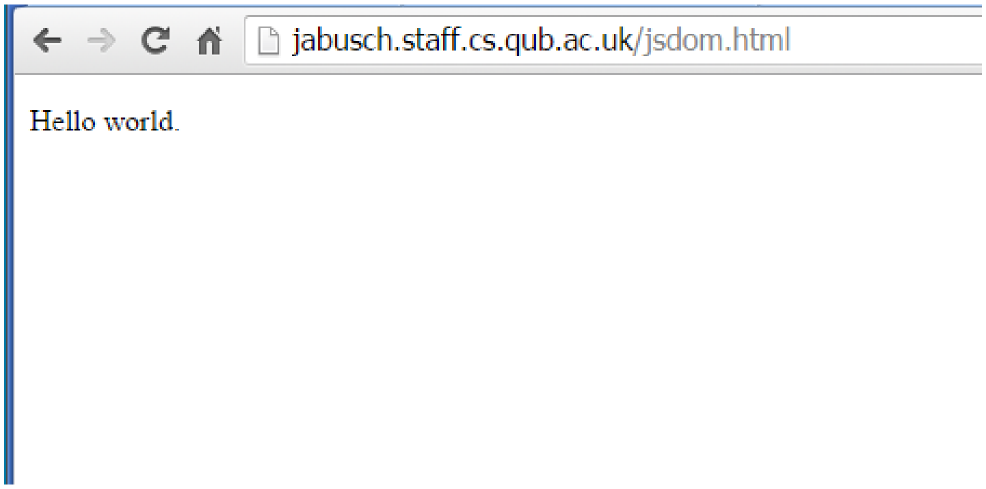
```
var intro= document.getElementById('mycontent');

intro.innerHTML = '<p>Make my dinner.</p>';
```

Look at the full code…



```
<!DOCTYPE html>
<html>
 <head>
    <script>
        var intro= document.getElementById('mycontent');
        intro.innerHTML = '<p>Make my dinner.</p>';
    </script>
 </head>
<body>

 <div id='mycontent'>
    <p>Hello world.</p>
 </div>

</body>
</html>
```

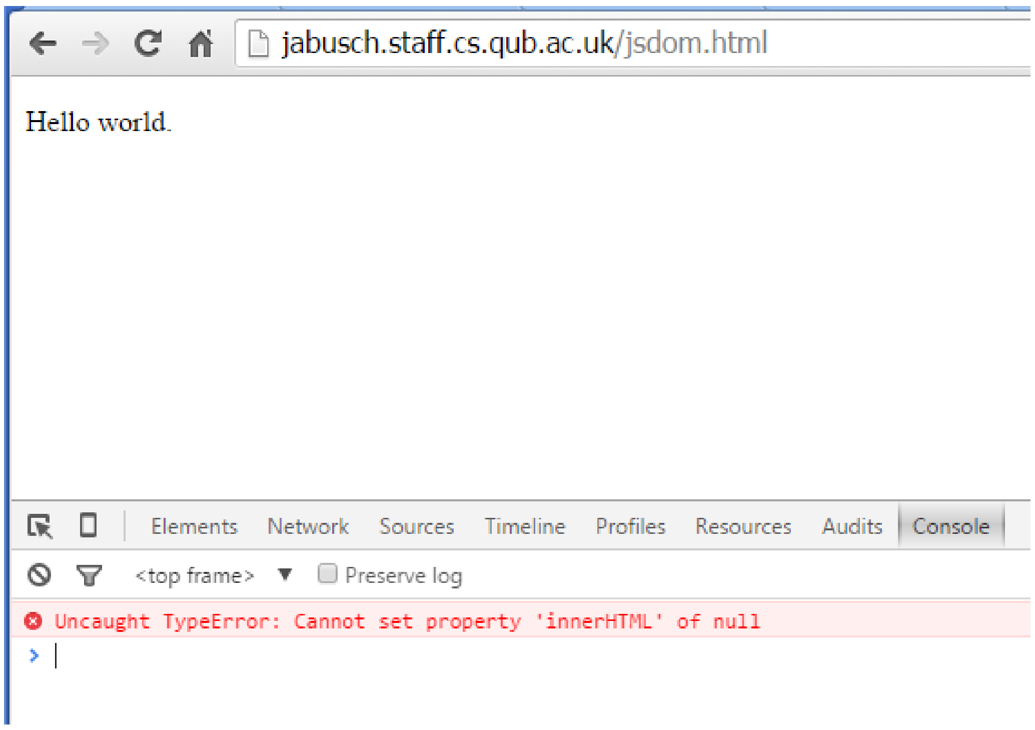
Render it…



Hello world.

But it hasn't changed? Surely the text should be `'Make my dinner.'`

This is what we want. The syntax and mark are fine but you get an error in the JavaScript console…

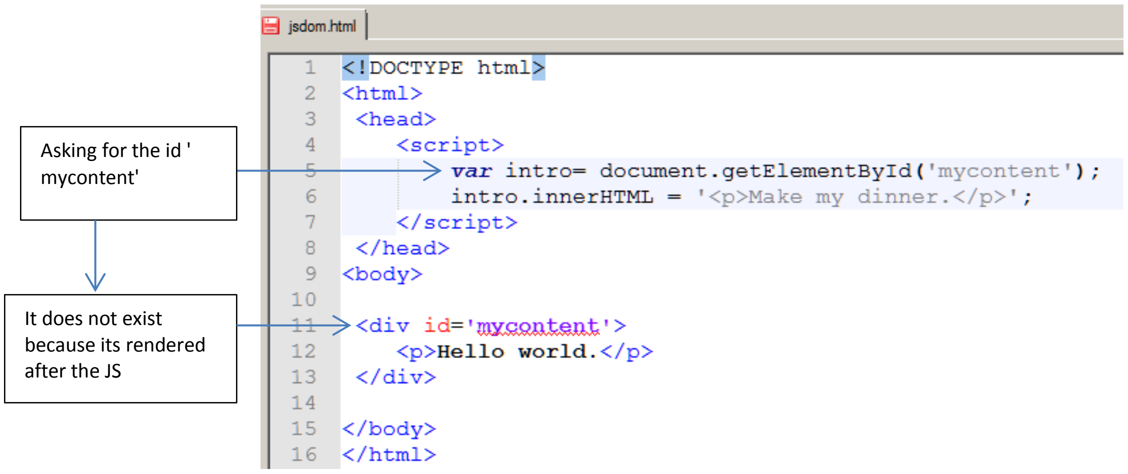


Hello world.

```
Uncaught TypeError: Cannot set property 'innerHTML' of null
```

JavaScript debugger is saying that the intro variable that is a reference to the DOM object it NULL. Why?

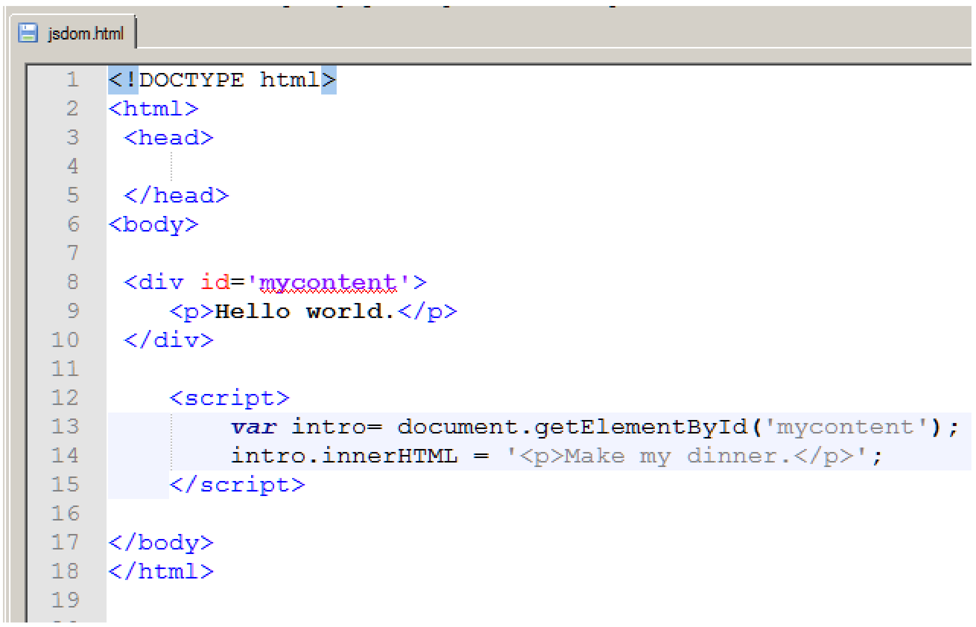
JavaScript and HTML render from top down. So looking at the structure of the code and markup we are referencing markup (id='mycontent') that has not been loaded into the DOM yet…


jsdom.html
Asking for the id ' mycontent'
It does not exist because its rendered after the JS

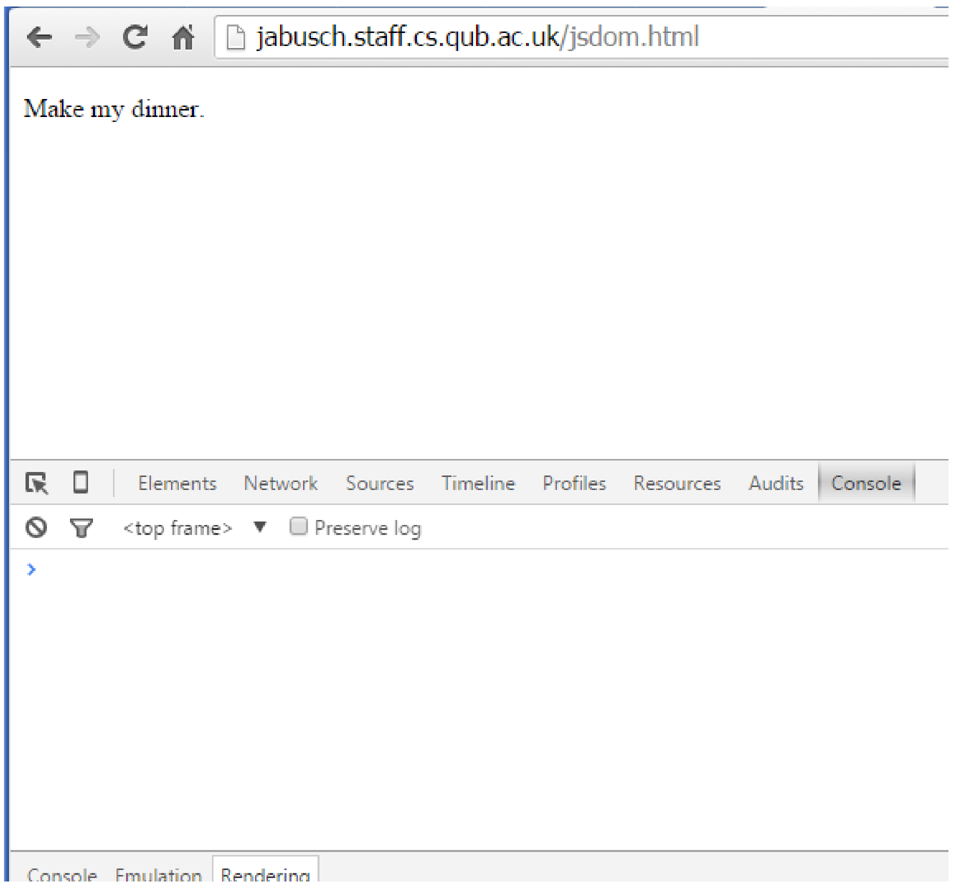
```
<!DOCTYPE html>
<html>
 <head>
    <script>
        var intro= document.getElementById('mycontent');
        intro.innerHTML = '<p>Make my dinner.</p>';
    </script>
 </head>
<body>

 <div id='mycontent'>
    <p>Hello world.</p>
 </div>

</body>
</html>
```

How to fix?

Well, the <div id='mycontent'> needs to be render first before the JS code. So move the <script> tag to the bottom of the structure. (We will start to use JQUERY JavaScript library to solve these rendering hierarchy issues)


jsdom.html

```
<!DOCTYPE html>
<html>
 <head>

 </head>
<body>

 <div id='mycontent'>
    <p>Hello world.</p>
 </div>

    <script>
        var intro= document.getElementById('mycontent');
        intro.innerHTML = '<p>Make my dinner.</p>';
    </script>

</body>
</html>

```


jabusch.staff.cs.qub.ac.uk/jsdom.html

Make my dinner.

FURTHER READING

Good article on the basics of JavaScript.

https://autotelicum.github.io/Smooth-CoffeeScript/literate/js-intro.html

It's NOT essential to understand JavaScript in full. Just try to be aware of its ability to interact with the elements on a web page.