Javascript & the DOM

Computer Science and IT, Massey University

A good (free) book on HTML5 & Javascript

The following are links to online copies of free books:

HTML5

"Dive into HTML5" by Mark Pilgrim and the community.

JavaScript

"Eloquent JavaScript" by Marijn Haverbeke

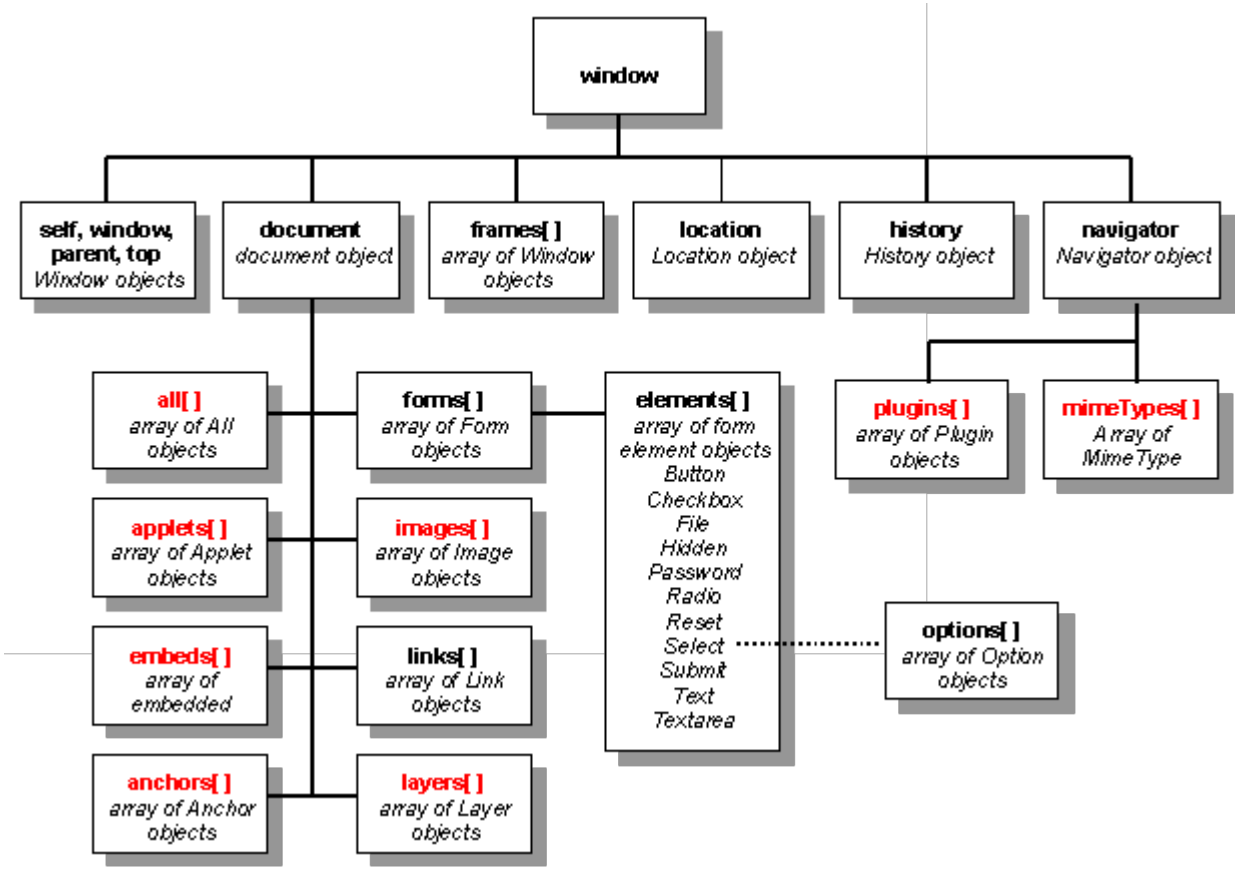
IMPORTANT - remember to use 'let' and 'const' for local variables

Coming from Python, it's easy to forget the variables in JavaScript function will be GLOBAL unless prefixed with 'let'

I may forget. This isn't intentional …

Remember to always put 'let' to get local variables

The *Document Object Model* (DOM) - I



The *Document Object Model* (DOM) - II

- The DOM is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure
- Each node in the DOM tree is an object representing a part of the document;
- The DOM represents a document with a logical tree;
- Each branch of the tree ends in a node, and each node contains objects;
- DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.[2]

The Document & Browser Object Models

The Window object

- window. is optional - all non-qualified variables/methods will be treated as belonging to the **.window** object so **window.document.getElementById('button1')** can be shortened to **document.getElementById('button1')**
- **Properties:** Window dimensions: .innerwidth, .innerHeight
- **Methods:**
  - *window.open()*
  - *window.close()*
  - *window.resizeTo(width, height)*
  - *window.moveTo(topLeftX, topLeftY)*

Dynamically building the page

Add elements to the page (document)

- use **document.createElement()**

then

- add if AFTER an element - use **element.appendChild()**, OR
- add it BEFORE an element - use **element.insertBefore()**

Can also remove elements.

Dynamic page building - example

```
    <p id='target'> <b>Target</b> </p>

<script>
function addAfter() {
  var newP = document.createElement('p')          // Make an empty paragraph
  var pText= document.createTextNode(new Date()) // Make some text
  newP.appendChild(pText)                         // Add text to paragraph

  document.getElementById('target').appendChild(newP)
}
</script>
<button onclick='addAfter()'>Add After Target</button>
```

Target

Try: Add After Target

Finding page elements

The DOM has a NodeList object - used to return collection of elements

by Tag

- e.g. **document.getElementsByTagName('p')**

by Class:

- using **document.getElementsByClassName()**
    - *e.g. document.getElementsByClassName('myDiv')*

or individually

- **document.getElementById()**

Finding an altering all paragraphs - by 'p' tag

Here's a paragraph - just a <p> tag

```
    <script>
function redParagraphs(){
  var paragraphs = document.getElementsByTagName('p')
  alert('No of paragraphs = ' + paragraphs.length)
  for (i=0; i< paragraphs.length; i++){
     thisPara = paragraphs[i].style.color = '#FF0000'
  }
}
</script>
<button onclick='redParagraphs()'> Make all paragraphs RED</button>
```

Make all paragraphs RED

Logging - console.log()

use **console.log()** to output logging info without using alert() boxes or making text part of the page

```
    <button onclick="outputToLog()">Output a Log line</button>
<script>
function outputToLog(){
  var timestamp = new Date()
  console.log("Gidday, it's " + timestamp)
}
</script>
```

To see the console, in Chrome:

- *right-click* the button & choose 'Inspect'
- select the 'Console' tab & **Logging** sub-tab
- click the button several times

Output a Log line

Creating New Windows

```
    <h3>New Window Demo</h3>
Width <input type=number id='newX'  min=320 max=1024 value=900><br />
Height <input type=number id='newY' min=240 max=1024 value=700><br />
<input type=button onClick='newWindow()' value="Open TheRegister.com">

<script>
function newWindow(){
   var x = parseInt(document.getElementById('newX').value)
   var y = parseInt(document.getElementById('newY').value)
   console.log('new Window X & Y = ' + x + ', ' + y)
   var newSize = "width="+x+", height="+y
   var newWin = window.open("http://theRegister.com", "T2", newSize)
   console.log("Window created")
}
</script>
```

New Window Demo

Width 900

Height 700

Open TheRegister.com

Using the colour selector

Note that the actual type is 'color' NOT the UK English 'colour'

```
    <h4>Change Text Colour</h4>
<div style='border:solid'>
<p id='colText'>Hi there, let's change my colour!</p>

<input type='color' id=colPicker value='#00FF00'>
<input type=submit value='Set Color' onClick='toRed()'>

<script>
function toRed(){
   var textItem = document.getElementById('colText')
   var colorValue = document.getElementById('colPicker').value
   textItem.style.color=colorValue
}
</script>
</div>
```

Change Text Colour

Hi there, let's change my colour!

Set Color

Javascript Objects

Objects can be created by defining them (like Python dictionaries)

- can add more properties later
- can add **methods** by setting functions as properties

```
    <script>
function objectDemo() {
    var p = {name: 'Belinda', eyecolor:'#8888FF'}
    p.height = 1.6

    p.getname = function () { return "My name is " + this.name }

    p.setname = function (x) {
       this.name = x
    }

    alert('Eye color: ' + p.eyecolor)
    alert('Name 1: ' + p.getname())
    p.setname('Fred')
    alert('Name 2: ' + p.getname())
}
</script>

<button onclick="objectDemo()"> Run Object Demo </button>
```

Run Object Demo

Creating object by Constructor

Make a function to recreate and return a new object

```
    <script>
function Person(name, height) {
    var p = {}
    p.name = name
    p.height = height
    p.getname = function ()  { return "My name is " + this.name }
    p.setname = function (x) { this.name = x }
    return p
}
function constructorDemo() {
    var will = new Person('William', '#FF0000') // make a new person
    alert('Name 1: ' + will.getname())
    will.setname('Bill')
    alert('Name 2: ' + will.getname())
}
</script>

<button onclick="constructorDemo()"> Run constructor demo </button>
```

Run constructor demo

A simpler object constructor

- **this** refers to the current object - we can use it as the prototype
- IMPORTANT: use **new** when invoking the constructor

```
    <script>
function Person(name, height) {
    this.name = name
    this.height = height
    this.getname = function ()  { return "My name is " + this.name }
    this.setname = function (x) { this.name = x }
    return this
}
function constructorDemo() {
    var p1 = new Person('Cynthia', '#FF0000') // make a new person
    alert('Person 1: ' + p1.getname())

    var p2 = new Person('David', '#00FF00')
    alert('Person 2: ' + p2.getname())
    alert('Person 1: ' + p1.getname() + "should be 'Cynthia'")
}
</script>

<button onclick="constructorDemo()"> Run constructor demo </button>
```

Run simpler constructor demo

Skim: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS

Javascript Exception Handling #1

Just as in Python, there are places where you know the code might fail.

There's the try-catch (NOT try-except) construct to hand this

```
    try {
   code that might fail
}
catch (err) {
   // err is an exception object - you can find what happened
}
```

err is an error object with properties:

- err.name
- err.message

Skim: https://www.w3schools.com/js/js_errors.asp

Javascript Exceptions - *finally* & *throw*

```
    try {
   code that might fail
}
catch (err) {
   // err is an exception object - you can find what happened
}
finally {    // This block is optional
   // always do this
}
```

***throw* can be used to raise exceptions**

for details, **skim** https://www.w3schools.com/js/js_errors.asp

Converting Numbers - parseInt() & parseFloat()

You can convert string to an *int* or a *float*

- only the first number will be returned
- if there's no number, the object **NaN** will be returned
- use isNaN(s) to test returned object:

```
    y = parseFloat(S)    // Try to convert string S
if (isNaN(y))
  alert("isNaN(y): S is not a number")
else
  // y is a float - now use it ...
```

parseInt() & parseFloat example

```
    <script>
  function testConversion() {
    alert(  parseInt('123.8andme'));
    alert(parseFloat('123.8andme'));

    var y = parseFloat("Gidday")

    if (y == 'NaN')
      alert("y == 'Nan': That's not a number")

    if (isNaN(y))
      alert("isNaN(y): That's not a number")

    try {
      if (y == Nan)
        alert("y == Nan: That's not a number");
    }
    catch (err) {
      alert("Got an Exception: " + err.message)
    }
  }
</script>

<p>Click the evaluate.</p>
<button onclick="testConversion()">test parseInt() & parseFloat() conversions</button>
```

Try it: test parseInt() & parseFloat() conversions