SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011 COS60004 COS60007 Creating Web Applications (and Databases)

Lecture 7

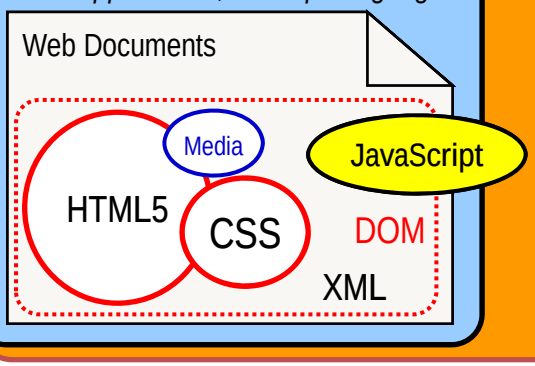
Document Object Model

Unit of Study Outline

Internet Technologies: TCP/IP, URLs, URIs, DNS, MIME, SSL

Web Technologies: HTTP, HTTPS, Web Architectural Principles

Client Side Technologies:
Web Applications, Markup Languages



Standards
Quality Assurance
Accessibility
Usability
Security

2

Previously – Linking JavaScript to HTML

HTML - *content*

```
<!DOCTYPE html>
<html lang="en">
<head>
    ...
    <script src="my_jsfile.js"></script>
</head>
<body>
    ...
    <p><span id="mymessage"></span></p>
    <p><button type="button" id="clickme"
     Click Me!</button></p>>
    ...
</body>
</html>
```

JavaScript - *behaviour*

```
/* Filename: my_jsfile.js
   ...
*/

function doSomething()
{
    var myString, outputMessage;     //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}

function init() {
    var clickme = document.getElementById("clickme");
    clickme.onclick = doSomething;
    }

window.onload = init;
```

3

Previously – JavaScript Syntax

- Data Types
  - Primitive
- Variables
  - Naming
- Constants
- Expressions
  - Operators
    - String, Arithmetic, Logical, Comparison
    - Assignment
- Functions
  - function definition
  - parameters
  - call and return
- Variable scope
- Statements
  - Sequence
  - Selection
  - Repetition

4 - Creating Web Applications, © Swinburne

Previously – Form Validation

- Regular Expressions
- Input data validation using HTML5
- Input data validation using JavaScript

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- Predefined Objects
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
  - JavaScript Core Objects and Global Functions
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Contents

Document Object Model and JavaScript

- **JavaScript Objects** **.properties .methods()**
- Predefined Objects
  - JavaScript Core Objects and Global Functions
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

JavaScript Objects

- JavaScript is an **object-based language** and does some things procedurally
- It can support polymorphism, inheritance and encapsulation.
- It can access objects such as
  - browser **object** such as **window**, **navigator**
  - webpage **objects** such as **document, images**, **dates, forms** and the hierarchy of form control elements such as **inputs, checkboxes, select**, and **buttons**, etc., within forms.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

JavaScript Objects (continued)

How do we manage objects in JavaScript?

- For example, imagine an object, say a ball.

  With this ball we can ask a couple of generalised "what" questions
  - What does the ball look like?
  - What can the ball do?

JavaScript Objects (continued)

We can make the questions more specific:

- **What attributes does our ball have?**
  - What colour
  - What size
  - What weight

These are referred to as **properties**

- **What can the ball do?**
  - What can it do
  - What can we do with it

These are referred to as **methods**

JavaScript Objects (continued)

Objects have

- **properties** which **describe** it.
  - a form input can have a value.
  - thought of as *nouns* as they *describe things*.
- **methods** which **describe actions** that an object can do.
  These are indicated using parentheses ()
  - a form can be submitted, buttons can be clicked
  - thought of as *verbs* as they *describe actions*.

JavaScript Objects (continued)

In JavaScript, we say

- **What properties does a ball have?**

```
ball.colour
ball.size
ball.weight
```

Dot notation

- **What methods does a ball have?**

```
ball.bounce()
ball.hit()
```

parenthesis

?

- Where have we seen *object.property* and *object.method* notation before?

```
function doSomething()
{
    var myString, outputMessage;    //declare local variables
    myString = prompt("Enter the string", "The string");
    alert("Your output: " + myString);
    outputMessage = document.getElementById("mymessage");
    outputMessage.textContent="Your output: " + myString;
}

function init() {
    var clickme = document.getElementById("clickme");
    clickme.onclick = doSomething;
    }

window.onload = init;
```

object property

object method

Custom Objects – Make Your Own

Create a specific instance of a **ball** object from class **Ball()** placed it in an object 'container' named **myObject**

- To create
  `myObject = new Ball(…)`
- To access the ball's **properties**
  `myObject.colour`
  `myObject.size`
- To access the ball's **methods**
  `myObject.bounce()`
  `myObject.hit()`

In this unit we will use pre-defined objects.

- ***JavaScript core objects***
- ***Browser / DOM objects***

In more advanced programming you will create you own objects.

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- **Predefined Objects**
  - **JavaScript Core Objects and Global Functions**
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Predefined Objects – JS Core Objects

- Array
- Boolean
- Date
- Math
- Number
- RegExp
- String

Example

```
// returns PI
var x = Math.PI;
```

Object prototype
Note: starts with a Capital Letter

https://developer.mozilla.org/en/JavaScript/Guide/Predefined_Core_Objects

Array Object

- an indexed collection of variables
- a particular location or **element** in an array is referenced by the name of the array and the **index** position.

For example:

index

an element

marks	80	75	85	95	70	65	90

`marks[0]` contains 80

`marks[4]` contains 70

`marks.length` contains 7

property

Array Object (continued)

- In JavaScript an **Array** is an object.
- The **new** keyword is used in JavaScript to create an instance of an Array object.

```
var marks;
marks = new Array(10);
                    // 10 places, 0-9
// or
var marks = new Array(15);
                    // 15 places, 0-14
```

parenthesis

uses plural form to indicate array

Array Object (continued)

- Element values may be set in an **initialiser list**:

```
subjects =
   new Array("CWA","WAD","WAA");
numbers =
   new Array(1,1,2,3,5,8,13);
```

- Alternatively values may be set after the array has been allocated by referring to the index position of the particular array element:

```
favSubjects = new Array(2);
favSubjects[0] = "CWA";
favSubjects[1] = "WAD";
```

parenthesis

Square brackets

Array Object (continued)

- The length of an array can be accessed using the **`length`** property. e.g. **`numbers.length`**
- Values can be set programmatically:

```
// create an array
var numbers = new Array(100)
// fill array with numbers
for (i=0; i < numbers.length; i++) {
    numbers[i] = i*2;
}
// display the last element
alert (numbers[numbers.length - 1]);
```

Why do not subtract 1?

Why subtract 1?

Array Object (continued)

- There are several **predefined** arrays in the **document** object, such as link**s**, frame**s**, image**s**

```
myLink = document.links[0];
myImage = document.images[5];
myNode =
 document.documentElement.children[0];
```

Pre-defined arrays uses plural form to indicate collection of elements

Array Object - Example

Example: Display scores array as a horizontal 'chart'

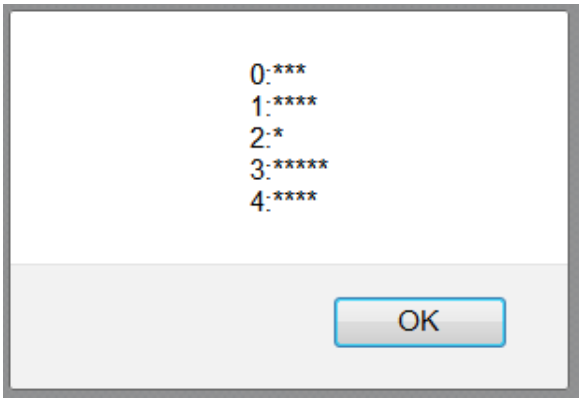
```
var scores = new Array(3,4,1,5,4);
var index;        // array index
var num;          // number
var ans = "";           // string for output
// Demonstrates how to use for-in loop
for (index in scores) {
   num = scores[index];
   ans = index.toString() + ": ";

   for (i=0; i<num; i++) {
      ans = ans + "*";
   }
   ans = ans + "\n";
}
alert (ans);
```

Method to convert a number to a string

\n for line break

Array Object - Example (continued)

Example: alert box will display



Array Object – Properties/Methods

Property/Method	Description
length	returns length of the array
join(delimiter)	makes a string delimited with the items
pop()	removes the last and return it
push(item)	Add item to end
reverse()	reverses the order of items
shift()	removes first item and returns it
slice(start,[end])	returns a sub-array
sort(fn)	fn needs (a<b)==-1, (a==b)=0, (a>b)==1
unshift(item)	add item to start of array

https://developer.mozilla.org/en/JavaScript/Guide/Predefined_Core_Objects

Date Object

- Represents a date that allows computation
- Numeric value is expressed as millisecond

```
var d = new Date("May 8, 2013 17:30:00");
```

Full or 3-letter month

- `var d = new Date();`

New instance of **client's** current date and time

- **Methods** can be used to obtain values within the date object

```
var n = d.getDate();
```


Date Object - Some Methods

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)

String Object

- Wrapper for string primitive
- Properties - length
- Methods

charAt()	Returns the character at the specified index (position)
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a start position
toLowerCase()	Converts a string to lowercase letters

http://www.w3schools.com/js/js_strings.asp

JavaScript - Global Functions

Be careful of case

Function	Description
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string

JavaScript - Global Functions (continued)

Function	Example	Result
eval()	eval("2 + 3")	5
isFinite()	isFinite(5) isFinite("Web")	true false
isNaN()	isNaN(5) isNaN("Web")	false true
Number()	Number("22") Number("2 2")	22 NaN
parseFloat()	parseFloat("2") parseFloat("2.34") parseFloat("2 34") parseFloat("2 units) parseFloat("unit 2)	2 2.34 2 2 NaN

JavaScript - Global Functions (continued)

Function	Example	Result
parseInt()	parseInt("2") parseInt("2.34") parseInt("2 34") parseInt("2 units) parseInt("unit 2)	2 2 2 2 NaN
String()	String (0) String (true) String ("2")	0 true 2

" are not displayed

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- **Predefined Objects**
  - JavaScript Core Objects and Global Functions
  - **Browser Objects** – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Predefined Objects - Browser Objects

- Window
  - document
- Navigator
- Screen
- History
- Location

Examples

```
window.alert("Hello");
window.print();
var ans=confirm("Are you sure?")
```

document is the main object of the window object.
This will be discussed in detail later

Window Object – Properties

- The **`window`** object is at the top of the hierarchy, and so its properties and methods may be used without explicitly referring to the "window" object.
  eg. `document` is same as `window.document`

- **Properties**:
  `document` - returns a reference to the document contained in the window
  `location` - gets/sets the location, or current URL, of the window object
  `history` - returns a reference to the history object, an array of visited URLs
  `name` - gets/sets the window's name
  `navigator` - returns a reference to the navigator object
  `defaultStatus` - gets/sets the message in the status bar
  `status` - gets/sets the transient message in the status bar
  `self` - identifies the current window being referenced
  `parent` - identifies the window containing a particular window

Note: *This is **not** a complete list of properties! For more information see:*
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

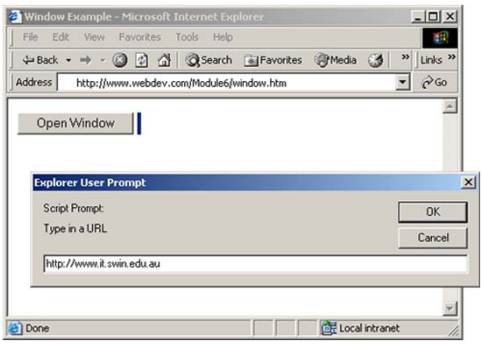
Window Object – Methods

- **Methods** *(this is **not** a complete list of methods)*
  `alert(text)` - pops up an alert box
  `confirm(text)` - pops up a box with 'OK' or 'Cancel'
  `prompt(text,def)` - retrieves a line of text from the user
  `open(url,[ops])` - opens up a new window
  `close()` - closes a window
  `focus()` - gives focus to a window
  `blur()` - removes focus from a window

- **Window HTML Event Handling**
  `onload` - occurs when the page has completed the loading process.
  `onunload` - occurs just before the document is cleared from the browser window. Usually used for background statistical purposes etc.

Window Object – Example

```
function newWindow() {
   theUrl = prompt("Type in a URL",
                                window.location);
   window.open(theUrl);
}
```



Navigator Object

- The **navigator** object does not fall within the normal Browser window object hierarchy.
  (It relates to the 'environment' in which the window sits)
- The **navigator** object ***may*** be used to gather information about the **client platform**. eg. if it has GPS
- The **navigator** object ***was*** often used to identify **browser dependent** features that a script may need to use.

```
if (navigator.appName == "Netscape") {
  // insert code here for Netscape
} else {
  // insert code here for other
  // browsers
}
```

Now best to use other DOM methods
http://www.w3.org/TR/html5/webappapis.html#the-navigator-object

Navigator Object – Properties/Methods

Properties

`appCodeName`	The coded name of the browser
`appName`	The name of the browser
`appVersion`	The version of the browser
`language`	The language supported by the browser
`mimeTypes[]`	An array of the MIME types recognised
`platform`	The platform the browser is running on
`plugins[]`	An array of the plugins installed

Many of these properties are superseded. See HTML5 spec., device guides.

http://www.w3.org/TR/html5/webappapis.html#the-navigator-object

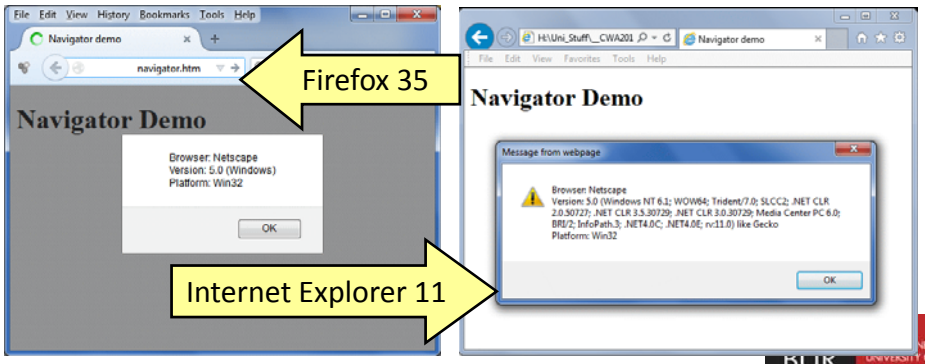
Methods

`javaEnabled()`	Returns true if the browser supports Java applets
`preferences()`	Checks or sets user preferences

Note: *Properties and Methods may differ between browsers.*

Navigator Object – Example

```
function showInfo() {
  var msg="Browser: " + navigator.appName +"\n";
  msg += "Version: " + navigator.appVersion + "\n";
  msg += "Platform: " + navigator.platform + "\n";
  alert(msg);
}
```



Other Browser Objects

history *.back(),*
.forward(),
.go(n)

Avoid using these. Changing them can confuse users.

location *.href,*
.host,
.pathname,
.protocol,
.search,
.reload([force]),
.replace(URL)

Useful for redirection, and for determining current webpage, so scripts can enhance menus by highlighting the current page.

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- **Predefined Objects**
  - JavaScript Core Objects and Global Functions
  - Browser Objects – window navigator
  - **Document Object Model**
    - **General DOM**
    - HTML objects
    - CSS objects
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Document Object Model (DOM)

- a platform and language neutral interface to allow programs and scripts to dynamically access and update the content, structure and style of a document [W3C]

  http://www.w3.org/DOM/
- a way to represent and navigate an HTML document or any XML document as a tree.

Note: The DOM Core applies to any XML, and any HTML that complies with XML.

DOM

- DOM is not part of core JavaScript, but JavaScript uses the DOM to interact with the Web browser. This technique is referred to as **DOM manipulation**
- DOM uses JavaScript's Core Objects *such as Array, Boolean, Date, Math, Number, RegExp, String, …*
- Current standard is DOM Level 3, 2004. Standard is relatively stable. http://www.w3.org/DOM/DOMTR

DOM Levels

- The W3C has developed DOM "levels" to represent the different features that may be supported
  - DOM Level 0: The earlier *"vendor specific* intermediate" DOMs
  - DOM Level 1: HTML & XML document tree structures, including HTML specific elements and node add / move / delete.
  - DOM Level 2: XML namespaces, styles, views, and events
  - **DOM Level 3:** Divided into specific modular sections
  - ***DOM Level 4:*** ***2014*** Aims at supporting mutimedia, and removing things that haven't been implemented http://www.w3.org/DOM/DOMTR

How well are the Core and HTML DOMs implemented in browsers?
http://quirksmode.org/dom/core/
http://quirksmode.org/dom/w3c_html.html

DOM Support

- As with HTML5, different browser provide various levels of support for DOM.
- W3C DOM Level 1 (rec. Oct 1998) and DOM Level 2 (rec. Nov 2000) are now largely supported by recent browsers.
- See what DOM your browser supports
  http://www.w3.org/2003/02/06-dom-support.html
- See the DOM compatibility tests
  http://www.quirksmode.org/compatibility.html

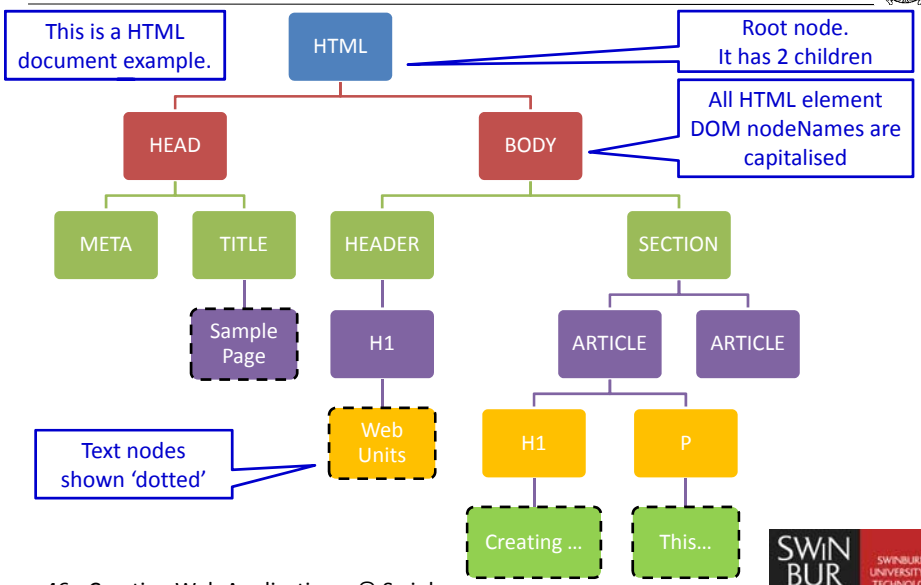
Document Object – Example

- A **document** is represented as a tree of nodes
- The first node is referred to as the **root node**
- Each node can have **children**
- A node with no children is referred to as **leaf node**

This example is a HTML document. But this applies to any XML document.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Sample Page</title>
</head>
<body>
  <header>
    <h1 id="pgHead">Web Units</h1>
  </header>
  <section>
     <article>
       <h1>Creating Web Apps</h1>
       <p>This unit covers … </p>
     </article>
     <article>
     </article>
  <section>
</body>
</html>
```


Document Object – Tree Structure



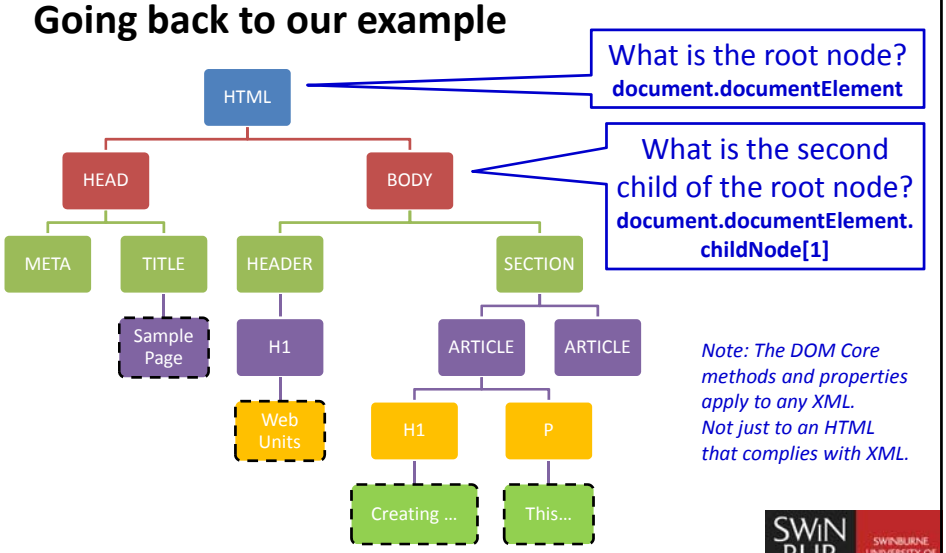
Document Object

Where are the objects?

- The entire HTML page is made up of **objects**
- Using the tree representation, each node is an **object.**
- In our example, we have 16 nodes or 16 objects
- We can use the **DOM Core** properties and methods to find out about these nodes

Document Object – Tree Structure

Going back to our example



Note: The DOM Core methods and properties apply to any XML. Not just to an HTML that complies with XML.

Document Object – Property/Method

- a **property** of document object is
  `documentElement`

  Object type

  contains the reference to the root node (or root element) of the document.

  e.g. `document.documentElement`
- a **method** of document object is
  `document.getElementById(<id>)`

  returns the reference to a specific node (also referred to as an element) using the ID attribute specified in the element. Sample use:

  e.g. `document.getElementById("pageHead")`

Document Object – Property/Method

- Some useful document properties and methods

`document.`

Pre-defined object

```
documentElement
getElementById()
getElementsByName()
getElementsByTagName()
createElement()
createTextNode()
createAttribute()
```

Document Object – as Node

- Use document property and method to obtain as node

```
node1 = document.documentElement;
node2 = document.getElementById("pgHead");
```

- What are some properties of a node?

```
node2.nodeName
node2.nodeValue
node2.nodeType
```

String type
String type
Number type

Document Object – as Node

- The **`nodeName`** property
  - specifies the name of a node
  - is *read-only*
  - of an **element** node is the same as the element name
  - of an **attribute** node is the attribute name
  - of a **text** node is always #text
  - of the **document** node is always #document

For HTML, nodeName always contains the *uppercase* element name of an HTML element.

Document Object – as Node

- The **`nodeValue`** property
  - specifies the value of a node.
  - for **element** nodes is undefined
  - for **text** nodes is the text itself
  - for **attribute** nodes is the attribute value
  - can be changed

Document Object – as Node

- The **nodeType** property returns the type of node.
  - nodeType is *read only*.
- The most important node types are:

Element Type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Document Object – as Node

Other node properties

`theNode.`

- `nodeType`
- `parentNode`
- `firstChild`
- `lastChild`
- `previousSibling`
- `nextSibling`
- `children[]`
- `childNodes[]`

`theNode`, shown here is just a sample object defined from the DOM

```
theNode = document.documentElement;
or
theNode = document.getElementById("pgHead");
```

[] are arrays
See later discussion on arrays

For example, `myNode.nodeType`

Document Object – as Element

Element properties

`objElement.`

- `id`
- `className`
- `tagName`
- `getElementsByTagName()`
- `getAttribute()`
- `setAttribute()`
- `removeAttribute()`

`objElement`, shown here is just a sample object defined from the DOM

```
objElement = document.documentElement;
```

or

```
objElement =
    document.getElementById("pgHead");
```

For example, `myElement.tagName`

Contents

Document Object Model and JavaScript

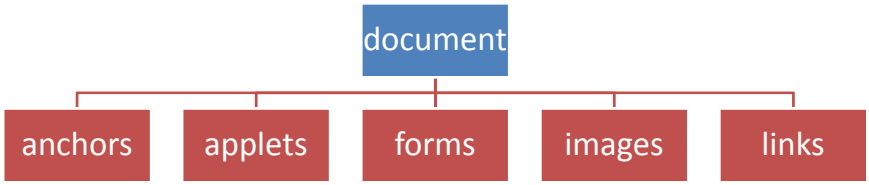
- JavaScript Objects .properties .methods()
- **Predefined Objects**
  - JavaScript Core Objects and Global Functions
  - Browser Objects – window navigator
  - **Document Object Model**
    - General DOM
    - **HTML objects**
    - CSS objects
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Document Object – as Element

- The following HTML elements have additional properties:
  - Links `<a …>…</a>`
  - Forms `<form …>…</form>`
  - Select / Option elements `<select …>… </select>`
  - Input (text, radio, checkbox, password, hidden, submit) `<input … />`
  - Textarea `<textarea… >… </textarea>`
  - Images `<img … />`

Predefined Objects - Document Object

HTML document object and its array objects



- These are collections of specific objects, e.g. form**s** is a collection of form objects.

Note array names (collections) are often expressed in plural form

Document Object - Anchor Element

Anchor Element `<a ></a>`

```
objElement.
      href
      rel
      target
```

For example, `myAnchor.href`

Document Object - Form Element

Form Element `<form …>…</form>`

```
objElement.
      elements[]
      length
      action
      method
      enctype
      target
      submit()
      reset()
```

An array of all the elements in the form
(See later discussion on arrays)

For example, `myForm.length`

Document Object - Select Element

Select Element `<select …>…</select>`

```
objElement.
      type              disabled
      selectedIndex     multiple
      value             name
      length            size
      form              add()
      options[]         remove() …
```

For example, `mySelect.value`

Document Object - Option Element

Option Element `<option …>…</option>`

```
objElement.
    form
    text
    disabled
    selected
    value, …
```

For example, `myOption.text`

Document Object - Input Element

Input Element `<input … />`

```
objElement.
    form            readOnly
    checked         value
    disabled        select()
    name            click(), …
```

For example, `myInput.checked`

Document Object - Textarea Element

Text Area Element `<textarea …>…</textarea>`

```
objElement.
    form
    disabled
    name
    readOnly
    value
    select(), …
```

For example, `myTextArea.value`

Document Object - Image Element

Image Element `<img … />`

```
objElement.
    name
    src
    alt …
```

For example, `myImage.src`

Document Object - Examples

- Get the body element
  (get all tags named "body")

```
var bodyElements =
  document.getElementsByTagName("body");
```

- Get all images from the body element

```
var imgElements =
  bodyElement.getElementsByTagName("img");
```

Will return a collection/array. Use a **plural** object name to indicate multiple elements

Document Object - Examples

- Get the element with **id="intro"**

```
var introElement =
  document.getElementById("intro");
```

Use a use **singular** object name to indicate 1 element

- Get all **<p>** elements that are descendants of the element with **id="main"**

```
var mainParagraphElements =
  document.getElementById("main")
          .getElementsByTagName("p");
```

Will return a collection/array. Use a **plural** object name to indicate multiple elements

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- **Predefined Objects**
  - JavaScript Core Objects and Global Functions
  - Browser Objects – window navigator
  - **Document Object Model**
    - General DOM
    - HTML objects
    - **CSS objects**
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Document Object (Class and Style)

- Usually element ***attribute names*** are directly matched to DOM property names.
  For example the **`href`** attribute
  `<a href="page1.htm" class="button">`
  is mapped to `objElement.href`

- But the the **`class`** attribute
  is mapped to `objElement.className`
  NOT TO `".class"` as "class" is
  a ***reserved word*** in JavaScript

Document Object (Class and Style)

- **`class`** is often used to associate style with elements. If we change the class in JavaScript, the browser changes the associated presentation

```
objElement.className = "styleRule2";
```

- **Style** properties are typically hyphenated words, **but this *does not work* in JavaScript**, so CSS style properties are joined together using 'camelBack' notation. e.g.

`some-css-property` becomes `someCssProperty`

Document Object (Class and Style)

- **`objElement.style.`**

```
background
backgroundAttachment
backgroundColor
backgroundImage
backgroundPosition
backgroundPositionX
backgroundPositionY
backgroundRepeat
```

```
border
borderCollapse
borderColor
borderSpacing
borderSpacing
borderStyle
border[side]
border[side]Color
border[side]Style
border[side]Width
```

For example,

```
objElement.style.display
```

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- Predefined Objects
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
  - JavaScript Core Objects and Global Functions
- **Using JavaScript**
  - **Checking Form Data: *an Example***
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- Predefined Objects
  - JavaScript Core Objects and Global Functions
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- **Using JavaScript**
  - Checking Form Data: *an Example*
  - **Image Manipulation: *an Example***
- Storing 'State'
  - Web Storage
  - Cookies

Content and JavaScript

JavaScript can enrich user experiences by changing content and providing:

- slideshows,
- **cycling images,**
- 'drag and drop' interfaces,
- re-sorting / re-displaying page information,
- hiding /showing page information,

… and lots more …

Example: Cycling images for Weather Radar at: http://www.bom.gov.au/products/IDR024.loop.shtml

Cycling Images - Example

Given the following HTML page segment, *take note of the **IDs***

```
<article>
   <h3>Cycling an image</h3>
   <!– html5 figure and figcaption elements
       could have been used instead -->
  <p>
  <img src="pic1.jpg" id="picImage"
   alt="Native Flowers" width="190"
   height="190" />
   </p>
  <p id="picText"></p>
</article>
```

Cycling Images - Example (continued)

Using the JavaScript template:

```
function cycleImage() {
  var figImg =
   document.getElementById("picImage");
  var figCap =
   document.getElementById("picText");
  /* more code here */
}
function init() {
   cycleImage(); }

window.onload = init;
```

image ID

image text ID

Cycle function is called on page load event

This is fixed

Cycling Images - Example (continued)

```
  var currentImg = 0;      // set start position as global
function cycleImage() {
  var theImages = new Array("img1.jpg","img2.jpg","img3.jpg");
  var theTexts  = new Array("text1","text2","text3");
  var numImgs = theImages.length;
  var figImg = document.getElementById("picImage");
  var figCap = document.getElementById("picText");
  if(document.images) {
      currentImg++;
      if (currentImg == numImgs) {
          currentImg = 0;    // reset start position
      }
      figImg.src = theImages[currentImg];
      figCap.textContent = theTexts[currentImg];
      setTimeout("cycleImage()", 1000);
   }
}
```

setTimeout() is a pre-defined browser window function

cycleImage() function calls itself in a time sequence, changing **figImg.src** every 1000 milliseconds

Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- Predefined Objects
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
  - JavaScript Core Objects and Global Functions
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- **Storing 'State'**
  - **Web Storage**
  - **Cookies**

Also see :Extra Notes: Sessions

Web Storage

Web storage

- allows HTML5 web pages to store data ***locally*** *within the browser*
- is a separate specification http://www.w3.org/TR/webstorage/
- stores data in key/value pairs
- is more secure and faster compared to cookies *(data is not included as part of the HTTP header)*
- can only be used to access data by the webpage that created it
- allows the storage of a *large amounts* of data *(at least 5mb per origin depending on browser)*
- can only by accessed by client scripts

Web Storage (continued)

Two objects for storing data

- **localStorage**
  - stores data with no expiration, even when the browser is closed
- **sessionStorage**
  - stores data for one session, defined by the lifetime of the current window.

Web Storage (continued)

Can check if Web Storage is supported

```
if(typeof(Storage)!=="undefined"){
 // localStorage and
      sessionStorage supported

 }else {
   // No web storage supported.
 }
```

Web Storage (continued)

Setting and reading localStorage

```
// Store value on the browser
localStorage.setItem('key', 'value');

// Retrieve value, even after
re-opening browser
var a = localStorage.getItem('key');
```


Web Storage (continued)

Setting and reading sessionStorage

```
// Store value on browser only for the session
sessionStorage.setItem('key', 'value');

// Retrieve value for the session
var a = sessionStorage.getItem('key');
```

Cookies

- A Cookie is a variable that contains ***a small piece of information*** that can be passed by a **web server** to the client **browser**.
- This variable is ***stored in the client machine*** through the browser.
- The browser may chose not to accept a cookie
- A Cookie:
  - is stored as plain text record (maximum of 4Kb)
  - can be accessed by client and sent back with **HTTP Request** to **web server**
- **Reference:** https://developer.mozilla.org/en-US/docs/DOM/document.cookie

Cookies (continued)

The text record consists of the following variable-length fields:

- **name=*value*** pair used to set cookies
- **domain=*hostName*** is the domain name where the cookie can be used.
- **path=*directoryPath*** is the path to the directory where the cookie can be used.
  This is usually the path to the web page that set the cookie. Webpages from a different directory can access the cookie if left blank.

Cookies (continued)

… Cookie text record continued

- **expires=*stringDate*** is the date when the cookie will expire. If blank, the cookie will expire when browser is closed.
- **secure** is use to restrict the retrieval of the cookie from a secure server. If left blank, no such restriction exists.

Cookies – Checking

Can check if Cookies are enabled

```
if(navigator.cookieEnabled)){
  // cookies enabled

}else {
  // cookies disabled
}
```

Cookies – Setting

Syntax to manage cookies

```
document.cookie = "field=value;";
```

Document object

Setting field values

Note:
Cookie *values* may not include semicolons, commas, or whitespace, use the JavaScript escape() and unescape() functions to encode and decode the value respectively

Cookies – Setting (continued)

Setting a cookie record with no expiration:

```
document.cookie =
 "lname=Smith;fname=Jack;"
```

Setting a cookie record with expiration (session)

```
now = new Date();
document.cookie =
 "lname=Smith;fname=Jack; expires="
 + now.toUTCString()
 + ";domain=.swinburne.edu.au;
    path=/;secure;"
```

Cookies – Setting (continued)

Wrong way, there are 6 Cookie records here

```
document.cookie = "lname=Smith;";
document.cookie = "fname=Jack;";
document.cookie = "expires=" +
        now.toUTCString() + ";"
document.cookie =
        "domain=.swinburne.edu.au;"
document.cookie = "path=/;"
document.cookie = "secure;"
```


Cookies – Deleting

Setting expiration date (deleting a cookie)

```
expireDate = new Date();
expireDate.setTime(expireDate.getTime()
     + 3600000*24* ______);
```

Replace with – to delete cookies

Replace with number of days

```
document.cookie = "key=value;expires=" +
     expireDate.toUTCString() + ";"
```

Cookies – Reading

```
// Get all the cookies pairs
var allCookies = document.cookie;
// Split each pair as an element in an array
cookieArray  = allCookies.split(';');
// Access each pair as an element
for(var i=0; i<cookieArray.length; i++){
  // split each element into name and value
  name = cookieArray[i].split('=')[0];
  value = cookieArray[i].split('=')[1];
  alert("Key is " + name +
        " and value is " + value);
}
```


Contents

Document Object Model and JavaScript

- JavaScript Objects .properties .methods()
- Predefined Objects
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
  - JavaScript Core Objects and Global Functions
- Using JavaScript
  - Checking Form Data: *an Example*
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies

Contents

- JavaScript
  - JavaScript Objects
  - Predefined Objects
    - Browser Objects
    - Document Object Model
    - JavaScript Core Objects and Global Functions
  - Checking Form Data: An Example
  - Image Manipulation: An Example
  - Storing 'State':
    - Web Storage
    - Cookies

Next Lecture

What's Next?
- Introduction to Server-Side Processing