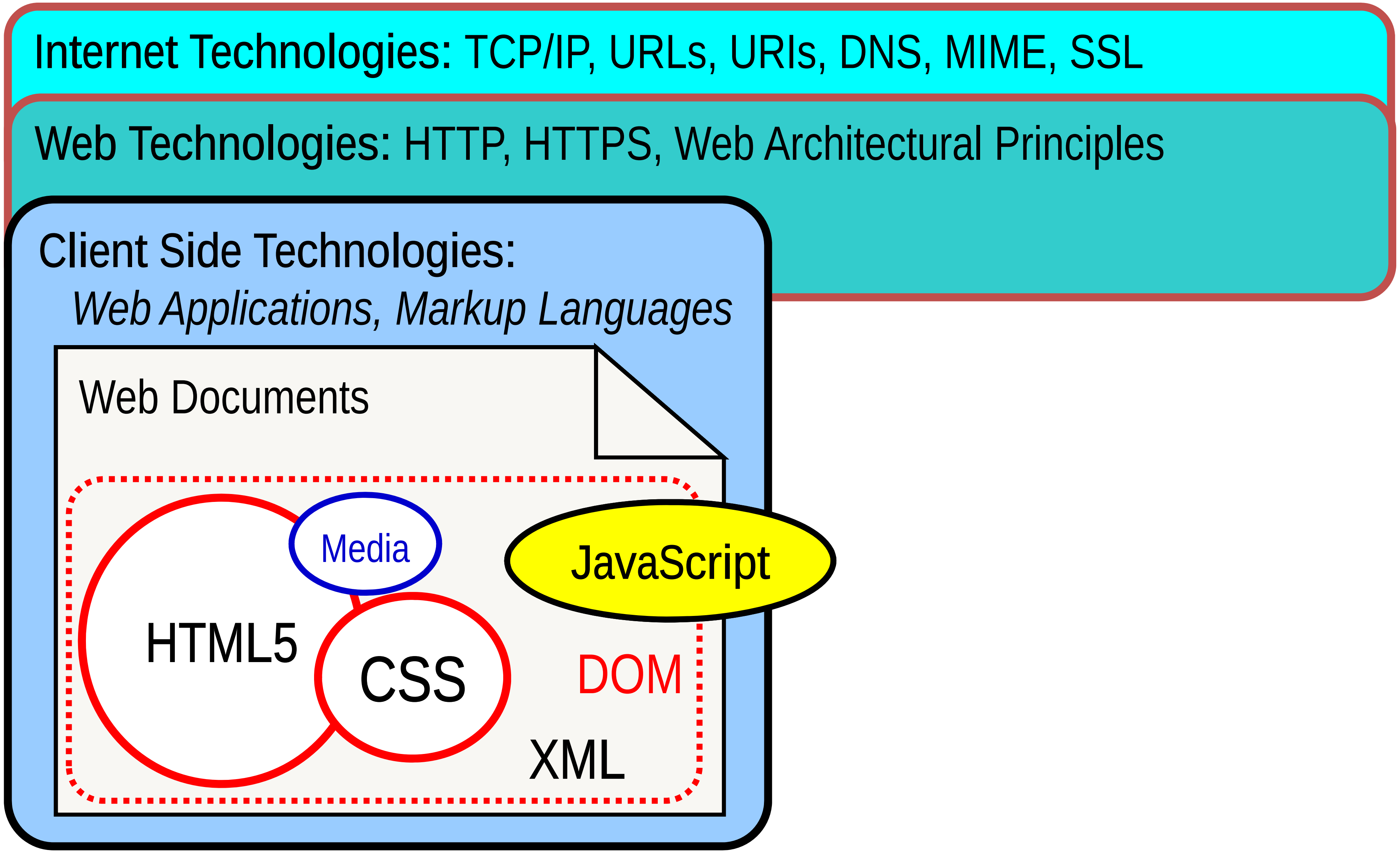
# COS10011
# Creating Web Applications

## Lecture 7 – Document Object Model (DOM)

*Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)*

## Assignment 2 out now!

# 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*
Web Documents
Media
HTML5
CSS
JavaScript
DOM
XML

# 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>
```

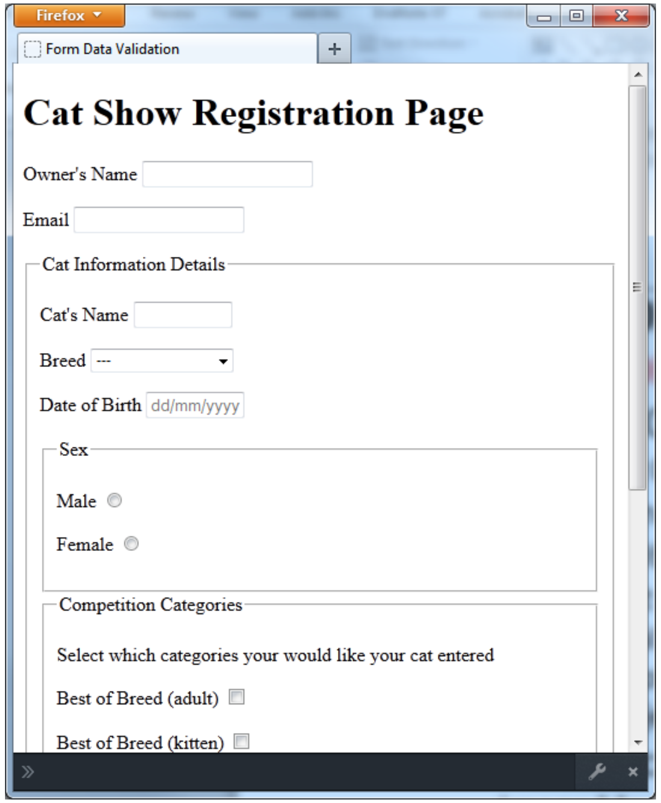
JavaScript - *behaviour*

```
/* 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;
```

## Previously – Form Validation

- Regular Expressions
- Input data validation using JavaScript

Demo



Cat Show Registration Page

Owner's Name

Email

Cat Information Details

Cat's Name

Breed ---

Date of Birth dd/mm/yyyy

Sex

Male

Female

Competition Categories

Select which categories your would like your cat entered

Best of Breed (adult)

Best of Breed (kitten)

## Contents

### Document Object Model and JavaScript

- Predefined Objects
  - **JavaScript Core Objects – examples Array, Date, String**
  - **Global Functions**
  - Browser Objects – window  navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

**Previous lecture**

## Contents

### Document Object Model and JavaScript

- Predefined Objects
  - Browser Objects – window  navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing ‘State’
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

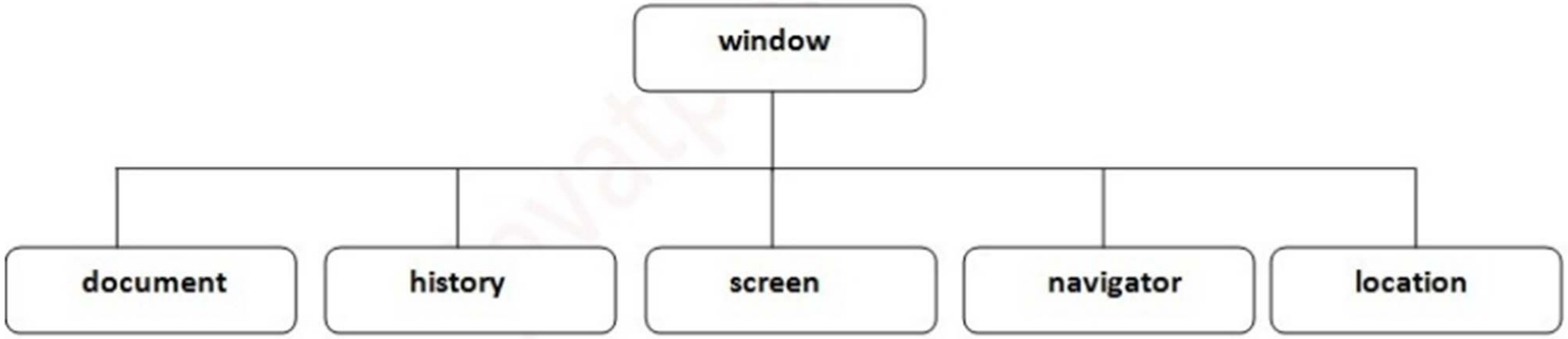
# Predefined Objects - Browser Objects

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

**Examples**

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

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



https://www.javatpoint.com/browser-object-model

# 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`
- Represents an open window in a browser. Created automatically by the browser
- **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` | - sets or returns the text in the statusbar of a window |
| `self` | - identifies the current window being referenced |
| `parent` | - returns the parent window of the current 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)*

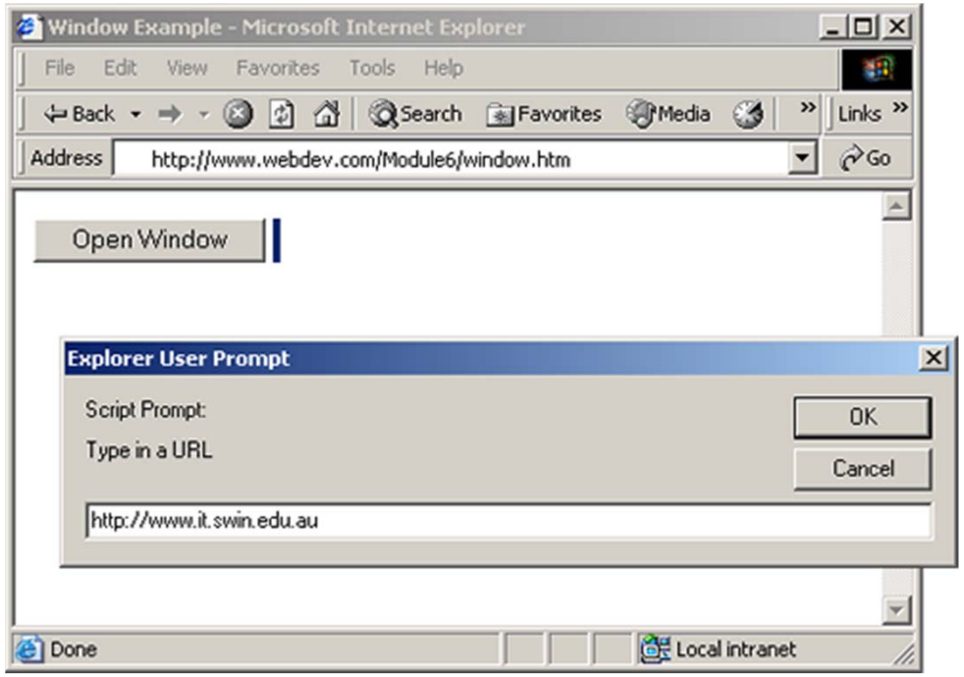
| | |
|---|---|
| `alert(`*`text`*`)` | - pops up an alert box with ok button |
| `confirm(`*`text`*`)` | - pops up a box with ‘OK’ or ‘Cancel’ |
| `prompt(`*`text,def`*`)` | - retrieves a line of text from the user |
| `open(`*`url, _blank`*`)` | - URL is loaded into a new window |
| | `Other options: _parent, _self` |
| `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);
}
```


Window Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media Links
Address http://www.webdev.com/Module6/window.htm Go
Open Window
Explorer User Prompt
OK
Cancel
Done
Local intranet

Script Prompt:
Type in a URL

http://www.it.swin.edu.au

## 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)
- Contains information about the browser
- 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*

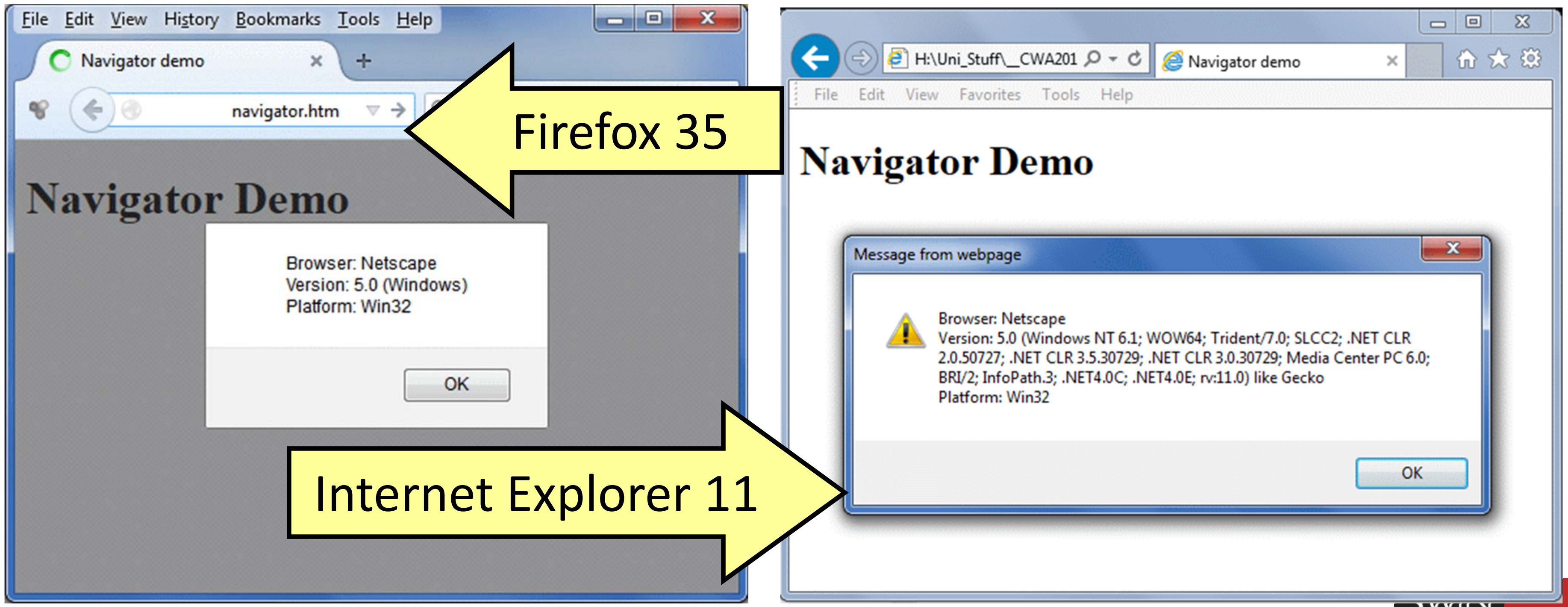
## Methods

| | |
|---|---|
| `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);
}
```



```
Browser: Netscape
Version: 5.0 (Windows)
Platform: Win32
```

```
Browser: Netscape
Version: 5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0;
BRI/2; InfoPath.3; .NET4.0C; .NET4.0E; rv:11.0) like Gecko
Platform: Win32
```

## Other Browser Objects

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

Avoid using these. Changing them can confuse users.

**`location`** `.href`,
`.host`, (Sets or returns the hostname and port number of a *URL*)
• `pathname`,
`.protocol`,
`.search`,
`.reload([`*`force`*`])`,
`.replace(URL)`

contains information about the current URL

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

# Other Browser Objects

## Screen

- contains information about the visitor's screen

| Property | Description |
|---|---|
| availHeight | Returns the height of the screen (excluding the Windows Taskbar) |
| availWidth | Returns the width of the screen (excluding the Windows Taskbar) |
| colorDepth | Returns the bit depth of the color palette for displaying images |
| height | Returns the total height of the screen |
| pixelDepth | Returns the color resolution (in bits per pixel) of the screen |
| width | Returns the total width of the screen |

https://www.w3schools.com/jsref/obj_screen.asp

# Contents

## Document Object Model and JavaScript

- Predefined Objects
  - Browser Objects – window  navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

## 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/
- treats an HTML, XHTML, or XML document as a tree structure

  *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

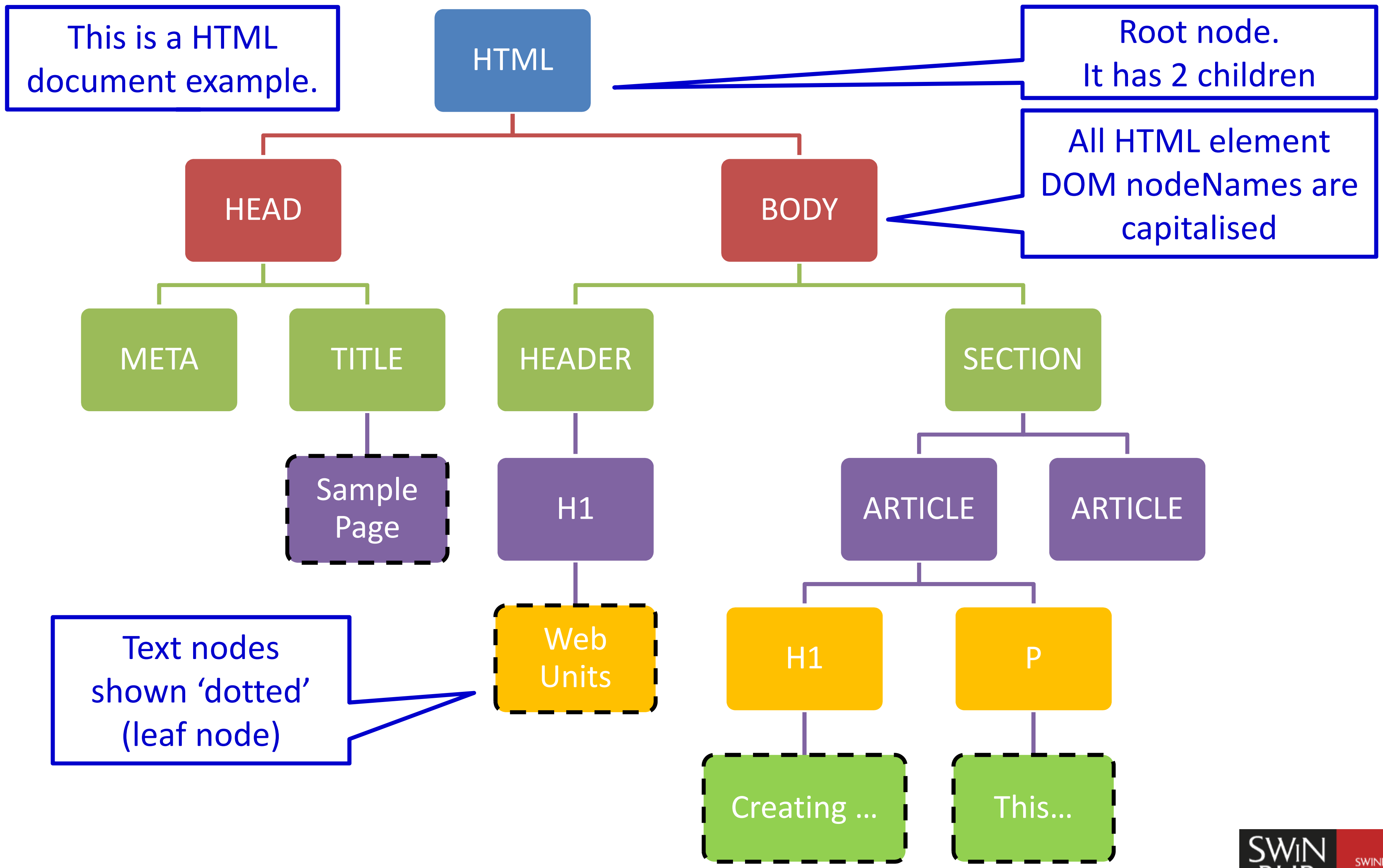
## Document Object – Example

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

- 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**

```
<!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


This is a HTML document example.
HTML
Root node. It has 2 children
HEAD
BODY
All HTML element DOM nodeNames are capitalised
META
TITLE
HEADER
SECTION
Sample Page
H1
ARTICLE
ARTICLE
Web Units
H1
P
Text nodes shown ‘dotted’ (leaf node)
Creating …
This…

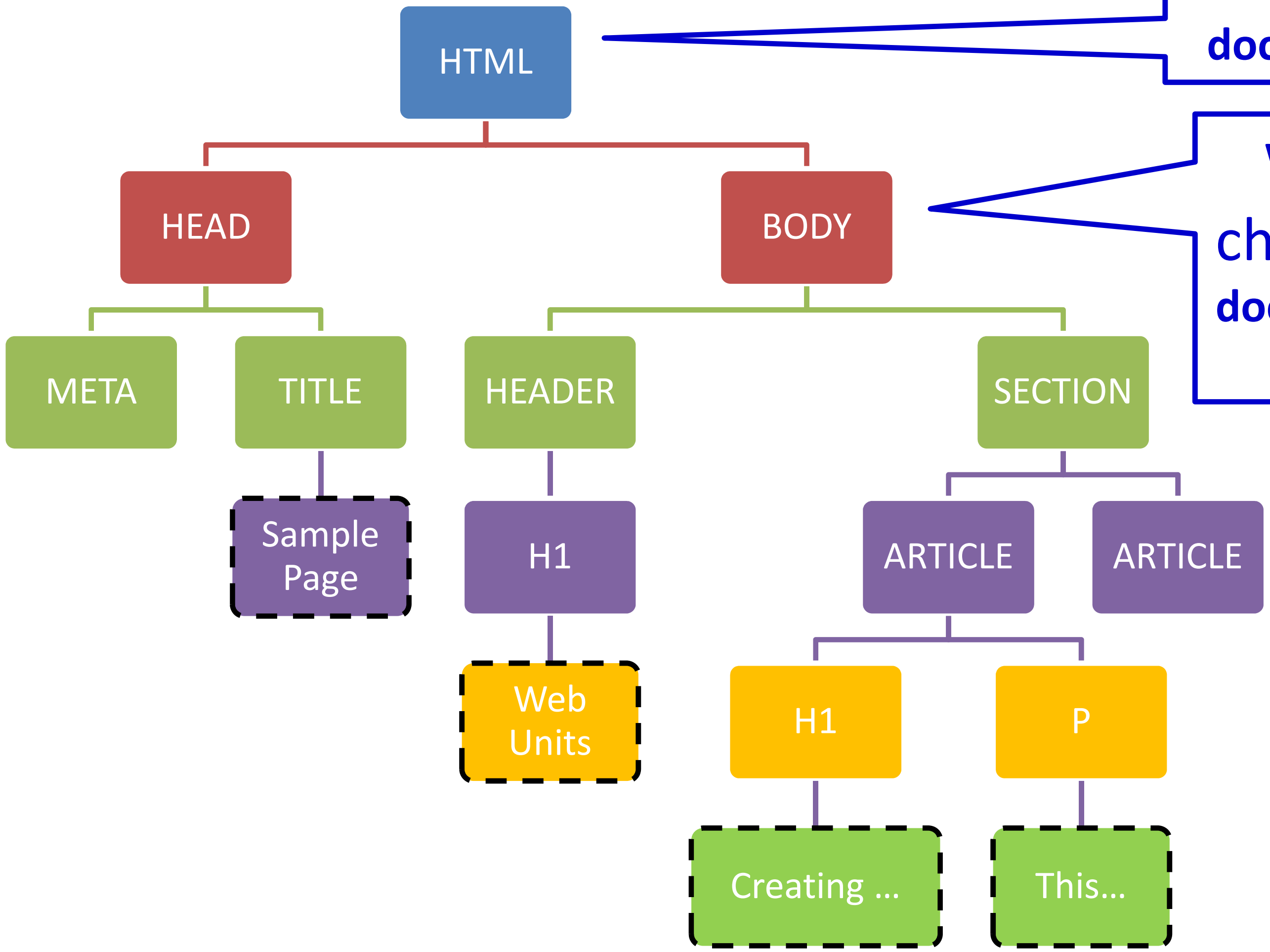
## Document Object

### **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



What is the root node?
**document.documentElement**

What is the second child of the root node?
**document.documentElement.childNode[1]**

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

# Document Object – Property/Method

- Some useful document properties and methods

**document.** Pre-defined object

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

Referring to the root

## From our demo ....

```
function isCategorySelected(){
      ...
      var categories =
         document.getElementById("categories").getElementsByTagName("input");
      var labels =
          document.getElementById("categories").getElementsByTagName("label");
      var label = "";
      var catList = "";
      for (i=0; i<categories.length; i++){                    //for each category element
            selected = selected || categories[i].checked;         //see if it is checked
            label = labels[i].firstChild.textContent;                   //get its label
            catList = catList + label + "\n";
      }
      ...
}
```

text node content

```
<fieldset id="categories">
            <legend>Competition Categories</legend>
            <p>Select which categories your would like your cat entered</p>
            <p><label for="bestbreed">Best of Breed (adult)</label>
                        <input type="checkbox" id="bestbreed" name="categories[] " value="best"/>
            </p>
            <p><label for="kit">Best of Breed (kitten)</label>
                        <input type="checkbox" id="kit" name="categories[]" value="kitten"/>
            </p>
```

## Document Object – as Node

- Use document property and method to obtain as node

```
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.

```
<p id="myP">Click the button to get the node name of this element.</p>
```

```
document.getElementById("myP").nodeName;
```

Will give a ‘P’

# 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

```
<p id="myP">Click the button to get the node name of this element.</p>
```

```
Var node = document.getElementById("myP").childNodes[0];
node.nodeValue
```

# 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

- More examples:

```
<div id="myDIV">This is a div element.</div>
<button onclick="myFunction()">Try it</button>
```

```
<script>
function myFunction() {
    var x = document.getElementById("myDIV").firstChild;
    var txt1 = "The node name: " + x.nodeName ;
    var txt2 = "The node value: " + x.nodeValue ;
    var txt3 = "The node type: " + x.nodeType;
}
</script>
```

```
The node name: #text
The node value: This is a div element.
The node type: 3
```

## Document Object – as Node

Other node properties

```
theNode.
        nodeType
        parentNode
        firstChild
        lastChild
        previousSibling
        nextSibling
        children[]      //contain only element nodes
        childNodes[]  //contain all nodes, including text nodes
                        and comment nodes
```

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

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

ref demo example

For example, `myNode.nodeType`

## Document Object – as Node

- Create a node

```
<div id="div1">
   <p id="p1">This is a paragraph.</p>
   <p id="p2">This is another paragraph.</p>
</div>
```

```
<script>
   var para = document.createElement("p");
   var node = document.createTextNode("This is new.");
   para.appendChild(node);

   var element = document.getElementById("div1");
   var child = document.getElementById("p1");
   element.insertBefore(para, child);
</script>
```

Output:
This is new.
This is a paragraph.
This is another paragraph.

## Contents

### Document Object Model and JavaScript

- Predefined Objects
  - Browser Objects – window navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

# Document Object – as Element

Element is one of node types

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;
```

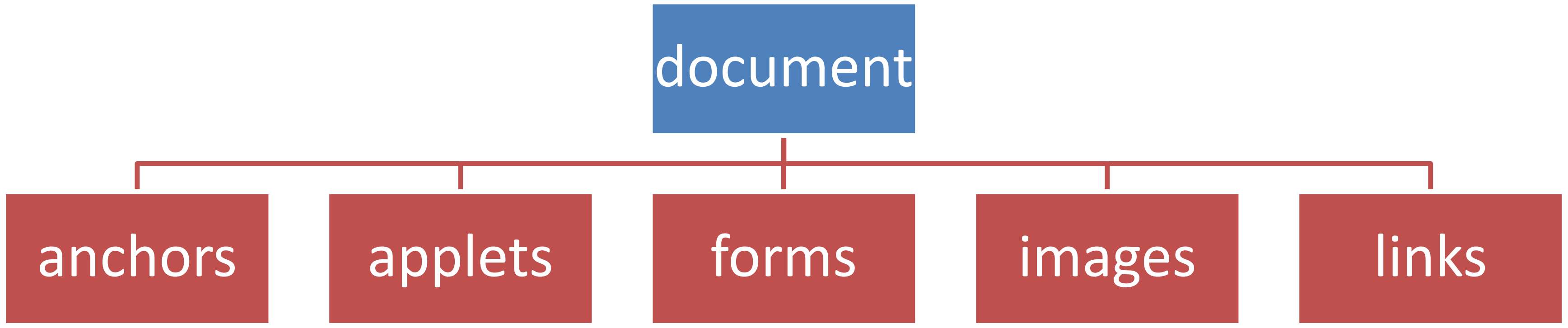
or

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

For example, `myElement.tagName`

# 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

## From our demo ….

```
//register onblur events for all the input elements
function registerInputsOnBlur(){
        var inputElements =
                document.getElementById("regForm")
                .getElementsByTagName("input");
    for (var i = 0; i < inputElements.length; i++){
        inputElements[i].onblur = validateInputOnBlur;
    }
}
function validateInputOnBlur(){
    var objectLostFocus_id = this.id;
    var isOk = false;
    switch (objectLostFocus_id){
        case "owner":
        …
```

The keyword "**this**" refers to the object which fire (trigger) the checkdata function (method). You can use if statement to check the "id" value and perform corresponding tests

## 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 … />`

# 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

For example, `myForm.length`

# Document Object - Select Element

**Select Element** `<select …>…</select>`

`objElement.`

- `type`
- `selectedIndex`
- `value`
- `length`

  Returns the number of <option> elements in a drop-down list
- `form`
- `options[]`
- `disabled`
- `multiple`
- `name`
- `size`

  Sets the size of a drop-down list
- `add()`
- `remove()` …

For example, `mySelect.value`

## Document Object - Option Element

**Option Element** `<option …>…</option>`

`objElement.`

> `Form` (Returns a reference to the form that contains the option)
>
> `text` (Sets or returns the text of an option)
>
> `disabled`
>
> `selected` (Sets the selected state of an option)
>
> `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 all images from the body element

```
var imgElements =
 document.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

- Predefined Objects
  - Browser Objects – window  navigator
  - Document Object Model
    - General DOM
    - HTML objects
    - CSS objects
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing 'State'
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

# Document Object (Style)

From our demo ....

```
function chkOwnerName () {
    //check owner name valid
    var owner = document.getElementById("owner").value;
    …
     //highlight the textbox if not valid
    if (!nameOk){
    document.getElementById("owner").style.borderColor = "red";
    }
    return  nameOk;
}
```

???? *border-color* ????

## Document Object (Style)

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

  **`some-css-property`** becomes

  **`someCssProperty`**

# 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";
```

- 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 `class` attribute is mapped to `objElement.className`

  **NOT `".class"`** as "class" is
  a ***reserved word*** in JavaScript

## 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
- Using JavaScript
  - Image Manipulation: *an Example*
- Storing ‘State’
  - Web Storage
  - Cookies
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

## 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 …*

## 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) {       //returns a collection of all <img> elements
       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

- Predefined Objects
  - 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                    Also see: Extra Notes: Sessions
- Multiple files
  - One HTML : many JS
  - One JS : many HTML

## 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.
  - Data is lost when the browser tab is closed

## 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 sessionStorage**

Store value on browser only for the session

```
sessionStorage.setItem('key', 'value');
```

***Examples***

sessionStorage.uname = document.getElementById("username").value;

**or**

sessionStorage.setItem("uname","username");

Retrieve value for the session

```
var a = sessionStorage.getItem('key');
```

***Examples***

var a = sessionStorage. uname;

**or**

var a =sessionStorage.getItem("uname");

## Web Storage (continued)

**Setting and reading localStorage**

Store value on the browser

```
localStorage.setItem('key', 'value');
```

**or**

```
localStorage.key = 'value';
```

Retrieve value, even after re-opening browser

```
var a = localStorage.getItem('key');
```

**or**

```
var a = localStorage.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:

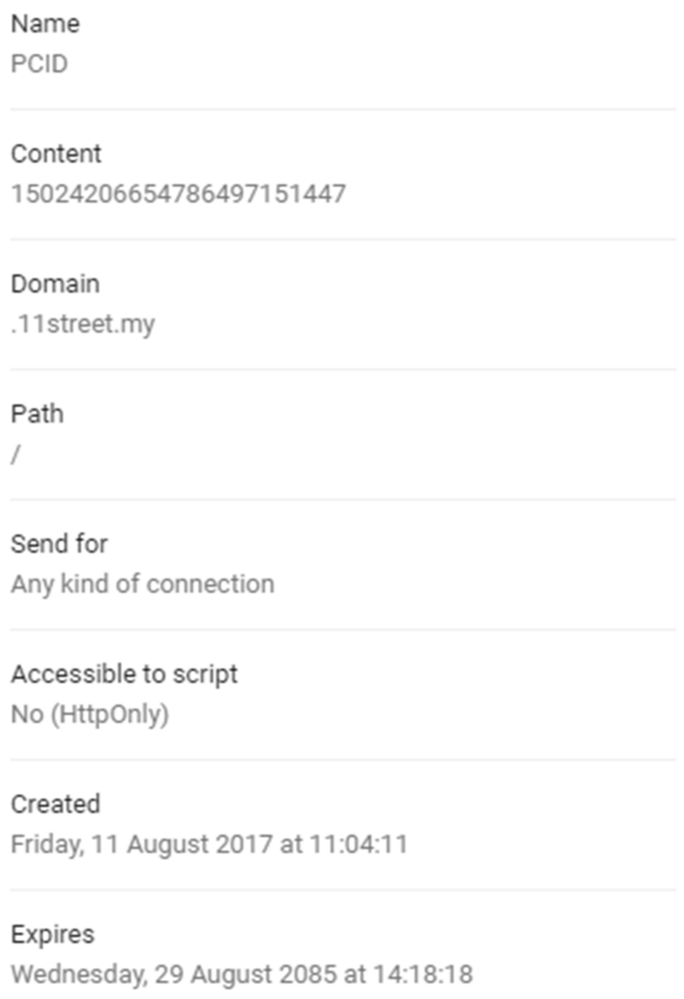
- **name=*value*** pair used to set cookies
- **domain=*hostName*** is the domain name where the cookie can be used.
- **path=*directoryPath*** is the path to the directory where the cookie can be used.
  This is usually the path to the web page that set the cookie. Webpages from a different directory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

# Examples of cookie


Name
PCID

Content
150242066547864971514447

Domain
.11street.my

Path
/

Send for
Any kind of connection

Accessible to script
No (HttpOnly)

Created
Friday, 11 August 2017 at 11:04:11

Expires
Wednesday, 29 August 2085 at 14:18:18

## 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

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

Also see :Extra Notes: Sessions

- **Multiple files**
  - **One HTML : many JS**
  - **One JS : many HTML – See demo**

SWINBURNE UNIVERSITY OF TECHNOLOGY

# COS10011 Creating Web Applications

## What's Next? - Introduction to Server-Side Processing (PHP)

*Acknowledgement: The contents in this document was adopted from learning material prepared by Alan Colman (Convener, SUT)*