Session 7

JavaScript
Part 2

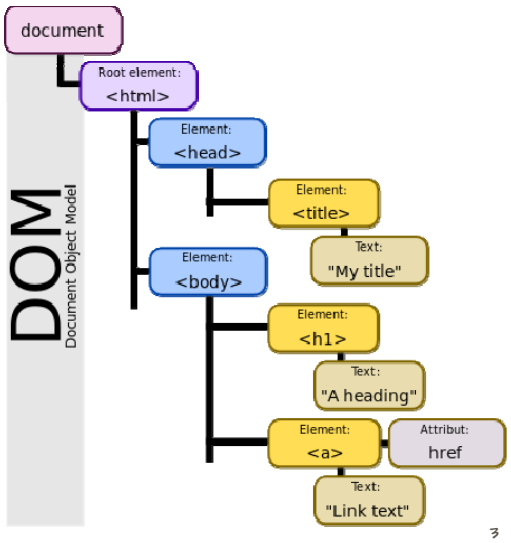
W3C DOM Reading and Reference

- Background and introduction
  - developer.mozilla.org/en-US/docs/DOM/DOM_Reference/Introduction
  - en.wikipedia.org/wiki/Document_Object_Model
  - www.w3schools.com/js/js_htmldom.asp
- Reference:
  - JavaScript DOM properties - Flanagan book (Chapter 15)
  - DOM Reference
    - developer.mozilla.org/en-US/docs/DOM/DOM_Reference

Use the HTML Interfaces

Learning Goals

- Understand the Document Object Model
- Understand how to perform client side form validation
- Understand the JavaScript event model



3

Access to the Document

- JavaScript begins to be useful when you can access and modify the html in the document

"DOM" can mean different things

- Approaches
  - Legacy DOM (Document Object Model) - Defined by Netscape in the early days of the WWW
  - DOM Level 3
    - well supported on modern browsers
    - Includes the legacy DOM (known as Level 0 DOM)
  - DOM Level 4

You may see the use of many of the supported DOMs in current code

4

What is DOM?

- Document Object Model
- Convention for representing and interacting with HTML, XHTML, and XML documents as a tree structure
- Cross platform
- Binding with various languages
- Implemented as an API in JavaScript

Currently being developed by the WHATWG (Web Hypertext Application Technology Working Group)

5

Legacy DOM

- Does not take full advantage of the tree structure of html documents
- Tends to reference html elements as members of an array, for example images[], links[] and forms[]
- Naming
  - document.forms[0]
  - document.forms.f1
  - document.forms["f1"]

```
<form name="f1">
```

Assuming the order of elements in an html document can cause maintenance problems

6

W3C DOM

- Defines
  - a standard set of objects (object tree) for an html document
  - Set of methods (language independent) to access the html object
- Your Java and JavaScript (and other) programs can
  - Access a given node (element)
  - Walk the tree
  - Search for particular nodes or data (e.g., img tags)
  - Modify the nodes and insert sub-trees

W3C was moving too slowly for browser vendors, so W3C stepped aside around 2004, and deferred to the WHATWG

7

JavaScript/DOM

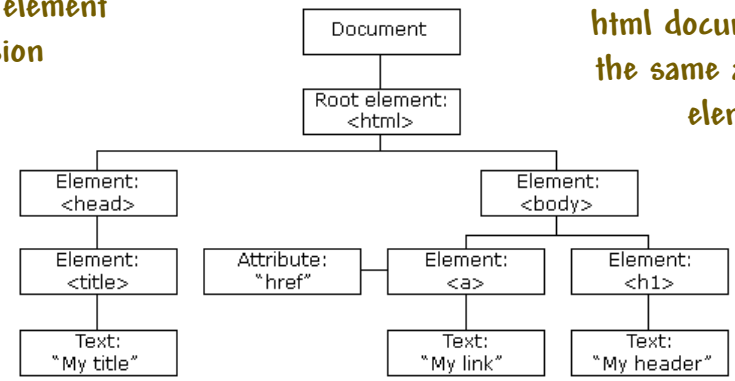
- When a web page is loaded, the browser creates a Document Object Model of the page
- With the object model, JavaScript is fully enabled to create dynamic HTML:
  - JavaScript can add, change, and remove all the HTML elements and attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can react to all existing events in the page
  - JavaScript can create new events in the page

From WIkipediaa

DOM Access to html

This should clarify the tag vs. element discussion

Note that the root of the html document is not the same as the root element



9

Node Object

- HTML elements are of type Node/Element/HTMLElement (inheritance hierarchy)
- You can get a handle to a node, and modify its appearance
- Methods of Document can return
  - An Element object (e.g., getElementById)
  - A NodeList object (e.g., getElementsByTagName)

Since DOM is language independent, it includes its own data structure types

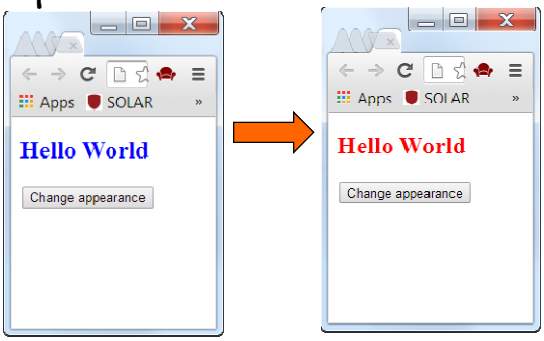
Notice whether the method uses singular or plural

10

Example

- Illustrates
  - Response to an event
  - Modification of the style property of a node
- Actions
  - Obtain a handle to an html element
  - Modify the html element

Clicking the button changes the page appearance



11

Example - Changing Styles

- An easy way to change the appearance of an element is to change its class attribute

```
...
<style type="text/css">
      .blue {color:blue;}
      .red {color:red;}
</style>
<script>
      function change() {
      var y = document.getElementById("X4");
      y.className="red"; }
</script>
...
<p id="X4" class="blue" >Hello World</p>
<p> <input type="button" onClick="change()"
value="Change appearance" > </p>
```

"class" is a reserved name in JavaScript, so the class property is "className"

y is a Node object

HTMLElement is a subclass of Element

DOM-HelloClassName

12

Example - Alternate Approach

```
<head>
...
  <script>
      function change() {
      var y = document.getElementById("X1");
      y.style.color="red"; }
  </script>
</head>
<body style="color:blue;">
  <form method="get" action="HelloDOM" >
  <h2 id="X1" style="color:blue;">Hello World</h2>
  <input type="button" onclick="change()"
      value="Change appearance" />
  </form>
</body>
```

You can directly set the style, but a css style sheet is preferred

Attributes are usually set as properties

Level 2 CSS2Properties object

Clicking the button invokes the change() function

13

CSS2Properties Object

- Convenience mechanism
- The style property of the Node object is of type CSS2Properties

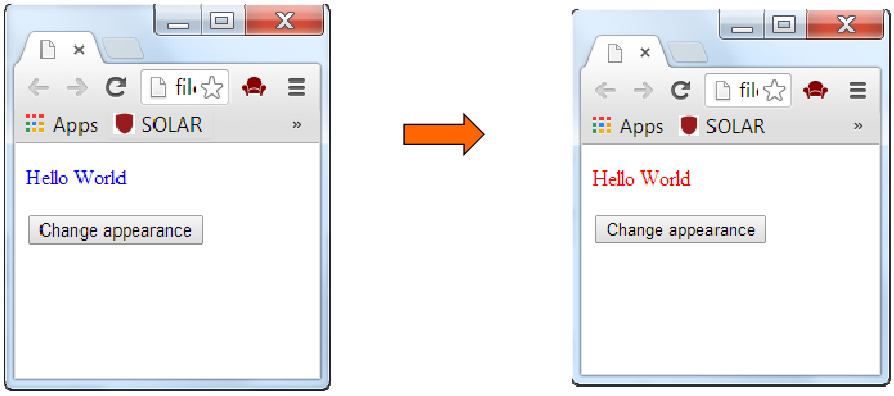
```
p.style.color="red"; }
```

- The CSS2Properties object refers to the inline styles of the element (not from the style sheet)
- Property values are strings
- Units are required
- Property names are similar to CSS property names, except where it interferes with JavaScript naming (e.g., font-family => fontFamily)

14

Example

- Illustrates access to an array of elements



15

Example - Access Elements By Name

```
<head>
...
   <script>
        function change() {
        var y = document.getElementsByTagName("p");
        y[0].style.color="red"; }
   </script>  </head>
   <body style="color:blue;">
     <form method="get" action="HelloDOM" >
       <p id="X1" style="color:blue;">Hello World</p>
       <p><input type="button" onclick="change()"
           value="Change appearance" /></p>
     </form>
</body>
```

Notice that p is accessed as an array in this example

The HTMLDocument object also supports a getElementsByName method

DOM-HelloNodeArray.html

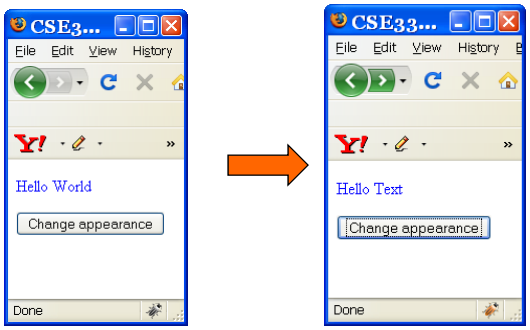
16

Example - Changing Element Contents

```
function change() {
  var y = document.getElementById("X3");
  y.innerHTML="Hello Text";
}
```

Same html as last example

innerHTML is an element property that corresponds to all the markup and content within the element



Setting an innerHTML property parses html text into the html tree

Do not use innerHTML when inserting plain text; instead, use node.textContent

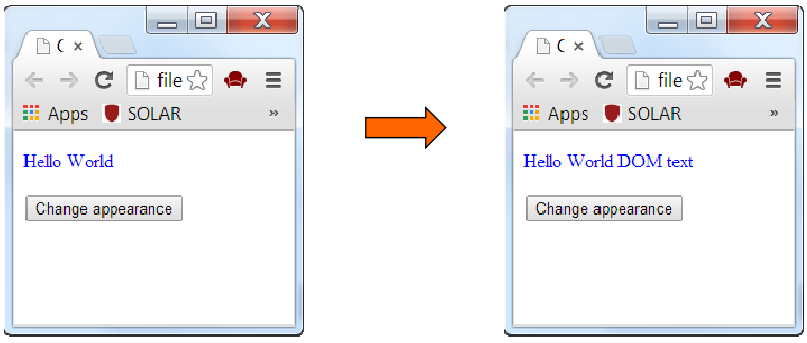
innerHTML is a useful relic of older DOM specs

DOM-HelloText.html

17

Example - Insert a Sub-Tree

- Instantiate a sub-tree
- Manipulate the sub-tree
- Insert into the HTML tree

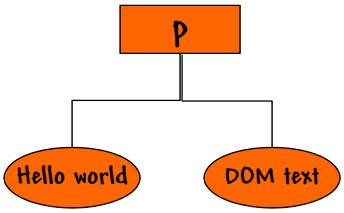


18

"Pure" DOM HTML Change

- DOM provides methods to delete, create, clone, and insert branches within the DOM tree

```
function change() {
      var y = document.getElementById("X6");
      var text = document.createTextNode(" DOM text");
      y.appendChild(text);
}
...
<p id="X6" class="blue">
      Hello World</p>
```

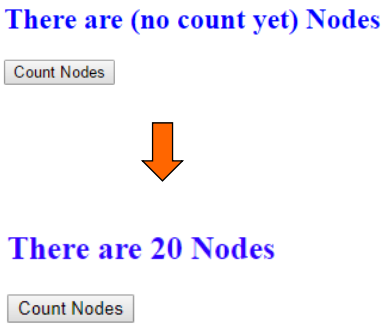


DOM-HelloDOMText.html

19

How Many Nodes are in the Example?

```
<html>
  <head>
    <title>Hello DOM Counter</title>
    <script>
    ...
    </script>
  </head>
  <body style="color:blue;">
    <form method="get" action="HelloDOM" >
      <h2 id="X1" style="color:blue;">
        There are <span id="counter">
(no count yet) </span> Nodes</h2>
      <input type="button" onclick="countNodes()"
        value="Count Nodes" />
    </form></body></html>
```



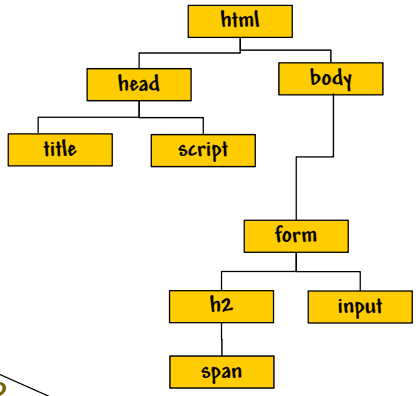
HelloDOMCounter.html

20

Let's Count the Nodes

```
var numNodes=0;
function countNodes() {
  var p = document;
  h=p.getElementsByTagName("html");
  nextLevel(h[0]);
  cc = document.getElementById("counter");
  cc.innerHTML=numNodes;
}
function nextLevel(n) {
  numNodes=numNodes+1;
  if (n.hasChildNodes()) {
    var children=n.childNodes;
    for(var i = 0; i<children.length ; i++) {
      nextLevel(children[i]);
    }
  }
  return;
}
```

A span element, enclosing the count



Why?

Implicit declaration not quite the same as explicit

There are 20 Nodes

HelloDOMCounter.html

21

NodeList

```
<script>
var numNodes=0;
function countNodes() {
  var p = document;
  h=p.getElementsByTagName("html");
  nextLevel(h[0]);
  cc = document.getElementById("counter");
  cc.innerHTML=numNodes;
}
function nextLevel(n) {
  numNodes=numNodes+1;
  if (n.hasChildNodes()) {
    var children=n.childNodes;
    for(var i = 0; i<children.length ; i++) {
      nextLevel(children[i]);
    } }
  return; } </script>
```

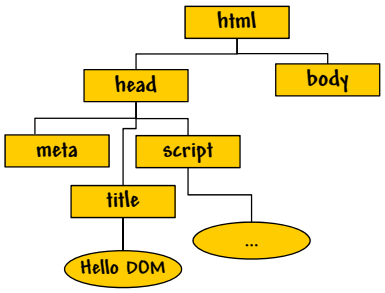
get methods usually return a NodeList object

You can access an item in a NodeList using the item method or using array notation

22

Text Nodes

- Text Nodes
  - Title node contains text ("Hello DOM")
- White Space Text Nodes
  - Head node has more children when you count white space nodes

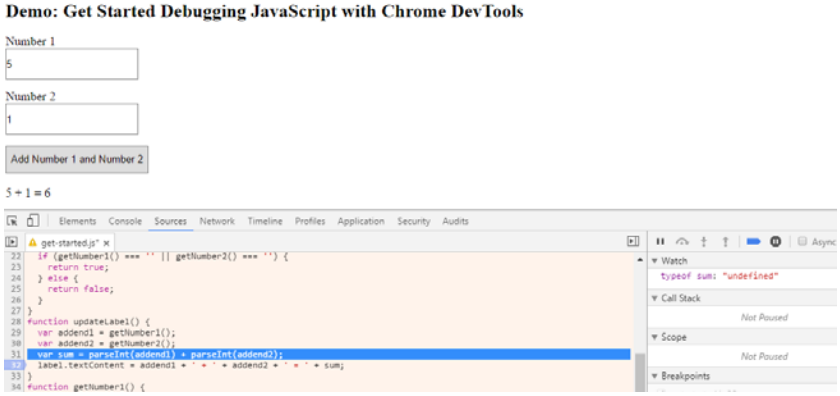


23

JavaScript Debugging

- If you use Chrome, take a look at the following tutorial that shows you how to use the Chrome debugger

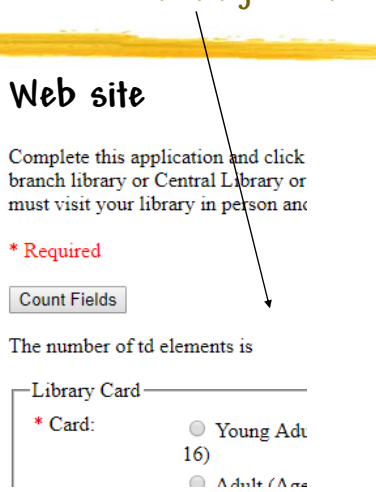
developers.google.com/web/tools/chrome-devtools/javascript/



24

Are We on Track?

- Download Track-Fall2018.html from the class Web site
- Add JavaScript/html to
  - Count total number of td elements
  - Display the results in the area shown



Were We on Track?

```
function countFields() {
  var p = document;
  var td = p.getElementsByTagName("td");
  var c = p.getElementById("fieldCount");
  var text = p.createTextNode(td.length);
    c.appendChild(text);
  }
```

```
<input type="button" onclick="countFields();"
value="Count Fields" />
```

Form Processing

- You can validate your form data in JavaScript with a function invoked by the onsubmit event
- If your form handler function returns false, the form data is not sent to the server

```
<form name="z" onsubmit="return isValid(...)" >
<input name="zipcode" ...     >
```


27

Prepare for the Next HW

- Once you have completed development of your version of the Brooklyn Library form, you can use the approach in the track to identify and count your form elements.

28

Cautions

- JavaScript is case sensitive
  - maxlength html attribute of input element is accessed as the maxLength property of JavaScript input element
- JavaScript keyword issues
  - class attribute is accessed as className
  - for attribute of label element is accessed as htmlFor property
- DOM is modularized so that not all DOM modules may be implemented on a browser

29

Have You Satisfied the Learning Goals?

- Understand the Document Object Model
- Understand how to perform client side form validation
- Understand JavaScript event model

30