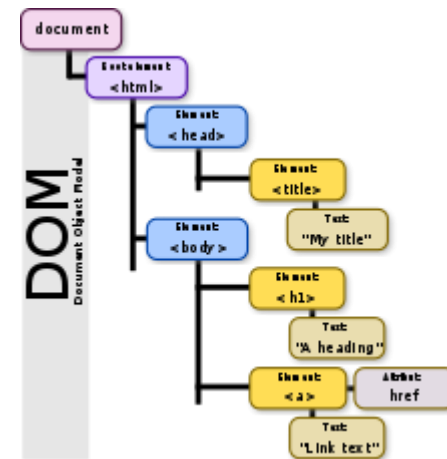


SWE 363: Web Engineering & Development

Module 4-2

Document Object Model



Objectives

- ❑ Understand the Document Object Model
- ❑ Learn the concept of DOM nodes and DOM trees
- ❑ Use JavaScript to manipulate DOM Objects
- ❑ Learn the concept of Browser Object Model (BOM)

- ❑ DOM Tree
- ❑ Manipulation of DOM using JavaScript
- ❑ HTML DOM
- ❑ Node Navigation
- ❑ Browser Object Model

References

- ❑ *Internet and World Wide Web How to Program: International Edition, 5/E*, Pearson Education Inc. 2012.
 - Chapter 12: Document Object Model (DOM): Objects and Collections



- ❑ W3C- W3 Schools JavaScript Tutorial
: http://www.w3schools.com/js/js_examples.asp

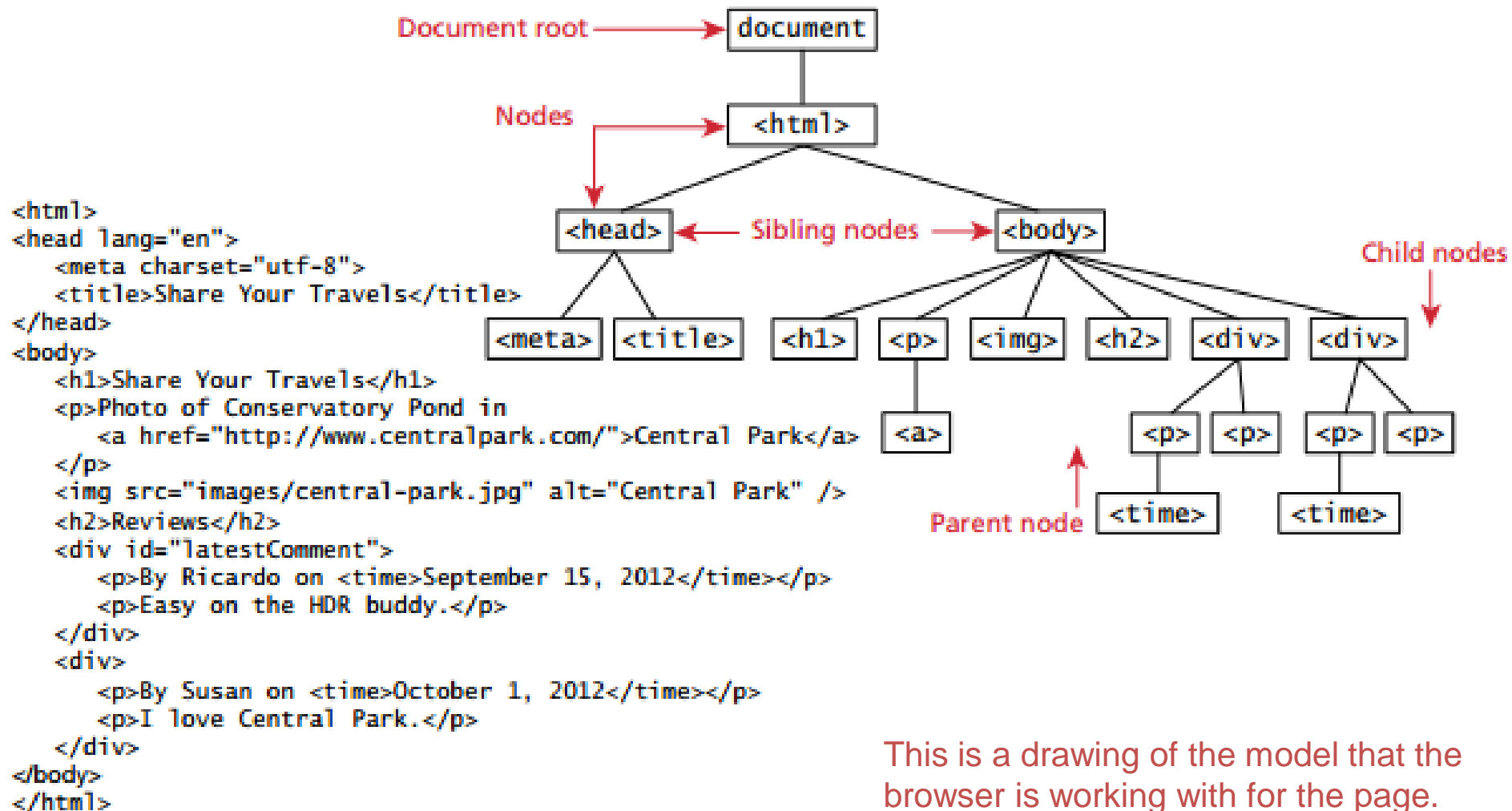
- ❑ Udemy
 - <https://www.udemy.com/>



Introduction

- ❑ DOM is an application programming interface (API) for valid HTML and well-formed XML documents.
- ❑ With DOM, programmers can build documents, navigate their structure, and add, modify, or delete elements and content.
 - Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the DOM (with a few exceptions)
- ❑ DOM provides the access to the elements and attributes within the HTML.
 - In the DOM, all HTML elements are defined as **objects**.
 - All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
- ❑ The programming interface is the **properties** and **methods** of each object
 - A **property** is a value that you can get or set (like changing the content of an HTML element).
 - A **method** is an action you can do (like add or deleting an HTML element).

DOM Tree example



Dynamic HTML pages !

- ❑ With the object model, JavaScript gets all the power it needs to create dynamic HTML:
 - JS can change all the HTML **elements** in the page
 - JS can change all the HTML **attributes** in the page
 - JS can change all the CSS **styles** in the page
 - JS can **remove** existing HTML elements and attributes
 - JS can **add** new HTML elements and attributes
 - JS can **react** to all existing HTML events in the page
 - JS can create new HTML **events** in the page

JavaScript HTML DOM Elements

- ❑ With JavaScript, you want to **manipulate HTML elements**. so, you have to find the elements first.

- **Get Element by Matching the Value of the 「id」 Attribute**

```
var myElement = document.getElementById(id_string );
```

The *myElement* is `objectHTMLCollection`

- **Get Elements by Tag Name**

```
var x = document.getElementsByTagName(tag_name )
```

The *tag_name* is `"div"`, `"span"`, `"p"`, etc.

- **Get Elements by Matching the Value of the 「class」 Attribute**

```
var x = document.getElementsByClassName(class_values);
```

The *class_values* can be multiple classes separated by space

- **Get Elements using CSS Selector Syntax**

```
var x = document.querySelector(css_selector);
```

Note: `document.querySelector("p");` //Get the first <p> element in the document:

Getting all elements of a certain type

```
var parags = document.getElementsByName("p");  
for (var i = 0; i < parags.length; i++) {  
    parags[i].style.backgroundColor = "yellow";  
}
```

JS

```
<body>  
    <p>This is the first paragraph</p>  
    <p>This is the second paragraph</p>  
    <p>You get the idea...</p>  
</body>
```

HTML

More examples:

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

Finding HTML Elements by Tag Name: Example

```
<head>
<script>
  function getAllParaElems() {
    var allPara = document.getElementsByTagName('p');
    var num = allPara.length;
    var text= prompt("Enter a new text to be assigned:", "");
    allPara[0].innerHTML = text;
  }
</script>
</head>
```

```
<body>
  <p> Computer Science </p>
  <p> Software Engineering</p>

  <button onclick="getAllParaElems();">
    Change the Content</button><br />

</body>
```

How to access paragraphs
located in a specific div?

```
var div = document.getElementsByTagName('div');
var allPara = div[0].getElementsByTagName('p');
```

Finding HTML Elements by Class Name: Example

```
<html>
<body>
  <div id="parent-id">
    <p> HTML</p>
    <p class="test"> CSS</p>
    <p >Javascript</p>
    <p>JQuery</p>
  </div>
```

```
<script>

  var testTarget= document.getElementsByClassName("test");
  testTarget[0].innerHTML= "NEW content";
</script>

</body>
</html>
```

returns a collection of nodes,
the first of which is referred to
by XXX[0].

Finding HTML Elements by CSS Selectors: Example

```
<head> <script>
function myFunction() {
    var element=document.querySelector(".example");
    element.style.backgroundColor = "red";

    //or using querySelectorAll
    var list=document.querySelectorAll(".example");
    list[0].style.backgroundColor = "red";
}
</script> </head>
```

```
<body>
    <h2 class="example">A heading with class="example"</h2>
    <p class="example">A paragraph with class="example".</p>
    <button onclick="myFunction()">Try it</button>
</body>
```

Finding HTML Elements by HTML Object Collections

: Example

```
<head><script>
function myFunction() {
    var x = document.forms["myForm"];
    var text = "";
    for (var i = 0; i < x.length ;i++) {
        text += x.elements[i].value + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
}
</script></head>
```

```
<body>
    <form id="myForm">
        First name: <input type="text" name="fname"><br>
        Last name: <input type="text" name="lname" ><br><br>
        <input type="submit" value="Submit">
    </form>
    <p>Click "Try it" to display the value of each element in the form.</p>
    <button onclick="myFunction()">Try it</button>
    <p id="demo"></p>
</body>
```

HTML DOM

Changing HTML

❑ The HTML DOM allows JavaScript to change the content of HTML elements.

- Changing the HTML Output Stream

- `document.write();`

- Changing HTML Content

- `document.getElementById(id).innerHTML = new HTML content`

- Changing the Value of an Attribute

- `document.getElementById(id).attribute = new value`

```
element.setAttribute(attribute, value)
```

- Changing the style of an HTML element

- `document.getElementById(id).style.attribute = new value`

```
element.style.property = new style
```

HTML DOM

Adding and Deleting Elements

- ❑ Creates an Element node
 - [document.createElement\(\)](#)

- ❑ Adds a new child node, to an element, as the last child node
 - [element.appendChild\(\)](#)

- ❑ Removes a child node from an element
 - [element.removeChild\(\)](#)

- ❑ Replaces a child node in an element
 - [element.replaceChild\(\)](#)

Adding Elements: Example

```
<html> <body>
  <div id="div1">A text will created dynamically.</div>
  <button onclick="addElement()"> Add Paragraph </button>

<script>
  function addElement() {
    // create a new div element and give it some content
    var newDiv = document.createElement("div");
    var newContent = document.createTextNode("This is a new text!");
    newDiv.appendChild(newContent);

    // add the newly created element and its content into the DOM
    var currentDiv = document.getElementById("div1");
    currentDiv.appendChild(newDiv);

    // to insert it before
    //document.body.insertBefore(newDiv, currentDiv);
  }
</script>
</body> </html>
```


Removing Existing HTML Elements

```
<body>

  <div id="div1">
    <p id="p1">This is paragraph #1.</p>
    <p id="p2">This is paragraph #2.</p>
  </div>

  <script>
    var parent = document.getElementById("div1");
    var child = document.getElementById("p1");
    parent.removeChild(child);
  </script>

</body>
```

```
<body>
  <ul id="myList">
    <li>HTML </li><li>CSS</li><li>JavaScript</li>
  </ul>
  <p>Click the button to remove the first item from the list.</p>
  <button onclick="myFunction()">Try it</button>

  <script>
    function myFunction() {
      var list = document.getElementById("myList");
      list.removeChild(list.childNodes[0]);
    }
  </script>
</body>
```

HTML DOM Events

- ❑ HTML DOM events **allow JavaScript to register different event handlers** on elements in an HTML document.
 - Mouse Events
 - Keyboard Events
 - Frame/Object Events
 - Form Events
 - ... etc.

- ❑ The event model was standardized by the W3C in DOM Level 2.
 - **DOM Levels** are the versions of the specification for defining how the Document Object Model should work, similarly to how we have HTML4, HTML5, and CSS2.1 specifications.
 - The most recent spec is DOM Level 3, published in April 2004.

HTML DOM Events

Mouse Events

[onclick](#)

[oncontextmenu](#)

[ondblclick](#)

[onmousedown](#)

[onmouseenter](#)

[onmouseleave](#)

[onmousemove](#)

[onmouseover](#)

[onmouseout](#)

[onmouseup](#)

Keyboard Events

[onkeydown](#)

[onkeypress](#)

[onkeyup](#)

Frame/Object Events

[onabort](#)

[onbeforeunload](#)

[onerror](#)

[onhashchange](#)

[onload](#)

[onpageshow](#)

[onpagehide](#)

[onresize](#)

[onscroll](#)

[onunload](#)

Form Events

[onblur](#)

[onchange](#)

[onfocus](#)

[onfocusin](#)

[onfocusout](#)

[oninput](#)

[oninvalid](#)

[onreset](#)

[onsearch](#)

[onselect](#)

[onsubmit](#)

HTML DOM Events...

- ❑ Many more Events related to
 - Drag Events ([drag](#), [dragend](#) , [dragenter](#) , [dragleave](#) , [drop](#) , etc)
 - Clipboard Events ([copy](#) , [paste](#), etc)
 - Print Events ([Afterprint](#) or [beforeprint](#))
 - Media Events ([play](#) , [pause](#) ,etc)
 - Animation Events ([Animationend](#) , [Animationiteration](#))
 - Connection Events ([offline](#) , [online](#) ,etc)
 - Touch Events ([fullscreenchange](#), [Blur](#), [focus](#) , [focusin](#) , [focusout](#))
 - More..
- ❑ See discretion and examples
 - https://www.w3schools.com/Jsref/dom_obj_event.asp

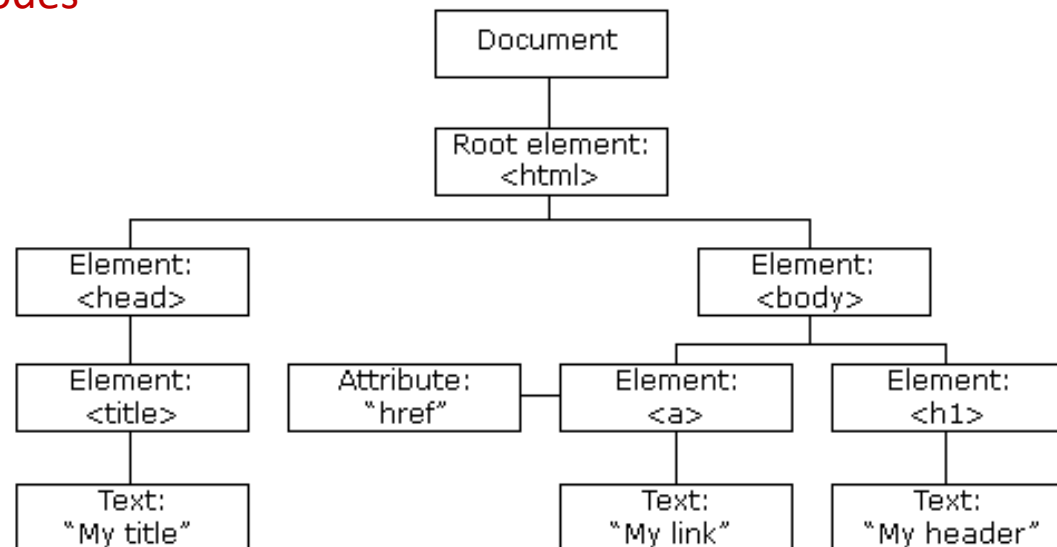
Node Navigation

https://www.w3schools.com/js/js_htmlDOM_navigation.asp

DOM Nodes

❑ According to the W3C HTML DOM standard, **everything** in an HTML document is a **node**:

- The entire document is a **document node**
 - Every HTML element is an **element node**
 - The text inside HTML elements are **text nodes**
 - Every HTML attribute is an **attribute node** (deprecated)
 - All comments are **comment nodes**
-
- With the HTML DOM, all nodes in the node tree **can be accessed by JavaScript**.
 - New nodes can be **created**, and all nodes can be **modified** or **deleted**.



Node Relationships

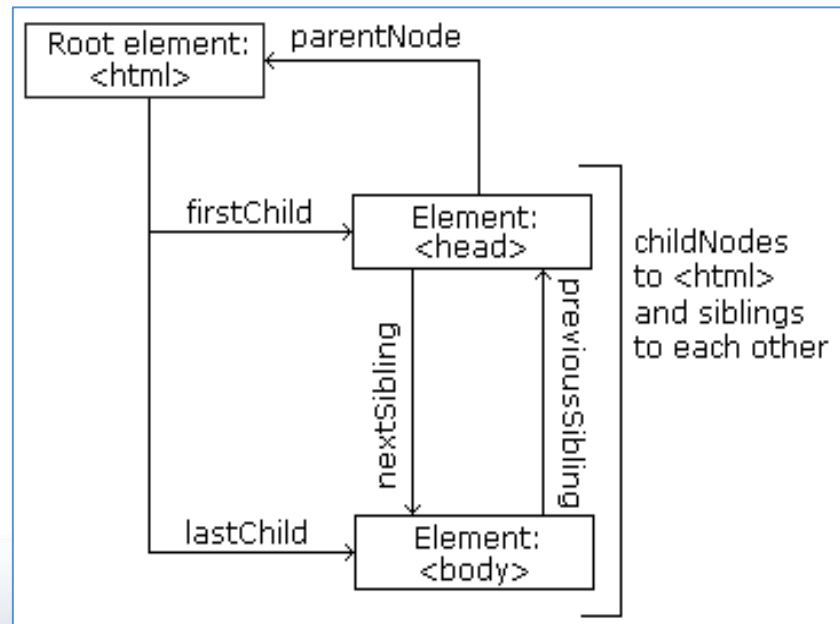
- ❑ With the HTML DOM, you can **navigate the node tree** using node **relationships**.
- ❑ The nodes in the node tree have a **hierarchical relationship** to each other.
- ❑ The terms **parent**, **child**, and **sibling** are used to describe the relationships.
 - In a node tree, the top node is called the **root** (or root node)
 - Every node has exactly **one parent**, except the root (which has no parent)
 - A node can have **a number of children**
 - **Siblings** (brothers or sisters) are nodes with the same parent

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>

  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



Navigating Between Nodes

- ❑ You can use the following **node properties** to navigate between nodes with JavaScript:

- **parentNode**
- **childNodes[nodenum]**
- **firstChild**
- **lastChild**
- **nextSibling**
- **previousSibling**

- ❑ **Examples:**

- Accessing the innerHTML property is the same as accessing the **nodeValue** of the first child:

```
var myTitle = document.getElementById("demo").firstChild.nodeValue;
```

- Accessing the first child can also be done like this:

```
var myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```


Examples

Example list:

- SWE
- ICS

Try it

Example list:

- SWE
- ICS

Try it

SWE

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

```
<p>Example list:</p>
```

```
<ul id="myList"><li>SWE</li><li>ICS</li></ul>
```

```
<button onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>
```

```
<script>
```

```
function myFunction() {
```

```
    var list = document.getElementById("myList").firstChild.innerHTML;
```

```
    document.getElementById("demo").innerHTML = list;
```

```
}
```

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

```
var list = document.getElementById("myList").childNodes[0].innerHTML;
```

The nodeName Property

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

```
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeName;
</script>
</body>
```

My First Page

H1

Note: nodeName always contains the uppercase tag name of an HTML element.

The nodeValue Property

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

nodeValue: Example

```
<html> <head><script>
```

```
var switchCount = 0;
var adjectives = ["simple","complex", "unique"];
function switcher() {
    if (switchCount == (adjectives.length - 1))
        switchCount = 0;
    else
        switchCount++;
    var italicNode = document.getElementById("adjPhrase");
    italicNode.firstChild.nodeValue = adjectives[switchCount];
}
```

```
</script> </head>
```

```
<body>
  <h1>An HTML Document</h1>
  <p>This is a <i id="adjPhrase">simple</i> document </p>
  <form>
    <button type="button" onclick="switcher()">switch</button>
  </form>
</body> </html>
```

The nodeType Property

- ❑ The nodeType property is read only. It returns the type of a node.

```
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeType;
</script>

</body>
```

My First Page

1

Node	Type
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
COMMENT_NODE	8
DOCUMENT_NODE	9

Creating new nodes

name	description
<code>document.createElement("tag")</code>	creates and returns a new empty DOM node representing an element of that type
<code>document.createTextNode("text")</code>	creates and returns a text node containing given text

```
// create a new <h2> node
var newHeading = document.createElement("h2");
newHeading.innerHTML = "This is a heading";
newHeading.style.color = "green";
```

- ❑ merely creating a node does not add it to the page
- ❑ you must add the new node as a child of an existing element on the page...

Create Node

```
<head><script>
    function handleCreate() {
        var parent=document.getElementById("div1");
        var child =document.createElement("p");
        var text=document.createTextNode("I am new in here");
        child.appendChild(text);
        parent.appendChild(child);
    }
</script></head>
```

```
<body>
    <div id="div1">
        <p id="p1"> Paragraph number 1 </p>
        <p id="p2"> Paragraph number 2 </p>
    </div>
    <input type="button" value="Create" onclick="handleCreate()">
</body>
```

Removing a node from the page

```
function slideClick() {  
    var bullets = document.getElementsByTagName("li");  
    for (var i = 0; i < bullets.length; i++) {  
        if (bullets[i].innerHTML.indexOf("children") >= 0) {  
            bullets[i].remove();  
        }  
    }  
}
```

- ❑ each DOM object has a `removeChild` method to remove its children from the page
- ❑ Prototype adds a `remove` method for a node to remove itself

Remove Node

```
<head><script>
    function handleRemove() {
        var parent=document.getElementById("div1");
        var child =document.getElementById("p1");
        parent.removeChild(child);
    }
</script></head>

<body>
    <div id="div1">
        <p id="p1"> Paragraph number 1 </p>
        <p id="p2"> Paragraph number 2 </p>
    </div>
    <input type="button" value="remove" onclick="handleRemove()">
</body>
```

The Browser Object Model

JavaScript Window

The Browser Object Model (BOM)

- ❑ The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- ❑ There are no official standards for the **Browser Object Model (BOM)**.
 - Window
 - Screen
 - Location
 - History
 - Navigator

The Window Object

- ❑ The *window object* is **supported by all browsers**. It represents the browser's window.
- ❑ All global JavaScript objects, functions, and variables automatically become **members of the window object**.
 - Global variables are **properties** of the window object.
 - Global functions are **methods** of the window object.
 - Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

JavaScript Window Screen

- ❑ The *window.screen* object contains information about the user's screen.
 - The **window.screen** object can be written without the window prefix.
- ❑ Some properties:
 - **screen.width**: returns the width of the visitor's screen in pixels
 - **screen.height**: returns the height of the visitor's screen in pixels.
 - **screen.availWidth**: returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.
 - **screen.availHeight**
 - **screen.colorDepth**
 - **screen.pixelDepth**

➤ The browser window is NOT including toolbars and scrollbars.

Screen: Example

```
<head>
<script>
  function myFunction() {
    var w = window.screen.width;
    var h = window.screen.height;
    var txt = "Screen size: width=" + w + ", height=" + h;
    document.getElementById("demo").innerHTML = txt;
  }
</script>
```

```
</head>
<body onresize="myFunction()">
  <p>Try to resize the browser window to display the windows height and width.</p>
  <h2 id="demo"></h2>
</body>
```

JavaScript Window Location

- ❑ The `window.location` object can be used to get the current page address (URL) and to redirect the browser to a new page.
 - The `window.location` object can be written without the window prefix.
- ❑ Some examples:
 - `window.location.href` returns the href (URL) of the current page
 - `window.location.hostname` returns the domain name of the web host
 - `window.location.pathname` returns the path and filename of the current page
 - `window.location.protocol` returns the web protocol used (http: or https:)
 - `window.location.assign` loads a new document

```
<script>
function newDoc() {
    window.location.assign("https://www.w3schools.com")
}
</script>
```

JavaScript Window History

- ❑ The `window.history` object contains the browsers history.
 - The `window.history` object can be written without the window prefix.
- ❑ To **protect the privacy of the users**, there are limitations to how JavaScript can access this object.
- ❑ Some methods:
 - `history.back()` - same as clicking back in the browser
 - `history.forward()` - same as clicking forward in the browser

```
<head> <script>
    function goBack() {
        window.history.back(); }
</script> </head>
<body>
    <p> Click on the buuton to navigate back</p>
    <input type="button" value="Back" onclick="goBack()">
</body>
```


JavaScript Window Navigator

- ❑ The `window.navigator` object contains information about the visitor's browser.

```
<div id="example"></div>
<script>
txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";
txt+= "<p>Browser Name: " + navigator.appName + "</p>";
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
txt+= "<p>Browser Language: " + navigator.language + "</p>";
txt+= "<p>Browser Online: " + navigator.onLine + "</p>";
txt+= "<p>Platform: " + navigator.platform + "</p>";
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
txt+= "<p>User-agent language: " + navigator.systemLanguage + "</p>";
document.getElementById("example").innerHTML=txt;
</script>
```

Browser CodeName: Mozilla

Browser Name: Netscape

Browser Version: 5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.117 Safari/537.36

Cookies Enabled: true

Browser Language: en-US

Browser Online: true

Platform: Win32

User-agent header: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.117 Safari/537.36

User-agent language: undefined

- ❑ JavaScript is a big, complex language
 - We've only scratched the surface
 - It's easy to get started in JavaScript, but if you need to use it heavily, plan to invest time in learning it well
 - Write and test your programs a little bit at a time
- ❑ JavaScript is not totally platform independent
 - Expect different browsers to behave differently
 - Write and test your programs a little bit at a time
- ❑ Browsers aren't designed to report errors
 - Don't expect to get any helpful error messages
 - Write and test your programs a little bit at a time

