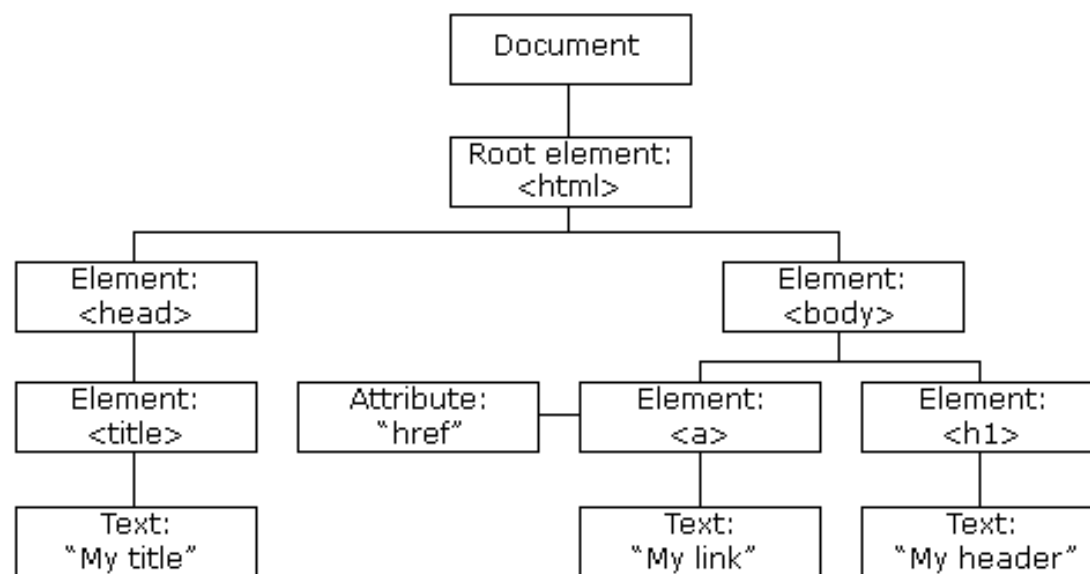


# The HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a DOM of the page.



- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents:
- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

# JavaScript and DOM



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

# The DOM Programming Interface



- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as 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).

# Example



```
<html>
```

```
<body>
```

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

```
<script>
```

```
document.getElementById( "demo" ).innerHTML = "Some Text" ;
```

```
</script>
```

```
</body>
```

```
</html>
```

# Finding HTML Elements

Method	Description
<code>document.getElementById()</code>	Find an element by element id
<code>document.getElementsByTagName()</code>	Find elements by tag name
<code>document.getElementsByClassName()</code>	Find elements by class name

# Changing HTML Elements

Method	Description
<code>element.innerHTML=</code>	Change the inner HTML of an element
<code>element.attribute=</code>	Change the attribute of an HTML element
<code>element.setAttribute(attribute,value)</code>	Change the attribute of an HTML element
<code>element.style.property=</code>	Change the style of an HTML element

# Adding and Deleting Elements

Method	Description
<code>document.createElement()</code>	Create an HTML element
<code>document.removeChild()</code>	Remove an HTML element
<code>document.appendChild()</code>	Add an HTML element
<code>document.replaceChild()</code>	Replace an HTML element
<code>document.write(text)</code>	Write into the HTML output stream

# Adding Events Handlers

Method	Description
<pre>document.getElementById(id).onclick=function() {   code }</pre>	Adding event handler code to an onclick event



Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

# Finding HTML Elements



- With JavaScript, we could manipulate HTML elements.
- To do so, you have to find the elements first.
- There are a couple of ways to do this:
  - ▣ Finding HTML elements by id
  - ▣ Finding HTML elements by tag name
  - ▣ Finding HTML elements by class name
  - ▣ Finding HTML elements by CSS selectors
  - ▣ Finding HTML elements by HTML object collections

# Finding HTML Elements by Id

- The easiest way to find HTML elements in the DOM, is by using the element id.
- This example finds the element with id="intro":

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

- If the element is found, the method will return the element as an object (in x).
- If the element is not found, x will contain null.

# Finding HTML Elements by Tag Name



- This example finds all <p> elements:

```
var x = document.getElementsByTagName( "p" );
```

- This example finds the element with id="main", and then finds all <p> elements inside "main":

```
var x = document.getElementById( "main" );
```

```
var y = x.getElementsByTagName( "p" );
```

# Finding HTML Elements by Class Name



- If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.
- This example returns a list of all elements with `class="intro"`.

```
var x = document.getElementsByClassName("intro");
```

# Finding HTML Elements by CSS Selectors

- If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.
- This example returns a list of all `<p>` elements with `class="intro"`.

```
var x = document.querySelectorAll("p.intro");
```

# Finding HTML Elements by HTML Object Collections

- This example finds the form element with id="frm1", in the forms collection, and displays all element values:

```
var x = document.getElementById("frm1");  
var text = "";  
var i;
```

```
for (i = 0; i < x.length; i++) {  
    text += x.elements[i].value + "<br>";  
}
```

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

# Changing the HTML Output Stream

- In JavaScript, `document.write()` can be used to write directly to the HTML output stream:

```
<!DOCTYPE html>
<html>
<body>
<script>
document.write(Date());
</script>
</body>
</html>
```

- *Never use `document.write()` after the document is loaded.*
- *It will overwrite the document*



# Changing HTML Content

- The easiest way to modify the content of an HTML element is by using the innerHTML property.

- To change the content of an HTML element, use this syntax:

**`document.getElementById(id).innerHTML = new HTML`**

- This example changes the content of a <p> element:

```
<html>
<body>
<p id="p1">Hello World!</p>
<script>
document.getElementById( "p1" ).innerHTML = "new msg";
</script>
</body>
</html>
```

# Changing the Value of an Attribute

- To change the value of an HTML attribute, use this syntax:

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

- This example changes the value of the src attribute of an <img> element:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```

```

```
<script>
```

```
document.getElementById("myImage").src = "one.jpg";
```

```
</script>
```

```
</body>
```

```
</html>
```

# Changing HTML Style

- To change the style of an HTML element, use this syntax:

```
document.getElementById(id).style.property=new style
```

- The following example changes the style of a <p> element:

```
<html>
<body>
<p id="p2">Hello World!</p>
<script>
document.getElementById("p2").style.color = "blue";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

# Using Events

- The HTML DOM allows you to execute code when an event occurs.
- Events are generated by the browser when "things happen" to HTML elements:
  - An element is clicked on
  - The page has loaded
  - Input fields are changed
- This example changes the style of the HTML element with id="id1", when the user clicks a button:

```
<html>
<body>
<h1 id="id1">My Heading 1</h1>
<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>
</body>
</html>
```

# Reacting to Events

- JS can be executed when an event occurs, like a user clicking on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:  
**`onclick=JavaScript`**
- Examples of HTML events:
  - When a user clicks the mouse
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key
- In this example, the content of the `<h1>` element is changed when a user clicks on it:  
**`<html>`**  
**`<body>`**  
**`<h1 onclick="this.innerHTML='Oops!'">Click on this text!</h1>`**  
**`</body>`**  
**`</html>`**

# Assign Events Using the HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:
- Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = function()
{
    displayDate()
};
</script>
```

# The `addEventListener()` method

- Add an event listener that fires when a user clicks a button:  
`document.getElementById("myBtn").addEventListener("click", displayDate);`
- The `addEventListener()` method attaches an event handler to the specified element.
- The `addEventListener()` method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The `addEventListener()` method makes it easier to control how the event reacts to bubbling.
- When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

# Syntax



`element.addEventListener(event, function, useCapture);`

- The first parameter is the type of the event (like "click" or "mousedown").
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.



# Add an Event Handler to an Element

- Example
- Alert "Hello World!" when the user clicks on an element:  
`element.addEventListener("click", function(){ alert("Hello World!"); });`
- You can also refer to an external "named" function:
- Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);
```

```
function myFunction()  
{  
    alert ("Hello World!");  
}
```

# Add Many Event Handlers to the Same Element

- The `addEventListener()` method allows you to add many events to the same element, without overwriting existing events:

- Example

```
element.addEventListener("click", myFunction);  
element.addEventListener("click", mySecondFunction);
```

- You can add events of different types to the same element:

- Example






```
element.addEventListener("mouseover", myFunction);  
element.addEventListener("click", mySecondFunction);  
element.addEventListener("mouseout",  
myThirdFunction);
```

# The removeEventListener() method

- The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:
- Example  
`element.removeEventListener("mousemove", myFunction);`

# Browser Support

The numbers in the table specifies the first browser version that fully supports these methods.

Method					
addEventListener()	1.0	9.0	1.0	1.0	7.0
removeEventListener()	1.0	9.0	1.0	1.0	7.0

**Note:** The `addEventListener()` and `removeEventListener()` methods are not supported in IE 8 and earlier versions and Opera 7.0 and earlier versions. However, for these specific browser versions, you can use the `attachEvent()` method to attach an event handlers to the element, and the `detachEvent()` method to remove it:

```
element.attachEvent(event, function);  
element.detachEvent(event, function);
```

## Example

Cross-browser solution:

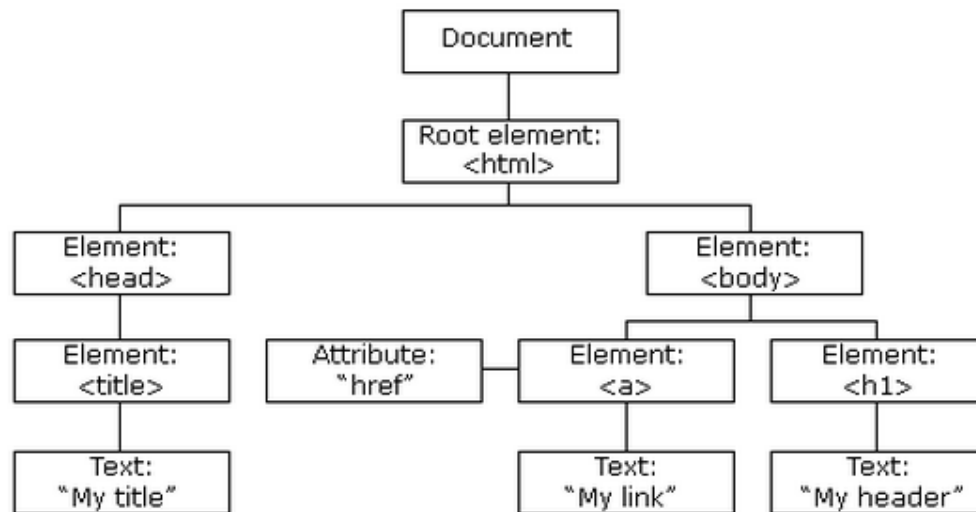
```
var x = document.getElementById("myBtn");  
if (x.addEventListener) { // For all major browsers, except IE 8 and  
    earlier  
    x.addEventListener("click", myFunction);  
} else if (x.attachEvent) { // For IE 8 and earlier versions  
    x.attachEvent("onclick", myFunction);  
}
```

# DOM Navigation

## 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
- 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

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.

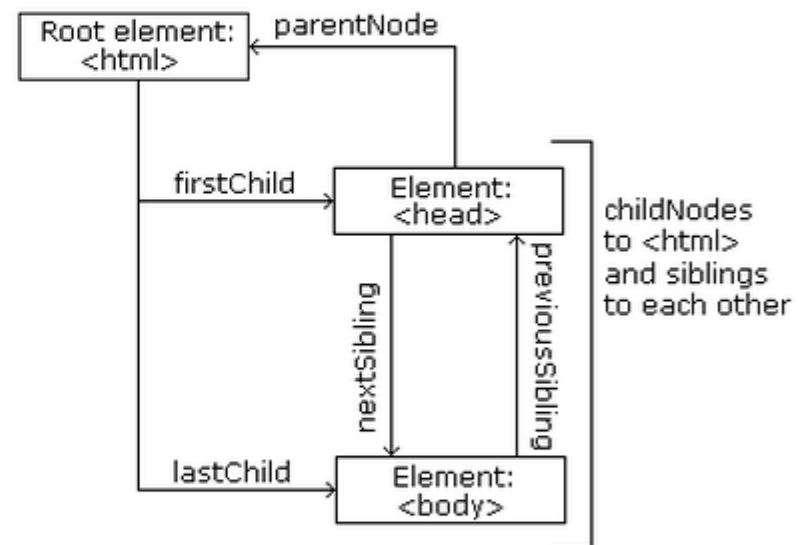
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<html>

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

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

</html>
```



From the HTML above you can read:

- <html> is the root node
- <html> has no parents
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>

# Navigating Between Nodes



- You can use the following node properties to navigate between nodes with JS:
  - ▣ parentNode
  - ▣ childNodes[nodenumbr]
  - ▣ firstChild
  - ▣ lastChild
  - ▣ nextSibling
  - ▣ previousSibling

# Common Error in DOM Processing



- A common error in DOM processing is to expect an element node to contain text.
- In this example: `<title>DOM Tutorial</title>`, the element node `<title>` does not contain text. It contains a text node with the value "DOM Tutorial".
- The value of the text node can be accessed by the node's `innerHTML` property, or the `nodeValue`.



# Creating New HTML Elements (Nodes)

- To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

```
<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");  
element.appendChild(para);  
</script>
```

# Creating new HTML Elements - insertBefore()

- The appendChild() method in the previous example, appended the new element as the last child of the parent.
- If you don't want that you can use the insertBefore() method:

```
<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>
```

# Removing Existing HTML Elements

- To remove an HTML element, you must know the parent of the element

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

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