IT614

Document Object Model (DOM)

Reference: https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference/Introduction

1. Document Object Model (DOM)

The HTML DOM defines a standard way for accessing and manipulating HTML documents. DOM is developed by W3C (http://www.w3.org/DOM/) to provide programming languages access to different parts of an HTML document. You can use any of the several languages that support DOM. HTML DOM provides APIs (Application Programming Interface) for programmers to develop HTML applications. We will use JavaScript to process HTML documents using DOM APIs.

An informal way of understanding DOM is, it is a tree representation of the HTML document, with nodes and child nodes and so on.

Current browsers have built-in HTML parsers, which convert an HTML document into HTML DOM.

Most of my examples work in Firefox, Google Chrome and Internet Explorer Web browsers.

There are plenty of tutorials available on the Web for JavaScript and DOM. One of the best references for DOM is https://developer.mozilla.org/en-US/docs/Gecko_DOM_Reference

2. Basics of DOM

W3C DOM treats an HTML document as a hierarchical tree structure with basic units called nodes.

DOM treats everything in an HTML document as a node:

- The entire document is a document node
- Every HTML element is an element node
- The text in the HTML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

The parser actually converts the HTML document into a DOM tree and saves it memory for JavaScript (or other programming languages) to access it. The DOM describes one or more Interfaces each of which has a number of methods and properties.

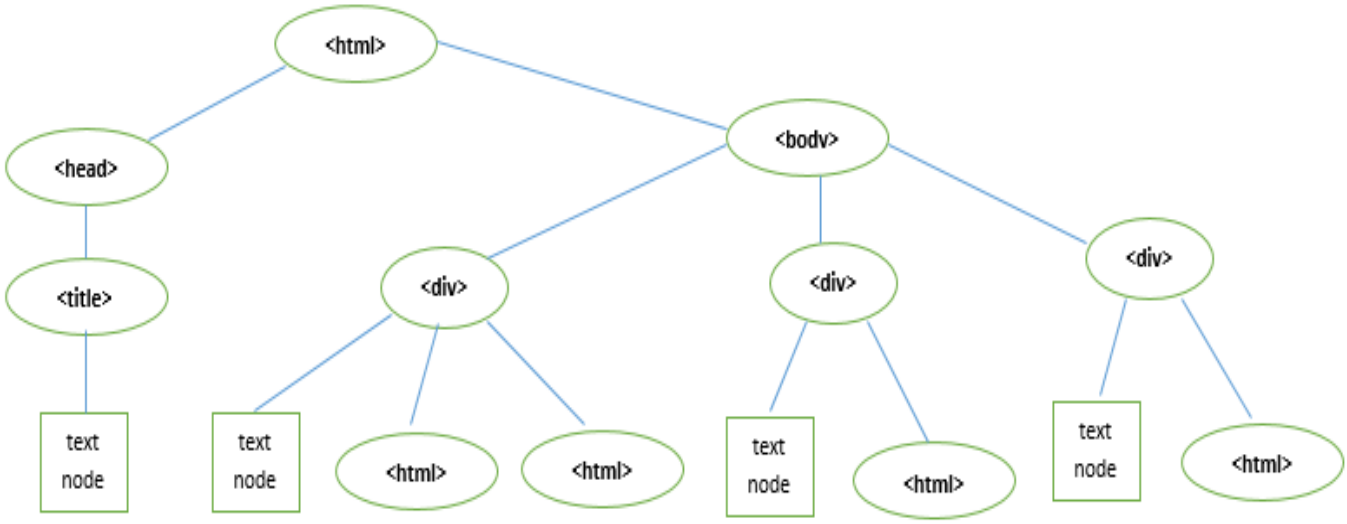
Consider the following simple HTML document, **ThreeCities.html**:

```
<!DOCTYPE html>
<html>
	<head>
		<title>A DOM example</title>
	</head>
	<body>
		<div id="nyc"> New York City
			<p> Nice City
			<p> Busy city
		</div>
		<div id="bos"> Boston
			<p> Beautiful City
		</div>
		<div id="was"> Washington
			<p> Capital city
		</div>
	</body>
</html>
```

In the DOM representation of this HTML document,

The root element: <html>
Child nodes of <html> are <head> and <body>
Child node of <head> is <title>
Child node of <title> is a text-node
<body> has three child nodes: all three nodes
The first node has three child nodes: a text node, and two <p> nodes
The second node has two child nodes: a text node and one <p> node
The third node has two child nodes: a text node and one <p> node

A graphical (tree) representation of this HTML DOM:



3. Whitespaces are text nodes

Firefox, Chrome and other non-IE browsers consider white spaces between HTML tags as text nodes. IE does not do this. This is peculiar.

Note: These whitespaces are called "phantom" nodes by techies. See a long discussion on these phantom nodes here: https://bugzilla.mozilla.org/show_bug.cgi?id=26179. Also see this: https://developer.mozilla.org/en-US/docs/Whitespace_in_the_DOM.

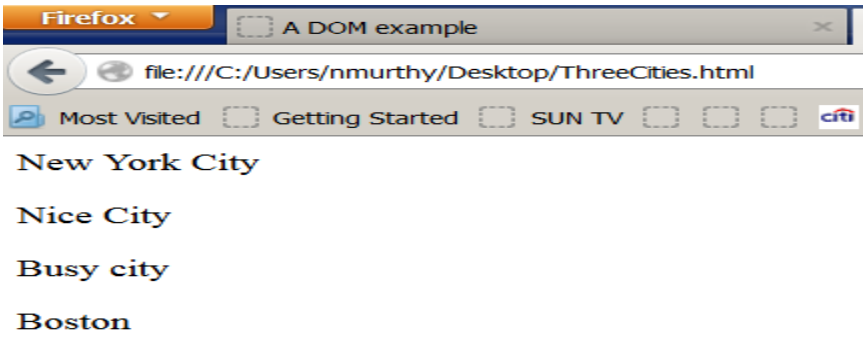
---

4. DOM inspector in Firefox

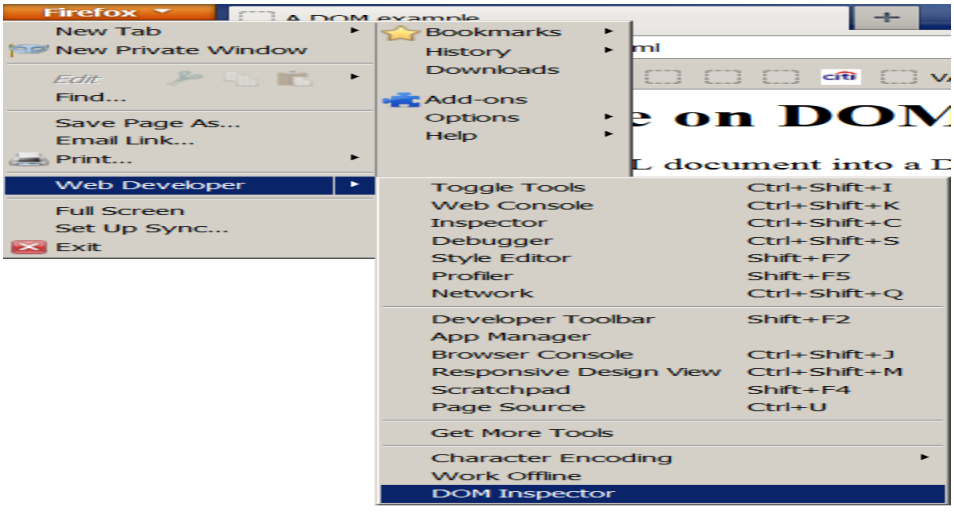
The **DOM Inspector** (also known as **DOMi**) is a Firefox add-on, which developers can use to inspect, browse, and edit the DOM of documents. The DOM hierarchy can be navigated using a two-paned window that allows for a variety of different views on the document and all nodes within. You can download and install Dom Inspector for Firefox from https://addons.mozilla.org/en-US/firefox/addon/dom-inspector-6622/.

Using DOM Inspector

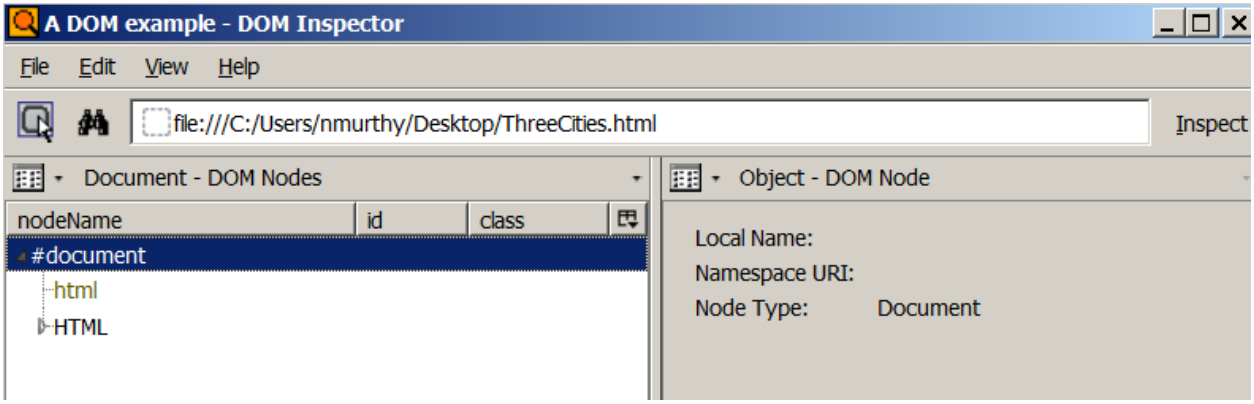
In Firefox, open an HTML document. In this example, I have opened **ThreeCities.HTML**:



To open DOM Inspector, click Tools > Web Developer > DOM Inspector:



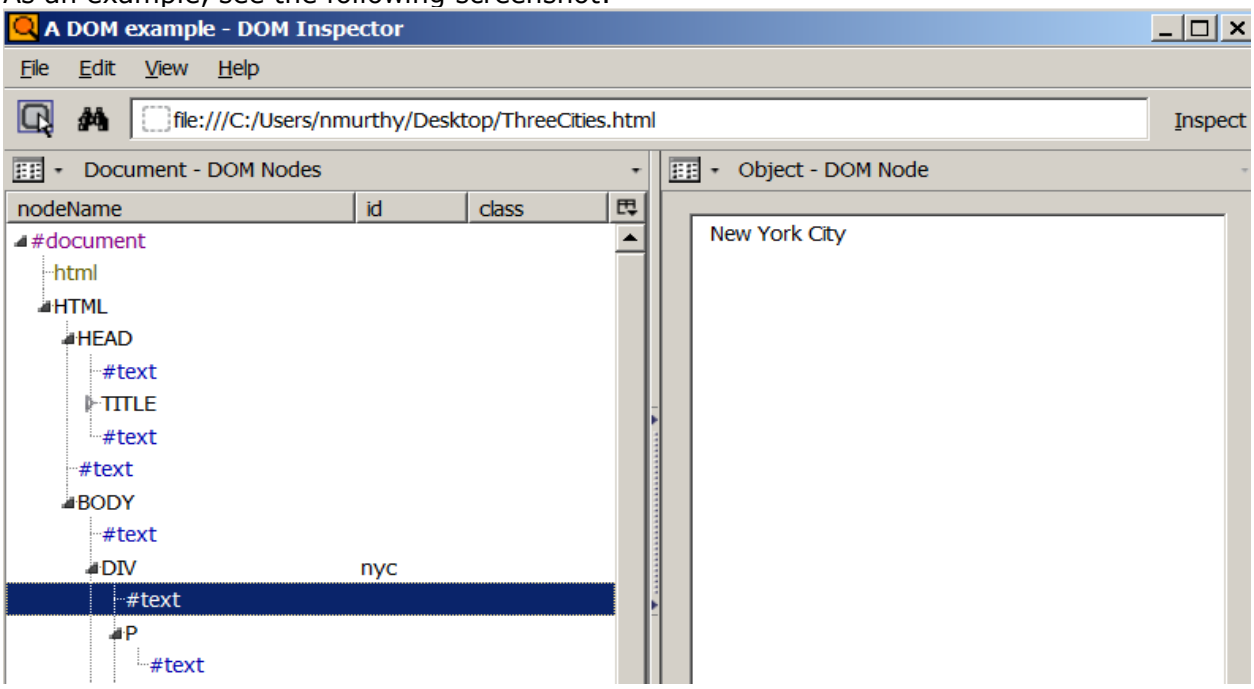
You will see the DOM Inspector screen:



This screen contains two panels: left panel with document DOM nodes and the right panel object DOM nodes.

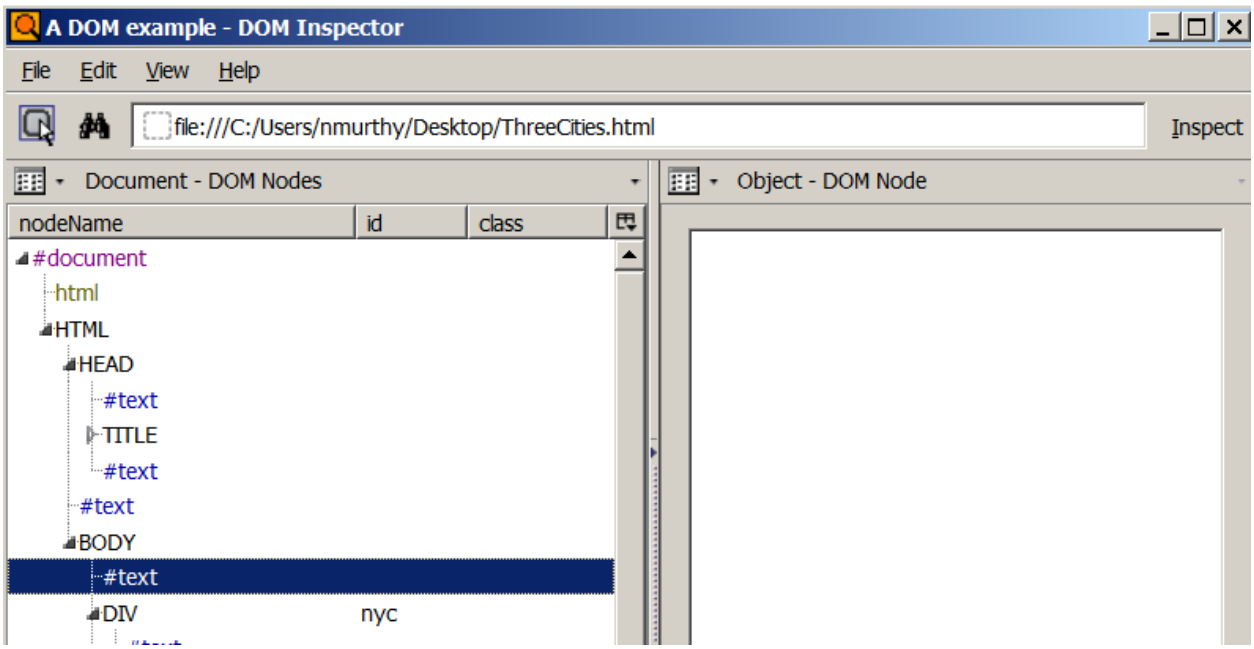
On the left panel, you will see the structure of the DOM tree. When you select a node on the tree the right panel gives details about the node. Experiment with the two panels by clicking various nodes.

As an example, see the following screenshot:



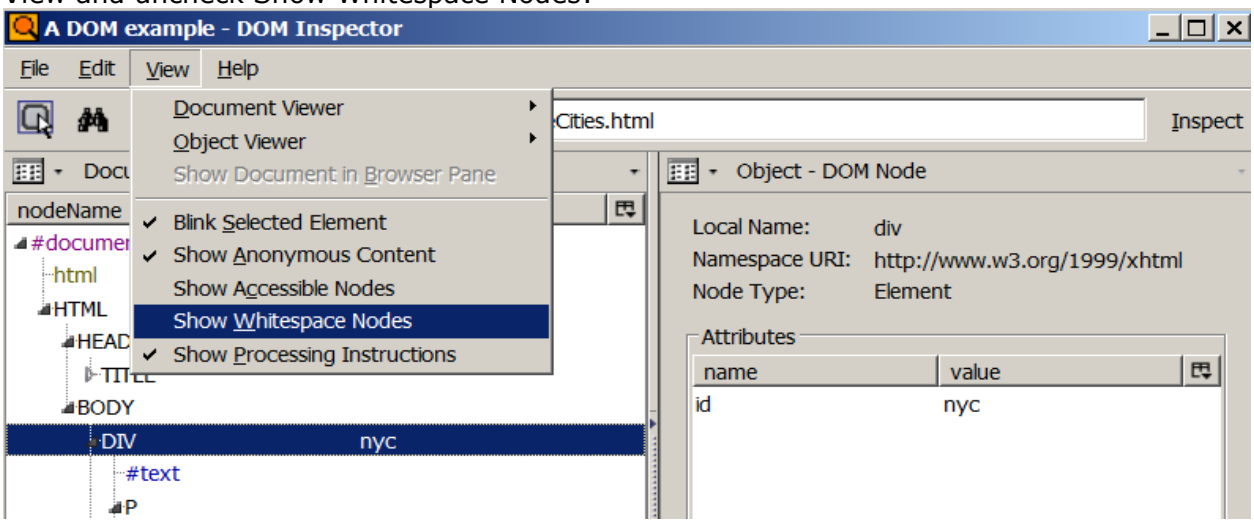
In this example, I have clicked #text node under DIV node. Notice the text value of the node on the right panel: New York City.

When you click the #text node below this #BODY node:



This #text node has no value (you don't see any value on the right panel). This is the "phantom" node we discussed earlier.

The DOM Inspector helps you to hide all the "phantom" white space nodes. To do this, click View and uncheck Show Whitespace Nodes:



This will hide whitespace nodes.

See
https://developer.mozilla.org/en-US/docs/DOM_Inspector/Introduction_to_DOM_Inspector
for more information on DOM Inspector.

Note:
Chrome Browser has built-in DOM inspector. To open press CTRL+Shift+I (same as JavaScript debugger) and click Elements. I like Firefox DOM Inspector better.

---

5. Node objects, their properties and methods

Each node is accessed as an object. As you know, each object in JavaScript has properties and methods. A property is a variable with a value and a method is a function. We will see shortly how to access node objects in JavaScript and retrieve their properties and use their methods.

As you know,

The syntax for accessing an attribute value of an object is:

objectName.propertyName

The syntax for invoking a method of an object is:

Objectname.methodname(arguments)

---

6. Two important objects: document and *element*

- **document object**

Each web page loaded in the browser has its own **document** object. It refers to the entire HTML document.

WE have already used a couple of properties and methods of the document object: **document.bgColor, document.fgColor,** and **document.write().**

We will soon see other important properties and methods of **document** object.

- ***element* object**

Informally, every element in an HTML document is an Element object.

Here are several important properties and methods of an element object:
(The use of these will be clear after we see some examples – below).

Examples of element properties:

Property	**Meaning**
element.**id**	Gets or sets the element's identifier (attribute id).
element.**className**	Gets and sets the value of the class attribute of the specified element.
element.**length**	Returns the number of items in a NodeList.
element.**tagName**	Returns the name of the element.

An element object has several important methods. We will see some of them in detail.

---

7. CSS properties in DOM

You can dynamically change CSS properties of elements using DOM (and JavaScript). There are some differences in the names of the CSS properties.

This is how you convert a CSS property name into DOM name: If the CSS property name has one word, DOM's equivalent is the same. If the CSS property name has multiple words separated by hyphen ("-"), to form the DOM equivalent, remove the hyphen and capitalize the next word.

Examples:

CSS property	DOM equivalent
background	background
background-color	backgroundColor
background-image	backgroundImage
border	border
border-color	borderColor
border-style	borderStyle

Using style with element

You can use CSS properties with DOM name equivalents with element objects.

Syntax:

element.**style**.*propertyName= "value";*

Where *propertyName* is a DOM equivalent name for CSS property (see explanation above).

---

8. document.getElementById("idValue") method

document.getElementById("*idValue*") returns the element whose id is the specified argument.

Note:

- getElementById() is case sensitive. getElementByID() does not work.
- If there is no element with the given id, this function returns `null`.

Example:
Using **ThreeCities.html** document,

```
<script>
var idElement = document.getElementById("bos");
document.write(idElement.id);
</script>
```

The idElement variable is referring to the element,

```
<div id="bos"> Boston
        <p> Beautiful City
</div>
```

The value of idElement.id is bos.

Examples:

Here are some examples applying CSS properties to element objects.

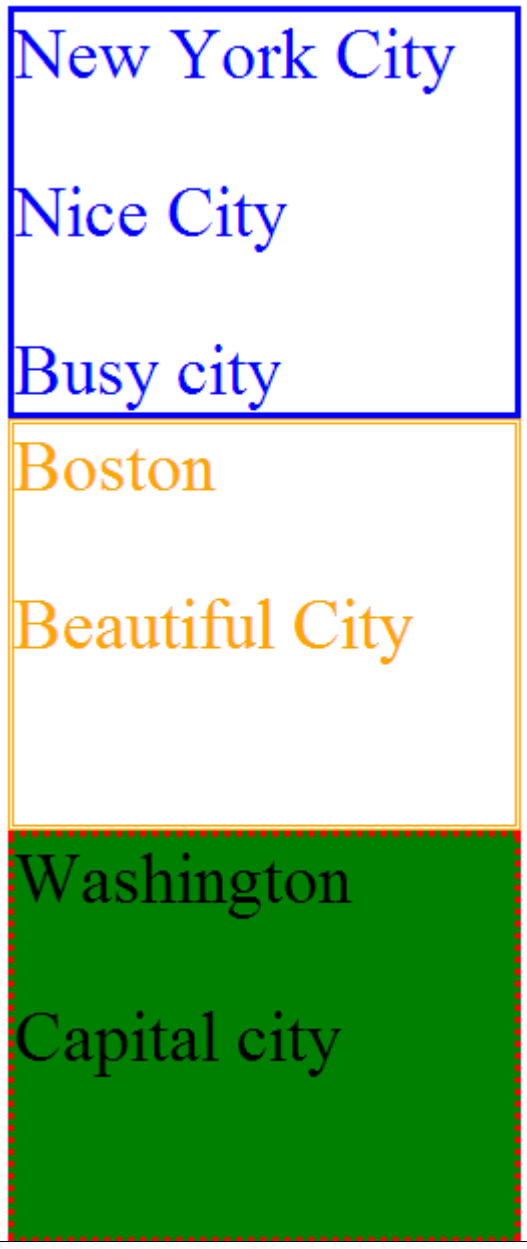
```
var idElement = document.getElementById("bos");
```

JavaScript statement	Result
idElement.style.color="blue";	Boston Beautiful City
idElement.style.borderStyle="dotted";	Boston Beautiful City
idElement.style.fontSize="40px";	Boston Beautiful City
idElement.style.backgroundColor="red";	Boston Beautiful City

A complete example:

```
<!DOCTYPE html>
<html><head><title>A DOM example</title>
<script>
function colorMe(){
var nycElement = document.getElementById("nyc");
var bosElement = document.getElementById("bos");
var wasElement = document.getElementById("was");

nycElement.style.height = "200px";
nycElement.style.width = "250px";
nycElement.style.borderStyle = "solid";
nycElement.style.fontSize = "36px";
nycElement.style.color="blue";

bosElement.style.height = "200px";
bosElement.style.width = "250px";
bosElement.style.borderStyle = "double";
bosElement.style.fontSize = "36px";
bosElement.style.color="orange";

wasElement.style.height = "200px";
wasElement.style.width = "250px";
wasElement.style.borderStyle = "dotted";
wasElement.style.fontSize = "36px";
wasElement.style.backgroundColor="green";
wasElement.style.borderColor = "red";
}
</script></head>

<body>
<div id="nyc"> New York City
   <p> Nice City
   <p> Busy city
</div>
<div id="bos"> Boston
   <p> Beautiful City
</div>
<div id="was"> Washington
   <p> Capital city
</div>
<a href="javascript:colorMe()">color Me</a>
</body>
</html>
```

Result:

Before you click the link	After you click the link
New York City Nice City Busy city Boston Beautiful City Washington Capital city color Me	

9. getElementsByTagName() method

The **getElementsByTagName**("*tagName*") method returns an object of the type **NodeList**, which consists of a list of all the nodes which have the same tag name specified in the argument. For all practical purposes, a NodeList is an array. To access each node, we use the subscript notation in the same way as in an array.

There are two ways of using **getElementsByTagName**("*tagName*"):

document.getElementsByTagName("*tagName*");

element.**getElementsByTagName**("*tagName*");

The first one returns all the nodes in the entire document with the specified tag name. The second one returns all the nodes under the element, *element*.

Example:
Consider the HTML document,

```
<html><head><title>A DOM example</title></head>
<body>
<div id="nyc"> New York City
        <p> Nice City
        <p> Busy city
</div>
<div id="bos"> Boston
        <p> Beautiful City
</div>
<div id="was"> Washington
        <p> Capital city
</div>
<script>
var allPNodes = document.getElementsByTagName("p");
document.write("<p>Number of p nodes in the entire document : ",allPNodes.length);
var nycNode = document.getElementById("nyc");
var pnycNodes = nycNode.getElementsByTagName("p");
document.write('<p>Number of p nodes under div id="nyc" : ',pnycNodes.length);
</script>
</body></html>
```

Notice all the <p> tags. There are 4 <p> tags in the document. There are two <p> tags within … .

```
var allPNodes = document.getElementsByTagName("p");
document.write("<p>Number of p nodes in the entire document : ",allPNodes.length);
```

Displays 4.

```
var nycNode = document.getElementById("nyc");
var pnycNodes = nycNode.getElementsByTagName("p");
document.write('<p>Number of p nodes under div id="nyc" : ',pnycNodes.length);
```

Displays 2.

10. getElementsByClassName() method

The **getElementsByClassName**("*tagName*") method returns an object of the type **NodeList**, which consists of a list of all the nodes which have the same tag name specified in the argument. For all practical purposes, a NodeList is an array. To access each node, we use the subscript notation in the same way as in an array.

There are two ways of using **getElementsByClassName**("*className*"):

document.getElementsByClassName("*className*");
element.**getElementsByClassName**("*className*");

The first one returns all the nodes in the entire document with the specified class name. The second one returns all the nodes under the element, *element*.

---

11. Important properties of a node object

Informal difference between an element and a node: a node is more general than an element. Anything in an HTML document is a node (e.g. attributes, comments, doctype, etc), but only tags are elements.

Here is a partial list of node object properties:

- **childNodes**
  Returns a list of child nodes as an object of the type **nodeList**. . You can access a child node in the list by using the usual subscript notation.

  Example:
  Consider the JavaScript statement (in ThreeCities.html):

  var divNodes = document.getElementsByTagName("div");

  Here divNodes is an array of nodes with three elements (the three div elements).
  divNodes[0] refers to the div element with id="nyc".
  divNodes[1] refers to the div element with id="bos".
  and
  divNodes[0] refers to the div element with id="was".

  This statement,

  var nycChildren = divNodes[0].childNodes;

  Retrieves all the children of the first div node. Thus, nycChildren is an array containing the child nodes of (div id="nyc">. There are three child nodes.

  nycChildren[0] refers to the text node, "New York City".
  nycChildren[1] refers to the node <p>Nice City
  nycChildren[2] refers to the node <p>Busy city

- **nodeType**
  Returns an integer defining the type of this node. Two important types: The type value of an element node is 1 and that of a text node is 3.

  Continuing from the previous example,
  nycChildren[0] is of the type 3 (because it is a text node).
  nycChildren[1] and nycChildren[2] are of type 1 (because they are elements).

- **nodeName**
  Returns a string defining the name of this node. Two important results: For element node it returns the name of the tag and for text nodes it returns the string **#text**.

  Continuing our example,

Node	**nodeName**
nycChildren[0]	#text
nycChildren[1]	p
nycChildren[2]	p
divNodes[0]	div

- **nodeValue**
  Returns the value, if any, for the node. For element node it returns **null** and for textnodes it returns the actual text.

  Continuing our example,

Node	**nodeValue**
nycChildren[0]	New York City
nycChildren[1]	null
nycChildren[2]	null
divNodes[0]	null

- **tagName**
  Returns the tag name for element tag. Does not work for textnodes (returns "undefined").

  Continuing our example,

Node	**nodeName**
nycChildren[0]	undefined
nycChildren[1]	p
nycChildren[2]	p
divNodes[0]	div

- **firstChild**
  The first child of the node, if there are any, or **null**. This is a read-only attribute.
  Continuing our example,

Node	**firstChild**
nycChildren[0]	null (it is text node)
nycChildren[1]	Text node, "Nice City"

nycChildren[2]	Text node, " Busy city"
divNodes[0]	Text node, " New York City"
divNodes[1]	Text node, "Boston"

- **lastChild**
  The last child of the node, if there are any, or **None**. This is a read-only attribute.

- **nextSibling**
  The node that immediately follows this one with the same parent. If this is the last child of the parent, this attribute will be **None**.

- **previousSibling**
  Returns the previous (one before this one in a UP direction) **Node** that is a child of the current node. If there is none it will return NULL.

- **parentNode**
  Returns the parent **Node** of the current node. If there is none it will return NULL i.e. this is the topmost node in the document.

---

12. Action Methods – Manipulating Nodes

We have been looking and retrieving nodes and values from a DOM tree without changing the structure. We will now examine methods which will enable us to change the structure of the tree (meaning add/delete elements).

In examples below, we will use the HTML document, **ThreeCities.html**.

Here is ThreeCities.html and how a browser renders it:

ThreeCities.html document	**Browser display**

<!DOCTYPE html>
<html>
  <head>
  <title>A DOM example</title>
  </head>
  <body>
   New York City
  <p> Nice City
  <p> Busy city
 	
   Boston
  <p> Beautiful City
 	
   Washington
  <p> Capital city
 	
  </body>
</html>
	New York City
	Nice City
	Busy city
	Boston
	Beautiful City
	Washington
	Capital city

We will now discuss several methods to make changes to the HTML document.

- **Changing the value of a text node: nodeValue**

Syntax:

textNode.**nodeValue** = "*new value*";

Example:

Add this JavaScript code in ThreeCities.html:

```
<script>
var divNodes = document.getElementsByTagName("div");
divNodes[1].firstChild.nodeValue = "Providence";
</script>
```

The variable divNodes is an array of nodes. divNodes[1] refers to the node and divNodes[1].firstChild refers to the text node with value Boston.

divNodes[1].firstChild.nodeValue = "Providence";

Changes the value Boston to Providence.

Open the HTML file in a browser:

ThreeCities.html document with JavaScript	**Browser display**
<!DOCTYPE html> <html><head><title>A DOM example</title> </head> <body> New York City <p> Nice City <p> Busy city Boston <p> Beautiful City Washington <p> Capital city <script> var divNodes = document.getElementsByTagName("div"); divNodes[1].firstChild.nodeValue = "Providence"; </script> </body> </html>	New York City Nice City Busy city Providence Beautiful City Washington Capital city

Notice that in the display, Boston is replaced by Providence.

- **Creating a new text node: createTextNode()**

To create a new text node use the method **document**.**createTextNode("***textValue***")** method. Using this method just creates a new text node, we still need to place it in tree using some other method.

Example:
var newTextNode = document.createTextNode("sample")

newTextNode is the newly created text node.
Note:
The text node is just created. It is not added into DOM yet.

- **Creating a new element node: createElement**

To create a new element use the method **document**.**createElement**("*elementTag*") method. Using this method just creates a node, we still need to place it in tree using some other method.

Example:
var newDivNode = document.createElement("div");

A node, newDivNode is the newly created node.

- **Appending a child node: appendChild()**

You can append new child node to an existing node.

Syntax:

existingNode.**appendChild**(*newNode*);

Example:
var newTextNode = document.createTextNode("sample") /* Creates a new text node */
var newDivNode = document.createElement("div"); /*Creates a new node */

The following appends the new text node to the new node:

newDivNode.appendChild(newTextNode);

But, the new is not yet added into DOM.

Let us append the new element to element:

var divNodes = document.getElementsByTagName("div");
divNodes[1].appendChild(newDivNode);

See the whole result:

ThreeCities.html document with JavaScript	Browser display

<!DOCTYPE html>
<html><head><title>A DOM example</title>
</head>
<body>
 New York City
<p> Nice City
<p> Busy city
 	
 Boston
<p> Beautiful City
 	
 Washington
<p> Capital city
 	
<script>
var newTextNode = document.createTextNode("sample")
var newDivNode = document.createElement("div");
newDivNode.appendChild(newTextNode);
var divNodes = document.getElementsByTagName("div");
divNodes[1].appendChild(newDivNode);
</script>
</body>
</html>
	New York City
	Nice City
	Busy city
	Boston
	Beautiful City
	sample
	Washington
	Capital city

Notice (the browser display) that the new sample node is in the DOM.

- **Inserting new node before an existing node**

Syntax:
document.body.insertBefore(*newNode*, *existingNode*);

Example:
```
var newTextNode = document.createTextNode("sample")
var newDivNode = document.createElement("div");
newDivNode.appendChild(newTextNode);
var divNodes = document.getElementsByTagName("div");
document.body.insertBefore(newDivNode,divNodes[1]);
```

The new node is inserted before divNodes[1], which is the node.

See the whole result:

ThreeCities.html document with JavaScript	Browser display

<!DOCTYPE html>
<html><head><title>A DOM example</title>
</head>
<body>
 New York City
<p> Nice City
<p> Busy city
 	
 Boston
<p> Beautiful City
 	
 Washington
<p> Capital city
 	
<script>
var newTextNode = document.createTextNode("sample")
var newDivNode = document.createElement("div");
newDivNode.appendChild(newTextNode);
var divNodes = document.getElementsByTagName("div");
document.body.insertBefore(newDivNode,divNodes[1]);
</script>
</body>
</html>
	New York City
	Nice City
	Busy city
	sample
	Boston
	Beautiful City
	Washington
	Capital city

Notice sample before Boston in browser output.

Note:
To see how div node count changes after inserting a new div node, add this line in JavaScript,
document.write("<p>Number od div nodes: ",divNodes.length);

Add this twice: once before inserting the new node and once after:

```
var newTextNode = document.createTextNode("sample")
var newDivNode = document.createElement("div");
newDivNode.appendChild(newTextNode);
var divNodes = document.getElementsByTagName("div");
document.write("<p>Number of div nodes: ",divNodes.length);
document.body.insertBefore(newDivNode,divNodes[1]);
document.write("<p>Number of div nodes: ",divNodes.length);
```

This will print,

Number of div nodes: 3
Number of div nodes: 4

There are other methods to manipulate nodes in a DOM. Here is list of other methods, which we will not discuss in detail.

- **Adding additional text to a text node value: appendData()**

Syntax:

***textNode*.appendData**(" additional text to be added");

The additional text will be added to the current text value.

- **Deleting a portion of text node value: deleteData()**

Syntax:

***textNode*.deleteData**(n,m);

The two arguments n and m are two numbers. The **deleteData** method deletes m characters starting from character in position n. Character counting starts from 0.

- **Inserting new text into a text node value: insertData**

Syntax:

***textNode*.insertData**(n,"text to insert");

The **insertData** method inserts the specified characters in the second argument in the text node value starting from position n. Character counting starts from 0.

- **Replacing new text in place of certain text in a text node value: replaceData**

Syntax:

***textNode*.replaceData**(n,m,"text to replace");

The **replaceData** method replaces m characters in the text node value starting from position n by the specified characters in the third argument.

- **Returning a portion of a string from a text node value: substringData**

Syntax:
***textNode*.substringData**(n,m)

The **substringData** method returns the substring consisting of m characters in the text node value starting from position n.

- **Creating a new attribute to an element: createAttribute()**

The **createAttribute** method adds a new attribute to an existing element. This requires several steps.

First create a new attribute node:

newAttributeNode = **document.createAttribute**(*'attributeName'*);

Next assign a value to the new attribute:

***newAttributeNode*.nodeValue** = "value for the attribute";

Finally add the attribute to an existing node:

***existingNode*.setAttributeNode(*newAttributeNode*);**

- **To find out if a node has a specified attribute: hasAttribute**

Syntax:

***Node*.hasAttribute("*attributeName*");**

The **hasAttribute** method returns true of the node has an attribute specified in the argument. Otherwise it returns false.

- **To remove a specified attribute from an element: removeAttribute**

Syntax:
***Node*.removeAttribute("*attributeName*");**

Removes the attribute specified in the argument from the node.

- **To find out if a node has a child nodes: hasChildNodes**

Syntax:
***Node*.hasChildNodes();**
The **hasChildNodes()** method returns true of the node has child nodes. Otherwise it returns false.

---

13. The querySelector() method

We have seen three methods to retrieve nodes from a DOM tree:
getElementById() with argument an id value
getElementsByTagName() with argument a tag name
and
getElementsByClassName() with argument a class value.

When we discussed CSS properties, we considered three basic kinds of selectors: tagname, #idValue and .classname. As you know, a selector can be much more general than these basic kinds. For example, if you have an element <p> within another element , then **div p** is a selector identifying the <p> element within . Similarly there are several other complex selectors.

Now, if you like to retrieve nodes from a DOM tree by using these complex selectors, use **querySelector()** method.

Syntax:
document.querySelector("*selectors*")

Where *selectors* is any list of selectors, separated by commas. This method returns the first element that is a descendent of the element on which it is invoked that matches the specified group of selectors.

Example:

var pNode = document.querySelector("#was p");

The value of pNode is the first node under the selector "#was p".

That is, pNode is the element, <p> Capital city

Note:
The method, **querySelector()** has another version:
element.**quesrySelectorAll**("*selectors"*).

Returns a NodeList of all elements descended from the *element* on which it is invoked that match the specified group of CSS selectors.

---

14. The querySelectorAll() method

The **querySelectorAll**("*selectors*") method returns a nodeList of all the nodes that match the specified group of selectors. Where *selectors* is any list of selectors, separated by commas.

(Uses pre-order traversal of the document's nodes – not the order specified in the *selectors* list.)

Example:

var *pNodes* = **document**.**querySelectorAll**("#nyc p, #was p");

pNodes consists of these three <p> elements:
<p> Nice City
<p> Busy city
<p> Capital city

Note:
The method **querySelectorAll()** has another version:
element.**quesrySelectorAll**("*selectors"*).

Returns a NodeList of all elements descended from the *element* on which it is invoked that match the specified group of CSS selectors.

---

15. The *element*.InnerHTML property

The innerHTML of an element is the string content in the entire element. In other words, remove all the tags under the element, and the remaining text is the innerHTML.

You can use element.innerHTML in two ways:

var *content* = *element*.**innerHTML**;

This assigns the variable *content* the **inner.HTML** value of *element*.

element.**innerHTML** = "*newValue*";

The *newValue* replaces *element's* **innerHTML**.

Example:
Consider the HTML document, ThreeCities.html.

Try code,
```
var nycNode = document.getElementById("nyc");
document.write(nycNode.innerHTML);
```

Here nycNode is,
```
<div id="nyc"> New York City
   <p> Nice City
   <p> Busy city
</div>
```

To get the InnerHTML of this, remove all the tags, and get the remaining text:
New York City
Nice City
Busy city
