CHAPTER 5 – Working with the DOM

# Contents

[Understanding the Browser Object Model and the Document Object Model 1](#_Toc13443374)

[The Browser Object Model 1](#_Toc13443375)

[The Document Object Model 2](#_Toc13443376)

[The DOM and DHTML 2](#_Toc13443377)

[The DOM Tree 2](#_Toc13443378)

[DOM Document Object Methods 3](#_Toc13443379)

[DOM Document Object Properties 4](#_Toc13443380)

[Accessing Document Elements, Content, Properties, and Attributes 4](#_Toc13443381)

[Accessing Elements by id Value: 4](#_Toc13443382)

[Accessing Elements by Tag Name 4](#_Toc13443383)

[Accessing Elements by Class Name 5](#_Toc13443384)

[Accessing Elements by Name 5](#_Toc13443385)

[Accessing Elements with CSS Selectors 6](#_Toc13443386)

[Accessing an Element’s Content 7](#_Toc13443387)

[Accessing Elements Attributes 7](#_Toc13443388)

[Accessing Element Attributes 8](#_Toc13443389)

[Adding and Removing Document Nodes 8](#_Toc13443390)

[Creating Nodes 8](#_Toc13443391)

[Attaching Nodes 9](#_Toc13443392)

[Cloning Nodes 9](#_Toc13443393)

[Inserting Nodes at Specific Positions in the Document Tree 10](#_Toc13443394)

[Removing Nodes 12](#_Toc13443395)

[Manipulating the Browser with the Window Object 12](#_Toc13443396)

[Opening and Closing Windows and Tabs 13](#_Toc13443397)

[Opening a Window or Tab 14](#_Toc13443398)

[Closing a Window 15](#_Toc13443399)

[Working with Timeouts and Intervals 15](#_Toc13443400)

[Working with the History, Location, Navigation, and Screen objects 16](#_Toc13443401)

[The History Object 16](#_Toc13443402)

[The Location Object 17](#_Toc13443403)

[The Navigation Object 18](#_Toc13443404)

[The Screen Object 19](#_Toc13443405)

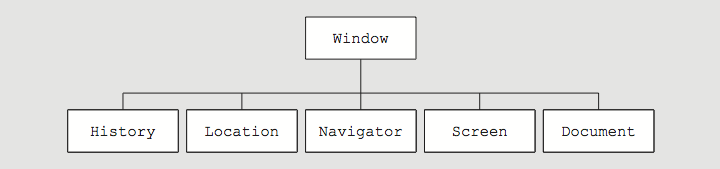
## Understanding the Browser Object Model and the Document Object Model

* JavaScript treats web page content as set of related components
  + objects
* Every element on web page is an object
  + You can also create objects - a function is an object

### The Browser Object Model

The browser object model (BOM) or client-side object model describes the relationship between object within the web browser, including within the current document.

The objects in the BOM are arranged in a hierarchy.



Although this is the model used by all major browsers, note that the BOM is not standard agreed by any organization. Instead, it is a general model that all browser manufacturers follow by convention.

You do not have to create any of the objects explicitly in the browser object model; they are created automatically when a web browser opens a web page.

The top-level object in the browser object model is the ***Window object***, which represents a web browser window. The Window object is called the ***global object*** because all other objects in the browser object model are contained within it.

For example, the Window object contains the Document object, just as a web browser window contains a web page document. You use the methods and properties of the Window object to control the web browser window, while you use the methods and properties of the Document object to control the web page.

### The Document Object Model

The Document Object is the most important object in the browser object model because it represents the web page displayed in a browser.

All elements on a web page are contained within the Document object, and each element is represented in JavaScript by its own object. This means that the Document object contains all the elements you create on a web page.

### The DOM and DHTML

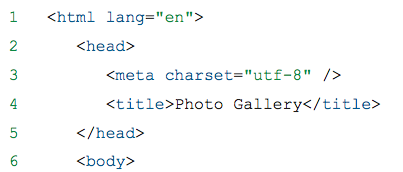
The combination of HTML and CSS with JavaScript, which enables the interactivity on the modern web, is sometimes referred to as ***dynamic HTML (DHTML).***

The DOM is what allows you to write JavaScript that changes the HTML and CSS of a web document. The DOM is an example of an ***application programming interface (API),*** which is a specification of how different software components can interact with each other. By codifying a structure for the objects in a web document, along with a standard set of properties and methods, the DOM creates tools for making web documents dynamic.

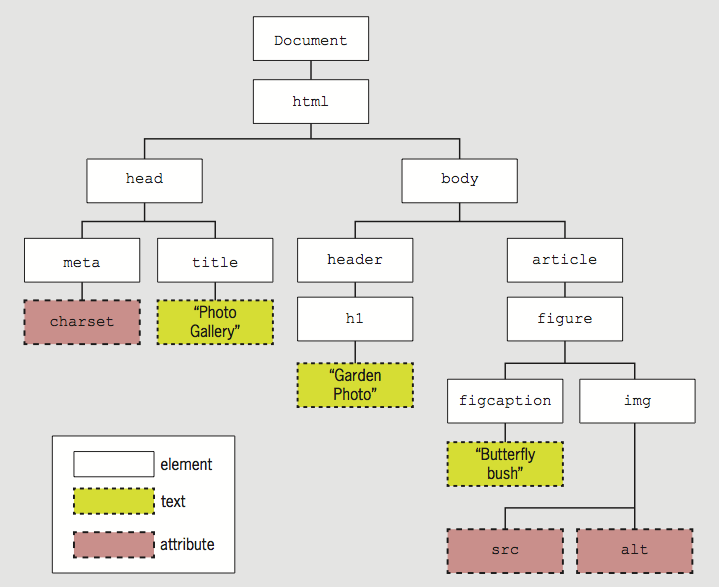
The DOM represents the HTML of a web page that is displayed in a browser. Each element on a web page is represented in the DOM by its own object. The fact that each element is an object makes it possible for a JavaScript app to access individual elements on a web page and change them individually, without having to reload the page from the server.

### 

### The DOM Tree

The DOM hierarchy depends on a document’s content

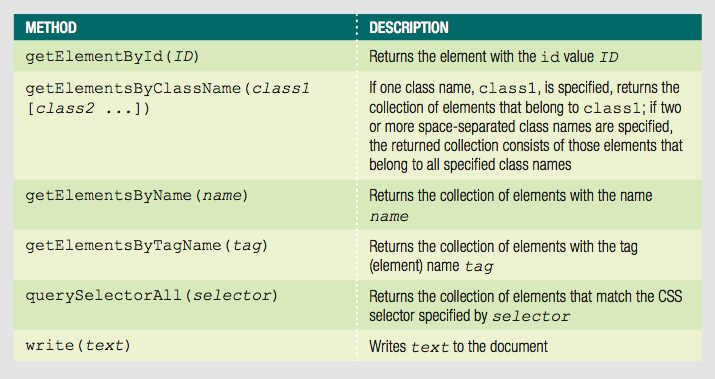




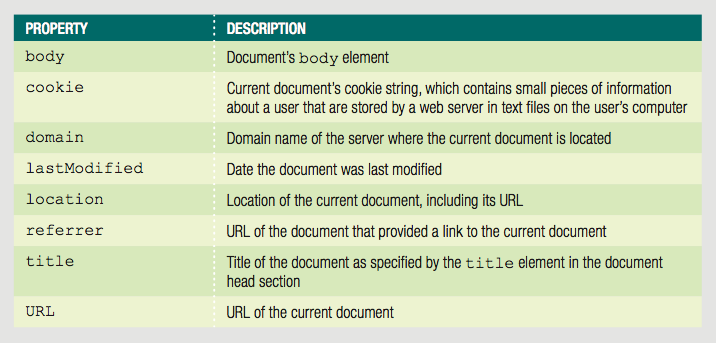
* Each item in the DOM tree is a node
* Element, attribute, and text content nodes are most commonly used

### DOM Document Object Methods

The DOM document object includes several methods used for dynamically generating web pages and manipulating elements.



### DOM Document Object Properties



## Accessing Document Elements, Content, Properties, and Attributes

### Accessing Elements by id Value:

**getElementById()**

* + Returns the first element in a document with a matching id attribute
  + If you need to directly access a specific element that has an id value

Example:

<input type="number" id="zip" />

var zipField = document.getElementById("zip");

### Accessing Elements by Tag Name

**getElementsByTagName()**

* + Returns array of elements matching a specified tag name in order by their appearance in the document.

Depending on the browser, this collection may be either a node list or an HTML collection.

* + Node list is an indexed collection of nodes
  + HTML collection is an indexed collection of HTML elements

Either set uses array syntax; this means you can reference a specific element on the page by using its index number in the collection.

Example:

var secondH1 = document.getElementsByTagName("h1")[1];

### Accessing Elements by Class Name

**getElementsByClassName()**

* + Returns node list or HTML collection of elements with a class attribute matching a specified value

Example - All elements with class value side:

var sideElements = document.getElementsByClassName("side");

* class attribute takes multiple values, so getElementsByClassName() method takes multiple arguments
* Arguments are enclosed in a single set of quotes, with class names separated by spaces

Example - All elements with class values side and green:

var sideGreenElements = document.getElementsByClassName("side green");

### Accessing Elements by Name

**getElementsByName()**

* + Returns node list or HTML collection of elements with a name attribute matching a specified value
  + Not as useful as preceding options; but creates more concise code when accessing set of option buttons or check boxes in a form.

Example:

<form>

<legend><span>Choose a color:</span></legend>

<input type="radio" name="color" id="redOption" value="red"/>

<label for="redOption">Red</label>

<input type="radio" name="color" id="greenOption" value="green"/>

<label for="greenOption">Green</label>

</form>

var colorButtons = document.getElementsByName("color");

### Accessing Elements with CSS Selectors

**querySelector()**

* + References elements using CSS syntax
  + Returns first occurrence of element matching a CSS selector

Example:

<header>

<h1><img class="logo" src="images/logo.png" alt="Blue Jay Photography" /></h1>

</header>

*JavaScript to reference img element:*

querySelector("header h1 img")

**querySelectorAll()**

* + Returns collection of elements matching selector

Example:

<nav>

<ul>

<li>About Us</li>

<li>Order</li>

<li>Support</li>

</ul>

</nav>

*JavaScript to reference all three li element:*

querySelectorAll("nav ul li")

### Accessing an Element’s Content

**textContent** property

* + Accesses and changes text that an element contains
  + Unlike innerHTML, textContent strips out HTML tags
  + textContent property is more secure
  + Not supported by IE8 or earlier
  + Some developers use if/else construction to implement textContent only on supported browsers

Example:

<ul>

<li class="topnav"><a href="aboutus.htm">About Us</a></li>

<li class="topnav"><a href="order.htm">Order</a></li>

<li class="topnav"><a href="support.htm">Support</a></li>

</ul>

*JavaScript to reference and access first li element:*

var button1 = querySelectorAll("li.topNav")[0];

var allContent = button1.innerHTML;

// <a href="aboutus.htm">About Us</a>

var justText = button1.textContent;

// About Us

### Accessing Elements Attributes

Can access CSS properties through DOM

* + Use dot notation(syntax) by adding a period and the name of the attribute after the element reference.

Example - change value of CSS display property to none for element with id value logo:

document.getElementById("logo").style.display = "none";

* + When a CSS property name includes hyphen (-), remove hyphen and capitalize letter following hyphen

Example - font-family becomes fontFamily

* When you specify a CSS value using a DOM reference, it is added as an inline style to the relevant element. This means that the setting generally has higher priority than styles set in embedded or external style sheet.

Example - to remove a style you previously added with a DOM reference, you simply set its

value to an empty string:

document.getElementById("navbar").style.color = "";

### Accessing Element Attributes

* Access element attribute with period and name of attribute after element reference.

Example – You can look up the value of the href attribute and assign that value to a variable, as follows:

var homeURL = document.getElementById("homeLink").href;

Example – You can use the attribute reference to assign a new value to the attribute, as follows:

document.getElementById("homeLink").href=”<http://w3.org>”;

## Adding and Removing Document Nodes

You’ve seen that you can change an element’s attribute values to alter web page content without reloading the page. The DOM also includes methods that let you create brand new elements and add or remove elements from the DOM tree.

## Creating Nodes

**createElement()** method

* + Creates a new element

Syntax:

**document.createElement("*element*");**

### Attaching Nodes

A Node that you create with the createElement() method is not automatically attacked to the DOM tree or to any other nodes. Instead, it exists independently of the DOM tree. The document object includes several methods for attaching nodes to the DOM. The most basic method is appendChild():

**parentNode.appendChild(childNode)**

Example:

* + Create new li element and attach to element with id value navList:

var list = document.getElementById("navList");

var contact = document.createElement("li");

list.**appendChild**(contact);

When you create two elements, for instance an img and figure element and add the img element as a child of the figure element, this creates a ***document fragment***. Document fragment is a set of connected nodes that are not part of a document. You’ll need to use the appendChild() method to add the document fragment to the DOM tree for the document, as a child of the article element.

Note that a node attached with the appendChild() method is always added as the last child element of the parent element.

Next you need to create the figure and img elements to display the first image in the gallery. Rather that create and configure new elements from scratch, you can create a copy of the document fragment you already created.

### Cloning Nodes

Sometimes you want to create a new node that is the same as or similar to an existing node in your document. You can simply use the cloneNode() method of the Document object to duplicate an existing node.

Syntax:

***existingNode*.cloneNode(true | false)**

true argument clones child nodes

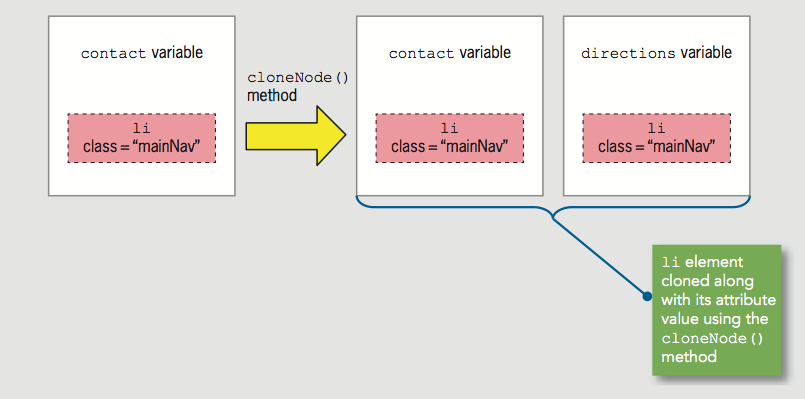
false argument clones only specified parent node

Example: The following code creates a new li element and assigns it to the contact variable, and then specifies a class name mainNav for this element. The final statement clones the contact node and stores the copy in the directions variable.

var contact = document.createElement("li");

contact.className = "mainNav";

var directions = contact.**cloneNode(true)**;



### Inserting Nodes at Specific Positions in the Document Tree

The createElement() method creates a node that is not attached to the DOM tree. The final step is creating new nodes is to add them to the DOM tree in the appropriate place. You can use the appendChild() method to add a node as a child of a parent node, however, the new child node is always appended after any existing child nodes.

If you need to add a node in a specific place among existing children of the same parent element, you instead use the insertBefore() method, which has the following syntax:

***parentNode*.insertBefore(*newChildNode*, *existingChildNode*)**

Example:

HTML –

<ul id="topnav">

<li><a href="aboutus.htm">About Us</a></li>

<li><a href="order.htm">Order</a></li>

<li><a href="support.htm">Support</a></li>

</ul>

JavaScript –

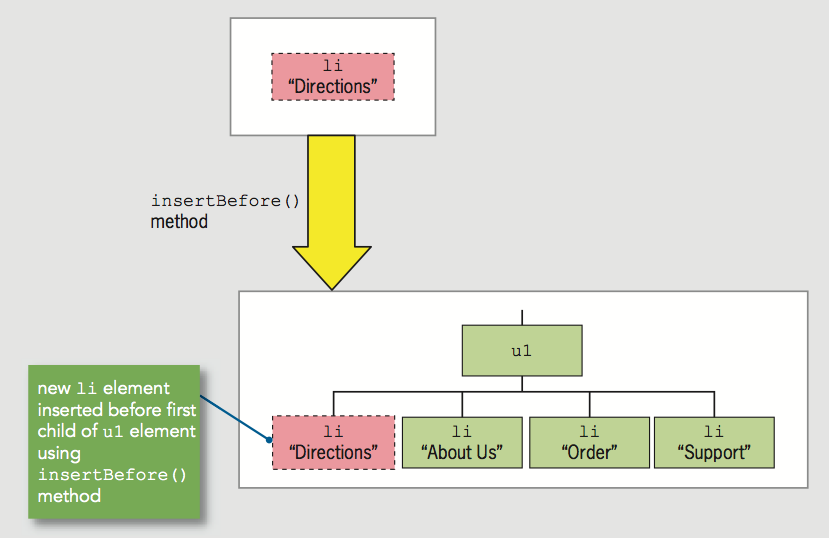
var list = document.getElementById("topnav");

var directions = document.createElement("li");

directions.innerHTML = "Directions";

var aboutus = document.querySelectorAll("#topnav li")[0];

list.**insertBefore(**directions, aboutus);



### Removing Nodes

**removeNode()**

Syntax:

***parentNode*.removeChild(*childNode*)**

Can assign removed node to variable:

var list = document.getElementById("topnav");

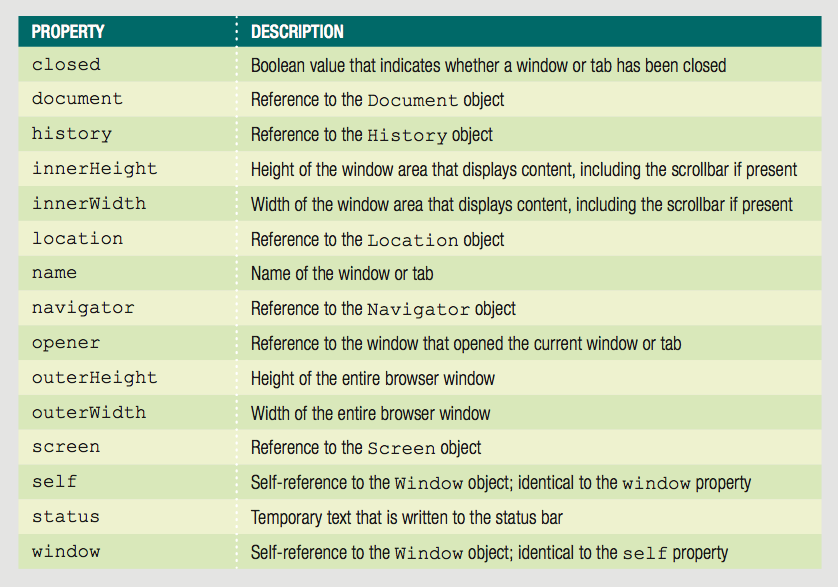
var aboutus = document.querySelectorAll("#topnav li")[0];

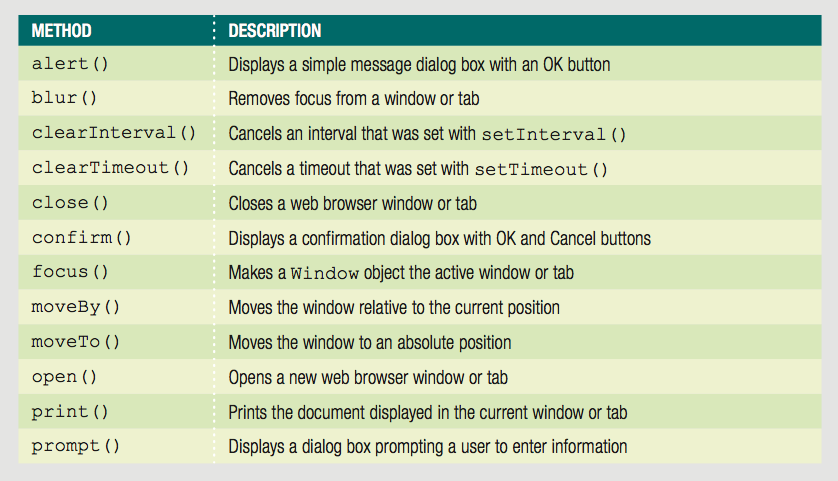
var aboutNode = list.**removeChild**(aboutus);

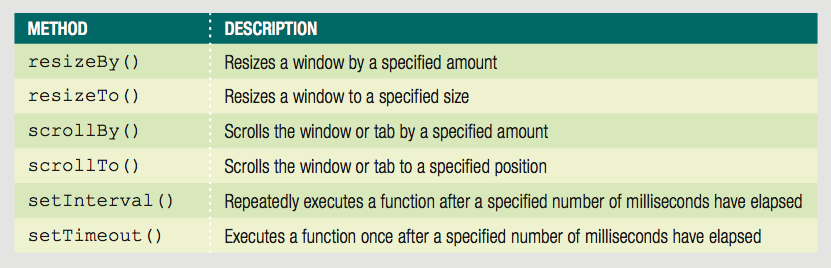
Node removed without being assigned to a variable is deleted during garbage collection.

## Manipulating the Browser with the Window Object

* Window object
  + Includes properties containing information about the web browser window or tab
  + Contains methods to manipulate the web browser window or tab itself







* **self** property
  + Refers to the current Window object
  + Identical to using the window property to refer to the Window object

Examples:

window.close();

self.close();

### Opening and Closing Windows and Tabs

* Reasons to open a new Web browser window
  + To launch a new Web page in a separate window
  + To use an additional window to display information
* When new Web browser window opened:
  + New Window object created

Represents the new window

* Know how to open a link in a new window using the a element’s target attribute:

Whenever a user clicks the preceding link, the browser first checks for a browser window or tab named wikiWindow. If the window or tab exists, then the link is opened in it. If the window or tab does not exist, then a new window or tab, name wikiWindow, is created where the link opens.

<a href="http://www.wikipedia.org/"↵

target="wikiWindow">Wikipedia home page</a>

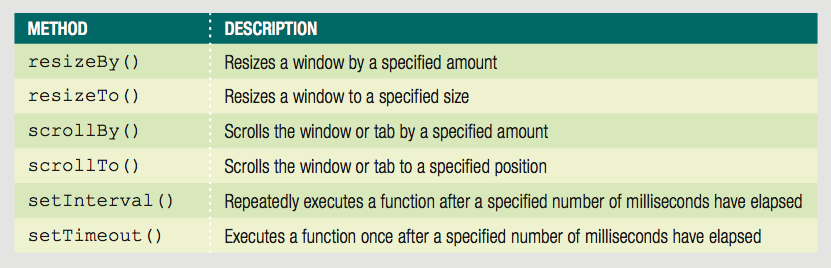
### Opening a Window or Tab

* **open()** method of the Window object
  + Opens new windows
* Include all (or none) window.open() method arguments

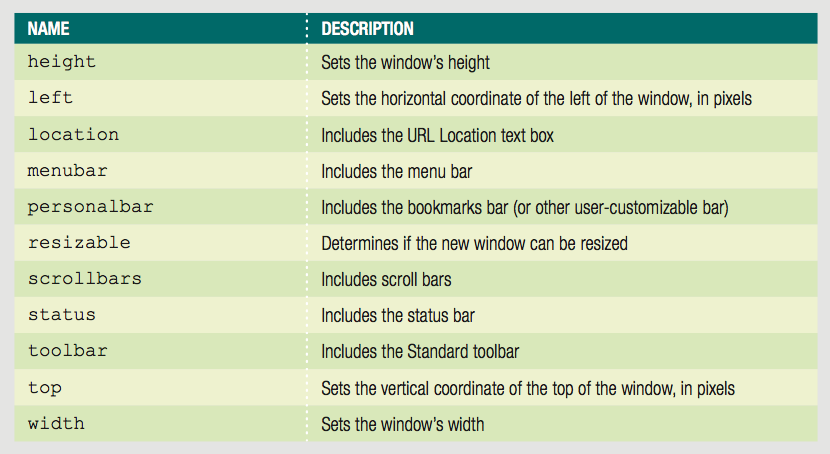
Syntax:

window.**open**(*url*, *name*, *options*, *replace*);

window.open("http://www.wikipedia.org");



* Customize new browser window or tab appearance
  + Use window.open() method options argument



* Window object’s name property used to specify a target window with a link
  + Cannot be used in JavaScript code
* Assign the new Window object created with the window.open() method to a variable to control it
* **focus()** method
  + Makes a window the active window

### Closing a Window

* **close()** method
  + Closes a web browser window
* window.close() or self.close()
  + Closes the current window

### Working with Timeouts and Intervals

* Window object’s timeout and interval methods
  + Creates code that executes automatically
* **setTimeout()** method
  + Executes code after a specific amount of time
  + Executes only once

Syntax:

var *variable* = **setTimeout("*code*", *milliseconds*);**

* **clearTimeout()** method
  + Cancel setTimeout() before its code executes

Example:

var buttonNotPressed = **setTimeout**("window.alert('Your↵

changes have been saved')",10000);

function buttonPressed() {

**clearTimeout**(buttonNotPressed);

window.open(index.htm);

* **setInterval()** method
  + Repeatedly executes the same code after being called only once

Syntax:

var *variable* = setInterval("*code*", *milliseconds*);

* **clearInterval()** method
  + Used to clear setInterval() method call

## Working with the History, Location, Navigation, and Screen objects

### The History Object

* History object

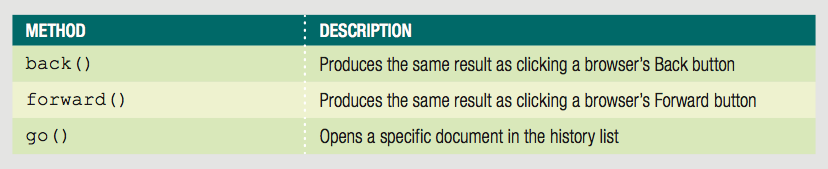
Maintains internal list (history list)

All documents opened during current web browser session

* Security features

Will not display URLs contained in the history list, but you can write a script that uses the history list to navigate to web pages that have been opened during a web browser session.

The History object includes three methods:



* **go()** method

Allows navigation to a specific previously visited web page

* History object length property

Provides specific number of documents opened during the current browser session

Example - Return to first document opened in current browser session:

history.**go(**-(history.length - 1));

### The Location Object

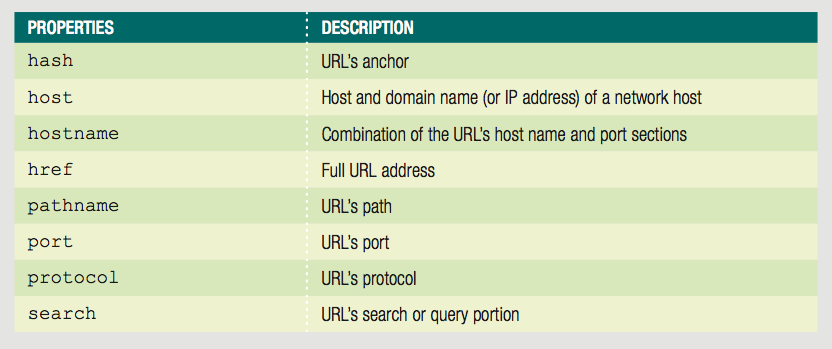
* Location object

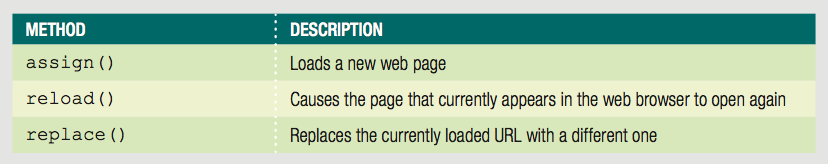
Allows changes to a new web page from within JavaScript code

One reason you may want to change web pages with JavaScript code is to briefly display a message letting a user know that the action they took – such as submitting a request or changing account information – was successful, and then automatically redirect the visitor back to the content they were viewing before making their request.

* Location object properties allow modification of URL individual portions

Web browser automatically attempts to open that new URL





* Location object’s **assign()** method

Same action as changing the href property

Loads a new web page

Example – the following two statements are equivalent:

**location**.assign(“http://www.google.com”);

**Location**.href = “http://www.google.com”;

* Location object’s **reload()** method

Equivalent to the browser Reload or Refresh button

Causes current page to open again

* Location object’s **replace()** method

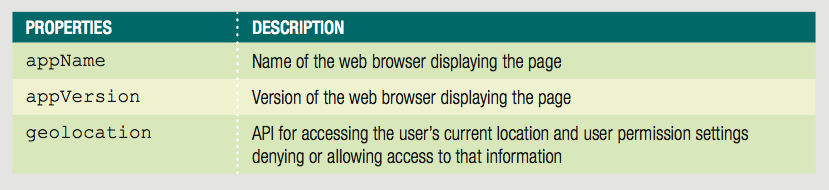
Replaces currently loaded URL with a different one

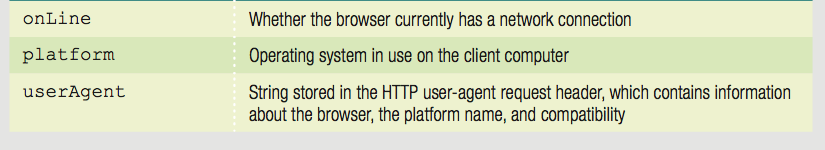
### The Navigation Object

* Navigator object

Obtains information about current web browser

Example: determine type of web browser running





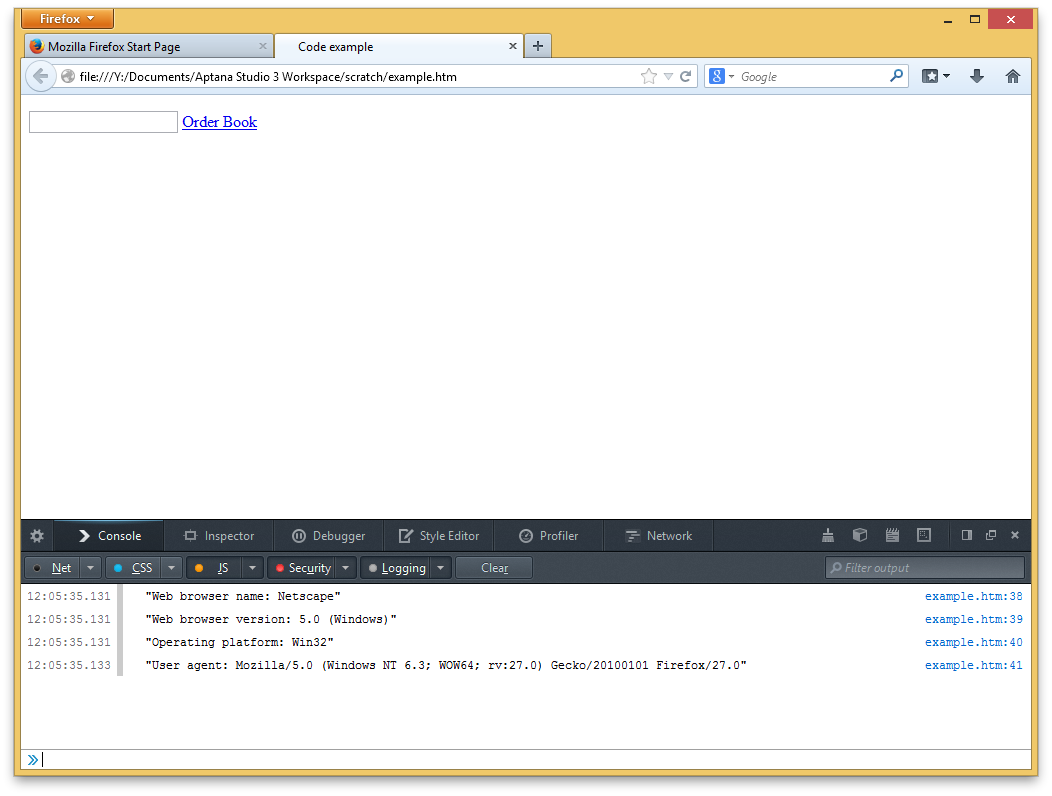
Example:

console.log("Web browser name: " + **navigator**.appName);

console.log("Web browser version: " + **navigator**.appVersion);

console.log("Operating platform: " + **navigator**.platform);

console.log("User agent: " + **navigator**.userAgent);



In the past, the Navigator object was commonly used to determine which type of browser was running, in order to execute the correct code for different implementations of JavaScript. Today, JavaScript implementations are standardized across the current versions of all major browsers.

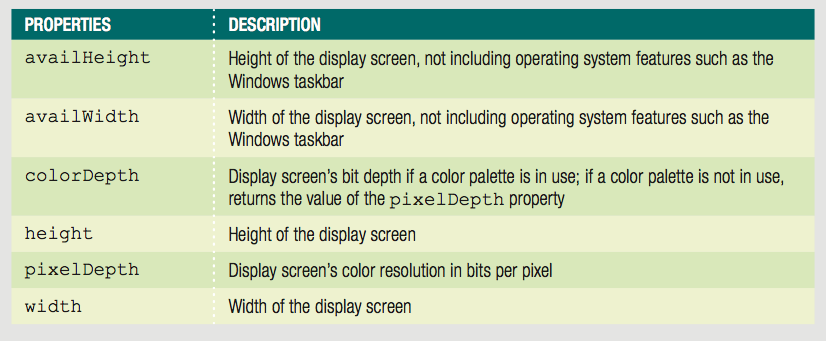
### The Screen Object

* Screen object

Obtains information about display screen’s size, resolution, color depth

* Common use of Screen object properties

Centering a web browser window in the middle of the display area



Example:

var winWidth = 300;

var winHeight = 200;

var leftPosition = (screen.width - winWidth) / 2;

var topPosition = (screen.height - winHeight) / 2;

var optionString = "width=" + winWidth + ",height="↵

+ winHeight + ",left=" + leftPosition + ",top="↵

+ topPosition;

var openWin = window.open("", "CtrlWindow", optionString);

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<title>Photo gallery</title>

<link rel="stylesheet" href="photos.css" />

<script src="modernizr.custom.05819.js"></script>

</head>

<body>

<div>

<header>

<h1>Photo gallery</h1>

</header>

<article>

<div id="leftarrow">

<p>&lt;</p>

</div>

<figure id="fig2">

<img width="360" height="202" />

</figure>

<figure id="fig3">

<img width="480" height="270" />

</figure>

<figure id="fig4">

<img width="360" height="202" />

</figure>

<div id="rightarrow">

<p>&gt;</p>

</div>

<div id="fiveButton">

<p>Show more images</p>

</div>

</article>

</div>

<script src="photos.js"></script>

</body>

</html>

"use strict"; // interpret document contents in JavaScript strict mode

/\* global variables \*/

var photoOrder = [1,2,3,4,5];

var figureCount = 3;

//makes the photo gallery images advance automatically

var autoAdvance = setInterval(rightAdvance,2000);

/\*add src values to img elements based on order specified in photoOrder array \*/

function populateFigures() {

var filename;

var currentFig;

//ensures that when five images are displayed in the app

if (figureCount === 3) {

for (var i = 1; i < 4; i++) {

filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";

currentFig = document.getElementsByTagName("img")[i - 1];

currentFig.src = filename;

}

}else{

for (var i = 0; i < 5; i++) {

filename = "images/IMG\_0" + photoOrder[i] + "sm.jpg";

currentFig = document.getElementsByTagName("img")[i];

currentFig.src = filename;

}

}

}

/\* shift all images one figure to the left, and change values in photoOrder array to match \*/

function rightAdvance() {

for (var i = 0; i < 5; i++) {

if ((photoOrder[i] + 1) === 6) {

photoOrder[i] = 1;

} else {

photoOrder[i] += 1;

}

populateFigures();

}

}

/\* stop automatic image switching and call rightAdvance function \*/

function rightArrow(){

clearInterval(autoAdvance);

rightAdvance();

}

/\* shift all images one figure to the right, and change values in photoOrder array to match \*/

function leftArrow() {

//cancel the photo gallery slide show

clearInterval(autoAdvance);

for (var i = 0; i < 5; i++) {

if ((photoOrder[i] - 1) === 0) {

photoOrder[i] = 5;

} else {

photoOrder[i] -= 1;

}

populateFigures();

}

}

/\* switch to 5-image layout - (creating nodes) \*/

function previewFive(){

//create figure and img elements for fifth image

var lastFigure = document.createElement("figure");

lastFigure.id="fig5";

lastFigure.style.zIndex="5";

lastFigure.style.position="absolute";

lastFigure.style.right="45px";

lastFigure.style.top="67px";

var lastImage = document.createElement("img");

lastImage.width="240";

lastImage.height="135";

// appending the img and figure document fragments to the DOM tree:

var articleEl = document.getElementsByTagName("article")[0];

lastFigure.appendChild(lastImage);

//inserting the lastFigure nodes in the appropriate places in the DOM tree.

articleEl.insertBefore(lastFigure,document.getElementById("rightarrow"));

//clone the lastFigure node to create the firstFigure node:

var firstFigure = lastFigure.cloneNode(true);

firstFigure.id="fig1";

firstFigure.style.right="";

firstFigure.style.left="45px";

//inserting the firstFigure nodes in the appropriate places in the DOM tree.

articleEl.insertBefore(firstFigure,document.getElementById("fig2"));

//add appropriate src values to two new img elements

document.getElementsByTagName("img")[0].src = "images/IMG\_0"+photoOrder[0]+"sm.jpg";

document.getElementsByTagName("img")[4].src = "images/IMG\_0"+photoOrder[4]+"sm.jpg";

//by default 3; when user switches to viewing five images images.

figureCount = 5;

//we give the user the option of returning to the original layout

//change button to hide extra images

var numberButton = document.querySelector("#fiveButton p");

numberButton.innerHTML = "Show fewer images";

if(numberButton.addEventListener){

numberButton.removeEventListener("click",previewFive,false);

numberButton.addEventListener("click",previewThree,false);

}else if(numberButton.attachEvent){

numberButton.detachEvent("onclick",previewFive);

numberButton.attachEvent("onclick",previewThree);

}

}

/\* switch to 3-image layout \*/

function previewThree(){

var articleEl = document.getElementsByTagName("article")[0];

var numberButton = document.querySelector("#fiveButton p");

//remove the first and fifth fugure elements

articleEl.removeChild(document.getElementById("fig1"));

articleEl.removeChild(document.getElementById("fig5"));

//set back to default value with 3 images displayed

figureCount = 3

//change the button display back to show more images

numberButton.innerHTML = "Show more images";

if(numberButton.addEventListener){

numberButton.removeEventListener("click", previewThree,false);

numberButton.addEventListener("click",previewFive,false);

}else if(numberButton.attachEvent){

numberButton.detachEvent("onclick",previewThree);

numberButton.attachEvent("onclick",previewFive);

}

}

/\* open center figure in separate window \*/

function zoomFig() {

//images is centered on the user's screen:

var propertyWidth = 960;

var propertyHeight = 600;

var winLeft = ((screen.width - propertyWidth) / 2 );

var winTop = ((screen.height - propertyHeight) / 2);

var winOptions = "width=960,height=600";

winOptions += ",left=" +winLeft;

winOptions += ",top=" + winTop;

var zoomWindow = window.open("zoom.htm", "zoomwin",winOptions);

zoomWindow.focus();

}

/\* create event listeners and populate image elements \*/

function setUpPage() {

createEventListeners();

populateFigures();

}

/\* run setUpPage() function when page finishes loading \*/

if (window.addEventListener) {

window.addEventListener("load", setUpPage, false);

} else if (window.attachEvent) {

window.attachEvent("onload", setUpPage);

}

/\*create event listeners for the left arrow, right arrow and center figure element \*/

function createEventListeners(){

var leftarrow = document.getElementById("leftarrow");

if(leftarrow.addEventListener){

leftarrow.addEventListener("click",leftArrow,false);

} else if(leftarrow.attachEvent){

leftarrow.attachEvent("onclick",leftArrow);

}

var rightarrow = document.getElementById("rightarrow");

if(rightarrow.addEventListener){

rightarrow.addEventListener("click",rightArrow,false);

}else if(rightarrow.attachEvent){

rightarrow.attachEvent("onclick",rightArrow);

}

//this code calls a function when user clicks the middle image in the gallery.

var mainFig = document.getElementsByTagName("img")[1];

if(mainFig.addEventListener){

mainFig.addEventListener("click",zoomFig,false);

}else if(mainFig.attachEvent){

mainFig.attachEvent("onclick",zoomFig);

}

var showAllButton = document.querySelector("#fiveButton p");

if(showAllButton.addEventListener){

showAllButton.addEventListener("click",previewFive,false);

}else if(showAllButton.attachEvent){

showAllButton.attachEvent("onclick",previewFive);

}

}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<title>Full size photo</title>

<link rel="stylesheet" href="zoom.css" />

<!--[if lt IE 9]>

<link rel="stylesheet" href="zoomie.css" />

<![endif]-->

<script src="modernizr.custom.05819.js"></script>

</head>

<body>

<figure>

<img src="" width="960" height="540" />

</figure>

<footer>

<div><p>Close Window</p></div>

</footer>

<script src="zoom.js"></script>

</body>

</html>

"use strict"; // interpret document contents in JavaScript strict mode

/\* global variables \*/

var photoOrderArray = window.opener.photoOrder;

var figFilename = "images/IMG\_0" + photoOrderArray[2] + "jpg";

/\* populate img element and create event listener \*/

function pageSetup() {

document.getElementsByTagName("img")[0].src = figFilename; // assign filename to img element

createEventListener();

}

/\* close window \*/

function closeWin(){

window.close();

}

/\* create event listener for close button \*/

function createEventListener(){

var closeWindowDiv = document.getElementsByTagName("p")[0];

if(closeWindowDiv.addEventListener){

closeWindowDiv.addEventListener("click",closeWin,false);

}else if(closeWindowDiv.attachEvent){

closeWindowDiv.attachEvent("onclick",closeWin);

}

}

/\* add img src value and create event listener when page finishes loading \*/

window.onload = pageSetup;