mood-book

HTML FORMS:

The <form> Element

HTML forms are used to collect user input.

The **<form>** element defines an HTML form:

```
<form>
.
form elements
.
</form>
```

HTML forms contain **form elements**.

Form elements are different types of input elements, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

The **<input>** element is the most important **form element**.

The <input> element has many variations, depending on the **type** attribute.

Here are the types used in this chapter:

Type	Description
text	Defines normal text input
radio	Defines radio button input (for selecting one of many choices)
submit	Defines a submit button (for submitting the form)

Text Input

<input type="text"> defines a one-line input field for **text input**

Radio Button Input

<input type="radio"> defines a **radio button**.

Radio buttons let a user select ONE of a limited number of choices

The Submit Button

<input type="submit"> defines a button for **submitting** a form to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's **action** attribute

The Action Attribute

The **action attribute** defines the action to be performed when the form is submitted.

The common way to submit a form to a server, is by using a submit button.

Normally, the form is submitted to a web page on a web server.

In the example above, a server-side script is specified to handle the submitted form:

<form **action="action_page.php**">

If the action attribute is omitted, the action is set to the current page.

The Method Attribute

The **method attribute** specifies the HTTP method (**GET** or **POST**) to be used when submitting the forms:

<form action="action_page.php" **method="GET"**>

or:

<form action="action_page.php" **method="POST"**>

- **When to Use GET?**

If the form submission is passive (like a search engine query), and without sensitive information.
When you use GET, the form data will be visible in the page address:
action_page.php?firstname=Mickey&lastname=Mouse

- **When to Use POST?**

If the form is updating data, or includes sensitive information (password).
POST offers better security because the submitted data is not visible in the page address.

The Name Attribute

To be submitted correctly, each input field must have a name attribute.

Grouping Form Data with <fieldset>

The **<fieldset>** element groups related data in a form.
The **<legend>** element defines a caption for the <fieldset> element.

HTML Form Attributes

An HTML <form> element, with all possible attributes set, will look like this:
<form action="action_page.php" method="GET" target="_blank" accept-charset="UTF-8" enctype="application/x-www-form-urlencoded" autocomplete="off" novalidate>
.
form elements
.
</form>
Here is the list of <form> attributes:

Attribute	**Description**
accept-charset	Specifies the charset used in the submitted form (default: the page charset).
action	Specifies an address (url) where to submit the form (default: the submitting page).
autocomplete	Specifies if the browser should autocomplete the form (default: on).
enctype	Specifies the encoding of the submitted data (default: is url-encoded).
method	Specifies the HTTP method used when submitting the form (default: GET).
name	Specifies a name used to identify the form (for DOM usage: document.forms.name).
novalidate	Specifies that the browser should not validate the form.
target	Specifies the target of the address in the action attribute (default: _self).

HTML FORM ELEMENTS

The <input> Element

The most important form element is the **<input>** element.
The <input> element can vary in many ways, depending on the **type** attribute.

The <select> Element (Drop-Down List)

The **<select>** element defines a **drop-down** list:
Example
<select name="cars">

```
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat">Fiat</option>
<option value="audi">Audi</option>
</select>
```

The **<option>** elements defines the options to select.
The list will normally show the first item as selected.
You can add a selected attribute to define a predefined option.
Example

```
<option value="fiat" selected>Fiat</option>
```

The <textarea> Element

The **<textarea>** element defines a multi-line input field (**a text area**):

The <button> Element

The **<button>** element defines a clickable **button**:
Example

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

HTML5 Form Elements

HTML5 added the following form elements:

- <datalist>
- <keygen>
- <output>

HTML5 <datalist> Element

The **<datalist>** element specifies a list of pre-defined options for an <input> element.
Users will see a drop-down list of pre-defined options as they input data.
The **list** attribute of the <input> element, must refer to the **id** attribute of the <datalist> element.

HTML5 <keygen> Element

The purpose of the **<keygen>** element is to provide a secure way to authenticate users.
The <keygen> element specifies a key-pair generator field in a form.
When the form is submitted, two keys are generated, one private and one public.
The private key is stored locally, and the public key is sent to the server.
The public key could be used to generate a client certificate to authenticate the user in the future.

HTML5 <output> Element

The <output> element represents the result of a calculation (like one performed by a script).

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element

<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop-down list
<optgroup>	Defines a group of related options in a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button
<datalist>	Specifies a list of pre-defined options for input controls
<keygen>	Defines a key-pair generator field (for forms)
<output>	Defines the result of a calculation

HTML Frameset:

The <frameset> tag is not supported in HTML5.

The <frameset> tag defines a frameset.

The <frameset> element holds one or more <frame> elements. Each <frame> element can hold a separate document.

The <frameset> element specifies HOW MANY columns or rows there will be in the frameset, and HOW MUCH percentage/pixels of space will occupy each of them.

Example

A simple three-framed page:

```
<frameset cols="25%,*,25%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
```

SCRIPTING BASICS

A scripting language is a form of programming language that is usually interpreted rather than compiled.

Conventional programs are converted permanently into executable files before they are run.

In contrast, programs in scripting language are interpreted one command at a time.

Scripting languages are often written to facilitate enhanced features of Web sites. These features are processed on the server but the script in a specific page runs on the user's browser.

In most cases, it is easier to write the code in a scripting language than in a compiled language. However, scripting languages are slower because the instructions are not handled solely by the basic instruction processor.

Scripting languages allow rapid development and can communicate easily with programs written in other languages.

Scripting languages can be used to create specialized GUIs (graphical user interfaces) and forms that enhance the convenience of search engines, Web-based e-mail and e-commerce.

Many Web sites require that the user's browser be set to run scripts to take advantage of all the features of the site. In some cases, Web sites are practically useless unless the user's computer is set to run programs locally in a scripting language.

Client-side Scripting Environment (Browsers and Editors)

The World Wide Web (WWW) began as a text-only medium-the first browsers didn't even support images within web pages.

Today's websites can include a lot of features: graphics, sounds, animation, video, and occasionally useful content.

Web scripting languages also known as client-side scripting languages, such as JavaScript, are one of the easiest ways to add dynamic state to a web page and to interact with users in new ways.

HTML is a simple text markup language, it can't respond to the user, make decisions, or automate repetitive tasks. Interactive tasks such as these require a more sophisticated language: a programming language, or a scripting language.

Although many programming languages are complex, scripting languages are generally simple. They have a simple syntax, can perform tasks with a minimum of commands, and are easy to learn. Web scripting languages enable you to combine scripting with HTML to create interactive web pages.

Scripts and Programs

A movie or a play follows a script-a list of actions (or lines) for the actors to perform. A web script provides the same type of instructions for the web browser. A script can range from a single line to a full-scale application. (In either case, Scripts usually run within a browser.)

Some programming languages must be compiled, or translated, into machine code before they can be executed. Client-side scripts, on the other hand, are an interpreted language: The browser executes each line of script as it comes to it.

There is one main advantage to interpreted languages: Writing or changing a script is very simple. Changing a client-side script is as easy as changing a typical HTML document, and the change is executed as soon as you refresh the document in the browser.

Characteristics of Interpreted Languages

One of the main benefits of interpreted languages is that they require no compilation.

The language is interpreted at run-time so the instructions are executed immediately.

Any errors in an interpreted program will result in the execution of the code to be halted.

Interpreted languages also have a simple syntax which, for the user:

- makes them easy to learn and use
- assumes minimum programming knowledge or experience
- allows complex tasks to be performed in relatively few steps
- allows simple creation and editing in a variety of text editors
- allows the addition of dynamic and interactive activities to web pages

Also, interpreted languages are generally portable across various hardware and network platforms and scripts can be embedded in standard text documents for added functionality.

Unlike a compiler, an interpreter checks syntax and generates object code one source line at a time. Think of this as very similar to a group of translators at a United Nations' Conference, who each have to convert sentences spoken by delegates into the native language of their representative.

When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter executes each line of code at a time the programmer is able to see the results of their programs immediately which can also help with debugging.

Advantages and Disadvantages of Interpreted Languages

Advantages

- easy to learn and use
- minimum programming knowledge or experience
- allows complex tasks to be performed in relatively few steps
- allows simple creation and editing in a variety of text editors
- allows the addition of dynamic and interactive activities to web pages

- edit and running of code is fast.

Disadvantages

- usually run quite slowly
- limited access to low level and speed optimization code.
- limited commands to run detailed operations on graphics.

<u>Differences between Client-side and Server-side Scripting</u>

Client-side Environment

The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web server to the users computer over the internet and run directly in the browser.

The scripting language needs to be **enabled** on the client computer. Sometimes if a user is conscious of **security risks** they may switch the scripting facility off. When this is the case a message usually pops up to alert the user when script is attempting to run.

Server-side Environment

The **server-side environment** that runs a scripting language is a web server. A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server.

This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

Types of scripting languages

Specialised scripting languages include:

Perl (Practical Extraction and Report Language). This is a popular string processing language for writing small scripts for system administrators and web site maintainers. Much web development is now done using Perl.

Hypertalk is another example. It is the underlying scripting language of HyperCard.

Lingo is the scripting language of Macromedia Director, an authoring system for develop high-performance multimedia content and applications for CDs, DVDs and the Internet.

AppleScript, a scripting language for the Macintosh allows the user to send commands to the operating system to, for example open applications, carry out complex data operations.

JavaScript, perhaps the most publicised and well-known scripting language was initially developed by Netscape as LiveScript to allow more functionality and enhancement to web page authoring that raw HTML could not accommodate. A standard version of JavaScript was later developed to work in both Netscape and Microsoft's Internet Explorer, thus making the language to a large extent, universal. This means that JavaScript code can run on any platform that has a JavaScript interpreter.
VBScript, a cut-down version of Visual Basic, used to enhance the features of web pages in Internet Explorer.

INTRODUCING JAVASCRIPT

JavaScript was developed by Netscape Communications Corporation, the maker of the Netscape web browser. JavaScript was the first web scripting language to be supported by browsers, and it is still by far the most popular.
JavaScript was originally called LiveScript and was first introduced in Netscape Navigator 2.0 in 1995. Netscape Navigator was an early browser that held a large share of the browser market. It was soon renamed JavaScript to indicate a marketing relationship with Sun's Java language.
JavaScript is almost as easy to learn as HTML, and it can be included directly in HTML documents.

What can you do with Javascript?

- Display messages to the user as part of a web page, in the browser's status line, or in alert boxes.
- Validate the contents of a form and make calculations (for example, an order form can automatically display a running total as you enter item quantities).
- Animate images or create images that change when you move the mouse over them.
- Create ad banners that interact with the user, rather than simply displaying a graphic.
- Detect the browser in use or its features and perform advanced functions only on browsers that support them.
- Detect installed plug-ins and notify the user if a plug-in is required.
- Modify all or part of a web page without requiring the user to reload it.
- Display or interact with data retrieved from a remote server.

JavaScript Browser Support

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. The procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera are listed below.

JavaScript in Internet Explorer

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer −

- Follow **Tools → Internet Options** from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find **Scripting option.**
- Select *Enable* radio button under **Active scripting**.
- Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox −

- Open a new tab → type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I'll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option → **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toogle**. If javascript is disabled; it gets enabled upon clicking toggle.

JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome −

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.

- Under the **Privacy** section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera −

- Follow **Tools** → **Preferences** from the menu.
- Select **Advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

To disable JavaScript support in your Opera, you should not select the **Enable JavaScript checkbox**.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.
You can add a **noscript** block immediately after the script block as follows −

```
<html>
  <body>
    <script language="javascript" type="text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
    <noscript>
      Sorry...JavaScript is needed to go ahead.
    </noscript>
  </body>
</html>
```

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

Where to put Javascript?

JavaScript can be put in the <head> or in the <body> of an HTML document

- JavaScript *functions* should be defined in the <head>. This ensures that the function is loaded before it is needed
- JavaScript in the <body> will be executed as the page loads

JavaScript can be put in a separate .js file

- <script src="myJavaScriptFile.js"></script>

Put this HTML wherever you would put the actual JavaScript code

An external .js file lets you use the same JavaScript on multiple HTML pages

The external .js file cannot itself contain a <script> tag

JavaScript can be put in HTML *form object*, such as a button

This JavaScript will be executed when the form object is used

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by the computer.

In a programming language, these program instructions are called **statements**.

JavaScript is a **programming language**.

JavaScript statements are separated by **semicolons**.

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Statements</h1>
<p>Statements are separated by semicolons.</p>
<p>The variables x, y, and z are assigned the values 5, 6, and 11:</p>
<p id="demo"></p>
<script>
var x = 5;
var y = 6;
var z = x + y;
```

```
document.getElementById("demo").innerHTML = z;
</script>
</body>
</html>
```

JAVASCRIPT STATEMENTS

JavaScript statements are composed of:
Values, Operators, Expressions, Keywords, and Comments.

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.
Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Literals

- Array literals
- Boolean literals
- Floating-point literals
- Integers
- Object literals
- String literals

JavaScript : Array literals

In Javascript an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [ ] ' . When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified. If no value is supplied it creates an empty array with zero length.

Creating an empty array :
var fruits = [ ];
Creating an array with four elements.
var fruits = ["Orange", "Apple", "Banana", "Mango"]

Comma in array literals

There is no need to specify all elements in an array literal. If we put two commas in a row at any position in an array then an unspecified element will be created in that place.
The following example creates the fruits array :
fruits = ["Orange", , "Mango"]
This array has one empty element in the middle and two elements with values. (fruits[0] is "Orange", fruits[1] is set to undefined, and fruits[2] is "Mango").
If you include a single comma at the end of the elements, the comma is ignored. In the following example, the length of the array is three. There is no fruits[2].
fruits = ["Orange", "Mango",]
In the following example, the length of the array is four, and fruits[0] and fruits[2] are undefined.
fruits = [, 'Apple', , 'Orange'];

JavaScript : Integers literals

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative, if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.
1. Decimal (base 10)
Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.
Example : 123, -20, 12345
2. Hexadecimal (base 16)
Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f. A leading 0x or 0X indicates the number is hexadecimal.
Example : 7b, -14, 3039
3. Octal (base 8)
Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.
Example : 173, -24, 30071

JavaScript : Floating number literals

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Example of some floating numbers :

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4 x 10^3]
- 4E-3 [Equivalent to 4 x 10^{-3} => .004]

JavaScript : Boolean literals

The Boolean type has two literal values :

- true
- false

JavaScript : Object literals

An object literal is zero or more pairs of comma separated list of property names and associated values, enclosed by a pair of curly braces.
In JavaScript an object literal is declared as follows:
1. An object literal without properties:
var userObject = {}
2. An object literal with a few properties :
var student = {
First-name : "Suresy",
Last-name : "Rayy",

Roll-No : 12
};

Syntax Rules

Object literals maintain the following syntax rules:

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.
- There will be no comma after the last property name/value pair.

JavaScript : String literals

JavaScript has its own way to deal with string literals. A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings. The following are the examples of string literals :

- string1 = "w3resource.com"
- string1 = 'w3resource.com'
- string1 = "1000"
- string1 = "google" + ".com"

In addition to ordinary characters, you can include special characters in strings, as shown in the following table.
string1 = "First line. \n Second line."

List of special characters used in JavaScript string :

Character	**Meaning**
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return

	overwrite the characters previously output on that line.
\t	Tab
\'	Single quote
\"	Double quote
\\	Backslash character (\)
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. For example, \100 is the octal sequence for the @ symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. For example, \x40 is the hexadecimal sequence for the @ symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u0040 is the Unicode sequence for the @ symbol.

The most important rules for writing fixed values are:

Numbers are written with or without decimals:

10.50

1001

Strings are text, written within double or single quotes:

"John Doe"

'John Doe'

Expressions can also represent fixed values:

5 + 6

5 * 10

JAVASCRIPT VARIABLES

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **define** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
var x;
x = 6;
```

Variables names must begin with a letter or underscore

Variable names are case-sensitive. All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

```
lastName = "Doe";
lastname = "Peterson";
```

Variables are *untyped* (they can hold values of any type)

Variables declared within a function are local to that function (accessible only within that function)

Variables declared outside a function are global (accessible from anywhere on the page)

JavaScript and Camel Case

Historically, programmers have used three ways of joining multiple words into one variable name:

Hyphens:

first-name, last-name, master-card, inter-city.

Underscore:

first_name, last_name, master_card, inter_city.

Camel Case:

FirstName, LastName, MasterCard, InterCity.

In programming languages, especially in JavaScript, camel case often starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

Difference between Javascript Literal and Variable:

JAVASCRIPT DATA TYPES

- JavaScript has three “primitive” types: number, string, and boolean

- Everything else is an object
- Numbers are always stored as floating-point values
  - Hexadecimal numbers begin with 0x
  - Some platforms treat 0123 as octal, others treat it as decimal
- Strings may be enclosed in single quotes or double quotes
  - Strings can contains \n (newline), \" (double quote), etc.
- Booleans are either true or false
  - 0, "0", empty strings, undefined, null, and NaN are false , other values are true

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void

default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

JAVASCRIPT OBJECTS

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	**Properties**	**Methods**

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

All cars have the same **properties**, but the property values differ from car to car.

All cars have the same **methods**, but the methods are performed at different times.

Objects are variables too. But objects can contain many values.

var car = {type:"Fiat", model:500, color:"white"};

The values are written as **name:value** pairs (name and value separated by a colon).

The name:values pairs (in JavaScript objects) are called **properties**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName[propertyName]

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

JAVASCRIPT OPERATORS

Arithmetic operators:

```
+   -   *   /   %   ++   --
```

Comparison operators:

```
<   <=   ==   !=   >=   >
```

Logical operators:

&& || !   (&& and || are *short-circuit* operators)

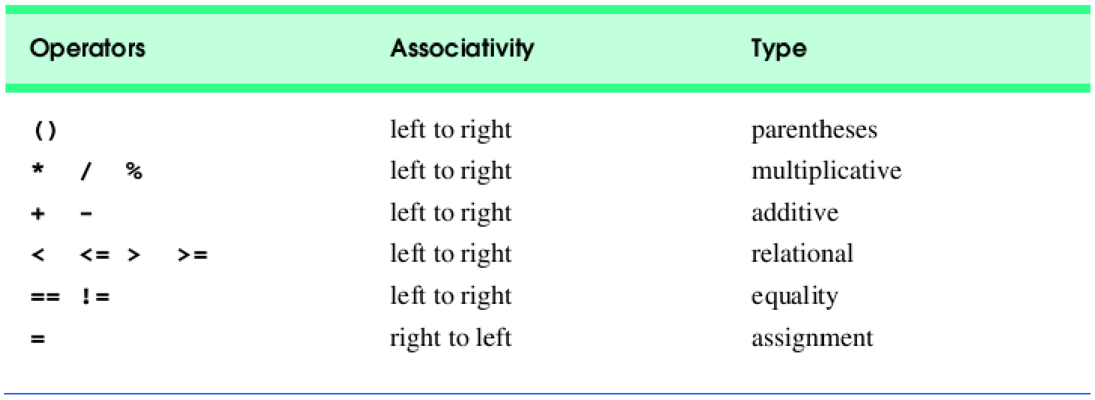
Bitwise operators:

```
&   |   ^   ~   <<   >>   >>>
```

Assignment operators:

```
+=  -=  *=  /=  %=  <<=  >>=  >>>=  &=  ^=  |=
```

Associativity.



Operators	Associativity	Type
`()`	left to right	parentheses
`*  /  %`	left to right	multiplicative
`+  -`	left to right	additive
`<  <=  >  >=`	left to right	relational
`==  !=`	left to right	equality
`=`	right to left	assignment

JAVASCRIPT EXPRESSIONS

An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value. The value may be a number, a string, or a logical value. Conceptually, there are two types of expressions: those that assign a value to a variable, and those that simply have a value. For example, the expression
x = 7
is an expression that assigns x the value 7. This expression itself evaluates to 7. Such expressions use *assignment operators*. On the other hand, the expression
3 + 4
simply evaluates to 7; it does not perform an assignment. The operators used in such expressions are referred to simply as *operators*.
JavaScript has the following kinds of expressions:

- Arithmetic: evaluates to a number
- String: evaluates to a character string, for example "Fred" or "234"
- Logical: evaluates to true or false

The special keyword **null** denotes a null value. In contrast, variables that have not been assigned a value are *undefined*, and cannot be used without a run-time error.

Conditional Expressions

A conditional expression can have one of two values based on a condition. The syntax is

(*condition*) ? *val1* : *val2*

If *condition* is true, the expression has the value of *val1*, Otherwise it has the value of *val2*. You can use a conditional expression anywhere you would use a standard expression.
For example,

status = (age >= 18) ? "adult" : "minor"
This statement assigns the value "adult" to the variable status if age is eighteen or greater. Otherwise, it assigns the value "minor" to status.

JAVASCRIPT STATEMENTS

Algorithm : Any computing problem can be solved by executing a series of actions in a specific order.

A procedure for solving a problem in terms of

1. the actions to be executed and
2. the order in which the actions are to be executed

is called an algorithm.

Pseudocode

Pseudocode is an artificial and informal language that helps programmers develop algo-rithms.

Pseudocode is similar to everyday English; it is convenient and user friendly, although it is not an actual computer program-ming language.

JavaScript statements are "instructions" to be "executed" by the web browser.

JavaScript Programs

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

Semicolons ;

Semicolons separate JavaScript statements.

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

JAVASCRIPT CONTROL STATEMENTS

- Sequential execution
  - Statements execute in the order they are written
- Transfer of control
  - Next statement to execute may not be the next one in sequence
- Three control structures
  - Sequence structure

- Selection structure
    - if
    - if…else
    - switch
- Repetition structure
    - while
    - do…while
    - for
    - for…in

if Selection Statement

- Single-entry/single-exit structure
- Indicate action only when the condition evaluates to true

if…else Selection Statement

- Indicate different actions to be perform when condition is true or false
- Conditional operator (?:)
    - JavaScript's only ternary operator
        - Three operands
        - Forms a conditional expression
- Dangling-else problem

switch Multiple-Selection Statement

- Controlling expression
- Case labels
- Default case

while Repetition Statement

- Repetition structure (loop)
    - Repeat action while some condition remains true

do…while Repetition Statement

- Similar to the while statement
- Tests the loop continuation condition after the loop body executes
- Loop body always executes at least once

break and continue Statements

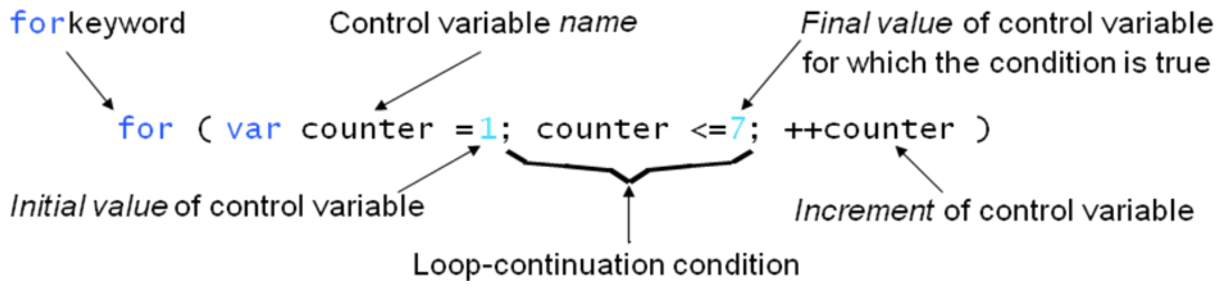
- break
  - Immediate exit from the structure
  - Used to escape early from a loop
  - Skip the remainder of a switch statement
- continue
  - Skips the remaining statements in the body of the structure
  - Proceeds with the next iteration of the loop

Labeled break and continue Statements

- Labeled break statement
  - Break out of a nested set of structures
  - Immediate exit from that structure and enclosing repetition structures
  - Execution resumes with first statement after enclosing labeled statement
- Labeled continue statement
  - Skips the remaining statements in structure's body and enclosing repetition structures
  - Proceeds with next iteration of enclosing labeled repetition structure
  - Loop-continuation test evaluates immediately after the continue statement executes

for Repetition Statement

- It is a repetition statement
  - Handles all the details of counter-controlled repetition
  - for structure header
    - The first line


for keyword
Control variable name
Final value of control variable for which the condition is true
for (var counter =1; counter <=7; ++counter)
Initial value of control variable
Increment of control variable
Loop-continuation condition

<u>Example</u>

<?xml version = "1.0"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Class Average Program -->

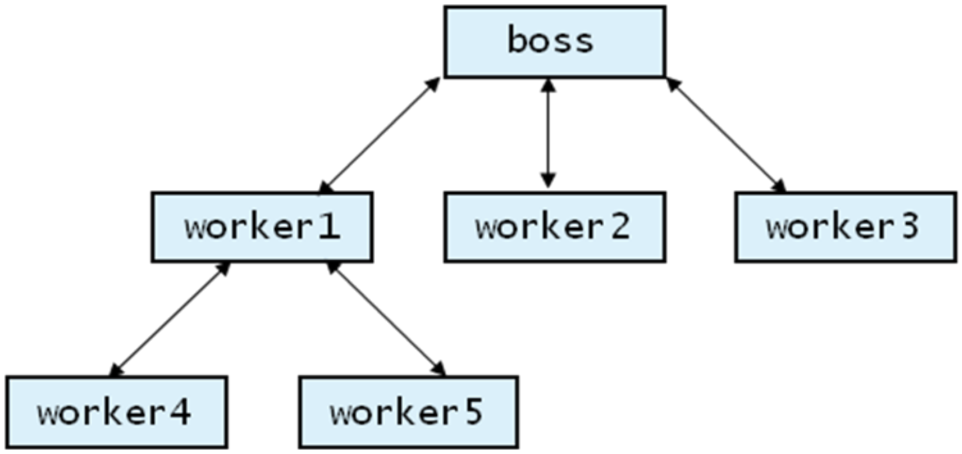
<html xmlns = "http://www.w3.org/1999/xhtml">

```
<head>
  <title>Class Average Program</title>
    <script type = "text/javascript">
    <!--
    var total,         // sum of grades
        gradeCounter,  // number of grades entered
        gradeValue,    // grade value
        average,       // average of all grades
        grade;         // grade typed by user
      // Initialization Phase
    total = 0;         // clear total
    gradeCounter = 1;  // prepare to loop
    // Processing Phase
    while ( gradeCounter <= 10 ) {  // loop 10 times
        // prompt for input and read grade from user
      grade = window.prompt( "Enter integer grade:", "0" );
        // convert grade from a string to an integer
      gradeValue = parseInt( grade );
        // add gradeValue to total
      total = total + gradeValue;
        // add 1 to gradeCounter
      gradeCounter = gradeCounter + 1;
    }
        // Termination Phase
    average = total / 10;  // calculate the average
      // display average of exam grades
    document.writeln(
      "<h1>Class average is " + average + "</h1>" );
    // -->
  </script>
```

JAVASCRIPT FUNCTIONS

- Functions

- Started by function call
- Receive necessary information via arguments (parameters)
- Boss-Worker relationship
  - Calling function
  - Called function
  - Return value when finished
  - Can have many tiers



Defining functions

- All variables declared in function are called local
  - Do not exist outside current function
- Parameters
  - Also local variables
- Promotes reusability
  - Keep short
  - Name clearly

Format of a function definition

function *function-name*(*parameter-list*)

{

  declarations and statements

}

- Function name any valid identifier
- Parameter list names of variables that will receive arguments
  - Must have same number as function call

    - May be empty
  - Declarations and statements
    - Function body ("block" of code)

Returning control

  - return statement
  - Can return either nothing, or a value

return *expression*;

  - No return statement same as return;
  - Not returning a value when expected is an error

JavaScript Global Functions

Global function	Description
escape	This function takes a string argument and returns a string in which all spaces, punctuation, accent characters and any other character that is not in the ASCII character set are encoded in a hexadecimal format that can be represented on all platforms.
eval	This function takes a string argument representing JavaScript code to execute. The JavaScript interpreter evaluates the code and executes it when the eval function is called. This function allows JavaScript code to be stored as strings and executed dynamically.
isFinite	This function takes a numeric argument and returns true if the value of the argument is not NaN, Number.POSITIVE_INFINITY or Number.NEGATIVE_INFINITY; otherwise, the function returns false.
isNaN	This function takes a numeric argument and returns true if the value of the argument is not a number; otherwise, it returns false. The function is commonly used with the return value of parseInt or parseFloat to determine whether the result is a proper numeric value.
parseFloat	This function takes a string argument and attempts to convert the beginning of the string into a floating-point value. If the conversion is unsuccessful, the function returns NaN; otherwise, it

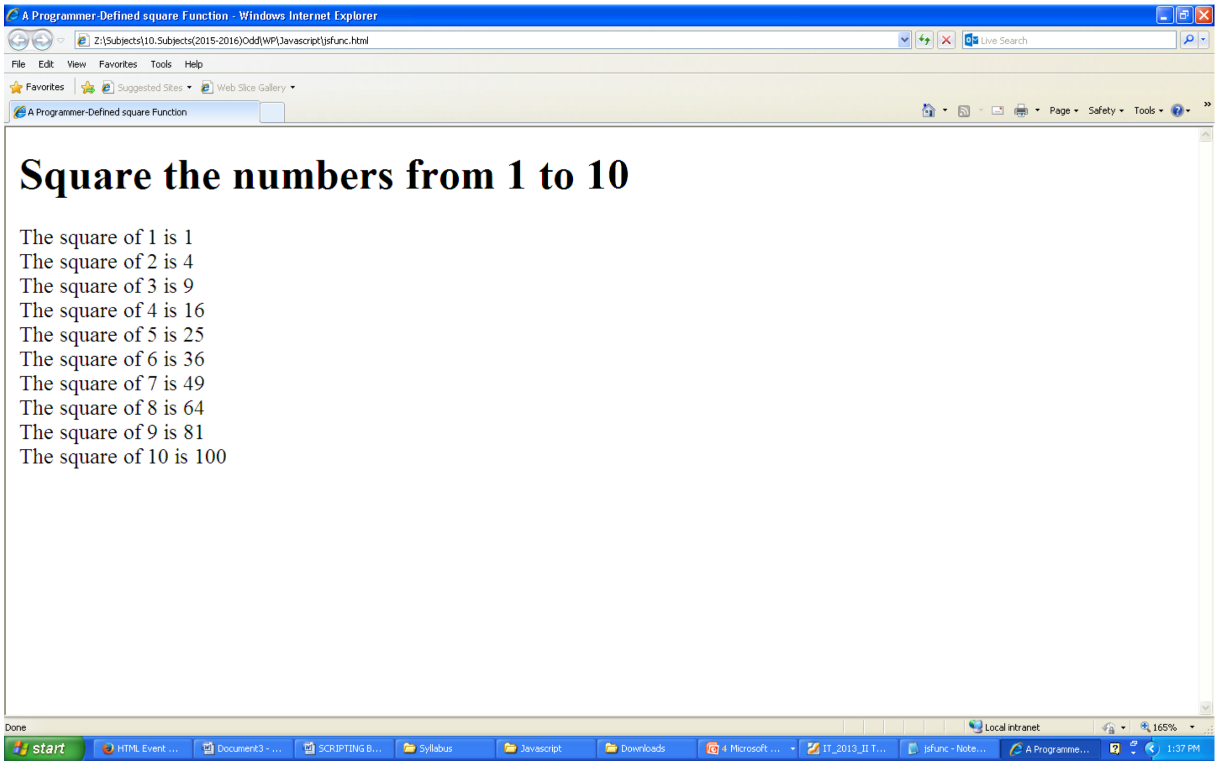
	returns the converted value (e.g., parseFloat("abc123.45") returns NaN, and parseFloat("123.45abc") returns the value 123.45).
parseInt	This function takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns NaN; otherwise, it returns the converted value (e.g., parseInt("abc123") returns NaN, and parseInt("123abc") returns the integer value 123). This function takes an optional second argument, from 2 to 36, specifying the **radix** (or **base**) of the number. Base 2 indicates that the first argument string is in **binary** format, base 8 indicates that the first argument string is in **octal** format and base 16 indicates that the first argument string is in **hexadecimal** format.
unescape	This function takes a string as its argument and returns a string in which all characters previously encoded with escape are decoded.

Example

Square of the numbers from 1 to 10 (jsfunc.html)

```
 <html xmlns = "http://www.w3.org/1999/xhtml">
    <head>
      <title>A Programmer-Defined square Function</title>
        <script type = "text/javascript">
        <!--
        document.writeln(
          "<h1>Square the numbers from 1 to 10</h1>" );
               // square the numbers from 1 to 10
        for ( var x = 1; x <= 10; ++x )
          document.writeln( "The square of " + x + " is " +
            square( x ) + "<br />" );
        // The following square function's body is executed
        // only when the function is explicitly called.
```

```
            // square function definition
        function square( y )
        {
          return y * y;
        }
        // -->
      </script>
      </head><body></body>
  </html>
```


A Programmer-Defined square Function - Windows Internet Explorer
Z:\Subjects\10.Subjects(2015-2016)Odd\WP\Javascript\jsfunc.html
File Edit View Favorites Tools Help

Square the numbers from 1 to 10

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```

<u>JAVASCRIPT EVENTS</u>

- Events are the actions that occur as a result of browser activities or user interactions with the web pages.
  - Such as the user performs an action (mouse click or enters data)
  - We can validate the data entered by a user in a web form
  - Communicate with Java applets and browser plug-ins

Event Categories

- Keyboard and mouse events
  - Capturing a mouse event is simple

- Load events
  - The page first appears on the screen: “Loads”, leaves: “Unloads”, …
- Form-related events
  - onFocus() refers to placing the cursor into the text input in the form.
- Others
  - Errors, window resizing.

HTML elements	**HTML tags**	**JavaScript defined events**	**Description**
Link	<a>	click dblClick mouseDown mouseUp mouseOver	Mouse is clicked on a link Mouse is double-clicked on a link Mouse button is pressed Mouse button is released Mouse is moved over a link
Image	<img>	load abort error	Image is loaded into a browser Image loading is abandoned An error occurs during the image loading
Area	<area>	mouseOver mouseOut dblClick	The mouse is moved over an image map area The mouse is moved from image map to outside The mouse is double-clicked on an image map
Form	<form>	submit	The user submits a form

		Reset	The user refreshes a form
…	…	…	…

Event Handler

- When an event occurs, a code segment is executed in response to a specific event is called "event handler".
- Event handler names are quite similar to the name of events they handle.
- E.g the event handler for the "click" event is "onClick".
- <HTMLtag eventhandler="JavaScript Code">

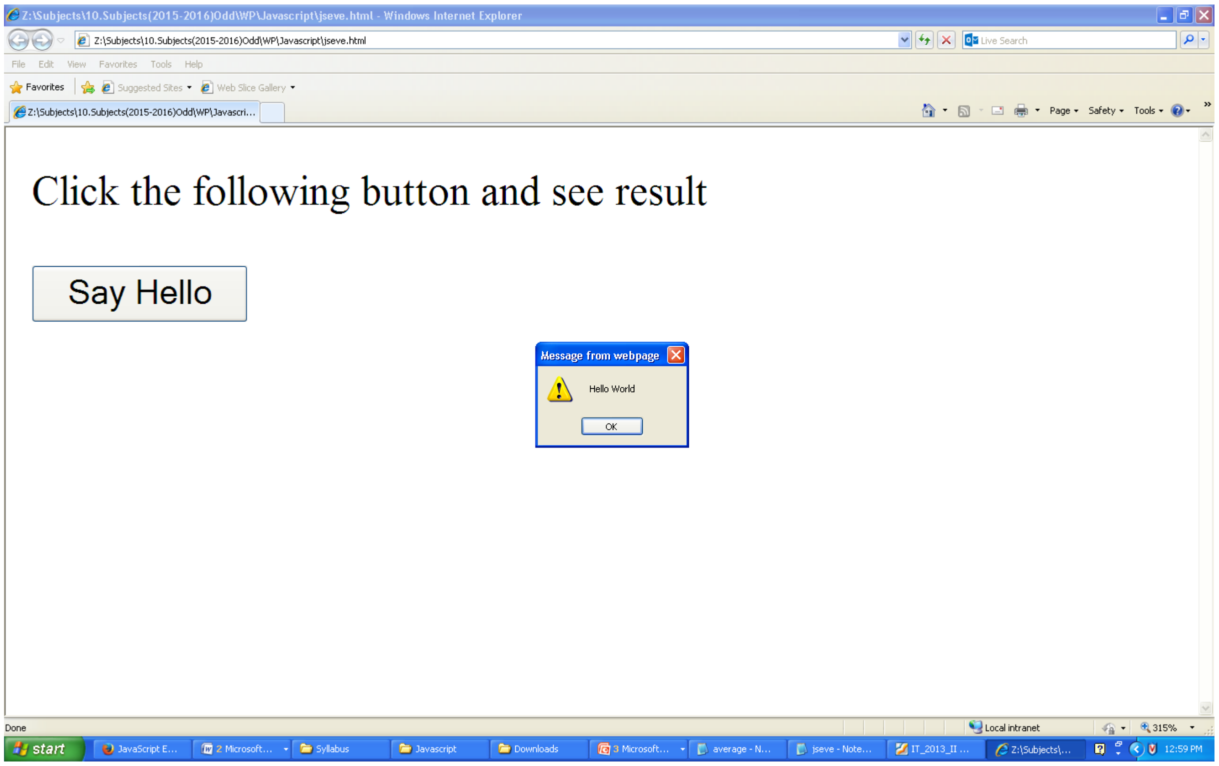
Event Handlers	**Triggered when**
onChange	The value of the text field, textarea, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDblClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

Example

OnClick- Event Handler (jseveonclick.html)

```
<html>
```

```
    <head>
      <script type="text/javascript">
        <!--
          function sayHello() {
            alert("Hello World")
          }
        //-->
      </script>
    </head>
    <body>
      <p>Click the following button and see result</p>
      <form>
        <input type="button" onclick="sayHello()" value="Say Hello" />
      </form>
    </body>
</html>
```



Z:\Subjects\10.Subjects(2015-2016)Odd\WP\Javascript\jseve.html - Windows Internet Explorer

Click the following button and see result

Message from webpage

Hello World

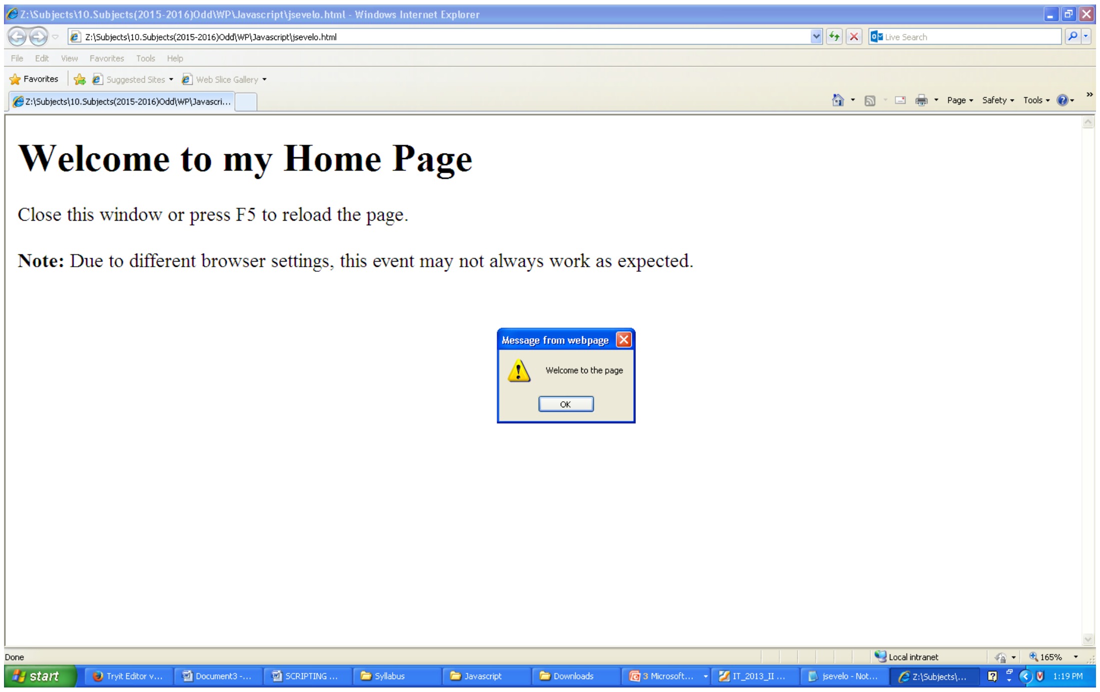
onLoad event Handler (jsevelo.html)

```
<!DOCTYPE html>
```

```
<html>
<body onload="initial()" onunload="myFunction()">

<h1>Welcome to my Home Page</h1>
<p>Close this window or press F5 to reload the page.</p>
<p><strong>Note:</strong> Due to different browser settings, this event may not always work as expected.</p>
<script>
function initial()
{
alert("Welcome to the page");
}
function myFunction() {
   alert("Thank you for visiting!");
}
</script>
</body>
</html>
```

Onload



Z:\Subjects\10.Subjects(2015-2016)Odd\WP\Javascript\jsevelo.html - Windows Internet Explorer

Welcome to my Home Page

Close this window or press F5 to reload the page.

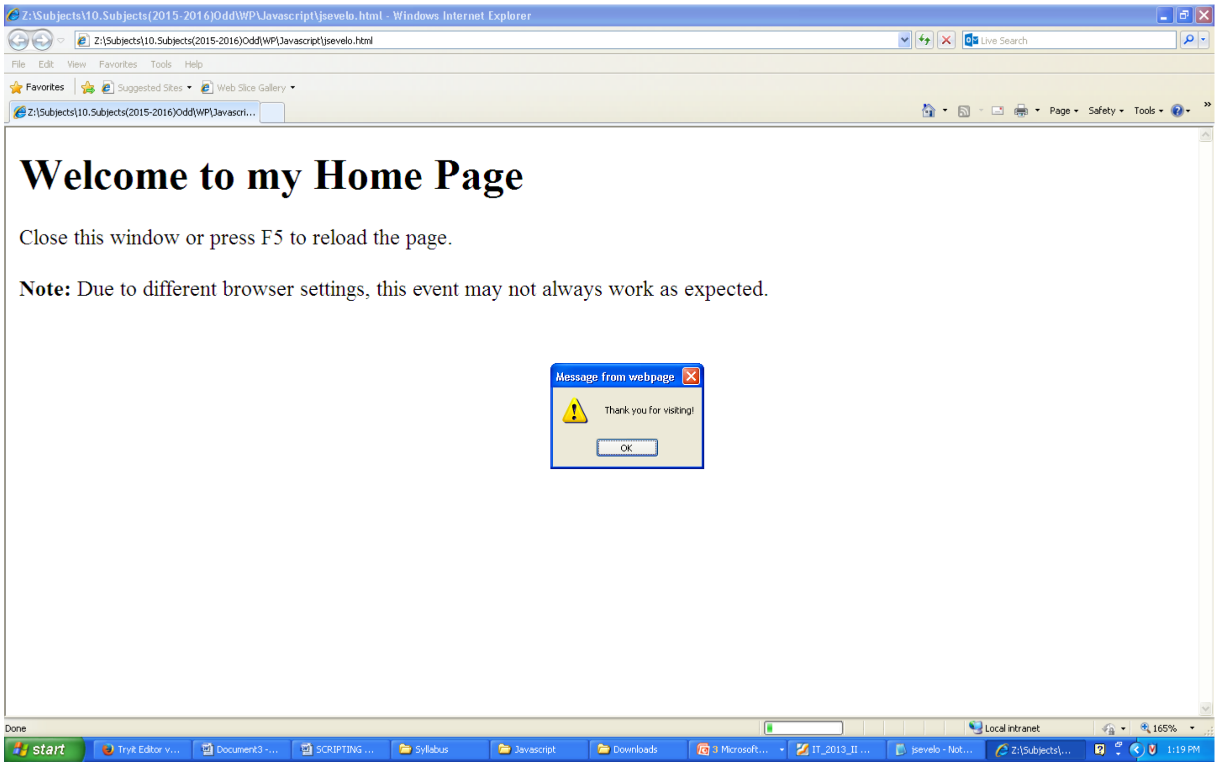
Note: Due to different browser settings, this event may not always work as expected.

Message from webpage

Welcome to the page

OK

Unload



Welcome to my Home Page

Close this window or press F5 to reload the page.

Note: Due to different browser settings, this event may not always work as expected.

Message from webpage

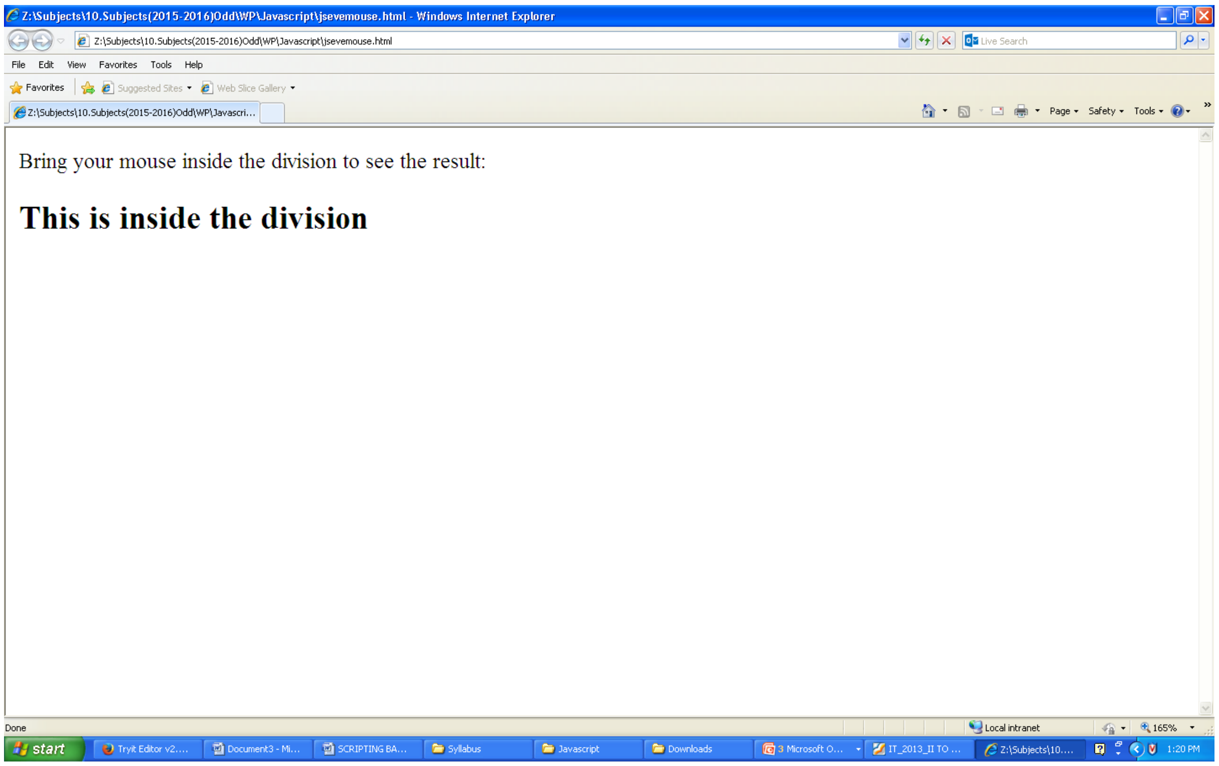
Thank you for visiting!

OK

Onmouseout and onmouseover (jsevemouse.html)

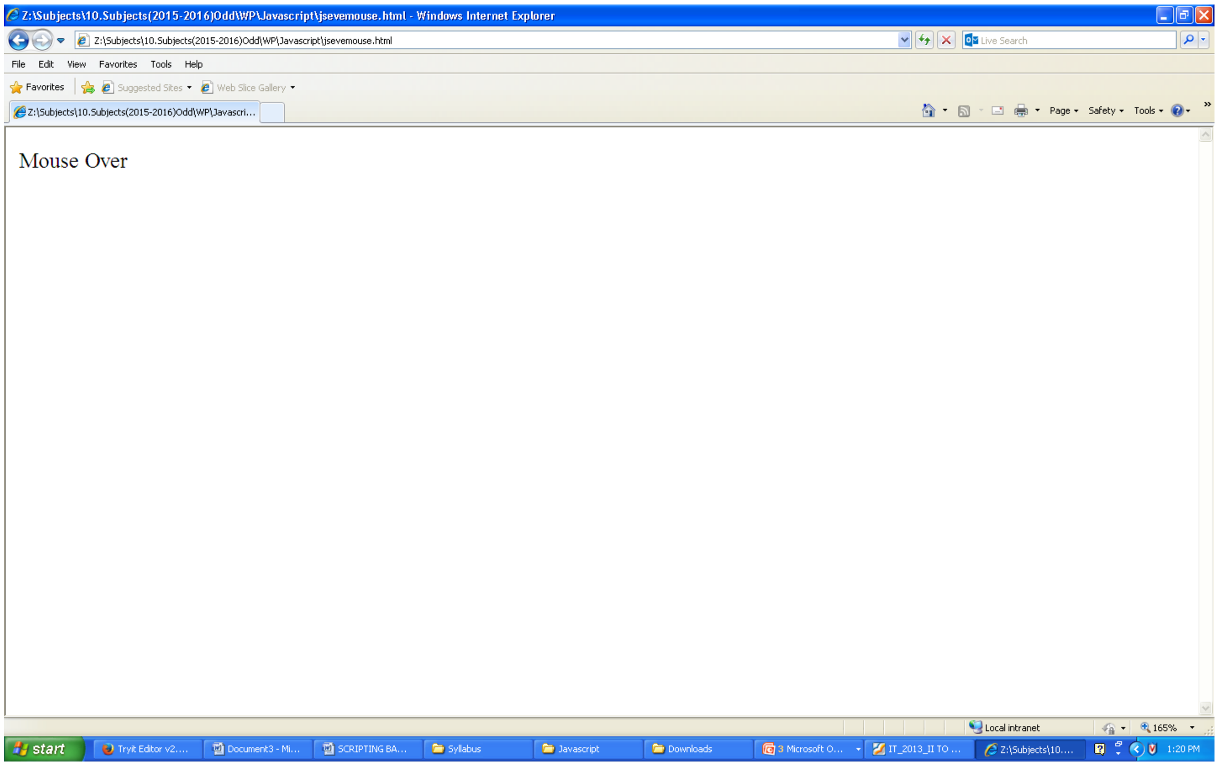
```
<html>
  <head>
    <script type="text/javascript">
      <!--
        function over() {
          document.write ("Mouse Over");
        }
        function out() {
          document.write ("Mouse Out");
        }
      //-->
    </script>
  </head>
  <body>
    <p>Bring your mouse inside the division to see the result:</p>
```

```
        <div onmouseover="over()" onmouseout="out()">
          <h2> This is inside the division </h2>
        </div>
    </body>
</html>
```



Bring your mouse inside the division to see the result:

This is inside the division


Z:\Subjects\10.Subjects(2015-2016)Odd\WP\Javascript\jsevemouse.html - Windows Internet Explorer
File Edit View Favorites Tools Help

Mouse Over

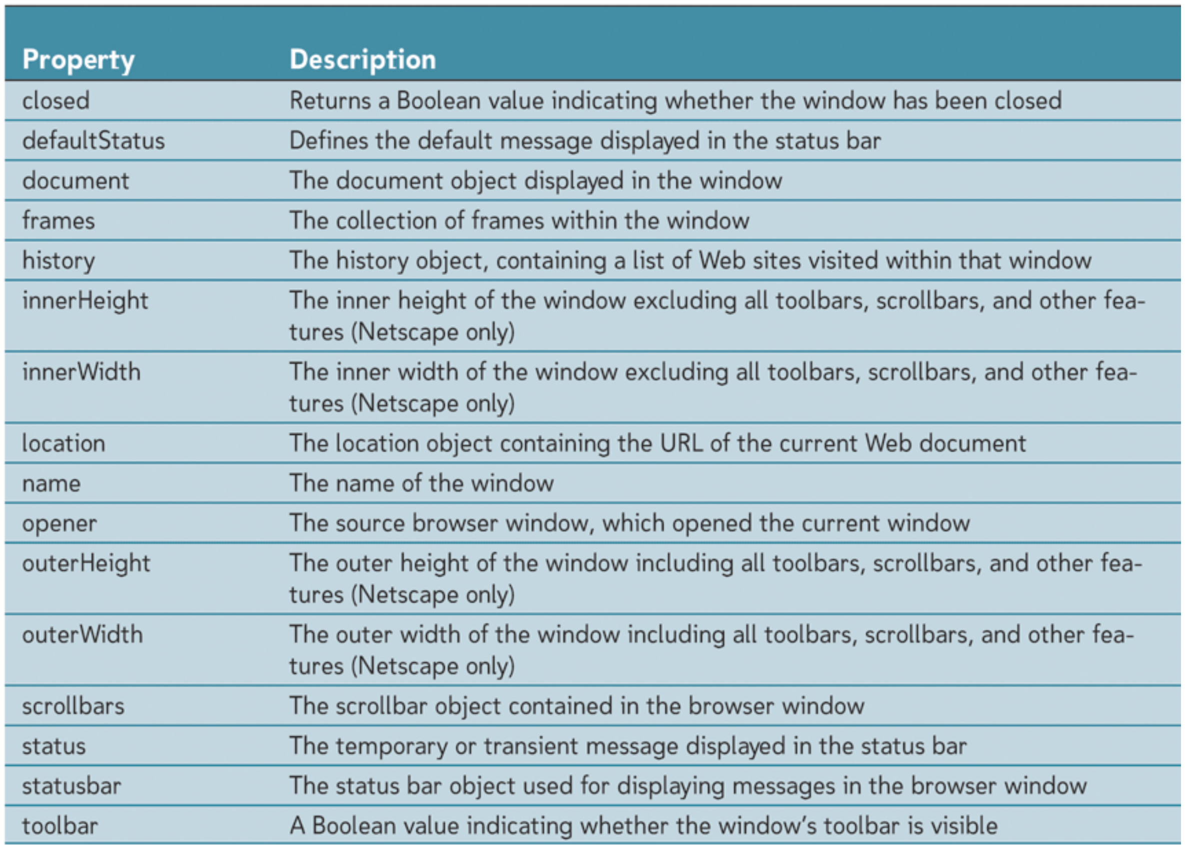
Event bubbling

- The process whereby events fired on *child* elements "bubble" up to their *parent* elements
- When an event is fired on an element, it is first delivered to the element's event handler (if any), then to the parent element's event handler (if any)

If you intend to handle an event in a child element alone, you should cancel the bubbling of the event in the child element's event-handling code by using the cancelBubble property of the event object

<u>JAVASCRIPT WINDOWS</u>

- JavaScript considers the browser window an object, which it calls the **window object**



Property	Description
closed	Returns a Boolean value indicating whether the window has been closed
defaultStatus	Defines the default message displayed in the status bar
document	The document object displayed in the window
frames	The collection of frames within the window
history	The history object, containing a list of Web sites visited within that window
innerHeight	The inner height of the window excluding all toolbars, scrollbars, and other features (Netscape only)
innerWidth	The inner width of the window excluding all toolbars, scrollbars, and other features (Netscape only)
location	The location object containing the URL of the current Web document
name	The name of the window
opener	The source browser window, which opened the current window
outerHeight	The outer height of the window including all toolbars, scrollbars, and other features (Netscape only)
outerWidth	The outer width of the window including all toolbars, scrollbars, and other features (Netscape only)
scrollbars	The scrollbar object contained in the browser window
status	The temporary or transient message displayed in the status bar
statusbar	The status bar object used for displaying messages in the browser window
toolbar	A Boolean value indicating whether the window's toolbar is visible

Creating New Browser Windows

- Opening New Windows with JavaScript
  - The JavaScript command to create a new browser window is

WinObj=window.open("url","name","features")

- The following are the components of the window.open() statement:
  - The WinObj variable is used to store the new windowobject. You can access methods and properties of the new object by using this name.
  - The first parameter of the window.open() method is a URL, which will be loaded into the new window. If it's left blank, no web page will be loaded. In this case, you could use JavaScript to fill the window with content.
  - .The second parameter specifies a window name (here, WindowName). This is assigned to the window object's name property and is used to refer to the window.
  - The third parameter is a list of optional features, separated by commas. You can customize the new window by choosing whether to include the toolbar, status line, and other features. This enables you to create a variety of "floating" windows, which might look nothing like a typical browser window

- Setting the Features of a Pop-up Window
  - The feature list obeys the following syntax:

“feature1=value1, feature2=value2…featureN=valueN”

- The window.close() method closes a window.
- Browsers don’t normally allow you to close the main browser window without the user’s permission; this method’s main purpose is for closing windows you have created.
- For example, this statement closes a window called updatewindow: updatewindow.close();

Moving and Resizing Window

- window.moveTo() moves the window to a new position. The parameters specify the x (column) and y (row) position.
- window.moveBy() moves the window relative to its current position. The x and y parameters can be positive or negative, and are added to the current values to reach the new position.
- window.resizeTo() resizes the window to the width and height specified as parameters.
- window.resizeBy() resizes the window relative to its current size. The parameters are used to modify the current width and height

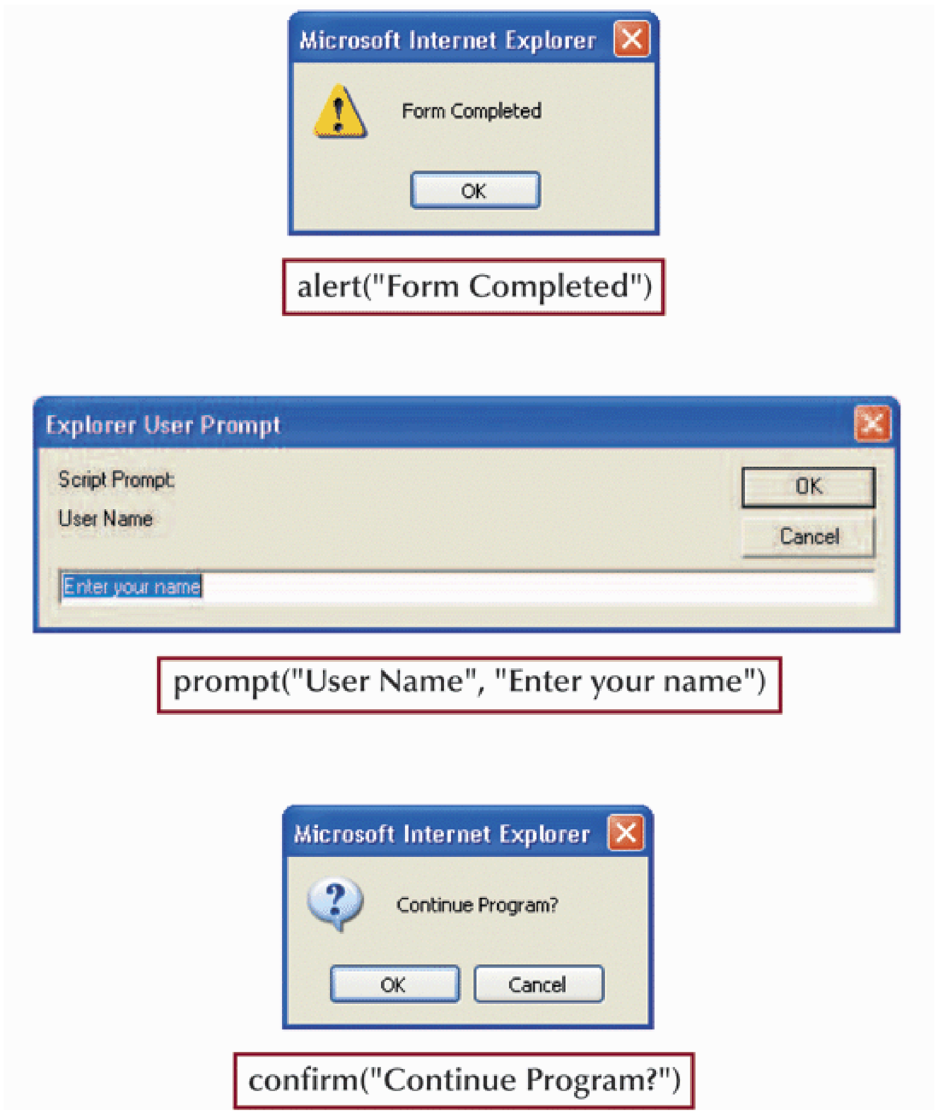
Using Timeouts

- The window.setTimeout method enables you to specify a time delay and a command that will execute after the delay passes.
- setTimeout() method, which has two parameters.
- The first is a JavaScript statement, or group of statements, enclosed in quotes.
- The second parameter is the time to wait in milliseconds (thousandths of seconds).
- For example, the following statement displays an alert dialog box after 10 seconds:
  - ident=window.setTimeout(“alert(‘Time’s up!’)”,10000)
- Before a timeout has elapsed, you can stop it with the clearTimeout() method, specifying the identifier of the timeout to stop:
  - window.clearTimeout(ident)

Displaying Dialog Box

- window.alert(message) displays an alert dialog box. This dialog box simply gives the user a message.
- window.confirm(message) displays a confirmation dialog box. This displays a message and includes OK and Cancel buttons. This method returns true if OK is pressed and false if Cancel is pressed.

- window.prompt(message,default) displays a message and prompts the user for input. It returns the text entered by the user. If the user does not enter anything, the default value is used.



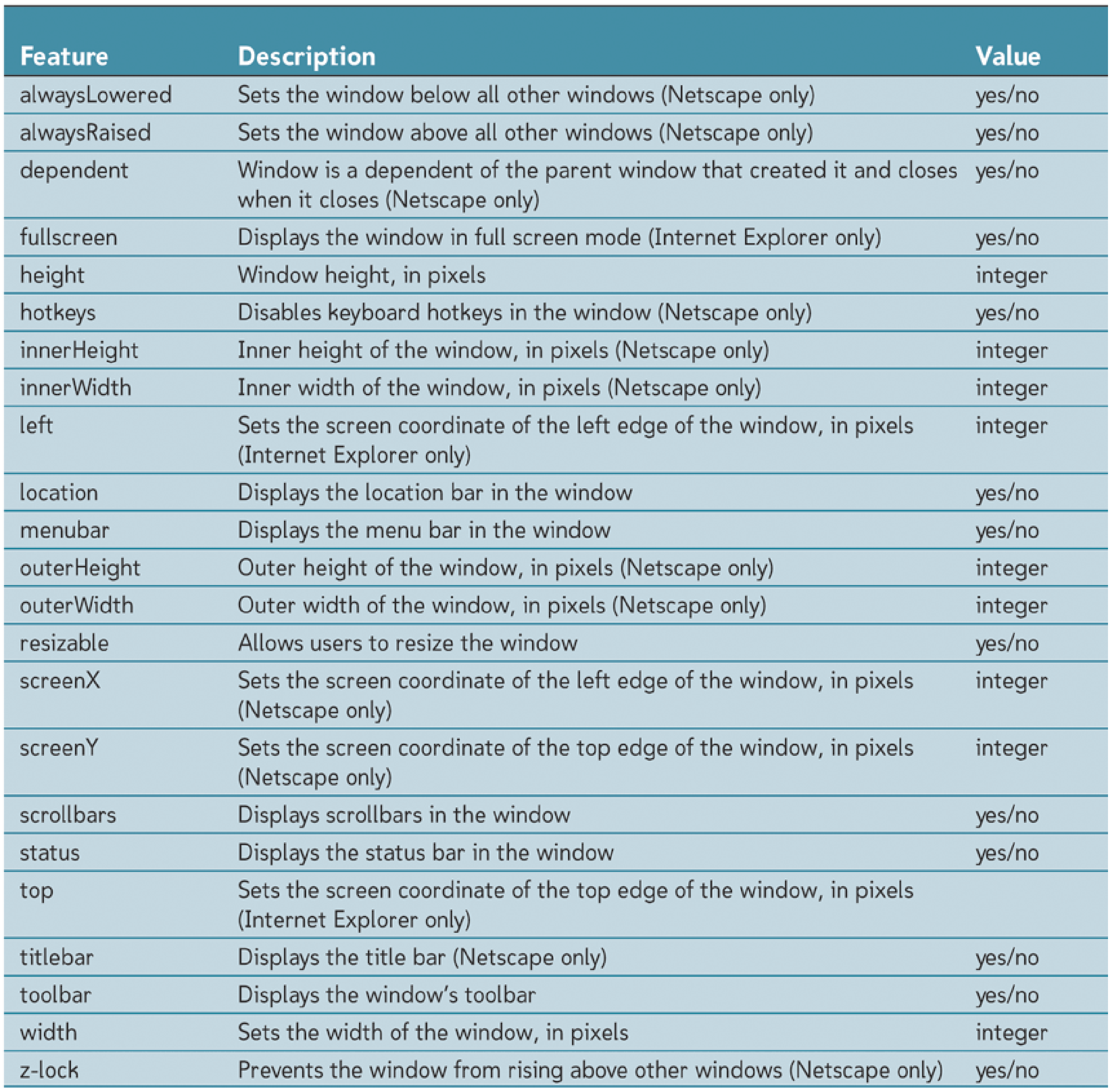
```
alert("Form Completed")
```

```
prompt("User Name", "Enter your name")
```

```
confirm("Continue Program?")
```

Creating New Browser Windows

- Setting the Features of a Pop-up Window



Feature	Description	Value
alwaysLowered	Sets the window below all other windows (Netscape only)	yes/no
alwaysRaised	Sets the window above all other windows (Netscape only)	yes/no
dependent	Window is a dependent of the parent window that created it and closes when it closes (Netscape only)	yes/no
fullscreen	Displays the window in full screen mode (Internet Explorer only)	yes/no
height	Window height, in pixels	integer
hotkeys	Disables keyboard hotkeys in the window (Netscape only)	yes/no
innerHeight	Inner height of the window, in pixels (Netscape only)	integer
innerWidth	Inner width of the window, in pixels (Netscape only)	integer
left	Sets the screen coordinate of the left edge of the window, in pixels (Internet Explorer only)	integer
location	Displays the location bar in the window	yes/no
menubar	Displays the menu bar in the window	yes/no
outerHeight	Outer height of the window, in pixels (Netscape only)	integer
outerWidth	Outer width of the window, in pixels (Netscape only)	integer
resizable	Allows users to resize the window	yes/no
screenX	Sets the screen coordinate of the left edge of the window, in pixels (Netscape only)	integer
screenY	Sets the screen coordinate of the top edge of the window, in pixels (Netscape only)	integer
scrollbars	Displays scrollbars in the window	yes/no
status	Displays the status bar in the window	yes/no
top	Sets the screen coordinate of the top edge of the window, in pixels (Internet Explorer only)	
titlebar	Displays the title bar (Netscape only)	yes/no
toolbar	Displays the window's toolbar	yes/no
width	Sets the width of the window, in pixels	integer
z-lock	Prevents the window from rising above other windows (Netscape only)	yes/no

- Working with Pop-up Blockers
  - Pop-up blockers prevent pop-up windows from opening
  - You can have the browser check whether the pop-up window has been opened or not

```
function popWin(url) {
   windowObj = window.open("url","name","features");
   test=(windowObj==null ||
     typeof(windowObj)=="undefined) ? true : false;
   return test;
}
<a href="url" onclick="return(popWin("url"))">Link Text</a>
```

Model and Modeless Window

- A **modal window** is a window that prevents users from doing work in any other window or dialog box until the window is closed
- A **modeless window** allows users to work in other dialog boxes and windows even if the window stays open

JAVASCRIPT DOCUMENTS

The HTML DOM Document

In the HTML DOM object model, the document object represents your web page.

The document object is the owner of all other objects in your web page.

If you want to access objects in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Difference between document.write and document.writeln method

- write() does NOT add a new line after each statement
- writeln() add a new line after each statement

Finding HTML Elements

Method	**Description**
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Changing HTML Elements

Method	**Description**
element.innerHTML=	Change the inner HTML of an element
element.attribute=	Change the attribute of an HTML element

element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property=*	Change the style of an HTML element

Adding and Deleting Elements

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

Adding Events Handlers

Method	**Description**
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

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

JAVASCRIPT FRAMES

- The name attribute of a frame is used when creating links whose targets are designed to appear in specific frames
- To reference a specific frame in you JavaScript code, you need to use the id attribute

  <frame id="top" name="top" src="home.htm" />
- Working with the frame and frameset objects
  - Each frame in a frameset is part of the frames collection

    windowObject.frames[idref]
  - To reference the header frame

    window.frames[0]

    window.framses["header"]
- Navigating between frames
  - JavaScript treats the frames in a frameset as elements in a hierarchical tree
  - The **parent keyword** refers to any object that is placed immediately above another object in the hierarchy
  - If you want to go directly to the top of the hierarchy, you can use the **top keyword**
- Treating frames as windows
  - In most cases JavaScript treats a frame as a separate browser window

    frameObject.document.write(content)

    frameObject.document.close()

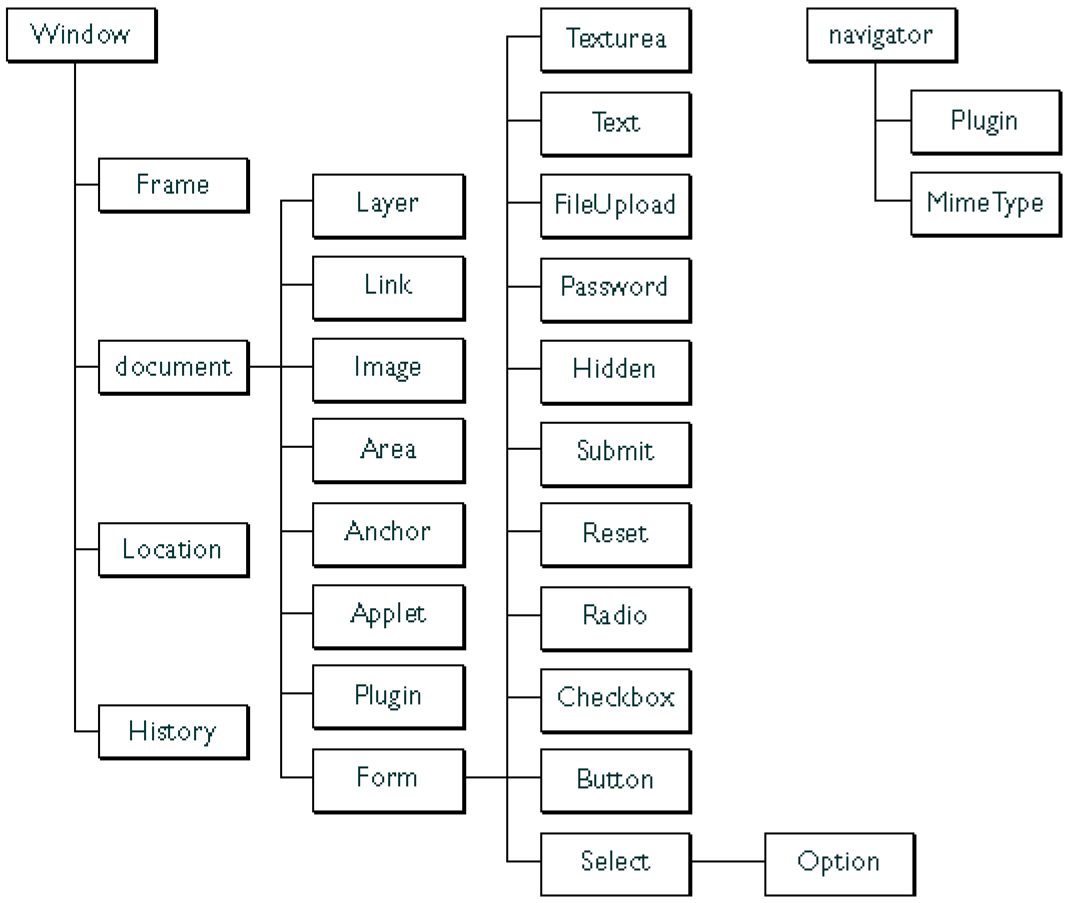
    frameObject.location.href = "url"
- Setting the frameset layout
  - In JavaScript

frameset.rows = “text”

frameset.cols = “text”

Browser Object Model

- The Browser Object Model (BOM) is a larger representation of everything provided by the browser including the current document, location, history, frames, and any other functionality the browser may expose to JavaScript.
- The Browser Object Model is not standardized and can change based on different browsers.
- The BOM's most important task is managing browser windows and enabling communication between the windows. Therefore, the window object stands at the center of the BOM.
- Every HTML document loaded into a browser window becomes a **Document** object which is an object in the **Browser Object Model** (BOM). All nodes with their contents in the BOM can be modified or deleted, and new node can be created by Javascript.



- The Document Object Model (DOM) is standardized and is specific to current HTML document. It is subset of BOM.

- BOM is the catch-all category of JavaScript. There are many objects, methods, and properties that are not part of the DOM structure. For example, the URL of the current page or the identification string of the browser are defined on the window object.

VERIFYING FORMS

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button.
If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information.
This was really a lengthy process which used to put a lot of burden on the server.
JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example (vali_form.html)

```
<html>
    <head>
    <title>Form Validation</title>
        <script type="text/javascript">
      <!--
        // Form validation code will come here.
        function validate()
        {
      var emailID = document.myForm.EMail.value;
      atpos = emailID.indexOf("@");
      dotpos = emailID.lastIndexOf(".");
     if( document.myForm.Name.value == "" )
      {
        alert( "Please provide your name!" );
```

```
            document.myForm.Name.focus() ;
            return false;
         }
         if( document.myForm.EMail.value == "" )
         {
            alert( "Please provide your Email!" );
            document.myForm.EMail.focus() ;
            return false;
         }
    if (atpos < 1 || ( dotpos - atpos < 2 ))
         {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
         }
         if( document.myForm.Zip.value == "" ||
         isNaN( document.myForm.Zip.value ) ||
         document.myForm.Zip.value.length != 5 )
         {
            alert( "Please provide a zip in the format #####." );
            document.myForm.Zip.focus() ;
            return false;
         }
         if( document.myForm.Country.value == "-1" )
         {
            alert( "Please provide your country!" );
            return false;
         }
         return( true );
       }
         //-->
       </script>
```

```
</head>
<body>
  <form action="/cgi-bin/test.cgi" name="myForm" onsubmit="return(validate());">
    <table cellspacing="2" cellpadding="2" border="1">
      <tr>
        <td align="right">Name</td>
        <td><input type="text" name="Name" /></td>
      </tr>
      <tr>
        <td align="right">EMail</td>
        <td><input type="text" name="EMail" /></td>
      </tr>
      <tr>
        <td align="right">Zip Code</td>
        <td><input type="text" name="Zip" /></td>
      </tr>
      <tr>
        <td align="right">Country</td>
        <td>
          <select name="Country">
            <option value="-1" selected>[choose yours]</option>
            <option value="1">USA</option>
            <option value="2">UK</option>
            <option value="3">INDIA</option>
          </select>
        </td>
      </tr>
      <tr>
        <td align="right"></td>
        <td><input type="submit" value="Submit" /></td>
      </tr>
    </table>
```

```
      </form>
   </body>
</html>
```

<u>HTML5</u>

HTML5 Introduction

- Successor of HTML 4.01 and XHTML 1.1
- It comes with new tags, features and APIs
- Below is a non exhaustive list of features that tend to be labelled as "HTML5" in the medias:
  - New structural elements (<header>, <footer>, <nav> and more)
  - Forms 2.0 and client-side validation
  - Native browser support for audio and video (<video>, <audio>)
  - Canvas API and SVG
  - Web storage
  - Offline applications
  - Geolocation
  - Drag & Drop
  - Web Workers
  - New communications API (Server Sent Events, Web Sockets, …)

History

- December 1997: HTML 4.0 is published by the W3C
- February - March 1998: XML 1.0 is published
- December 1999 - January 2000: ECMAScript 3rd Edition, XHTML 1.0 (Basically HTML tags
  reformulated in XML) and, HTML 4.01 recommendations are published
- May 2001: XHTML 1.1 recommendation is published
- August 2002: XHTML 2.0 first working draft is released.
- December 2002: XHTML 2.0 second working draft published.
- January 2008: First W3C working draft of HTML5 is published!!

Future of HTML5

- 84% of Developers Plan to Adopt Key HTML5 Features

- The key to understanding HTML5 is that it is not one, but a group of technologies. Within HTML5, developers have a tremendous amount of choice regarding what they use and what they don't use
- The power of HTML5 being ready for prime-time can be seen in Microsoft's choice to utilize it in Windows 8

New and Updated HTML5 Elements

HTML5 introduces 28 new elements:

<section>, <article>, <aside>, <hgroup>, <header>,<footer>, <nav>, <figure>, <figcaption>, <video>, <audio>, <source>, <embed>, <mark>,<progress>, <meter>, <time>, <ruby>, <rt>, <rp >,<wbr>, <canvas>, <command>, <details>,<summary>, <datalist>, <keygen> and <output>

An HTML page first starts with the DOCTYPE declaration

HTML5 also update some of the previous existing elements to better reflect how they are used on the Web or to make them more useful such as:

- The <a> element can now also contain flow content instead of just phrasing content
- The <hr> element is now representing a paragraph-level thematic break
- The <cite> element only represent the title of a work
- The <strong> element is now representing importance rather than strong emphasis

HTML5 offers new elements for better document structure:

Tag	**Description**
<article>	Defines an article in the document
<aside>	Defines content aside from the page content
<bdi>	Defines a part of text that might be formatted in a different direction from other text
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or window
<figcaption>	Defines a caption for a <figure> element
<figure>	Defines self-contained content, like illustrations, diagrams, photos, code listings, etc.

<footer>	Defines a footer for the document or a section
<header>	Defines a header for the document or a section
<main>	Defines the main content of a document
<mark>	Defines marked or highlighted text
<menuitem>	Defines a command/menu item that the user can invoke from a popup menu
<meter>	Defines a scalar measurement within a known range (a gauge)
<nav>	Defines navigation links in the document
<progress>	Defines the progress of a task
<rp>	Defines what to show in browsers that do not support ruby annotations
<rt>	Defines an explanation/pronunciation of characters (for East Asian typography)
<ruby>	Defines a ruby annotation (for East Asian typography)
<section>	Defines a section in the document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time
<wbr>	Defines a possible line-break

New Form Elements

Tag	**Description**
<datalist>	Defines pre-defined options for input controls

<keygen>	Defines a key-pair generator field (for forms)
<output>	Defines the result of a calculation

New Input Types

New Input Types	**New Input Attributes**
• color • date • datetime • datetime-local • email • month • number • range • search • tel • time • url • week	• autocomplete • autofocus • form • formaction • formenctype • formmethod • formnovalidate • formtarget • height and width • list • min and max • multiple • pattern (regexp) • placeholder • required • step

HTML5 - New Attribute Syntax

HTML5 allows four different syntaxes for attributes.

Type	**Example**
Empty	<input type="text" value="John" **disabled**>

Unquoted	<input type="text" **value=John**>
Double-quoted	<input type="text" **value="John Doe"**>
Single-quoted	<input type="text" **value='John Doe'**>

HTML5 Graphics

Tag	**Description**
<canvas>	Defines graphic drawing using JavaScript
<svg>	Defines graphic drawing using SVG

New Media Elements

Tag	**Description**
<audio>	Defines sound or music content
<embed>	Defines containers for external applications (like plug-ins)
<source>	Defines sources for <video> and <audio>
<track>	Defines tracks for <video> and <audio>
<video>	Defines video or movie content

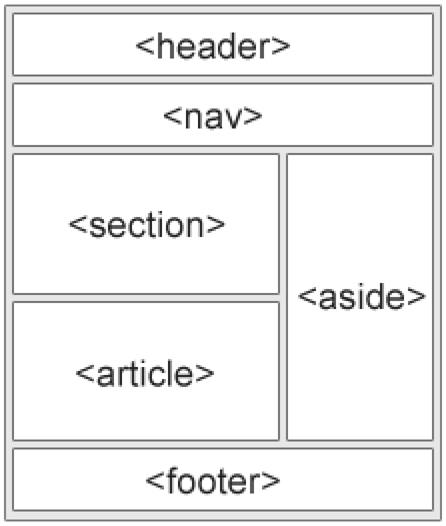
Semantic Elements

Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer"> to indicate navigation, header, and footer.

HTML5 offers new semantic elements to define different parts of a web page:

- <article>
- <aside>

- <details>
- <figcaption>
- <figure>
- <footer>
- <header>
- <main>
- <mark>
- <nav>
- <section>
- <summary>
- <time>



HTML5 <section> Element

The <section> element defines a section in a document.

HTML5 <article> Element

The <article> element specifies independent, self-contained content.
An article should make sense on its own, and it should be possible to read it independently from the rest of the web site.
Examples of where an <article> element can be used:

- Forum post
- Blog post
- Newspaper article

HTML5 <header> Element

The <header> element specifies a header for a document or section.
The <header> element should be used as a container for introductory content.

HTML5 <footer> Element

The <footer> element specifies a footer for a document or section.
A <footer> element should contain information about its containing element.

HTML5 <nav> Element

The <nav> element defines a set of navigation links.
The <nav> element is intended for large blocks of navigation links. However, not all links in a document should be inside a <nav> element!

HTML5 <aside> Element

The <aside> element defines some content aside from the content it is placed in (like a sidebar).

The aside content should be related to the surrounding content.

HTML5 <figure> and <figcaption> Elements

In books and newspapers, it is common to have captions with images.
The purpose of a caption is to add a visual explanation to an image.

Why Semantic HTML5 Elements?

With HTML4, developers used their own favorite attribute names to style page elements:
header, top, bottom, footer, menu, navigation, main, container, content, article, sidebar, topnav, ...
This made it impossible for search engines to identify the correct web page content.

With HTML5 elements like: <header> <footer> <nav> <section> <article>, this will become easier.

HTML5 <progress>

<Progress>: The new "progress" element appears to be very similar to the "meter" element. It is created to indicate progress of a specific task.

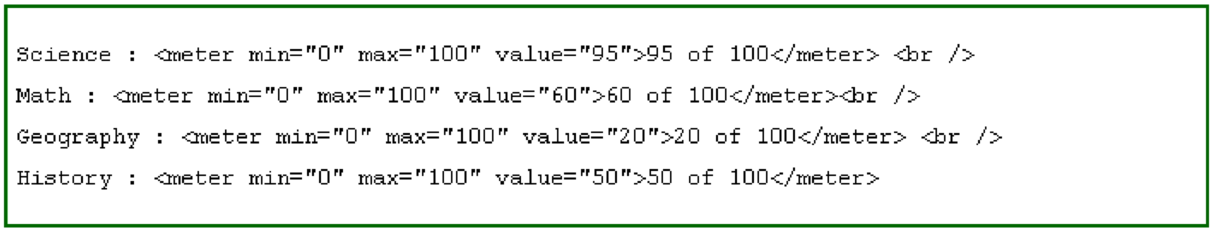
The progress can be either determinate OR interdeterminate. Which means, you can use "progress" element to indicate a progress that you do not even know how much more work is to be done yet.

Progress of Task A : <progress value="60" max="100">60%</progress>

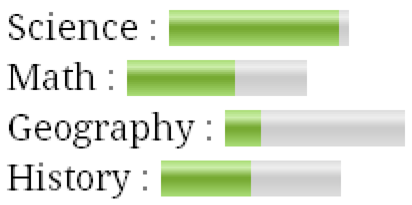
HTML5 <meter>

<meter>: "Meter" is a new element in HTML5 which represenet value of a known range as a gauge. The keyword here is "known range". That means, you are only allowed to use it when you are clearly aware of its minimum value and maximum value.

One example is score of rating. I would rate this movie <meter min="0" max="10" value="8">8 of 10</meter>.



```
Science : <meter min="0" max="100" value="95">95 of 100</meter> <br />
Math : <meter min="0" max="100" value="60">60 of 100</meter><br />
Geography : <meter min="0" max="100" value="20">20 of 100</meter> <br />
History : <meter min="0" max="100" value="50">50 of 100</meter>
```



HTML5 <mark>

<mark>: The mark <mark> element represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context.

Basically, it is used to bring the reader's attention to a part of the text that might not have been

<u>HTML5 CANVAS</u>

What is HTML Canvas?

The HTML <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics.
Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

With HTML5's Canvas API, we can draw anything and not only the rectangles, all through JavaScript. This can improve the performance of websites by avoiding the need to download images off the network. With canvas, we can draw shapes and lines, arcs and text, gradients and patterns. In addition, canvas gives us the power to manipulate pixels in images and even video.

The Canvas 2D Context spec is supported in:

- Safari 2.0+
- Chrome 3.0+
- Firefox 3.0+
- Internet Explorer 9.0+
- Opera 10.0+
- iOS (Mobile Safari) 1.0+
- Android 1.0+

Canvas Examples

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content. The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Basic Canvas Example

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
</canvas>
```

Drawing with JavaScript

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
```

Draw a Line

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

Draw a Circle

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

Draw a Text

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World",10,50);
```

Stroke Text

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World",10,50);
```

Draw Linear Gradient

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
```

```
// Create gradient
var grd = ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

Draw Circular Gradient

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

Draw Image

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById("scream");
ctx.drawImage(img,10,10);
```

What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define graphics for the Web

- SVG is a W3C recommendation

The HTML <svg> Element

The HTML <svg> element (introduced in HTML5) is a container for SVG graphics.
SVG has several methods for drawing paths, boxes, circles, text, and graphic images.

SVG Circle

Example

```
<!DOCTYPE html>
<html>
<body>


<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
</svg>


</body>
</html>
```

SVG Rectangle

Example

```
<svg width="400" height="100">
  <rect width="400" height="100" style="fill:rgb(0,0,255);stroke-width:10;stroke:rgb(0,0,0)" />
</svg>
```

SVG Rounded Rectangle

Example

```
<svg width="400" height="180">
  <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
  style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
</svg>
```

SVG Star

Example

```
<svg width="300" height="200">
  <polygon points="100,10 40,198 190,78 10,78 160,198"
  style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
</svg>
```

SVG Logo

Example

```
<svg height="130" width="500">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="100" cy="70" rx="85" ry="55" fill="url(#grad1)" />
  <text fill="#ffffff" font-size="45" font-family="Verdana" x="50" y="86">SVG</text>
  Sorry, your browser does not support inline SVG.
</svg>
```

Differences Between SVG and Canvas

SVG is a language for describing 2D graphics in XML.
Canvas draws 2D graphics, on the fly (with a JavaScript).
SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.
In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.
Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

Canvas	**SVG**
• Resolution dependent • No support for event handlers • Poor text rendering capabilities • You can save the resulting image as .png or .jpg • Well suited for graphic-intensive games	• Resolution independent • Support for event handlers • Best suited for applications with large rendering areas (Google Maps) • Slow rendering if complex (anything that uses the DOM a lot will be slow) • Not suited for game applications

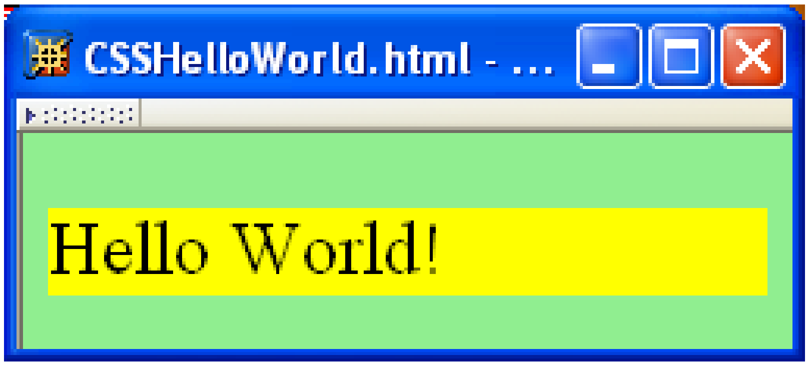
<u>CSS</u>

HTML markup can be used to represent

- ■ Semantics: h1 means that an element is a top-level heading
- ■ Presentation: h1 elements look a certain way

- It's advisable to separate semantics from presentation because:
  - ■ It's easier to present documents on multiple platforms (browser, cell phone, spoken, …)
  - ■ It's easier to generate documents with consistent look
  - ■ Semantic and presentation changes can be made independently of one another (division of labor)
  - ■ User control of presentation is facilitated
- Cascading Style Sheets (CSS)
  - ■ Applies to (X)HTML as well as XML documents in general

A styled HTML document



```
body  { background-color:lime }

p     { font-size:x-large; background-color:yellow }
```

<u>**CSS Selectors**</u>

CSS selectors allow you to select and manipulate HTML elements.
CSS selectors are used to "find" (or select) HTML elements based on their id, class, type, attribute, and more.

The element Selector

The element selector selects elements based on the element name.
You can select all <p> elements on a page like this: (all <p> elements will be center-aligned, with a red text color)

Example

```
p {
        text-align: center;
        color: red;
    }
```

The id Selector

The id selector uses the id attribute of an HTML element to select a specific element.
An id should be unique within a page, so the id selector is used if you want to select a single, unique element.

To select an element with a specific id, write a hash character, followed by the id of the element.
The style rule below will be applied to the HTML element with id="para1":

Example

```
#para1 {
    text-align: center;
    color: red;
}
```

The class Selector

The class selector selects elements with a specific class attribute.
To select elements with a specific class, write a period character, followed by the name of the class:
In the example below, all HTML elements with class="center" will be center-aligned:

Example

```
.center {
            text-align: center;
            color: red;
        }
```

You can also specify that only specific HTML elements should be affected by a class. In the example below, all <p> elements with class="center" will be center-aligned:

Example

```
p.center {
    text-align: center;
    color: red;
}
```

Grouping Selectors

If you have elements with the same style definitions, like this:

```
h1 {
    text-align: center;
    color: red;
}

h2 {
    text-align: center;
    color: red;
}

p {
    text-align: center;
    color: red;
}
```

you can group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

In the example below we have grouped the selectors from the code above:

Example

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file!

Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension. An example of a style sheet file called "myStyle.css", is shown below:

```
body {
    background-color: lightblue;
}


h1 {
    color: navy;
    margin-left: 20px;
}
```

Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.
Internal styles are defined within the <style> element, inside the head section of an HTML page:

Example

```
<head>
<style>
body {
    background-color: linen;
}


h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

Inline Styles

An inline style may be used to apply a unique style for a single element.
An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly!
To use inline styles, add the style attribute to the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a <h1> element:

Example

```
<h1 style="color:blue;margin-left:30px;">This is a heading.</h1>
```

Multiple Style Sheets

If some properties have been defined for the same selector in different style sheets, the value will be inherited from the more specific style sheet.
For example, assume that an external style sheet has the following properties for the <h1> element:

```
h1 {
    color: navy;
    margin-left: 20px;
}
```

then, assume that an internal style sheet also has the following property for the <h1> element:

```
h1 {
    color: orange;
}
```

If the page with the internal style sheet also links to the external style sheet the properties for the <h1> element will be:

color: orange;
margin-left: 20px;

The left margin is inherited from the external style sheet and the color is replaced by the internal style sheet.

Multiple Styles Will Cascade into One

Styles can be specified:

- in an external CSS file
- inside the <head> section of an HTML page
- inside an HTML element

Cascading order

What style will be used when there is more than one style specified for an HTML element?
Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number three has the highest priority:

1. Browser default
2. External and internal style sheets (in the head section)
3. Inline style (inside an HTML element)

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

CSS3 Introduction

CSS3 is the latest standard for CSS.
CSS3 is completely backwards-compatible with earlier versions of CSS.

CSS3 has been split into "modules". It contains the "old CSS specification" (which has been split into smaller pieces). In addition, new modules are added.

Some of the most important CSS3 modules are:

- Selectors
- Box Model
- Backgrounds and Borders
- Image Values and Replaced Content
- Text Effects
- 2D/3D Transformations
- Animations
- Multiple Column Layout
- User Interface

CSS3 border-radius Property

If you specify only one value for the border-radius property, this radius will be applied to all 4 corners.

However, you can specify each corner separately if you wish. Here are the rules:

- **Four values:** first value applies to top-left, second value applies to top-right, third value applies to bottom-right, and fourth value applies to bottom-left corner
- **Three values:** first value applies to top-left, second value applies to top-right and bottom-left, and third value applies to bottom-right
- **Two values:** first value applies to top-left and bottom-right corner, and the second value applies to top-right and bottom-left corner
- **One value:** all four corners are rounded equally

Example

```
#rcorners4 {
   border-radius: 15px 50px 30px 5px;
   background: #8AC007;
   padding: 20px;
   width: 200px;
   height: 150px;
}

#rcorners5 {
   border-radius: 15px 50px 30px;
   background: #8AC007;
   padding: 20px;
   width: 200px;
   height: 150px;
}

#rcorners6 {
   border-radius: 15px 50px;
   background: #8AC007;
   padding: 20px;
   width: 200px;
   height: 150px;
}
```

CSS3 Rounded Corners Properties

Property	Description
border-radius	A shorthand property for setting all the four border-*-*-radius properties
border-top-left-radius	Defines the shape of the border of the top-left corner
border-top-right-radius	Defines the shape of the border of the top-right corner
border-bottom-right-radius	Defines the shape of the border of the bottom-right corner
border-bottom-left-radius	Defines the shape of the border of the bottom-left corner

CSS3 border-image Property

The CSS3 border-image property allows you to specify an image to be used instead of the normal border around an element.
The property has three parts:

1. The image to use as the border
2. Where to slice the image
3. Define whether the middle sections should be repeated or stretched

CSS3 Border Properties

Property	**Description**
border-image	A shorthand property for setting all the border-image-* properties
border-image-source	Specifies the path to the image to be used as a border
border-image-slice	Specifies how to slice the border image
border-image-width	Specifies the widths of the border image
border-image-outset	Specifies the amount by which the border image area extends beyond the border box
border-image-repeat	Specifies whether the border image should be repeated, rounded or stretched

CSS3 Backgrounds

CSS3 contains a few new background properties, which allow greater control of the background element.
In this chapter you will learn how to add multiple background images to one element.
You will also learn about the following new CSS3 properties:

- background-size
- background-origin

- background-clip

CSS3 Multiple Backgrounds

CSS3 allows you to add multiple background images for an element, through the background-image property.
The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer.
The following example has two background images, the first image is a flower (aligned to the bottom and right) and the second image is a paper background (aligned to the top-left corner):

Example

```
#example1 {
    background-image: url(img_flwr.gif), url(paper.gif);
    background-position: right bottom, left top;
    background-repeat: no-repeat, repeat;
}
```

Multiple background images can be specified using either the individual background properties (as above) or the background shorthand property.
The following example uses the background shorthand property (same result as example above):

Example

```
#example1 {
    background: url(img_flwr.gif) right bottom no-repeat, url(paper.gif) left top repeat;
}
```

CSS3 Background Properties

Property	Description
background	A shorthand property for setting all the background properties in one

	declaration
background-clip	Specifies the painting area of the background
background-image	Specifies one or more background images for an element
background-origin	Specifies where the background image(s) is/are positioned
background-size	Specifies the size of the background image(s)

CSS3 background-origin Property

The CSS3 background-origin property specifies where the background image is positioned.
The property takes three different values:

- border-box - the background image starts from the upper left corner of the border
- padding-box - (default) the background image starts from the upper left corner of the padding edge
- content-box - the background image starts from the upper left corner of the content

CSS3 background-clip Property

The CSS3 background-clip property specifies the painting area of the background.
The property takes three different values:

- border-box - (default) the background is painted to the outside edge of the border
- padding-box - the background is painted to the outside edge of the padding
- content-box - the background is painted within the content box

CSS3 Colors

CSS supports color names, hexadecimal and RGB colors.
In addition, CSS3 also introduces:

- RGBA colors
- HSL colors

- HSLA colors
- opacity

RGBA Colors

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.
An RGBA color value is specified with: rgba(red, green, blue, alpha). The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).
Example
#p1 {background-color: rgba(255, 0, 0, 0.3);} /* red with opacity */

HSL Colors

HSL stands for Hue, Saturation and Lightness.
An HSL color value is specified with: hsl(hue, saturation, lightness).

1. Hue is a degree on the color wheel (from 0 to 360):
   - 0 (or 360) is red
   - 120 is green
   - 240 is blue
2. Saturation is a percentage value: 100% is the full color.
3. Lightness is also a percentage; 0% is dark (black) and 100% is white.

Example: #p1 {background-color: rgba(255, 0, 0, 0.3);} /* red with opacity */

HSLA Colors

HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.
An HSLA color value is specified with: hsla(hue, saturation, lightness, alpha), where the alpha parameter defines the opacity. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Example

#p1 {background-color: hsla(120, 100%, 50%, 0.3);} /* green with opacity */

Opacity

The CSS3 opacity property sets the opacity for a specified RGB value.
The opacity property value must be a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Example: #p1 {background-color:rgb(255,0,0);opacity:0.6;} /* red with opacity */

CSS3 Gradients

CSS3 gradients let you display smooth transitions between two or more specified colors.
CSS3 gradients you can reduce download time and bandwidth usage. In addition, elements with gradients look better when zoomed, because the gradient is generated by the browser.
CSS3 defines two types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**

CSS3 Linear Gradients

To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.
Example of Linear Gradient:

Syntax

background: linear-gradient(*direction*, *color-stop1*, *color-stop2*, ...);

Example

A linear gradient from top to bottom: background: linear-gradient(red, blue); /* Standard syntax */

A linear gradient from left to right: background: linear-gradient(to right, red , blue);

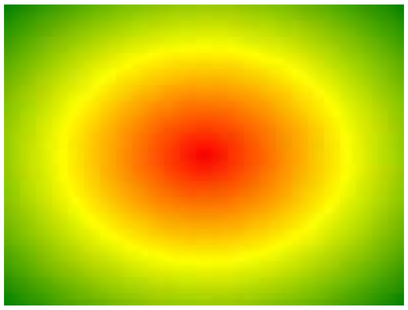
A linear gradient that starts at top left (and goes to bottom right): background: linear-gradient(to bottom right, red , blue); /* Standard syntax */

CSS3 Radial Gradients

A radial gradient is defined by its center.

To create a radial gradient you must also define at least two color stops.

Example of Radial Gradient:



Syntax

background: radial-gradient(*shape size* at *position, start-color, ..., last-color*);

CSS3 Text Shadow

The CSS3 text-shadow property applies shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

text-shadow: *h-shadow v-shadow blur-radius color*|none|initial|inherit;

CSS3 Box Shadow

box-shadow: none|*h-shadow v-shadow blur spread color* |inset|initial|inherit;

CSS3 Shadow Properties

The following table lists the CSS3 shadow properties:

Property	**Description**

box-shadow	Adds one or more shadows to an element
text-shadow	Adds one or more shadows to a text

CSS3 2D Transforms

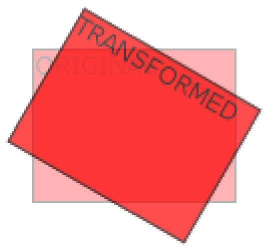
The following are 2D transformation methods:

- translate()
- rotate()
- scale()
- skewX()
- skewY()
- matrix()

The translate() Method

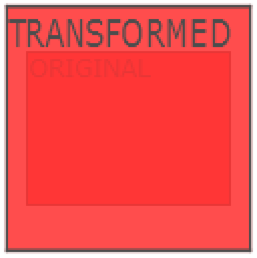
The translate() method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

The rotate() Method



The rotate() method rotates an element clockwise or counter-clockwise according to a given degree.

The scale() Method



The scale() method increases or decreases the size of an element (according to the parameters given for the width and height).

The skewX() Method

The skewX() method skews an element along the X-axis by the given angle.

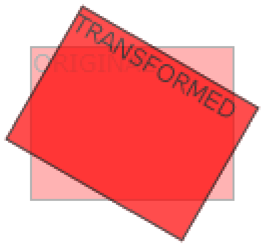
The skewY() Method

The skewY() method skews an element along the Y-axis by the given angle.

The skew() Method

The skew() method skews an element along the X and Y-axis by the given angles.

T he matrix() Method



The matrix() method combines all the 2D transform methods into one.

CSS3 Transform Properties

The following table lists all the 2D transform properties:

Property	**Description**

transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements

2D Transform Methods

Function	Description
matrix(*n*,*n*,*n*,*n*,*n*,*n*)	Defines a 2D transformation, using a matrix of six values
translate(*x*,*y*)	Defines a 2D translation, moving the element along the X- and the Y-axis
translateX(*n*)	Defines a 2D translation, moving the element along the X-axis
translateY(*n*)	Defines a 2D translation, moving the element along the Y-axis
scale(*x*,*y*)	Defines a 2D scale transformation, changing the elements width and height
scaleX(*n*)	Defines a 2D scale transformation, changing the element's width
scaleY(*n*)	Defines a 2D scale transformation, changing the element's height
rotate(*angle*)	Defines a 2D rotation, the angle is specified in the parameter
skew(*x-angle*,*y-angle*)	Defines a 2D skew transformation along the X- and the Y-axis
skewX(*angle*)	Defines a 2D skew transformation along the X-axis
skewY(*angle*)	Defines a 2D skew transformation along the Y-axis

CSS3 Transitions

CSS3 transitions allows you to change property values smoothly (from one value to another), over a given duration.

Example: Mouse over the element below to see a CSS3 transition effect

CSS3 Transition Properties

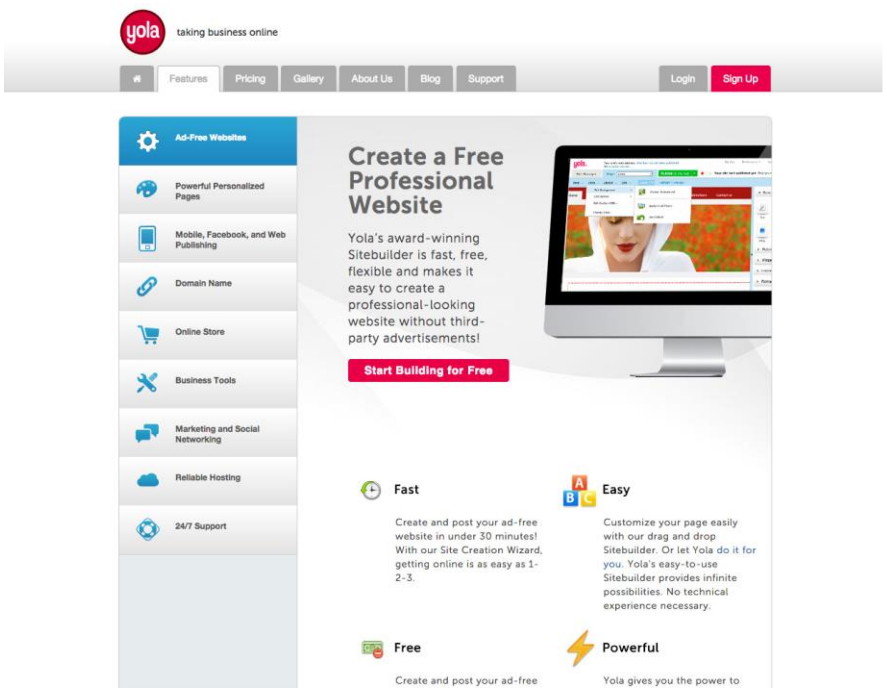
The following table lists all the transition properties:

Property	Description
transition	A shorthand property for setting the four transition properties into a single property
transition-delay	Specifies a delay (in seconds) for the transition effect
transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete
transition-property	Specifies the name of the CSS property the transition effect is for
transition-timing-function	Specifies the speed curve of the transition effect

WEBSITE CREATION USING TOOLS

Few of the tools which helps for easy website creation are

1. Yola



Create a Free Professional Website

Yola's award-winning Sitebuilder is fast, free, flexible and makes it easy to create a professional-looking website without third-party advertisements!

Fast

Create and post your ad-free website in under 30 minutes! With our Site Creation Wizard, getting online is as easy as 1-2-3.

Easy

Customize your page easily with our drag and drop Sitebuilder. Or let Yola do it for you. Yola's easy-to-use Sitebuilder provides infinite possibilities. No technical experience necessary.

Free

Create and post your ad-free

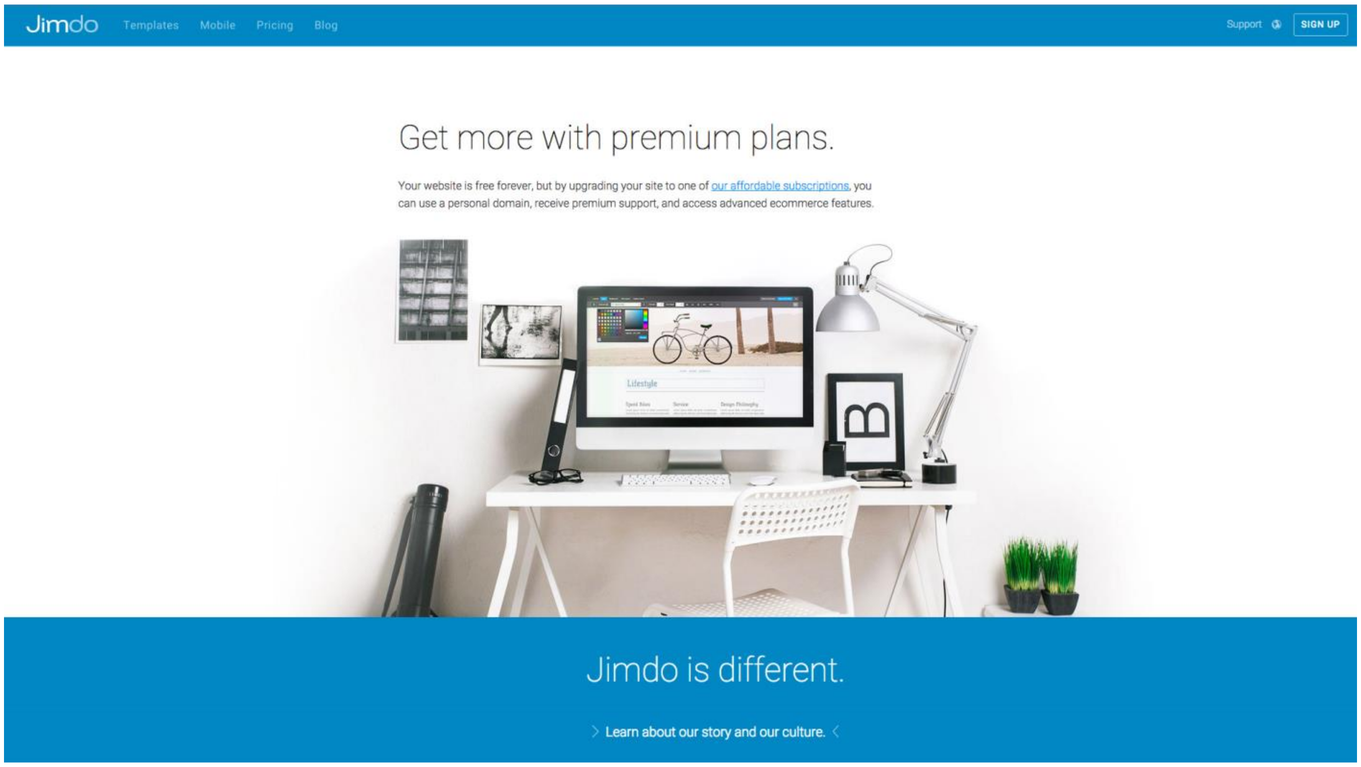
Powerful

Yola gives you the power to

What it does: Yola lets you build a basic website by picking a template and filling out a few simple forms. Once you have a rough outline, you fine-tune your site with an in-place editing tool. Yola has the goods to let you really dig into the web. You can integrate your site with an impressive list of third-party services such as Google Maps and PayPal, Flickr for photo sharing and Picnik for photo editing.

What it costs: The basic web-building tool and a Yola.com address are free. For extra features, better-looking templates and the ability to use your own domain name, the Yola Silver upgrade is $100 per year.

2. Jimdo



Get more with premium plans.

Your website is free forever, but by upgrading your site to one of our affordable subscriptions, you can use a personal domain, receive premium support, and access advanced ecommerce features.

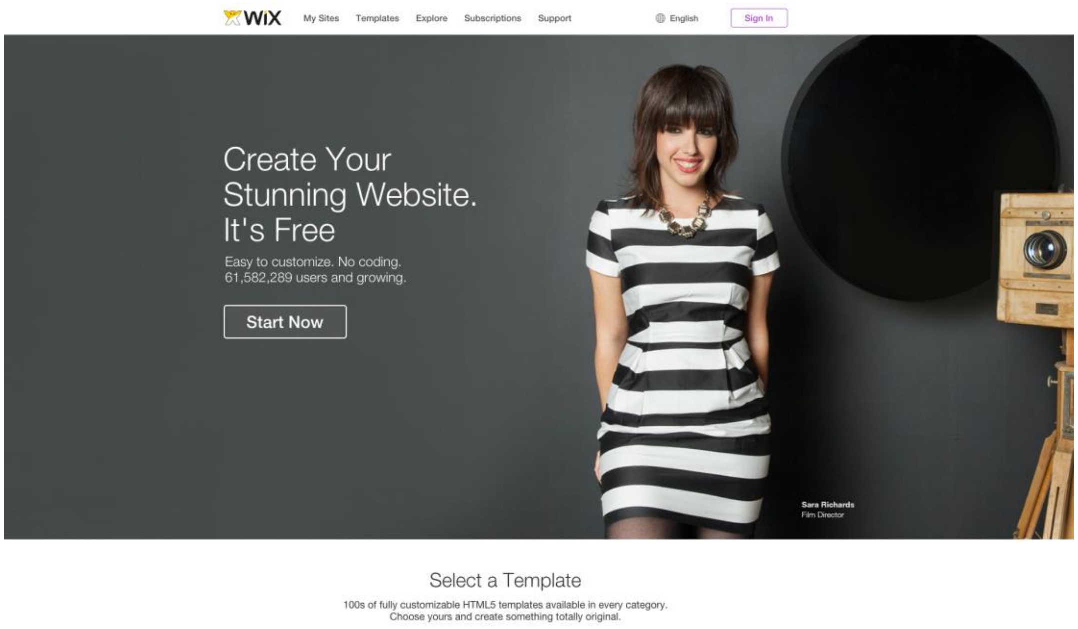
Jimdo is different.

Learn about our story and our culture.

What it does: Jimdo's free version does what a respectable website builder should do, and not much else. We suggest springing for the upgrades (which are reasonably priced) to unlock some cool business features, such as custom newsletters to keep in touch with your customers, page-view stats, PayPal stores and password-protected employees-only pages.

What it costs: Basic features and a Jimdo.com address are free. Jimdo Pro is $5 per month. Jimdo Business is $15 per month, including unlimited data storage and online selling, two domain names and business-specific site designs.

3. WIX

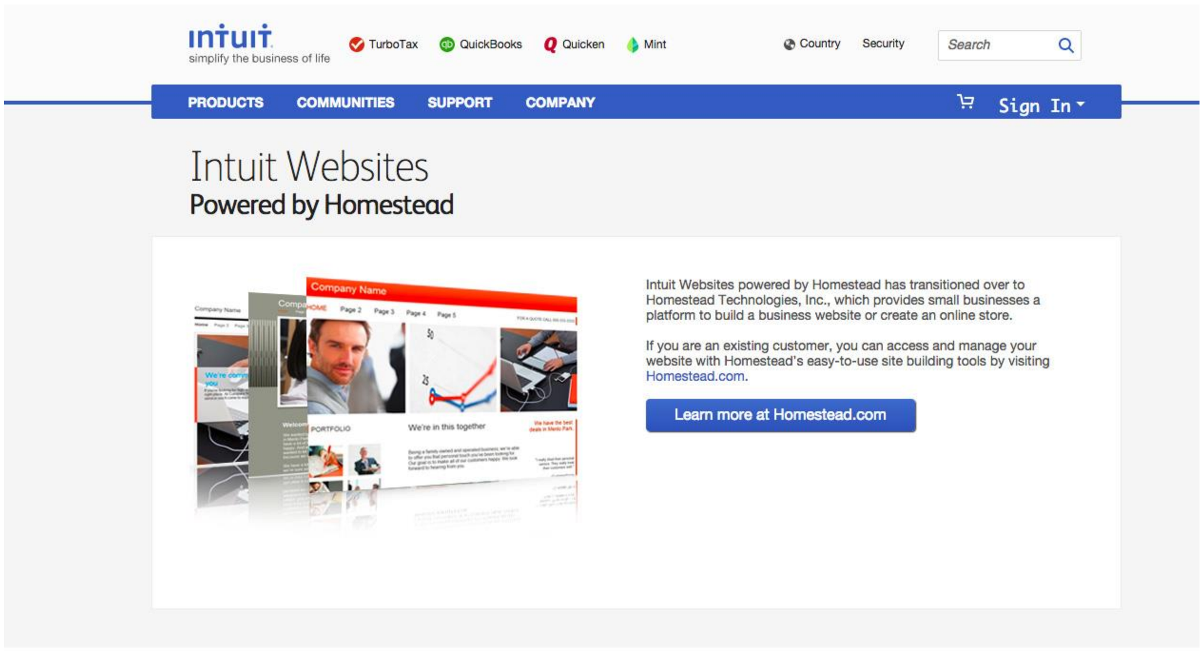

WiX
My Sites
Templates
Explore
Subscriptions
Support
English
Sign In
Create Your Stunning Website. It's Free
Easy to customize. No coding.
61,582,289 users and growing.
Start Now
Sara Richards
Film Director
Select a Template
100s of fully customizable HTML5 templates available in every category.
Choose yours and create something totally original.

What it does: Wix lets you build a great-looking website in no time with its easy-to-use, what-you-see-is-what-you-get editor. Here's the downside: The web development tool is based on Adobe Flash, which works on most PCs but isn't supported by some mobile devices, including the all-powerful Apple iPad. If that isn't a problem for you, Wix has lots of elegant templates that are easy to customize and can fit every business need. Wix's image-heavy sites are especially great for photo galleries, which really show clients what your business can do. A new mobile tool lets you build a simple, smartphone-optimized site to reach on-the-go clients.

What it costs: The full-featured website-building tool and Wix.com address are free. Paid subscriptions, which let you do things like remove ads and link a site to your own domain name, run $5 to $16 per month.

4. Intuit Websites

What it does: Starting a business takes creativity, but maybe you're not the artistic type. Luckily, even the most design-challenged among us can squeeze a respectable-looking website out of Intuit's somewhat bland but reliable web-editing tool. A quick survey helps you pick a template that's based on what your business does and your goals for the website. The template sorter goes into more detail than many website builders that make you wade through thousands of templates. From there you can tinker with the look and layout, but with some quick text and picture entries you'll be well on your way to a reasonable web presence.



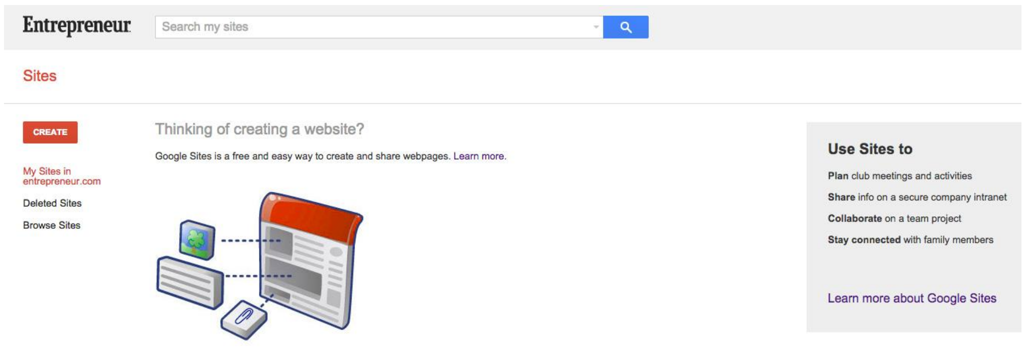
Intuit Websites
Powered by Homestead

Intuit Websites powered by Homestead has transitioned over to Homestead Technologies, Inc., which provides small businesses a platform to build a business website or create an online store.

If you are an existing customer, you can access and manage your website with Homestead's easy-to-use site building tools by visiting Homestead.com.

What it costs: The starter package is free for 30 days, then $5 per month. Business and Professional packages are $24 and $50 per month, respectively, and include features like custom domain names, online selling tools and search engine optimization. Insider's note: Intuit has several resale agreements with large telecom companies like Verizon, so don't be afraid to dig around to find a package discount.

5. Google Sites



Thinking of creating a website?

Google Sites is a free and easy way to create and share webpages. Learn more.

Use Sites to

Plan club meetings and activities
Share info on a secure company intranet
Collaborate on a team project
Stay connected with family members

Learn more about Google Sites

What it does: This service can give you a simple Google web presence for free. But you probably don't need that when there are better, faster and easier options from the likes of Facebook, Twitter and LinkedIn. What Google Sites does best happens outside the public eye. With this tool you can create private, team-based websites within your business--which turns that Google Apps account of yours into a powerful business organization tool. Google Sites puts nifty collaboration tools like announcements, documents and task lists in one online location, allowing you and your colleagues to access them from anywhere. It's a great way to bring some sanity to the startup chaos.

What it costs: It's free.

Source:

http://www.sqa.org.uk/e-learning/ClientSide01CD/index.htm

http://www.w3schools.com/

http://www.entrepreneur.com/article/220447

http://www.tutorialspoint.com/javascript/

https://www.codeschool.com/paths/javascript

http://www.htmlandcssbook.com

PPT present in internet in regard to the above topics

Books:

1. Harvey Deitel, Abbey Deitel, Internet and World Wide Web: How To Program 5th Edition.

2. Michael Moncur, Sams Teach yourself Javascript in 24 hours

3. Jeffrey C. Jackson, WEB TECHNOLOGIES A Computer Science Perspective