UNIT II
Style Sheets: CSS

Introduction to Cascading Style Sheets

Cascading Style Sheets (CSS) is a slightly misleading term, since a website might have only one CSS file (style sheet), or the CSS might be embedded within an HTML file.

It is better to think of CSS as a technology (in the singular). CSS is comprised of statements that control the styling of HTML documents. Simply put, an HTML document should convey content.

A CSS document should control the styling of that content. `<div align="center"></div> <img src="this.gif" border="0" alt="" /> <table height="200">... <td width="30"></td>` All these examples can easily be replaced with CSS. Don't worry if you don't understand these declarations yet. `div {text-align: center;} img {border: 0 none;} table {height: 200px;} td {width: 30px;}` An HTML file points to one or more external style sheets (or in some cases a list of declarations embedded within the head of the HTML file) which then controls the style of the HTML document. These style declarations are called CSS rules.

Features :-

The latest version of Cascade Style Sheets, CSS 3, was developed to make Web design easier but it became a hot topic for a while because not all browsers supported it.

However, trends change quickly in technology and all browser makers currently are implementing complete CSS 3 support.

Making that process easier for the browser manufacturers is CSS 3's modularized specification, which allows them to provide support for modules incrementally without having to perform major refactoring of the browsers' codebases.

The modularization concept not only makes the process of approving individual CSS 3 modules easier and faster, but it also makes documenting the spec easier. Eventually, CSS 3 -- along with HTML5 -- are going to be the future of the web.

You should begin making your Web pages compatible with these latest specifications. In this article, I explore 10 of the exciting new features in CSS 3, which is going to change the way developers who used CSS2 build websites.Some of the features are:

- **CSS Text Shadow**
- **CSS Selectors**
- **CSS Rounded Corners**
- **CSS Border Image**

Core Syntax

At-Rules

As we learned when we studied CSS statements, there are two types of statements. The most common is the rule-sets statement, and the other is the at-rules statement. As opposed to rule sets, at-rules statements consist of three things: the at-keyword, @, an identifier, and a declaration. This declaration is defined as all content contained within a set of curly braces, or by all content up until the next semicolon.

@import

Perhaps the most commonly used of the at-rules, @import, is used to import an external style sheet into a document. It can be used to replace the LINK element, and serves the same function, except that imported style sheets have a lower weight (due to having less proximity) than linked style sheets. `<style type="text/css" media="screen"> @import url(imported.css); </style> @import url(addonstyles.css); @import "addonstyles.css";` Relative and absolute URLs are allowed, but only one is allowed per instance of @import. One or more comma-separated target media may be used here.

@charset

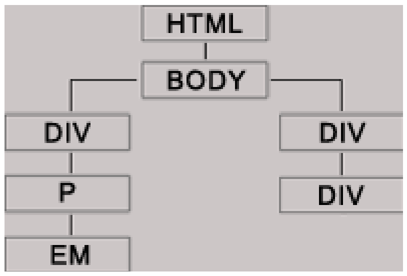
@charset is used to specify the character encoding of a document, and must appear no more than once. It must be the very first declaration in the external style sheet, and cannot appear in embedded style sheets. @charset is used by XML documents to define a character set. `@charset "utf-8";`

@namespace

The @namespace rule allows the declaration of a namespace prefix to be used by selectors in a style sheet. If the optional namespace prefix is omitted, then the URL provided becomes the default namespace. Any @namespace rules in a style sheet must come after all @import and @charset at-rules, and come before all CSS rule-sets. `@namespace foo url("http://www.example.com/");` @namespace can be used together with the new CSS3 selectors (see below). It defines which XML namespace to use in the CSS. If the XML document doesn't have matching XML namespace information, the CSS is ignored.

@font-face

CSS1 Selectors



Selectors refer to elements in an HTML document tree. Using CSS, they are pattern-matched in order to apply styles to those elements. A selector consists of one or more elements, classes, or IDs, and may also contain pseudo-elements and/or pseudo-classes.

Type Selector

The type selector is the simplest selector of all, and matches all occurrences of an element. In this example, all <p> tags throughout the document will have the following style applied, unless overridden. `p {color: #666;}`

Universal Selector

The universal selector, used alone, matches all elements in the document tree, and thus will apply styles to all elements. It is in effect a wildcard. `* {margin: 0; padding: 0;}` In this example, all tags are reset to have no padding or margin. This, by the way, is a practice to gain control over all the default padding and margin inherent in the way User Agents (UAs) display HTML.

Class Selector

The class selector matches a classname. `.largeFont {font-size: 1.5em;} h3.cartHeader {text-align: center;}` The "largeFont" class will apply to all elements into which it is called. The "cartHeader" class will only function as styled if called into an H3 element. This is useful if you have another "cartHeader" declaration that you wish to override in the context of an H3 element, or if you wish to enforce the placement of this class. Powered by

ID Selector

The ID selector matches an ID. IDs are identifiers unique to a page. They bear a resemblance to classes, but are used a bit differently. IDs will be treated more fully below. The first two ID examples below refer to sections of a web page, while the last refers to a specific occurrence of an item, say, an image in a DHTML menu. IDs have a higher specificity than classes. `#header {height: 100px;} #footer {color: #F00;} #xyz123 {font-size: 9px;}`

Descendant Selector

A selector can itself be a chain of one or more selectors, and is thus sometimes called a compound selector. The descendant selector is the only compound selector in CSS1, and consists of two or more selectors and one or more white space combinators. In the example below, the white space between the H1 and EM elements is the descendant combinator.

In other words, white space conveys a hierarchy. (If a comma were to have intervened instead, it would mean that we were styling H1 and EM elements alike.) Selectors using combinators are used for more precise drill-down to specific points within the document tree. In this example <em> tags will have the color red applied to them if they are within an <h1> tag. `h1 em {color: #F00;}` Note that EM elements do not have to be immediately inside an H1 heading, that is, they do not have to be children, but merely descendants of their ancestor. The previous style would apply to an EM element in either of the following statements. `<h1>This is a <em>main</em> heading</h1> <h1>This is <strong>another <em>main</em> heading</strong></h1>`

STYLE SHEETS AND HTML STYLE RULE

To apply a style, CSS uses the HTML document tree to match an element, attribute, or value in an HTML file. For an HTML page to properly use CSS, it should be well-formed and valid, and possess a valid doctype.

If these conditions are not met the CSS match may not yield the desired results.There are two types of CSS statements: rule-sets and at-rules.

A rule set, also known simply as a rule, is the more common statement, and consists of a selector and a declaration block, sometimes simply called a block. The selector can be an element, class, or ID, and may include combinators, pseudo-elements, or pseudo-classes. **Statement Type 1: Rules Sets (Rules)** statement + statement block X {declaration; declaration;} X {property; value; property: value;} div > p {font-size: 1em; color #333;} **Statement Type 2: At-Rules** at-keyword + identifier + declaration @import "subs.css";

Properties

I have decided not to include a description of all CSS1 and CSS2.1 Properties (such as *font-size, text-transform, border, margin,* and many others) because they are numerous and can be examined in the Property References section of this site. Moreover, they are used throughout this tutorial and can be easily deduced. So we move directly to CSS1 selectors.

STYLE RULE CASCADING AND INHERITANCE:-

CSS are probably wondering what exactly *cascades* about cascading style sheets. In this section we look at the idea of cascading, and a related idea, that of inheritance.

Both are important underlying concepts that you will need to grasp, and understand the difference between, in order to work properly with style sheets.

Rule Cascade

A single style sheet associated with one or more web pages is valuable, but in quite a limited way. For small sites, the single style sheet is sufficient, but for larger sites, especially sites managed by more than one person (perhaps several teams who may never communicate) single style sheets don't provide the ability to share common styles, and extend these styles where necessary. This can be a significant limitation.

Cascading style sheets are unlike the style sheets you might have worked with using word processors, because they can be linked together to create a hierarchy of related style sheets. Managing style at large sites using @import Imagine how the web site for a large organization, say a corporation, might be structured. As sites grow in complexity, individual divisions, departments, and workgroups become more responsible for their own section of a site. We can already see a potential problem - how do we ensure a consistent look and feel across the whole site?A dedicated web development team can ensure that a style guide is adhered to.

Specificity

Get browser support information for specificity in the downloadable version of this guide or our browser support tables.At this point it might be timely to have a quick discussion of specificity. Both inside a single style sheet, and in a cascade of style sheets, it should be clear that more than one rule can apply to the same element. What happens when two properties in separate rules which both apply to an element contradict one another? Obviously they can't both apply (the text of an element can't be both red and blue, for example). CSS provides a mechanism for resolving these conflicts, called specificity. Some selectors are more specific than others. For example, the class and ID selectors are more specific than simple HTML element selectors. When two rules select the same element and the properties contradict one another, the rule with the more specific selector takes precedence. Specificity for selectors is a little involved. Without going into the full detail, most situations can be resolved with the following rules.

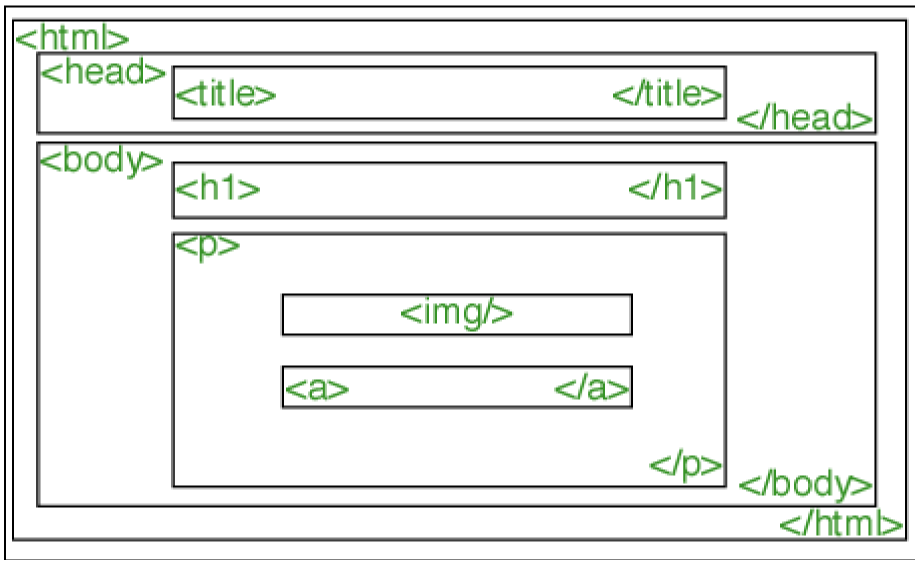
1. ID selectors are more specific than other selectors
2. Class selectors are more specific than HTML element selectors, and other selectors such as contextual, pseudo class and pseudo element selectors.
3. Contextual selectors, and other selectors involving more than one HTML element selector are more specific than a single element selector (and for two multiple element selectors, the one with more elements is more specific than the one with fewer.)

Style Inheritance

Any HTML page comprises a number of (perhaps a large number of) elements - headings, paragraphs, lists, and so on. Often, developers use the term "tag" to refer to an element, makingreference for example to "the p tag". But the tag is simply the `<p></p>` part of the element. The whole construction of `<p>This is the content of the paragraph</p>` is in fact the `<p>` element (as we refer to it in this guide).

What many web developers don't realize (largely because it wasn't particularly important until style sheets came along) is that every element is contained by another element, and may itself contain other elements. The technical term for this is the containment hierarchy of a web page. At the top of the containment hierarchy is the `<html>` element of the page.

Every other element on a web page is contained within the `<html>` element, or one of the elements contained within it, and so on. Similarly, many elements will be contained in paragraphs, while paragraphs are contained in the `<body>`. Graphically, we can understand it like this.



Text t properties :-

CSS Font Families

CSS font properties define the font family, boldness, size, and the style of a text.

Difference Between Serif and Sans-serif Fonts

On computer screens, sans-serif fonts are considered easier to read than serif fonts. In CSS, there are two types of font family names:

generic family - a group of font families with a similar look (like "Serif" or "Monospace")

font family - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New Lucida Console	All monospace characters have the same width

The CSS Box Model

BLOCK DIAGRAM

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout. The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content. The box model allows us to place a border around elements and space elements in relation to other elements. The image below illustrates the box model:

NORMAL FLOW BOX LAYOUT

Understanding the box model is critical to developing web pages that don't rely on tables for layout. In the early days of writing HTML, before the advent of CSS, using tables was the only way to have discreet content in separate boxes on a page. But tables were originally conceived to display tabular information.

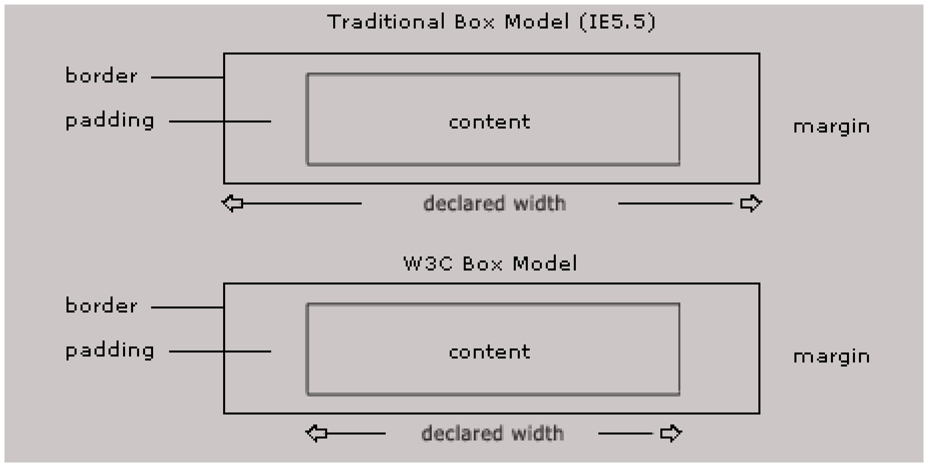
With the advent of CSS floating and positioning, there is no longer a need to use tables for layout, though many years later many, if not most, sites are still using tables in this manner.The box model, as defined by the W3C "describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model".

Don't be confused by the term "boxes". They need not appear as square boxes on the page. The term simply refers to discreet containers for content. In fact, every element in a document is considered to be a rectangular box.

Padding, Borders, Margins

Padding immediately surrounds the content, between content and borders. A margin is the space outside of the borders. If there are no borders both paddng and margin behave in roughly the same way, except that you can have negative margins, while you cannot have negative padding. Also padding does not collapse like margins. See below for the section on collapsing margins. The picture on the right illustrates padding, borders, and margins. The content area does not really have a border. The line around the content merely indicates the limits of the actual content.

Traditional vs. W3C Box Models



Beyond the Normal Flow

Positioning

The CSS positioning properties allow you to position an element. It can also place an element behind another, and specify what should happen when an element's content is too big.Elements can be positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the positioning method.There are four different positioning methods.

Static Positioning

HTML elements are positioned static by default. A static positioned element is always positioned according to the normal flow of the page.Static positioned elements are not affected by the top, bottom, left, and right properties.

Fixed Positioning

An element with fixed position is positioned relative to the browser window.It will not move even if the window is scrolled:

Example

p.pos_fixed { position:fixed; top:30px; right:5px; } Note: Internet Explorer supports the fixed value only if a !DOCTYPE is specified.Fixed positioned elements are removed from the normal flow. The document and other elements behave like the fixed positioned element does not exist.Fixed positioned elements can overlap other elements.

CLIENT SIDE PROGRAMMING:JAVA SCRIPT

Introduction

JavaScript is most commonly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.JavaScript can be used in other contexts than a Web browser.

Netscape created server-side JavaScript as a CGI-language that can do roughly the same as Perl or ASP.

There is no reason why JavaScript couldn't be used to write real, complex programs. However, this site exclusively deals with theuse of JavaScript in web browsers.I can also recommend Jeremy Keith, DOM Scripting: Web Design with JavaScript and the Document Object Model, 1st edition, Friends of Ed, 2005.

This, too, is a book that doesn't delve too deeply into technology, but gives non-programmers such as graphic designers/CSS wizards an excellent overview of the most common uses of JavaScript - as well as the most common problems.

History and Versions of The JavaScript language

JavaScript is not a programming language in strict sense. Instead, it is a scripting language because it uses the browser to do the dirty work. If you command an image to be replaced by another one, JavaScript tells the browser to go do it. Because the browser actually does the work, you only need to pull some strings by writing some relatively easy lines of code.

That's what makes JavaScript an easy language to start with. But don't be fooled by some beginner's luck: JavaScript can be pretty difficult, too. First of all, despite its simple appearance it is a full fledged programming language: it is possible to write quite complex programs in JavaScript. This is rarely necessary when dealing with web pages, but it is possible. This means that there are some complex programming structures that you'll only understand after protracted studies.

JavaScript versions

There have been several formal versions of JavaScript.
1.0: Netscape 2

1.1: Netscape 3 and Explorer 3 (the latter has *bad* JavaScript support, regardless of its version)

1.2: Early Version 4 browsers

1.3: Later Version 4 browsers and Version 5 browsers

1.4: Not used in browsers, only on Netscape servers

INTRODUCTION TO JAVA SCRIPT

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- A JavaScript consists of lines of executable computer code
- A JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

What can a JavaScript Do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer
- **BASIC SYNTAX**
- **How to Put a JavaScript Into an HTML Page**

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
```

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

Hello World!

Example Explained

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the "type=" attribute to define the scripting language. So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:

```
<html>

<body>

<script type="text/javascript">

...

</script>

</body>

</html>
```

JAVASCRIPT VARIABLES AND DATATYPES

As with algebra, JavaScript variables are used to hold values or expressions. A variable can have a short name, like **x**, or a more describing name like **length**. A JavaScript variable can also hold a text value like in **carname="Volvo"**. Rules for JavaScript variable names:
Variable names are case sensitive (**y** and **Y** are two different variables)
Variable names must **begin with a letter** or the underscore character

Example

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables. You can declare JavaScript variables with the **var statement**: var x; var carname; After the declaration shown above, the variables has no values, but you can assign values to the variables while you declare them: var x=5; var carname="Volvo";

Assigning Values to JavaScript Variables

You assign values to JavaScript variables with **assignment statements**: x=5; carname="Volvo"; The variable name is on the left side of the = sign, and the value you want to assign to the variable is on the right. After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that has not yet been declared, the variables will automatically be declared.These statements: x=5; carname="Volvo"; have the same effect as: var x=5; var carname="Volvo";

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value. var x=5; var x; After the execution of the statements above, the variable x will still have the value of 5. The value of x is not reset (or cleared) when you redeclare it.

DataTypes

- ✓ Numbers - are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
- ✓ Strings - are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
- ✓ Boolean (**true**/**false**) - lets you evaluate whether a condition meets or does not meet specified criteria.
- ✓ Null - is an empty value. **null** is not the same as 0 **--** 0 is a real, calculable number, whereas **null** is the **absence of any value**.

Data Types TYPE	**EXAMPLE**
Numbers	Any number, such as 17, 21, or 54e7
Strings	"Greetings!" or "Fun"
Boolean	Either true or false
Null	A special keyword for exactly that – the null value (that is, nothing)

JAVASCRIPT STATEMENTS

A JavaScript statement is a command to the browser. The purpose of the command is to tell the browser what to do.

This JavaScript statement tells the browser to write "Hello Dolly" to the web page: document.write("Hello Dolly"); It is normal to add a semicolon at the end of each executable statement. Most people think this is a good programming practice, and most often you will see this in JavaScript examples on the web.

The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end. **Note:** Using semicolons makes it possible to write multiple statements on one line.

JavaScript Code

JavaScript code (or just JavaScript) is a sequence of JavaScript statements. Each statement is executed by the browser in the sequence they are written.

This example will write a header and two paragraphs to a web page:

```
<script type="text/javascript">

document.write("<h1>This is a header</h1>");

document.write("<p>This is a paragraph</p>");

document.write("<p>This is another paragraph</p>");

</script>
```

JAVASCRIPT OPERATORS

= is used to assign values. + is used to add values.

The assignment operator = is used to assign values to JavaScript variables.

The arithmetic operator + is used to add values together. y=5; z=2; x=y+z; The value of x, after the execution of the statements above is 7.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Operator	**Description**	**Example**	**Result**
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1

++	Increment	x=++y	x=6

JAVASCRIPT FUNCTIONS

A function will be executed by an event or by a call to the function.

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function.A function contains code that will be executed by an event or by a call to the function.You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

OBJECTS

JavaScript Objects represent self contained entities consisting of variables (called *properties* in object terminology) and functions (called *methods*) that can be used to perform tasks and store complex data. JavaScript objects fall into three categories:

Built-in Objects, Custom Objects and Document Object Model (DOM) Objects. Built-in objects are objects that are provided with JavaScript to make your life as a JavaScript developer easier. In many of the examples given in this book we have used the document.write() mechanism to write text to the current web page.

Whether you knew it or not, you have been using the write() method of the JavaScript built-in document *object* when you have run these scripts. Document Object Model (DOM) Objects provide the foundation for creating dynamic web pages. The DOM provides the ability for a JavaScript script to access, manipulate, and extend the content of a web page dynamically (i.e. without having to reload the page).

The DOM essentially presents the web page as a tree hierarchy of objects representing the contents and elements of the web page. These objects, in turn, contain *properties* and *methods* that allow you to access and change parts of the web page. Custom objects are objects that you, as a JavaScript developer, create and use.

BUILT –IN OBJECTS

Java Script String

The String object is used to manipulate a stored piece of text.

Complete String Object Reference

For a complete reference of all the properties and methods that can be used with the String object, go to our complete String object reference. The reference contains a brief description and examples of use for each property and method!

String object

The String object is used to manipulate a stored piece of text. **Examples of use:** The following example uses the length property of the String object to find the length of a string: var txt="Hello world!"; document.write(txt.length);

The code above will result in the following output: 12 The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters: var txt="Hello world!"; document.write(txt.toUpperCase()); The code above will result in the following output: HELLO WORLD!

JavaScript Date Object

The Date object is used to work with dates and times.

Complete Date Object Reference

For a complete reference of all the properties and methods that can be used with the Date object, go to our complete Date object reference.The reference contains a brief description and examples of use for each property and method!

Create a Date Object

The Date object is used to work with dates and times. Date objects are created with the Date() constructor. There are four ways of instantiating a date: new Date() // current date and time new Date(milliseconds) //milliseconds since 1970/01/01 new Date(dateString) new Date(year, month, day, hours, minutes, seconds, milliseconds) Most parameters above are optional.

Not specifying, causes 0 to be passed in.Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.

Some examples of instantiating a date:

```
 today = new Date()

d1 = new Date("October 13, 1975 11:13:00")
```

d2 = new Date(79,5,24)

d3 = new Date(79,5,24,11,33,0)

JavaScript Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

Complete Boolean Object Reference

For a complete reference of all the properties and methods that can be used with the Boolean object, go to our complete Boolean object reference.The reference contains a brief description and examples of use for each property and method!

Create a Boolean Object

The Boolean object represents two values: "true" or "false".

The following code creates a Boolean object called myBoolean: var myBoolean=new Boolean();

Note: If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false")!

All the following lines of code create Boolean objects with an initial value of false:

var myBoolean=new Boolean();

var myBoolean=new Boolean(0);

var myBoolean=new Boolean(null);

var myBoolean=new Boolean("");

var myBoolean=new Boolean(false);

var myBoolean=new Boolean(NaN);

And all the following lines of code create Boolean objects with an initial value of true:

JAVASCRIPT DEBUGGERS

Firebug

Firebug is a powerful extension for Firefox that has many development and debugging tools including JavaScript debugger and profiler.

Venkman JavaScript Debugger

Venkman JavaScript Debugger (for Mozilla based browsers such as Netscape 7.x, Firefox/Phoenix/Firebird and Mozilla Suite 1.x)

Introduction to Venkman

Using Breakpoints in Venkman

Internet Explorer debugging

Microsoft Script Debugger (for Internet Explorer) The script debugger is from the Windows 98 and NT era. It has been succeeded by the Developer Toolbar

Internet Explorer Developer Toolbar

Microsofts Visual Web Developer Express is Microsofts free version of the Visual Studio IDE. It comes with a JS debugger. For a quick summary of its capabilities see [1]

Internet Explorer 8 has a firebug-like web development tool by default (no add-on) which can be accessed by pressing F12. The web development tool also provides the ability to switch between the IE8 and IE7 rendering engines.

JTF: Javascript Unit Testing Farm

JTF is a collaborative website that enables you to create test cases that will be tested by all browsers. It's the best way to do TDD and to be sure that your code will work well on all browsers.