~~Unit 1.pdf~~

~~Unit 2.pdf~~

~~Unit 3.pdf~~

~~Unit 4.pdf~~

[Unit 1 && 2: Introduction]

Web Technology ~~(CSC-353)~~

~~**Jagdish Bhatta**~~

~~**Central Department of Computer Science & Information Technology**~~
~~**Tribhuvan University**~~

Formatted: Centered, Line spacing: Multiple 0.99 li

Formatted: Line spacing: Multiple 0.99 li

Formatted: Centered, Line spacing: Multiple 0.99 li

Formatted: Line spacing: Multiple 0.99 li

Introduction::

Web technologies related to the interface between web servers and their clients. This information includes markup languages, programming interfaces and languages, and standards for document identification and display. In general web technology incorporates tools and techniques for web development.

Web Development is a broad term for the work involved in developing a web site for World Wide Web. This can include ***web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development*.** However, among web professionals, "web development" usually refers to the main non-design aspects of building web sites: writing markup and coding. Web development can range from developing the simplest static single page of plain text to the most complex web-based internet applications, electronic businesses, or social network services.

Web design is a broad term used to encompass the way that content (usually hypertext or hypermedia) is delivered to an end-user through the World Wide Web, using a web browser or other web-enabled software is displayed. The intent of web design is to create a website—a collection of online content including documents and applications that reside on a web servers. A website may include text, images, sounds and other content, and may be interactive.

For the typical web sites, the basic aspects of design are:

- **The *content*:** the substance, and information on the site should be relevant to the site and should target the area of the public that the website is concerned with.

- **The *usability*:** the site should be user-friendly, with the interface and navigation simple and reliable.

- **The *appearance*:** the graphics and text should include a single style that flows throughout, to show consistency. The style should be professional, appealing and relevant.

- **The *structure*:** of the web site as a whole.

Internet and its Evolution:

Internet is a short form of the technical term ***internetwork***, the result of interconnecting computer networks with special gateways or routers. The Internet is also often referred to as *the Net*. The Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet. Information that travels over the Internet does so via a variety of languages known as protocols. **The Internet is loosely connected compared with the randomized graph.**

The Internet is a globally distributed network comprising many voluntarily interconnected autonomous networks. It operates without a central governing body. However, to maintain interoperability, all technical and policy aspects of the underlying core infrastructure and the principal name spaces are administered by the **Internet Corporation for Assigned Names and Numbers (ICANN).**

The **history of the Internet** starts in the 1950s and 1960s with the development of computers. This began with point-to-point communication between mainframe computers and terminals, expanded to point-to-point connections between computers and then early research into packet switching.

Since the mid-1990s the Internet has had a drastic impact on culture and commerce, including the rise of near instant communication by electronic mail, instant messaging, Voice over Internet Protocol (VoIP) "phone calls", two-way interactive video calls, and the World Wide Web with its discussion forums, blogs, social networking, and online shopping sites. **(Just go through the brief history yourself)**

World Wide Web:

WWW is a system of interlinked hypertext documents accessed via the Internet. The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data. Web services, which use HTTP to allow applications to communicate in order to exchange business logic, use the Web to share information. The Web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video.

The Web is one of the services that runs on the Internet. It is a collection of textual documents and other resources, linked by hyperlinks and URLs, transmitted by web browsers and web servers. The Web is just one of the ways that information can be disseminated over the Internet, so the Web is just a portion of the Internet. In short, the Web can be thought of as an application "running" on the Internet

What is Hypertext?

Hypertext provides the links between different documents and different document types. In a hypertext document, links from one place in the document to another are included with the text. By selecting a link, you are able to jump immediately to another part of the document or even to a different document. In the WWW, links can go not only from one document to another, but from one computer to another

World Wide Consortium:

The **World Wide Web Consortium** (**W3C**) is the main international standards organization for the World Wide Web. W3C was created to ensure compatibility and agreement among industry members in the adoption of new standards. Prior to its creation, incompatible versions of HTML were offered by different vendors, increasing the potential for inconsistency between web pages. The consortium was created to get all those vendors to agree on a set of core principles and components which would be supported by everyone.

Web Page:

A **web page** is a document or information resource that is suitable for the World Wide Web and can be accessed through a web browser and displayed on a monitor or mobile device. This information is usually in HTML or XHTML format, and may provide navigation to other web pages via hypertext links. Web pages frequently subsume other resources such as style sheets, scripts and images into their final presentation.

Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).

Web pages may consist of files of static text and other content stored within the web server's file system (static web pages), or may be constructed by server-side software when they are requested (dynamic web pages). Client-side scripting can make web pages more responsive to user input once on the client browser.

Web Site:

A **website** or simply **site**, is a collection of related web pages containing images, videos or other digital assets. A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a Uniform Resource Locator. All publicly accessible websites collectively constitute the World Wide Web. *Web sites can be static or dynamic.*

Static Website:

A static website is one that has web pages stored on the server in the format that is sent to a client web browser. It is primarily coded in Hypertext Markup Language, HTML.

Simple forms or marketing examples of websites, such as *classic website*, a *five-page website* or a *brochure website* are often static websites, because they present pre-defined, static information to the user. This may include information about a company and its products and services via text, photos, animations, audio/video and interactive menus and navigation.

This type of website usually displays the same information to all visitors. Similar to handing out a printed brochure to customers or clients, a static website will generally provide consistent, standard information for an extended period of time. Although the website owner may make updates periodically, it is a manual process to edit the text, photos and other content and may require basic website design skills and software.

In summary, visitors are not able to control what information they receive via a static website, and must instead settle for whatever content the website owner has decided to offer at that time.

Dynamic Website:

A dynamic website is one that changes or customizes itself frequently and automatically, based on certain criteria.

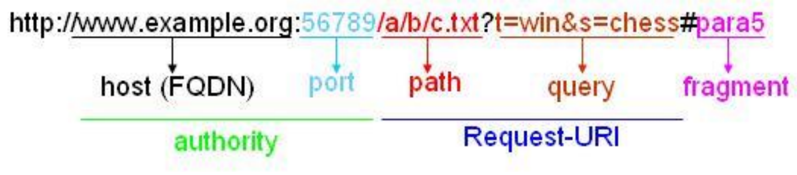
Dynamic websites can have two types of dynamic activity: Code and Content. Dynamic code is invisible or behind the scenes and dynamic content is visible or fully displayed.

The first type is a web page with dynamic code. The code is constructed dynamically on the fly using active programming language instead of plain, static HTML.

The second type is a website with dynamic content displayed in plain view. Variable content is displayed dynamically on the fly based on certain criteria, usually by retrieving content stored in a database

Domain Names, DNS, and URLs:

- IP addresses are not convenient for users to remember easily. So an IP address can be represented by a natural language convention called a **domain name**
- **Domain name system (DNS)** translates domain names into IP addresses. DNS is the "phone book" for the Internet, it maps between host names and IP addresses.
- A **uniform resource locator (URL)**, which is the address used by a Web browser to identify the location of content on the Web, also uses a domain name as part of the URL.
- **Syntax: scheme: scheme-depend-part.** Example: In http://www.example.com/, the scheme is http.



HTTP:

- HTTP is based on the request-response communication model:
  - Client sends a request
  - Server sends a response
  - HTTP is a stateless protocol: where the protocol does not require the server to remember anything about the client between requests.
- Normally implemented over a TCP connection (80 is standard port number for HTTP)
- The following is the typical browser-server interaction using HTTP:
  1. User enters Web address in browser
  2. Browser uses DNS to locate IP address
  3. Browser opens TCP connection to server
  4. Browser sends HTTP request over connection
  5. Server sends HTTP response to browser over connection
  6. Browser displays body of response in the client area of the browser window

Client/Server Computing:

- A model of computing in which powerful personal computers are connected in a network together with one or more servers
- **Client** is a powerful personal computer that is part of a network; service requester
- **Server** is a networked computer dedicated to common functions that the client computers on the network need; service provider
- Web is based on client/server technology. Web servers are included as part of a larger package of internet and intranet related programs for serving e-mail, downloading requests for FTP files and building and publishing web pages. Typically the e-commerce customer is the client and the business is the server. In the client/ server model single machine can be both client and the server The client/ server model utilises a database server in which RDBMS user queries can be answered directly by the server.
- The client/ server architecture reduces network traffic by providing a query response to the user rather than transferring total files. The client/ server model improves multi-user updating through a graphical user interface (GUI) front end to the shared database. In client/ server architectures client and server typically communicate through statements made in structured query language (SQL).

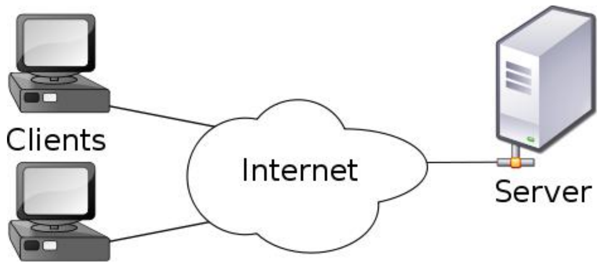


Fig: Client/ Server Model

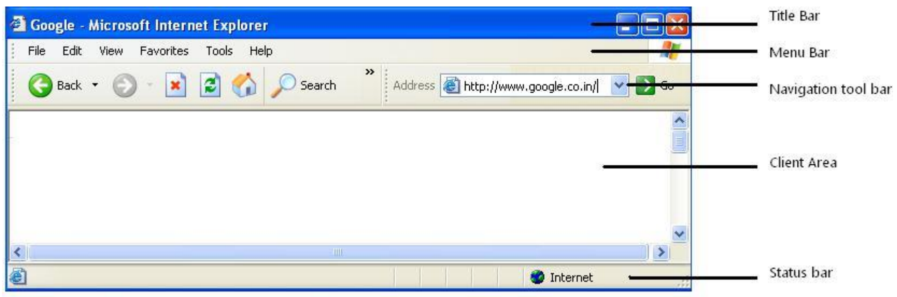
Web Clients:

It typically refers to the Web browser in the user's machine. It is a software application for retrieving, presenting, and traversing information resources on the web server. It is used to create a HTTP request message and for processing the HTTP response message.

User agent: Any web client is designed to directly support user access to web servers is known as user agent. Web browsers can run on desktop or laptop computers. Some of the browsers are: Internet Explorer, Mozilla, FireFox, Chrome, Safari, Opera, Netscape Navigator.

Web Browsers:

Browsers are software programs that allow you to search and view the many different kinds of information that's available on the World Wide Web. The information could be web sites, video or audio information.


Google - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Address http://www.google.co.in/
Go
Internet
Title Bar
Menu Bar
Navigation tool bar
Client Area
Status bar

Status Bar: You will find the status bar at the very bottom of your browser window. It basically tells you what you are doing at the moment. Mainly, it shows you load speed and the URL address of whatever your mouse is hovering over.

Title Bar: You will find this bar at the absolute top of your browser and in will be the colour blue for the major browsers. The purpose of the Title bar is to display the title of the web page that you are currently viewing.

Menu Bar: The menu bar contains a set of dropdown menus

Navigational Tool: A bar contains standard push button controls that allow the user to return to a previously viewed page, to reverse and refresh the page, to display the home page and to print the page etc.

Toolbar Icons: You will find the Toolbar directly under the Title Bar. The Toolbar is where you will find the back button, home button and the refresh button etc.

Client Area: It is a display window which is the space in which you view the website.

Scroll Bars: The Scroll bars, usually located to the right of the Display Window, allows you to "scroll" (move down or up the web page) so you can view information that is below or above what is currently in the Display Window.

Web Servers:

Basic functionality:

- It receives HTTP request via TCP
- It maps Host header to specific virtual host (one of many host names sharing an IP address)
- It maps Request-URI to specific resource associated with the virtual host
  - File: Return file in HTTP response
  - Program: Run program and return output in HTTP response
- It maps type of resource to appropriate MIME type and use to set Content-Type header in HTTP response
- It Logs information about the request and response
- All e-commerce site require basic Web server software to answer requests from customers like ;
  - **Apache**
    - Leading Web server software (47% of market)
    - Works with UNIX, Linux , Windows OSs
  - **Microsoft's Internet Information Server (IIS)**
    - Second major Web server software (25% of market)
    - Windows-based

Client-Side Scripting:

- Client-side scripting generally refers to writing the class of computer programs (scripts) on the web that are executed at *client-side*, by the user's web browser, instead of *server-side* (on the web server). Usually scripts are embedded in the HTML page itself.

- JavaScript , VBScript, Jscript, Java Applets etc. are the examples of client side scripting technologies. JavaScript is probably the most widely used client-side scripting language.
- Client-side scripts have greater access to the information and functions available on the user's browser, whereas server-side scripts have greater access to the information and functions available on the server. Upon request, the necessary files are sent to the user's computer by the web server (or servers) on which they reside. The user's web browser executes the script, then displays the document, including any visible output from the script.
- Client-side scripts may also contain instructions for the browser to follow in response to certain user actions, (e.g., clicking a button). Often, these instructions can be followed without further communication with the server.

Server-Side Scripting:

- Includes writing the applications executed by the server at run-time to process client input or generate document in response to client request. So server side script consists the directives embedded in Web page for *server* to process before passing page to requestor.
- It is usually used to provide interactive web sites that interface to databases or other data stores.
- 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.
- PHP, JSP, ASP…. etc, are the server side scripting technologies.

Web 2.0:

The term **Web 2.0** is associated with web applications that facilitate participatory *information sharing, interoperability, user-centered design, and collaboration* on the World Wide Web. A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community, in contrast to websites where users are limited to the passive viewing of content that was created for them. Examples of Web 2.0 include *social networking sites, blogs, wikis, video sharing sites, hosted services, web applications.*

I think following portion you have studied in Data Communication (So Self Study)

SMTP:

Simple Mail Transfer Protocol (**SMTP**) is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks.

POP:

In computing, the **Post Office Protocol** (**POP**) is an application-layer Internet standard protocol used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection.

HTML

HTML stands for **hypertext markup language**. It is not a programming language. A markup language specifies the *layout and style* of a document. A markup language consists of a set of **markup tags**. HTML uses markup tags to describe web pages. HTML tags are keywords surrounded by **angle brackets** like <html>. Most HTML tags normally come in pairs like <b> and </b>. The first tag is called the **start tag** (or **opening tag**) and the second tag is called the **end tag** (or **closing tag**). HTML documents describe Web pages. HTML documents contain HTML tags and plain text. HTML documents are also called Web pages. A web browser read HTML documents and displays them as Web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. A simple HTML document is given below:

```
<html>
	<head>
		<title>This is my first web page</title>
	</head>
	<body>
		<h1>My first heading</h1>
		<p>My first paragraph</p>
	</body>
</html>
```

Save this page with **.html** or **.htm** extension. However, it is good practice to use **.htm** extension.

HTML Elements

HTML documents are defined by HTML elements. An HTML element is everything from the start tag to the end tag. For example, <p>My first paragraph</p>. An HTML element consists of start tag, end tag, and element content. The element content is everything between the start tag and end tag. Empty elements are closed in the start tag. Most HTML elements can have attributes. For example, **src** attribute of **img** tag.

HTML Attributes

Attributes provide additional information about HTML elements. Attributes are always specified in the start tag. Attributes come in name/value pair like name = "value". For example, HTML links are defined with **<a>** tag and the link address is provided as an attribute **href** like **<a href = "http://www.tu.edu.np">cdcsit</a>**.
Note: Always quote attribute values and use lowercase attributes.

HTML Headings

HTML headings are defined with the <h1> to <h6> tags. <h1> displays largest text and <h6> smallest. For example, <h1>My first heading</h1>.

HTML Paragraphs

HTML paragraphs are defined with <p> tag. For example, <p>My first paragraph</p>.

HTML Rules (Lines)

We use <hr /> tag to create horizontal line.

HTML Comments

We use comments to make our HTML code more readable and understandable. Comments are ignored by the browser and are not displayed. Comments are written between <!-- and -->. For example, <!-- This is a comment -->.

HTML Line Breaks

If you want a new line (line break) without starting a new paragraph, use <br /> tag.

HTML Formatting Tags

We use different tags for formatting output. For example, <b> is used for bold and <i> is used for italic text. Some other tags are <big>, <small>, <sup>, <sub> etc.

HTML Styles

It is a new HTML attribute. It introduces CSS to HTML. The purpose of style attribute is to provide a common way to style all HTML elements. For example, <body style = "background-color:yellow">, <p style = "font-family:courier new; color:red; font-size:20px">, <h1 style = "text-align:center"> etc.

HTML Links

A link is the address to a resource on the web. HTML links are defined using an anchor tag (**<a>**). We can use this tag to point to a resource (an HTML page, an image, a sound file, a movie etc.) and an address inside a document.

We can use **href** attribute to define the link address. For example, <a href = "http://www.cdcsit.tu.edu.np">cdcsit</a>.

We can use **target** attribute to define where the linked document will be opened. For example, <a href = "http://www.cdcsit.tu.edu.np" target = "_blank">cdcsit</a> will open the document in a new window.

We can use **name** attribute to define a named anchor inside a HTML document. Named anchor are invisible to the reader. For example, <a name = "label">Any content</a> defines a named anchor and we use the syntax <a href = "#label">Any content</a> to link to the named anchor.

We can also use named anchor to link to some content within another document. For example, <a href="http://www.w3schools.com/html_tutorial.htm#tips">Jump to the Useful Tips section</a>.

HTML Images

HTML images are defined with **<img>** tag. To display an image on a page, you need to use the **src** attribute. We can also use **width** and **height** attributes with img tag. For example, <img src = “photo1.jpg” width = “104” height = “142” />.

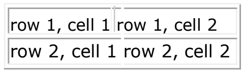
We can use alt attribute to define an alternate text for an image. For example, <img src = “photo1.jpg” width = “104” height = “142” alt = “My best poto”/>. The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image. It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

HTML Tables

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc. For example,

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

Output:



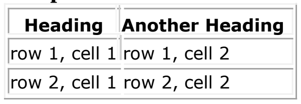
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

We use border attribute to display table with border as shown in the above example.
Headings in a table are defined with **<th>** tag. For example,

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
```

```
</tr>
</table>
```

Output:



Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

We can use <caption> tag inside a <table> to display caption for a table. We can define table cells that span more than one row or one column using **colspan** and **rowspan** attributes respectively. For example, <td colspan = “2”>Data</td>. We can use **cellpadding** and **cellspacing** attributes to create white space between the cell content and its borders, and to increase the distance between cells respectively. For example, <table border="1" cellpadding="10"> and <table border="1" cellspacing="10">. We can use align attribute to align the contents of a cell. For example, <td align = “left”>Data</td>.

HTML Lists

HTML supports *ordered*, *unordered* and *definition lists*. Ordered lists items are marked with numbers, letter etc. We use **<ol>** tag for ordered list and each list item starts with **<li>** tag. For example,

```
<ol type="A">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ol>
```

Output:

A. Apples
B. Bananas
C. Lemons
D. Oranges

If we do not use type attribute, items are marked with numbers. We use **type = “a”** for lowercase letters list, **type = “I”** for roman numbers list, and **type = “i”** for lowercase numbers list.

Unordered lists items are marked with bullets. We use **<ul>** tag for unordered list and each list item starts with **<li>** tag. For example,

```
<ul type="disc">
  <li>Apples</li>
  <li>Bananas</li>
  <li>Lemons</li>
  <li>Oranges</li>
</ul>
```

Output:

- Apples
- Bananas
- Lemons
- Oranges

If we do not use type attribute, items are marked with discs. We use **type = “circle”** for circle bullets list, and **type = “square”** for square bullets list.

Definition list is the list of items with a description of each item. We use **<dl>** tag for definition list, **<dt>** for definition term, and **<dd>** for definition description. For example,

```
<dl>
  <dt>Coffee</dt>
  <dd>Black hot drink</dd>
  <dt>Milk</dt>
  <dd>White cold drink</dd>
</dl>
```

Output:

```
   Coffee
         Black hot drink
   Milk
         White cold drink
```

<u>HTML Forms</u>

Forms are used to select different types of user input. A form is an area that contains different form elements (like text fields, text area fields, drop-down menus, radio buttons, checkboxes etc.). Form elements are elements that allow the user to enter information in a form. A form is defined with the **<form>** tag. For example,

```
<form>
    input elements
</form>
```

The most commonly used form tag is **<input>** tag. The type of input is specified with the **type attribute** within the <input> tag. For example,

```
<form>
  First name:
  <input type="text" name="firstname" />
  <br />
  Last name:
  <input type="text" name="lastname" />
</form>
```

Output:

```
First name: [                    ]
Last name:  [                    ]
```

Another input type is **radio button**. Radio buttons are used when you want the user to select one of a limited number of choices. For example,

```
<form>
<input type="radio" name="sex" value="male" /> Male
<br />
<input type="radio" name="sex" value="female" /> Female
</form>
```

Output:

◉ Male

○ Female

Another input type is **checkboxes**. Checkboxes are used when you want to select one or more options of a limited number of choices. For example,

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike"
/> <br />
I have a car:
<input type="checkbox" name="vehicle" value="Car" />
<br />
I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane" />
</form>
```

Output:

I have a bike: ☐

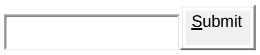
I have a car: ☐

I have an airplane: ☐

Another input type is **submit button**. When the user clicks on the "Submit" button, the content of the form is sent to the server. The form's **action attribute** defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input. For example,

```
<form name="input" action=" submit.php" method="get">
  Username:
  <input type="text" name="user" />
  <input type="submit" value="Submit" />
</form>
```

Output:

Username: 

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "submit.php". The page will show you the received input.

***Note:** You can use other different form elements as well.*

The **method** attribute of <form> tag specifies how to send form-data (the form-data is sent to the page specified in the action attribute). We can use **“get”** and **“post”** as values of method attribute. When we use get, form-data can be sent as URL variables and when we use post, form-data are sent as HTTP post.

Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- There is a limit to how much data you can place in a URL (varies between browsers), therefore, you cannot be sure that all of the form-data will be correctly transferred
- Never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

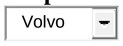
Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

We can create a simple drop-down box on an HTML page. A drop-down box is a selectable list. See code below:

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

Output:



HTML Color

HTML colors are displayed using RED, GREEN, and BLUE light. Colors are defined using hexadecimal (hex) notation for combination of red, green, and blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00) and the highest values is 255 (hex FF). We can use HEX (e.g. **#2000FF**) as well as RGB (e.g. **rgb(32, 0, 255)**) values to define different colors.

The combination of Red, Green and Blue values from 0 to 255 gives a total of more than 16 million different colors to play with (256 x 256 x 256).

We can also use color names instead of hex and rgb values. The World Wide Web Consortium (W3C) has listed 16 valid color names for HTML and CSS: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. Some examples are given below:

```
<body style = "background:rgb(12, 32, 255)">
<body style = "background:#0008FF>
<body style = "background:red">
```

HTML Frames

We can use frames to display more than one web page in the same browser window. Each HTML document is called a frame, and each frame is independent of the others. The disadvantages of using frames are:

- The web developer must keep track of more HTML documents
- It is difficult to print the entire page

We use **<frameset>** tag to define how to divide the window into frames. Each frameset defines a set of rows or columns. Within frameset, we use **<frame>** tag to define what HTML document to put into each frame.

If a frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, you can add noresize="noresize" to the <frame> tag. Add the <noframes> tag for browsers that do not support frames.

Important: You cannot use the <body></body> tags together with the <frameset></frameset> tags. However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags.

Example 1:

```
<frameset cols="25%,50%,25%">
  <frame src="frame_a.htm" noresize="noresize"/>
  <frame src="frame_b.htm"/> <frame
  src="frame_c.htm"/>
<noframes>
<body>Your browser does not handle frames!</body>
</noframes>
</frameset>
```

Example 2:

```
<frameset rows="25%,50%,25%">
```

```
  <frame src="frame_a.htm"/>
  <frame src="frame_b.htm"/>
  <frame src="frame_c.htm"/>
</frameset>
```

Example 3: Mixed Frameset

```
<frameset rows="50%,50%">
  <frame src="frame_a.htm"/>
  <frameset cols="25%,75%">
    <frame src="frame_b.htm"/>
    <frame src="frame_c.htm"/>
  </frameset>
</frameset>
```

HTML Fonts

The **<font>** tag in HTML is deprecated. It is supposed to be removed in a future version of HTML. For example,

```
<p>
  <font size="2" face="Verdana" color = "red">
    This is a paragraph.
  </font>
</p>
```

HTML Character Entities

Character entities are replaced with reserved characters. A character entity looks *&entity_name* **OR** *&#entity_number*. Some commonly used character entities are:

Result	**Description**	**Entity Name**	**Entity Number**
	non-breaking space	&nbsp;	&#160;
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	Ampersand	&amp;	&#38;
¢	Cent	&cent;	&#162;
£	Pound	&pound;	&#163;
¥	Yen	&yen;	&#165;
€	Euro	&euro;	&#8364;
©	Copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

HTML Head

The head element contains general information, also called meta-information, about a document. The elements inside the head element should not be displayed by a browser. According to the HTML standard, only a few tags are legal inside the head section. These are: <base>, <link>, <meta>, <title>, <style>, and <script>.

You must use this element and it should be used just once. It must start immediately after the opening <html> tag and end directly before the opening <body> tag.

HTML Meta

HTML includes a meta element that goes inside the head element. The purpose of the meta element is to provide meta-information about the document. Meta elements are purely used for search engine's use and to provide some additional information about your pages. We use three attributes (**name**, **content**, and **http-equiv**) with **<meta>** tag.

We use **name = "keywords"** to provide information for a search engine. If the keywords you have chosen are the same as the ones they have put in, you come up in the search engine's result pages. For example,

<meta name="keywords" content="HTML, DHTML, CSS, XML, XHTML, JavaScript" />

We use **name = "description"** to define a description of your page. It is sort summary of the content of the page. Depending on the search engine, this will be displayed along with the title of your page in an index. For example,

<meta name="description" content="Free Web tutorials on HTML, CSS, XML, and XHTML" />

We use **name = "generator"** to define a description for the program you used to write your pages. For example,

<meta name="generator" content="Homesite 4.5" />

We use **name = "author"** and **name = "copyright"** for author and copyright details. For example,

<meta name="author" content="W3schools" />

<meta name="copyright" content="W3schools 2005" />

We use **name = "expires"** to give the browsers a data, after which the page is deleted from the browsers cache, and must be downloaded again. This is useful if you want to make sure your visitors are reading the most current version of a page. For example,

<meta name="expires" content="13 July 2008" />

We use **http-equiv = “expires”** to refresh itself to the most current version or change to another location (page) entirely after some time. This is useful if you’ve moved a page to a new url and want any visitors to the old address to be quietly sent to the new location. For example,

```
<meta http-equiv = "refresh" content="5;url=http://www.tu.edu.np" />
```

Here, the number is the number of seconds to wait before changing to the new page. Setting it to 0 results in an instant redirect.

HTML Div

The **<div>** element defines logical divisions within the document. When you use a <div> element, you are indicating that the enclosed content is specific section of the page and you can format the section with CSS (Cascading Style Sheet). For example,

```
<div style="background-color:orange;text-align:center">
  <p>Navigation section</p>
</div>
<div style="border:1px solid black">
  <p>Content section</p>
</div>
```

HTML Events

Events trigger actions in the browser, like starting a JavaScript when a user clicks on an HTML element. Below is a list of attributes that can be inserted to HTML tags to define event actions. These HTML events are given below:

Window Events (**Only valid in body and frameset elements)**

Attribute	**Value**	**Description**
Onload	*Script*	Script to be run when a document loads
Onunload	*Script*	Script to be run when a document unloads

Form Element Events **(Only valid in form elements)**

Attribute	**Value**	**Description**
Onchange	*Script*	Script to be run when the element changes
Onsubmit	*Script*	Script to be run when the form is submitted
Onreset	*Script*	Script to be run when the form is reset
Onselect	*script*	Script to be run when the element is selected
Onblur	*script*	Script to be run when the element loses focus
Onfocus	*script*	Script to be run when the element gets focus

Keyboard Events **(Not valid in *base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style,* and *title* elements)**

Attribute	**Value**	**Description**
Onkeydown	*script*	What to do when key is pressed
Onkeypress	*script*	What to do when key is pressed and released
Onkeyup	*script*	What to do when key is released

Mouse Events **(Not valid in *base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style, title* elements)**

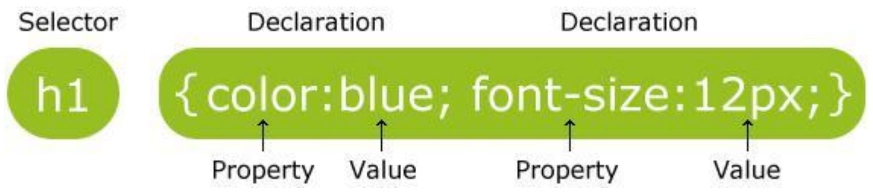
Attribute	**Value**	**Description**
Onclick	*script*	What to do on a mouse click
Ondblclick	*script*	What to do on a mouse double-click
Onmousedown	*script*	What to do when mouse button is pressed
Onmousemove	*script*	What to do when mouse pointer moves
Onmouseout	*Script*	What to do when mouse pointer moves out of an element
Onmouseover	*Script*	What to do when mouse pointer moves over an element
Onmouseup	*script*	What to do when mouse button is released

CSS (Cascading Style Sheets)

CSS stands for cascading style sheets. It was first developed in 1997, as a way for Web developers to define the **look and feel** of their Web pages. It was intended to allow developers to separate content from design and layout so that HTML could perform more of the function without worry about the design and layout. It is used to separate style from content.

Syntax

A CSS rule has two main parts: a *selector* and one or more *declarations*. **Selector** is normally the HTML element you want to style and each **declaration** consists of a *property* and *value*. The **property** is the style attribute we want to use and each property has a **value** associated with it.



Example:

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

Inserting CSS

We can use style sheets in three different ways in out HTML document. There are **external style sheet**, **internal style sheet** and **inline style**.

External Style Sheet

If we want to apply the same style to many pages, we use external style sheet. With an external style sheet, you can change the look of an entire Web site by changing one style sheet file. Each page must link to the style sheet using the **<link>** tag. The <link> tag goes inside the head section. For example,

```
<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. Your style sheet should be saved with a **.css** extension. An example of a style sheet file is shown below:

```
hr {color:sienna;}
p {margin-left:20px;} /*Note: Do not leave space between property value and
units*/ body {background-image:url("images/back40.gif");}
```

Internal Style Sheet

If you want a unique style to a single document, an internal style sheet should be used. You define internal styles in the head section of an HTML page, by using the **<style>** tag. For example,

```
<head>
<style type="text/css">
hr {color:red;}
p {margin-left:20px;}
body {background-image:url("images/back40.gif");}
</style>
</head>
```

Inline Styles

If you want a unique style to a single element, an inline style sheet should be used. An inline style loses many of the advantages of style sheets by mixing content with presentation. To use inline styles you use the **style attribute** in the relevant tag. The style attribute can contain any CSS property. For example,

```
<p style="color:yellow;margin-left:20px">This is a paragraph.</p>
```

Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers. A CSS comment begins with "/*", and ends with "*/".

Id and Class Selectors

The **id** selector is used to specify a style for a single, unique element. The id selector uses id attribute of the HTML element and is defined with **"#"**. For example,

```
<head>
<style type="text/css">
#para1
{
text-align:center;
color:red;
}
</style>
</head>
<body>
<p id="para1">Hello World!</p>
<p>This paragraph is not affected by the style.</p>
</body>
```

The **class** selector is used to specify a style for a group of elements. Unlike the id selector, the class selector is most often used on several elements. This allows you to set a particular style for any HTML elements with the same class. The class selector uses the HTML class attribute, and is defined with a ".". For example,

```
<head>
<style type="text/css">
.center
{
text-align:center;
}
</style>
</head>
<body>
<h1 class="center">Center-aligned heading</h1>
<p class="center">Center-aligned paragraph.</p>
</body>
```

You can also specify that only specific HTML elements should be affected by a class. For example,

```
<head>
<style type="text/css">
p.center
{
text-align:center;
}
</style>
</head>
<body>
<h1 class="center">This heading will not be affected</h1>
<p class="center">This paragraph will be center-aligned.</p>
</body>
```

Multiple Styles Will Cascade into One

Styles can be specified:

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

Tip: Even multiple external style sheets can be referenced inside a single HTML document.

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 four has the highest priority:

1. Browser default
2. External style sheet
3. Internal style sheet (in the head section)
4. 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).

Note: If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

CSS Background

Background properties are used to define the background effects of an HTML element. CSS properties used to define background effects are: **background-color**, **background-image**, **background-repeat**, **background-attachment**, and **background-position**.

Background Image

The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.

The background image for a page can be set like this:

```
body {background-image:url('paper.gif');}
```

Background Image - Repeat Horizontally or Vertically

By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example

```
body
{
```

```
background-image:url('gradient2.png');
}
```

If the image is repeated only horizontally (repeat-x), the background will look better:

Example

```
body
{
background-image:url('gradient2.png');
background-repeat:repeat-x;
}
```

Background Image - Set position and no-repeat

When using a background image, use an image that does not disturb the text. Showing the image only once is specified by the background-repeat property:

Example

```
body
{
background-image:url('img_tree.png');
background-repeat:no-repeat;
}
```

In the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much.

The position of the image is specified by the background-position property:

Example

```
body
{
background-image:url('img_tree.png');
background-repeat:no-repeat;
background-position:right top;
}
```

Shorthand Property

To shorten the code, it is also possible to specify all the properties in one single property. This is called a shorthand property. The shorthand property for background is simply "background". When using the shorthand property the order of the property values are: background-color, background-image, background-repeat, background-attachment, and background-position. For example, body {background:#ffffff url('img_tree.png') no-repeat right top;}

Grouping Selectors

In style sheets there are often elements with the same style.

```
h1
{
color:green;
}
h2
{
color:green;
}
p
{
color:green;
}
```

To minimize the code, you can 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
{
color:green;
}
```

CSS Display and Visibility

The display property specifies if/how an element is displayed, and the visibility property specifies if an element should be visible or hidden.

Hiding an Element - display:none or visibility:hidden

Hiding an element can be done by setting the display property to "none" or the visibility property to "hidden". However, notice that these two methods produce different results:

visibility: hidden hides an element, but it will still take up the same space as before. The element will be hidden, but still affect the layout.

Example

```
h1.hidden {visibility:hidden;}
```

display: none hides an element, and it will not take up any space. The element will be hidden, and the page will be displayed as the element is not there:

Example

h1.hidden {display:none;}

CSS Display - Block and Inline Elements

A block element is an element that takes up the full width available, and has a line break before and after it.

Examples of block elements:

- <h1>
- <p>
- <div>

An inline element only takes up as much width as necessary, and does not force line breaks.

Examples of inline elements:

- <span>
- <a>

Changing How an Element is Displayed

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow web standards.

The following example displays list items as inline elements:

Example

li {display:inline;}

The following example displays span elements as block elements:

Example

span {display:block;}

Changing the display type of an element changes only how the element is displayed, NOT what kind of element it is. For example: An inline element set to display:block is not allowed to have a block element nested inside of it.

CSS Padding Property:

Example

Set the padding of a p element:

```
p
{
padding:2cm 4cm 3cm 4cm;
}
```

Definition and Usage

The padding shorthand property sets all the padding properties in one declaration. This property can have from one to four values.

Examples:

- **padding:10px 5px 15px 20px;**
  - top padding is 10px
  - right padding is 5px
  - bottom padding is 15px
  - left padding is 20px
- **padding:10px 5px 15px;**
  - top padding is 10px
  - right and left padding are 5px
  - bottom padding is 15px
- **padding:10px 5px;**
  - top and bottom padding are 10px
  - right and left padding are 5px
- **padding:10px;**
  - all four paddings are 10px

Note: Negative values are not allowed.

CSS Float:

With CSS float, an element can be pushed to the left or right, allowing other elements to wrap around it. Float is very often used for images, but it is also useful when working with layouts.

How Elements Float

Elements are floated horizontally; this means that an element can only be floated left or right, not up or down. A floated element will move as far to the left or right as it can. Usually this means all the way to the left or right of the containing element. The elements after the floating element will flow around it. The elements before the floating element will not be affected. If an image is floated to the right, a following text flows around it, to the left.

Example

```
img
{
float:right;
}
```

Floating Elements Next to Each Other

If you place several floating elements after each other, they will float next to each other if there is room. Here we have made an image gallery using the float property:

Example

```
.thumbnail
{
float:left;
width:110px;
height:90px;
margin:5px;
}
```

Turning off Float - Using Clear

Elements after the floating element will flow around it. To avoid this, use the clear property.

The clear property specifies which sides of an element other floating elements are not allowed.

Add a text line into the image gallery, using the clear property:

Example

```
.text_line
{
clear:both;
}
```

JavaScript

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- 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

Are Java and JavaScript the same?

NO! Java and JavaScript are two completely different languages in both concept and design! Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

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

The Real Name is ECMAScript

- JavaScript's official name is ECMAScript.
- ECMAScript is developed and maintained by the ECMA organization.
- ECMA-262 is the official JavaScript standard.

- The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.
- The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.
- The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998.
- The development of the standard is still in progress.
- The HTML <script> tag is used to insert a JavaScript into an HTML page.

The example below shows how to use JavaScript to write text on a web page:

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

The example below shows how to add HTML tags to the JavaScript:

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

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>
```

The **document.write** command is a standard JavaScript command for writing output to a page.

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

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

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello World!") command as pure text, and just write the entire line on the page.

How to Handle Simple Browsers

Browsers that do not support JavaScript, will display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.

Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement, like this:

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

The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

JavaScripts can be put in the body and in the head sections of an HTML page.

Where to Put the JavaScript

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, or at a later event, such as when a user clicks a button. When this is the case we put the script inside a function, you will learn about functions in a later chapter.

Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, are placed in functions. Put your functions in the head section, this way they are all in one place, and they do not interfere with page content.

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```

Scripts in <body>

If you don't want your script to be placed inside a function, or if your script should write page content, it should be placed in the body section.

Example

```
<html>
<head>
</head>

<body>
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>
</html>
```

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

Example

```
<html>
```

```
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>

<body onload="message()">
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>

</html>
```

Using an External JavaScript

If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.

Save the external JavaScript file with a .js file extension.

Note: The external script cannot contain the <script></script> tags!

To use the external script, point to the .js file in the "src" attribute of the <script> tag:

```
<html>
<head>
<script type="text/javascript" src="xxx.js"></script>
</head>
<body>
</body>
</html>
```

Note: Remember to place the script exactly where you normally would write the script!
JavaScript is a sequence of statements to be executed by the browser.

JavaScript is Case Sensitive

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

JavaScript Statements

A JavaScript statement is a command to a 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. Following example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Blocks

JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and ends with a right curly bracket }. The purpose of a block is to make the sequence of statements execute together. Following example will write a heading and two paragraphs to a web page:

Example

```
<script type="text/javascript">
{
document.write("<h1>This is a heading</h1>");
document.write("<p>This is a paragraph.</p>");
document.write("<p>This is another paragraph tested at
pmc.</p>"); }
</script>
```

The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements together in a function or in a condition (where a group of statements should be executed if a condition is met).

JavaScript Variables
As with algebra, JavaScript variables are used to hold values or expressions. A variable can have a short name, like x, or a more descriptive name, like carname.

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

Note: Because JavaScript is case-sensitive, variable names are case-sensitive.

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 are empty (they have no values yet). However, you can also assign values to the variables when you declare them:

```
var x=5;
var carname="Volvo";
```

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.

Note: *When you assign a text value to a variable, use quotes around the value.*

Assigning Values to Undeclared JavaScript Variables
If you assign values to variables that have 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.

JavaScript Arithmetic

As with algebra, you can do arithmetic operations with JavaScript variables:

y=x-5;
z=y+5;

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x=5**, the table below explains the comparison operators:

Operator	Description	Example
= =	is equal to	x==8 is false
= = =	is exactly equal to (value and type)	x===5 is true x==="5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Logical Operators

Logical operators are used to determine the logic between variables or values. Given that **x=6 and y=3**, the table below explains the logical operators:

Operator	Description	Example
&&	And	(x < 10 && y > 1) is true
||	Or	(x==5 || y==5) is false
!	Not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

variablename=(condition)?value1:value2

~~Jagdish Bhatta~~SA

Example

greeting=(visitor=="PRES")?"Dear President ":"Dear ";

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear".

Flow Control

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

Looping Structures

- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
- In JavaScript, there are two different kind of loops:
- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

- The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
```

code to be executed

}

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

JavaScript While Loop

- The while loop loops through a block of code while a specified condition is true.

Syntax

- while (var<=endvalue)
  {
  code to be executed
  }

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
  {
```

```
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
</script>
</body>
</html>
```

Javascript do while loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
  {
  code to be executed
  }
while (var<=endvalue);
```

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
while (i<=5);
</script>
</body>
</html>
```

The Break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
    break;
    }
  document.write("The number is " + i);
  document.write("<br />");
  }
</script>
</body>
</html>
```

Javascript for ….. in statement

The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

```
for (variable in object)
 {
 code to be executed
 }
```

Note: The code in the body of the for...in loop is executed once for each element/property.

Example: Use the for...in statement to loop through an array:

```
<html>
<body>
<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
for (x in mycars)
  {
  document.write(mycars[x] + "<br />");
  }
</script>
</body>
</html>
```

Functions

- A function is simply a block of code with a name, which allows the block of code to be called by other components in the scripts to perform certain tasks.
- Functions can also accept parameters that they use complete their task.
- JavaScript actually comes with a number of built-in functions to accomplish a variety of tasks.

Creating Custom Functions

- In addition to using the functions provided by javaScript, you can also create and use your own functions.
- General syntax for creating a function in JavaScript is as follows:

```
function name_of_function(argument1,argument2,…arguments)
{
…………………………………………
//Block of Code
…………………………………………
}
```

Calling functions

- There are two common ways to call a function: From an *event handler* and from another function.
- Calling a function is simple. You have to specify its name followed by the pair of parenthesis.

```
<SCRIPT TYPE="TEXT/JAVASCRIPT">
      name_of_function(argument1,argument2,…arguments)
 </SCRIPT>
```

Example

```
<html>
          <head> <title>PMC</title>
                    <Script Language="JavaScript">
                    function welcomeMessage()
                              {
                                document.write("Welcome to Patan Campus!");
                              }
                     </Script>
          </head>
```

```
        <body>
            <h1>Patan Multiple Campus CSIT</h1>
             <h3>Testing the function in PMC</h3>
            <Script Language="JavaScript">
                welcomeMessage();
            </Script>
        </body>
    </html>
```

<u>Popup Boxes</u>

Alert Box:

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
    alert("sometext");
```

Example

```
    <html>
    <head>
    <script type="text/javascript">
    function show_alert()
    {
    alert("I am an alert box!");
    }
    </script>
    </head>
    <body>
    <input type="button" onclick="show_alert()" value="Show alert box" />
    </body>
    </html>
```

Confirmation Box:

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  document.write("You pressed OK!");
  }
else
  {
  document.write("You pressed Cancel!");
  }
}
</script>
</head>
<body>
<input  type="button"  onclick="show_confirm()"  value="Show  confirm  box"  />
</body>
</html>
```

Prompt Box:
A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

Example

```
<html>
<head>
<script type="text/javascript">
var name=prompt("Please enter your name",“Rajendra");
</script>
</head>
<body>
<script type="text/javascript">
document.write("Hello "+name + "You have worked will with variables");
</script>
</body>
</html>
```

JavaScript objects

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types. An object is just a special kind of data. An object has properties and methods.

Properties: Properties are the values associated with an object.
Methods: Methods are the actions that can be performed on objects.

Array Object in JavaScript

An array is a special variable, which can hold more than one value, at a time. An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own ID so that it can be easily accessed. The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are three ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1.) Conventional array: The classic conventional array looks like the following:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

You can expand and contract the array as desired, by adding new array elements. Note that like in most other programming languages, the first array element should have an index number of 0.

With a conventional array, you have the option of presetting the array's length when defining it, by passing in a numeric integer into the Array() constructor:

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2.) Condensed array: The second way of defining an array is called a condensed array, and differs from the above simply in that it allows you to combine the array and array elements definitions into one step:

```
var myCars=new Array("Saab","Volvo","BMW");
```

This is convinient if you know all the array element values in advance.

3.) Literal array: Finally, we arrive at literal arrays. Introduced in JavaScript1.2 and support by all modern browsers (IE/NS4+), literal arrays sacrafice intuitiveness somewhat in exchange for tremendous robustness. The syntax looks like:

```
var myCars=["Saab","Volvo","BMW"];
```

Literal array with 5 elements (middle 3 with undefined values).

```
var mystudents=["giri", , , "tulsi"]
```

As you can see, enclose all the array elements within an outter square bracket ([ ]), each separated by a comma (,). To create array elements with an initial undefined value just enter a comma (,) as shown in the second example above.

Literal arrays really shine when it comes to defining multi-dimensional arrays. It is as easy as adding containing brackets ([ ]) within the outermost bracket. For example:

```
var myarray=[["Subash", "Pandey", "Gautam"], Kalanki, Sanepa]
```

Here the first array element is actually a two dimensional array in itself containing various cities names. To access LA, then, you'd use the syntax:

```
myarray[0][1] //returns "Pandey"
```

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Accessing the Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0. In above initialized array, the code line *document.write(myCars[0]);* will result in the following output: Saab

To modify a value in an existing array, just add a new value to the array with a specified index number:

```
myCars[0]="Opel";
```

Now, the following code line:

document.write(myCars[0]); will result in the following output: Opel.

Some methods associated with array

- **concat( ):** Joins two or more arrays, and returns a copy of the joined arrays
- **join( ):** Joins all elements of an array into a string
- **pop( ):** Removes the last element of an array, and returns that element
- **push( ):** Adds new elements to the end of an array, and returns the new length
- **reverse( ):** Reverses the order of the elements in an array
- **shift( ):** Removes the first element of an array, and returns that element
- **sort( ):** Sorts the elements of an array
- **toString( ):** Converts an array to a string, and returns the result
- **unshift( ):** Adds new elements to the beginning of an array, and returns the new length

Example

Concat( ) : Joining Two Arrays

```
<script type="text/javascript">
var parents = ["Giri", "Pari"];
var children = ["Cactus", "Rose"];
var family = parents.concat(children);
document.write(family);
</script>
```

The output will be :

Giri, Pari, Cactus, Rose

String Object in JavaScript

The String object is used to manipulate a stored piece of text. String objects are created with new String().

Syntax

```
var txt = new String(string);or more simply:
var txt = string;
```

Some methods associated with String object:

- **toLowerCase( ):** Converts a string to lowercase letters
- **toUpperCase( ):** Converts a string to uppercase letters
- **concat( ):** Joins two or more strings, and returns a copy of the joined strings
- **charAt( ):** Returns the character at the specified index
- **indexOf( ):** Returns the position of the first found occurrence of a specified value in a string
- **replace( ):** Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring

Examples

In the following example we are using the length property of the String object to return the number of characters in a string:

```
<script type="text/javascript">
var txt="Hello World!";
document.write(txt.length);
</script>
The output of the code above will be: 12
```

In the following example we are using the toUpperCase( ) method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
var str="hello its me webtech!";
document.write(str.toUpperCase());
</script>
```

The output of the code above will be:

```
HELLO ITS ME WEBTECH
```

Example: IndexOf ( ) method

The indexOf( ) method returns the position of the first occurrence of a specified value in a string. This method returns -1 if the value to search for never occurs. The indexOf( ) method is case sensitive.

Syntax

string.indexOf(searchstring, start)

searchstring: Required. The string to search for.

start: Optional. The start position in the string to start the search. If omitted, the search starts from position 0

```
<script type="text/javascript">
var str="Patan world!";
document.write(str.indexOf("d") + "<br />");
document.write(str.indexOf("WORLD") + "<br />");
document.write(str.indexOf("world"));

</script>
```

Output

```
10
-1
6
```

Math Object in Javascript

The Math object allows you to perform mathematical tasks. The Math object includes several mathematical constants and methods. For example

```
var pi_value=Math.PI;
var sqrt_value=Math.sqrt(16);
```

Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Properties

- **Math.E:** Returns Euler's number (approx. 2.718)
- **Math.LN2:** Returns the natural logarithm of 2 (approx. 0.693)
- **Math.LN10:** Returns the natural logarithm of 10 (approx. 2.302)
- **Math.LOG2E:** Returns the base-2 logarithm of E (approx. 1.442)
- **Math.LOG10E:** Returns the base-10 logarithm of E (approx. 0.434)
- **Math.PI:** Returns PI (approx. 3.14159)
- **Math.SQRT1_2:** Returns the square root of 1/2 (approx. 0.707)
- **Math.SQRT2:** Returns the square root of 2 (approx. 1.414)

Methods

- **abs(x):** Returns the absolute value of x
- **ceil(x):** Returns x, rounded upwards to the nearest integer
- **floor(x):** Returns x, rounded downwards to the nearest integer
- **log(x):** Returns the natural logarithm (base E) of x

- **max(x,y,z,...,n):** Returns the number with the highest value
- **min(x,y,z,...,n):** Returns the number with the lowest value
- **pow(x,y):** Returns the value of x to the power of y
- **sqrt(x):** Returns the square root of x
- **random( ):** Returns a random number between 0 and 1
- **round(x):** Rounds x to the nearest integer
- **sin(x):** Returns the sine of x (x is in radians)
- **cos(x):** Returns the cosine of x (x is in radians)
- **tan(x):** Returns the tangent of an angle

Examples

```
document.write(Math.round(4.7));
Output: 5

document.write(Math.random());
Output: 0.19733826867061233

document.write(Math.floor(Math.random()*6));
Output: 3
```

<u>Date Object in Javascript</u>

The Date object is used to work with dates and times. Date objects are created with the Date( ) constructor. We can easily manipulate the date by using the methods available for the Date object. In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date();
myDate.setDate(myDate.getDate()+5);
```

Note: If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Methods

- getDate() Returns the day of the month (from 1-31)
- getDay() Returns the day of the week (from 0-6)
- getFullYear() Returns the year (four digits)
- getHours() Returns the hour (from 0-23)
- getMilliseconds() Returns the milliseconds (from 0-999)
- getMinutes() Returns the minutes (from 0-59)
- getMonth() Returns the month (from 0-11)
- getSeconds() Returns the seconds (from 0-59)
- setDate() Sets the day of the month (from 1-31)
- setFullYear() Sets the year (four digits)
- setHours() Sets the hour (from 0-23)
- setMilliseconds() Sets the milliseconds (from 0-999)
- setMinutes() Set the minutes (from 0-59)
- setMonth() Sets the month (from 0-11)
- setSeconds() Sets the seconds (from 0-59)
- toString() Converts a Date object to a string

Examples

The Date object is also used to compare two dates. The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
```

```
var today = new Date();
if (myDate>today)
  {
  alert("Today is before 15th December 2011");
  }
else
  {
  alert("Today is after 15th January 2011");
  }
```

Examples

```
<html>
      <head>
      <script type="text/javascript">
      function displayDate()
      {
      document.getElementById("demo").innerHTML=Date();
      }
      </script>
      </head>
      <body>

      <h1>My First Web Page</h1>
      <p id="demo">This is a paragraph.</p>

      <button type="button" onclick="displayDate()">Display Date</button>

      </body>
</html>
```

```
<html>
        <body>
        <script type="text/javascript">
        var d=new Date();
        document.write(d);
        </script>
        </body>
</html>
```

Example: Displaying the clock

```
<html>
        <head>
        <script type="text/javascript">
        function startTime()
        {
        var today=new Date();
        var h=today.getHours();
        var m=today.getMinutes();
        var s=today.getSeconds();
        // add a zero in front of numbers<10 //m=checkTime(m);
        //s=checkTime(s);
        document.getElementById('txt').innerHTML=h+":"+m+":"+s;
        t=setTimeout('startTime()',1000);
        }

        //to concat 0 if i is not double digit
        /*function checkTime(i)
        {
        if (i<10)
          {
```

```
    i="0" + i;
    }
   return i;
   } */
   </script>
   </head>
   <body onload="startTime()">
   <div id="txt"></div>
   </body>
</html>
```

With JavaScript, it is possible to execute some code after a specified time-interval. This is called timing events It's very easy to time events in JavaScript. The two key methods that are used are:

- *setTimeout() - executes a code some time in the future*
- *clearTimeout() - cancels the setTimeout()*

Note: *The setTimeout() and clearTimeout() are both methods of the HTML DOM Window object.*

The setTimeout() method returns a value. In the syntax defined above, the value is stored in a variable called t. If you want to cancel the setTimeout() function, you can refer to it using the variable name. The first parameter of setTimeout() can be a string of executable code, or a call to a function. The second parameter indicates how many milliseconds from now you want to execute the first parameter.

Note: *There are 1000 milliseconds in one second.*

In above example the function startTime( ) get executed after each second, showing the content of div tag getting refreshed each time so as to display the clock.

User defined objects in JavaScript:

We have seen that JavaScript has several built-in objects, like String, Date, Array, and more. In addition to these built-in objects, you can also create your own.

An object is just a special kind of data, with a collection of properties and methods.

Let's illustrate with an example: A person is an object. Properties are the values associated with the object. The persons' properties include name, height, weight, age, skin tone, eye color, etc. All persons have these properties, but the values of those properties will differ from person to person. Objects also have methods. Methods are the actions that can be performed on objects. The persons' methods could be eat(), sleep(), work(), play(), etc.

The syntax for accessing a property of an object is:

```
objName.propName
```

You can call a method with the following syntax:

```
objName.methodName()
```

Note: Parameters required for the method can be passed between the parentheses.

There are different ways to create a new object:

1. Create a direct instance of an object

The following code creates an new instance of an object, and adds four properties to it:

```
personObj=new Object();
personObj.firstname="Jyoti";
personObj.lastname="Joshi";
personObj.age=25;
personObj.eyecolor="black";
```

alternative syntax (using object literals):

```
personObj={firstname:"Jyoti", lastname:"Joshi", age:25, eyecolor:"black"};
```

Adding a method to the personObj is also simple. The following code adds a method called eat() to the personObj:

```
personObj.eat=eat;

function eat( )
{
// code for the function
}
```

2. Create an object constructor

Create a function that constructs objects:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

Inside the function you need to assign things to this.propertyName. The reason for all the "this" stuff is that you're going to have more than one person at a time (which person you're dealing with must be clear). That's what "this" is: the instance of the object at hand.

Once you have the object constructor, you can create new instances of the object, like this:

```
var myFather=new person("Ramesh","Joshi",50,"black");
var myMother=new person("Gita","Joshi",48,"blue");
```

You can also add some methods to the person object. This is also done inside the function:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;

this.newlastname=newlastname;
}
```

Note that methods are just functions attached to objects. Then we will have to write the newlastname( ) function:

```
function newlastname(new_lastname)
{
this.lastname=new_lastname;
}
```

The newlastname( ) function defines the person's new last name and assigns that to the person. JavaScript knows which person you're talking about by using "this." . So, now you can write: myMother.newlastname("Joshi").

Example: Creating a circle object

```
<html>
        <head>
        <script type="text/javascript">
        // mycircle object defined
        function mycircle(r) {
          this.radius = r; this.retArea
          = getTheArea;
        }
        function getTheArea( )
        {
          return ( Math.PI * this.radius * this.radius );
        }
        function createcircle ( )
        {
        //create a mycircle called testcircle wtih radius 10
        var testcircle = new mycircle(10);
        window.alert( 'The area of the circle is ' + testcircle.retArea );
        }
        </script>
        </head>

        <body onLoad="createcircle()"> </body>
</html>
```

HTML Document Object Model

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. DOM provides a language-independent, object-based model for accessing / modifying and adding to these tags.

The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents. All HTML elements, along with their containing text and attributes, can be accessed through the DOM. The contents can be modified or deleted, and new elements can be created. The HTML DOM is platform and language independent. It can be used by any programming language like Java, JavaScript, and VBScript.

When an HTML page is rendered in a browser, the browser assembles all the elements (objects) that are contained in the HTML page, downloaded from web-server in its memory. Once done the browser then renders these objects in the browser window as text, forms, input boxes, etc. Once the HTML page is rendered in web-browser window, the browser can no longer recognize individual HTML elements (Objects).

Since the JavaScript enabled browser uses the **D**ocument **O**bject **M**odel (DOM), after the page has been rendered, JavaScript enabled browsers are capable of recognizing individual objects in an HTML page.

The HTML objects, which belong to the DOM, have a descending relationship with each other.

The topmost object in the DOM is the **Navigator** (*i.e.* Browser) itself. The next level in the DOM is the browser's **Window**, and under that are the **Documents** displayed in Browser's Window.

```
DOM
   |-> Window
       |-> Document
             |-> Anchor
             |-> Link
             |-> Form
                  |-> Text-box
                  |-> Text Area
                  |-> Radio Button
                  |-> Check Box
                  |-> Select
                  |-> Button

                  ……….
```

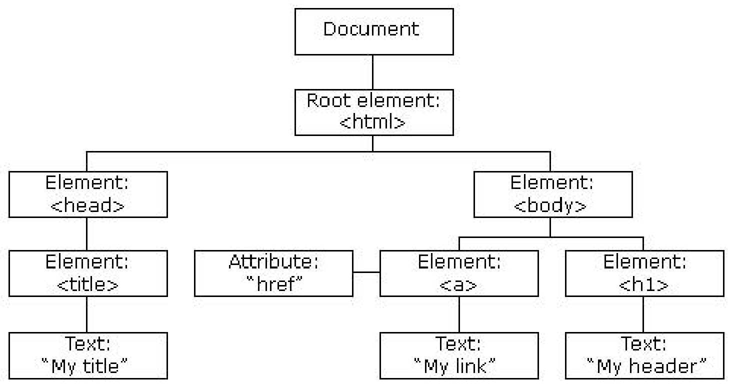


Fig: HTML DOM Tree Example

The Form Object:

The Form object represents an HTML form. For each <form> tag in an HTML document, a Form object is created. Forms are used to collect user input, and contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select menus, textarea, fieldset, legend, and label elements. Forms are used to pass data to a server.

Form Object Collections

Collection	**Description**
elements[]	Returns an array of all elements in a form

Form Object Properties

Property	**Description**
acceptCharset	Sets or returns the value of the accept-charset attribute in a form
action	Sets or returns the value of the action attribute in a form
enctype	Sets or returns the value of the enctype attribute in a form
length	Returns the number of elements in a form
method	Sets or returns the value of the method attribute in a form
name	Sets or returns the value of the name attribute in a form
target	Sets or returns the value of the target attribute in a form

Form Object Methods

Method	**Description**
reset()	Resets a form
submit()	Submits a form

Form Object Events

Event	**The event occurs when...**
onreset	The reset button is clicked
onsubmit	The submit button is clicked

Form Method Property

The method property sets or returns the value of the method attribute in a form. The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute).

formObject.method=*value*

The method property can have one of the following values:

Value	**Description**
get	Appends the form-data to the URL: URL?name=value&name=value (this is default)
post	Sends the form-data as an HTTP post transaction

RegExp Object:

A regular expression is an object that describes a pattern of characters. When you search in a text, you can use a pattern to describe what you are searching for. A simple pattern can be one single character. A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.

Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

```
var patt=new RegExp(pattern,modifiers);
```

or more simply:

```
var patt=/pattern/modifiers;
```

- pattern specifies the pattern of an expression
- modifiers specify if a search should be global, case-sensitive, etc.

Modifiers: Modifiers are used to perform case-insensitive and global searches. The ***i*** modifier is used to perform case-insensitive matching. The ***g*** modifier is used to perform a global match (find all matches rather than stopping after the first match).

For example:

```
<html>
<body>

<script type="text/javascript">
var str = "Visit W3Schools";
var patt1 = /w3schools/i;
document.write(str.match(patt1));
</script>

</body>
</html>
```

The output: W3Schools

```
<html>
<body>
```

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

var str="Is this all there is?";
var patt1=/is/g;
document.write(str.match(patt1));

</script>

</body>
</html>
```

The output : is, is

```
<html>
<body>

<script type="text/javascript">

var str="Is this all there is?";
var patt1=/is/gi;
document.write(str.match(patt1));

</script>

</body>
</html>
```

The output : Is,is,is

test()

The test() method searches a string for a specified value, and returns true or false, depending on the result. The following example searches a string for the character "e":

```
<html>
<body>

<script type="text/javascript">
var patt1=new RegExp("e");

document.write(patt1.test("The best things in life are free"));
</script>

</body>
</html>
```

exec()

The exec() method searches a string for a specified value, and returns the text of the found value. If no match is found, it returns *null*. The following example searches a string for the character "e":

```
<html>
<body>

<script type="text/javascript">
var patt1=new RegExp("e");

document.write(patt1.exec("The best things in life are
free")); </script>

</body>
</html>
```

A caret (^) at the beginning of a regular expression indicates that the string being searched must start with this pattern.

- The pattern ^foo can be found in "food", but not in "barfood".

A dollar sign ($) at the end of a regular expression indicates that the string being searched must end with this pattern.

- The pattern foo$ can be found in "curfoo", but not in "food"

Number of Occurrences (? + * {})

The following symbols affect the number of occurrences of the preceding character: ?, +, *, and {}.

A questionmark (?) indicates that the preceding character should appear zero or one times in the pattern.

- The pattern foo? can be found in "food" and "fod", but not "faod".

A plus sign (+) indicates that the preceding character should appear one or more times in the pattern.

- The pattern fo+ can be found in "fod", "food" and "foood", but not "fd".

A asterisk (*) indicates that the preceding character should appear zero or more times in the pattern.

- The pattern fo*d can be found in "fd", "fod" and "food".

Curly brackets with one parameter ({n}) indicate that the preceding character should appear exactly n times in the pattern.

- The pattern fo{3}d can be found in "foood" , but not "food" or "fooood".

Curly brackets with two parameters ({n1,n2}) indicate that the preceding character should appear between n1 and n2 times in the pattern.

- The pattern fo{2,4}d can be found in "food","foood" and "fooood", but not "fod" or "foooood".

Curly brackets with one parameter and an empty second paramenter ({n,}) indicate that the preceding character should appear at least n times in the pattern.

- The pattern fo{2,}d can be found in "food" and "foooood", but not "fod".

Common Characters (. \d \D \w \W \s \S)

A period (.) represents any character except a newline.

- The pattern fo.d can be found in "food", "foad", "fo9d", and

"fo*d". Backslash-d (\d) represents any digit. It is the equivalent of [0-9].

- The pattern fo\dd can be found in "fo1d", "fo4d" and "fo0d", but not in "food" or "fodd".

Backslash-D (\D) represents any character except a digit. It is the equivalent of [^0-9].

- The pattern fo\Dd can be found in "food" and "foad", but not in "fo4d".

Backslash-w (\w) represents any word character (letters, digits, and the underscore (_)).

- The pattern fo\wd can be found in "food", "fo_d" and "fo4d", but not in "fo*d".

Backslash-W (\W) represents any character except a word character.

- The pattern fo\Wd can be found in "fo*d", "fo@d" and "fo.d", but not in "food".

Backslash-s (\s) represents any whitespace character (e.g, space, tab, newline, etc.).

- The pattern fo\sd can be found in "fo d", but not in "food".

Backslash-S (\S) represents any character except a whitespace character.

- The pattern fo\Sd can be found in "fo*d", "food" and "fo4d", but not in "fo d".

Form Validation:

Form validation is the process of checking that a form has been filled in correctly before it is processed. For example, if your form has a box for the user to type their email address, you might want your form handler to check that they've filled in their address before you deal with the rest of the form.

There are two main methods for validating forms: ***server-side*** (using CGI scripts, ASP, etc), and ***client-side*** (usually done using JavaScript). Server-side validation is more secure but often more tricky to code and it also increases load of server computer, whereas client-side (JavaScript) validation is easier to do and quicker too (the browser doesn't have to connect to the server to validate the form, so the user finds out instantly if they've missed out that required field!) and it also decreases the load of server computer and hence server computer can focus on business logic processing.

Form Validation - Checking for Non-Empty

This has to be the most common type of form validation. You want to be sure that your visitors enter data into the HTML fields you have "required" for a valid submission. Below is the JavaScript code to perform this basic check to see if a given HTML input is empty or not.

```
<script type='text/javascript'>
function notEmpty()
{
	var v= document.getElementById('elem').value;
	if(v.length == 0)
	{
		alert("Field should not be empty:");
		document.getElementById('elem').value=” ”;
		document.getElementById('elem').focus();
```

```
        }
}
</script>
<form>
Required Field: <input type='text' id='elem'/>
<input type='button' onclick="notEmpty()" value='Check'/>
</form>
```

Form Validation - Checking for All Numbers

If someone is entering a credit card, phone number, zip code, similar information you want to be able to ensure that the input is all numbers. The quickest way to check if an input's string value is all numbers is to use a regular expression /^[0-9]+$/ that will only *match* if the string is all numbers and is at least one character long. <script type='text/javascript'>

```
function validate()
{
        var patt=/^[0-9]+$/;
        var v= document.getElementById('elem').value;
        if(v.match(patt))
        {
                alert("valid entry");
        }
        else
        {
                alert("Invalid entry:");
                document.getElementById('elem').value="";
                document.getElementById('elem').focus();
        }
}
</script>
<form>
```

```
Required Field: <input type='text' id='elem'/>
<input type='button' onclick="validate()" value='Check'/>
</form>
```

Form Validation - Checking for All Letters

If we wanted to see if a string contained only letters we need to specify an expression that allows for both lowercase and uppercase letters: /^[a-zA-Z]+$/ .

```
<script
type='text/javascript'>
function validate()
{
        var patt=/^[a-zA-Z]+$/;
        var v= document.getElementById('elem').value;
        if(v.match(patt))
        {
                alert("valid entry");
        }
        else
        {
                alert("Invalid entry:");
                document.getElementById('elem').value="";
                document.getElementById('elem').focus();
        }
}
</script>
<form>
Required Field: <input type='text' id='elem'/>
<input type='button' onclick="validate()" value='Check'/>
</form>
```

Form Validation - Restricting the Length

Being able to restrict the number of characters a user can enter into a field is one of the best ways to prevent bad data. Below we have created a function that checks for length of input.

```
<script type='text/javascript'>
function validate()
{
        var minlen=6;
        var v= document.getElementById('elem').value;
        if(v.length<6)
        {
                alert("User ID must have at least 6 Characters");
                document.getElementById('elem').value="";
                document.getElementById('elem').focus();
        }
        else
        {
                alert("Valid entry:");

        }
}
</script>
<form>
User ID: <input type='text' id='elem'/>
<input type='button' onclick="validate()" value='Check'/>
</form>
```

Form Validation - Selection Made

To be sure that someone has actually selected a choice from an HTML select input you can use a simple trick of making the first option as helpful prompt to the user and a red flag to you for your validation code. By making the first option of your select input something like "Please Choose" you can spur the user to both make a selection and allow you to check to see if the default option "Please Choose" is still selected when he/she submit the form.

```
<script type='text/javascript'>
function validate()
{
        var si=document.getElementById('con').selectedIndex;
        var v= document.getElementById('con').options[si].text;
        if(v=="Please Choose")
        {
                alert("You must choose the country");

        }
        else
        {
                alert("Your Country is:"+v);

        }
}
</script>
<form>
Select Country: <select id='con'>
        <option>Please Choose</option> <option>Nepal</option>
        <option>India</option> <option>China</option>
</select>
<input type='button' onclick='validate()'
value='Check'/> </form>
```

Validating radio buttons

Radio buttons are used if we want to choose any one out of many options such as gender. In such case any one of the radio button must be selected. We can validate radio button selection as below:

```
<script type='text/javascript'>
function validate()
{
        var sex=document.getElementsByName("gen");
        if(sex[0].checked==false && sex[1].checked==false)
        {
                alert("You must choose Gender");
        }
        else
        {
                if(sex[0].checked==true)
                alert("Male");
                else
                alert("Female");
        }
}
</script>
<form>
Select Gender:
        <input type=radio name='gen'>Male
        <input type=radio name='gen'>Female
        <input type='button' onclick='validate()'
value='Check'/> </form>
```

Form Validation - Email Validation

How to check to see if a user's email address is valid? Every email is made up for 5 parts:

1. A combination of letters, numbers, periods, hyphens, plus signs, and/or underscores
2. The at symbol @
3. A combination of letters, numbers, hyphens, and/or periods
4. A period
5. The top level domain (com, net, org, us, gov, ...)

- ~~jagdish~~sanjay@ntc.net
- ~~jagdish~~sanjay+bhatta@gmail.com
- ~~jagdish~~sanjay-

bhatta@patan.edu.np Invalid

Examples:

- @deleted.net - no characters before the @
- free!dom@bravehe.art - invalid character !
- shoes@need_shining.com - underscores are not allowed in the domain name

```
<script type='text/javascript'>
function validate()
{
	var patt=/^[\w\-\.\+]+\@[a-zA-Z0-9\.\-]+\.[a-zA-z0-9]{2,4}$/;
	var v= document.getElementById('elem').value;
	if(v.match(patt))
	{
		alert("valid Email");
	}
	else
	{
		alert("Invalid Email"); document.getElementById('elem').value="";
		document.getElementById('elem').focus();
	}
}
</script>
```

```
<form>
        Email ID: <input type='text' id='elem'/>
        <input type='button' onclick="validate()" value='Check'/>
</form>
```

Handling Cookies in JavaScript:

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values. A cookie is nothing but a small text file that's stored in your browser. It contains some data:

1. A name-value pair containing the actual data
2. An expiry date after which it is no longer valid
3. The domain and path of the server it should be sent to

As soon as you request a page from a server to which a cookie should be sent, the cookie is added to the HTTP header. Server side programs can then read out the information and decide that you have the right to view the page you requested. So every time you visit the site the cookie comes from, information about you is available. This is very nice sometimes, at other times it may somewhat endanger your privacy. Cookies can be read by JavaScript too. They're mostly used for storing user preferences.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome John Doe!" The name is retrieved from the stored cookie
- Password cookie - The first time a visitor arrives to your web page, he or she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie

- Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie
- And so on.

document.cookie:

Cookies can be created, read and erased by JavaScript. They are accessible through the property document.cookie. Though you can treat document.cookie as if it's a string, it isn't really, and you have only access to the name-value pairs. If you want to set a cookie for this domain with a name-value pair 'ppkcookie1=testcookie' that expires in seven days from the moment you should write this sentence,

document.cookie = “ppkcookie1=testcookie; expires=Thu, 2 Aug 2001 20:47:11 UTC; path=/”

1. First the name-value pair ('ppkcookie1=testcookie')
2. then a semicolon and a space
3. then the expiry date in the correct format ('expires=Thu, 2 Aug 2001 20:47:11 UTC')
4. again a semicolon and a space
5. then the path (path=/)

Example:

```
function createCookie(name, value, days) {
    if (days) {
        var date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        var expires = "; expires=" + date.toGMTString();
    }
    else var expires = "";
    document.cookie = name + "=" + value + expires + "; path=/";
}

function getCookie(c_name) {
    if (document.cookie.length > 0) {
        c_start = document.cookie.indexOf(c_name + "=");
        if (c_start != -1) {
            c_start = c_start + c_name.length + 1;
            c_end = document.cookie.indexOf(";", c_start);
            if (c_end == -1) {
                c_end = document.cookie.length;
            }
            return unescape(document.cookie.substring(c_start, c_end));
        }
    }
    return "";
}
```

More we can set cookie as below with the proper paths, domain and other parameters;

```
function setCookie(name, value, expires, path, domain)
{

            /* Some characters - including spaces - are not allowed in cookies so we escape to
            change the value we have entered into a form acceptable to the cookie.*/

            var thisCookie = name + "=" + escape(value) +
            ((expires) ? "; expires=" + expires.toGMTString() : "") +
            ((path) ? "; path=" + path : "") + ((domain) ? ";
            domain=" + domain : "") ;

            document.cookie = thisCookie;

}
```

Simply we can display cookie in alert box as;

```
function showCookie()
{
            alert(unescape(document.cookie));
}
```

More Example;

```
<html>
<head>
<script type="text/javascript">
function setCookie()
{
    var name="Cookie1";
    var value="~~Jagdish~~Sanjay";
    var ed=new Date();
    ed.setDate(ed.getDate() +2);
    document.cookie = name + "=" + value+" ;expires="+ed.toGMTString()+" ;path=/";
}
function getCookie()
{
        var l=document.cookie.length;
        setCookie();
        var ind=document.cookie.indexOf("Cookie1=");
        if(ind==-1)
        {
                alert("Cookie not found");
        }
        else
        {
                var n=document.cookie.substring(ind+8,l);
                alert("Wel come:"+n);
        }
}
</script> </head>
<body>
        <input type=button value="setcookie" onclick="setCookie()">
        <input type=button value="getcookie" onclick="getCookie()">
```

</body> </html>

<u>Handling runtime errors in JavaScript:</u>

An exception is an error that occurs at **runtime** due to an illegal operation during execution. Examples of exceptions include trying to reference an undefined variable, or calling a non-existent method. S**yntax** errors occur when there is a problem with your JavaScript syntax. Consider the following examples of syntax errors versus exceptions:

```
alert("I am missing a closing parenthesis //syntax error
alert(x) //exception assuming "x" isn't defined yet
undefinedfunction() //exception
```

It is almost impossible for a programmer to write a program without errors. Programming languages include exceptions, or errors, that can be tracked and controlled. Exception handling is a very important concept in programming technology. In earlier versions of JavaScript, the exceptions handling was not so efficient and programmers found it difficult to use. Later versions of JavaScript resolved this difficulty with exceptions handling features like try..catch handlers, which presented a more convenient solution for programmers. Normally whenever the browser runs into an exception somewhere in a JavaScript code, it displays an error message to the user while aborting the execution of the remaining code. There are mainly two ways of trapping errors in JavaScript.

- Using try…catch statement
- Using onerror event

Using try…catch statement:

The try..catch statement has two blocks in it: try block and catch block. In the try block, the code contains a block of code that is to be tested for errors. The catch block contains the code that is to be executed if an error occurs. The general syntax of try..catch statement is as follows:

```
try
{
            …………
            …………//Block of code which is to be tested for errors
}
catch (err)
{
            …………
            …………    //Block of code which is to be executed if an error occurs
}
```

When, in the above structure, an error occurs in the try block then the control is immediately transferred to the catch block with the error information also passed to the catch block. Thus, the try..catch block helps to handle errors without aborting the program and therefore proves user-friendly.

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
        try
        {
                adddlert("Welcome guest!");
                alert("test");
        }
        catch(err)
        {
                txt="There was an error on this page.\n\n";
                txt+="Click OK to continue viewing this page,\n";
                txt+="or Cancel to return to the home page.\n\n";
                if(!confirm(txt))
                {
                        document.location.href="http://www.w3schools.com/";
                }
        }
 }
</script>
</head>
<body>
        <input type="button" value="View message" onclick="message()" />
</body>
</html>
```

There is another statement called throw available in JavaScript that can be used along with. try…catch statements to throw exceptions and thereby helps in generating. General syntax of this throw statement is as follows:

```
        throw(exception)
```

```
<html>
 <body>
        <script type="text/javascript">
        try
        {
                var a=10;
                var b=0;
                if(b==0)
                {
                        throw "Division by zero!!!!"
                }
        }

        catch(err)
        {
                alert(err);
        }
        </script>
</body>
</html>
```

Although finally is not used as often as catch, it can often be useful. The finally clause is guaranteed to be executed if any portion of the try block is executed, regardless of how the code in the try block completes. It is generally used to clean up after the code in the try clause. If an exception occurs in the try block and there is an associated catch block to handle the exception, control transfers first to the catch block and then to the finally block. If there is no local catch block to handle the exception, control transfers first to the finally.

```
<head>
<script type="text/javascript">
<!--
        function myFunc()
        {
           var a = 100;
           try
          {
              alert("Value of variable a is : " + a );
           }
        catch ( e )
        {
              alert("Error: " + e.description );
        }
        finally
        {
              alert("Finally block will always execute!" );
        }
        }
        //-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
```

```
<form>
<input type="button" value="Click Me" onclick="myFunc()" />
</form>
</body>
</html>
```

Using onerror event

The onerror event fires when a page has a script error. This onerror event occurs in JavaScript when an image or document causes an error during loading. This does not mean that it is a browser error. This event handler will only be triggered by a JavaScript error, not a browser error. The general syntax of onerror event is as follows:

```
onerror=functionname()
function functionname()
{
   //Error Handling Code
}
```

Example:

```
<html>
<head>
<script type="text/javascript">
        onerror=exfoerr
        var text1=""
        function exfoerr(msg,url,line)
        {
              text1="Error Displayed\n\n"
              text1+="Error: " + msg + "\n"
              text1+="URL: " + url + "\n"
              text1+="Line Number: " + line + "\n\n"
               text1+="Click OK to continue.\n\n"
              alert(text1)
```

```
            return true
        }
        function display()
        {
            addxlert("Click to Proceed!!!!")
        }
     </script>
  </head>
   <body>
      <input type="button" value="View message"
       onclick="display()" />
  </body>
</html>
```

In the above example program, the function display() has an error in it (the addalert is typed wrongly as addxlert). When the program reads this error, the onerror event handler fires and the function exfor( ) is called with the three parameters passed to it (the error message, the url of the page and the line number of error 18)

~~Web Technology (CSC-353)~~

~~Jagdish Bhatta~~

~~Central Department of Computer Science & Information Technology~~

~~Tribhuvan University~~