ARYAN SCHOOL OF ENGINEERING & ECHNOLOGY

BARAKUDA, PANCHAGAON, BHUBANESWAR, KHORDHA-752050

LECTURE NOTE

SUBJECT NAME- INTERNET AND WEB TECHNOLOGY

BRANCH-COMPUTER SCIENCE ENGG.

SEMESTER-5TH SEM

ACADEMIC SESSION-2022-23

PREPARED BY- PRAKASH KUMAR DEHURY

Understanding the WWW and the Internet:

Internet: The Internet is a global system of interconnected computer networks that use the standard Internet Protocol Suite (TCP/IP) to serve billions of users worldwide. It is a *network of networks* that consists of millions of private, public academic, business, and government networks.
WWW: The World Wide Web, abbreviated as WWW and commonly known as the Web, is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them via hyperlinks.
Emergence of Web: Between the summers of 1991 and 1994, the load on the first Web server ("info.cern.ch") rose steadily by a factor of 10 every year.
In 1992 academia, and in 1993 industry, was taking notice. World Wide Web Consortium is formed in September 1994, with a base at MIT is the USA, INRIA in France, and now also at Keio University in Japan.
With the dramatic flood of rich material of all kinds onto the Web in the 1990s, the first part of the dream is largely realized, although still very few people in practice have access to intuitive hypertext creation tools.
The second part has yet to happen, but there are signs and plans which make us confident. The great need for information about information, to help us categorize, sort, pay for own information is driving the design of languages for the web designed for processing by machines, rather than people. The web of human readable document is being merged with a web of machine-understandable data. The potential of the mixture of humans and machines working together and communicating through the web could be immense.
WEB Servers: To view and browse pages on the Web, all you need is a web browser. To publish pages on the Web, you need a web server. A web server is the program that runs on a computer and is responsible for replying to web browser requests for files. You need a web server to publish documents on the Web. When you use a browser to request a page on a website, that browser makes a web connection to a server using the HTTP protocol. The browser then formats the information it got from the server. Server accepts the connection, sends the contents of the requested files and then closes.
WEB Browsers:
A web browser is the program you use to view pages and navigate the World Wide Web. A wide array of web browsers is available for just about every platform you can imagine. Microsoft Internet Explorer, for example, is included with Windows and Safari is included with Mac OS X. Mozilla Firefox, Netscape Navigator, and Opera are all available for free.
What the Browser Does The core purpose of a web browser is to connect to web servers, request documents, and then properly format and display those documents. Web browsers can also display files on your local computer, download files that are not meant to be displayed. Each web page is a file written in a language called the Hypertext Markup

Language (HTML) that includes the text of the page, a description of its structure, and links to other documents, images, or other media.

Protocols: In computing, a protocol is a set of rules which is used by computers to communicate with each other across a network. A protocol is a convention or standard that controls or enables the connection, communication, and data transfer between computing endpoints.

Internet Protocol Suite: The Internet Protocol Suite is the set of communications protocols used for the Internet and other similar networks. It is commonly also known as TCP/IP named from two of the most important protocols in it: The Transmission Control Protocol (TCP) and the Internet Protocol (IP), which were the first two networking protocols defined in this standard.

Building Web sites: It's a good idea to first think about and design your site. That way, you'll give yourself direction and you'll need to reorganize less later.

To design your site:

1. Figure out why you're creating this site. What do you want to convey?
2. Think about your audience. How can you tailor your content to appeal to this audience? For example, should you add lots of graphics or is it more important that your page download quickly?
3. How many pages will you need? What sort of structure would you like it to have? Do you want visitors to go through your site in a particular direction, or do you want to make it easy for them to explore in any direction?
4. Sketch out your site on paper.

Devise a simple, consistent naming system for your pages, images and other external files.

HTML

Planning for designing web pages:

Breaking Up Your Content into Main Topics

With your goals in mind, try to organize your content into main topics or sections, chunking related information together under a single topic.

Ideas for Organization and Navigation

At this point, you should have a good idea of what you want to talk about as well as a list of topics. The next step is to actually start structuring the information you have into a set of web pages. Before you do that, however, consider some standard structures that have been used in other help systems and online tools. This section describes some of these structures, their various features, some important considerations, including the following

Model and Structure of a Web site:

You need to know what the following terms mean and how they apply to the body of work you're developing for the Web:

Website: A collection of one or more web pages linked together in a meaningful way that, as a whole, describes a body of information or creates an overall effect.

Web server: A computer on the Internet or an intranet that delivers Web pages and other files in response to browser requests.

Web page: A single document on a website, usually consisting of an HTML document and any items that are displayed within that document such as inline images.

Home page: The entry page for a website, which can link to additional pages on the same website or pages on other sites.

Developing websites:

Designing a website, like designing a book outline, a building plan, or a painting, can sometimes be a complex and involved process. Having a plan before you begin can help you keep the details straight and help you develop the finished product with fewer false starts. Today, you learned how to put together a simple plan and structure for creating a set of web pages, including the following:

- Deciding what sort of content to present
- Coming up with a set of goals for that content
- Deciding on a set of topics
- Organizing and storyboarding the website

Basic HTML: HTML stands for Hypertext Markup Language. The idea here is that most documents have common elements for example, titles, paragraphs, and lists. Before you start writing, therefore, you can identify and define the set of elements in that document and give them appropriate names.

How Markup Works

HTML is a markup language. Writing in a markup language means that you start with the text of your page and add special tags around words and paragraphs. The tags indicate the different parts of the page and produce different effects in the browser. HTML has a defined set of tags you can use. You can't make up your own tags to create new styles or features.

What HTML Files Look Like

Pages written in HTML are plain text files (ASCII), which means that they contain no platform- or program-specific information. Any editor that supports text can read them. HTML files contain the following:

- The text of the page itself
- HTML tags that indicate page elements, structure, formatting, and hypertext links to

other pages or to included media. Most HTML tags look something like the following:

```
<thetagname>affected text</thetagname>
```

The tag name itself (here, thetagname) is enclosed in brackets (< >). HTML tags generally have a beginning and an ending tag surrounding the text they affect. The beginning tag "turns on" a feature (such as headings, bold, and so on), and the ending tag turns it off. Closing tags have the tag name preceded by a slash (/). The opening tag (for example, <p> for paragraphs) and closing tag (for example, </p> for paragraphs) compose what is officially called an HTML element.

Text Formatting and HTML

When an HTML page is parsed by a browser, any formatting you might have done by hand that is, any extra spaces, tabs, returns, and so on is ignored. The only thing that specifies formatting in an HTML page is an HTML tag. If you spend hours carefully editing a plain text file to have nicely formatted paragraphs and columns of numbers but don't include any tags, when a web browser loads the page, all the text will flow into one paragraph. All your work will have been in vain.

The advantage of having all white space (spaces, tabs, returns) ignored is that you can put your tags wherever you want. The following examples all produce the same output. Try them!

```
<h1>If music be the food of love, play on.</h1>
<h1>
If music be the food of love, play on.
</h1>
<h1>
If music be the food of love, play on. </h1>
<h1> If music be the food of love,
play on. </h1 >
```

Structuring Your HTML

The DOCTYPE Identifier

Although it's not a page structure tag, the XHTML 1.0 recommendation includes one additional requirement for your web pages. The first line of each page must include a DOCTYPE identifier that defines the XHTML 1.0 version to which your page conforms, and the document type definition (DTD) that defines the specification. This is followed by the <html>, <head>, and <body> tags. In the following example, the XHTML 1.0 Strict document type appears before the page structure tags:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/strict.dtd">
<html>
<head>
<title>Page Title</title>
</head>
<body>
...your page content...
</body>
</html>
```

Three types of HTML 4.01 document types are specified in the XHTML 1.0 specification:
Strict, Transitional, and Frameset.

The <html> Tag

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
...your page...
</html>
```

The <head> Tag

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>This is the Title. It will be explained later on</title>
</head>
...your page...
</html>
```

The <body> Tag

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>This is the Title. It will be explained later on</title>
</head>
<body>
...your page...
</body>
</html>
```

The Title

Each HTML page needs a title to indicate what the page describes. It appears in the title bar of the browser when people view the web page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>The Lion, The Witch, and the Wardrobe</title>
</head>
<body>
...your page...
</body>
</html>
```

Headings

Headings are used to add titles to sections of a page. HTML defines six levels of headings.

Heading tags look like the following:

```
<h1>Movies</h1>
<h2>Action/Adventure</h2>
<h3>Caper</h3>
<h3>Sports</h3>
<h3>Thriller</h3>
```

```
<h3>War</h3>
<h2>Comedy</h2>
<h3>Romantic Comedy</h3>
<h3>Slapstick</h3>
<h2>Drama</h2>
<h3>Buddy Movies</h3>
<h3>Mystery</h3>
<h3>Romance</h3>
<h2>Horror</h2>
```

Paragraphs

As of the HTML 4.01 standard, paragraph tags are two-sided (<p>...</p>), and <p> indicates the beginning of the paragraph. The closing tag is no longer optional, so rather than using <p> to indicate where one paragraph ends and another begins, you enclose each paragraph within a <p> tag.

Input

```
<p>The dragon fell to the ground, releasing an anguished cry and seething in pain. The thrust of Enigern's sword proved fatal as the dragon breathed its last breath. Now Enigern was free to release Lady Aelfleada from her imprisonment in the dragon's lair.
</p>
```

Image:

Images displayed on the Web should be converted to one of the formats supported by most browsers: GIF, JPEG, or PNG. GIF and JPEG are the popular standards, and every graphical browser supports them. PNG is a newer image format that was created in response to some patent issue s with the GIF format.

The most important attribute of the <img> tag is src, which is the URL of the image you want to include. Paths to images are derived in the same way as the paths in the href attribute of links. So, to point to a GIF file named image.gif in the same directory as the HTML document, you can use the following HTML tag:

```
<img src="image.gif" />
```

Input:

```
<p><img src="house.jpg" alt="House of Terror" /></p>
<h1>Welcome to The Halloween House of Terror!!</h1>
```

Output:

Links:

To create a link in an HTML page, you use the HTML link tag <a>...</a>. The <a> tag often is called an anchor tag because it also can be used to create anchors for links.

Input

```
Go back to <a href="menu.html">
Main Menu</a>
```

Lists:

HTML 4.01 defines these three types of lists:

• Numbered or ordered lists, which are typically labeled with numbers

• Bulleted or unordered lists, which are typically labeled with bullets or some other symbol

• Glossary lists, in which each item in the list has a term and a definition for that term, arranged so that the term is somehow highlighted or drawn out from the text

List Tags

All the list tags have the following common elements:

• The entire list is surrounded by the appropriate opening and closing tag for the type of list (for example, <ul> and </ul> for unordered lists, or <ol> and </ol> for ordered lists).
• Each list item within the list has its own tag:
<dt> and <dd> for the glossary lists, and <li> for all the other lists.

Input

```
• <p>Installing Your New Operating System</p>
• <ol>
• <li>Insert the CD-ROM into your CD-ROM drive.</li>
• <li>Choose RUN.</li>
• <li>Enter the drive letter of your CD-ROM (example: D:\), followed by SETUP.EXE.</li>
• <li>Follow the prompts in the setup program.</li>
• <li>Reboot your computer after all files are installed.</li>
• <li>Cross your fingers.</li>
• </ol>
```

Customizing Ordered Lists

You can customize ordered lists in two main ways: how they're numbered and the number with which the list starts. HTML 3.2 provides the type attribute that can take one of five values to de fine which type of numbering to use on the list:

• "1" Specifies that standard Arabic numerals should be used to number the list (that is, 1, 2, 3, 4, and so on)
• "a" Specifies that lowercase letters should be used to number the list (that is, a, b, c, d, and so on)
• "A" Specifies that uppercase letters should be used to number the list (that is, A, B, C, D, and so on)
• "i" Specifies that lowercase Roman numerals should be used to number the list (that is, i, ii, iii, iv, and so on)
• "I" Specifies that uppercase Roman numerals should be used to number the list (that is, I, II, III, IV, and so on)

You can specify types of numbering in the <ol> tag, as follows: <ol type="a">. By default type="1" is assumed.

Input

```
<p>The Days of the Week in French:</p>
<ol type="I">
<li>Lundi</li>
<li>Mardi</li>
<li>Mercredi</li>
<li>Jeudi</li>
<li>Vendredi</li>
<li>Samedi</li>
<li>Dimanche</li>
</ol>
```

Input

```
<p>The Last Six Months of the Year (and the Beginning of the NextYear):</p>
<ol type="I" start="7">
<li>July</li>
<li>August</li>
```

```
<li>September</li>
<li>October</li>
<li>November</li>
<li>December</li>
<li type="1">January</li>
</ol>
```

Tables:

Table Parts

Before getting into the actual HTML code to create a table, let's look at the following terms so that we both know what we're talking about:

• The caption indicates what the table is about: for example, "Voting Statistics" or "Toy Distribution Per Room" Captions are optional.

• The table headings label the rows, columns, or both. Usually they're in an emphasized font that's different from the rest of the table. They're optional.

• Table cells are the individual squares in the table. A cell can contain normal table data or a table heading.

• Table data is the values in the table itself. The combination of the table headings and table data makes up the sum of the table.

The <table> Element

The to create a table in HTML, you use <table>...</table> element to enclose the code for an optional caption, and then add the contents of the table itself:

```
<table>
...table caption (optional) and contents...
</table>
```

Rows and Cells

The cells within each row are created using one of two elements:

• <th>...</th> elements are used for heading cells. Generally, browsers center the contents of a <th> cell and render any text in the cell in boldface.

• <td>...</td> elements are used for data cells. TD stands for table data.

Input

```
<tr>
<th>Name</th>
<td>Alison</td>
<td>Tom</td>
<td>Susan</td>
</tr>
<tr>
<th>Height</th>
<td>5'4"</td>
<td>6'0"</td>
<td>5'1"</td>
</tr>
<tr>
<th>Weight</th>
<td>140</td>
<td>165</td>
```

```
<td>97</td>
</tr>
<tr>
<th>Eye Color</th>
<td>Blue</td>
<td>Blue</td>
<td>Brown</td>
</tr>
```

Setting Table Widths

To make a table as wide as the browser window, you add the width attribute to the table, as shown in the following line of code:

```
<table border="1" width="100%">
```

Changing Table Borders

You can change the width of the border drawn around the table. If border has a numeric value, the border around the outside of the table is drawn with that pixel width. The default is border="1". border="0" suppresses the border, just as if you had omitted the border attribute altogether.

Input

```
<table border="10" width="100%">
```

Cell Padding

The cell padding attribute defines the amount of space between the edges of the cells and the content inside a cell.

Input

```
<table cellpadding="10" border="1">
```

Cell Spacing

Cell spacing is similar to cell padding except that it affects the amount of space between cells that is, the width of the space between the inner and outer lines that make up the table border.

Input

```
<table cellpadding="10" border="4" cellspacing="8">
```

Spanning Multiple Rows or Columns

The tables you've created up to this point all had one value per cell or the occasional empty cell. You also can create cells that span multiple rows or columns within the table. Those spanned cells then can hold headings that have subheadings in the next row or column or you can create other special effects within the table layout.

Input

```
<html>
<head>
<title>Row and Column Spans</title>
</head>
<body>
<table border="1" summary="span example">
<tr>
<th colspan="2">Gender</th>
</tr>
<tr>
<th>Male</th>
<th>Female</th>
```

```
</tr>
<tr>
<td>15</td>
<td>23</td>
</tr>
</table>
</body>
</html>
```

Forms:

Using the <form> Tag

To accept input from a user, you must wrap all of your input fields inside a <form> tag. The purpose of the <form> tag is to indicate where and how the user's input should be sent. First, let's look at how the <form> tag affects page layout. Forms are block-level elements.

Input

```
<p>Please enter your username <form><input /> and password <input /></form> to log in.</p>
```

The two most commonly used attributes of the <form> tag are action and method. Both of these attributes are optional. The following example shows how the <form> tag is typically used:

```
<form action="someaction" method="get or post">
content, form controls, and other HTML elements
</form>
```

action specifies the URL to which the form is submitted. Again, remember that for the form to be submitted successfully, the script must be in the exact location you specify and must work properly.

The method attribute supports two values: get or post. The method indicates how the form data should be packaged in the request that's sent back to the server. The get method appends the form data to the URL in the request.

Creating Text Controls

Text controls enable you to gather information from a user in small quantities. This control type creates a single-line text input field in which users can type information, such as their name or a search term.

Input

```
<p>Enter your pet's name:
<input type="text" name="petname" /></ p>
```

Creating Password Controls

The password and text field types are identical in every way except that the data entered in a password field is masked so that someone looking over the shoulder of the person entering information can't see the value that was typed into the field.

Input

```
<p>Enter your password: <input type="password" name="userpassword" size="8"
maxlength="8" /></p>
```

Creating Submit Buttons

Submit buttons are used to indicate that the user is finished filling out the form. Setting the type attribute of the form to submit places a submit button on the page with the default label determined by the browser, usually Submit Query. To change the button text, use the value attribute and enter your own label, as follows:

```
<input type="submit" value="Send Form Data" />
```

Creating Reset Buttons

Reset buttons set all the form controls to their default values. These are the values included in the value attributes of each field in the form (or in the case of selectable fields, the values that are preselected). As with the Submit button, you can change the label of a Reset button to one of your own choosing by using the value attribute, like this:

```
<input type="reset" value="Clear Form" />
```

Creating Check Box Controls

Check boxes are fields that can be set to two states: on and off. To create a check box, set the input tag's type attribute to checkbox. The name attribute is also required, as shown in the following example:

Input

```
<p>Check to receive SPAM email <input type="checkbox" name="spam" /></p>
```

Creating Radio Buttons

Radio buttons, which generally appear in groups, are designed so that when one button in the group is selected, the other buttons in the group are automatically unselected. They enable you to provide users with a list of options from which only one option can be selected. To create a radio button, set the type attribute of an <input> tag to radio. To create a radio button group, set the name attributes of all the fields in the group to the same value. To cre ate a radio button group with three options, the following code is used:

Input

```
<p>Select a color:<br />
<input type="radio" name="color" value="red" /> Red<br />
<input type="radio" name="color" value="blue" /> Blue<br />
<input type="radio" name="color" value="green" /> Green<br />
</p>
```

Creating Menus with select and option

The select element creates a menu that can be configured to enable users to select one or more options from a pull-down menu or a scrollable menu that shows several options at once. The <select> tag defines how the menu will be displayed and the name of the parameter associated with the field. The <option> tag is used to add selections to the menu. The default appearance of select lists is to display a pull-down list that enables the user to select one of the options. Here's an example of how one is created:

Input

```
<p>Please pick a travel destination:
<select name="location">
<option>Indiana</option>
<option>Fuji</option>
<option>Timbuktu</option >
<option>Alaska</option>
</select>
</p>
```

Frames for designing a good website:

The first HTML document you need to create is called the frameset document. In this document, you define the layout of your frames, and the locations of the documents to be initially loaded in each frame. Each of the three HTML documents other than the frameset document, the ones that load in the frames, contain normal HTML tags that define the contents of each separate frame area. These documents are referenced by the frameset document.

The <frameset> Tag

To create a frameset document, you begin with the <frameset> tag. When used in an HTML document, the <frameset> tag replaces the <body> tag, as shown in the following code:

```
<html>
<head>
<title>Page Title</title>
</head>
<frameset>
.. your frameset goes here ...
</frameset>
</html>
```

It's important that you understand up front how a frameset document differs from a normal HTML document. If you include a <frameset> tag in an HTML document, you cannot include a <body> tag also.

The cols Attribute

When you define a <frameset> tag, you must include one of two attributes as part of the tag definition. The first of these attributes is the cols attribute, which takes the following form:

```
<frameset cols="column width, column width, ...">
```

Input

```
<html>
<head>
<title>Three Columns</title>
</head>
<frameset cols="100,50%,*">
<frame src="leftcol.html">
<frame src="midcol.html">
<frame src="rightcol.html">
</frameset>
</html>
```

The rows Attribute

The rows attribute works the same as the cols attribute, except that it splits the screen into horizontal frames rather than vertical ones. To split the screen into two frames of equal height, you would write the following:

Input

```
<html>
<head>
<title>Two Rows</title>
</head>
<frameset rows="50%,50%">
<frame src="toprow.html">
<frame src="botrow.html">
</frameset>
```

```
</html>
```

The <frame> Tag

After you have your basic frameset laid out, you need to associate an HTML document with each frame by using the <frame> tag, which takes the following form:

```
<frame src="document URL">
```

For each frame defined in the <frameset> tag, you must include a corresponding <frame> tag, as shown in the following:

Input

```
<html>
<head>
<title>The FRAME Tag</title>
</head>
<frameset rows="*,*,*">
<frame src="document1.html" />
<frame src="document2.html" />
<frame src="document3.html" />
</frameset>
</html>
```

Changing Frame Borders

Start with the <frame> tag. By using two attributes, bordercolor and frameborder, you can turn borders on and off and specify their color. You can assign bordercolor any valid color value, either as a name or a hexadecimal triplet. frameborder takes two possible values:

1(to display borders) or 0 (to turn off the display of borders).

```
<html>
<head>
<title>Conflicting Borders</title>
</head>
<frameset frameborder="0" rows="*,*,*">
<frame frameborder="1" bordercolor="yellow" src="document1.html">
<frame bordercolor="#cc3333" src="document2.html">
<frame src="document3.html">
</frameset>
</html>
```

long questions:

1. What is internet? Want is WWW? What is the difference between them?
2. What are the different lists available explain briefly.
3. Explain the different tags and attributes available in table briefly.
4. What are the different tags available to create the elements of a form explain in detail.

Java Script, CSS and DOM

Java Script:

Programming Fundamentals:

JavaScript, originally called LiveScript, was developed by Brendan Eich at Netscape in 1995 and was shipped with Netscape Navigator 2.0 beta releases. JavaScript programs are used to detect and react to user-initiated events, such as a mouse going over a link or graphic. They can improve a Web site with navigational aids, scrolling messages and rollovers, dialog boxes, dynamic images, shopping carts, and so forth.

Client-side JavaScript programs are embedded in an HTML document between HTML head tags <head> and </head> or between the body tags <body> and </body>. Many developers prefer to put JavaScript code within the <head> tags, and at times, as you will see later, it is the best place to store function definitions and objects. If you want text displayed at a specific spot in the document, you may want to place the JavaScript code within the <body> tags. Or you may have multiple scripts within a page, and place the JavaScript code within both the <head> and <body> tags. In either case, a JavaScript program starts with a <script> tag, and and ends with a </script> tag. And if the JavaScript code is going to be long and involved, or may be reused, it can be placed in an external file (ending in .js) and loaded into the page.

```
<html>
<head><title>First JavaScript Sample</title></head>
<body bgcolor="yellow" text="blue">
<script language = "JavaScript" type="text/javascript">
document.writeln("<h2>Welcome to the JavaScript
World!</h1>");
</script>
<font size="+2">This is just plain old HTML stuff.</font>
</body>
</html>
```

Statements and Expressions:

Comments

Single line comments start with a double slash:

```
// This is a comment
```

For a block of comments, use the /* */ symbols:

```
/* This is a block of comments
that continues for a number of lines
*/
```

The <script> Tag

```
<script>
JavaScript statements...
</script>
```

```
<script>
document.write("Hello, world!<br>");
</script>
```

Attributes

The <script> tag also has attributes to modify the behavior of the tag. The attributes are

- language
- type
- src

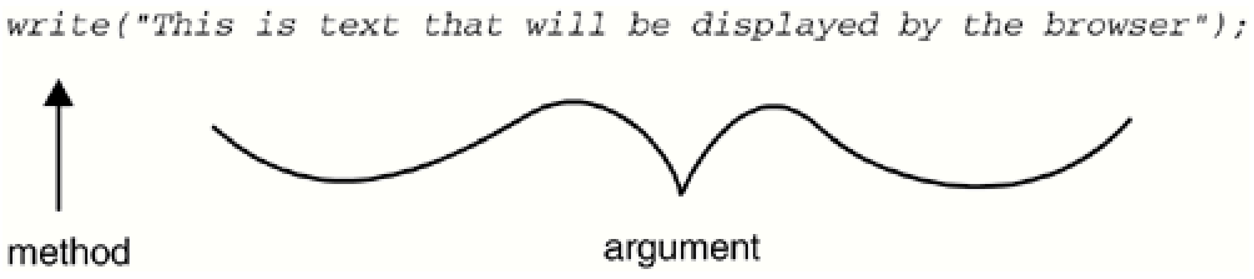
```
<script language="JavaScript"
type="text/javascript"
src="directory/sample.js">
</script>
```

String Concatenation

Concatenation is caused when two strings are joined together. The plus (+) sign is used to concatenate strings; for example:

"hot" + "dog or "San Francisco" + "</br>"

The write() and writeln() Methods



```
write("This is text that will be displayed by the browser");
```

```
<html>
<head><title>Printing Output</title></head>
<body bgcolor="yellow" text="blue">
<b>Comparing the <em>document.write</em> and <em>document.writeln
</em> methods</b><br>
<script language="JavaScript">
document.write("<h3>One, "); // No newline
document.writeln("Two, ");
document.writeln("Three, ");
document.write("Blast off... <br>"); // break tag
document.write("The browser you are using is " +
navigator.userAgent + "<br>");
```

```
</script>
<pre>
<script language="JavaScript">
document.writeln("With the <em>HTML &lt;pre&gt;
</em> tags, ");
document.writeln("the <em>writeln</em> method produces a newline.");
document.writeln("Slam");
document.writeln("Bang");
document.writeln("Dunk!");
</script>
</pre>
</body></html>
```

Data Types

Primitive Data Types

Primitive data types are the simplest building blocks of a program. They are types that can be assigned a single literal value such as the number 5.7, or a string of characters such as "hello". JavaScript supports three core or basic data types:

- numeric
- string
- Boolean

In addition to the three core data types, there are two other special types that consist of a single value:

- null
- undefined

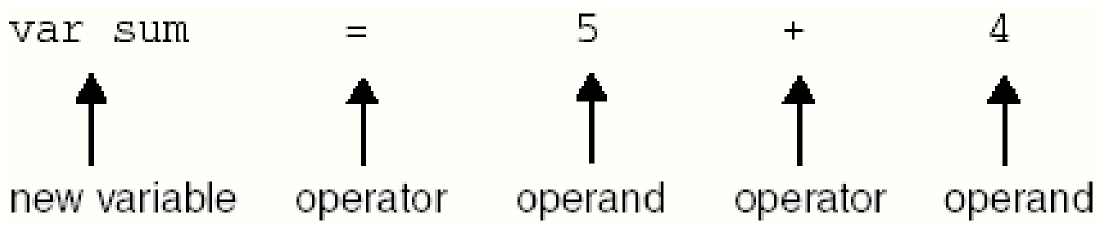
Variables

Variables are fundamental to all programming languages. They are data items that represent a memory storage location in the computer. Variables are containers that hold data such as numbers and strings. Variables have a name, a type, and a value. JavaScript variables can be assigned three types of data:

- numeric
- string
- Boolean

Operators:

Data objects can be manipulated in a number of ways by the large number of operators provided by JavaScript. Operators are symbols, such as +, –, =, >, and <, that produce a result based on some rules.



Precedence and associativity

Operator Description Associativity

() Parentheses Left to right

++ — Auto increment, decrement Right to left

! Logical NOT Right to left

* / % Multiply, divide, modulus Left to right

+ – Add, subtract Left to right

+ Concatenation Left to right

<<= Less than, less than equal to Left to right

>>= Greater than, greater than equal to Left to right

= = != Equal to, not equal to Left to right

= = = != = Identical to (same type), not identical to Left to right

&Bitwise AND Left to right

Operator Description Associativity

Bitwise OR

^ Bitwise XOR

~ Bitwise NOT

<< Bitwise left shift

>> Bitwise right shift

>>> Bitwise zero-filled, right shift

&& Logical AND Left to right

Logical OR Left to right

? : Ternary, conditional Right to left

= += – = *= /= %= <<= >>= Assignment Right to left

, (comma)

Types of Operators:

Arithmetic Operators

Arithmetic operators.

Operator/Operands	**Function**
x + y	Addition
x – y	Subtraction
x * y	Multiplication
x / y	Division
x % y	Modulus

Shortcut Assignment Operators

Assignment operators.

Operator Example Meaning

= var x = 5; Assign 5 to variable x.

+= x += 3; Add 3 to x and assign result to x.

–= x –= 2; Subtract 2 from x and assign result to x.

*= x *= 4; Multiply x by 4 and assign result to x.

/= x /= 2; Divide x by 2 and assign result to x.

**= x **= 2; Square x and assign result to x.

%= x %= 2 Divide x by 2 and assign remainder to x.

Auto increment and Auto decrement Operators

To make programs easier to read, to simplify typing, and, at the machine level, to produce more efficient code, the auto increment (++) and auto decrement (– –) operators are provided. Auto increment and auto decrement operators.

Operator Function What It Does Example

++x Pre-increment Adds 1 to x x = 3; x++; x is now 4

x++ Post-increment Adds 1 to x x = 3; ++x; x is now 4

– –x Pre-decrement Subtracts 1 from x x = 3; x – –; x is now 2

x– – Post-decrement Subtracts 1 from x x = 3; – –x; x is now 2

Concatenation Operator

As shown in previous examples, the + sign is used for concatenation and addition. The concatenation operator, the + sign, is a string operator used to join together one or more strings. In fact, the concatenation operator is the only operator JavaScript provides to manipulate strings.

Operator Example Meaning

+ "hot" + "dog" Concatenates (joins) two strings; creates "hotdog".

"22" + 8 Converts number 8 to string "8", then concatenates resulting in "228". In statements involving other operators, JavaScript does not convert numeric values to strings.

+= x ="cow"; x+= "boy"; Concatenates two strings and assigns the result to x; x becomes "cowboy".

Comparison Operators

Operator/Operands	**Function**
x == y	x is equal to y
x != y	x is not equal to y
x > y	x is greater than y
x >= y	x is greater than or equal to y
x < y	x is less than y
x <= y	x is less than or equal to y
x = = = y	x is identical to y in value and type
x != = y	x is not identical to y

Logical Operators

Operator/Operands Function

num1 && num2 True, if num1 and num2 are both true. Returns num1 if evaluated to false; otherwise returns num2. If operands are Boolean values, returns true if both operands are true; otherwise returns false.

num1 num2 True, if num1 is true or if num2 is true.

! num1 Not num1; true if num1 is false; false if num1 is true.

The Conditional Operator

FORMAT
conditional expression ? expression : expression

Examples:

x ?y : z If x evaluates to true, the value of the expression becomes y, else the value of the expression becomes z

big = (x > y) ? x : y If x is greater than y, x is assigned to variable big, else y is assigned to variable big

Bitwise Operators

Operator Function Example What It Does

&Bitwise AND x & y Returns a 1 in each bit position if both corresponding bits are 1. Bitwise OR x y Returns a 1 in each bit position if one or both corresponding bits are 1.

^ Bitwise XOR x ^ y Returns a 1 in each bit position if one, but not both, of the corresponding bits are 1.

– Bitwise NOT –x Inverts the bits of its operands. 1 becomes 0; 0 becomes 1.

<< Left shift x << y Shifts x in binary representation y bits to left, shifting in zeros from the right.

>> Right shift x >> y Shifts x in binary representation y bits to right, discarding bits shifted off.

>>> Zero-fill right x shift >>> b Shifts x in binary representation y bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Popup boxes:

JavaScript uses dialog boxes to interact with the user. The dialog boxes are created with three methods:

- alert()
- prompt()
- confirm()

The alert() Method

```
<html>
<head><title>Dialog Box</title></head>
<body bgcolor="yellow" text="blue">
```

```
<b>Testing the alert method</b><br>
<script language="JavaScript">
document.write("<font size='+2'>");
document.write("It's a bird, ");
document.write("It's a plane, <br>");
alert("It's Superman!");
</script>
</body></html>
```

The Prompt Box

Since JavaScript does not provide a simple method for accepting user input, the prompt dialog box and HTML forms are used. The prompt dialog box pops up with a simple text field box. After the user enters text into the prompt dialog box, its value is returned.

FORMAT

```
prompt(message);

prompt(message, defaultText);
```

Example:

```
prompt("What is your name? ", "");

prompt("Where is your name? ", name);
```

The Confirm Box

The confirm dialog box is used to confirm a user's answer to a question. A question mark will appear in the box with an OK button and a Cancel button. If the user presses the OK button, true is returned; if he presses the Cancel button, false is returned. This method takes only one argument, the question you will ask the user.

Example

```
<html>
<head>
<title>Using the JavaScript confirm box</title>
</head>
<body>
<script language = "JavaScript">
document.clear // Clears the page
if(confirm("Are you really OK?") == true){
alert("Then we can proceed!");
}
else{
alert("We'll try when you feel better? ");
}
</script>
</body>
</html>
```

Control Statements: Conditionals

```
if (condition){
statements;
}
```

Example:

```
if ( age > 21 ){
alert("Let's Party!"); }
```

if/else

```
if (condition){
statements1;
}
else{
statements2;
}
```

Example:

```
if ( x > y ){
alert( "x is larger");
}
else{
alert( "y is larger");
}
```

Example

```
<html>
<head>
<title>Conditional Flow Control</title>
</head>
<body>
<script language=javascript>
<!-- Hiding JavaScript from old browsers document.write("<h3>");
var age=prompt("How old are you? ","");
if( age >= 55 ){
document.write("You pay the senior fare! ");
}
else{
document.write("You pay the regular adult fare. ");
}
document.write("</h3>");
//-->
</script>
</body>
</html>
```

if/else if

```
if (condition) {
statements1;
}
else if (condition) {
statements2;
}
else if (condition) {
statements3;
}
else{
statements4;
}
```

Switch/case

```
switch (expression){
case label :
statement(s);
break;
case label :
statement(s);
break;
...
default : statement;
}
```

Example:

```
switch (color){
case "red":
alert("Hot!");
break;
case "blue":
alert("Cold.");
break;
default:
alert("Not a good choice.");
break;
}
```

Example

```
<html>
<head>
<title>The Switch Statement</title>
</head>
<body>
<script language=javascript>
```

```
<!--
var color=prompt("What is your color?","");
switch(color){
case "red":
document.bgColor="color";
document.write("Red is hot.");
break;
case "yellow":
document.bgColor=color;
document.write("Yellow is warm.");
break;
case "green":
document.bgColor="lightgreen";
document.write("Green is soothing.");
break;
case "blue":
document.bgColor="#RRGGBB";
document.write("Blue is cool.");
break;
default:
document.bgColor="white";
document.write("Not available today. We'll use white");
break;
}
//-->
</script>
</body>
</html>
```

Loops

Loops are used to execute a segment of code repeatedly until some condition is met. JavaScript's basic looping constructs are

- while
- for
- do/while

The while Loop

The while statement executes its statement block as long as the expression after the while evaluates to true; that is, non-null, non-zero, non-false. If the condition never changes and is true, the loop will iterate forever (infinite loop). If the condition is false control goes to the statement right after the closing curly brace of the loop's statement block. The break and continue functions are used for loop control.

```
while (condition) {
statements;
increment/decrement counter;
}
```

Example

```
<html>
<head>
<title>Looping Constructs</title>
</head>
<body>
<h2>While Loop</h2>
<script language="JavaScript">
document.write("<font size='+2'>");
vari=0; // Initialize loop counter
while ( i< 10 ){ // Test
document.writeln(i);
i++; // Increment the counter
} // End of loop block
</script>
</body>
</html>
```

The do/while Loop

The do/while statement executes a block of statements repeatedly until a condition becomes false. Owing to its structure, this loop necessarily executes the statements in the body of the loop at least once before testing its expression, which is found at the bottom of the block.

```
do
{ statements;}
while (condition);
```

Example

```
<html>
<head>
<title>Looping Constructs</title>
</head>
<body>
<h2>Do While Loop</h2>
<script language="JavaScript">
document.write("<font size='+2'>");
vari=0;
do{
document.writeln(i);
i++;
} while ( i< 10 )
</script>
</body>
</html>
```

The for Loop

The for loop consists of the for keyword followed by three expressions separated by semicolons and enclosed within parentheses. Any or all of the expressions can be omitted, but the two semicolons cannot. The first expression is used to set the initial value of variables and is executed just once, the second expression is used to test whether the loop should continue or stop, and the third expression updates the loop variables; that is, it increments or decrements a counter, which will usually determine how many times the loop is repeated.

```
for(Expression1;Expression2;Expression3)
{statement(s);}
for (initialize; test; increment/decrement)
{statement(s);}
```

```
<html>
<head>
<title>Looping Constructs</title>
</head>
<body>
<h2>For Loop</h2>
<script language="JavaScript">
document.write("<font size='+2'>");
for(vari = 0; i< 10; i++ ){
document.writeln(i);
}
</script>
</body>
</html>
```

Try… Catch and Throw statements:

Catching errors in JavaScript:

It is very important that the errors thrown must be catched or trapped so that they can be handled more efficiently and conveniently and the users can move better through the web page.

Using try…catch statement:

The try..catch statement has two blocks in it:

- try block
- 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.

The concept of try…catch statement shown in an example:

```
<html>
<head>
<script type="text/javascript">
try
{
document.write(junkVariable)
}
catch(err)
{
document.write(err.message + "<br/>")
}
</script>
</head>
<body>
</body>
</html>
```

The output of the above program is ‘junkVariable’ is undefined

In the above program, the variable *junkVariable*is undefined and the usage of this in try block gives an error. The control is transferred to the catch block with this error and this error message is printed in the catch block.

throw in JavaScript:

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)
```

exception can be any variable of type integer or boolean or string.

for example:

```
<html>
<head>
<script type="text/javascript">
try
{
varexfor=10
if(exfor!=20)
{
throw "PlaceError"
}
}
catch(err)
{
if(err == "PlaceError")
document.write ("Example to illustrate Throw
Statement: Variable exfor not equal to 20.
<br/>")
}
</script>
</head>
<body>
</body>
</html>
```

The output of the above program is:

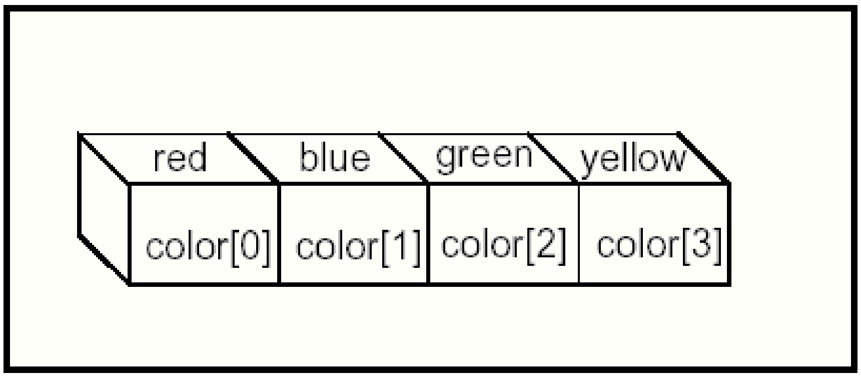
```
Example to illustrate Throw Statement: Variable exfor not equal to 20.
```

In the above example program, the try block has the variable exfor initialized to 10. Using the if statement, the variable value is checked to see whether it is equal to 20. Since exfor is not equal to 20, the exception is thrown using the throw statement. This is named Place Error and the control transfers to the catch block. The error catched is checked and since this is equal to the PlaceError, the statement placed inside the error message is displayed and the output is displayed as above.

Objects of java script:

Array Objects

An array is a collection of like values—called elements—such as an array of colors, an array of strings, or an array of images. Each element of the array is accessed with an index value enclosed in square brackets. An index is also called a subscript. There are two types of index values: a non-negative integer and a string. Arrays indexed by strings are called associative arrays. In JavaScript, arrays are built-in objects with some added functionality.



Declaring an Array

The following array is called array_name and its size is not specified.

vararray_name = new Array();

In the next example, the size or length of the array is passed as an argument to the Array() constructor. The new array has 100 undefined elements.

vararray_name = new Array(100);

And in the next example, the array is given a list of initial values of any data type:

vararray_name = new Array("red", "green", "yellow", 1 ,2, 3);

Example

```
<html>
<head><title>The Array Object</title>
<h2>An Array of Books</h2>
<script language="JavaScript">
var book = new Array(6); // Create an Array object
book[0] = "War and Peace"; // Assign values to its elements
book[1] = "Huckleberry Finn";
book[2] = "The Return of the Native";
book[3] = "A Christmas Carol";
book[4] = "The Yearling";
book[5] = "Exodus";
</script>
</head>
<body bgcolor="lightblue">
<script language="JavaScript">
document.write("<h3>");
for(vari in book){
document.write("book[" + i + "] "+ book[i] + "<br>");
}
</script>
</body>
</html>
```

Array Properties and Methods

Since an array is an object in JavaScript, it has properties to describe it and methods to manipulate it. The length of an array, for example, can be determined by the length property, and the array can be shortened by using the pop() method. For a complete list of array properties and methods

Array Object Properties

The Array object only has three properties. The most used is the length property which determines the number of elements in the array, that is, the size of the array.

Property What It Does

constructor References the object's constructor length Returns the number of elements in

the array prototype Extends the definition of the array by adding properties and methods

Array Methods

Whether you have an array of colors, names, or numbers, there are many ways you might want to manipulate the array elements. For example, you might want to add a new name or color to the beginning or end of the array, remove a number from the end of the array, or sort out all the elements, reverse the array, and so on. JavaScript provides a whole set of methods for doing all of these things and more

Method What It Does

concat() Concatenates elements from one array to another array join() Joins the elements of an array by a separator to form a string pop() Removes and returns the last element of an array push() Adds elements to the end of an array reverse() Reverses the order of the elements in an array sort() Sorts an array alphabetically, or numerically toString() Returns a string representation of the array

The Date Object

JavaScript provides the Date object for manipulating date and time. Like the String and Array objects, you can create as many instances as you like.

Example

var Date = new Date(); // The new constructor returns a Date object.
var Date = new Date("July 4, 2004, 6:25:22");
var Date = new Date("July 4, 2004");
var Date = new Date(2004, 7, 4, 6, 25, 22);
var Date = new Date(2004, 7, 4);
var Date = new Date(Milliseconds);
Using the Date Object Methods

Method	**What It Does**
getDate	Returns the day of the month (1–31)
getDay	Returns the day of the week (0–6); 0 is Sunday, 1 is Monday, etc.
getFullYear	Returns the year with 4 digits
getHours	Returns the hour (0–23)
getMilliseconds	Returns the millisecond
getMinutes	Returns hours since midnight (0–23)
getMonth	Returns number of month (0–11); 0 is January, 1 is February, etc.
getSeconds	Returns the second (0–59)
setDate(value)	Sets day of the month (1–31)
setFullYear()	Sets the year as a four-digit number
setHours()	Sets the hour within the day (0–23)
setHours(hr,min,sec,msec)	Sets hour in local
setMilliseconds	Sets the millisecond
setMinutes(min,sec, msec)	Sets minute in local time
setMonth(month,date)	Sets month in local time
setSeconds()	Sets the second
setTime()	Sets time from January 1, 1970, in milliseconds
setYear()	Sets the number of years since 1900 (00–99)
toGMTString()	Returns the date string in universal format
toString	Returns string representing date and time
valueOf()	Returns the equivalence of the Date object in milliseconds

Example

```
<html>
<head><title>Time and Date</title></head>
<body bgcolor="lightblue"><h2>Date and Time</h2>
<script language="JavaScript">
var now = new Date(); // Now is an instance of a Date object
document.write("<font size='+1'>");
document.write("<b>Local time:</b> " + now + "<br>");
var hours=now.getHours();
var minutes=now.getMinutes();
var seconds=now.getSeconds();
var year=now.getFullYear();
document.write("The full year is " + year +"<br>");
document.write("<b>The time is:</b> " +
hours + ":" + minutes + ":" + seconds);
document.write("</font>");
</script>
</body>
</html>
```

The Math Object

The Math object allows you to work with more advanced arithmetic calculations, such as square root, trigonometric functions, logarithms, and random numbers, than are provided by the basic numeric operators. If you are doing simple calculations, you really won't need it.

Math object methods.

Method	**Functionality**
Math.abs(Number)	Returns the absolute (unsigned) value of Number
Math.exp(x)	Euler's constant to some power (see footnote)
Math.floor(Number)	Rounds Number down to the next closest integer
Math.log(Number)	Returns the natural logarithm of Number (base E)
Math.max(Number1, Number2)	Returns larger value of Number1 and Number2
Math.min(Number1, Number2)	Returns smaller value of Number1 and Number2
Math.pow(x, y)	Returns the value of x to the power of y(x), where x is the base and y is the exponent
Math.random()	Generates pseudorandom number between 0.0 and 1.0
Math.round(Number)	Rounds Number to the closest integer
Math.sin(Number)	Arc sine of Number in radians
Math.sqrt(Number)	Square root of Number
Math.tan(Number)	Tangent of Number in radians
Math.toString(Number)	Converts Number to string

Example

```
<html>
<head><title>The Math Object</title></head>
<body>
<h2>Math object Methods--sqrt(),pow()<br>
Math object Property--PI</h2>
<P>
<script language="JavaScript">
varnum=16;
document.write("<h3>The square root of " +num+ " is ");
document.write(Math.sqrt(num),".<br>");
document.write("PI is ");
document.write(Math.PI);
document.write(".<br>"+num+" raised to the 3rd power is " );
document.write(Math.pow(num,3));
document.write(".</h3></font>");
</script>
</body></html>
```

The Boolean Object

The Boolean object was included in JavaScript 1.1. It is used to convert a non-Boolean value to a Boolean value, either true or false. There is one property, the prototype property, and one method, the toString() method, which converts a Boolean value to a string; thus, true is converted to "true" and false is converted to "false".

```
var object = new Boolean(value);
```

Example:

```
var b1 = new Boolean(5);
var b2 = new Boolean(null);
<html><head><title>Boolean Object</title>
</head>
<body bgcolor=aqua>
<font face="arial" size="+1"><b>
The Boolean Object<br>
<font size="-1">
<script language="JavaScript">
var bool1= new Boolean( 0);
var bool2 = new Boolean(1);
var bool3 = new Boolean("");
var bool4 = new Boolean(null);
var bool5 = new Boolean(NaN);
document.write("The value 0 is boolean "+ bool1 +"<br>");
document.write("The value 1 is boolean "+ bool2 +"<br>");
document.write("The value of the empty string is boolean "+ bool3+ "<br>");
document.write("The value of null is boolean "+ bool4+ "<br>");
document.write("The value of NaN is boolean "+ bool5 +"<br>");
</script>
</body></html>
```

Cascading Style Sheets:

Style can be delivered to a document by a variety of methods. The method with which style is connected with a document is referred to as *integration* . There are a variety of ways to integrate style, and how you decide to integrate style will depend largely upon what you are trying to accomplish with a specific document or number of documents.

Inline Style Sheets

The inline integration method allows you to take any tag and add a style to it. Using inline style gives you maximum control over a precise element of a web document, even just one character. Say you want to control the look and feel of a specific paragraph. You could simply add a style="x" attribute to the paragraph tag, and the browser would display that paragraph using the style values you added to the code.

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Inline Style Sample</title>
</head>
<body>
<h1 style="font-family: Arial" >Welcome!</h1>
</body>
</html>
```

Inline style is useful for getting precise control over something in a single document, but because it only applies to the element in question, you most likely won't be using inline style as frequently as other integration methods.

Internal Style Sheets

Embedding allows for control of a full document. Using the style element, which you place within the head section of a document, you can insert detailed style attributes to be applied to the entire page.
Embedding is an extremely useful way of styling individual pages that may also have other style methods influencing them. You can also style a single page or use multiple embedded sheets. The latter is especially useful if you'd like your document to have different styles for different media types.

Internal Style Sheet:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Embedded Style Sample</title>
<style type="text/css" media="screen">
h1 {
font: Arial;
}
</style>
</head>
<body>
<h1>Welcome!</h1>
</body></html>
```

As you can see, the style rule looks essentially the same as it did in the inline example, but it’s now applied via the style element as opposed to the style attribute. Unlike the inline example, which applied only to that specific h1, this rule will apply to all the h1 elements within the document, unless a class or inline style is applied.

External Style Sheets:

An external style sheet contains as many style rules as you like and helps to provide a most powerful means for you to create master styles that you can apply to one page or one billion pages.

An external style sheet is exactly that—all of the style is placed in an external file. You can link to the style sheet from any document you wish, using the link element in the head portion of those documents with which you’d like to integrate the style.

The external style document is a text document that you can write in any editor or tool that allows you to save a document as text. To create a linked style sheet, follow these steps:

1. Open your text or HTML editor of choice.
2. Enter the style rule or rules you’d like.

   ```
   h1 {
   font: Arial;
   }
   ```
3. Select File, Save and save your file with the name of h1style and a .css extension

   (style.css).

You’ll notice that the CSS file contains no additional information and tags. This is because an external style sheet is simply a list of style rules. You may also use style sheet commenting, but no declarations, elements, attributes, scripting, or other constructs should be in this document.

The next step is to link the document or documents you want to integrate with this style sheet:

1. In your document, place a link element within the head section. I’m using XHTML, so my link element, which is an empty element, uses the trailing slash, unlike HTML:

   ```
   <link />
   ```

2. Add the rel attribute, which describes the integration relationship type, in this case, a style sheet:

   ```
   <link rel="stylesheet" />
   ```

3. Add the type attribute and appropriate type, just as you would for an embedded sheet:

   ```
   <link rel="stylesheet" type="text/css" />
   ```

4. Include the media for which the sheet is intended. This can be any of the media types described earlier: print, screen, Braille, aural, and so on. In this instance, I'll use the screen value.

   ```
   <link rel="stylesheet" type="text/css" media="screen" />
   ```

5. Reference the source file using the href attribute and the location of the source file. In this case, both documents are residing in the same directory, so I'll reference it relatively:

   ```
   <link rel="stylesheet" type="text/css" media="all" href="style.css" />
   ```

6. Save your document as h1styletest.html.

The following code shows the complete XHTML document with the link element included.

The XHTML Document and External Style Sheet Are Now Integrated

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Linked Style Sample</title>
<link rel="stylesheet" type="text/css" media="screen" href="style.css" />
</head>
<body>
<h1>Welcome!</h1>
</body>
</html>
```

Class selectors

Working with HTML, authors may use the period (.) notation representing the class attribute. The attribute value must immediately follow the "period" (.).

For example, we can assign style information to all elements with class="pastoral" as follows:

.pastoral { color: green } / all elements with class=pastoral */ or just

.pastoral { color: green } /* all elements with class=pastoral */

The following assigns style only to H1 elements with class="pastoral":

```
H1.pastoral { color: green } /* H1 elements with class=pastoral */
```

Given these rules, the first H1 instance below would not have green text, while the second would:

```
<H1>Not green</H1>
<H1 class="pastoral">Very green</H1>
```

To match a subset of "class" values, each value must be preceded by a ".".

For example, the following rule matches any P element whose "class" attribute has been assigned a list of space-separated values that includes "pastoral" and "marine":

```
p.marine.pastoral{ color: green }
```

This rule matches when class="pastoral blue aqua marine" but does not match for class="pastoral blue".

CSS gives so much power to the "class" attribute, that authors could conceivably design their own "document language" based on elements with almost no associated presentation (such as DIV and SPAN in HTML) and assigning style information through the "class" attribute. Authors should avoid this practice since the structural elements of a document language often have recognized and accepted meanings and author-defined classes may not.

If an element has multiple class attributes, their values must be concatenated with spaces between the values before searching for the class. As of this time the working group is not aware of any manner in which this situation can be reached, however, so this behavior is explicitly non-normative in this specification.

Div and Span tag

Div

Div (short for division) divides the content into individual sections. Each section can then have its own formatting, as specified by the CSS. Div is a block-level container, meaning that there is a line feed after the </div> tag.
For example, if we have the following CSS declaration:

```
large {
color: #00FF00;
font-family:arial;
font-size: 4pt;
}
```

The HTML code

```
<div class="large">
This is a DIV sample.
</div>
```

gets displayed as This is a DIV sample.

Span

Span is similar to div in that they both divide the content into individual sections. The difference is that span goes into a finer level, so we can span to format a single character if needed. There is no line feed after the </span> tag.

For example, if we have the following CSS declaration:

```
.largefont {
color: #0066FF;
font-family:arial;
font-size: 6px;
}
```

The HTML code

Span is not at the <span class="largefont">block level</span>.

gets displayed as

block level

Span is not at the

DOM

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.
The DOM defines a standard for accessing documents like HTML and XML:
"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

What is the HTML DOM?

The HTML DOM is:

- A standard object model for HTML
- A standard programming interface for HTML
- Platform- and language-independent
- A W3C standard

The HTML DOM defines the **objects and properties** of all HTML elements, and the **methods** (interface) to access them.

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

DOM Nodes

According to the DOM, everything in an HTML document is a node.

The DOM says:

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

Text is Always Stored in Text Nodes

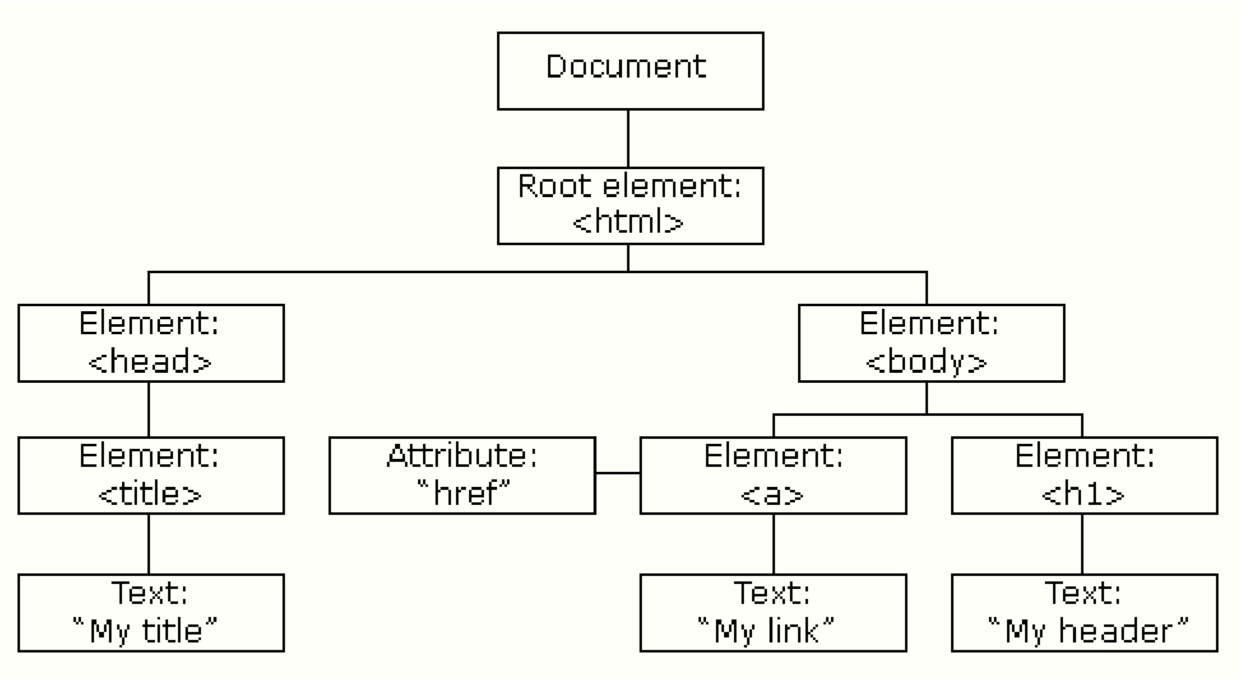
A common error in DOM processing is to expect an element node to contain text. However, the text of an element node is stored in a text node.

In this example: **<title>DOM Tutorial</title>**, the element node <title>, holds a text node with the value "DOM Tutorial".

"DOM Tutorial" is **not** the value of the <title> element!

The HTML DOM Node Tree

The HTML DOM views an HTML document as a tree-structure. The tree structure is called a **node-tree.** All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created. The node tree below shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:



Inner HTML

The most useful property of our web page objects that we can access is not a part of the official standard at all. The **innerHTML** property was introduced by Microsoft in Internet Explorer as a convenient way of being able to access the entire content of the HTML container all at once. It turned out to be so convenient that all of the other browsers quickly added support for this property.

We can use inner HTML either to retrieve the current content of the container or to insert new content into that container. Let's look at some examples. Here are a couple of div containers that we might have in our HTML.

```
<div id="first">
<p>Some text.</p>
<p>Some more text.</p></div>
<div id="second"></div>
```

The first of our example divs displays two paragraphs of text on the page while the second displays nothing on the page and is simply a placeholder. We can retrieve the content of the first div like this:

```
var content = document.getElementById('first') innerHTML;
```

The variable **content** now contains all of the text from the two paragraphs as well as the paragraph tags themselves. We can now replace those paragraphs completely by assigning a new value.

Dynamic HTML, or **DHTML**, is for a collection of technologies used together to create interactive and animated by using a combination of a static markup language (such as HTML), a client-side scripting language (such as JavaScript), a presentation definition language (such as CSS), and the Document Object Model.

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, *after* the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load.

By contrast, a dynamic web page is a broader concept — any web page generated differently for each user, load occurrence, or specific variable values. This includes pages created by client-side scripting, and ones created by server-side scripting (such as PHP, Perl, JSP or ASP.NET) where the web server generates content before sending it to the client.

Uses

DHTML allows authors to add effects to their pages that are otherwise difficult to achieve. For example, DHTML allows the page author to:

- Animate text and images in their document, independently moving each element from any starting point to any ending point, following a predetermined path or one chosen by the user.

- Embed a ticker that automatically refreshes its content with the latest news, stock quotes, or other data.
- Use a form to capture user input, and then process and respond to that data without having to send data back to the server.
- Include rollover buttons or drop-down menus.

DHTML FORMS

- Forms are key components of all Web-based applications. But important as they are, Web developers often present users with forms that are difficult to use. There are three common problems:
- Forms can be too long. A seemingly endless list of questions is sure to make the user click the back button or jump to another site.
- In many situations a specific user will need to fill out only some of the form elements. If you present needless questions to a user, you'll add clutter to your page and encourage the user to go elsewhere.
- Form entries often need to conform to certain formats and instructions. Adding this information to a Web page can make for a cluttered and unappealing screen.

Choosing The Form You Want

A long form can be shortened in a number of ways. If you have multiple versions of a form, the chief task becomes pointing people to the right form. Often, a simple set of links will do: "click here for the simple form, click here for the more complicated one." Alternatively, a single page can show one of several forms that the visitor can choose between using radio buttons.

This approach uses Dynamic HTML (DHTML), which has several benefits. First, DHTML allows for more flexible formatting. You can apply background images, borders, fonts, and all the other features you've learned to expect from HTML and Cascading Style Sheets to
DHTML objects. Second, if someone fills out one form, switches to another, then switches back, there's a good chance that the browser will lose the information that was initially entered. This problem doesn't exist in the DHTML solution. Third, with DHTML you can do tricky things like clipping and moving the form around the page.

What is the XML DOM?

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.
In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

DOM Nodes

According to the DOM, everything in an XML document is a **node**. The DOM says:

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

DOM Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="children">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="web">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
<author>Per Bothner</author>
<author>Kurt Cagle</author>
<author>James Linn</author>
<author>VaidyanathanNagarajan</author>
<year>2003</year>
<price>49.99</price>
</book>
<book category="web" cover="paperback">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

Text is Always Stored in Text Nodes

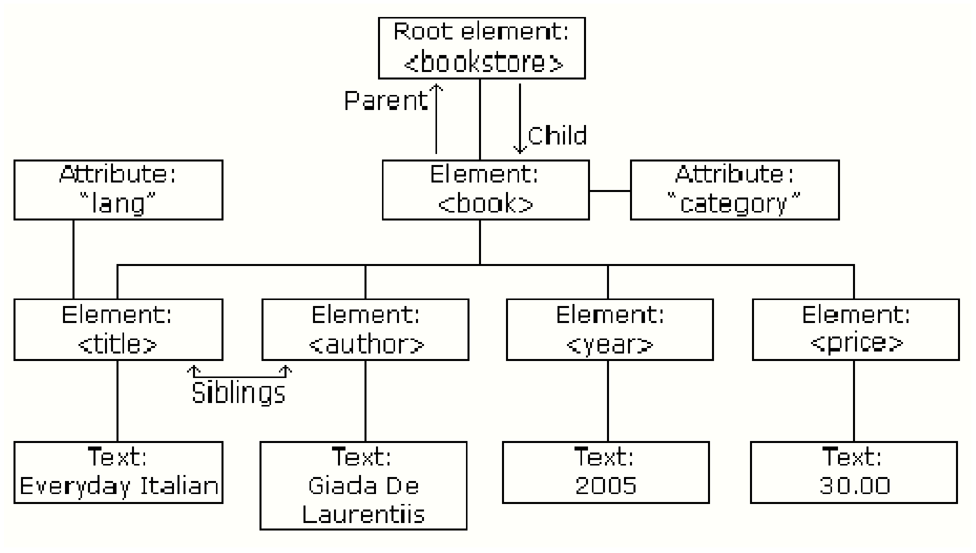
A common error in DOM processing is to expect an element node to contain text. However, the text of an element node is stored in a text node.

In this example: **<year>2005</year>**, the element node <year>, holds a text node with the value "2005".

"2005" is **not** the value of the <year> element!

The XML DOM Node Tree

The XML DOM views an XML document as a tree-structure. The tree structure is called **node-tree.** All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created. The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree:



CGI, PERL, Java Applet

Introduction to CGI:

The **Common Gateway Interface** (**CGI**) is a standard that defines how web server software can delegate the generation of web pages to a stand-alone application, an executable file. Such applications are known as *CGI scripts*; they can be written in any programming language, although scripting languages are often used.

The common gateway interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.

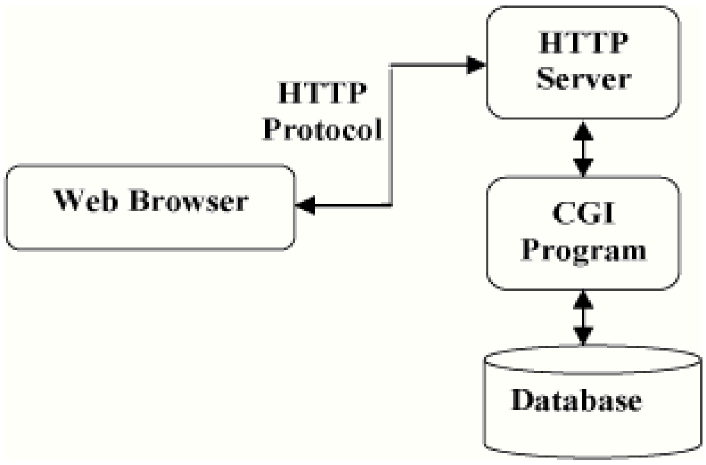
When the user requests a Web page (for example, by clicking on a hyperlink or entering a Web site address), the server sends back the requested page. However, when a user fills out a form on a Web page and sends it in, it usually needs to be processed by an application program. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This method or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI).

If you are creating a Web site and want a CGI application to get control, you specify the name of the application in the uniform resource locator (URL) that you code in an HTML file. This URL can be specified as part of the forms tags if you are creating a form. For example, you might code:

<form method="POST" action="http://www.mybiz.com/cgi-bin/formprog.pl">

and the server at "mybiz.com" would pass control to the CGI application called "formprog.pl" to record the entered data and return a confirmation message. (The ".pl" indicates a program written in Perl but other languages could have been used.)

The common gateway interface provides a consistent way for data to be passed from the user's request to the application program and back to the user. This means that the person who writes the application program can make sure it gets used no matter which operating system the server uses (Windows, Linux, Macintosh, UNIX, OS/390, or others). It's simply a basic way for information to be passed from the Web server about your request to the application program and back again.



Testing and debugging Perl CGI Scripts:

First CGI Program

Here is a simple link which is linked to a CGI script called hello.cgi. This file is being kept in /cgi-bin/ directory and it has following content.

```
#!/usr/bin/perl
print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello Word - First CGI Program</title>';
print '</head>';
print '<body>';
print '<h2>Hello Word! This is my first CGI program</h2>';
print '</body>';
print '</html>';
```

If you click hello.cgi then this produces following output:

Hello Word! This is my first CGI program

Using scalar variables and operators in Perl:

Variables, Arrays and Hashes

Technically speaking, Perl has three main variable types: scalars, arrays, and hashes. I will try to keep this as simple as possible.

Scalars

These are the variable that represents a single value. These variables always has a "$" (dollar sign) before it:

```
my $name = "Binny";
$phrase = "Ain't this cool";
$day = 12;
```

The 'my' keyword can be used if you want a local variable. Local variable is the variables that exist only in the block that it is defined in. It is safer to use 'my' while initializing variables.

Arrays

Array variables are variables that hold multiple values. All values can be accessed individually with its index. Now please remember this - arrays usually have a '@' symbol at the beginning when it is accessed as whole. But when declaring slots individually, you should use a '$' sign.

Hashes

Also known as associative arrays. A hash represents a set of key/value pairs. This is great for storing related information. It can be used as the array is used - but no index number is necessary - you can use a sting in its place. Below given is an example of hashes.

```
#!/usr/local/bin/perl
use strict;
print "Medical Dictionary for Rednecks and Blonds\n";
my %meanings = (Artery => "The study of painting",
Bacteria => "The back door of the cafeteria",
Barium => "What the doctors do when patients die",
Bowel => "A letter like A, E, I, O, or U",
Seizure => "A Roman emperor",
Tablet => "A small table",
Tumor => "More than one",
Urine => "Opposite of you're out"
);
```

print "Meaning of Artery is $meanings{'Artery'}, Tumor means $meanings{'Tumor'} while Seizure is $meanings{'Seizure'}";

Operators

There is no lack of operators in perl. Only the most important and the most used are listed below.

Note

: The variable $a has the value 3 and $b has the value 8. These values will NOT change - in other words, these values will be same for every example.

Arithmetic operators

Operator, explanation, example, example result

+ addition	$a + $b 11
- subtraction	5 - $a 2
* multiplication	3 * $b 24
/ division	6 / $a 2

Extras Operators

Operators, explanation, example, example result

++ Adds 1 to the value on the left (so if $i were equal to 1, $i++ would equal 2) $a++ $a will be 4

-- Subtracts 1 from the value on the left. $a-- $a will be $a +=

+= Adds the values to the left and right of the operator and then stores the value in the left variable.

$b; $a will be 11

Subtracts the values to the left and right of the -= operator and then stores the value in the left $b -= $a; $b will be 5 variable.

Multiplies the values to the left and right of the *= operator and then stores the value in the left $a *= $b; $a will be 24 variable.

Numeric comparison Operators

Operator Explanation Example Example Result

== equality if($a == $b) False

!= inequality if($a != $b) True

< less than if($a < $b) True

> greater than if($a > $b) False

<= less than or equal if($a <= $b) True

>= greater than or equal if($a >= $b) False

String comparison Operators

Operator Explanation Example Example Result

See whether two eq "String" eq "String" True **strings** are equal.

See whether two ne "String" ne "String" False **strings** are NOT equal.

Boolean logic

Operator Explanation Example Example Result

&& AND if($a>$b && $b>7) False

OR if($a>$b $b>7) True

! NOT if(!$b) False

Other operators

Operator Explanation Example Example Result

= assignment $a = 5; $a will become 5

$a = "Str1". string concatenation (In case you can't see the operator, its is a dot - a period - '.')

"Str2"; $a will be "Str1Str2" x string multiplication $a = "Me" x 5 $a will be "MeMeMeMeMe"

Perl And CGI(Common Gateway Interface)

Save the following the following into a file called guestbook.htm

```
<html><head>
<title>Guestbook</title>
</head>
<body>
<form action="/cgi-bin/guestbook.pl" method="get">
<table>
<tr><td>Name</td><td><input name="name" type="text" value=""></td></tr>
<tr><td>E-Mail</td><td><input name="email" type="text" value=""></td></tr>
<tr><td>Location</td><td><input name="loc" type="text" value=""></td></tr>
<tr><td>Comments</td><td>
<TEXTAREA name="comments" rows="10" cols="32"></TEXTAREA></td></tr>
</table><br><br>
<input type="submit" value="Add Entry">
</form>
</body>
</html>
```

See the <form action="/cgi-bin/guestbook.pl" method="get"> line? Of course you do. action="/cgi-bin/guestbook.pl" Tells the server where the CGI script is kept. method="get" tells the Server which input method is used. There is two input methods – get and post.

Now let’s create a perl script to get the input from this file.

Create a file called 'guestbook.pl' in the cgi-bin folder. Make sure that the above form points to this file in the action attribute.

First of all the mandatory first line.

```
#!/usr/local/bin/perl
```

Now lets get the input...

```
my $query_string = "";
#Get the input
if ($ENV{REQUEST_METHOD} eq 'POST') {
read(STDIN, $query_string, $ENV{CONTENT_LENGTH});
} else {
$query_string = $ENV{QUERY_STRING};
}
print "Content-Type: text/html\n\n";
print "Query String is \n<br> $query_string";
```

Get a server if you want to test you scripts. I use Sambar Server. This server is very easy to use and is very useful while testing your scripts.

Now put the perl file(save the above lines to a file called guestbook.pl) in the cgi-bin folder.

If you have sambar server it is usually C:\Program Files\Sambar\cgi-bin. Now copy the guestbook.htm file to the documents folder - the root folder(for sambar this is "C:\Program Files\Sambar\docs" by default). Now fire up your server and open the guestbook.htm file from within the server(For sambar server just open up a Internet

Browser like IE and type "http://127.0.0.1/guestbook.htm" in the address bar after opening the server by clicking the sambar shortcut in the start menu.)
If everything went well, you will see the html file we created here. Now enter the below given values
Name - Binny
E-Mail - whatever@wherever.com
Location - Omnipresent
Comments - Hello World! Here I am.
and press the "Add Entry" button.
This is the information that was passed between two files. Now the content part will look like this.
Query String is
name=Binny&email=whatever@wherever.com&loc=Omnipresent&comments=Hello+World %21+Here+I+am.
This is the data we got from the form. The data will be in this format - <Input name>=<Inputted Data>&<Next Input name>=<Next Inputted Data> and so on Input name stands for the name of the form element in the html file. If you take a look at it you will see that first element is name, next is email and so on. Now we need to convert this data to a more useful format.
Now we have created a perl file that will take input and give the output in HTML. You can customize the output page as you wish. You may like the following better. Replace the lines
"$FORM{'name'} came from $FORM{'loc'}. E-mail address is $FORM{'email'}. Comments :
$FORM{'comments'}" with the following.
Dear $FORM{'name'},<BR>Thank You for filling out our Guest Book.
I appreciate this effort in your part.<br><br>
<table>
<tr><td>Name</td><td>$FORM{'name'}</td></tr>
<tr><td>E Mail</td><td>
<a href="mailto:$FORM{'email'}">$FORM{'email'}</a></td></tr>
<tr><td>Location</td><td>$FORM{'loc'}</td></tr>
<tr><td>Comments</td><td>$FORM{'comments'}</td></tr>
</table>
Now our script is really cool - but it don't do the only thing that is expected of a Guest book program - Saving the result to a file. To do that we add the lines,
# Open Guest Book File
open (FILE, ">>guests.txt") die "Can't open guests.txt: $!\n";
#Write the information to the file
print FILE "$FORM{'name'} came from $FORM{'loc'}.";
print FILE "E-mail address is $FORM{'email'}.";
print FILE "Comments : $FORM{'comments'}\n";
close(FILE);
The script is finished. We have created a working guest book program. But I must warn you this script has its limitations. For a better guest book program that I have created, go to http://www.bin-co.com/perl/cgi/guestbook.html. For more guest books by others go to CGI Resources
The full script will look something like this...

```
#!/usr/local/bin/perl
my $query_string = '';
#Get the input
if ($ENV{REQUEST_METHOD} eq 'POST') {
read(STDIN, $query_string, $ENV{CONTENT_LENGTH});
} else {
$query_string = $ENV{QUERY_STRING};
}
# Split the name-value pairs
@pairs = split(/&/, $query_string);
foreach $pair (@pairs) {
($name, $value) = split(/=/, $pair);
# Making the input English. And removing unwanted things
$value =~ tr/+/ /;
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
$FORM{$name} = $value;
}
#Give output
print <<START;
Content-Type: text/html\n\n
<html><head>
<title>Guest book Result</title>
<body>
<h1 align="center">Guest book Results</h1>
Dear $FORM{'name'},<BR>Thank You for filling out our Guest Book.
I appreciate this effort in your part.<br><br>
<table>
<tr><td>Name</td><td>$FORM{'name'}</td></tr>
<tr><td>E Mail</td><td>
<a href="mailto:$FORM{'email'}">$FORM{'email'}</a></td></tr>
<tr><td>Location</td><td>$FORM{'loc'}</td></tr>
<tr><td>Comments</td><td>$FORM{'comments'}</td></tr>
</table>
</body>
</html>
START
# Open Guest Book File
open (FILE, ">>guests.txt") die "Can't open guests.txt: $!\n";
#Write the information to the file
print FILE "$FORM{'name'} came from $FORM{'loc'}.";
print FILE "E-mail address is $FORM{'email'}.";
print FILE "Comments : $FORM{'comments'}\n";
close(FILE);
```

Now you have a guest book. But it has its limitations and problems. Modify this guest book yourself. Make it better. Make it the best.

Java Applet

Introduction to Java

JAVA offers a number of advantages to developers.

Java is simple

Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages. The reason that why Java is much simpler than C++ is, because Java uses automatic memory allocation and garbage collection where else C++ requires the programmer to allocate memory and to collect garbage.

Java is object-oriented

Java is object-oriented because programming in Java is centred on creating objects, manipulating objects, and making objects work together. This allows you to create modular programs and reusable code.

Java is platform-independent

One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Java is distributed

Distributed computing involves several computers on a network working together. Java is designed to make distributed computing easy with the networking capability that is inherently integrated into it. Writing network programs in Java is like sending and receiving data to and from a file. For example, the diagram below shows three programs running on three different systems, communicating with each other to perform a joint task.

Java is interpreted

An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.

Java is secure

Java is one of the first programming languages to consider security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind.

Java is robust

Robust means reliable and no programming language can really assure reliability. Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages.

Java is multithreaded

Multithreaded is the capability for a program to perform several tasks simultaneously within a program. In Java, multithreaded programming has been smoothly integrated into it, while in other languages, operating system-specific procedures have to be called in order to enable multithreading. Multithreading is a necessity in visual and network programming.

Writing java Applets

Java applet is an applet delivered to users in the form of Java bytecode. Java applets can run in a Web browser using a Java Virtual Machine (JVM), or in Sun's AppletViewer, a standalone tool for testing applets.

Java applets are executed in a *sandbox* by most web browsers, preventing them from accessing local data like clipboard or file system. The code of the applet is downloaded from a web server and the browser either embeds the applet into a web page or opens a new window showing the applet's user interface.

A Java applet extends the class java.applet. Applet, or in the case of a Swing applet, javax.swing.JApplet. The class must override methods from the applet class to set up a user interface inside itself (Applet is a descendant of Panel which is a descendant of Container.

As applet inherits from container, it has largely the same user interface possibilities as an ordinary Java application, including regions with user specific visualization. The applet can be displayed on the web page by making use of the deprecated applet HTML element, or the recommended object element.

Example

The following example is made simple enough to illustrate the essential use of Java applets through its java.applet package. It also uses classes from the Java Abstract Window Toolkit (AWT) for producing actual output (in this case, the "Hello, world!" message).

```
import java.applet.Applet;
import java.awt.*;
// Applet code for the "Hello, world!" example.
// This should be saved in a file named as "HelloWorld.java".
public class HelloWorld extends Applet {
// This method is mandatory, but can be empty (i.e., have no actual code).
public void init() { }
// This method is mandatory, but can be empty.(i.e.,have no actual code).
public void stop() { }
// Print a message on the screen (x=20, y=10).
public void paint(Graphics g) {
g.drawString("Hello, world!", 20,10);
// Draws a circle on the screen (x=40, y=30).
g.drawArc(40,30,20,20,0,360);
}
}
```

For compiling, this code is saved on a plain-ASCII file with the same name as the class and .java extension, i.e. HelloWorld.java. The resulting HelloWorld.class applet should be placed on the web server and is invoked within an HTML page by using an <APPLET> or an <OBJECT> tag. For example:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>HelloWorld_example.html</TITLE>
</HEAD>
<BODY>
<H1>A Java applet example</H1>
```

```
<P>Here it is: <APPLET code="HelloWorld.class" WIDTH="200" HEIGHT="40">
This is where HelloWorld.class runs.</APPLET></P>
</BODY>
</HTML>
```

Displaying the HelloWorld_example.html page from a Web server, the result should look as this:

A Java applet example

Here it is: Hello, world!

Advantages

A Java applet can have any or all of the following advantages:

- It is simple to make it work on Linux, Microsoft Windows and Mac OS X i.e. to make it cross platform. Applets are supported by most web browsers.
- The same applet can work on "all" installed versions of Java at the same time, rather than just the latest plug-in version only. However, if an applet requires a later version of the Java Runtime Environment (JRE) the client will be forced to wait during the large download.
- Most web browsers cache applets, so will be quick to load when returning to a web page. Applets also improve with use: after a first applet is run, the JVM is already running and starts quickly (the JVM will need to restart each time the browser starts afresh).
- It can move the work from the server to the client, making a web solution more scalable with the number of users/clients.
- If a standalone program (like Google Earth) talks to a web server, that server normally needs to support all prior versions for users which have not kept their client software updated. In contrast, a properly configured browser loads (and caches) the latest applet version, so there is no need to support legacy versions.
- The applet naturally supports the changing user state, such as figure positions on the Chessboard
- Developers can develop and debug an applet direct simply by creating a main routine (either in the applet's class or in a separate class) and calling init() and start() on the applet, thus allowing for development in their favorite Java SE development environment. All one has to do after that is re-test the applet in the AppletViewer program or a web browser to ensure it conforms to security restrictions.
- An untrusted applet has no access to the local machine and can only access the server it came from. This makes such an applet much safer to run than a standalone executable that it could replace. However, a signed applet can have full access to the machine it is running on if the user agrees.
- Java applets are fast - and can even have similar performance to native installed software.

Life cycle of An Applet

Introduction

In this Section you will learn about the lifecycle of an applet and different methods of an applet. Applet runs in the browser and its lifecycle method are called by JVM when it is loaded and destroyed. Here are the lifecycle methods of an Applet:

init(): This method is called to initialized an applet

start(): This method is called after the initialization of the applet.

stop(): This method can be called multiple times in the life cycle of an Applet.

destroy(): This method is called only once in the life cycle of the applet when applet is destroyed.

init () method:

The life cycle of an applet is begin on that time when the applet is first loaded into the browser and called the init() method. The init() method is called only one time in the life cycle on an applet. The init() method is basically called to read the PARAM tag in the html file. The init () method retrieve the passed parameter through the PARAM tag of html file using get Parameter() method All the initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method .After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method.

Start () method:

The start method of an applet is called after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted. For Example if the user wants to return to the Applet, in this situation the start Method() of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.

Stop () method:

The stop() method can be called multiple times in the life cycle of applet like the start () method. Or should be called at least one time. There is only miner difference between the start() method and stop () method. For example the stop() method is called by the web browser on that time When the user leaves one applet to go another applet and the start() method is called on that time when the user wants to go back into the first program or Applet.

destroy() method:

The destroy() method is called only one time in the life cycle of Applet like init() method. This method is called only on that time when the browser needs to Shutdown.

Applet versus Application

Applets as previously described, are the small programs while applications are larger programs. Applets don't have the main method while in an application execution starts with the main method. Applets can run in our browser's window or in an appletviewer. To run the applet in an appletviewer will be an advantage for debugging. Applets are designed for the client site programming purpose while the applications don't have such type of criteria. Applet are the powerful tools because it covers half of the java language picture. Java applets are the best way of creating the programs in java. Applets are designed just for handling the client site problems. While the java applications are designed to work with the client as well as server. Applications are designed to exist in a secure area while the applets are typically used. Applications and applets have much of the similarity such as both have most of the same features and share the same resources. Applets are created by extending the java.applet.Applet class while the java applications start execution from the main method.

Applications are not too small to embed into a html page so that the user can view the application in your browser. On the other hand applet have the accessibility criteria of the resources. The key feature is that while they have so many differences but both can perform the same purpose.

Review of Java Applets:

To create an applet just create a class that extends the java.applet.Applet class and inherit all the features available in the parent class. The following programs make all the things clear.

```
import java.awt.*;
import java.applet.*;
class Myclass extends Applet {
public void init() {
/* All the variables, methods and images initialize here
will be called only once because this method is called only
once when the applet is first initializes */
}
public void start() {
/* The components needed to be initialize more than once
in your applet are written here or if the reader
switches back and forth in the applets. This method
can be called more than once.*/
}
public void stop() {
/* This method is the counterpart to start(). The code,
used to stop the execution is written here*/
}
public void destroy() {
/* This method contains the code that result in to release the resources to the applet before
it is finished. This method is called only once. */ }
public void paint(Graphics g) {
/* Write the code in this method to draw, write, or color
things on the applet pane are */
}
}
```

In the above applet you have seen that there are five methods. In which two (init() and destroy) are called only once while remaining three (start() , stop() , and paint()) can be called any number of times as per the requirements. The major difference between the two (applet and application) is that java applications are designed to work under the homogenous and more secure areas. On contrary to that, java applets are designed to run the heterogeneous and probably unsecured environment. Internet has imposed several restrictions on it.

Applets are not capable of reading and writing the user's file system. This means that the applet neither can access nor place anything locally. One more thing to point here is that applets are unable to use the native methods, run any program on the user system or load shared libraries. The major security concern here is that the local shared libraries and the native methods may results in the loophole in the java security model.

Applets are not capable of communicating the server than one from which they are originating. There are the cases in which an encryption key is used for the verification purpose for a particular applet to a server. But accessing a remote server is not possible.

The conclusion is that the java applets provides a wide variety of formats for program execution and a very tight security model on the open environment as on the Internet.

Introduction to Java Application:

Java applications have the majority of differences with the java applets. If we talk at the source code level, then we don't extend any class of the standard java library that means we are not restricted to use the already defined method or to override them for the execution of the program. Instead we make set of classes that contains the various parts of the program and attach the main method with these classes for the execution of the code written in these classes. The following program illustrates the structure of the java application.

```
public class MyClass {
/* Various methods and variable used by the class
MyClass are written here */
class myClass {
/* This contains the body of the class myClass */
}
public static void main(String args[]) {
/* The application starts it's actual execution from this place. **/
}
}
```

The main method here is nothing but the system method used to invoke the application.
The code that results an action should locate in the main method. Therefore this method is more than the other method in any java application. If we don't specify the main method in our application, then on running the application will through an exception like this one: In the class MyClass: void main(String args[]) is undefined But at higher level major concern is that in a typical java application security model, an application can access the user's file system and can use native methods. On properly configuring the user's environment and the java application it will allow access to all kind of stuff from the Internet. In most of the cases it is seen that the java application seems like a typical C/C++ application. Now we are going to create plenty of applications to exemplify some of the methods and features of a specific Java application.