

Subject Name: Web Technologies
Subject code: CSE71
Semester : SEVENTH

FUNDAMENTALS OF WEB, XHTML – 1

A BRIEF INTRODUCTION ABOUT THE INTERNET

Origins:

- **1960s**
 - U.S. Department of Defence (DoD) became interested in developing a new large-scale computer network
 - The purposes of this network were communications, program sharing, and remote computer access for researchers working on defence-related contracts.
 - The DoD's Advanced Research Projects Agency (ARPA) funded the construction of the first such network. Hence it was named as ARPAnet.
 - The primary early use of ARPAnet was simple text-based communications through e-mail.
- **late 1970s and early 1980s**
 - BITNET, which is an acronym for *Because It's Time NETwork*, began at the City University of New York. It was built initially to provide electronic mail and file transfers.
 - CSNET is an acronym for *Computer Science NETwork*. Its initial purpose was to provide electronic mail.
- **1990s**
 - NSFnet which was created in 1986 replaced ARPAnet by 1990.
 - It was sponsored by the National Science Foundation (NSF).
 - By 1992 NSFnet, connected more than 1 million computers around the world.
 - In 1995, a small part of NSFnet returned to being a research network. The rest became known as the *Internet*.

What Is the Internet?

- The Internet is a huge collection of computers connected in a communications network.
- The Transmission Control Protocol/Internet Protocol (TCP/IP) became the standard for computer network connections in 1982.
- Rather than connecting every computer on the Internet directly to every other computer on the Internet, normally the individual computers in an organization are connected to each other in a local network. One node on this local network is physically connected to the Internet.
- So, the Internet is actually a ***network of networks***, rather than a network of computers.
- Obviously, all devices connected to the Internet must be uniquely identifiable.

Internet Protocol Addresses

- The Internet Protocol (IP) address of a machine connected to the Internet is a unique 32-bit number.
- IP addresses usually are written (and thought of) as four 8-bit numbers, separated by periods.
- The four parts are separately used by Internet-routing computers to decide where a message must go next to get to its destination.
- Although people nearly always type domain names into their browsers, the IP works just as well.
- For example, the IP for United Airlines (www.ual.com) is 209.87.113.93. So, if a browser is pointed at <http://209.87.113.93>, it will be connected to the United Airlines Web site.

Domain Names

The IP addresses are numbers. Hence, it would be difficult for the users to remember IP address. To solve this problem, text based names were introduced. These are technically known as ***domain name system (DNS)***.

These names begin with the names of the host machine, followed by progressively larger enclosing collection of machines, called **domains**. There may be two, three or more domain names.

DNS is of the form **hostname.domainName.domainName** . Example: **New**

The steps for conversion from DNS to IP:

- The DNS has to be converted to IP address before destination is reached.
- This conversion is needed because computer understands only numbers.
- The conversion is done with the help of *name server*.
- As soon as domain name is provided, it will be sent across the internet to contact name servers.
- This name server is responsible for converting domain name to IP
- If one of the *name servers* is not able to convert DNS to IP, it contacts other name server.
- This process continues until IP address is generated.
- Once the IP address is generated, the host can be accessed.
- The hostname and all domain names form what is known as FULLY QUALIFIED DOMAIN NAME.

This is as shown below:

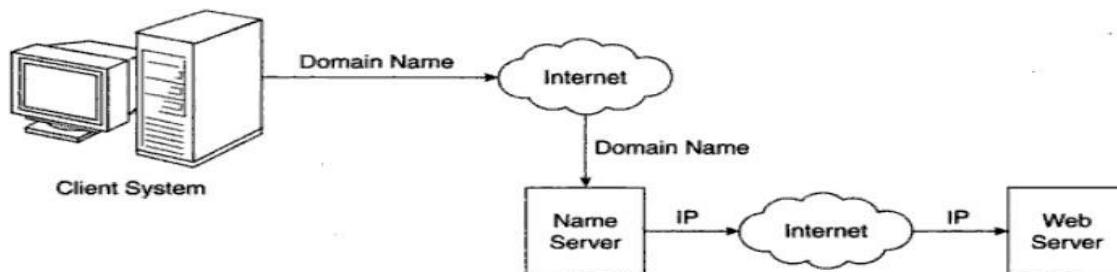


Figure 1.1 Domain name conversion

The World Wide Web

Origins

- Tim Berners Lee and his group proposed a new protocol for the Internet whose intention was to allow scientists around the world to use the Internet to exchange documents describing their work.
- The proposed new system was designed to allow a user anywhere on the Internet to search for and retrieve documents from the databases on any number of different document-serving computers.
- The system used *hypertext*, which is text with embedded links to text in other documents to allow non-sequential browsing of textual material.
- The units of web are referred as pages, documents and resources.
- Web is merely a vast collection of documents, some of which are connected by links.
- These documents can be accessed by web browsers and are provided by web servers.

Web or Internet?

It is important to understand that the Internet and the Web is not the same thing.

- ✓ The **Internet** is a collection of computers and other devices connected by equipment that allows them to communicate with each other.
- ✓ The **Web** is a collection of software and protocols that has been installed on most, if not all, of the computers on the Internet.

WEB BROWSERS

- Documents provided by servers on the Web are requested by **browsers**, which are programs running on client machines.

- They are called browsers because they allow the user to browse the resources available on servers.
- Mosaic was the first browser with a graphical user interface.
- A browser is a client on the Web because it initiates the communication with a server, which waits for a request from the client before doing anything.
- In the simplest case, a browser requests a static document from a server.
- The server locates the document among its servable documents and sends it to the browser, which displays it for the user.
- Sometimes a browser directly requests the execution of a program stored on the server. The output of the program is then returned to the browser.
- Examples: Internet Explorer, Mozilla Firefox, Netscape Navigator, Google Chrome, Opera etc.,

WEB SERVERS

Web servers are programs that provide documents to requesting browsers. Example: Apache

Web server operations:

- All the communications between a web client and a web server use the HTTP
- When a web server begins execution, it informs the OS under which it is running & it runs as a background process
- A web client or browser, opens a network connection to a web server, sends information requests and possibly data to the server, receives information from the server and closes the connection.
- The primary task of web server is to monitor a communication port on host machine, accept HTTP commands through that port and perform the operations specified by the commands.
- When the URL is received, it is translated into either a filename or a program name.

General characteristics of web server:

- The file structure of a web server has two separate directories
- The root of one of these is called **document root** which stores web documents
- The root of the other directory is called the **server root** which stores server and its support softwares
- The files stored directly in the document root are those available to clients through top level URLs
- The secondary areas from which documents can be served are called **virtual document trees**.
- Many servers can support more than one site on a computer, potentially reducing the cost of each site and making their maintenance more convenient. Such secondary hosts are called **virtual hosts**.
- Some servers can serve documents that are in the document root of other machines on the web; in this case they are called as **proxy servers**

Apache

- ▶ Apache is the most widely used Web server.
- ▶ The primary reasons are as follows: Apache is an excellent server because it is both fast and reliable.
- ▶ Furthermore, it is open-source software, which means that it is free and is managed by a large team of volunteers, a process that efficiently and effectively maintains the system.
- ▶ Finally, it is one of the best available servers for Unix-based systems, which are the most popular for Web servers.
- ▶ Apache is capable of providing a long list of services beyond the basic process of serving documents to clients.
- ▶ When Apache begins execution, it reads its configuration information from a file and sets its parameters to operate accordingly.

IIS

- ▶ Microsoft IIS server is supplied as part of Windows—and because it is a reasonably good server—most Windows-based Web servers use IIS.
- ▶ With IIS, server behaviour is modified by changes made through a window-based management program, named the IIS snap-in, which controls both IIS and ftp.

- ▶ This program allows the site manager to set parameters for the server.
- ▶ Under Windows XP and Vista, the IIS snap-in is accessed by going to *Control Panel, Administrative Tools, and IIS Admin.*

UNIFORM RESOURCE LOCATORS

- Uniform Resource Locators (URLs) are used to identify different kinds of resources on Internet.
- If the web browser wants some document from web server, just giving domain name is not sufficient because domain name can only be used for locating the server.
- It does not have information about which document client needs. Therefore, URL should be provided.
- The general format of URL is: **scheme: object-address**
- Example: <http://www.vtu.ac.in/results.php>
- The scheme indicates protocols being used. (http, ftp, telnet...)
- In case of http, the full form of the object address of a URL is as follows:
//fully-qualified-domain-name/path-to-document
- URLs can never have embedded spaces
- It cannot use special characters like semicolons, ampersands and colons
- The path to the document for http protocol is a sequence of directory names and a filename, all separated by whatever special character the OS uses. (forward or backward slashes)
- The path in a URL can differ from a path to a file because a URL need not include all directories on the path
- A path that includes all directories along the way is called a **complete path**.
- Example: <http://www.New Horizon.ac.in/index.html>
- In most cases, the path to the document is relative to some base path that is specified in the configuration files of the server. Such paths are called **partial paths**.
- Example: <http://www.New Horizon.ac.in/>

MULTIPURPOSE INTERNET MAIL EXTENSIONS

- MIME stands for **Multipurpose Internet Mail Extension**.
- The server system apart from sending the requested document, it will also send MIME information.
- The MIME information is used by web browser for rendering the document properly.
- The format of MIME is: **type/subtype**
- Example: text/html , text/doc , image/jpeg , video/mpeg
- When the type is either text or image, the browser renders the document without any problem
- However, if the type is video or audio, it cannot render the document
- It has to take the help of other software like media player, winamp etc.,
- These softwares are called as **helper applications or plugins**
- These non-textual information are known as **HYPER MEDIA**
- Experimental document types are used when user wants to create a customized information & make it available in the internet
- The format of experimental document type is: **type/x-subtype**
- Example: database/x-xbase , video/x-msvideo
- Along with creating customized information, the user should also create helper applications.
- This helper application will be used for rendering the document by browser.
- The list of MIME specifications is stored in configuration file of web server.

THE HYPERTEXT TRANSFER PROTOCOL

Request Phase:

The general form of an HTTP request is as follows:

1. HTTP method Domain part of the URL HTTP version
2. Header fields
3. Blank line
4. Message body

The following is an example of the first line of an HTTP request:

```
GET /storefront.html HTTP/1.1
```

Table 1.1 HTTP request methods

Method	Description
GET	Returns the contents of the specified document
HEAD	Returns the header information for the specified document
POST	Executes the specified document, using the enclosed data
PUT	Replaces the specified document with the enclosed data
DELETE	Deletes the specified document

The format of a header field is the field name followed by a colon and the value of the field. There are four categories of header fields:

1. **General:** For general information, such as the date
2. **Request:** Included in request headers
3. **Response:** For response headers
4. **Entity:** Used in both request and response headers

A wildcard character, the asterisk (*), can be used to specify that part of a MIME type can be anything.

Accept: text/plain
Accept: text/html



Accept: text/*

The **Host:** *host name* request field gives the name of the host. The **Host** field is required for HTTP 1.1. The **If-Modified-Since:** *date* request field specifies that the requested file should be sent only if it has been modified since the given date. If the request has a body, the length of that body must be given with a **Content-length** field. The header of a request must be followed by a blank line, which is used to separate the header from the body of the request.

The Response Phase:

The general form of an HTTP response is as follows:

1. Status line
2. Response header fields
3. Blank line
4. Response body

The status line includes the HTTP version used, a three-digit status code for the response, and a short textual explanation of the status code. For example, most responses begin with the following:

```
HTTP/1.1 200 OK
```

The status codes begin with 1, 2, 3, 4, or 5. The general meanings of the five categories specified by these first digits are shown in **Table 1.2**.

Table 1.2 First digits of HTTP status codes

First Digit	Category
1	Informational
2	Success
3	Redirection
4	Client error
5	Server error

One of the more common status codes is one user never want to see: 404 Not Found, which means the requested file could not be found.

SECURITY

Security is one of the major concerns in the Internet. The server system can be accessed easily with basic hardware support, internet connection & web browser. The client can retrieve very important information from the server. Similarly, the server system can introduce virus on the client system. These viruses can destroy the hardware and software in client.

While programming the web, following requirements should be considered:

- **Privacy:** it means message should be readable only to communicating parties and not to intruder.
- **Integrity:** it means message should not be modified during transmission.
- **Authentication:** it means communicating parties must be able to know each other's identity
- **Non-repudiation:** it means that it should be possible to prove that message was sent and received properly

Security can be provided using cryptographic algorithm. Ex: private key, public key

Protection against viruses and worms is provided by antivirus software, which must be updated frequently so that it can detect and protect against the continuous stream of new viruses and worms.

THE WEB PROGRAMMER'S TOOLBOX

Web programmers use several languages to create the documents that servers can provide to browsers.

- ▶ The most basic of these is **XHTML**, the standard mark-up language for describing how Web documents should be presented by browsers. Tools that can be used without specific knowledge of XHTML are available to create XHTML documents.
- ▶ A **plug-in** is a program that can be integrated with a word processor to make it possible to use the word processor to create XHTML. A **filter** converts a document written in some other format to XHTML.
- ▶ **XML** is a meta-mark-up language that provides a standard way to define new mark-up languages.
- ▶ **JavaScript** is a client-side scripting language that can be embedded in XHTML to describe simple computations. JavaScript code is interpreted by the browser on the client machine; it provides access to the elements of an XHTML document, as well as the ability to change those elements dynamically.
- ▶ **Flash** is a framework for building animation into XHTML documents. A browser must have a Flash player plug-in to be able to display the movies created with the Flash framework.
- ▶ **Ajax** is an approach to building Web applications in which partial document requests are handled asynchronously. Ajax can significantly increase the speed of user interactions, so it is most useful for building systems that have frequent interactions.
- ▶ **PHP** is the server-side equivalent of JavaScript. It is an interpreted language whose code is embedded in XHTML documents. PHP is used primarily for form processing and database access from browsers.
- ▶ **Servlets** are server-side Java programs that are used for form processing, database access, or building dynamic documents. JSP documents, which are translated into servlets, are an alternative approach to building these applications. JSF is a development framework for specifying forms and their processing in JSP documents.
- ▶ **ASP.NET** is a Web development framework. The code used in ASP.NET documents, which is executed on the server, can be written in any .NET programming language.
- ▶ **Ruby** is a relatively recent object-oriented scripting language that is introduced here primarily because of its use in Rails, a Web applications framework.
- ▶ **Rails** provides a significant part of the code required to build Web applications that access databases, allowing the developer to spend his or her time on the specifics of the application without the drudgery of dealing with all of the housekeeping details.

ORIGINS AND EVOLUTION OF HTML AND XHTML

HTML → Hyper Text Mark-up Language

XHTML → eXtensible Hyper Text Mark-up Language

HTML	XHTML
HTML is much easier to write	XHTML requires a level of discipline many of us naturally resist
huge number of HTML documents available on the Web, browsers will continue to support HTML as far as one can see into the future.	some older browsers have problems with some parts of XHTML.
HTML has few syntactic rules, and HTML processors (e.g., browsers) do not enforce the rules it does have. Therefore, HTML authors have a high degree of freedom to use their own syntactic preferences to create documents. Because of this freedom, HTML documents lack consistency, both in low-level syntax and in overall structure.	XHTML has strict syntactic rules that impose a consistent structure on all XHTML documents. Another significant reason for using XHTML is that when you create an XHTML document, its syntactic correctness can be checked, either by an XML browser or by a validation tool
Used for displaying the data	Used for describing the data

BASIC SYNTAX

- The fundamental syntactic units of HTML are called **tags**.
 - In general, tags are used to specify categories of content.
 - The syntax of a tag is the tag's name surrounded by *angle brackets* (< and >).
 - Tag names must be written in all lowercase letters.
 - Most tags appear in pairs: an *opening tag* and a *closing tag*.
 - The name of a closing tag is the name of its corresponding opening tag with a slash attached to the beginning. For example, if the tag's name is p, the corresponding closing tag is named /p.
 - Whatever appears between a tag and its closing tag is the **content** of the tag. Not all tags can have content.
 - The opening tag and its closing tag together specify a container for the content they enclose.
 - The container and its content together are called an **element**.
 - Example: **<p> This is NEW HORIZON Web Programming Notes. </p>**
 - The paragraph tag, <p>, marks the beginning of the content; the </p> tag marks the end of the content of the paragraph element.
 - Attributes, which are used to specify alternative meanings of a tag, can appear between an opening tag's name and its right angle bracket.
 - They are specified in keyword form, which means that the attribute's name is followed by an equal's sign and the attribute's value.
 - Attribute names, like tag names, are written in lowercase letters.
 - Attribute values must be delimited by double quotes.
 - Comments in programs increase the readability of those programs. Comments in XHTML have the same purpose. They can appear in XHTML in the following form:
<!-- anything except two adjacent dashes -->
 - Browsers ignore XHTML comments—they are for people only. Comments can be spread over as many lines as are needed. For example, you could have the following comment:
- ```

<!-- CopyRights.html
This notes is prepared by Abhishek K of Information Science
Department NEW HORIZON, Bangalore -->

```

## Standard XHTML Document Structure

- Every XHTML document must begin with an xml declaration element that simply identifies the document as being one based on XML. This element includes an attribute that specifies the version number 1.0.
- The xml declaration usually includes a second attribute, encoding, which specifies the encoding used for the document [utf-8].
- Following is the xml declaration element, which should be the first line of every XHTML document:

**<?xml version = "1.0" encoding = "utf-8"?>**

- Note that this declaration must begin in the first character position of the document file.
- The xml declaration element is followed immediately by an SGML DOCTYPE command, which specifies the particular SGML document-type definition (DTD) with which the document complies, among other things.
- The following command states that the document in which it is included complies with the XHTML 1.0 Strict standard:

**<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">**

- An XHTML document must include the four tags <html>, <head>, <title>, and <body>.
- The <html> tag identifies the root element of the document. So, XHTML documents always have an <html> tag immediately following the DOCTYPE command, and they always end with the closing html tag, </html>.
- The html element includes an attribute, xmlns, that specifies the XHTML namespace, as shown in the following element:

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

- Although the xmlns attribute's value looks like a URL, it does not specify a document. It is just a name that happens to have the form of a URL.
- An XHTML document consists of two parts, named the *head* and the *body*.
- The <head> element contains the head part of the document, which provides information about the document and does not provide the content of the document.
- The body of a document provides the content of the document.
- The content of the title element is displayed by the browser at the top of its display window, usually in the browser window's title bar.

## BASIC TEXT MARKUP

We will have a look at a complete XHTML document:

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!-- complete.html
A document which must be followed throughout the notes
-->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
 <title> My first program </title>
</head>
<body>
 <p>
 My Dear VTU Friends, All The Best..!! Have a Happy Reading of my notes..!!
 </p>
</body>
</html>
```



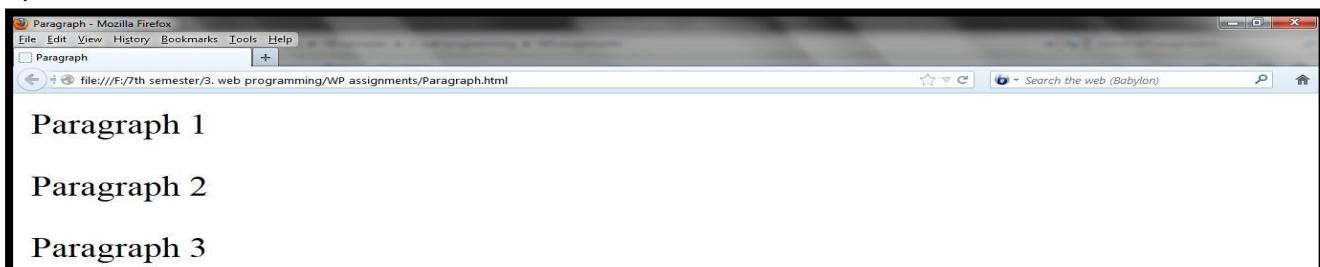
**PLEASE NOTE:** From here onwards programming in XHTML will begin. Please add the following compulsory document structure to all programs in the first 4 lines and skip the simple <html> tag of first line because I have begun the coding part directly.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
```

## Paragraphs:

It begins with <p> and ends with </p>. Multiple paragraphs may appear in a single document.

```
<html>
<head>
 <title> Paragraph </title>
</head>
<body>
 <p> Paragraph 1 </p>
 <p> Paragraph 2 </p>
 <p> Paragraph 3 </p>
</body>
</html>
```



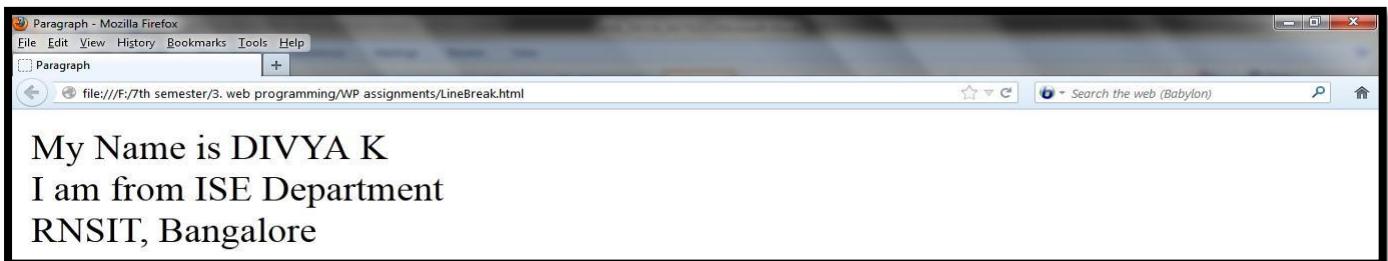
## Line Breaks:

The break tag is specified as <br />. The slash indicates that the tag is both an opening and closing tag.

```
<html>
<head>
 <title> br tag </title>
</head>
<body>
 <p>
 My Name is ABHISHEK K

 I am from ISE Department

 NEW HORIZON, Bangalore
 </p>
</body>
</html>
```



## Preserving White Space

Sometimes it is desirable to preserve the white space in text—that is, to prevent the browser from eliminating multiple spaces and ignoring embedded line breaks. This can be specified with the `<pre>` tag.

```

<html>
<head>
 <title> Pre Tag </title>
</head>
<body>
 <p><pre> My Name is ABHISHEK K
 I am from ISE Department
 NEW HORIZON, Bangalore
 </pre></p>
 </body>
</html>

```



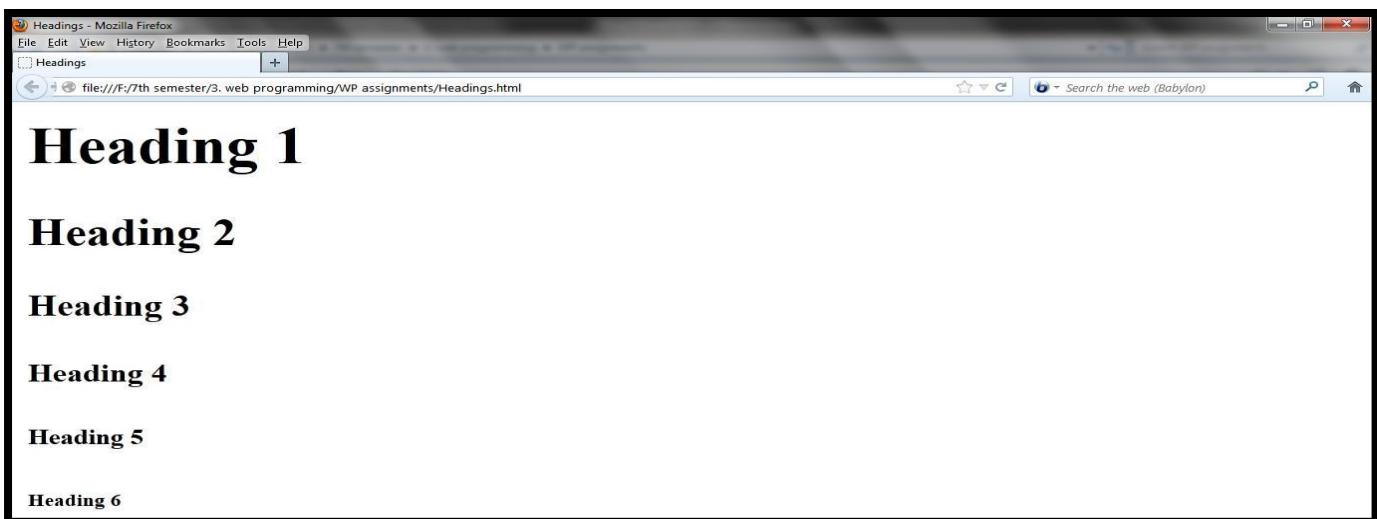
## Headings:

In XHTML, there are six levels of headings, specified by the tags `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`, where `<h1>` specifies the highest-level heading. Headings are usually displayed in a boldface font whose default size depends on the number in the heading tag. On most browsers, `<h1>`, `<h2>`, and `<h3>` use font sizes that are larger than that of the default size of text, `<h4>` uses the default size, and `<h5>` and `<h6>` use smaller sizes. The heading tags always break the current line, so their content always appears on a new line. Browsers usually insert some vertical space before and after all headings.

```

<html>
<head>
 <title> Headings </title>
</head>
<body>
 <h1> Heading 1 </h1>
 <h2> Heading 2 </h2>
 <h3> Heading 3 </h3>
 <h4> Heading 4 </h4>
 <h5> Heading 5 </h5>
 <h6> Heading 6 </h6>
</body>
</html>

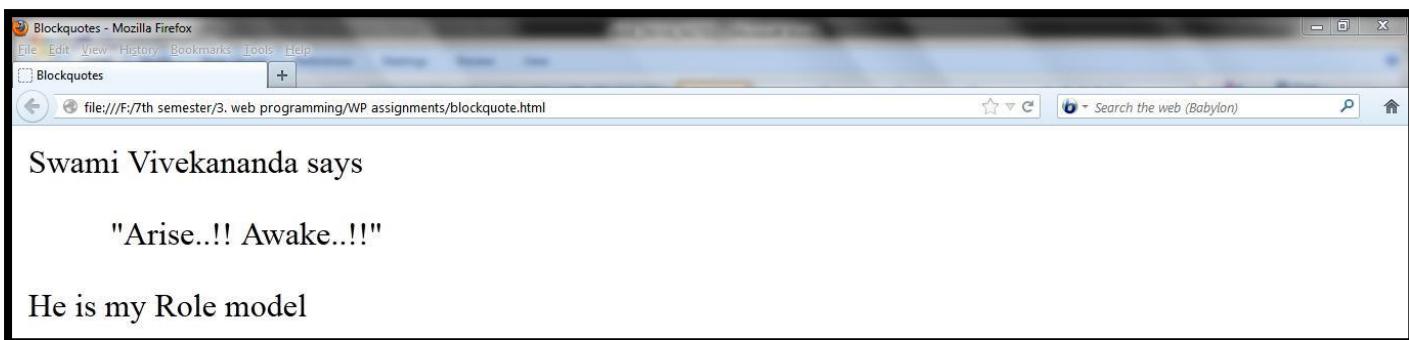
```



## Block Quotations:

The `<blockquote>` tag is used to make the contents look different from the surrounding text.

```
<html>
<head> <title> Blockquotes </title>
</head>
<body>
<p> Swami Vivekananda says </p>
<blockquote>
<p> "Arise..!! Awake..!!" </p>
</blockquote>
<p> He is my Role model </p>
</body>
</html>
```



## Font Styles and Sizes:

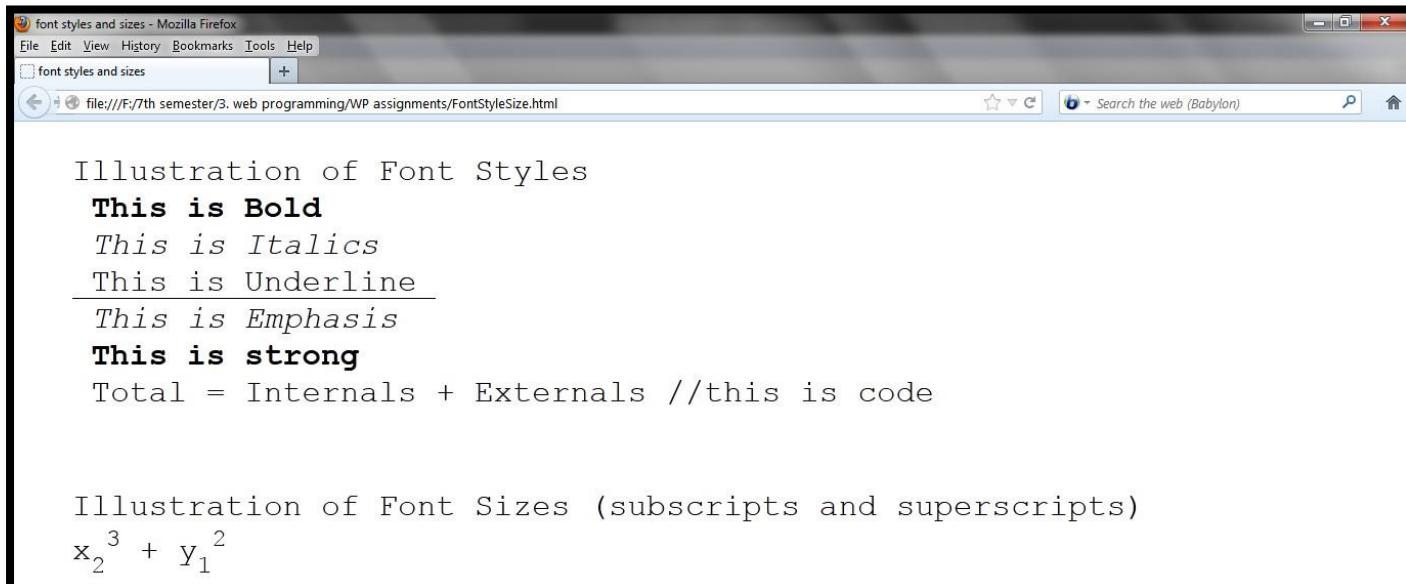
- `<b>`, `<i>` and `<u>` specifies bold, italics and underline respectively.
- The emphasis tag, `<em>`, specifies that its textual content is special and should be displayed in some way that indicates this distinctiveness. Most browsers use italics for such content.
- The strong tag, `<strong>` is like the emphasis tag, but more so. Browsers often set the content of strong elements in bold.
- The code tag, `<code>`, is used to specify a monospace font, usually for program code.

```
<html>
<head> <title> font styles and sizes </title>
</head>
<body>
<p><pre>
Illustration of Font Styles
```

```

 This is Bold
<i> This is Italics </i>
<u> This is Underline </u>
 This is Emphasis
 This is strong
<code> Total = Internals + Externals //this is code</code>
</pre></p>
<p><pre>
 Illustration of Font Sizes (subscripts and superscripts)
 x₂³ + y₁²
</pre></p>
</body>
</html>

```



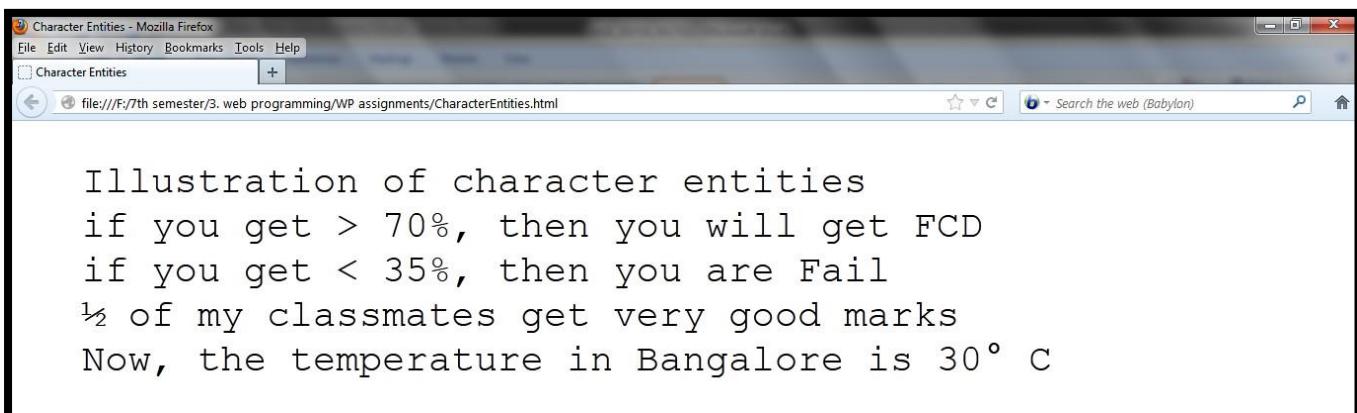
## Character Entities:

XHTML provides a collection of special characters that are sometimes needed in a document but cannot be typed as themselves. In some cases, these characters are used in XHTML in some special way—for example, >, <, and &. In other cases, the characters do not appear on keyboards, such as the small raised circle that represents “degrees” in a reference to temperature. These special characters are defined as **entities**, which are codes for the characters. An entity in a document is replaced by its associated character by the browser.

```

<html>
<head> <title> Character Entities </title>
</head>
<body>
<p><pre>
 Illustration of character entities
 if you get > 70%, then you will get FCD
 if you get < 35%, then you are Fail
 ½ of my classmates get very good marks
 Now, the temperature in Bangalore is 30° C
</pre></p>
</body>
</html>

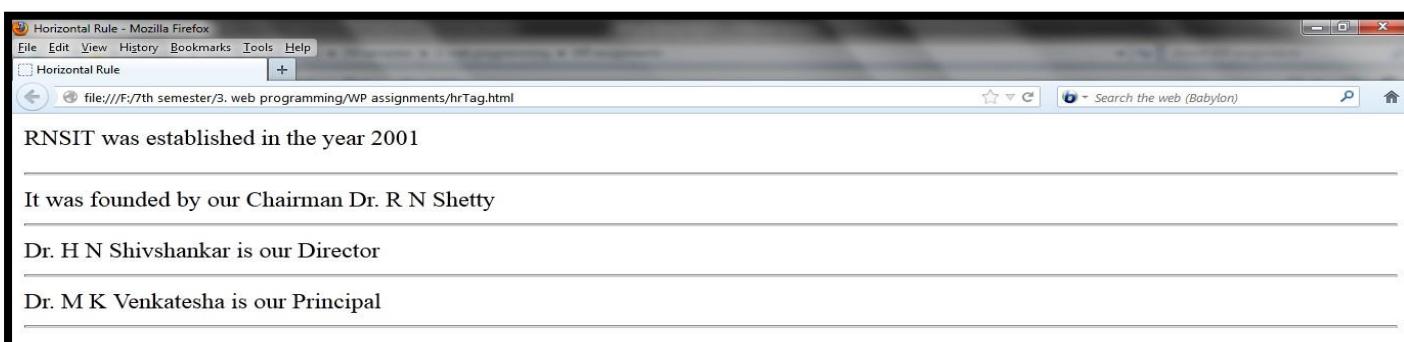
```



## Horizontal Rules:

The parts of a document can be separated from each other, making the document easier to read, by placing horizontal lines between them. Such lines are called horizontal rules. The block tag that creates them is `<hr />`. The `<hr />` tag causes a line break (ending the current line) and places a line across the screen. Note again the slash in the `<hr />` tag, indicating that this tag has no content and no closing tag.

```
<html>
<head>
 <title> Horizontal Rule </title>
</head>
<body>
<p>
 NEW HORIZON was established in the year 2001 <hr/>
 It was founded by our Chairman Dr. R N Shetty <hr/>
 Dr. H N Shivshankar is our Director <hr/>
 Dr. M K Venkatesha is our Principal <hr/>
</p>
</body>
</html>
```



## The meta Element:

The `meta` element is used to provide additional information about a document. The `meta` tag has no content; rather, all of the information provided is specified with attributes. The two attributes that are used to provide information are `name` and `content`. The user makes up a name as the value of the `name` attribute and specifies information through the `content` attribute. One commonly chosen name is `keywords`; the value of the `content` attribute associated with the `keywords` are those which the author of a document believes characterizes his or her document. An example is

```
<meta name = "Title" content = "Programming the Web" />
<meta name = "Author" content = "Abhishek K" />
```

Web search engines use the information provided with the `meta` element to categorize Web documents in their indices.

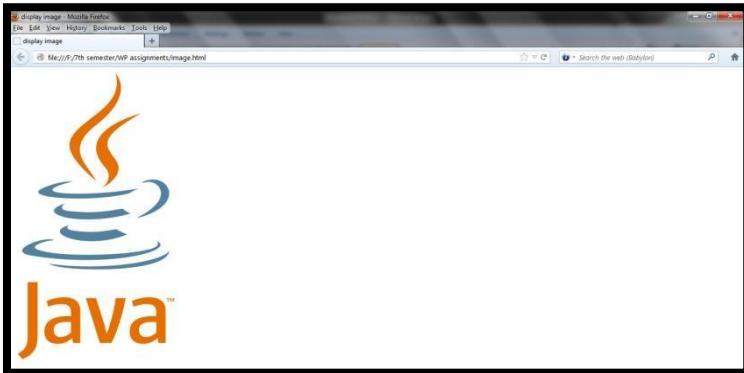
## XHTML - 2

### IMAGES

- Image can be displayed on the web page using <img> tag.
- When the <img> tag is used, it should also be mentioned which image needs to be displayed. This is done using **src** attribute.
- Attribute means extra information given to the browser
- Whenever <img> tag is used, alt attribute is also used.
- Alt stands for alert.
- Some very old browsers would not have the capacity to display the images.
- In this case, whatever is the message given to alt attribute, that would be displayed.
- Another use of alt is → when image display option has been disabled by user. The option is normally disabled when the size of the image is huge and takes time for downloading.

```
<html>
 <head>
 <title>display image</title>
 </head>
 <body>

 </body>
</html>
```



#### NOTE:

- JPEG → Joint Photographic Experts Group
- GIF → Graphic Interchange Format
- PNG → Portable Network Graphics

### XHTML Document Validation:

The W3C provides a convenient Web-based way to validate XHTML documents against its standards.

Step 1: The URL of the service is <http://validator.w3.org/file-upload.html>. Copy & paste this link.

Step 2: You will be driven to “**Validate by File Upload**” option automatically.

Step 3: Browse for a XHTML program file in your computer. (example: F:/complete.html)

Step 4: Click on “**More Options**” and select your criteria like *show source*

Step 5: After all the settings, click on “**Check**” button

Now you will be navigated to another page which shows success or failure.

In our example, the file *complete.html* is a valid XHTML file. So the output shows success...!!

The screenshot shows the W3C Markup Validation Service interface in Mozilla Firefox. The page title is "The W3C Markup Validation Service - Mozilla Firefox". The URL in the address bar is "validator.w3.org/#validate\_by\_upload+with\_options".

- 1:** A red speech bubble points to the number "1" located in the top left corner of the browser window.
- 2:** A red speech bubble points to the "W3C" logo and the text "Markup Validation Service" in the header.
- 3:** A red speech bubble points to the "File" input field where "F:\complete.html" is selected.
- 4:** A red speech bubble points to the "More Options" link.
- 5:** A red speech bubble points to the "Check" button at the bottom of the form.

**Note:** file upload may not work with Internet Explorer on some versions of Windows XP Service Pack 2, see our [information page](#) on the W3C QA Website.

This validator checks the [markup validity](#) of Web documents in HTML, XHTML, SMIL, MathML, etc. If you wish to validate specific content such as [RSS/Atom feeds](#) or [CSS stylesheets](#), [MobileOK](#) content, or to [find broken links](#), there are [other validators and tools](#) available. As an alternative you can also try our [non-DTD-based validator](#).

## Output:

The screenshot shows the "Markup Validation of complete.html - W3C Markup Validator - Mozilla Firefox" page. The URL in the address bar is "validator.w3.org/check".

The main content area includes the following information:

- Result:** Passed, 1 warning(s)
- File:**  Browse... Use the file selection box above if you wish to re-validate the uploaded file complete.html
- Modified:** (undefined)
- Server:** Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
- Size:** (undefined)
- Content-Type:** text/html
- Encoding:** utf-8 (detect automatically)
- Doctype:** XHTML 1.0 Strict (detect automatically)
- Root Element:** html
- Root Namespace:** <http://www.w3.org/1999/xhtml>

A red oval highlights the green banner message: **This document was successfully checked as XHTML 1.0 Strict!**

A blue speech bubble contains the text: **Hey Success..!!**

At the bottom of the page, there is a "I ❤ VALIDATOR" button, a note about community support and donations, and a "3523 Flattr" button.

## HYPERTEXT LINKS

### Links:

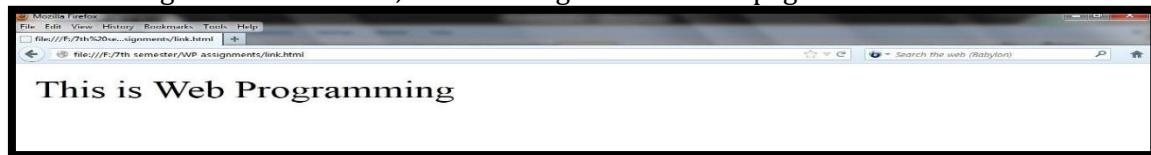
- Hyperlinks are the mechanism which allows the navigation from one page to another.
- The term “hyper” means beyond and “link” means connection
- Whichever text helps in navigation is called hypertext
- Hyperlinks can be created using `<a>` (anchor tag)
- The attribute that should be used for `<a>` is `href`

Program: *hyper.html*

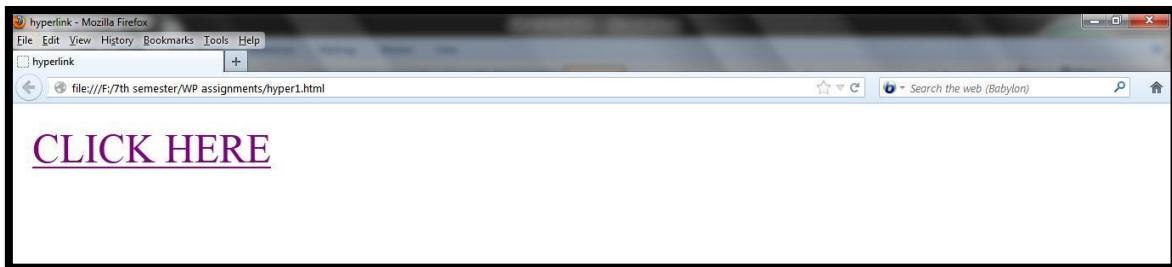
```
<html>
<head>
<title> hyperlink </title>
</head>
 CLICK HERE
</html>
```

Program: *link.html*

After clicking on the above text, we can navigate to another page “link.html” as shown below



```
<html>
<body> This is Web Programming </body>
</html>
```

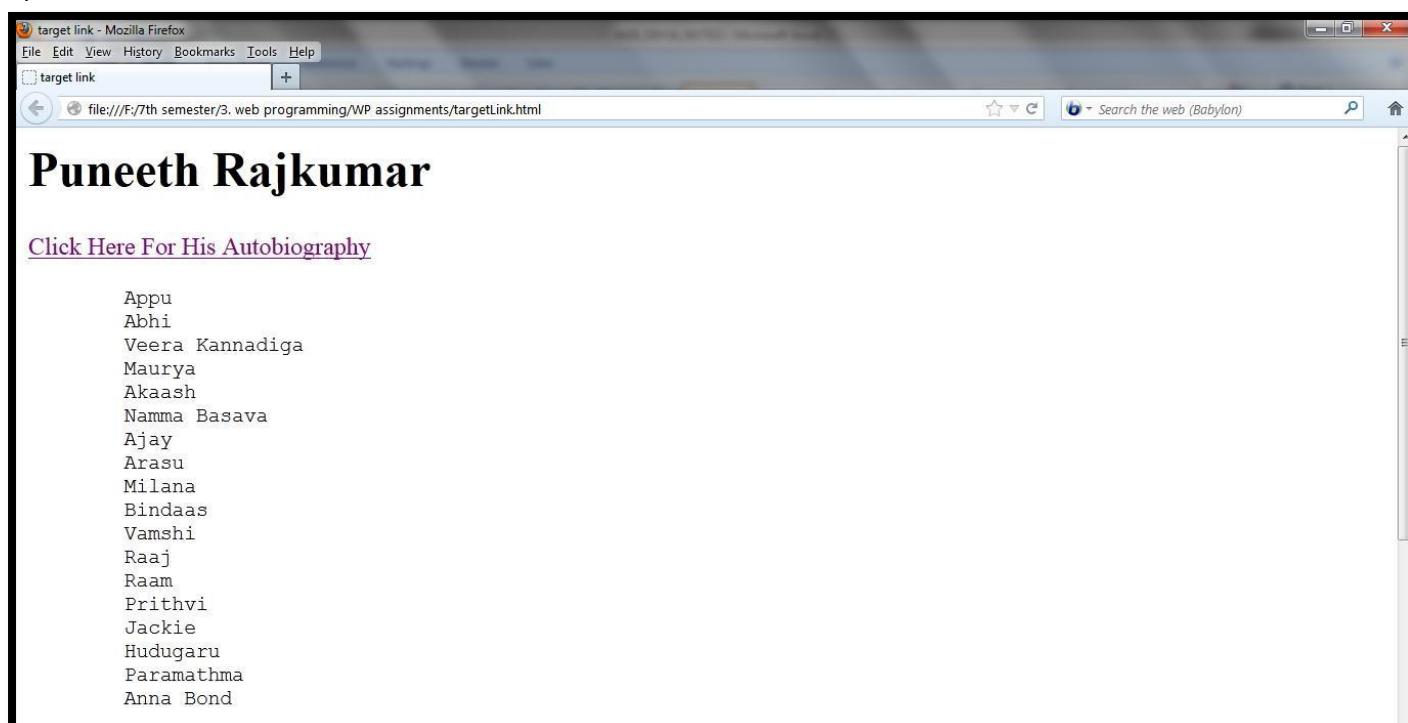


### Targets within Documents:

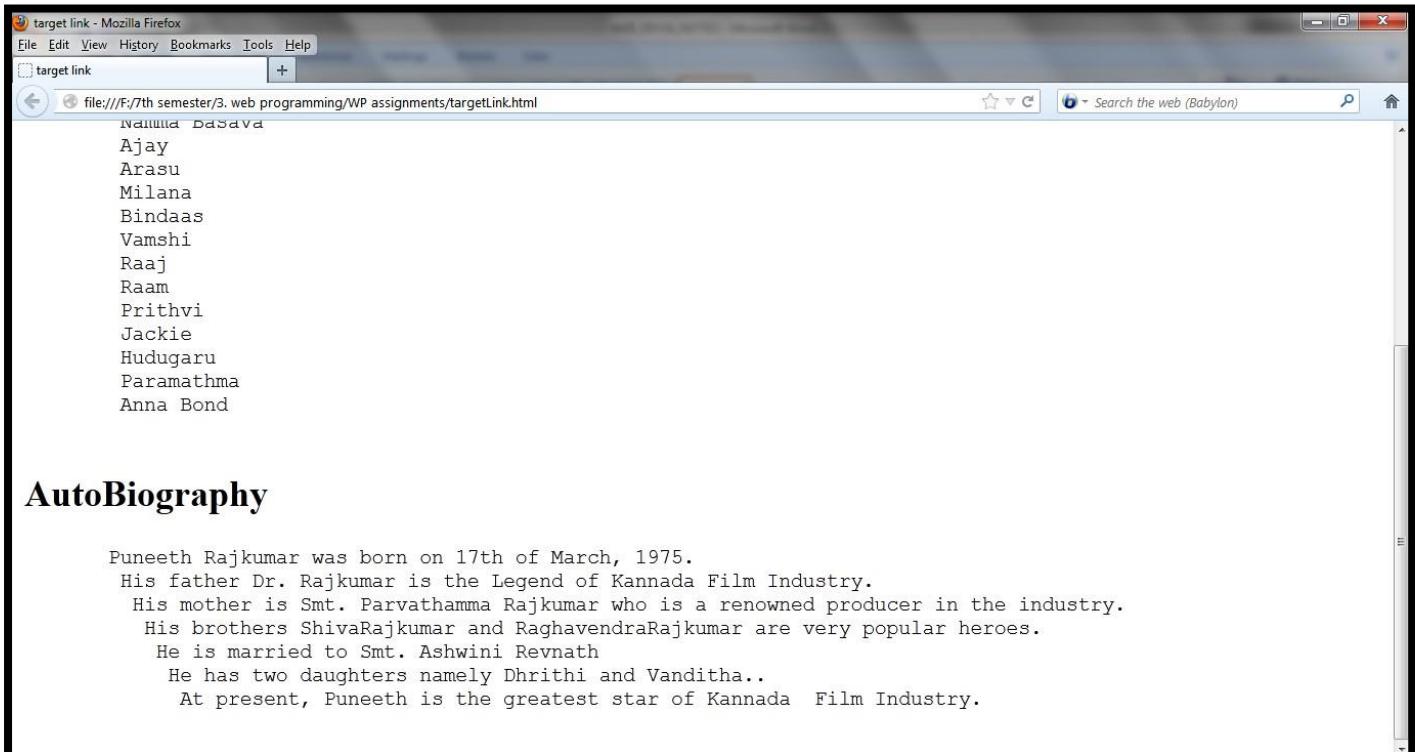
If the target of a link is not at the beginning of a document, it must be some element within the document, in which case there must be some means of specifying it. The target element can include an `id` attribute, which can then be used to identify it in an `href` attribute. (observe the scroll bar in the outputs given)

```
<html>
<head>
<title> target link</title>
</head>
<body>
<h1> Puneeth Rajkumar </h1>
 Click Here For His Autobiography
<p><pre>
Appu
Abhi
```

```
Veera Kannadiga
Maurya
Akaash
Namma Basava
Ajay
Arasu
Milana
Bindaas
Vamshi
Raaj
Raam
Prithvi
Jackie
Hudugaru
Paramathma
Anna Bond
</pre></p>
<h2> AutoBiography </h2>
<p id = "bottom"> <pre>
Puneeth Rajkumar was born on 17th of March, 1975.
His father Dr. Rajkumar is the Legend of Kannada Film Industry.
His mother is Smt. Parvathamma Rajkumar who is a renowned producer in the industry.
His brothers ShivaRajkumar and RaghavendraRajkumar are very popular heroes.
He is married to Smt. Ashwini Revnath
He has two daughters namely Dhrithi and Vanditha..
At present, Puneeth is the greatest star of Kannada Film Industry.
</pre></p>
</body>
</html>
```



Actually, here we are not creating two separate files, but we are specifying a target within the same document itself. If you click on the above link, you will be redirected to the bottom of the page which contains Autobiography of Puneeth Rajkumar. This is useful for lengthy documents like e-newspaper, e-magazine etc.,



## AutoBiography

Puneeth Rajkumar was born on 17th of March, 1975.  
His father Dr. Rajkumar is the Legend of Kannada Film Industry.  
His mother is Smt. Parvathamma Rajkumar who is a renowned producer in the industry.  
His brothers ShivaRajkumar and RaghavendraRajkumar are very popular heroes.  
He is married to Smt. Ashwini Revnath  
He has two daughters namely Dhrithi and Vanditha..  
At present, Puneeth is the greatest star of Kannada Film Industry.

## LISTS

### Unordered Lists:

The `<ul>` tag, which is a block tag, creates an unordered list. Each item in a list is specified with an `<li>` tag (`li` is an acronym for `list item`). Any tags can appear in a list item, including nested lists. When displayed, each list item is implicitly preceded by a bullet.

```
<html>
<head>
<title> Unordered List </title>
</head>
<body>
<h1> Heroines acted with Puneeth Rajkumar </h1>

 Rakshitha
 Ramya
 Nathasha
 Meera Jasmine
 Anuradha Mehtha
 Parvathi Menon
 Hansika
 Nikitha
 Nisha Kothari
 Priya Mani
 Bhavana Menon
 Radhika Pandit
 Deepa Sannidhi

</body>
</html>
```

**Heroines acted with Puneeth Rajkumar**

- Rakshitha
- Ramya
- Nathasha
- Meera Jasmine
- Anuradha Mehtha
- Parvathi Menon
- Hansika
- Nikitha
- Nisha Kothari
- Priya Mani
- Bhavana Menon
- Radhika Pandit
- Deepa Sannidhi

### Ordered Lists:

Ordered lists are lists in which the order of items is important. This orderedness of a list is shown in the display of the list by the implicit attachment of a sequential value to the beginning of each item. The default sequential values are Arabic numerals, beginning with 1.

An ordered list is created within the block tag `<ol>`. The items are specified and displayed just as are those in unordered lists, except that the items in an ordered list are preceded by sequential values instead of bullets.

```

<html>
 <head>
 <title> ordered List </title>
 </head>
 <body>
 <h1>Chicken Masala</h1>

 For 1 kg of chicken, add 20g Teju Chicken Masala
 Fry 2 big onions with 3tbsp ghee/oil till golden brown
 Add 2 tomato, 1tsp ginger garlic paste, 2-3 green chillies and fry
 When tomato becomes soft, add chicken and 100ml water
 Add 25g coriander leaves and cook till the chicken is soft and gravy turns thick
 Ready to serve

 </body>
</html>

```

**Chicken Masala**

1. For 1 kg of chicken, add 20g Teju Chicken Masala
2. Fry 2 big onions with 3tbsp ghee/oil till golden brown
3. Add 2 tomato, 1tsp ginger garlic paste, 2-3 green chillies and fry
4. When tomato becomes soft, add chicken and 100ml water
5. Add 25g coriander leaves and cook till the chicken is soft and gravy turns thick
6. Ready to serve

**Nested Lists:**

```

<html>
<head>
<title> nested lists </title>
</head>

 Information Science

OOMD
Java & J2ee

classes and methods
exceptions
applets
servelets

Computer Networks

Part 1
Part 2

DBMS
Operations Research

 Computer Science

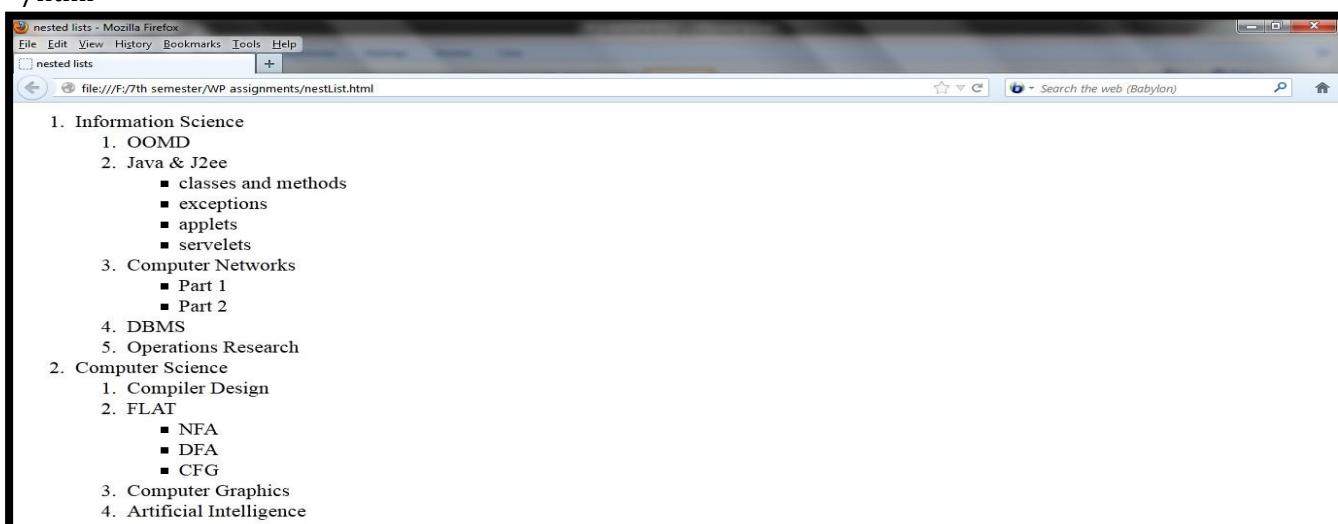
Compiler Design
FLAT

NFA
DFA
CFG

Computer Graphics
Artificial Intelligence

</html>

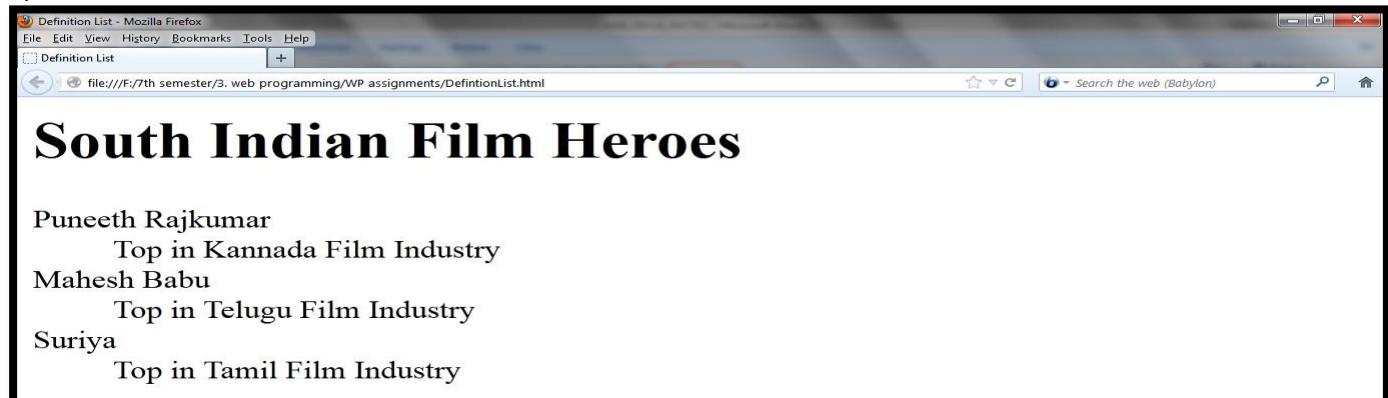
```



## Definition Lists:

As the name implies, definition lists are used to specify lists of terms and their definitions, as in glossaries. A definition list is given as the content of a `<dl>` tag, which is a block tag. Each term to be defined in the definition list is given as the content of a `<dt>` tag. The definitions themselves are specified as the content of `<dd>` tags. The defined terms of a definition list are usually displayed in the left margin; the definitions are usually shown indented on the line or lines following the term.

```
<html>
<head>
<title> Definition List </title>
</head>
<body>
<h1> South Indian Film Heroes </h1>
<dl>
 <dt> Puneeth Rajkumar </dt>
 <dd> Top in Kannada Film Industry </dd>
 <dt> Mahesh Babu </dt>
 <dd> Top in Telugu Film Industry </dd>
 <dt> Suriya </dt>
 <dd> Top in Tamil Film Industry </dd>
</dl>
</body>
</html>
```



## TABLES

A table is a matrix of cells. The cells in the top row often contain column labels, those in the leftmost column often contain row labels, and most of the rest of the cells contain the data of the table. The content of a cell can be almost any document element, including text, a heading, a horizontal rule, an image, and a nested table.

### Basic Table Tags:

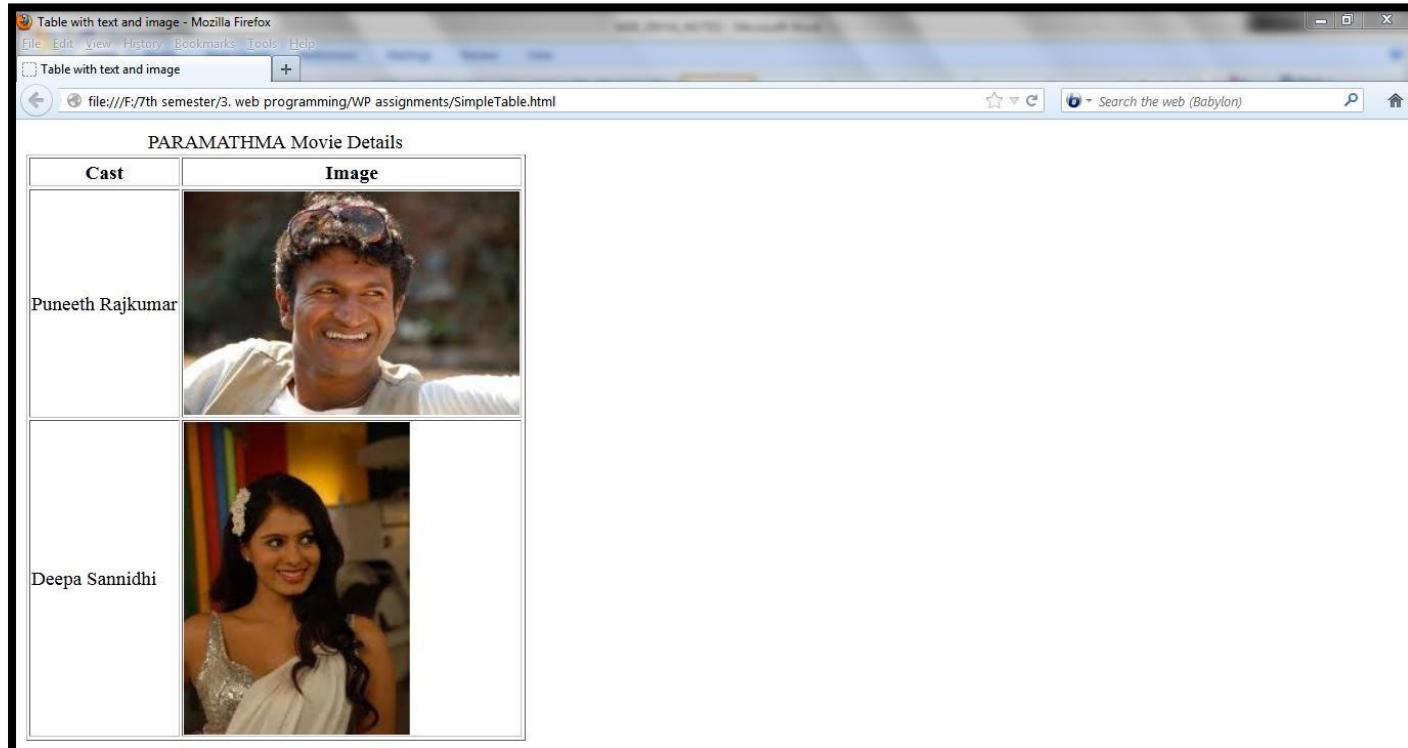
- A table is specified as the content of the block tag `<table>`.
- There are two kinds of lines in tables: the line around the outside of the whole table is called the *border*; the lines that separate the cells from each other are called *rules*.
- It can be obtained using **border** attribute. The possible values are “border” or any number.
- The table heading can be created using `<caption>` tag.
- The table row can be created using `<tr>` tag.
- The column can be created either by using `<th>` tag (stands for table header which is suitable for headings) or `<td>` tag (stands for table data which is suitable for other data).

```
<html>
<head>
<title> Table with text and image </title>
```

```

</head>
<body>
<table border = "border">
<caption>PARAMATHMA Movie Details </caption>
<tr>
<th> Cast</th>
<th> Image </th>
</tr>
<tr>
<td> Puneeth Rajkumar </td>
<td> </td>
</tr>
<tr>
<td> Deepa Sannidhi</td>
<td> </td>
</tr>
</table>
</body>
</html>

```



### The rowspan and colspan Attributes:

Multiple-level labels can be specified with the rowspan and colspan attributes.

```

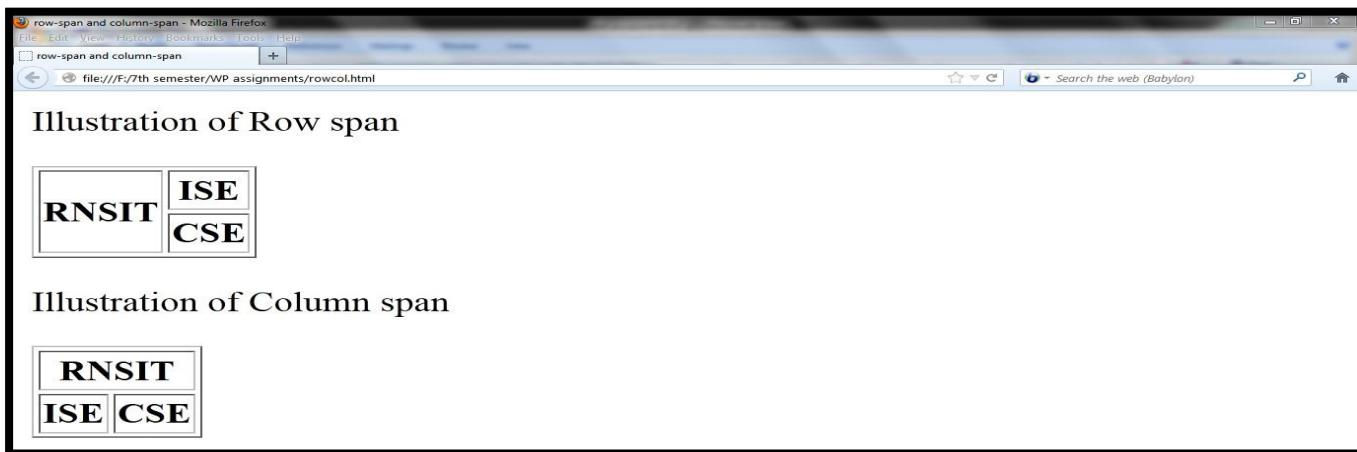
<html>
<head>
<title>row-span and column-span</title>
</head>
<body>
<p> Illustration of Row span</p>
<table border="border">
<tr>

```

```

<th rowspan="2"> NEW HORIZON </th>
<th>ISE</th>
</tr>
<tr>
<th>CSE</th>
</tr>
</table>
<p> Illustration of Column span</p>
<table border="border">
<tr>
<th colspan="2"> NEW HORIZON </th>
</tr>
<tr>
<th>ISE</th>
<th>CSE</th>
</tr>
</table>
</body>
</html>

```



## The align and valign Attributes:

The placement of the content within a table cell can be specified with the **align** and **valign** attributes in the **<tr>**, **<th>**, and **<td>** tags.

The **align** attribute has the possible values **left**, **right**, and **center**, with the obvious meanings for horizontal placement of the content within a cell. The default alignment for **th** cells is **center**; for **td** cells, it is **left**.

The **valign** attribute of the **<th>** and **<td>** tags has the possible values **top** and **bottom**. The default vertical alignment for both headings and data is **center**.

```

<html>
<head>
<title> Align and valign </title>
</head>
<body>
<p>Table having entries with different alignments</p>
<table border="border">
<tr align = "center">
<th> </th>
<th> Puneeth Rajkumar </th>
<th> Darshan Thoogudeep</th>
<th> Kichcha Sudeep </th>

```

```

</tr>
<tr>
 <th> Ramya </th>
 <td align = "left"> Akaash </td>
 <td align = "center"> Datta </td>
 <td align = "right"> Ranga </td>
</tr>
<tr>
 <th>
Rakshitha

</th>
 <td> Appu </td>
 <td valign = "top"> Kalasipalya </td>
 <td valign = "bottom"> Kaashi from village </td>
</tr>
</table>
</body>
</html>

```



## The cellpadding and cellspacing Attributes:

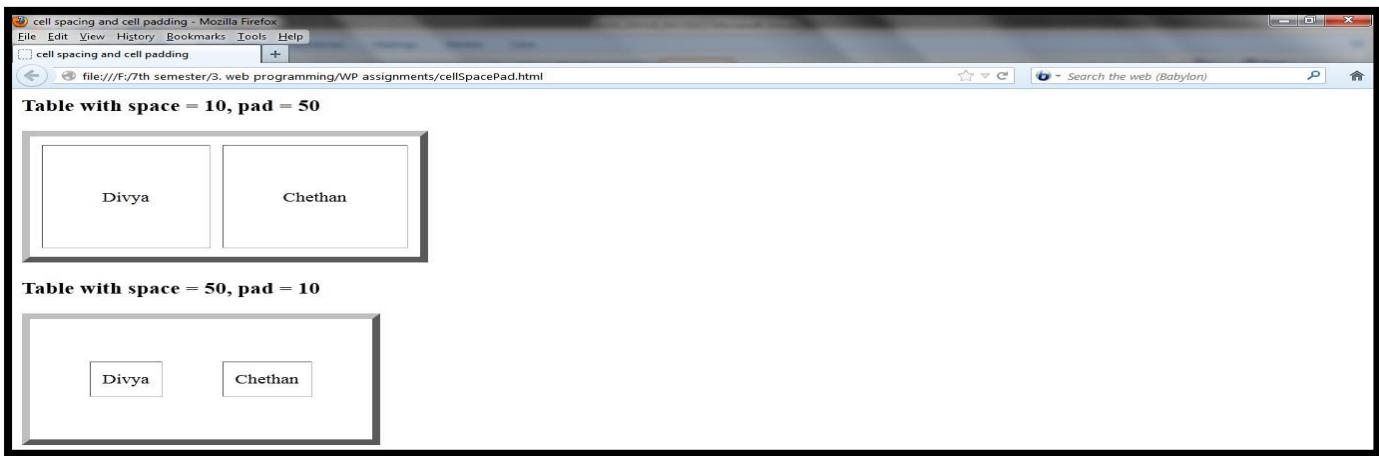
Cellspacing is the distance between cells.

Cellpadding is the distance between the edges of the cell to its content.

```

<html>
<head>
 <title> cell spacing and cell padding </title>
</head>
<body>
<h3>Table with space = 10, pad = 50</h3>
<table border = "7" cellspacing = "10" cellpadding = "50">
<tr>
 <td> Abhishek </td>
 <td>Chethan </td>
</tr>
</table>
<h3>Table with space = 50, pad = 10</h3>
<table border = "7" cellspacing = "50" cellpadding = "10">
<tr>
 <td> Abhishek </td>
 <td>Chethan </td>
</tr>
</table>
</body>
</html>

```



## Table Sections:

Tables naturally occur in two and sometimes three parts: header, body, and footer. (Not all tables have a natural footer.) These three parts can be respectively denoted in XHTML with the `thead`, `tbody`, and `tfoot` elements. The header includes the column labels, regardless of the number of levels in those labels. The body includes the data of the table, including the row labels. The footer, when it appears, sometimes has the column labels repeated after the body. In some tables, the footer contains totals for the columns of data above. A table can have multiple body sections, in which case the browser may delimit them with horizontal lines that are thicker than the rule lines within a body section.

## FORMS

The most common way for a user to communicate information from a Web browser to the server is through a form. XHTML provides tags to generate the commonly used objects on a screen form. These objects are called *controls* or *widgets*. There are controls for single-line and multiple-line text collection, checkboxes, radio buttons, and menus, among others. All control tags are inline tags.

### The `<form>` Tag:

All of the controls of a form appear in the content of a `<form>` tag. A block tag, `<form>`, can have several different attributes, only one of which, `action`, is required. The `action` attribute specifies the URL of the application on the Web server that is to be called when the user clicks the *Submit* button. Our examples of form elements will not have corresponding application programs, so the value of their `action` attributes will be the empty string ("").

### The `<input>` Tag:

Many of the commonly used controls are specified with the inline tag `<input>`, including those for text, passwords, checkboxes, radio buttons, and the action buttons *Reset*, *Submit*, and *plain*.

#### ❖ Text Box

- ✓ It is a type of input which takes the text.
- ✓ Any type of input can be created using `<input>`
- ✓ The `type` attribute indicates what type of input is needed for the text box, the value should be given as text.
- ✓ For any type of input, a name has to be provided which is done using `name` attribute.
- ✓ The size of the text can be controlled using `size` attribute.
- ✓ Every browser has a limit on the number of characters it can collect. If this limit is exceeded, the extra characters are chopped off. To prevent this chopping, `maxlength` attribute can be used. When `maxlength` is used, users can enter only those many characters that is given as a value to the attribute.

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

```

<title>Text Box</title>
</head>
<body>
<form action = " ">
<p>
 <label>Enter your Name:</label>
 <input type = "text" name = "myname" size = "20" maxlength = "20" />
</p>
</form>
</body>
</html>

```



Enter your Name:

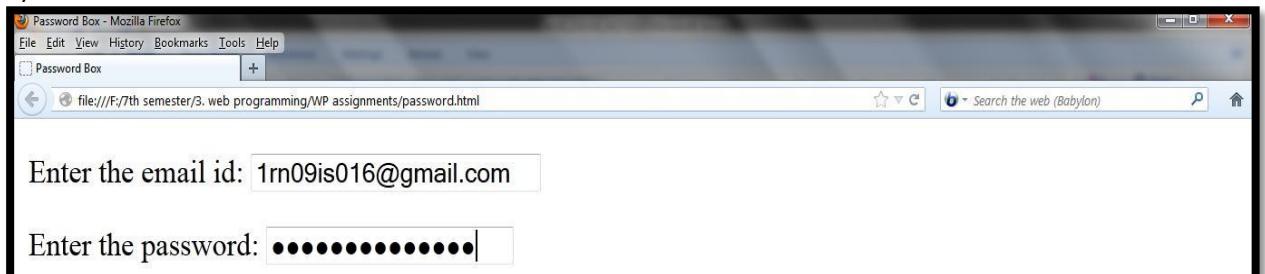
### ❖ Password Box

- ✓ If the contents of a text box should not be displayed when they are entered by the user, a password control can be used.
- ✓ In this case, regardless of what characters are typed into the password control, only bullets or asterisks are displayed by the browser.

```

<html>
<head>
<title>Password Box</title>
</head>
<body>
<form action = " ">
<p>
 <label>Enter the email id:</label>
 <input type = "text" name = "myname" size = "24" maxlength = "25" />
</p>
<p>
 <label>Enter the password:</label>
 <input type = "password" name = "mypass" size = "20" maxlength = "20" />
</p>
</form>
</body>
</html>

```



## ❖ Radio Button

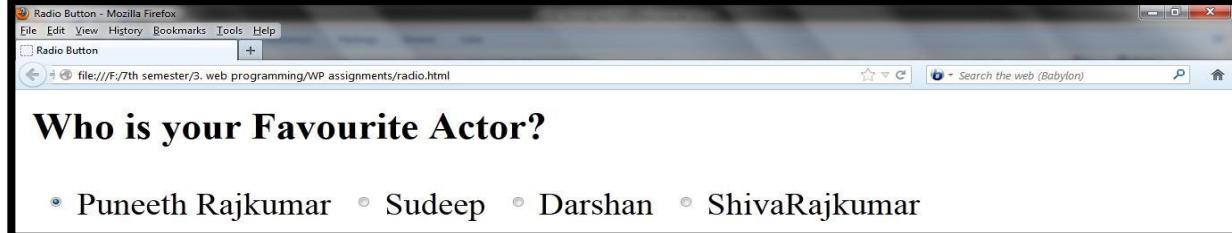
- ✓ Radio buttons are special type of buttons which allows the user to select only individual option
- ✓ Radio buttons are created using the `input` tag with the `type` attribute having the value **radio**.
- ✓ When radio buttons are created, values must be provided with the help of `value` attribute.
- ✓ All the radio buttons which are created would have same name. This is because the radio buttons are group elements.
- ✓ If one of the radio buttons has to be selected as soon as the web page is loaded, checked attribute should be used. The value also would be checked.

```
<html>
<head>
<title>Radio Button</title>
</head>
<body>
<h3>Who is your Favourite Actor?</h3>
<form action = " ">
<p>
<label><input type="radio" name="act" value="one"/>Puneeth Rajkumar</label>

<label><input type="radio" name="act" value="two"/>Sudeep</label>

<label><input type="radio" name="act" value="three"/>Darshan</label>

<label><input type="radio" name="act" value="four"/>ShivaRajkumar</label>
</p>
</form>
</body>
</html>
```



## ❖ Check Box

- ✓ Check box is a type of input using which multiple options can be selected.
- ✓ Check box can also be created using the `input` tag with the `type` having the value “checkbox”.
- ✓ During the creation of check box, the value should be provided using the `value` attribute.
- ✓ All the checkbox which are created would have the same name because they are group elements.
- ✓ If one of the check box have to be selected as soon as the page is loaded, checked attribute should be used with the value checked.

```
<html>
<head>
<title>Check Box</title>
</head>
<body>
<h3>Who is your Favourite Actress?</h3>
<form action = " ">
<p>
<label><input type="checkbox" name="act" value="one"/>Ragini</label>

<label><input type="checkbox" name="act" value="two"/>Ramya</label>
```

```

<label><input type="checkbox" name="act" value="three"/>Aindritha</label>
<label><input type="checkbox" name="act" value="four"/>Radhika</label>
<label><input type="checkbox" name="act" value="four"/>Rakshitha</label>
</p>
</form>
</body>
</html>

```



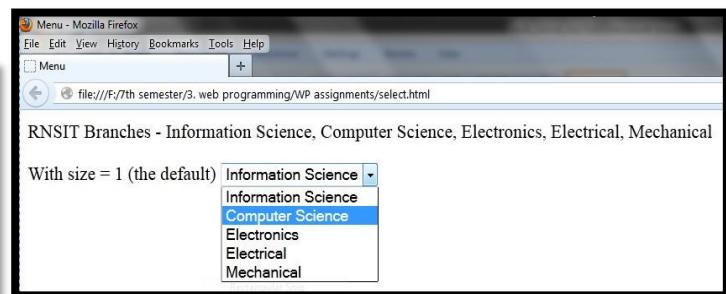
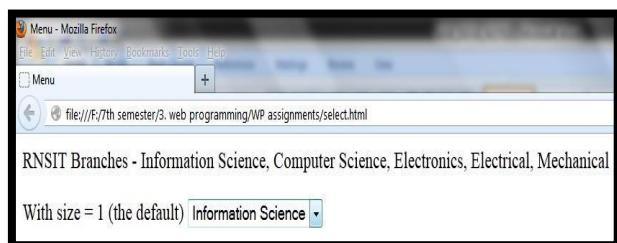
## The <select> Tag:

- Menu items is another type of input that can be created on the page.
- To create the menu item, <select> tag is used.
- To insert the item in the menu, <option> tag is used.

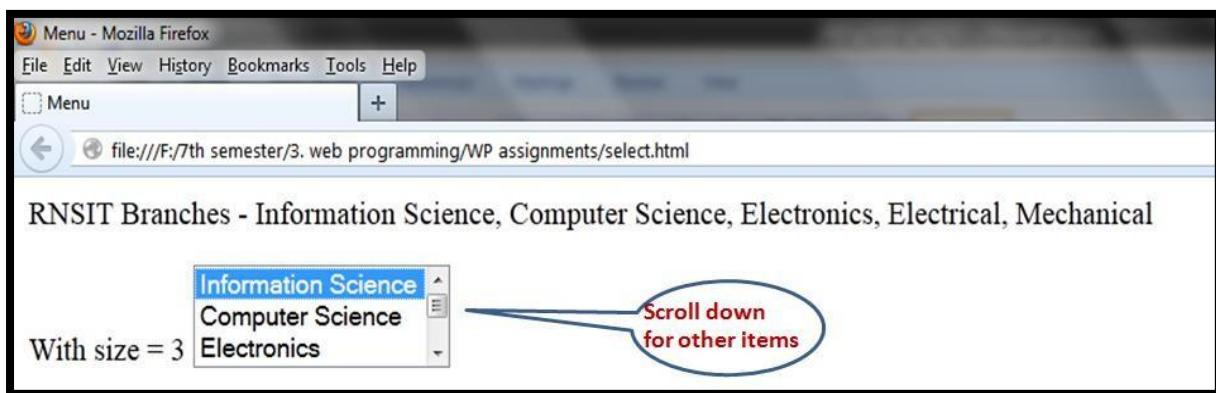
```

<html>
<head> <title> Menu </title>
</head>
<body>
<p>
NEW HORIZON Branches - Information Science, Computer Science, Electronics, Electrical, Mechanical
</p>
<form action = "">
<p>
With size = 1 (the default)
<select name = "branches">
<option> Information Science </option>
<option> Computer Science </option>
<option> Electronics </option>
<option> Electrical </option>
<option> Mechanical </option>
</select>
</p>
</form>
</body>
</html>

```



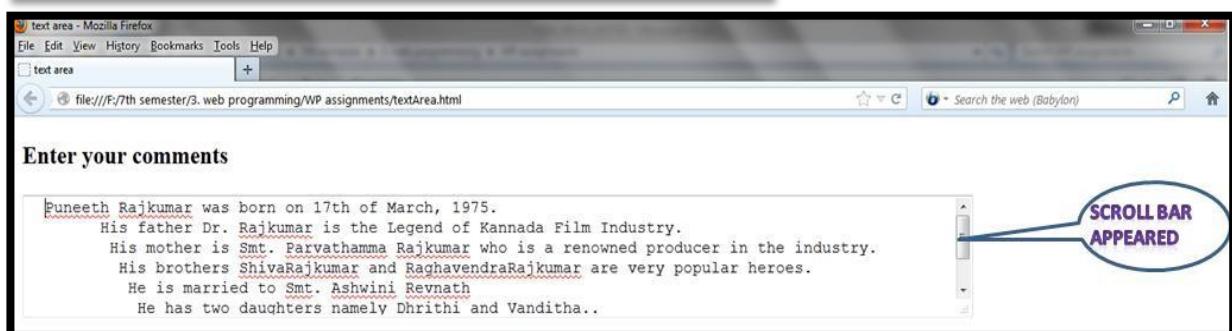
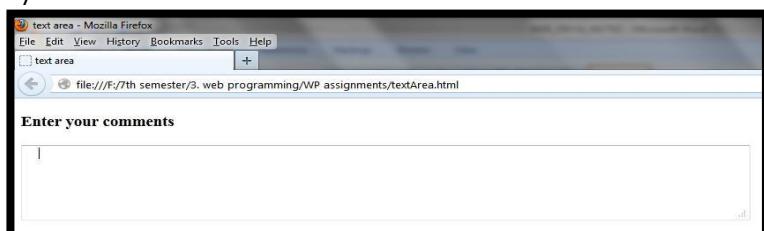
If you give `<select name = "branches" size = "3">`, then you will get a scroll bar instead of drop down menu. It is as shown in the output given below:



## The <textarea> Tag:

- Text area is a type of input using which multiple statements can be entered.
- Text area is created using <textarea> tag.
- Text area should have the name.
- During the creation of text area, it should be mentioned how many sentences can be entered. This is done using *rows* attribute.
- Similarly, it should also be mentioned how many characters can be entered in a line. This is done using *cols* attribute.
- If the value given to rows is exceeded i.e. if users enter sentences more than specified, the *scroll bar* automatically appears.

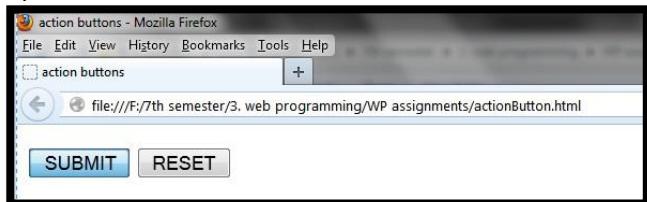
```
<html>
<head>
<title> text area </title>
</head>
<body>
<form action="">
<h3> Enter your comments</h3>
<p>
<textarea name="feedback" rows="5" cols="100">
</textarea>
</p>
</form>
</body>
</html>
```



## The Action Buttons:

The *Reset* button clears all of the controls in the form to their initial states. The *Submit* button has two actions: First, the form data is encoded and sent to the server; second, the server is requested to execute the server-resident program specified in the *action* attribute of the *<form>* tag. The purpose of such a server-resident program is to process the form data and return some response to the user. Every form requires a *Submit* button. The *Submit* and *Reset* buttons are created with the *<input>* tag.

```
<html>
<head>
<title> action buttons </title>
</head>
<body>
<form action="">
<p>
 <input type="SUBMIT" value="SUBMIT"/>
 <input type="RESET" value="RESET"/>
</p>
</form>
</body>
</html>
```



NOTE: A *plain* button has the type button. *Plain* buttons are used to choose an action.

## Example of a Complete Form:

```
<html>
<head>
<title> CompleteForm</title>
</head>
<body>
<h1>Registration Form</h1>
<form action="">
<p>
<label>Enter your email id:</label>
<input type = "text" name = "myname" size = "24" maxlength = "25" />
</p>
<p>
<label>Enter the password:</label>
<input type = "password" name = "mypass" size = "20" maxlength = "20" />
</p>
<p>Sex</p>
<p>
<label><input type="radio" name="act" value="one"/>Male</label>
<label><input type="radio" name="act" value="two"/>Female</label>
</p>
<p>Which of the following Accounts do you have?</p>
<p>
<label><input type="checkbox" name="act" value="one"/>Gmail</label>
```

```

<label><input type="checkbox" name="act" value="two"/>Facebook</label>
<label><input type="checkbox" name="act" value="three"/>Twitter</label>
<label><input type="checkbox" name="act" value="four"/>Google+</label>
</p>
<p> Any Suggestions?</p>
<p>
<textarea name="feedback" rows="5" cols="100">
</textarea>
</p>
<p>Click on Submit if you want to register</p>
<p>
<input type="SUBMIT" value="SUBMIT"/>
<input type="RESET" value="RESET"/>
</p>
</form>
</body>
</html>

```

CompleteForm - Mozilla Firefox

File Edit View History Bookmarks Tools Help

CompleteForm

file:///F:/7th semester/3. web programming/WP assignments/completeForm.html?myname=1rn09is016%40gmail.com&mypass=webprogramming&ac

Search the web (Babylon)

## Registration Form

Enter your email id: 1rn09is016@gmail.com

Enter the password: [REDACTED]

Sex

Male  Female

Which of the following Accounts do you have?

Gmail  Facebook  Twitter  Google+

Any Suggestions?

PLEASE PROVIDE A FASTER AND BETTER SERVICE..!!|

Click on Submit if you want to register

SUBMIT RESET

# Module II

## CASCADING STYLE SHEETS

### INTRODUCTION

XHTML style sheets are called *cascading* style sheets because they can be defined at three different levels to specify the style of a document. Lower level style sheets can override higher level style sheets, so the style of the content of a tag is determined, in effect, through a cascade of style-sheet applications.

### LEVELS OF STYLE SHEETS

- The three levels of style sheets, in order from lowest level to highest level, are inline, document level, and external.
- **Inline style sheets** apply to the content of a single XHTML element.
- **Document-level style sheets** apply to the whole body of a document.
- **External style sheets** can apply to the bodies of any number of documents.
- Inline style sheets have precedence over document style sheets, which have precedence over external style sheets.
- Inline style specifications appear within the opening tag and apply only to the content of that tag.
- Document-level style specifications appear in the document head section and apply to the entire body of the document.
- External style sheets stored separately and are referenced in all documents that use them.
- External style sheets are written as text files with the MIME type `text/css`.
- They can be stored on any computer on the Web. The browser fetches external style sheets just as it fetches documents.
- The `<link>` tag is used to specify external style sheets. Within `<link>`, the `rel` attribute is used to specify the relationship of the linked-to document to the document in which the link appears. The `href` attribute of `<link>` is used to specify the URL of the style sheet document.

#### EXAMPLE WHICH USES EXTERNAL STYLE SHEET

```

<html>
<head>
<title>Sample CSS</title>
<link rel = "stylesheet" type = "text/css"
 href = "Style1.css" />
</head>
<h1>Puneeth Rajkumar</h1>
</html>

```

#### Style1.css

```

h1
{
 font-family: 'Lucida Handwriting';
 font-size: 50pt;
 color: Red;
}

```



#### EXAMPLE WHICH USES DOCUMENT LEVEL STYLE SHEET

```

<html>
<head>
<title>Sample CSS</title>

```

```

<style type = "text/css">
h1
{
 font-family: 'Lucida Handwriting';
 font-size: 50pt;
 color: Red;
}
</style>
</head>
<h1>Puneeth Rajkumar</h1>
</html>

```



#### EXAMPLE WHICH USES INLINE STYLE SHEET

```

<html>
<head>
<title>Sample CSS</title>
</head>
<h1 style ="font-family: 'Lucida Handwriting'; font-size: 50pt; color: Red;">
 Puneeth Rajkumar </h1>
</html>

```



### STYLE SPECIFICATION FORMATS

#### Inline Style Specification:

*Style = "Property1 : Value1;      Property2 : Value2;      Property3 : Value3; ..... Property\_n : Value\_n;"*

#### Document Style Specification:

```

<style type = "text/css">
 Rule list
</style>

```

Each style rule in a rule list has two parts: a selector, which indicates the tag or tags affected by the rule, and a list of property-value pairs. The list has the same form as the quoted list for inline style sheets, except that it is delimited by braces rather than double quotes. So, the form of a style rule is as follows:

**Selector { Property1 : Value1; Property2 : Value2; Property3 : Value3; ..... Property\_n : Value\_n; }**

[For examples on all three levels of style sheets along with specifications, Please refer the previous examples]

## SELECTOR FORMS

### Simple Selector Forms:

In case of simple selector, a tag is used. If the properties of the tag are changed, then it reflects at all the places when used in the program. The selector can be any tag. If the new properties for a tag are not mentioned within the rule list, then the browser uses default behaviour of a tag.

```
<html>
<head>
<title>Sample CSS</title>
<style type = "text/css">
p
{
 font-family: 'Lucida Handwriting';
 font-size: 50pt;
 color: Red;
}
</style>
</head>
<body>
<p>Puneeth Rajkumar</p>
<p>Mahesh Babu</p>
<p>Suriya</p>
</body>
</html>
```

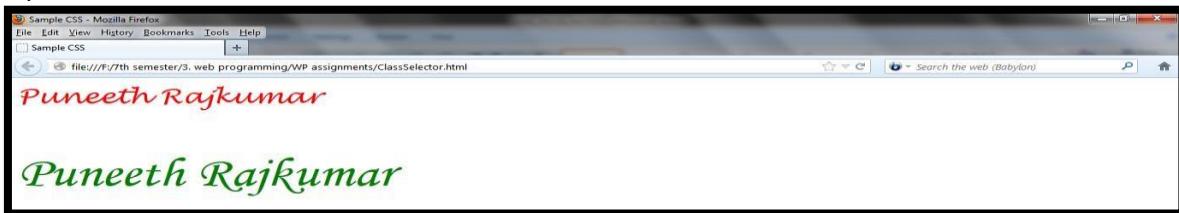


### Class Selectors:

In class selector, it is possible to give different properties for different elements

```
<html>
<head>
<title>Sample CSS</title>
<style type = "text/css">
p.one
{
 font-family: 'Lucida Handwriting';
 font-size: 25pt;
 color: Red;
}
p.two
{
 font-family: 'Monotype Corsiva';
 font-size: 50pt;
 color: green;
}
</style>
</head>
```

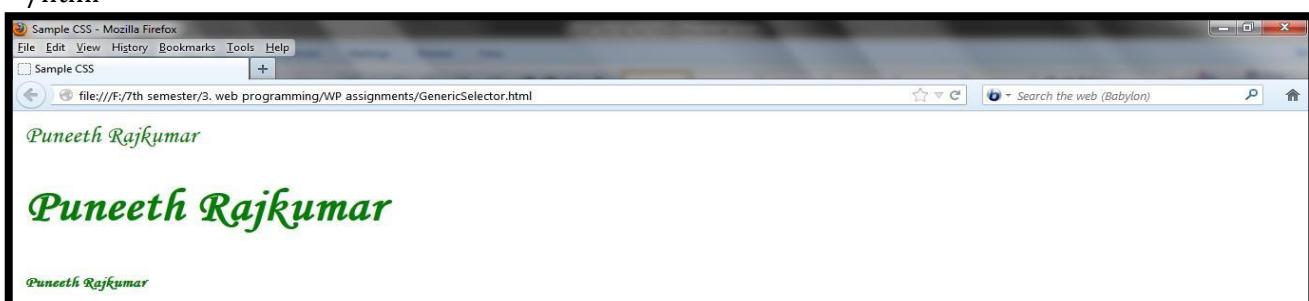
```
<body>
<p class = "one">Puneeth Rajkumar</p>
<p class = "two">Puneeth Rajkumar</p>
</body>
</html>
```



## Generic Selectors:

In case of generic selector, when the class is created, it would not be associated to any particular tag. In other words, it is generic in nature.

```
<html>
<head>
<title>Sample CSS</title>
<style type = "text/css">
.one
{
 font-family: 'Monotype Corsiva';
 color: green;
}
</style>
</head>
<body>
<p class = "one">Puneeth Rajkumar</p>
<h1 class = "one">Puneeth Rajkumar</h1>
<h6 class = "one">Puneeth Rajkumar</h6>
</body>
</html>
```



## id Selectors:

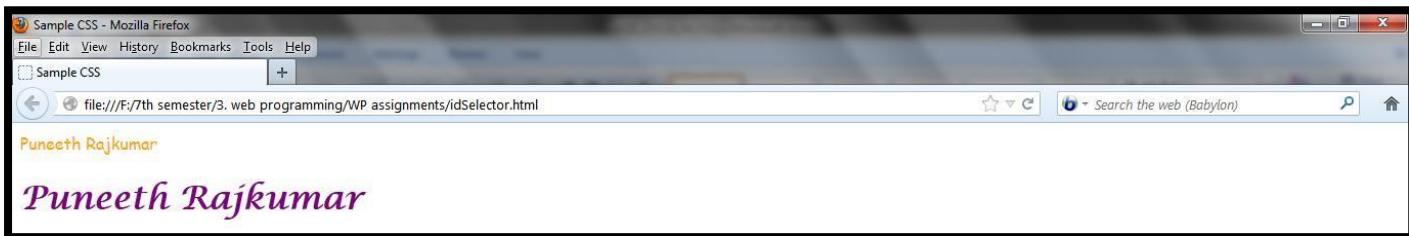
An `id` selector allows the application of a style to one specific element.

```
<html>
<head>
<title>Sample CSS</title>
<style type = "text/css">
#one
{
 font-family: 'lucida calligraphy';
 color: purple;
}
```

```

#two
{
 font-family: 'comic sans ms';
 color: orange;
}
</style>
</head>
<body>
 <p id = "two">Puneeth Rajkumar</p>
 <h1 id = "one">Puneeth Rajkumar</h1>
</body>
</html>

```



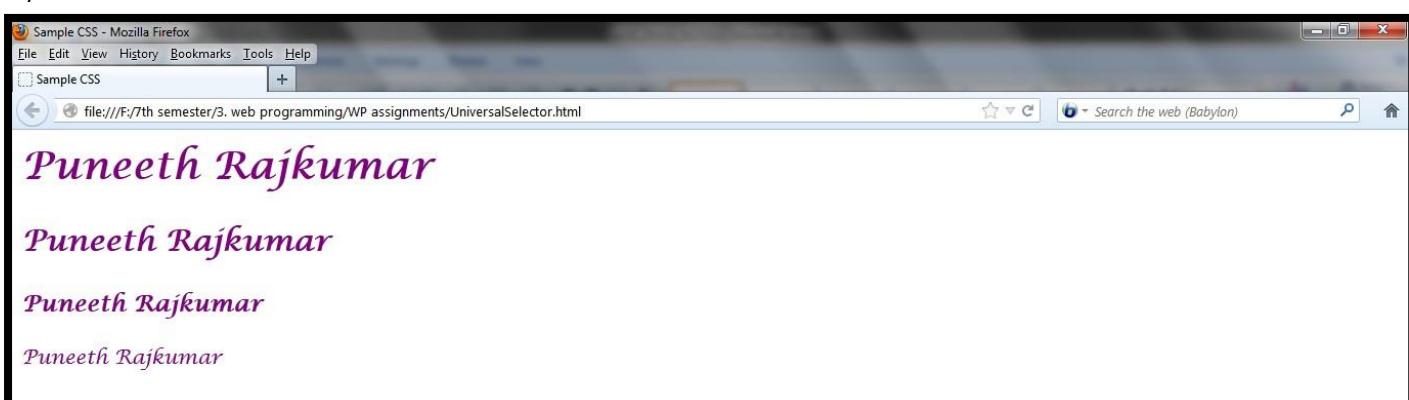
## Universal Selectors:

The universal selector, denoted by an asterisk (\*), applies its style to all elements in a document.

```

<html>
 <head>
 <title>Sample CSS</title>
 <style type = "text/css">
 *
 {
 font-family: 'lucida calligraphy';
 color: purple;
 }
 </style>
 </head>
 <body>
 <h1>Puneeth Rajkumar</h1>
 <h2>Puneeth Rajkumar</h2>
 <h3>Puneeth Rajkumar</h3>
 <p>Puneeth Rajkumar</p>
 </body>
</html>

```



## Pseudo Classes:

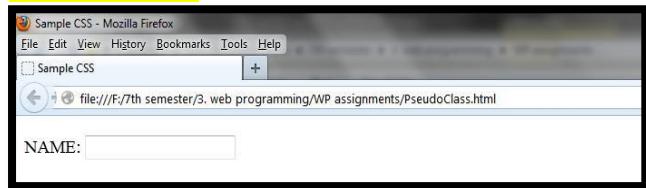
Pseudo class selectors are used if the properties are to be changed dynamically. For example: when mouse movement happens, in other words, hover happens or focus happens.

```

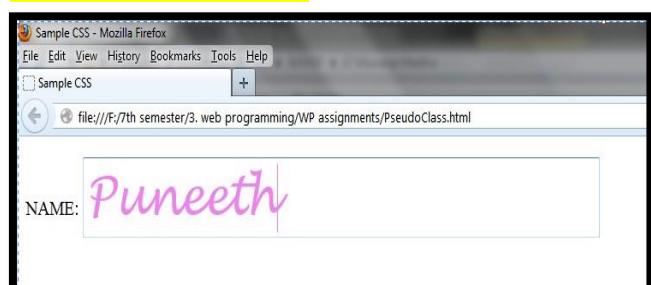
<html>
<head>
<title>Sample CSS</title>
<style type = "text/css">
input:focus
{
 font-family: 'lucida calligraphy';
 color: purple;
 font-size:100;
}
input:hover
{
 font-family: 'lucida handwriting';
 color: violet;
 font-size:40;
}
</style>
</head>
<body>
<form action = "">
<p>
<label>
 NAME:
 <input type = "text" />
</label>
</p>
</form>
</body>
</html>

```

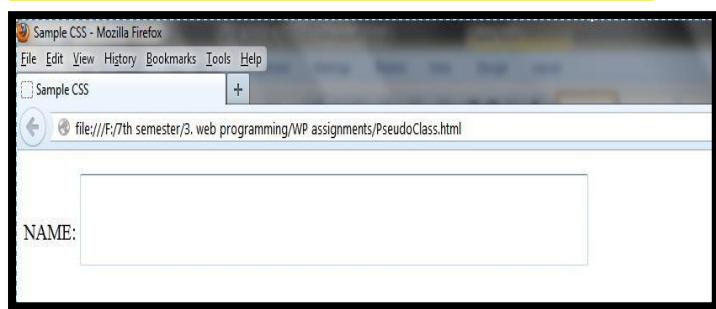
### STEP 1: Initial



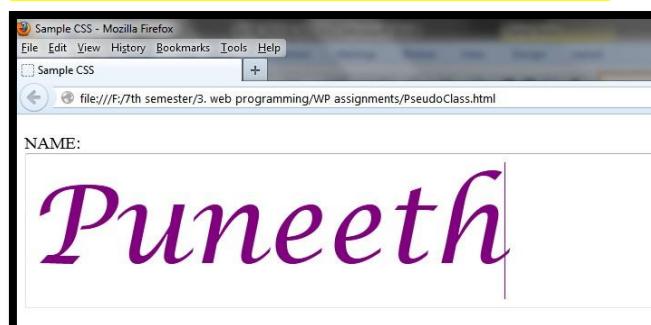
### STEP 3: Enter the data



### STEP 2: After placing mouse pointer on text area



### STEP 4: After taking away the mouse pointer



## PROPERTY VALUE FORMS

CSS includes 60 different properties in seven categories: fonts, lists, alignment of text, margins, colours, backgrounds, and borders. Property values can appear in a variety of forms.

- Keyword property values are used when there are only a few possible values and they are predefined.
- A number value can be either an integer or a sequence of digits with a decimal point and can be preceded by a sign (+ or -).
- Length values are specified as number values that are followed immediately by a two-character abbreviation of a unit name. The possible unit names are px, for pixels; in, for inches; cm, for centimeters; mm, for millimeters; pt, for points.
- Percentage values are used to provide a measure that is relative to the previously used measure for a property value. Percentage values are numbers that are followed immediately by a percent sign (%). Percentage values can be signed. If preceded by a plus sign, the percentage is added to the previous value; if negative, the percentage is subtracted.
- There can be no space between url and the left parenthesis.
- Color property values can be specified as color names, as six-digit hexadecimal numbers, or in RGB form. RGB form is just the word rgb followed by a parenthesized list of three numbers that specify the levels of red, green, and blue, respectively. The RGB values can be given either as decimal numbers between 0 and 255 or as percentages. Hexadecimal numbers must be preceded with pound signs (#), as in #43AF00.

## FONT PROPERTIES

### **Font Families:**

The font-family property is used to specify a list of font names. The browser uses the first font in the list that it supports. For example, the property:

**font-family: Arial, Helvetica, Futura**

tells the browser to use Arial if it supports that font. If not, it will use Helvetica if it supports it. If the browser supports neither Arial nor Helvetica, it will use Futura if it can. If the browser does not support any of the specified fonts, it will use an alternative of its choosing.

If a font name has more than one word, the whole name should be delimited by single quotes, as in the following example:

**font-family: 'Times New Roman'**

### **Font Sizes:**

The font-size property does what its name implies. For example, the following property specification sets the font size for text to 10 points:

**font-size: 10pt**

Many relative font-size values are defined, including xx-small, x-small, small, medium, large, x-large, and xx-large. In addition, smaller or larger can be specified. Furthermore, the value can be a percentage relative to the current font size.

### **Font Variants:**

The default value of the font-variant property is normal, which specifies the usual character font. This property can be set to small-caps to specify small capital characters. These characters are all uppercase, but the letters that are normally uppercase are somewhat larger than those that are normally lowercase.

### **Font Styles:**

The font-style property is most commonly used to specify italic, as in

**font-style: italic**

## Font Weights:

The `font-weight` property is used to specify the degree of boldness, as in

**`font-weight: bold`**

Besides `bold`, the values `normal`, `bolder`, and `lighter` can be specified. Specific numbers also can be given in multiples of 100 from 100 to 900, where 400 is the same as `normal` and 700 is the same as `bold`.

## Font Shorthands:

If more than one font property must be specified, the values can be stated in a list as the value of the `font` property. The order in which the property values are given in a `font` value list is important. The order must be as follows: The font names must be last, the font size must be second to last, and the font style, font variant, and font weight, when they are included, can be in any order but must precede the font size and fontnames.

**`font: bold 14pt 'Times New Roman'`**

```
<html>
<head>
<title>Font Properties</title>
<style type = "text/css">
p.one
{
 font-family: 'lucida calligraphy';
 font-weight:bold;
 font-size:75pt;
 color: purple;
}
h1.two
{
 font-family: 'cambria';
 color: violet;
 font-style:italics;
}
p.three
{
 font: small-caps italic bold 50pt 'times new roman'
}
</style>
</head>
<body>
<p class = "one">Puneeth Rajkumar</p>
<h1 class = "two">Puneeth Rajkumar</h1>
<p class = "three">Puneeth Rajkumar</p>
</body>
</html>
```



## Text Decoration:

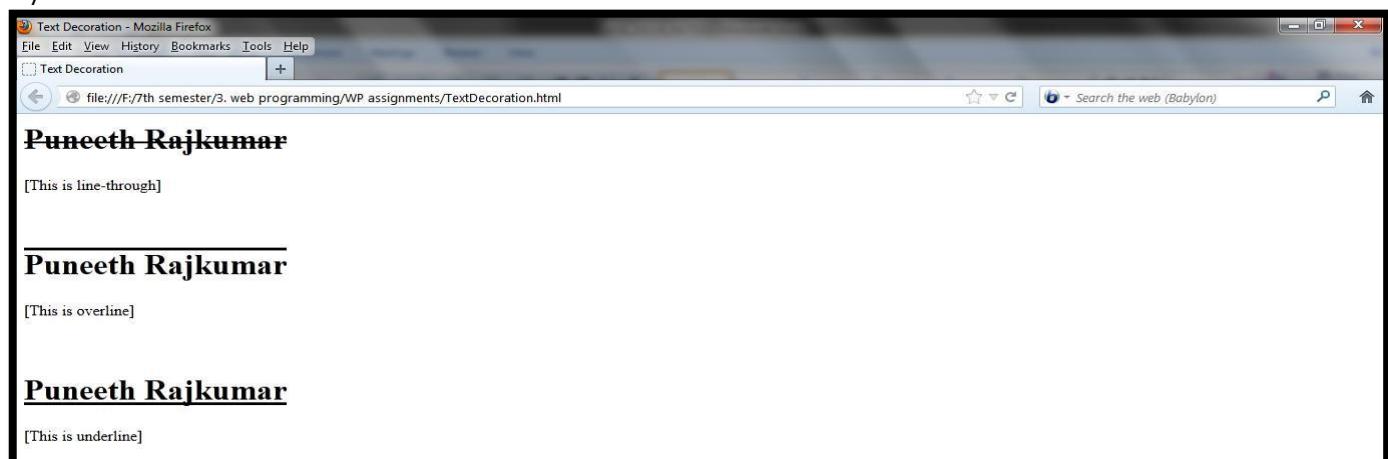
The `text-decoration` property is used to specify some special features of text. The available values are `line-through`, `overline`, `underline`, and `none`, which is the default.

```
<html>
<head>
<title>Text Decoration</title>
<style type = "text/css">
h1.one
{text-decoration: line-through;}
h1.two
{text-decoration: overline;}
h1.three
{text-decoration: underline;}
</style>
</head>
<body>
<h1 class = "one">Puneeth Rajkumar</h1> <p>[This is line-through]</p>

<h1 class = "two">Puneeth Rajkumar</h1> <p>[This is overline]</p>

<h1 class = "three">Puneeth Rajkumar</h1><p>[This is underline]</p>

</body>
</html>
```



## LIST PROPERTIES

Two presentation details of lists can be specified in XHTML documents: the shape of the bullets that precede the items in an unordered list and the sequencing values that precede the items in an ordered list. The `list-style-type` property is used to specify both of these.

The `list-style-type` property of an unordered list can be set to `disc`, `circle`, `square`, or `none`.

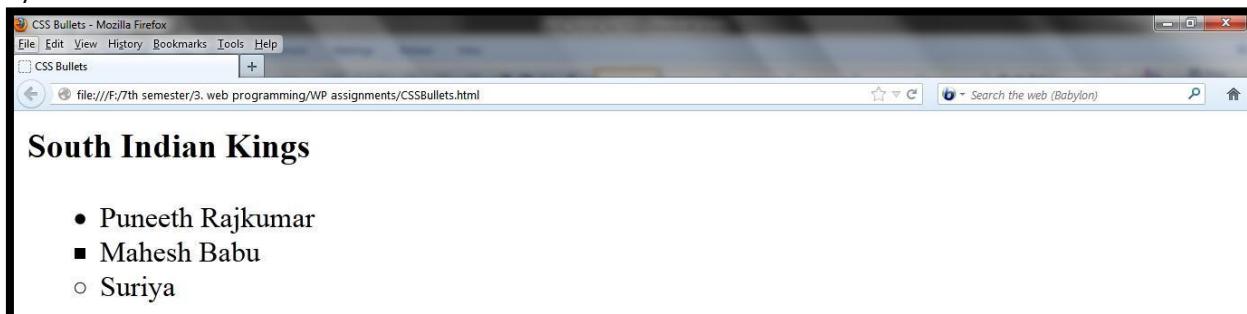
```
<html>
<head>
<title>CSS Bullets</title>
<style type = "text/css">
li.one {list-style-type:disc}
li.two{list-style-type:square}
li.three{list-style-type:circle}
</style>
</head>
<body>
<h3>South Indian Kings</h3>
```

```


 <li class = "one"> Puneeth Rajkumar
 <li class = "two"> Mahesh Babu
 <li class = "three"> Suriya

</body>
</html>

```



Bullets in unordered lists are not limited to discs, squares, and circles. Any image can be used in a list item bullet. Such a bullet is specified with the `list-style-image` property, whose value is specified with the `url` form.

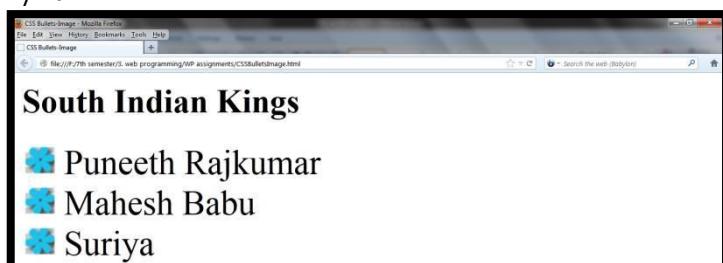
```

<html>
<head>
<title>CSS Bullets-Image</title>
<style type = "text/css">
 li.image {list-style-image: url(bullet.png); font-size:25pt;}
</style>
</head>
<body>
 <h1>South Indian Kings</h1>

 <li class = "image"> Puneeth Rajkumar
 <li class = "image"> Mahesh Babu
 <li class = "image"> Suriya

</body>
</html>

```



The following example illustrates the use of different sequence value types in nested lists:

```

<html>
<head>
<title> CSS nested lists </title>
<style type = "text/css">
 ol {list-style-type:upper-roman;}
 ol ol {list-style-type:upper-alpha;}
 ol ol ol {list-style-type:decimal;}
</style>
</head>

```

```

- Information Science

- OOMD
- Java & J2ee

- classes and methods
- exceptions
- applets
- servelets

- Computer Networks

- Part 1
- Part 2

- DBMS
- Operations Research

- Computer Science

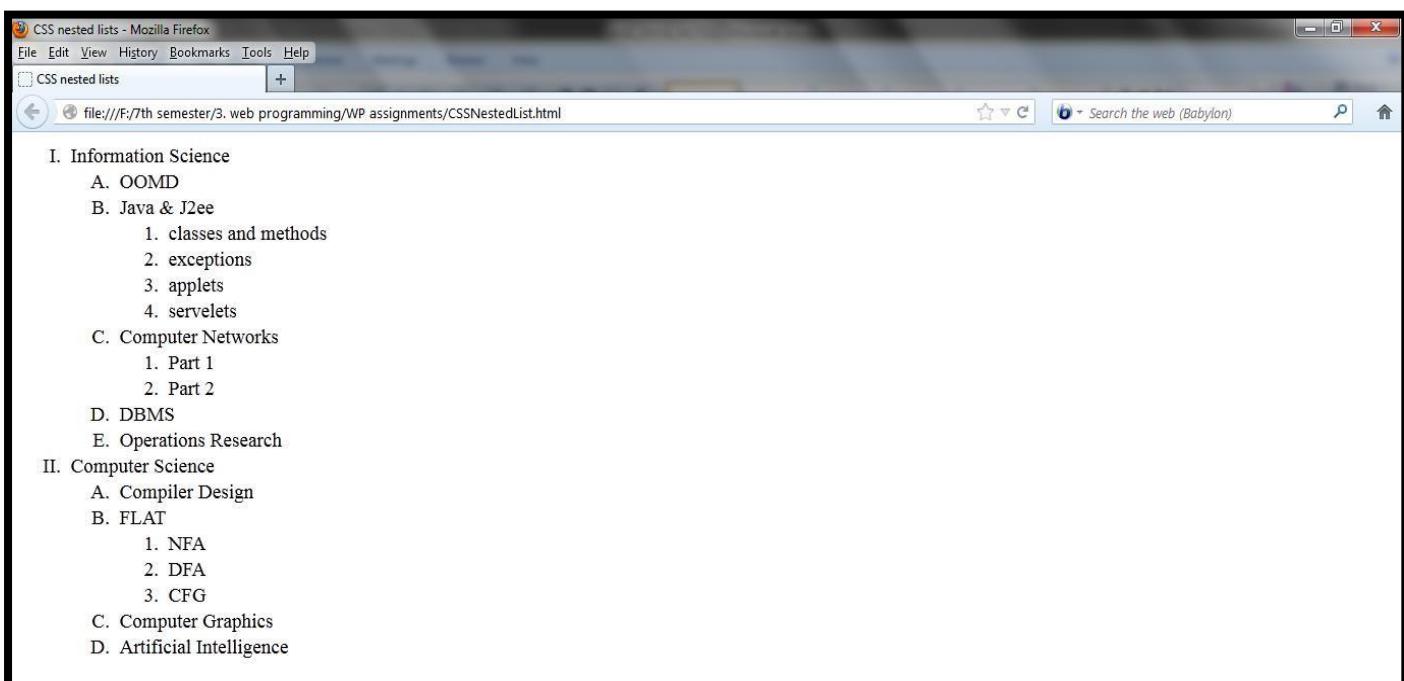
- Compiler Design
- FLAT

- NFA
- DFA
- CFG

- Computer Graphics
- Artificial Intelligence

</html>

```



## COLOR

### Color Groups:

Three levels of collections of colours might be used by an XHTML document. The smallest useful set of colours includes only those that have standard names and are guaranteed to be correctly displayable by all browsers on all color monitors. This collection of 17 colours is called the *named colours*.

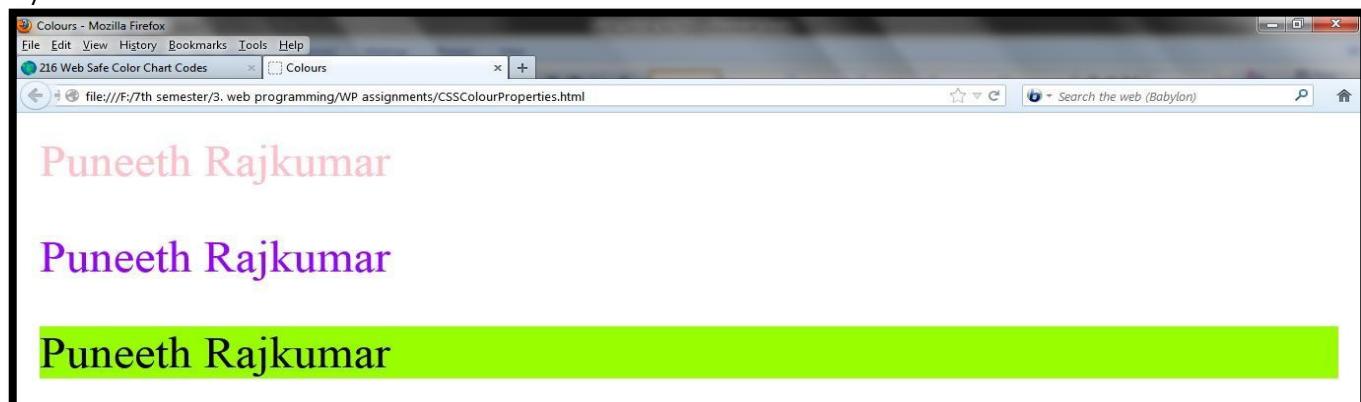
Name	Hexadecimal Code	Name	Hexadecimal Code
aqua	00FFFF	olive	808000
black	000000	orange	FFA500
blue	0000FF	purple	800080
fuchsia	FF00FF	red	FF0000
gray	808080	silver	C0C0C0
green	008000	teal	008080
lime	00FF00	white	FFFFFF
maroon	800000	yellow	FFFF00
navy	000080		

Larger set of colors, called the Web palette, consists of 216 colors. The colors of the Web palette can be viewed at [http://www.web-source.net/216\\_color\\_chart.htm](http://www.web-source.net/216_color_chart.htm)

### Color Properties:

The `color` property is used to specify the foreground color of XHTML elements.

```
<html>
<head>
<title>Colours</title>
<style type = "text/css">
p.one
{color: pink; }
p.two
{color: # 9900FF; }
p.three
{background-color:#99FF00;}
</style>
</head>
<body>
<p class = "one">Puneeth Rajkumar</p>
<p class = "two">Puneeth Rajkumar</p>
<p class = "three">Puneeth Rajkumar</p>
</body>
</html>
```



## ALIGNMENT OF TEXT

- The `text-indent` property can be used to indent the first line of a paragraph. This property takes either a length or a percentage value. The `text-align` property, for which the possible keyword values are left, center, right, and justify, is used to arrange text horizontally.
- The `float` property is used to specify that text should flow around some element, often an image or a table. The possible values for `float` are left, right, and none, which is the default.

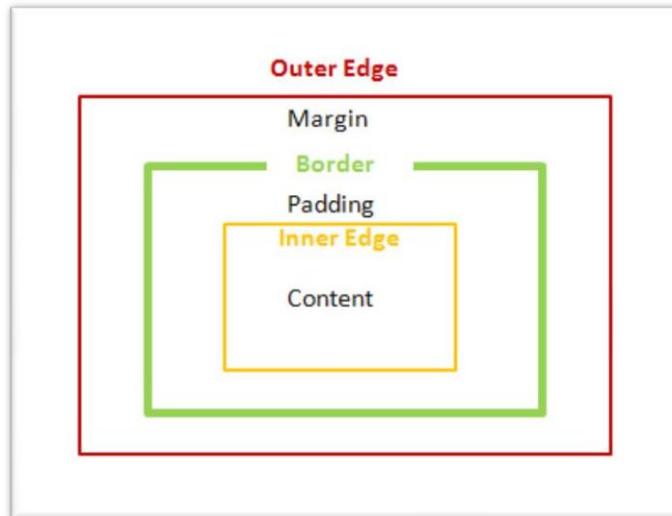
```
<html>
<head>
<title>Text Alignment</title>
<style type = "text/css">
h1.one
{text-align: center}
p.two
{text-indent: 0.5in; text-align: justify;}
img{float:right}
</style>
</head>
<body>
<h1 class = "one">Kannadada Kotyadhipathi</h1>
<p>

</p>
<p class = "two">Kannadada Kotyadhipathi is a Kannada primetime quiz show hosted by the power star of Kannada cinema Mr. Puneet Rajkumar. This is the biggest game show ever on Kannada Television. This show will be aired on Suvarna TV. This show gives the common man an opportunity to win Rs 1 crore. Kannadada Kotyadipathi is a Kannada primetime quiz and human drama show hosted by matinee idol Puneeth Rajkumar on Suvarna TV. Contestants participate in a game that allows them to win up to Rs. 1 crore. Short-listed contestants play a 'Fastest Finger First' round to make it to the main game. From there on, they play rounds with increasing levels of difficulty, and winning higher amounts of money, culminating in the Rs. 1 crore prize. Contestants can stop at any time having viewed the next question. Or they can avail of a 'Lifeline' and play on. Welcome to the world of high stakes chills and thrills! Welcome to the world of the crorepati!</p>
</body>
</html>
```



## THE BOX MODEL

- On a given web page or a document, all the elements can have borders.
- The borders have various styles, color and width.
- The amount of space between the content of the element and its border is known as *padding*.
- The space between border and adjacent element is known as *margin*.



## Borders:

### Border-style

It can be dotted, dashed, double

- ▶ Border-top-style
- ▶ Border-bottom-style
- ▶ Border-left-style
- ▶ Border-right-style

### Border-width

It can be thin, medium, thick or any length value

- ▶ Border-top-width
- ▶ Border-bottom-width
- ▶ Border-left-width
- ▶ Border-right-width

### Border-color

- ▶ Border-top-color
- ▶ Border-bottom-color
- ▶ Border-left-color
- ▶ Border-right-color

```
<html>
<head>
<title> Table with border effects </title>
<style type = "text/css">
table
{
 border-width:thick;
 border-top-color:red;
 border-left-color:orange;
 border-bottom-color:violet;
 border-right-color:green;
 border-top-style:dashed;
```

```

 border-bottom-style:double;
 border-right-style:dotted;
 }

```

```

</style>
</head>
<body>
<table border = "border">
<caption>PARAMATHMA </caption>
<tr>
 <td> Puneeth Rajkumar </td>
 <td> </td>
</tr>
</table>
</body>
</html>

```



## Margins and Padding:

The margin properties are named `margin`, which applies to all four sides of an element: `margin-left`, `margin-right`, `margin-top`, and `margin-bottom`. The padding properties are named `padding`, which applies to all four sides: `padding-left`, `padding-right`, `padding-top`, and `padding-bottom`.

```

<html>
<head>
<title> Margins and Padding </title>
<style type = "text/css">
.p.one
{
 margin:0.1in;
 padding:0.5in;
 background-color:#FF33FF;
 border-style:dotted;
}
.p.two
{
 margin:0.5in;
 padding:0.1in;
 background-color:#00FF33;
 border-style:dashed;
}
.p.three
{

```

```

margin:0.3in;
background-color:#FFFF00;
}
p.four
{
padding:0.3in;
background-color:#FF9900;
}

```

</style>

</head>

<body>

<p class = "one"> Puneeth Rajkumar is the Power Star of Sandalwood<br/>  
[margin=0.1in, padding=0.5in]</p>

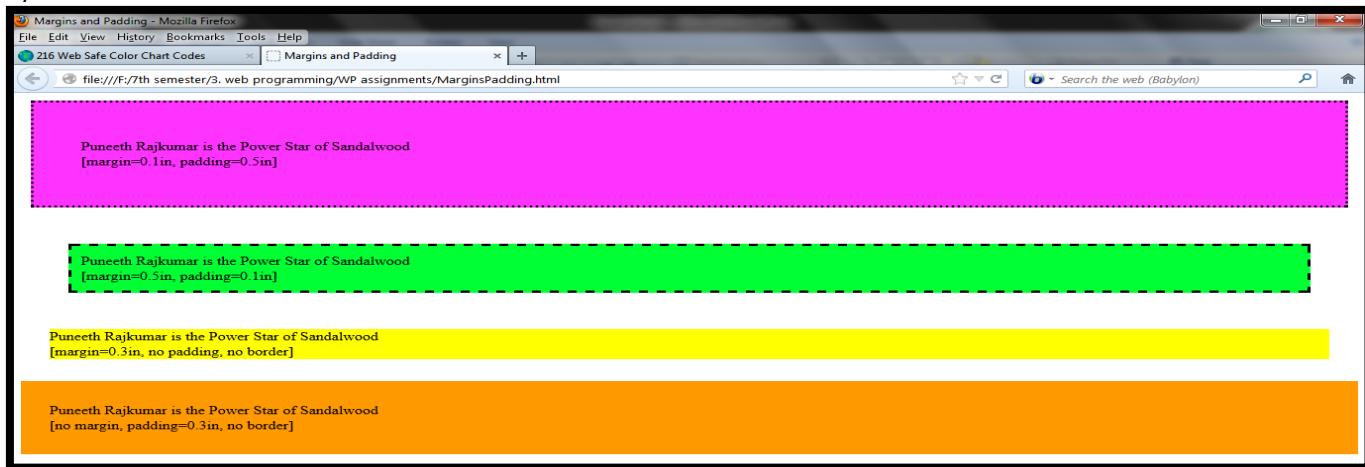
<p class = "two"> Puneeth Rajkumar is the Power Star of Sandalwood<br/>  
[margin=0.5in, padding=0.1in]</p>

<p class = "three"> Puneeth Rajkumar is the Power Star of Sandalwood<br/>  
[margin=0.3in, no padding, no border]</p>

<p class = "four"> Puneeth Rajkumar is the Power Star of Sandalwood<br/>  
[no margin, padding=0.3in, no border]</p>

</body>

</html>



## BACKGROUND IMAGES

The `background-image` property is used to place an image in the background of an element.

```

<html>
<head>
<title>Background Image</title>
<style type = "text/css">
body {background-image:url(bg3.jpg);}
p
{text-align: justify; color:white;font-size:25pt;}
</style>
</head>
<body>

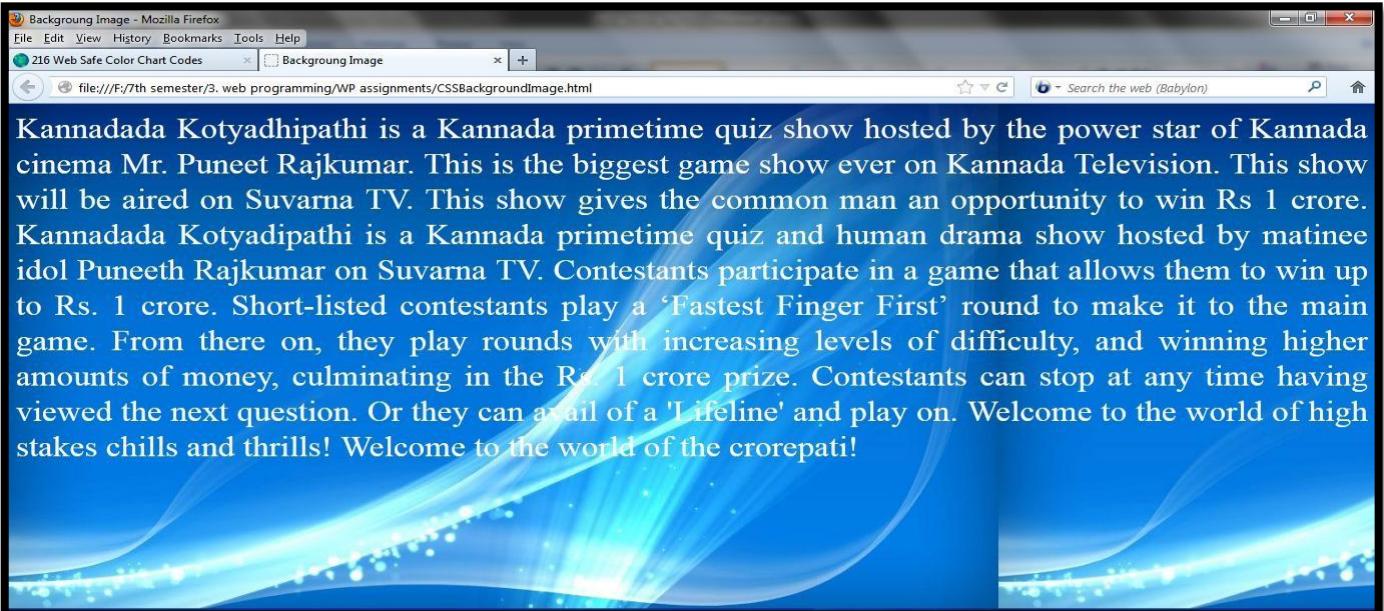
```

<p>Kannadada Kotyadhipathi is a Kannada primetime quiz show hosted by the power star of Kannada cinema Mr. Puneet Rajkumar. This is the biggest game show ever on Kannada Television. This show will be aired on Suvarna TV. This show gives the common man an opportunity to win Rs 1 crore. Kannadada Kotyadipathi is a Kannada primetime quiz and human drama show hosted by matinee idol Puneeth Rajkumar on Suvarna TV. Contestants participate in a game that allows them to win up to Rs. 1 crore. Short-listed

contestants play a 'Fastest Finger First' round to make it to the main game. From there on, they play rounds with increasing levels of difficulty, and winning higher amounts of money, culminating in the Rs. 1 crore prize. Contestants can stop at any time having viewed the next question. Or they can avail of a 'Lifeline' and play on. Welcome to the world of high stakes chills and thrills! Welcome to the world of the crorepati!

```
</body>
```

```
</html>
```



In the example, notice that the background image is replicated as necessary to fill the area of the element. This replication is called *tiling*. Tiling can be controlled with the `background-repeat` property, which can take the value `repeat` (the default), `no-repeat`, `repeat-x`, or `repeat-y`. The `no-repeat` value specifies that just one copy of the image is to be displayed. The `repeat-x` value means that the image is to be repeated horizontally; `repeat-y` means that the image is to be repeated vertically. In addition, the position of a non-repeated background image can be specified with the `background-position` property, which can take a large number of different values. The keyword values are `top`, `center`, `bottom`, `left`, and `right`, all of which can be used in many different combinations.

### THE `<span>` AND `<div>` TAGS

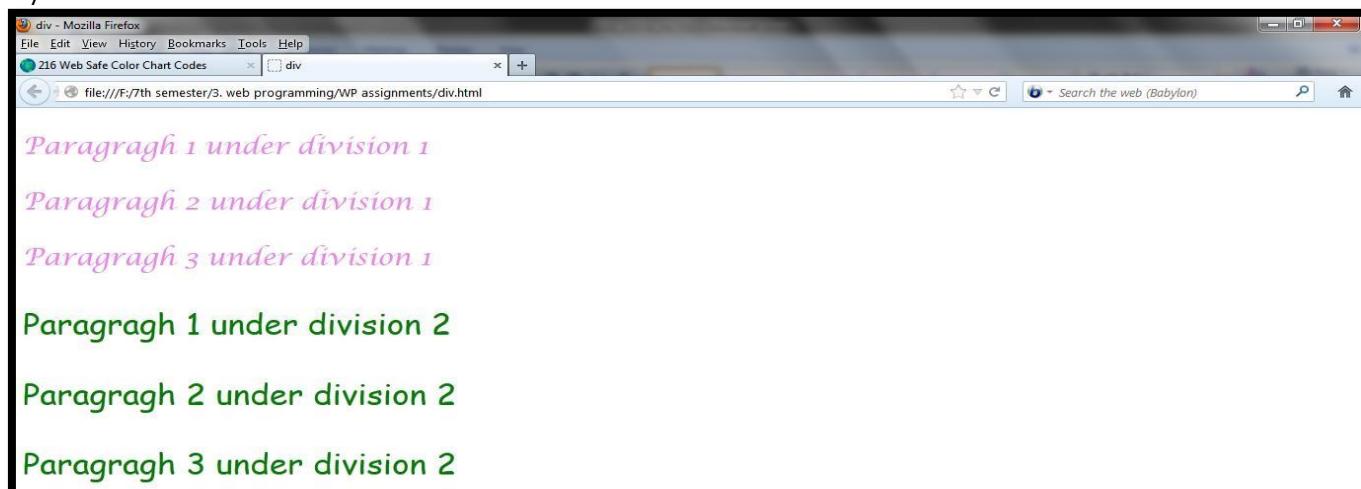
In many situations, we want to apply special font properties to less than a whole paragraph of text. The `<span>` tag is designed for just this purpose.

```
<html>
 <head> <title>span</title>
 <style type = "text/css">
 .spanviolet {font-size:25pt;font-family:'lucida calligraphy';color:violet;}
 </style>
 </head>
 <body>
 <p>Kannadada Kotyadhipathi is a Kannada primetime quiz show hosted by
 Puneeth Rajkumar , the power star of Kannada cinema </p>
 </body>
</html>
```



It is more convenient, however, to be able to apply a style to a section of a document rather than to each paragraph. This can be done with the `<div>` tag. As with `<span>`, there is no implied layout for the content of the `<div>` tag, so its primary use is to specify presentation details for a section or division of a document.

```
<html>
<head>
<title>div</title>
<style type = "text/css">
.one
{font-size:20pt;font-family:'lucida calligraphy';color:violet;}
.two
{font-size:25pt;font-family:'comic sans ms';color:green;}
</style>
</head>
<body>
<div class = "one">
<p>Paragragh 1 under division 1</p>
<p>Paragragh 2 under division 1</p>
<p>Paragragh 3 under division 1</p>
</div>
<div class = "two">
<p>Paragragh 1 under division 2</p>
<p>Paragragh 2 under division 2</p>
<p>Paragragh 3 under division 2</p>
</div>
</body>
</html>
```



## CONFLICT RESOLUTION

- Sometimes on a web page, there can be two different values for the same property on the same element leading to conflict.
- `h3 {color: blue;}`  
`body h3 {color: red;}`
- The browser has to resolve this conflict.
- There can be one or more type of conflict: i.e. when style sheets at 2 or more levels specify different value for same property on some element.
- This conflict is resolved by providing priority to the different levels of style sheets.
- The inline level gets the highest priority over the document level.

- The document level gets the higher priority over the external level
  - But the browser must be able to resolve the conflict in the first example using same technique.
  - There can be several different origins of the specification of property values.
  - One of the value may come from a style sheet created by the author or it can be specified by the user using the options provided by the browser.
  - The property values with different origin have different precedence.
  - The precedence can also be set for a property by marking it as important.
  - **p.special {font-style: italic !important; font-size: 14}**
  - This means that font-style:italic is important [this is known as weight of specification]
  - The process of conflict resolution is a multi-stage sorting process.
  - The first step is to gather information about levels of style sheet.
  - Next, all the origins and weights are sorted. The following rules are considered:
    1. Important declarations with user origin
    2. Important declarations with author origin
    3. Normal declarations with author origin
    4. Normal declarations with user origin
    5. Any declarations with browser (or other user agent) origin
  - If there are other conflicts even after sorting, the next step is sorting by specificity. Rules are:
    1. id selectors
    2. Class and pseudo class selectors
    3. Contextual selectors (more element type names means that they are more specific)
    4. Universal selectors
  - If there still conflicts, they are resolved by giving precedence to most recently seen specification.
-



# UNIT 4

## JAVASCRT

### OVERVIEW OF JAVASCRIPT

#### ORIGINS

- JavaScript, which was developed by Netscape, was originally named Mocha but soon was renamed LiveScript.
- In late 1995 LiveScript became a joint venture of Netscape and Sun Microsystems, and its name again was changed, this time to JavaScript.
- A language standard for JavaScript was developed in the late 1990s by the European Computer Manufacturers Association (ECMA) as ECMA-262.
- The official name of the standard language is ECMAScript.
- JavaScript can be divided into three parts: the core, client side, and server side.
- The **core** is the heart of the language, including its operators, expressions, statements, and subprograms.
- **Client-side** JavaScript is a collection of objects that support the control of a browser and interactions with users.
- **Server-side** JavaScript is a collection of objects that make the language useful on a Web server.

#### JAVASCRIPT AND JAVA

JAV A	JAVASCRIP T
Java is programming language	JavaScript is a scripting language
It is strongly typed language	It is dynamically typed language
Types are known at compile time	Compile time type checking is impossible
Objects in java are static	JavaScript objects are dynamic
Collection of data members and methods is fixed at compile time	The number of data members and methods of an object can change during execution
Object oriented programming language	Object based language

#### USES OF JAVASCRIPT

- ♥ The JavaScript was initially introduced to provide programming capability at both the server and client ends of web connection
- ♥ JavaScript therefore is implemented at 2 ends:
  - ♣ Client end
  - ♣ Server end
- ♥ The client side JavaScript is embedded in XHTML documents and is interpreted by the browser
- ♥ It also provides some means of computation, which serves as an alternative for some tasks done at the server side
- ♥ Interactions with users through form elements, such as buttons and menus, can be conveniently described in JavaScript. Because button clicks and mouse movements are easily detected with JavaScript, they can be used to trigger computations and provide feedback to the user.
- ♥ For example, when a user moves the mouse cursor from a text box, JavaScript can detect that movement and check the appropriateness of the text box's value (which presumably was just filled by the user).
- ♥ Even without forms, user interactions are both possible and simple to program in JavaScript. These interactions, which take place in dialog windows, include getting input from the user and allowing the

user to make choices through buttons. It is also easy to generate new content in the browser display dynamically.

- ♥ This transfer of task ensures that the server is not overloaded and performs only required task
- ♥ But client side JavaScript cannot replace server side JavaScript; because server side software supports file operations, database access, security, networking etc
- ♥ JavaScript is also used as an alternative to java applets.
- ♥ Programming in JavaScript is much simpler than compared to java
- ♥ JavaScript support DOM [Document Object Model] which enables JavaScript to access and modify CSS properties and content of any element of a displayed XHTML document

## EVENT-DRIVEN COMPUTATION OF JAVASCRIPT

- In JavaScript, the actions are often executed in response to actions of the users of documents like mouse clicks and form submissions.
- This form of computation supports user interactions through the XHTML form elements on the client display
- One of the common uses of JavaScript is to check the values provided in forms by users to determine whether the values are sensible.
- The program or script on the server that processes the form data must check for invalid input data.
- When invalid data is found, the server must transmit that information back to the browser.
- Since this process is time consuming, we can perform input checks at the client side itself which saves both server time and internet time.
- However, validity checking is done on the server side because client side validity checking can be subverted by an unscrupulous user.

## BROWSERS AND XHTML/JAVASCRIPT DOCUMENTS

- ★ If an XHTML document does not include embedded scripts, the browser reads the lines of the document and renders its window according to the tags, attributes, and content it finds.
- ★ When a JavaScript script is encountered in the document, the browser uses its JavaScript interpreter to “execute” the script.
- ★ Output from the script becomes the next markup to be rendered.
- ★ When the end of the script is reached, the browser goes back to reading the XHTML document and displaying its content.
- ★ There are two different ways to embed JavaScript in an XHTML document: implicitly and explicitly.
- ★ In explicit embedding, the JavaScript code physically resides in the XHTML document.
- ★ The JavaScript can be placed in its own file, separate from the XHTML document. This approach, called implicit embedding, has the advantage of hiding the script from the browser user.
- ★ When JavaScript scripts are explicitly embedded, they can appear in either part of an XHTML document—the head or the body—depending on the purpose of the script.

## OBJECT ORIENTATION AND JAVASCRIPT

- JavaScript is an object-based language
- It supports prototype-based inheritance
- Without class-based inheritance, JavaScript cannot support polymorphism.
- A polymorphic variable can reference related methods of objects of different classes within the same class hierarchy

## JAVASCRIPT OBJECTS

- In JavaScript, objects are collections of properties, which correspond to the members of classes in Java and C++.
- Each property is either a data property or a function or method property.
- Data properties appear in two categories: primitive values and references to other objects.

- JavaScript uses non-object types for some of its simplest types; these non-object types are called *primitives*.
- Primitives are used because they often can be implemented directly in hardware, resulting in faster operations on their values.
- All objects in a JavaScript program are indirectly accessed through variables.
- All primitive values in JavaScript are accessed directly—these are like the scalar types in Java and C++. These are often called *value types*.
- The properties of an object are referenced by attaching the name of the property to the variable that references the object.
- A JavaScript object appears, both internally and externally, as a list of property–value pairs.
- The properties are names; the values are data values or functions.
- All functions are objects and are referenced through variables.
- The collection of properties of a JavaScript object is dynamic: Properties can be added or deleted at any time.

## GENERAL SYNTACTIC CHARACTERISTICS

- Scripts can appear directly as the content of a `<script>` tag.
  - The type attribute of `<script>` must be set to “**text/javascript**”.
  - The JavaScript script can be indirectly embedded in an XHTML document with the src attribute of a `<script>` tag, whose value is the name of a file that contains the script—for example,
- ```
<script type = "text/javascript" src = "tst_number.js" >
</script>
```
- Notice that the script element requires the closing tag, even though it has no content when the src attribute is included.
 - In JavaScript, identifiers, or names, must begin with a letter, an underscore (_), or a dollar sign (\$). Subsequent characters may be letters, underscores, dollar signs, or digits. There is no length limitation for identifiers.
 - JavaScript has 25 reserved words

| | | | | |
|-----------------------|----------------------|-------------------------|---------------------|---------------------|
| <code>break</code> | <code>delete</code> | <code>function</code> | <code>return</code> | <code>typeof</code> |
| <code>case</code> | <code>do</code> | <code>if</code> | <code>switch</code> | <code>var</code> |
| <code>catch</code> | <code>else</code> | <code>in</code> | <code>this</code> | <code>void</code> |
| <code>continue</code> | <code>finally</code> | <code>instanceof</code> | <code>throw</code> | <code>while</code> |
| <code>default</code> | <code>for</code> | <code>new</code> | <code>try</code> | <code>with</code> |

- In addition, JavaScript has a large collection of predefined words, including `alert`, `open`, `java`, and `self`.
- JavaScript has two forms of comments, both of which are used in other languages. First, whenever two adjacent slashes (//) appear on a line, the rest of the line is considered a comment. Second, /* may be used to introduce a comment, and */ to terminate it, in both single- and multiple-line comments.
- The XHTML comment used to hide JavaScript uses the normal beginning syntax, <!--.
- The following XHTML comment form hides the enclosed script from browsers that do not have JavaScript interpreters, but makes it visible to browsers that do support JavaScript:

```
<!--
-- JavaScript script --
// -->
```

- The use of semicolons in JavaScript is unusual. The JavaScript interpreter tries to make semicolons unnecessary, but it does not always work.
- When the end of a line coincides with what could be the end of a statement, the interpreter effectively inserts a semicolon there. But this can lead to problems. For example,

```
return  
x;
```

- The interpreter will insert a semicolon after return, making x an invalid orphan.
- The safest way to organize JavaScript statements is to put each on its own line whenever possible and terminate each statement with a semicolon. If a statement does not fit on a line, be careful to break the statement at a place that will ensure that the first line does not have the form of a complete statement.

```
<?xml version = "1.0" encoding = "utf-8"?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
  
<!-- hello.html  
A trivial hello world example of XHTML/JavaScript  
-->  
<html xmlns = "http://www.w3.org/1999/xhtml">  
  <head>  
    <title> Hello world </title>  
  </head>  
  <body>  
    <script type = "text/javascript">  
      <!--  
        document.write("Hello, fellow Web programmers!");  
      // -->  
    </script>  
  </body>  
</html>
```

PRIMITIVES, OPERATIONS, AND EXPRESSIONS

PRIMITIVE TYPES

- JavaScript has five primitive types: Number, String, Boolean, Undefined, and Null.
- Each primitive value has one of these types.
- JavaScript includes predefined objects that are closely related to the Number, String, and Boolean types, named **Number**, **String**, and **Boolean**, respectively.
- These objects are called wrapper objects.
- Each contains a property that stores a value of the corresponding primitive type.
- The purpose of the wrapper objects is to provide properties and methods that are convenient for use with values of the primitive types.
- The difference between primitives and objects is shown in the following example.

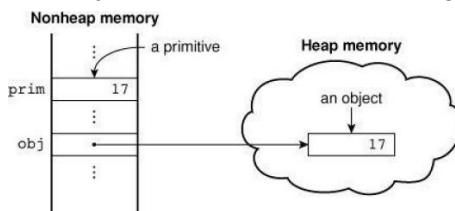


Figure 4.1 Primitives and objects

NUMERIC AND STRING LITERALS

- All numeric literals are values of type Number. The Number type values are represented internally in double-precision floating-point form.
- Integer literals are strings of digits.
- Floating-point literals can have decimal points, exponents, or both.
- Exponents are specified with an uppercase or lowercase e and a possibly signed integer literal.
- The following are valid numeric literals:

72 7.2 .72 72. 7E2 7e2 .7e2 7.e2 7.2E-2

- Integer literals can be written in hexadecimal form by preceding their first digit with either 0x or 0X.
- A string literal is a sequence of zero or more characters delimited by either single quotes ('') or double quotes ("").
- String literals can include characters specified with escape sequences, such as \n and \t. If you want an actual single-quote character in a string literal that is delimited by single quotes, the embedded single quote must be preceded by a backslash:
`“You’re the most lovely person I’ve ever met”`
- A double quote can be embedded in a double-quoted string literal by preceding it with a backslash. An actual backslash character in any string literal must be itself back-slashed, as in the following example:
`“D:\\bookfiles”`
- There is no difference between single-quoted and double-quoted literal strings.
- The null string (a string with no characters) can be denoted with either “” or “”.

OTHER PRIMITIVE TYPES

- The only value of type Null is the reserved word `null`, which indicates no value.
- The only value of type Undefined is `undefined`.
- The only values of type Boolean are `true` and `false`.

DECLARING VARIABLES

A variable can be declared either by assigning it a value, in which case the interpreter implicitly declares it to be a variable, or by listing it in a declaration statement that begins with the reserved word `var`. Initial values can be included in a `var` declaration, as with some of the variables in the following declaration:

```
var counter,
    index,
    pi = 3.14159265,
    quarterback = "Elway",
    stop_flag = true;
```

A variable that has been declared but not assigned a value, has the value `undefined`.

NUMERIC OPERATORS

- JavaScript has the typical collection of numeric operators: the binary operators `+` for addition, `-` for subtraction, `*` for multiplication, `/` for division, and `%` for modulus.
- The unary operators are plus `(+)`, negate `(-)`, decrement `(--)`, and increment `(++)`. The increment and decrement operators can be either prefix or postfix.
- For example, if the variable `a` has the value 7, the value of the following expression is 24:
`(++a) * 3`
- But the value of the following expression is 21:
`(a++) * 3`
- In both cases, `a` is set to 8.
- All numeric operations are done in double-precision floating point.
- The **precedence rules** of a language specify which operator is evaluated first when two operators with different precedence are adjacent in an expression.
- The **associativity rules** of a language specify which operator is evaluated first when two operators with the same precedence are adjacent in an expression.

Operator	Associativity
<code>++, --, unary -, unary +</code>	Right (though it is irrelevant)
<code>*, /, %</code>	Left
<code>Binary +, binary -</code>	Left

THE Math OBJECT

The `Math` object provides a collection of properties of `Number` objects and methods that operate on `Number` objects. The `Math` object has methods for the trigonometric functions, such as `sin` (for sine) and `cos` (for cosine), as well as for other commonly used mathematical operations.

Among these are `floor`, to truncate a number; `round`, to round a number; and `max`, to return the largest of two given numbers.

THE NUMBER OBJECT

The Number object includes a collection of useful properties that have constant values. Table 4.3 lists the properties of Number. These properties are referenced through Number.

Property	Meaning
<code>MAX_VALUE</code>	Largest representable number
<code>MIN_VALUE</code>	Smallest representable number
<code>NaN</code>	Not a number
<code>POSITIVE_INFINITY</code>	Special value to represent infinity
<code>NEGATIVE_INFINITY</code>	Special value to represent negative infinity
<code>PI</code>	The value of π

Any arithmetic operation that results in an error (e.g., division by zero) or that produces a value that cannot be represented as a double-precision floating-point number, such as a number that is too large (an overflow), returns the value “not a number,” which is displayed as NaN. If NaN is compared for equality against any number, the comparison fails. The Number object has a method, `toString`, which it inherits from Object but overrides. The `toString` method converts the number through which it is called to a string. Example:

```
var price = 427,  
str_price;  
...  
str_price = price.toString();
```

THE STRING CATENATION OPERATOR

JavaScript strings are not stored or treated as arrays of characters; rather, they are unit scalar values. String concatenation is specified with the operator denoted by a plus sign (+). For example, if the value of first is “Divya”, the value of the following expression is “Divya Gowda”:

```
first + " Gowda"
```

IMPLICIT TYPE CONVERSIONS

The JavaScript interpreter performs several different implicit type conversions. Such conversions are called *coercions*. If either operand of a + operator is a string, the operator is interpreted as a string catenation operator. If the other operand is not a string, it is coerced to a string.

Example1: `"August" + 1977 → "August 1997"`

Example2: `7 * "3" → 21` & will not be evaluated as string

EXPLICIT TYPE CONVERSIONS

Strings that contain numbers can be converted to numbers with the `String` constructor, as in the following code:

```
var str_value = String(value);
```

This conversion could also be done with the `toString` method, which has the advantage that it can be given a parameter to specify the base of the resulting number.

```
var num = 6;  
var str_value = num.toString(); //the result is "6"  
var str_value_binary = num.toString(2); //the result is "110"
```

A number also can be converted to a string by catenating it with the empty string. Also,

```
var number = Number(aString);
```

The number in the string cannot be followed by any character except a space. JavaScript has two predefined string functions that do not have this problem.

- The `parseInt` function searches its string parameter for an integer literal. If one is found at the beginning of the string, it is converted to a number and returned. If the string does not begin with a valid integer literal, `NaN` is returned.
- The `parseFloat` function is similar to `parseInt`, but it searches for a floating-point literal, which could have a decimal point, an exponent, or both. In both `parseInt` and `parseFloat`, the numeric literal could be followed by any nondigit character without causing any problem

String PROPERTIES AND METHODS

The `String` object includes one property, `length`, and a large collection of methods. The number of characters in a string is stored in the `length` property as follows:

```
var str = "George";
var len = str.length;           //now, len=6
```

Method	Parameters	Result
<code>charAt</code>	A number	Returns the character in the <code>String</code> object that is at the specified position
<code>indexOf</code>	One-character string	Returns the position in the <code>String</code> object of the parameter
<code>substring</code>	Two numbers	Returns the substring of the <code>String</code> object from the first parameter position to the second
<code>toLowerCase</code>	None	Converts any uppercase letters in the string to lowercase
<code>toUpperCase</code>	None	Converts any lowercase letters in the string to uppercase

Table 4.4 `String` methods

Consider, `var str = "George";`

Now, `str.charAt(2) is "o"`
`str.indexOf("r") is 3`
`str.substring(2, 4) is "org"`
`str.toLowerCase() is "george"`

THE `typeof` OPERATOR

- The `typeof` operator returns the type of its single operand.
- `typeof` produces “number”, “string”, or “boolean” if the operand is of primitive type Number, String, or Boolean, respectively.
- If the operand is an object or `null`, `typeof` produces “object”.
- If the operand is a variable that has not been assigned a value, `typeof` produces “undefined”, reflecting the fact that variables themselves are not typed.
- Notice that the `typeof` operator always returns a string.
- The operand for `typeof` can be placed in parentheses, making it appear to be a function.
- Therefore, `typeof x` and `typeof(x)` are equivalent.

ASSIGNMENT STATEMENTS

There is a simple assignment operator, denoted by `=`, and a host of compound assignment operators, such as `+=` and `/=`. For example, the statement `a += 7;` means the same as `a = a + 7;`

THE `Date` OBJECT

A `Date` object is created with the `new` operator and the `Date` constructor, which has several forms.

```
var today = new Date();
```

The date and time properties of a `Date` object are in two forms: local and Coordinated Universal Time (UTC, which was formerly named Greenwich Mean Time).

Table 4.5 shows the methods, along with the descriptions, that retrieve information from a `Date` object.

Method	Returns
<code>toLocaleString</code>	A string of the <code>Date</code> information
<code>getDate</code>	The day of the month
<code>getMonth</code>	The month of the year, as a number in the range from 0 to 11
<code>getDay</code>	The day of the week, as a number in the range from 0 to 6
<code>getFullYear</code>	The year
<code>getTime</code>	The number of milliseconds since January 1, 1970
<code>getHours</code>	The number of the hour, as a number in the range from 0 to 23
<code>getMinutes</code>	The number of the minute, as a number in the range from 0 to 59
<code>getSeconds</code>	The number of the second, as a number in the range from 0 to 59
<code>getMilliseconds</code>	The number of the millisecond, as a number in the range from 0 to 999

SCREEN OUTPUT AND KEYBOARD INPUT

- JavaScript models the XHTML document with the `Document` object.
- The window in which the browser displays an XHTML document is modelled with the `Window` object.
- The `Window` object includes two properties, `document` and `window`.
- The `document` property refers to the `Document` object.
- The `window` property is self-referential; it refers to the `Window` object.
- `write` is used to create XHTML code, the only useful punctuation in its parameter is in the form of XHTML tags. Therefore, the parameter of `write` often includes `
`.
- The `writeln` method implicitly adds "`\n`" to its parameter, but since browsers ignore line breaks when displaying XHTML, it has no effect on the output.
- The parameter of `write` can include any XHTML tags and content.
- The `write` method actually can take any number of parameters.
- Multiple parameters are concatenated and placed in the output.
- Example: `document.write("The result is: ", result, "
");`



- There are 3 types of pop-up boxes:
 - ▶ Alert
 - ▶ Confirm
 - ▶ Prompt
- The `alert` method opens a dialog window and displays its parameter in that window. It also displays an OK button.
- The string parameter of `alert` is not XHTML code; it is plain text. Therefore, the string parameter of `alert` may include `\n` but never should include `
`.

```
alert("The sum is:" + sum + "\n");
```



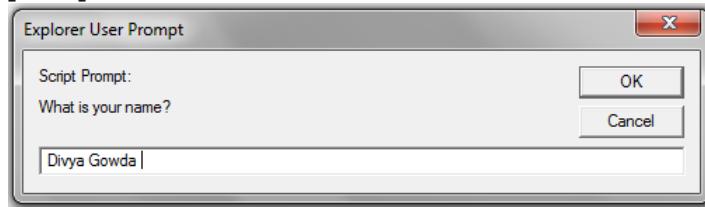
- The `confirm` method opens a dialog window in which the method displays its string parameter, along with two buttons: OK and Cancel.
- `confirm` returns a Boolean value that indicates the user's button input: `true` for OK and `false` for Cancel. This method is often used to offer the user the choice of continuing some process.

```
var question = confirm("Do you want to continue this download?");
```

- After the user presses one of the buttons in the `confirm` dialog window, the script can test the variable, `question`, and react accordingly.



- The `prompt` method creates a dialog window that contains a text box used to collect a string of input from the user, which `prompt` returns as its value.



Create an XHTML and JavaScript to compute the real roots of a given quadratic equation

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- roots.html
      A document for roots.js
      -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> roots.html </title>
  </head>
  <body>
    <script type = "text/javascript" src = "roots.js" >
    </script>
  </body>
</html>
```

```
// roots.js
// Compute the real roots of a given quadratic
// equation. If the roots are imaginary, this script
// displays NaN, because that is what results from
// taking the square root of a negative number

// Get the coefficients of the equation from the user
var a = prompt("What is the value of 'a'? \n", "");
var b = prompt("What is the value of 'b'? \n", "");
var c = prompt("What is the value of 'c'? \n", "");

// Compute the square root and denominator of the result
var root_part = Math.sqrt(b * b - 4.0 * a * c);
var denom = 2.0 * a;

// Compute and display the two roots
var root1 = (-b + root_part) / denom;
var root2 = (-b - root_part) / denom;
document.write("The first root is: ", root1, "<br />");
document.write("The second root is: ", root2, "<br />");
```

CONTROL STATEMENTS

A compound statement in JavaScript is a sequence of statements delimited by braces.

A control construct is a control statement together with the statement or compound statement whose execution it controls.

CONTROL EXPRESSIONS

The result of evaluating a control expression is one of the Boolean values `true` and `false`. If the value of a control expression is a string, it is interpreted as `true` unless it is either the empty string ("") or a zero string ("0"). If the value is a number, it is `true` unless it is zero (0). A relational expression has two operands and one relational operator.

Table 4.6 lists the relational operators.

Operation	Operator
Is equal to	<code>==</code>
Is not equal to	<code>!=</code>
Is less than	<code><</code>
Is greater than	<code>></code>
Is less than or equal to	<code><=</code>
Is greater than or equal to	<code>>=</code>
Is strictly equal to	<code>===</code>
Is strictly not equal to	<code>!==</code>

JavaScript has operators for the AND, OR, and NOT Boolean operations. These are `&&` (AND), `||` (OR), and `!` (NOT). Both `&&` and `||` are short-circuit operators.

Operators	Associativity
<code>++, --, unary -</code>	Right
<code>*, /, %</code>	Left
<code>+, -</code>	Left
<code>>, <, >= ,<=</code>	Left
<code>==, !=</code>	Left
<code>====, !===</code>	Left
<code>&&</code>	Left
<code> </code>	Left
<code>=, +=, -=, *=, /=, &&=, =, %=</code>	Right

Table 4.7 Operator precedence and associativity

Highest-precedence operators are listed first.

SELECTION STATEMENTS

The selection statements (`if-then` and `if-then-else`) of JavaScript are similar to those of the common programming languages. Either single statements or compound statements can be selected—for example,

```

if (a > b)
    document.write("a is greater than b <br />");
else {
    a = b;
    document.write("a was not greater than b <br />",
                   "Now they are equal <br />");
```

THE `switch` STATEMENT

JavaScript has a `switch` statement that is similar to that of Java. In any `case` segment, the statement(s) can be either a sequence of statements or a compound statement. The `break` statement transfers control out of the compound statement in which it appears.

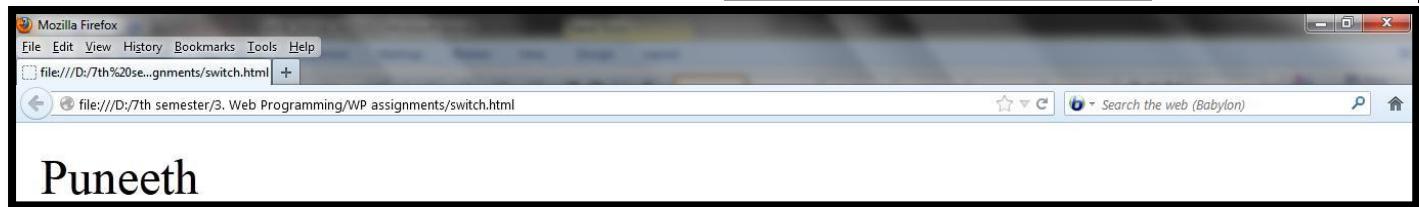
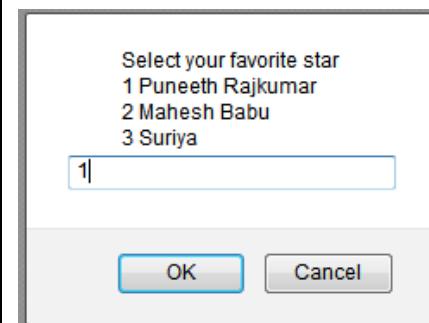
The control expression of a `switch` statement could evaluate to a number, a string, or a Boolean value. Case labels also can be numbers, strings, or Booleans, and different case values can be of different types.

```
switch (expression) {  
    case value_1:  
        // statement(s)  
    case value_2:  
        // statement(s)  
    ...  
    [default:  
        // statement(s)]  
}
```

Example:

```
//switch.js  
var choice = prompt("Select your favorite star \n" + "1  
                    Puneeth Rajkumar \n" +  
                    "2 Mahesh Babu \n" +  
                    "3 Suriya \n");  
  
switch(choice)  
{  
    case "1": document.write("Puneeth");  
               break;  
    case "2": document.write("Mahesh Babu");  
               break;  
    case "3": document.write("Suriya");  
               break;  
    default: document.write("invalid choice");  
}
```

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



Assignment: switch statement for table border size selection – it is similar but refer text book

LOOP STATEMENTS

The general form of the `while` statement is as follows:

```
while (control expression)  
    statement or compound statement
```

The general form of the `for` statement is as follows:

```
for (initial expression; control expression; increment expression)  
    statement or compound statement
```

The following example illustrates the `Date` object and a simple `for` loop:

```
// date.js  
// Illustrates the use of the Date object by  
// displaying the parts of a current date and  
// using two Date objects to time a calculation  
  
// Get the current date  
var today = new Date();
```

```
// Fetch the various parts of the date
var dateString = today.toLocaleString();
var day = today.getDay();
var month = today.getMonth();
var year = today.getFullYear();
var timeMilliseconds = today.getTime();
var hour = today.getHours();
var minute = today.getMinutes();
var second = today.getSeconds();
var millisecond = today.getMilliseconds();
```

```
// Display the parts
document.write(
    "Date: " + dateString + "<br />",
    "Day: " + day + "<br />",
    "Month: " + month + "<br />",
    "Year: " + year + "<br />",
    "Time in milliseconds: " + timeMilliseconds + "<br />",
    "Hour: " + hour + "<br />",
    "Minute: " + minute + "<br />",
    "Second: " + second + "<br />",
    "Millisecond: " + millisecond + "<br />");
```

```
// Time a loop
var dum1 = 1.00149265, product = 1;
var start = new Date();

for (var count = 0; count < 10000; count++)
    product = product + 1.000002 * dum1 / 1.00001;

var end = new Date();
var diff = end.getTime() - start.getTime();
document.write("<br />The loop took " + diff +
    " milliseconds <br />");
```

OUTPUT:

```
Date: Tuesday, August 11, 2009 1:54:01 PM
Day: 2
Month: 7
Year: 2009
Time in milliseconds: 1250020441561
Hour: 13
Minute: 54
Second: 1
Millisecond: 561
```

```
The loop took 9 milliseconds
```

JavaScript has a do-while statement, whose form is as follows:

```
do statement or compound statement  
while (control expression)
```

JavaScript includes one more loop statement, the for-in statement, which is most often used with objects.

```
for (identifier in object)  
statement or compound statement
```

OBJECT CREATION AND MODIFICATION

- Objects are often created with a new expression, which must include a call to a constructor method. The constructor that is called in the new expression creates the properties that characterize the new object.
- In JavaScript, however, the new operator creates a blank object—that is, one with no properties.
- The following statement creates an object that has no properties:
`var my_object = new Object();`
- In this case, the constructor called is that of Object, which endows the new object with no properties, although it does have access to some inherited methods.
- The variable my_object references the new object. Calls to constructors must include parentheses, even if there are no parameters.
- The properties of an object can be accessed with dot notation, in which the first word is the object name and the second is the property name. Because properties are not variables, they are never declared.
- The number of members of a class in a typical object-oriented language is fixed at compile time. The number of properties in a JavaScript object is dynamic.
- At any time during interpretation, properties can be added to or deleted from an object. A property for an object is created by assigning a value to that property's name. Consider the following example:
`var my_car = {make: "Ford", model: "Contour SVT"};`
- Properties can be accessed in two ways.
`var prop1 = my_car.make;
var prop2 = my_car["make"];`
the variables prop1 and prop2 both have the value "Ford".
- A property can be deleted with delete, as in the following example:
`delete my_car.model;`
- JavaScript has a loop statement, for-in, that is perfect for listing the properties of an object.
`for (var prop in my_car)
 document.write("Name: ", prop, "; Value: ",
 my_car[prop], "
");`

ARRAYS

Array OBJECT CREATION

The usual way to create any object is with the new operator and a call to a constructor. In the case of arrays, the constructor is named Array:

```
var my_list = new Array(1, 2, "three", "four");  
var your_list = new Array(100);
```

The second way to create an Array object is with a literal array value, which is a list of values enclosed in brackets:

```
var my_list_2 = [1, 2, "three", "four"];
```

CHARACTERISTICS OF Array OBJECTS

The lowest index of every JavaScript array is zero. Access to the elements of an array is specified with numeric subscript expressions placed in brackets. The length of an array is the highest subscript to which a value has been assigned, plus 1.

For example, if `my_list` is an array with four elements and the following statement is executed, the new length of `my_list` will be 48.

```
my_list[47] = 2222;
```

The length of an array is both read and write accessible through the `length` property, which is created for every array object by the `Array` constructor. For example,

```
my_list.length = 1002;
```

An array is lengthened by setting its `length` property to a larger value, shortened by setting its `length` property to a smaller value.

The next example, `insert_names.js`, illustrates JavaScript arrays. This script has an array of names, which are in alphabetical order. It uses `prompt` to get new names, one at a time, and inserts them into the existing array. Notice that each new name causes the array to grow by one element.

```
// The original list of names
var name_list = new Array("Al", "Betty", "Kasper",
                         "Michael", "Roberto", "Zimbo");
var new_name, index, last;

// Loop to get a new name and insert it
while (new_name =
       prompt("Please type a new name", "")) {
    last = name_list.length - 1;

    // Loop to find the place for the new name
    while (last >= 0 && name_list[last] > new_name) {
        name_list[last + 1] = name_list[last];
        last--;
    }

    // Insert the new name into its spot in the array
    name_list[last + 1] = new_name;

    // Display the new array
    document.write("<p><b>The new name list is:</b> ",
                  "<br />");
    for (index = 0; index < name_list.length; index++)
        document.write(name_list[index], "<br />");
    document.write("</p>");
} /** end of the outer while loop
```

Array METHODS

Array objects have a collection of useful methods, most of which are described in this section.

- The `join` method converts all of the elements of an array to strings and concatenates them into a single string. If no parameter is provided to join, the values in the new string are separated by commas. If a string parameter is provided, it is used as the element separator. Consider the following example:

```
var names = new Array["Mary", "Murray", "Murphy", "Max"];
...
var name_string = names.join(" : ");
```

The value of `name_string` is now "Mary : Murray : Murphy : Max".

- The **reverse** method reverses the order of the elements of the Array object through which it is called.
- The **sort** method coerces the elements of the array to become strings if they are not already strings and sorts them alphabetically
- The **concat** method concatenates its actual parameters to the end of the Array object on which it is called.

```
var names = new Array["Mary", "Murray", "Murphy", "Max"];
...
var new_names = names.concat("Moo", "Meow");
```

The `new_names` array now has length 6, with the elements of `names`, along with "Moo" and "Meow", as its fifth and sixth elements.

- The **slice** method does for arrays what the **substring** method does for strings, returning the part of the Array object specified by its parameters, which are used as subscripts. The array returned has the elements of the Array object through which it is called, from the first parameter up to, but not including, the second parameter.

```
var list = [2, 4, 6, 8, 10];
...
var list2 = list.slice(1, 3);
```

The value of `list2` is now [4, 6]. If `slice` is given just one parameter, the array that is returned has all of the elements of the object, starting with the specified index.

- When the **toString** method is called through an `Array` object, each of the elements of the object is converted (if necessary) to a string. These strings are concatenated, separated by commas. So, for `Array` objects, the `toString` method behaves much like `join`.
- The **push**, **pop**, **unshift**, and **shift** methods of `Array` allow the easy implementation of stacks and queues in arrays. The `pop` and `push` methods respectively remove and add an element to the high end of an array, as in the following code:

```
var list = ["Dasher", "Dancer", "Donner", "Blitzen"];
var deer = list.pop(); // deer is "Blitzen"
list.push("Blitzen");
// This puts "Blitzen" back on list
```

The `shift` and `unshift` methods respectively remove and add an element to the beginning of an array.

```
var deer = list.shift(); // deer is now "Dasher"
list.unshift("Dasher"); // This puts "Dasher" back on list
```

```
// nested_arrays.js
// An example illustrating an array of arrays

// Create an array object with three arrays as its elements
var nested_array = [[2, 4, 6], [1, 3, 5], [10, 20, 30]];
]

// Display the elements of nested_list
for (var row = 0; row <= 2; row++) {
  document.write("Row ", row, ": ");
  for (var col = 0; col <= 2; col++) {
    document.write(nested_array[row][col], " ");
  }
  document.write("<br />");
}
```

OUTPUT:

```
Row 0: 2 4 6
Row 1: 1 3 5
Row 2: 10 20 30
```

FUNCTIONS

FUNDAMENTALS

- A **function definition** consists of the function's header and a compound statement that describes the actions of the function. This compound statement is called the **body** of the function.
- A function **header** consists of the reserved word `function`, the function's name, and a parenthesized list of parameters if there are any.
- A `return` statement returns control from the function in which it appears to the function's caller. A function body may include one or more `return` statements. If there are no `return` statements in a function or if the specific `return` that is executed does not include an expression, the value returned is `undefined`.
- JavaScript functions are objects, so variables that reference them can be treated as are other object references—they can be passed as parameters, be assigned to other variables, and be the elements of an array. The following example is illustrative:

```
function fun() { document.write(  
    "This surely is fun! <br/>");}  
ref_fun = fun; // Now, ref_fun refers to the fun object  
fun(); // A call to fun  
ref_fun(); // Also a call to fun
```

- Because JavaScript functions are objects, their references can be properties in other objects, in which case they act as methods.

LOCAL VARIABLES

- The **scope** of a variable is the range of statements over which it is visible.
- When JavaScript is embedded in an XHTML document, the scope of a variable is the range of lines of the document over which the variable is visible.
- Variables that are implicitly declared have **global scope**—that is, they are visible in the entire XHTML document.
- It is usually best for variables that are used only within a function to have **local scope**, meaning that they are visible and can be used only within the body of the function. Any variable explicitly declared with `var` in the body of a function has local scope.
- If a variable that is defined both as a local variable and as a global variable appears in a function, the local variable has precedence, effectively hiding the global variable with the same name. This is the advantage of local variables.

PARAMETERS

- The parameter values that appear in a call to a function are called **actual parameters**.
- The parameter names that appear in the header of a function definition, which correspond to the actual parameters in calls to the function, are called **formal parameters**.
- JavaScript uses the pass-by-value parameter-passing method.
- When a function is called, the values of the actual parameters specified in the call are, in effect, copied into their corresponding formal parameters, which behave exactly like local variables.
- Because of JavaScript's dynamic typing, there is no type checking of parameters. The called function itself can check the types of parameters with the `typeof` operator.
- The number of parameters in a function call is not checked against the number of formal parameters in the called function.
- In the function, excess actual parameters that are passed are ignored; excess formal parameters are set to `undefined`.
- All parameters are communicated through a property array, `arguments`, that, like other array objects, has a property named `length`.
- By accessing `arguments.length`, a function can determine the number of actual parameters that were passed.

- The following example illustrates a variable number of function parameters:

```
// params.js
// The params function and a test driver for it.
// This example illustrates a variable number of
// function parameters

// Function params
// Parameters: A variable number of parameters
// Returns: nothing
// Displays its parameters
function params(a, b) {
    document.write("Function params was passed ",
        arguments.length, " parameter(s) <br />");
    document.write("Parameter values are: <br />");

    for (var arg = 0; arg < arguments.length; arg++)
        document.write(arguments[arg], "<br />");

    document.write("<br />");
}

// A test driver for function params
params("Mozart");
params("Mozart", "Beethoven");
params("Mozart", "Beethoven", "Tchaikowsky");
```

Output:

```
Function params was passed 1 parameter(s)
Parameter values are:
Mozart

Function params was passed 2 parameter(s)
Parameter values are:
Mozart
Beethoven

Function params was passed 3 parameter(s)
Parameter values are:
Mozart
Beethoven
Tchaikowsky
```

There is no elegant way in JavaScript to pass a primitive value by reference. One inelegant way is to put the value in an array and pass the array, as in the following script:

```
// Function by10
// Parameter: a number, passed as the first element
//             of an array
// Returns: nothing
// Effect: multiplies the parameter by 10
function by10(a) {
    a[0] *= 10;
}
...
var x;
var listx = new Array(1);
...
listx[0] = x;
by10(listx);
x = listx[0];
```

THE sort METHOD, REVISITED

- If you need to sort something other than strings, or if you want an array to be sorted in some order other than alphabetically as strings, the comparison operation must be supplied to the `sort` method by the caller. Such a comparison operation is passed as a parameter to `sort`.
- The comparison function must return a negative number if the two elements being compared are in the desired order, zero if they are equal, and a number greater than zero if they must be interchanged.
- For example, if you want to use the `sort` method to sort the array of numbers `num_list` into descending order, you could do so with the following code:

```
// Function num_order
// Parameter: Two numbers
// Returns: If the first parameter belongs before the
//           second in descending order, a negative number
//           If the two parameters are equal, 0
//           If the two parameters must be
//           interchanged, a positive number
function num_order(a, b) {return b - a;}
// Sort the array of numbers, list, into
// ascending order
num_list.sort(num_order);
```

AN EXAMPLE

```
// medians.js
// A function and a function tester
// Illustrates array operations

// Function median
// Parameter: An array of numbers
// Result: The median of the array
// Return value: none
function median(list) {
    list.sort(function (a, b) {return a - b;});
    var list_len = list.length;

    // Use the modulus operator to determine whether
    // the array's length is odd or even
    // Use Math.floor to truncate numbers
    // Use Math.round to round numbers
    if ((list_len % 2) == 1)
        return list[Math.floor(list_len / 2)];
    else
        return Math.round((list[list_len / 2 - 1] +
                           list[list_len / 2]) / 2);
} // end of function median
var my_list_1 = [8, 3, 9, 1, 4, 7];
var my_list_2 = [10, -2, 0, 5, 3, 1, 7];
var med = median(my_list_1);
document.write("Median of [", my_list_1, "] is: ",
               med, "<br />");
med = median(my_list_2);
document.write("Median of [", my_list_2, "] is: ",
               med, "<br />");
```

Output:

```
Median of[1,3,4,7,8,9] is: 6  
Median of[-2,0,1,3,5,7,10] is: 3
```

CONSTRUCTORS

- JavaScript constructors are special methods that create and initialize the properties of newly created objects.
- Every new expression must include a call to a constructor whose name is the same as that of the object being created.
- Constructors are actually called by the new operator, which immediately precedes them in the new expression.
- Obviously, a constructor must be able to reference the object on which it is to operate. JavaScript has a predefined reference variable for this purpose, named this.
- When the constructor is called, this is a reference to the newly created object. The this variable is used to construct and initialize the properties of the object.
- For example, the constructor

```
function car(new_make, new_model, new_year) {  
    this.make = new_make;  
    this.model = new_model;  
    this.year = new_year;  
}
```

could be used as in the following statement:

```
my_car = new car("Ford", "Contour SVT", "2000");
```

- For example, suppose you wanted a method for car objects that listed the property values. A function that could serve as such a method could be written as follows:

```
function display_car() {  
    document.write("Car make: ", this.make, "<br/>");  
    document.write("Car model: ", this.model, "<br/>");  
    document.write("Car year: ", this.year, "<br/>");  
}
```

- The following line must then be added to the car constructor:

```
this.display = display_car;
```

- Now the call my_car.display() will produce the following output

```
Car make: Ford  
Car model: Contour SVT  
Car year: 2000
```

PATTERN MATCHING BY USING REGULAR EXPRESSIONS

- JavaScript has powerful pattern-matching capabilities based on regular expressions.
- There are two approaches to pattern matching in JavaScript: one that is based on the RegExp object and one that is based on methods of the String object.
- The simplest pattern-matching method is search, which takes a pattern as a parameter.
- The search method returns the position in the String object (through which it is called) at which the pattern matched.
- If there is no match, search returns -1.
- Most characters are normal, which means that, in a pattern, they match themselves.
- The position of the first character in the string is 0.
- As an example, the following statements

```

var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position >= 0)
    document.write("'bits' appears in position", position,
                  "<br />");
else
    document.write("'bits' does not appear in str <br />");
```

produce the following output:

```
'bits' appears in position 3
```

CHARACTER AND CHARACTER-CLASS PATTERNS

- Metacharacters are characters that have special meanings in some contexts in patterns.
- The following are the pattern metacharacters:
 \backslash | () [] { } ^ \$ * + ? .
- Metacharacters can themselves be matched by being immediately preceded by a backslash.
- A period matches any character except newline.
- Example: `/snow./` matches “snowy”, “snowe”, and “snowd”
- Example: `/3\4/` matches 3.4. but `/3.4/` would match 3 . 4 and 374, among others.
- Example: `[abc]` matches ‘a’ , ‘b’ & ‘c’
- Example: `[a-h]` matches any lowercase letter from ‘a’ to ‘h’
- Example: `[^aeiou]` matches any lowercase letter except ‘a’, ‘e’, ‘i’, ‘o’ & ‘u’

Name	Equivalent Pattern	Matches
<code>\d</code>	<code>[0-9]</code>	A digit
<code>\D</code>	<code>[^0-9]</code>	Not a digit
<code>\w</code>	<code>[A-Za-z_0-9]</code>	A word character (alphanumeric)
<code>\W</code>	<code>[^A-Za-z_0-9]</code>	Not a word character
<code>\s</code>	<code>[\r\t\n\f]</code>	A white-space character
<code>\S</code>	<code>[^ \r\t\n\f]</code>	Not a white-space character

ANCHORS

- A pattern is tied to a string position with an anchor. A pattern can be specified to match only at the beginning of the string by preceding it with a circumflex (^) anchor.
- For example, the following pattern matches “pearls are pretty” but does not match “My pearls are pretty”:
`/^pearl/`
- A pattern can be specified to match at the end of a string only by following the pattern with a dollar sign anchor. For example, the following pattern matches “I like gold” but does not match “golden”:
`/gold$/`
- Anchor characters are like boundary-named patterns: They do not match specific characters in the string; rather, they match positions before, between, or after characters.

PATTERN MODIFIERS

- The modifiers are specified as letters just after the right delimiter of the pattern.
- The `i` modifier makes the letters in the pattern match either uppercase or lowercase letters in the string.
- For example, the pattern `/Apple/i` matches ‘APPLE’, ‘apple’, ‘APPlE’, and any other combination of uppercase and lowercase spellings of the word “apple.”
- The `x` modifier allows white space to appear in the pattern.

```

/\d+      # The street number
\s        # The space before the street name
[A-Z][a-z]+ # The street name
/x

is equivalent to

/\d+\s[A-Z][a-z]+/

```

OTHER PATTERN-MATCHING METHODS OF String

- The `replace` method is used to replace substrings of the `String` object that match the given pattern.
 - The `replace` method takes two parameters: the pattern and the replacement string.
 - The `g` modifier can be attached to the pattern if the replacement is to be global in the string, in which case the replacement is done for every match in the string.
 - The matched substrings of the string are made available through the predefined variables `$1`, `$2`, and so on. For example, consider the following statements:
- ```

var str = "Fred, Freddie, and Frederica were siblings";
str.replace(/Fre/g, "Boy");

```
- In this example, `str` is set to “Boyd, Boyd die, and Boy derica were siblings”, and `$1`, `$2`, and `$3` are all set to “Fre”.
  - The `match` method is the most general of the `String` pattern-matching methods.
  - The `match` method takes a single parameter: a pattern. It returns an array of the results of the pattern-matching operation.
  - If the pattern has the `g` modifier, the returned array has all of the substrings of the string that matched.
  - If the pattern does not include the `g` modifier, the returned array has the match as its first element, and the remainder of the array has the matches of parenthesized parts of the pattern if there are any:

```

var str =
 "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);

```

In this example, `matches` is set to `[4, 3]`.

- The `split` method of `String` splits its object string into substrings on the basis of a given string or pattern. The substrings are returned in an array. For example, consider the following code:

```

var str = "grapes:apples:oranges";
var fruit = str.split(":");

```

In this example, `fruit` is set to `[grapes, apples, oranges]`.

## ANOTHER EXAMPLE

```

// forms_check.js
// A function tst_phone_num is defined and tested.
// This function checks the validity of phone
// number input from a form

// Function tst_phone_num
// Parameter: A string
// Result: Returns true if the parameter has the form of a valid
// seven-digit phone number (3 digits, a dash, 4 digits)
function tst_phone_num(num) {

// Use a simple pattern to check the number of digits and the dash
var ok = num.search(/^\d{3}-\d{4}$/);

if (ok == 0)
 return true;
else
 return false;

} // end of function tst_phone_num

```

```

// A script to test tst_phone_num
var tst = tst_phone_num("444-5432");
if (tst)
 document.write("444-5432 is a valid phone number
");
else
 document.write("Error in tst_phone_num
");
tst = tst_phone_num("444-r432");
if (tst)
 document.write("Program error
");
else
 document.write(
 "444-r432 is not a valid phone number
");

tst = tst_phone_num("44-1234");
if (tst)
 document.write("Program error
");
else
 document.write("44-1234 is not a valid phone number
");

```

#### OUTPUT:

444-5432 is a valid phone number  
 444-r432 is not a valid phone number  
 44-1234 is not a valid phone number

## ERRORS IN SCRIPTS

The JavaScript interpreter is capable of detecting various errors in scripts. Debugging a script is a bit different from debugging a program in a more typical programming language, mostly because errors that are detected by the JavaScript interpreter are found while the browser is attempting to display a document. In some cases, a script error causes the browser not to display the document and does not produce an error message. Without a diagnostic message, you must simply examine the code to find the problem.

```

// debugdemo.js
// An example to illustrate debugging help

var row;
row = 0;
while(row != 4 {
 document.write("row is ", row, "
");
 row++;
}

```



# UNIT 5

## DOCUMENT OBJECT MODEL

### THE JAVASCRIPT EXECUTION ENVIRONMENT

- A browser displays an XHTML document in a window on the screen of the client.
- The JavaScript `Window` object represents the window that displays the document.
- The properties of the `Window` object are visible to all JavaScript scripts that appear either implicitly or explicitly in the window's XHTML document, so they include all of the global variables.
- Every `Window` object has a property named `document`, which is a reference to the `Document` object that the window displays.
- Every `Document` object has a `forms` array, each element of which represents a form in the document.
- Each `forms` array element has an `elements` array as a property, which contains the objects that represent the XHTML form elements, such as buttons and menus.
- `Document` objects also have property arrays for anchors, links, images, and applets.

### THE DOCUMENT OBJECT MODEL

- The original motivation for the standard DOM was to provide a specification that would allow Java programs and JavaScript scripts that deal with XHTML documents to be portable among various browsers.
- The DOM is an application programming interface (API) that defines an interface between XHTML documents and application programs.
- It is an abstract model because it must apply to a variety of application programming languages.
- Each language that interfaces with the DOM must define a binding to that interface.
- The actual DOM specification consists of a collection of interfaces, including one for each document tree node type.
- They define the objects, methods, and properties that are associated with their respective node types.
- With the DOM, users can write code in programming languages to create documents, move around in their structures, and change, add, or delete elements and their content.
- Documents in the DOM have a treelike structure, but there can be more than one tree in a document.
- Because the DOM is an abstract interface, it does not dictate that documents be implemented as trees or collections of trees.

```
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head> <title> A simple table </title>
 </head>
 <body>
 <table border = "border">
 <tr>
 <th> </th>
 <th> Apple </th>
 <th> Orange </th>
 </tr>
 <tr>
 <th> Breakfast </th>
 <td> 0 </td>
 <td> 1 </td>
 </tr>
 </table>
 </body>
</html>
```

A screenshot of Microsoft Internet Explorer Developer Tools showing the DOM tree for a simple table.

**Left Panel: Document View**

- File Find Disable View Outline Images Cache Tools Validate
- Browser Mode: IE8 Document Mode: IE8 Standards
- HTML CSS Script Profiler Search HTML

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
<!-- table2.html A simple table to demonstrate DOM trees
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
 <title>A simple table</title>
 <body>
 <table border="border">
 <tbody>
 <tr>
 <th>
 <th>Text - Apple
 <th>Text - Orange
 <tr>
 <th>
 <td>Text - Breakfast
 <td>Text - 0
 <td>Text - 1

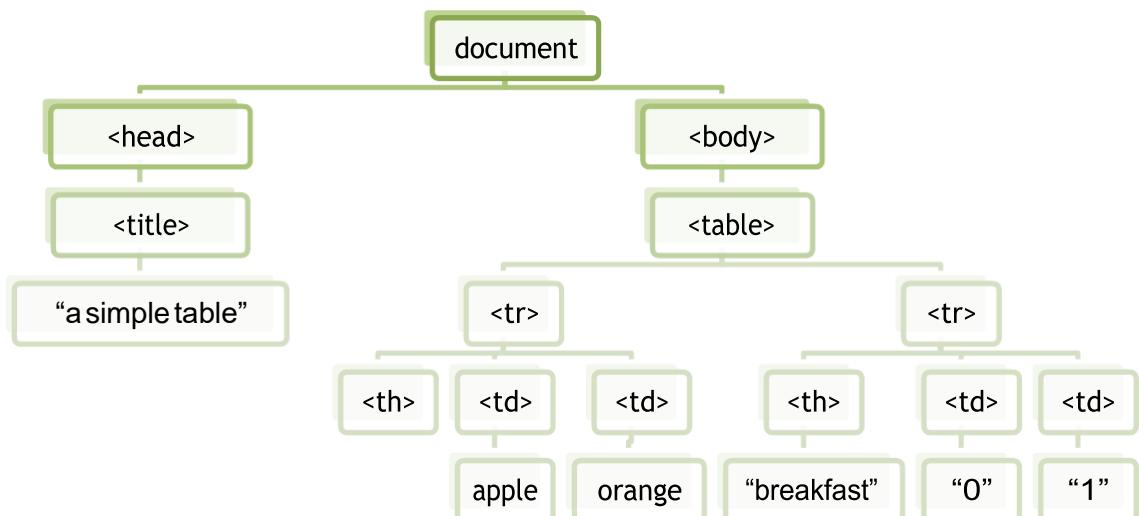
```

**Right Panels: DOM Nodes and Object**

Document - DOM Nodes

nodeName	id	class
#document		
HTML		
HEAD		
text		
#comment		
#text		
#text		
TITLE		
#text		
#text		
BODY		
#text		
TABLE		
#text		
TBODY		
TR		
#text		
TH		
#text		
#text		
TH		
#text		
#text		
TH		
#text		
#text		
TR		
#text		
TH		
#text		
#text		
TD		
#text		
TD		
#text		
TD		
#text		
#text		
#text		

Object - DOM Node



- A language that is designed to support the DOM must have a binding to the DOM constructs.
- In the JavaScript binding to the DOM, the elements of a document are objects, with both data and operations.
- The data are called properties, and the operations are, naturally, called methods.
- Example: `<input type = "text" name = "address">`

## ELEMENT ACCESS IN JAVASCRIPT

The elements of an XHTML document have corresponding objects that are visible to an embedded JavaScript script. There are several ways the object associated with an XHTML form element can be addressed in JavaScript. The original (DOM 0) way is to use the forms and elements arrays of the Document object, which is referenced through the document property of the Window object.

Example:

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> Access to form elements </title>
</head>
<body>
 <form action = "">
 <input type = "button" name = "turnItOn" />
 </form>
</body>
</html>
```

The DOM address of the button in this example, using the forms and elements arrays, is as follows:

```
var dom = document.forms[0].elements[0];
```

The problem with this approach to element addressing is that the DOM address is defined by address elements that could change—namely, the forms and elements arrays.

Another Approach:

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> <title> Access to form elements </title>
</head>
<body>
 <form name = "myForm" action = "">
 <input type = "button" name = "turnItOn" />
 </form>
</body>
</html>
```

Using the name attributes, the button's DOM address is as follows:

```
var dom = document.myForm.turnItOn;
```

One drawback of this approach is that the XHTML 1.1 standard does not allow the name attribute in the form element, even though the attribute is now valid for form elements. This is a validation problem, but it causes no difficulty for browsers. Hence, alternatively we use,

```
var dom = document.getElementById("turnItOn");
```

Buttons in a group of checkboxes often share the same name. The buttons in a radio button group always have the same name. In these cases, the names of the individual buttons obviously cannot be used in their DOM addresses. To access the arrays, the DOM address of the form object must first be obtained, as shown:

```
<form id = "vehicleGroup">
 <input type = "checkbox" name = "vehicles"
 value = "car" /> Car
 <input type = "checkbox" name = "vehicles"
 value = "truck" /> Truck
 <input type = "checkbox" name = "vehicles"
 value = "bike" /> Bike
</form>
```

The checked property of a checkbox object is set to true if the button is checked. For the preceding sample checkbox group, the following code would count the number of checkboxes that were checked:

```
var numChecked = 0;
var dom = document.getElementById("vehicleGroup");
for (index = 0; index < dom.vehicles.length; index++)
 if (dom.vehicles[index].checked)
 numChecked++;
```

Radio buttons can be addressed and handled exactly as are the checkboxes in the foregoing code.

## EVENTS AND EVENT HANDLING

### BASIC CONCEPTS OF EVENT HANDLING

- One important use of JavaScript for Web programming is to detect certain activities of the browser and the browser user and provide computation when those activities occur. These computations are specified with a special form of programming called **event-driven programming**.
- In conventional (non-event-driven) programming, the code itself specifies the order in which it is executed, although the order is usually affected by the program's input data.
- In event-driven programming, parts of the program are executed at completely unpredictable times, often triggered by user interactions with the program that is executing.
- An **event** is a notification that something specific has occurred, either with the browser, such as the completion of the loading of a document, or because of a browser user action, such as a mouse click on a form button.
- An **event handler** is a script that is implicitly executed in response to the appearance of an event. Event handlers enable a Web document to be responsive to browser and user activities.
- One of the most common uses of event handlers is to check for simple errors and omissions in user input to the elements of a form, either when they are changed or when the form is submitted.
- This kind of checking saves the time of sending incorrect form data to the server.
- Because events are JavaScript objects, their names are case sensitive. The names of all event objects have only lowercase letters.
- Events are created by activities associated with specific XHTML elements.
- The process of connecting an event handler to an event is called **registration**.
- There are two distinct approaches to event handler registration, one that assigns tag attributes and one that assigns handler addresses to object properties.

### EVENTS, ATTRIBUTES, AND TAGS

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

In many cases, the same attribute can appear in several different tags. The circumstances under which an event is created are related to a tag and an attribute, and they can be different for the same attribute when it appears in different tags.

Attribute	Tag	Description
onblur	<a>	The link loses the input focus
	<button>	The button loses the input focus
	<input>	The input element loses the input focus
	<textarea>	The text area loses the input focus
	<select>	The selection element loses the input focus
onchange	<input>	The input element is changed and loses the input focus
	<textarea>	The text area is changed and loses the input focus
	<select>	The selection element is changed and loses the input focus
onclick	<a>	The user clicks on the link
	<input>	The input element is clicked
ondblclick	Most elements	The user double-clicks the left mouse button
onfocus	<a>	The link acquires the input focus
	<input>	The input element receives the input focus
	<textarea>	A text area receives the input focus
	<select>	A selection element receives the input focus
onkeydown	<body>, form elements	A key is pressed down
onkeypress	<body>, form elements	A key is pressed down and released
onkeyup	<body>, form elements	A key is released
onload	<body>	The document is finished loading
onmousedown	Most elements	The user clicks the left mouse button
onmousemove	Most elements	The user moves the mouse cursor within the element
onmouseout	Most elements	The mouse cursor is moved away from being over the element
onmouseover	Most elements	The mouse cursor is moved over the element
onmouseup	Most elements	The left mouse button is unclicked
onreset	<form>	The reset button is clicked
onselect	<input>	Any text in the content of the element is selected
	<textarea>	Any text in the content of the element is selected
onsubmit	<form>	The Submit button is pressed
onunload	<body>	The user exits the document

As mentioned previously, there are two ways to register an event handler in the DOM 0 event model. One of these is by assigning the event handler script to an event tag attribute, as in the following example:

```
<input type = "button" id = "myButton"
 onclick = "alert('You clicked my button!');" />
```

In many cases, the handler consists of more than a single statement. In these cases, often a function is used and the literal string value of the attribute is the call to the function. Consider the example of a button element:

```
<input type = "button" id = "myButton"
 onclick = "myButtonHandler();" />
```

An event handler function could also be registered by assigning its name to the associated event property on the button object, as in the following example:

```
document.getElementById("myButton").onclick =
 myButtonHandler;
```

## HANDLING EVENTS FROM BODY ELEMENTS

The events most often created by body elements are `load` and `unload`. As our first example of event handling, we consider the simple case of producing an alert message when the body of the document has been loaded. In this case, we use the `onload` attribute of `<body>` to specify the event handler:

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- load.html
 A document for load.js
 -->
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head>
 <title> load.html </title>
 <script type = "text/javascript" src = "load.js" >
 </script>
 </head>
 <body onload="load_greeting();">
 <p />
 </body>
</html>
```

```
// load.js
// An example to illustrate the load event

// The onload event handler
function load_greeting () {
 alert("You are visiting the home page of \n" +
 "Pete's Pickled Peppers \n" + "WELCOME!!!");
}
```

Output:



The `unload` event is probably more useful than the `load` event. It is used to do some cleanup before a document is unloaded, as when the browser user goes on to some new document. For example, if the document opened a second browser window, that window could be closed by an `unload` event handler.

## HANDLING EVENTS FROM BUTTON ELEMENTS

Buttons in a Web document provide an effective way to collect simple input from the browser user. Example:

```
//radio_click.html
<html>
<head>
<title> radio_click.html</title>
<script type = "text/javascript" src = "radio_click.js">
</script>
</head>
<body>
 <h4> Choose your favourite Director in Kannada Film Industry</h4>
 <form id = "myForm" action = ">
```

```

<p>
 <label><input type = "radio" name = "dButton" value = "1" onclick = "dChoice(1)"/> Yogaraj Bhat</label>

 <label><input type = "radio" name = "dButton" value = "2" onclick = "dChoice(2)"/> Suri</label>

 <label><input type = "radio" name = "dButton" value = "3" onclick = "dChoice(3)"/> Guru Prasad</label>

 <label><input type = "radio" name = "dButton" value = "4" onclick = "dChoice(4)"/> Prakash</label>
</p>

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

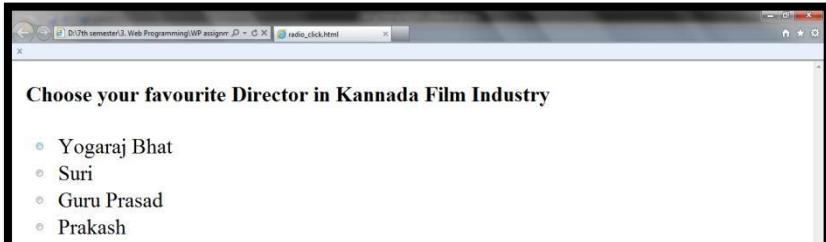
```

```

//radio_click.js
function dChoice(ch)
{
 switch(ch)
 {
 case 1: alert("Mungaaru Male");
 break;
 case 2: alert("Duniya");
 break;
 case 3: alert("Eddelu Manjunatha");
 break;
 case 4: alert("Milana");
 break;
 default: alert("Ooops..Invalid choice :O"); break;
 }
}

```

Output:



After clicking on an option, say “Guru Prasad”, we get



The next example, `radio_click2.html`, whose purpose is the same as that of `radio_click.html`, registers the event handler by assigning the name of the handler to the event properties of the radio button objects.

The following example uses three files—one for the XHTML, one for the script for the event handlers, and one for the script to register the handlers:

`//radio_click2.html`

```

<html>
<head>

```

```

<title> radio_click2.html</title>
<script type = "text/javascript" src = "radio_click2.js">
</script>
</head>
<body>
<h4> Choose your favourite Director in Kannada Film Industry</h4>
<form id = "myForm" action = " ">
<p>
<label><input type = "radio" name = "dButton" value = "1" id = "1"/> Yogaraj Bhat</label>

<label><input type = "radio" name = "dButton" value = "2" id = "2"/> Suri</label>

<label><input type = "radio" name = "dButton" value = "3" id = "3"/> Guru Prasad</label>

<label><input type = "radio" name = "dButton" value = "4" id = "4"/> Prakash</label>
</p>
</form>
<script type = "text/javascript" src = "radio_click2r.js">
</script>
</body>
</html>

```

### //radio\_click2.js

```

function dChoice(ch)
{
 var dom = document.getElementById("myForm");

 for(var index = 0; index < dom.dButton.length; index++)
 {
 if(dom.dButton[index].checked)
 {
 ch = dom.dButton[index].value;
 break;
 }
 }
 switch(ch)
 {
 case 1: alert("Mungaaru Male");
 break;
 case 2: alert("Duniya");
 break;
 case 3: alert("Eddelu Manjunatha");
 break;
 case 4: alert("Milana");
 break;
 default: alert("Ooops..Invalid choice :O");
 break;
 }
}

```

### //radio\_click2r.js

```

var dom = document.getElementById("myForm");
dom.getElementById("1").onclick = dChoice;
dom.getElementById("2").onclick = dChoice;
dom.getElementById("3").onclick = dChoice;
dom.getElementById("4").onclick = dChoice;

```

There are two advantages to registering handlers as properties over registering them in XHTML attributes.

- First, it is good to keep XHTML and Java-Script separated in the document. This allows a kind of modularization of XHTML documents, resulting in a cleaner design that will be easier to maintain.
- Second, having the handler function registered as the value of a property allows for the possibility of changing the function during use.

## THE FOCUS EVENT

```
// nochange.html
<html>
<head><title>nochange.html</title>
<script type = "text/javascript" src = "nochange.js">
</script>
</head>
<body>
<form action = " " >
<h3> Non-Veg Items Order Form</h3>
<table border="border">
<tr>
<th>Item</th>
<th>Price</th>
<th>Quantity</th>
</tr>
<tr>
<th>Chicken Kabab (full)</th>
<td>Rs. 150</td>
<td><input type = "text" id = "chicken" size = "2"/></td>
</tr>
<tr>
<th>Mutton Kaima (half)</th>
<td>Rs. 250</td>
<td><input type = "text" id = "mutton" size = "2"/></td>
</tr>
<tr>
<th>Fish Fry (2 pieces)</th>
<td>Rs. 100</td>
<td><input type = "text" id = "fish" size = "2"/></td>
</tr>
</table>
<p>
<input type = "button" value = "Total Cost" onclick =
"computeCost();"/>
<input type = "text" size = "5" id = "cost" onfocus =
"this.blur();"/>
</p>
<p>
<input type = "submit" value = "Submit Order"/>
<input type = "reset" value = "Clear Order Form"/>
</p>
</form>
</body>
</html>
```

```
/nochange.js
function computeCost()
{
var chicken = document.getElementById("chicken").value;
var mutton = document.getElementById("mutton").value; var
fish = document.getElementById("fish").value;

document.getElementById("cost").value = totalCost = chicken*150
+ mutton*250 + fish*100;
}
```

### Output:

Item	Price	Quantity
Chicken Kabab (full)	Rs. 150	
Mutton Kaima (half)	Rs. 250	
Fish Fry (2 pieces)	Rs. 100	

Total Cost

After taking the entries, we get,

Item	Price	Quantity
Chicken Kabab (full)	Rs. 150	2
Mutton Kaima (half)	Rs. 250	1
Fish Fry (2 pieces)	Rs. 100	3

Total Cost

## VALIDATING FORM INPUT

- One of the common uses of JavaScript is to check the values provided in forms by users to determine whether the values are sensible.
- When a user fills in a form input element incorrectly and a JavaScript event handler function detects the error, the function should produce an `alert` message indicating the error to the user and informing the user of the correct format for the input.
- The form in the next example includes the two password input elements, along with `Reset` and `Submit` buttons.
- The JavaScript function that checks the passwords is called either when the `Submit` button is pressed, using the `onsubmit` event to trigger the call, or when the second text box loses focus, using the `blur` event.

The function performs two different tests

- First, it determines whether the user typed the initial password (in the first input box) by testing the value of the element against the empty string. If no password has been typed into the first field, the function calls `alert` to produce an error message and returns `false`.
- The second test determines whether the two typed passwords are the same. If they are different, once again the function calls `alert` to generate an error message and returns `false`.
- If they are the same, it returns `true`.

//pswd\_chk.html

```
<html>
<head>
<title>Password Checking</title>
<script type = "text/javascript" src = "pswd_chk.js">
</script>
</head>
<body>
<h3>Password Input</h3>
<form id="myForm" action= " ">
<p>
<label>Your Password:</label> <input id="initial" type="password" size="10"/>

<label>Verify Password:</label> <input id="final" type="password" size="10"/>

<input type="reset" name="Reset"/>
<input type="submit" name="Submit"/>
</p>
</form>
<script type = "text/javascript" src = "pswd_chkr.js">
</script>
</body>
</html>
```

//pswd\_chk.js

```
function chkPass()
{
var init=document.getElementById("initial");
var fin=document.getElementById("final");
if(init.value=="")
{
alert("You did not enter a Password\n" + "Please enter atleast now");
init.focus();
return false;
}

if(init.value!=fin.value)
{
alert("The passwords you entered do not match.Try Again");
init.focus();
init.select();
return false;
}
Else return true;
}
```

## Output:



We now consider an example that checks the validity of the form values for a name and phone number obtained from text boxes. The pattern for matching names [LastName, FirstName, MiddleName] is as follows:

`/^ [A-Z] [a-z] +, ? [A-Z] [a-z] +, ? [A-Z] \. ?$/`

The pattern for phone numbers is as follows:

`/^d{3}-\d{8}$/`

The following is the XHTML document, `validator.html`, that displays the text boxes for a customer's name and phone number:

//validator.html

```
<html>
<head>
<title>Name and Phone check</title>
<script type = "text/javascript" src = "validator.js">
</script>
</head>
<body>
<h3>enter your details</h3>
```

```

<form action="">
<p>
<label><input type="text" id="custName"/>Name(last name, first name, middle initial)</label>

<label><input type="text" id="custPhone"/>Phone (ddd-ddddddd)</label>

<input type="reset" id="reset"/>
<input type="submit" id="submit"/>
</p>
</form>
<script type = "text/javascript">
document.getElementById("custName").onchange=chkName;
document.getElementById("custPhone").onchange=chkPhone;
</script>
</body>
</html>

```

```

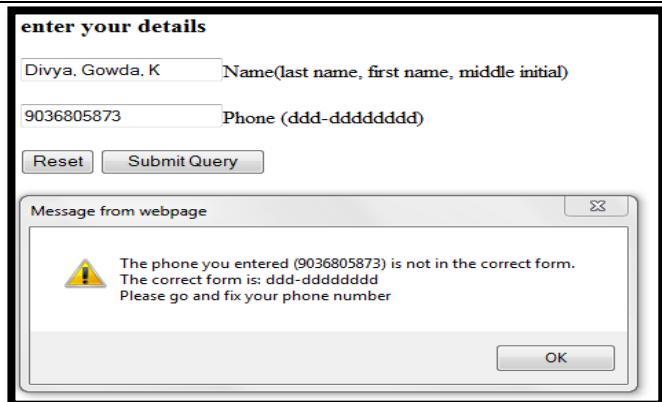
//validator.js function
chkName()
{
 var myName = document.getElementById("custName");
 var pos = myName.value.search(/^([A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?\$)/);
 if(pos != 0)
 {
 alert("The name you entered (" + myName.value + ") is not in the correct form.\n" + "The
 correct form is: " + "last-name, first-name, middle-initial \n" +
 "Please go and fix your name");
 myName.focus();
 myName.select(); return
 false;
 }
 else return true;
}
function chkPhone()
{
 var myPhone = document.getElementById("custPhone");
 var pos = myPhone.value.search(/^\d{3}-\d{8}$/); if(pos !=
 0)
 {
 alert("The phone you entered (" + myPhone.value + ") is not in the correct form.\n" +
 "The correct form is: " + "ddd-ddddddd \n" +
 "Please go and fix your phone number");
 myPhone.focus();
 myPhone.select(); return
 false;
 }
 else return true;
}

```

## OUTPUT:

**enter your details**

Divya, Gowda, K	Name(last name, first name, middle initial)
080-26525212	Phone (ddd-ddddddd)
<input type="button" value="Reset"/>	<input type="button" value="Submit Query"/>



## THE DOM 2 EVENT MODEL

The DOM 2 model is a modularized interface. One of the DOM 2 modules is Events, which includes several sub-modules. The ones most commonly used are HTMLEvents and MouseEvents. The interfaces and events defined by these modules are as follows:

Module	Event Interface	Event Types
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	click, mousedown, mousemove, mouseout, mouseover, mouseup

## **EVENT PROPAGATION:**

- A browser which understands DOM, on receiving the XHTML document from the server, creates a tree known as document tree.
- The tree constructed consists of elements of the document except the HTML
- The root of the document tree is document object itself
- The other elements will form the node of the tree
- In case of DOM2, the node which generates an event is known as target node
- Once the event is generated, it starts the propagation from root node
- During the propagation, if there are any event handlers on any node and if it is enabled then event handler is executed
- The event further propagates and reaches the target node.
- When the event handler reaches the target node, the event handler gets executed
- After this execution, the event is again re-propagated in backward direction
- During this propagation, if there are any event handlers which are enabled, will be executed.
- The propagation of the even from the root node towards the leaf node or the target node is known as **capturing phase**.
- The execution of the event handler on the target node is known as **execution phase**.
- This phase is similar to event handling mechanism in DOM – 0
- The propagation of the event from the leaf or from the target node is known as **bubbling phase**
- All events cannot be bubbled for ex: load and unload event
- If user wants to stop the propagation of an event, then stop propagation has to be executed.

## **EVENT REGISTRATION:**

- In case of DOM2, the events get registered using an API known as **addEventListener**
- The first arg is the eventName. Ex: click, change, blur, focus
- The second arg is the event handler function that has to be executed when there is an event
- The third arg is a Boolean argument that can either take a true or false value
- If the value is true, it means event handler is enabled in capturing phase
- If the event value if off (false), then event handler is enabled at target node
- The addEventListener method will return event object to eventhandler function. The event object can be accessed using the keyword “Event”
- The address of the node that generated event will be stored in **current target**, which is property of **event object**

## **AN EXAMPLE OF THE DOM 2 EVENT MODEL**

The next example is a revision of the validator.html document and validator.js script from previous example, which used the DOM 0 event model. Because this version uses the DOM 2 event model, it does not work with IE8.

```
//validator2.html
<html>
<head>
<title>Illustrate form input validation with DOM 2</title>
```

```

<script type = "text/javascript" src = "validator2.js">
</script>
</head>
<body>
<h3>enter your details</h3>
<form action="">
<p>
<label><input type="text" id="custName"/>Name(last name, first name, middle initial)</label>

<label><input type="text" id="custPhone"/>Phone (ddd-ddddddd)</label>

<input type="reset" />
<input type="submit" id="submitButton"/>
</p>
</form>
<script type = "text/javascript" src = "validator2r.js"/>
</body>
</html>

```

```

//validator2.js
function chkName(event)
{
var myName = event.currentTarget;
var pos = myName.value.search(/^[A-Z][a-z]+, ?[A-Z][a-z]+, ?[A-Z]\.?\$/);
if(pos != 0)
{
alert("The name you entered (" + myName.value + ") is not in the correct form.\n" +
 "correct form is: " + "last-name, first-name, middle-initial \n" +
 "Please go and fix your name");
myName.focus();
myName.select();
}
}
function chkPhone(event)
{
var myPhone = event.currentTarget;
var pos = myPhone.value.search(/^\d{3}-\d{8}\$/);
if(pos != 0)
{
alert("The phone you entered (" + myPhone.value + ") is not in the correct form.\n" +
 "The correct form is: " + "ddd-ddddddd \n" +
 "Please go and fix your phone number");
myPhone.focus();
myPhone.select();
}
}

```

```

//validator2r.js
var c = document.getElementById("custName"); var
p = document.getElementById("custPhone");
c.addEventListener("change",chkName,false);
p.addEventListener("change",chkPhone,false);

```

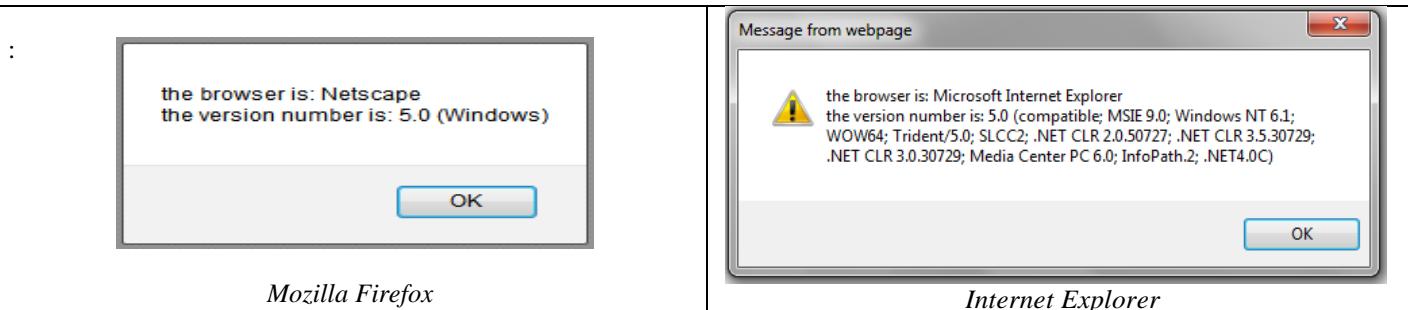
## THE navigator OBJECT

The navigator object indicates which browser is being used to view the XHTML document. The browser's name is stored in the appName property of the object. The version of the browser is stored in the appVersion property of the object. These properties allow the script to determine which browser is being used and to use

processes appropriate to that browser. The following example illustrates the use of navigator, in this case just to display the browser name and version number:

```
//navigate.html file
<html>
<head>
<title>Navigator</title>
<script type = "text/javascript" src = "navigate.js">
</script>
</head>
<body onload = "navProperties()">
</body>
</html>
//navigate.js file function
navProperties()
{
alert("the browser is: " + navigator.appName + "\n" + "the
version number is: " + navigator.appVersion + "\n");
}
```

#### OUTPUT



## DOM TREE TRAVERSAL AND MODIFICATION

### **DOM TREE TRAVERSAL**

The `parentNode` property has the DOM address of the parent node of the node through which it is referenced. The `childNodes` property is an array of the child nodes of the node through which it is referenced. The `previousSibling` property has the DOM address of the previous sibling node of the node through which it is referenced. The `nextSibling` property has the DOM address of the next sibling node of the node through which it is referenced. The `firstChild` and `lastChild` properties have the DOM addresses of the first and last child nodes, respectively, of the node through which they are referenced. The `nodeType` property has the type of the node through which it is referenced.

### **DOM TREE MODIFICATION**

A number of methods allow JavaScript code to modify an existing DOM tree structure. The `insertBefore(newChild, refChild)` method places the `newChild` node before the `refChild` node. The `replaceChild(newChild, oldChild)` method replaces the `oldChild` node with the `newChild` node. The `removeChild(oldChild)` method removes the specified node from the DOM structure. The `appendChild(newChild)` method adds the given node to the end of the list of siblings of the node through which it is called.

# UNIT 6

## DYNAMIC DOCUMENTS WITH JAVASCRIPT

### INTRODUCTION

- Informally, a dynamic XHTML document is an XHTML document that, in some way, can be changed while it is being displayed by a browser.
- Dynamic XHTML is not a new markup language.
- It is a collection of technologies that allows dynamic changes to documents defined with XHTML.
- Specifically, a dynamic XHTML document is an XHTML document whose tag attributes, tag contents, or element style properties can be changed by user interaction or the occurrence of a browser event after the document has been, and is still being, displayed.
- Such changes can be made with an embedded script that accesses the elements of the document as objects in the associated DOM structure.

### POSITIONING ELEMENTS

- ▲ Cascading Style Sheet – Positioning (CSS-P) is completely supported by IE8 and FX3.
- ▲ It provides the means not only to position any element anywhere in the display of a document, but also to move an element to a new position in the display dynamically, using JavaScript to change the positioning style properties of the element.
- ▲ These style properties, which are appropriately named `left` and `top`, dictate the distance from the left and top of some reference point to where the element is to appear.
- ▲ Another style property, `position`, interacts with `left` and `top` to provide a higher level of control of placement and movement of elements.
- ▲ The `position` property has three possible values: `absolute`, `relative`, and `static`.

### ABSOLUTE POSITIONING

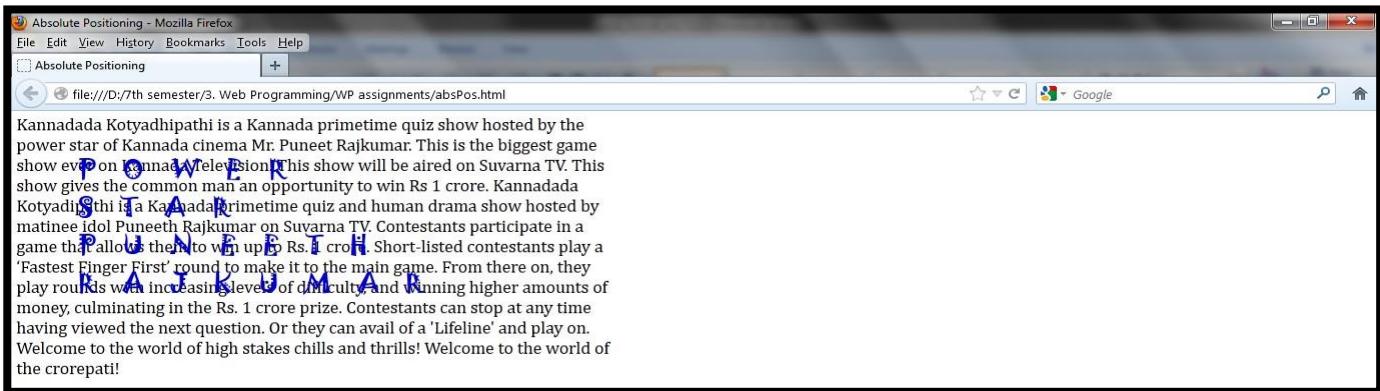
- ★ The `absolute` value is specified for `position` when the element is to be placed at a specific place in the document display without regard to the positions of other elements.
- ★ One use of absolute positioning is to superimpose special text over a paragraph of ordinary text to create an effect similar to a watermark on paper.
- ★ A larger italicized font, in a light-gray color and with space between the letters, could be used for the special text, allowing both the ordinary text and the special text to be legible.

```
//absPos.html
<html>
<title>Absolute Positioning</title>
<style type = "text/css">
.regtext
{
font-family: Cambria;
font-size:20pt; width:
900px;
}
.abstext
{
position: absolute;
top:25px;
left:100px;
font-family: jokerman;
font-size:30pt;
```

```

width: 500px;
color:#0000cd;
letter-spacing: 1em;
}
</style>
</head>
<body>
<p class="regtext">
Kannadada Kotyadhipathi is a Kannada primetime quiz show hosted by the power star of Kannada cinema Mr. Puneet Rajkumar. This is the biggest game show ever on Kannada Television. This show will be aired on Suvarna TV. This show gives the common man an opportunity to win Rs 1 crore. Kannadada Kotyadipathi is a Kannada primetime quiz and human drama show hosted by matinee idol Puneeth Rajkumar on Suvarna TV. Contestants participate in a game that allows them to win up to Rs. 1 crore. Short-listed contestants play a 'Fastest Finger First' round to make it to the main game. From there on, they play rounds with increasing levels of difficulty, and winning higher amounts of money, culminating in the Rs. 1 crore prize. Contestants can stop at any time having viewed the next question. Or they can avail of a 'Lifeline' and play on. Welcome to the world of high stakes chills and thrills! Welcome to the world of the crorepati!
</p>
<p class="abstext">
POWER STAR PUNEETH RAJKUMAR
</p>
</body>
</html>

```



To illustrate the placement of nested elements, the document `absPos.html` is modified to place the regular text 100 pixels from the top and 100 pixels from the left. The special text is nested inside the regular text by using `<div>` and `<span>` tags. The modified document, which is named `absPos2.html`, is as follows:

```

//absPos.html
<html>
<title>Nested Absolute Positioning</title>
<style type = "text/css">
.regtext
{
font-family: Cambria;
font-size: 20pt; width:
900px;
position:absolute; top:
100px;
left: 100px;
}
.abstext
{
position:absolute;
top:25px;
left:100px;
font-family: jokerman;
font-size:50pt;
width: 500px; color:#ddd000;

```

```

letter-spacing: 1em;
}
</style>
</head>
<body>
<div class="regtext">

```

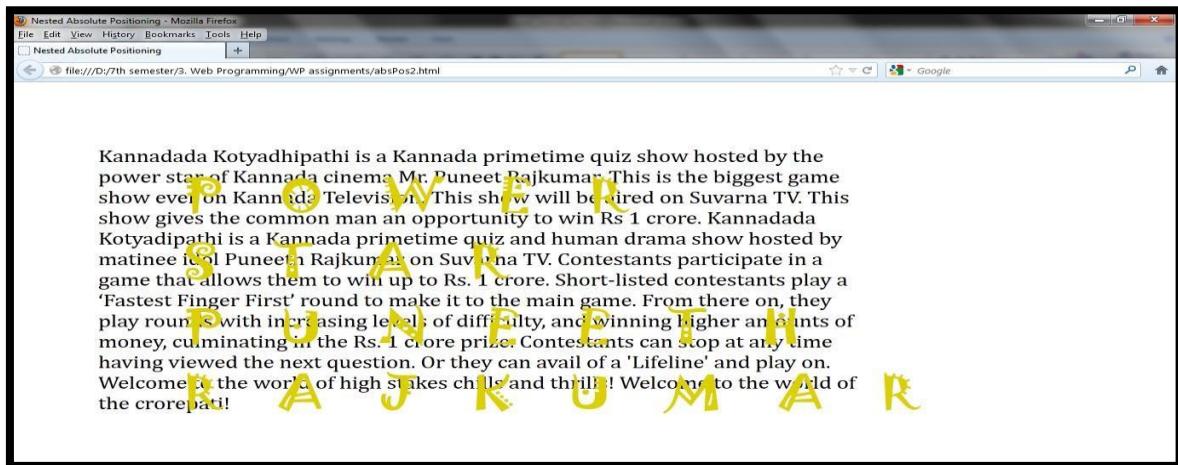
Kannadada Kotyadhipathi is a Kannada primetime quiz show hosted by the power star of Kannada cinema Mr. Puneet Rajkumar. This is the biggest game show ever on Kannada Television. This show will be aired on Suvarna TV. This show gives the common man an opportunity to win Rs 1 crore. Kannadada Kotyadipathi is a Kannada primetime quiz and human drama show hosted by matinee idol Puneeth Rajkumar on Suvarna TV. Contestants participate in a game that allows them to win up to Rs. 1 crore. Short-listed contestants play a 'Fastest Finger First' round to make it to the main game. From there on, they play rounds with increasing levels of difficulty, and winning higher amounts of money, culminating in the Rs. 1 crore prize. Contestants can stop at any time having viewed the next question. Or they can avail of a 'Lifeline' and play on. Welcome to the world of high stakes chills and thrills! Welcome to the world of the crorepati!

```


POWER STAR PUNEETH RAJKUMAR

</div>
</body>
</html>

```



## RELATIVE POSITIONING

- ★ An element that has the `position` property set to `relative`, but does not specify `top` and `left` property values, is placed in the document as if the `position` attribute were not set at all.
- ★ However, such an element can be moved later.
- ★ If the `top` and `left` properties are given values, they displace the element by the specified amount from the position where it would have been placed.
- ★ In both the case of an absolutely positioned element inside another element and the case of a relatively positioned element, negative values of `top` and `left` displace the element upward and to the left, respectively.
- ★ Relative positioning can be used for a variety of special effects in placing elements.

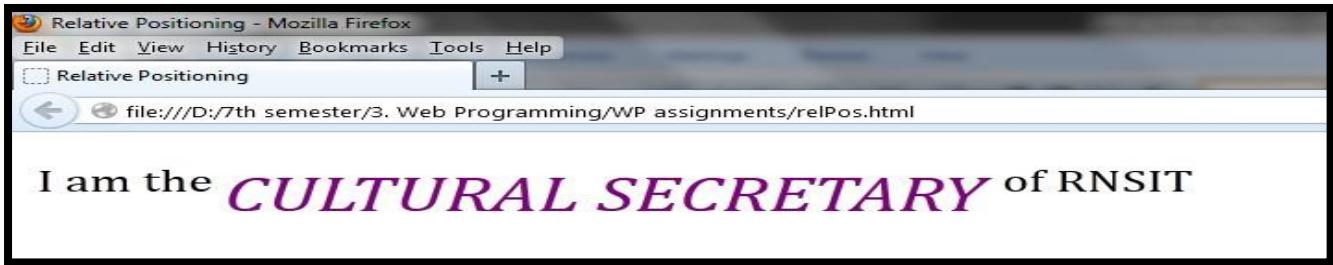
```

//relPos.html
<html>
<head><title>Relative Positioning</title></head>
<body>
<p> I am the

CULTURAL SECRETARY

of RNSIT </p>
</body>
</html>

```



## STATIC POSITIONING

The default value for the `position` property is `static`. A statically positioned element is placed in the document as if it had the `position` value of `relative` but no values for `top` or `left` were given. The difference is that a statically positioned element cannot have its `top` or `left` properties initially set or changed later. Therefore, a statically placed element cannot be displaced from its normal position and cannot be moved from that position later.

## MOVING ELEMENTS

Moving an element is simple:

- ★ Changing the `top` or `left` property values causes the element to move on the display.
- ★ If its `position` is set to `absolute`, the element moves to the new values of `top` and `left`; if its `position` is set to `relative`, it moves from its original position by distances given by the new values of `top` and `left`.
- ★ In the next example, an image is absolutely positioned in the display.
- ★ The document includes two text boxes, labeled `x` coordinate and `y` coordinate, respectively.
- ★ The user can enter new values for the `left` and `top` properties of the image in these boxes.
- ★ When the button labeled **Move It** is pressed, the values of the `left` and `top` properties of the image are changed to the given values, and the element is moved to its new position.
- ★ A JavaScript function, stored in a separate file, is used to change the values of `left` and `top` in this example.

```
//mover.html
<html>
<head><title>Moving Elements</title>
<script type = "text/javascript" src = "mover.js">
</script>
</head>
<body>
<form action="">
<p>
<label>
x - coordinate: <input type = "text" id = "leftCoord" size = "3" />
</label>

<label>
y - coordinate: <input type = "text" id = "topCoord" size = "3" />
</label>

<input type="button" value="Move It" onclick = "moveIt('rnsit', document.getElementById('topCoord').value,
document.getElementById('leftCoord').value)" />
</p>
</form>
<div id = "rnsit" style = "position:absolute;top:115px;left:0;">

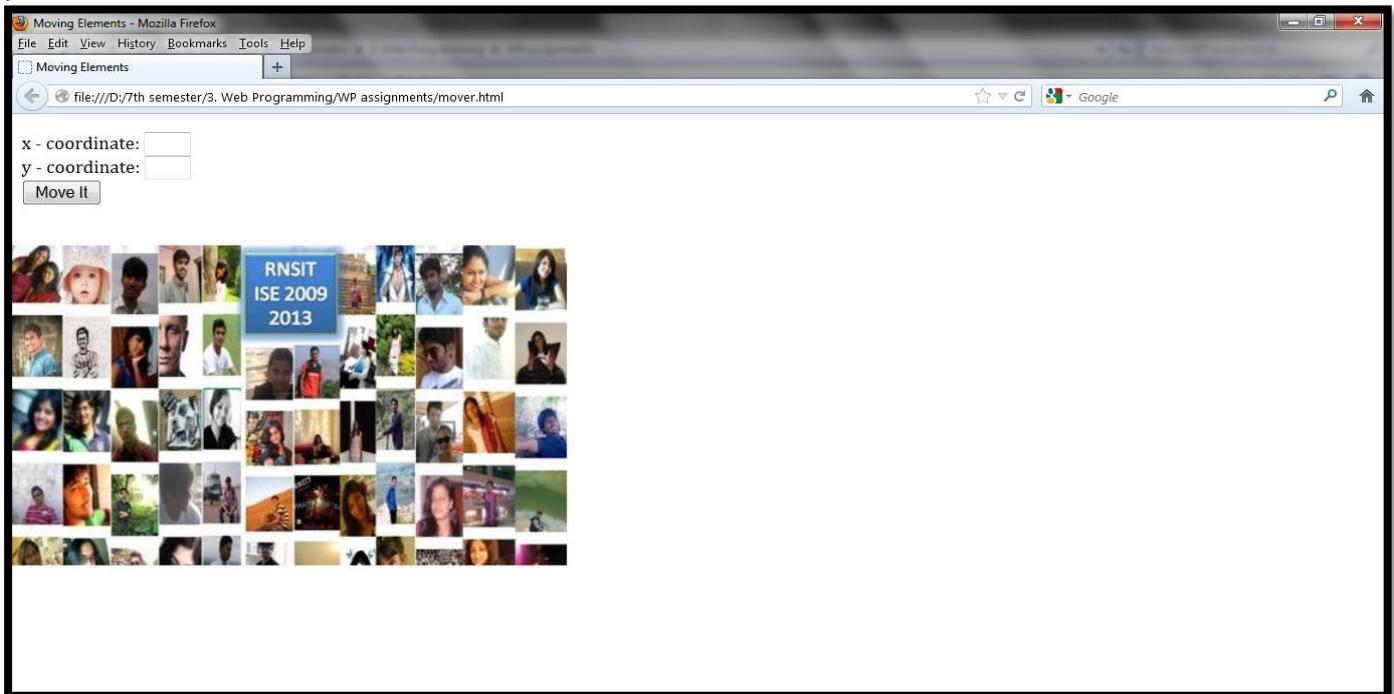
```

```

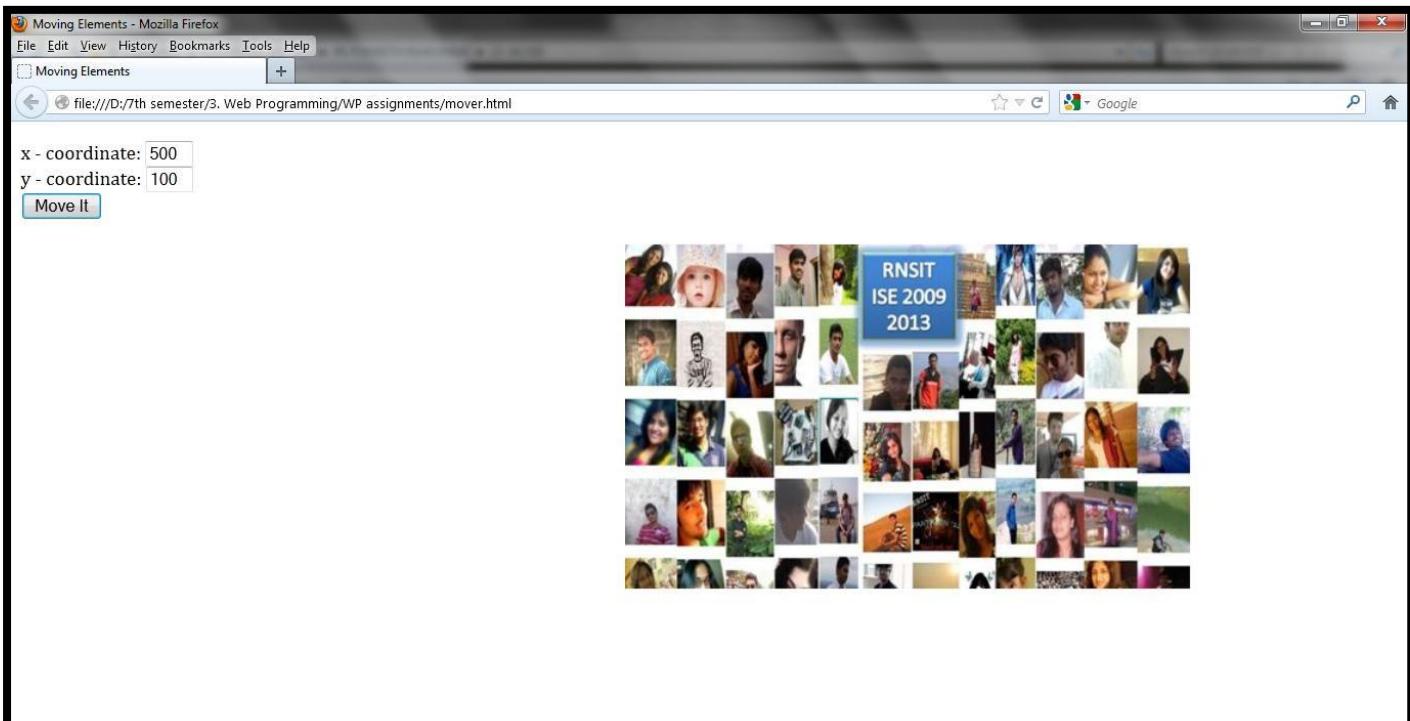
</div>
</body>
</html>

//mover.js
function moveIt(movee, newTop, newLeft)
{
 dom = document.getElementById(movee).style;
 dom.top=newTop + "px";
 dom.left=newLeft + "px";
}

```



After setting the coordinates and pressing “Move It”, we get the following output:



## ELEMENT VISIBILITY

- ★ Document elements can be specified to be visible or hidden with the value of their `visibility` property.
- ★ The two possible values for `visibility` are, quite naturally, `visible` and `hidden`.
- ★ The appearance or disappearance of an element can be controlled by the user through a widget.

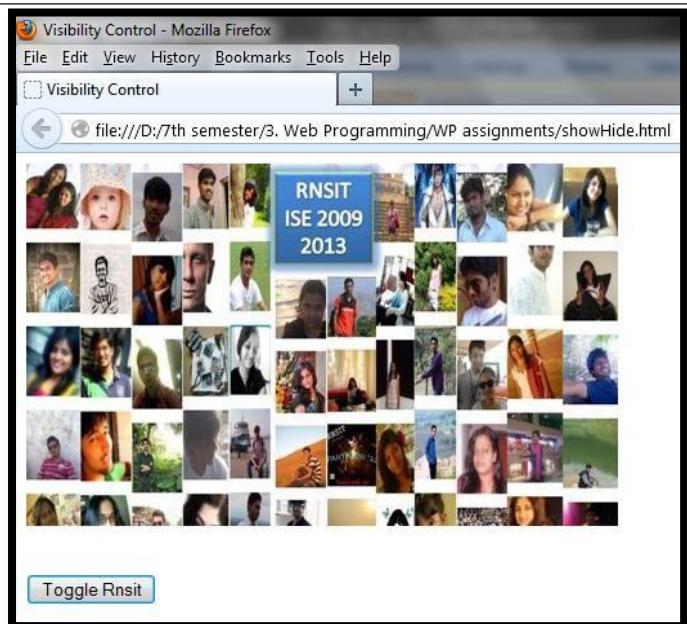
//showHide.html

```
<html>
<head>
<title>Visibility Control</title>
<script type="text/javascript" src="showHide.js">
</script>
</head>
<body>
<form action="">
<div id="rnsit" style="position:relative; visibility: visible;">

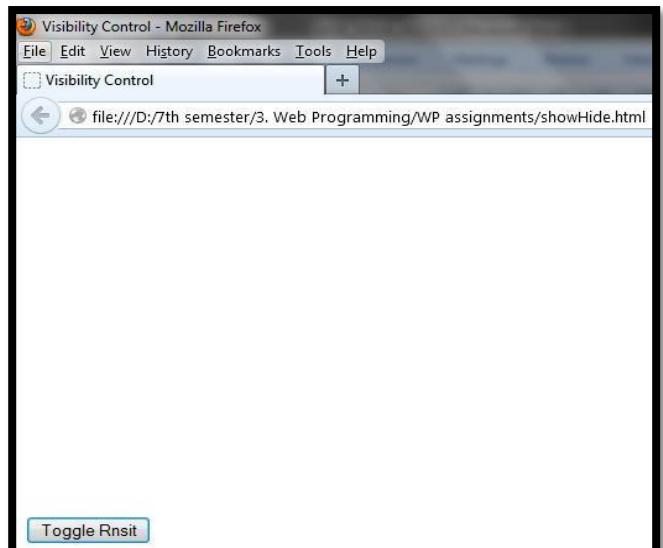
</div>
<p>
<input type="button" value="Toggle Rnsit" onclick = "flipImag()" /></p>
</form>
</body>
</html>
```

//showHide.js

```
function flipImag()
{
 dom=document.getElementById("rnsit").style;
 if(dom.visibility == "visible")
 dom.visibility = "hidden";
 else
 dom.visibility = "visible";
}
```



After clicking on Toggle Rnsit button, we get,



## CHANGING COLORS AND FONTS

The background and foreground colors of the document display can be dynamically changed, as can the font properties of the text.

## CHANGING COLORS

- ★ Dynamic changes to colors are relatively simple.
- ★ In the next example, the user is presented with two text boxes into which color specifications can be typed—one for the document background color and one for the foreground color.
- ★ The colors can be specified in any of the three ways that color properties can be given in CSS.
- ★ A JavaScript function that is called whenever one of the text boxes is changed makes the change in the document's appropriate color property: `backgroundColor` or `color`.
- ★ The first of the two parameters to the function specifies whether the new color is for the background or foreground; the second specifies the new color.
- ★ The new color is the `value` property of the text box that was changed by the user.

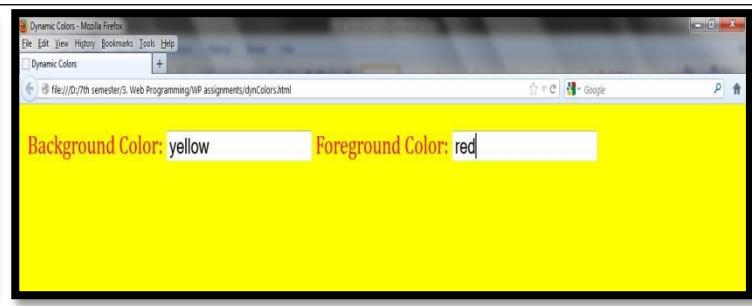
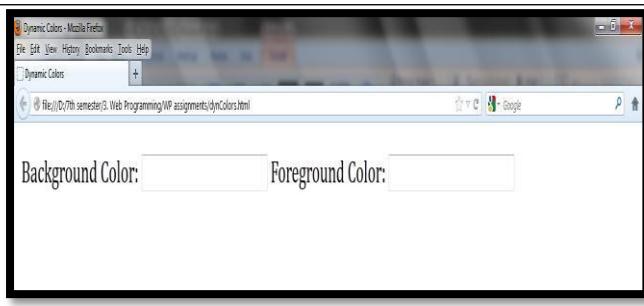
//dynColor.html

```
<html>
<head>
<title>Dynamic Colors</title>
<script type="text/javascript" src = "dynColors.js">
</script>
</head>
<body>
<form action="">
<p>
<label>Background Color: <input type="text" name="bg" size = "20" onchange="setColor('bg', this.value)" /></label>
<label>Foreground Color: <input type="text" name="fg" size = "20" onchange="setColor('fg', this.value)" /> </label>

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

//dynColor.js

```
function setColor(where, newValue)
{
 if(where == "bg")
 document.body.style.backgroundColor = newValue;
 else
 document.body.style.color = newValue;
}
```



## CHANGING FONTS

- ★ Web users are accustomed to having links in documents change color when the cursor is placed over them.
- ★ Use of the `mouseover` event to trigger a JavaScript event handler allows us to change any property of any element in a document, including text, when the mouse cursor is placed over it.
- ★ Thus, the font style and font size, as well as the color and background color of text, can be changed when the cursor is placed over the text.
- ★ The text can be changed back to its original form when an event handler is triggered with the `mouseout` event.

//dynLink.html

```
<html>
```

```

<head><title>Changing Fonts</title>
<style type = "text/css">
.regtext
{
font: cambria;
font-size:16pt;
}
</style>
</head>
<body>
<cp class="regtext">
The batch of
<a style = "color:purple;"
onmouseover="this.style.color='red'; this.style.font='italic 20pt Jokerman';"
onmouseout ="this.style.color='blue'; this.style.font='bold 17pt Times';" > RNSIT
ISE 2009-2013

always Rocks..!!!
</p>
</body>
</html>

```

### INITIAL



The batch of **RNSIT ISE 2009-2013** always Rocks..!!!

### MOUSEOVER



The batch of **RNSIT ISE 2009-2013** always Rocks..!!!

### MOUSEOUT



The batch of **RNSIT ISE 2009-2013** always Rocks..!!!

## DYNAMIC CONTENT

- ★ We now develop an example that illustrates one use of dynamic content.
- ★ Assistance to a browser user filling out a form can be provided with an associated text area, often **help box**.
- ★ The content of the help box can change, depending on the placement of the mouse cursor.
- ★ When the cursor is placed over a particular input field, the help box can display advice on how the **field** to be filled in.

- ★ When the cursor is moved away from an input field, the help box content can be changed to `tiny` to indicate that assistance is available.
- ★ In the next example, an array of messages that can be displayed in the help box is defined in JavaScript.
- ★ When the mouse cursor is placed over an input field, the `mouseover` event is used to call a `handler` function that changes the help box content to the appropriate value.
- ★ The appropriate value is specified with a parameter sent to the handler function.
- ★ The `mouseout` event is used to trigger the change of the content of the help box back to the `'standard'` value.

### //dynValue.html

```

<html>
<head>
<title>Dynamic Values</title>
<script type = "text/javascript" src = "dynValue.js">
</script>
</head>
<body>
<form action="">
<p style = "font-weight: bold">

Customer Information

<label>Name:
<input type="text" onmouseover="messages(0)" onmouseout="messages(4)" />
</label>

<label>Email:
<input type="text" onmouseover="messages(1)" onmouseout="messages(4)" />
</label>

To create an account, provide the following information:

<label>User ID:
<input type="text" onmouseover="messages(2)" onmouseout="messages(4)" />
</label>

<label>Password:
<input type="text" onmouseover="messages(3)" onmouseout="messages(4)" />
</label>

<textarea id="adviceBox" rows="3" cols="50" style="position:absolute; left:250px; top:0px">
This box provides advice on filling out the form on this page. Put the mouse cursor over any input field to get advice.
</textarea>

<input type="submit" value="Submit" />
<input type="reset" value="Reset" />
</p>
</form>
</body>
</html>

```

### //dynValue.js

```

var helpers = ["Your name must be in the form: \n first name, middle initial., last name",
 "Your email address must have the form: \ user@domain",
 "Your user ID must have at least six characters and it must include one digit",
 "This box provides advice on filling out the form on this page. Put the mouse cursor over any input field to get advice"]

function messages(adviceNumber)
{
 document.getElementById("adviceBox").value=helpers[adviceNumber];
}

```

**Dynamic Values - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

Dynamic Values +

file:///D:/7th semester/3. Web Programming/WP assignments/dynValue.html

**Customer Information**

Name:

Email:

This box provides advice on filling out the form on this page. Put the mouse cursor over any input field to get advice.

To create an account, provide the following information:

User ID:

Password:

Submit Reset

Initially

**Dynamic Values - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

Dynamic Values +

file:///D:/7th semester/3. Web Programming/WP assignments/dynValue.html

**Customer Information**

Name:

Email:

Your name must be in the form:  
first name, middle initial., last name

To create an account, provide the following information:

User ID:

Password:

Submit Reset

Cursor  
on Name

**Dynamic Values - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

Dynamic Values +

file:///D:/7th semester/3. Web Programming/WP assignments/dynValue.html

**Customer Information**

Name:

Email:

Your email address must have the form: user@domain

To create an account, provide the following information:

User ID:

Password:

Submit Reset

Cursor  
on Email

**Dynamic Values - Mozilla Firefox**

File Edit View History Bookmarks Tools Help

Dynamic Values +

file:///D:/7th semester/3. Web Programming/WP assignments/dynValue.html

**Customer Information**

Name:

Email:

Your user ID must have at least six characters and it must include one digit

To create an account, provide the following information:

User ID:

Password:

Submit Reset

Cursor on  
User ID

## STACKING ELEMENTS

★ The top and left properties allow the placement of an element anywhere in the two dimensions of the display of a document.

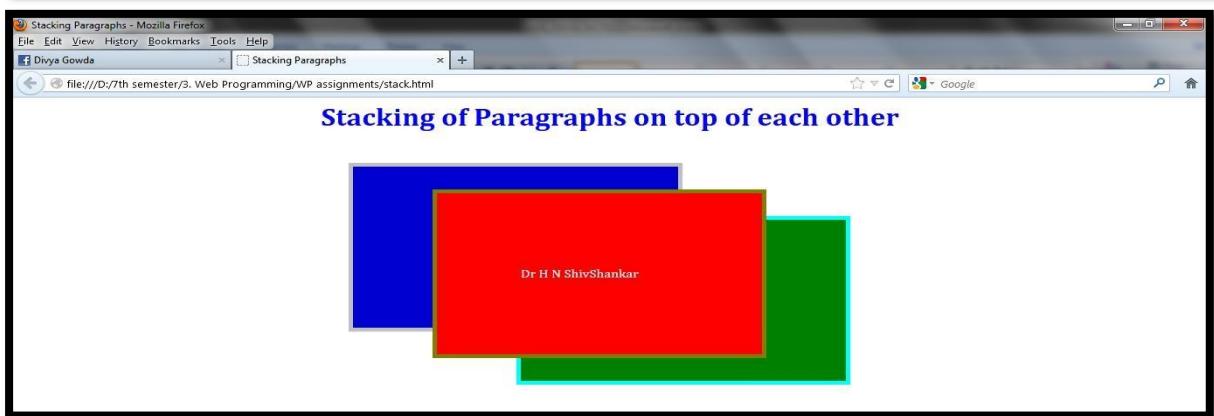
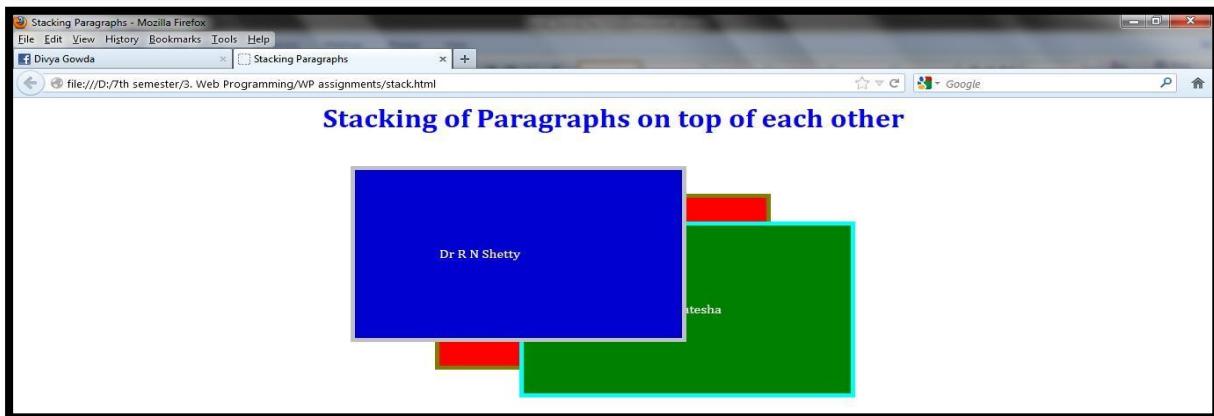
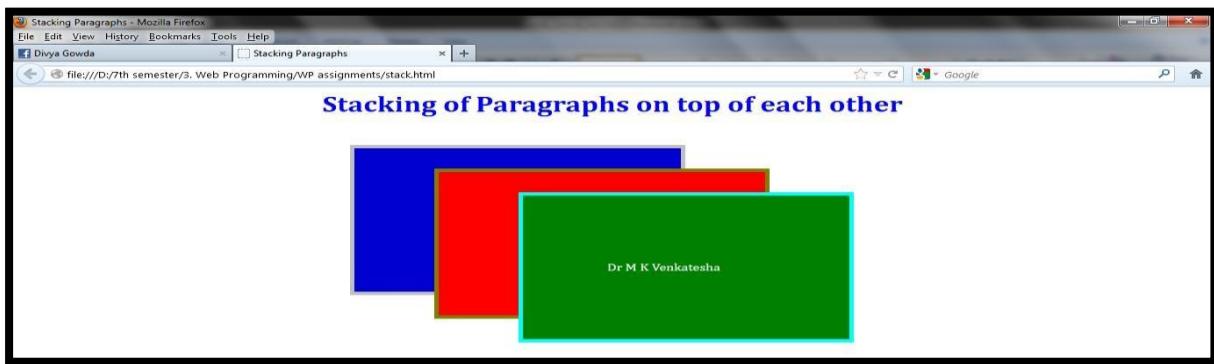
- ★ Although the display is restricted to two physical dimensions, the effect of a third dimension is ~~possible~~ through the simple concept of stacked elements, such as that used to stack windows in graphical user interfaces.
- ★ Although multiple elements can occupy the same space in the document, one is considered to be on ~~top~~ and is displayed.
- ★ The top element hides the parts of the lower elements on which it is superimposed.
- ★ The placement of elements in this third dimension is controlled by the `z-index` attribute of the element.
- ★ An element whose `z-index` is greater than that of an element in the same space will be ~~displayed~~ over the other element, effectively hiding the element with the smaller `z-index` value.
- ★ The JavaScript style property associated with the `z-index` attribute is `zIndex`.

//stack.html

```
<html>
<head><title>Stacking Paragraphs</title>
<style type="text/css">
 .para1
 {
 border: solid thick #C0C0C0;
 padding: 1in;
 width:180px;
 background-color:#0000D0;
 color:white; position:absolute;
 top:70px;
 left:4in;
 z-index:1;
 }
 .para2
 {
 border: solid thick #808000;
 padding: 1in;
 width:180px;
 background-color:red;
 color:white;
 position:absolute;
 top:105px;
 left:5in;
 z-index:2;
 }
 .para3
 {
 border: solid thick #00ffff;
 padding: 1in; width:180px;
 background-color:green;
 color:white;
 position:absolute;
 top:140px;
 left:6in;
 z-index:3;
 }
 .display
 {
 font-size:25pt;
 color:blue;
 text-align:center;
 }
 p:hover{background-color:rgb(250,200,150);font-size:25px;color:white;}
</style>
<script type="text/javascript">
 var stack1="stack1";
 function move(curStack)
 {
```

```
var oldStack=document.getElementById(stack1).style;
oldStack.zIndex="0";
var newStack=document.getElementById(curStack).style;
newStack.zIndex="10";
stack1=document.getElementById(curStack).id;
}
</script>
</head>
<body>

<h2 class="display">Stacking of Paragraphs on top of each other</h2>
<p class="para1" id="stack1" onmouseover="move('stack1')"> Dr
 R N Shetty
</p>
<p class="para2" id="stack2" onmouseover="move('stack2')"> Dr
 H N ShivShankar
</p>
<p class="para3" id="stack3" onmouseover="move('stack3')"> Dr
 M K Venkatesha
</p>
</body>
</html>
```



## LOCATING THE MOUSE CURSOR

- ★ A mouse-click event is an implementation of the `Mouse-Event` interface, which defines two pairs of properties that provide geometric coordinates of the position of the element in the display that created the event.
- ★ One of these pairs, `clientX` and `clientY`, gives the coordinates of the element relative to the top-left corner of the browser display window, in pixels.
- ★ The other pair, `screenX` and `screenY`, also gives coordinates of the element, but relative to the computer's screen.
- ★ In the next example, `where.html`, two pairs of text boxes are used to display these four properties every time the mouse button is clicked.
- ★ The handler is triggered by the `onclick` attribute of the body element.
- ★ The call to the handler in this example sends `event`, which is a reference to the event object created in the element, as a parameter.

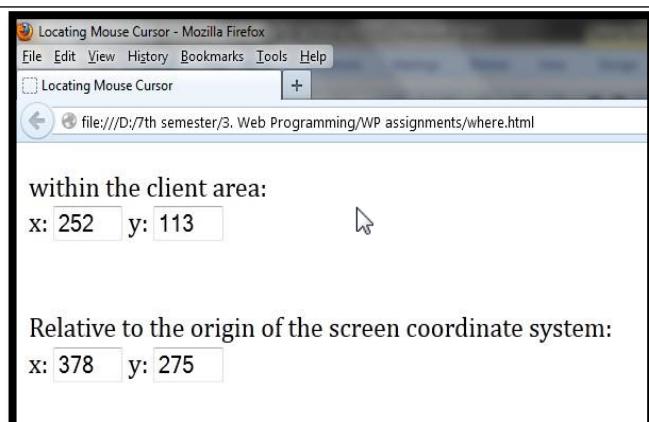
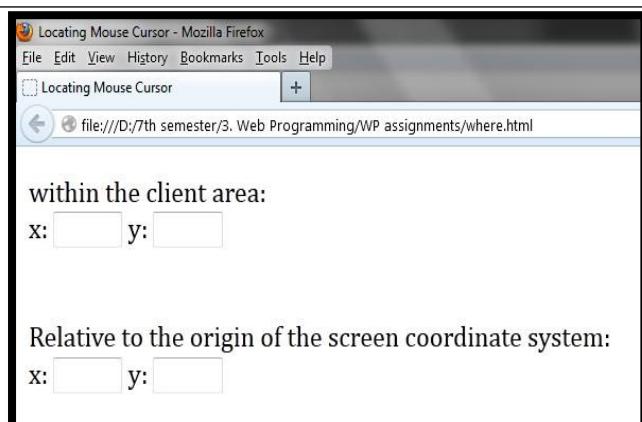
//where.html

```
<html>
<head>
<title>Locating Mouse Cursor</title>
<script type="text/javascript">
function findIt(evt)
{
 document.getElementById("xcoor1").value = evt.clientX;
 document.getElementById("ycoor1").value = evt.clientY;
 document.getElementById("xcoor2").value = evt.screenX;
 document.getElementById("ycoor2").value = evt.screenY;
}
</script>
</head>
<body onclick="findIt(event)">
<form action="">
<p>
within the client area:

x: <input type="text" id="xcoor1" size="4"/>
y: <input type="text" id="ycoor1" size="4"/>

Relative to the origin of the screen coordinate system:
 x:
<input type="text" id="xcoor2" size="4"/>
y: <input type="text" id="ycoor2" size="4"/>
</p>
</form>
</body>
</html>
```

NOTE: We are not entering any values, instead click somewhere on the screen, you will automatically get the x and y co-ordinate values in the text boxes.





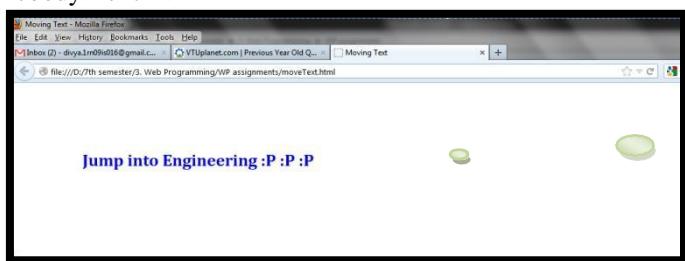
- ★ The second form of `setInterval` takes a variable number of parameters.
- ★ The first parameter is the name of a function to be called, the second is the interval in `milliseconds` between the calls to the function, and the remaining parameters are used as actual parameters to the function being called.
- ★ The initial position of the text is set in the `span` element that specifies the text.
- ★ The `onload` attribute of the `body` element is used to call a function, `initText`, to initialize the `x`- `y`- coordinates of the initial position to the `left` and `top` properties of the element and call the `mover` function.
- ★ The `mover` function, named `moveText`, takes the current coordinates of the text as parameters, moves them one pixel toward the final position, and then, using `setTimeout`, calls itself with the new coordinates.

### //moveText.html

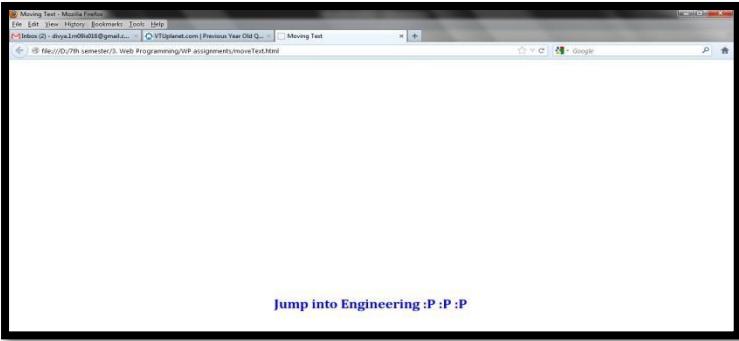
```

<html>
<head>
<title>Moving Text</title>
<script type="text/javascript">
var dom, x, y, finalx=500, finaly=500;
function initText()
{
 dom = document.getElementById('theText').style; var
 x=dom.left;
 var y=dom.top;
 x=x.match(/\d+/);
 y=y.match(/\d+/);
 moveText(x,y);
}
function moveText(x,y)
{
 if(x!=finalx)
 if(x>finalx) x--;
 else if(x<finalx) x++;
 if(y!=finaly)
 if(y>finaly) y--;
 else if(y<finaly) y++;
 if((x!=finalx)||(y!=finaly))
 {
 dom.left=x + "px";
 dom.top=y + "px";
 setTimeout("moveText(" + x + "," + y + ")", 1);
 }
}
</script>
<body onload = "initText()">
<p>
<span id = 'theText'
style = "position:absolute; left:100px; top:100px; font:bold 20pt cambria; color:blue;">Jump
into Engineering :P :P :P
</p>
</body></html>

```



Execute & check once. It will slide from top to bottom & do not directly jump from top.



## Dragging and Dropping Elements

- ★ The mouseup, mousedown, andmousemove events can be used to implement drag and drop.
- ★ To illustrate drag and drop, an XHTML document and a JavaScript file that creates a magnetic system is developed, showing two static lines of a poem and allowing the user to create the last two lines from a collection of movable words.
- ★ The DOM 0 model is used for the call to the handler for the mousedown event.
- ★ The rest of the process is designed with the DOM 2 model.
- ★ The mousedown event handler, grabber, takes the Event object as its parameter.
- ★ It gets the element to be moved from the currentTarget property of the Event object and putin a global variable so that it is available to the other handlers.
- ★ Then it determines the coordinates of the current position of the element to be moved and computesthe difference between each of them and the corresponding coordinates of the position of the mouse cursor.
- ★ The grabber handler also registers the event handlers formousemove andmouseup.
- ★ These two handlers are named mover and dropper, respectively.
- ★ The dropper handler disconnects mouse movements from the element-moving process by unregistering the handlers mover and dropper.

```
//dragNDrop.html
<html>
<head><title>Drag and Drop</title>
<script type = "text/javascript" src="dragNDrop.js">
</script>
</head>
<body>
<h3>Arrange the following subjects of VII Semester according to subject codes</h3>
<h3>1.
2.
3.
4.
5.
6.
</h3>
<p>
 SA
DM
 OOMD
 C#
 WP
 JAVA
</p>
</body>
</html>
```

```

//dragNDrop.js
var diffx, diffy, theElement;
function grabber(event)
{
 theElement=event.currentTarget;
 var posX=parseInt(theElement.style.left);
 var posY=parseInt(theElement.style.top);
 diffx=event.clientX - posX;
 diffy=event.clientY - posY;
 document.addEventListener("mousemove",mover,true);
 document.addEventListener("mouseup",dropper,true); event.stopPropagation();
 event.preventDefault();
}
function mover(event)
{
 theElement.style.left=(event.clientX - diffx) + "px";
 theElement.style.top=(event.clientY - diffy) + "px";
 event.stopPropagation();
}
function dropper(event)
{
 document.removeEventListener("mouseup", dropper, true);
 document.removeEventListener("mousemove", mover, true);
 event.stopPropagation();
}

```

**Arrange the following subjects of VII Semester according to subject codes**

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.

Execute and check if you want, you can drag and drop the elements which are highlighted. After arranging,

**Arrange the following subjects of VII Semester according to subject codes**

1. OOMD
2. SA
3. WP
4. DM
5. JAVA
6. C#

