

# HTML NOTES

## Introduction:

HTML stands for Hyper Text Markup Language, which is the most used language for developing web pages.

HTML was created by Berners-Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995.

HTML 4.01 was a major version of HTML and it was published in late 1999.

Now we are having the HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

HTML5 is markup language used for structuring and presenting the content on the World Wide Web.

It is the 5th and final major HTML version that is a World Wide Web Consortium (W3C) recommendation .

The below basic important tags at one place:

Tag 1:

`<!DOCTYPE html>`

- Represents the Document type is html and tells a browser that it's reading html.

Tag 2:

`<html> </html>`

- That tells the browser to read the html.

Tag 3:

**<head> </head>**

- Header section, won't appear in the webpage.

Tag 4 :

**<title>Login Page</title>**

- Title of your web page.

Tag 5 :

**- <meta charset="ISO-8859-1">**

Tag 6 :

**<body> </body>**

- After the head tag we are going to proceed to work on the actual page.  
(all the content we are writing here only)

Tag 7 :

**<H1> </H1>**

- Header tags will give more than 50% of the bigger value.
- Here we have h1 to h6 tags.

Tag 8 :

**<P> </p>**

- Paragraph

Tag 9 :

**<!-- <content> -- >**

- Comments tag

- ( ctrl + shift + C in eclipse
- Ctrl + / in VS Code)

### Tag 10 :

**<label>Username</label>**

- What you type and you will get the same.

### Tag 11 :

**<br> </br>**

- It will break the lines.

### Tag 12 :

**<input type="text">**

- It will create a Text box for us and we can type and see.

### Tag 13 :

**<input type="Password">**

- It will create a text box and we cannot see the text directly.

### Tag 14 :

**<input type="submit" value="log in">**

- It will create a submit button, we can change the value if you want based on your requirement.

### Tag 15 :

**<strong>**

- Hello hemanth, How are you, how is everything going !!</strong>

**Tag 16 :**

**<em></em>**

**Tag 17 :**

**<a href="https://www.google.com" Google </a>**

**<a href="https://www.facebook.com/login.php"> Facebook</a>**

- The anchor tag and we can use it to navigate any links from your page.

**Tag 18 : <img src =""></img>**

**Tag 19 : <Ol> <li> </li></Ol>**

- Example :

**<ol>**

**<li>Srikanth</li>**

**<li>Hemanth</li>**

**<li>manu</li>**

**</ol>**

**Tag 20 : <ul><li></li> </ul>**

**Example :**

**<ul>**

**<li>Srikanth</li>**

**<li>Hemanth</li>**

**<li>manu</li>**

**</ul>**

**Tag 21 : <Select> </select>**

**Example :**

**<select>**

**<option value="India">India</option>**

**<option value="USA">USA</option>**

**<option value="UK">UK</option>**

`</select>`

## Tag 22 :

`<table><th><tr><td></td></tr></th></table>`

Table

th --> Table Header

tr --> Table Row

td --> Table Description

## Tag 23 :

`<div></div>`

**=====end of tags=====**

# HelloWorld.html

`<!DOCTYPE html>`

`<html>`

`<head>`

`</head>`

`<body>`

`<h1>Hello World!</h1>`

`</body>`

`</html>`

# HTML Tags

HTML is a markup language and makes use of various tags to format the content.

These tags are enclosed within angle braces <Tag Name>.

Ex:

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

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

Except for a few tags, most of the tags have their corresponding closing tags.

For example <html> has its closing tag </html> and <body> tag has its closing tag </body> tag etc.

Above example of HTML document uses following tags:

Tag	Description
<!DOCTYPE...>	This tag defines the document type and HTML version.
<html>	This tag encloses the complete HTML document and mainly comprises document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.

<code>&lt;head&gt;</code>	This tag represents the document's header which can keep other HTML tags like <code>&lt;title&gt;</code> , <code>&lt;link&gt;</code> etc.
<code>&lt;title&gt;</code>	The <code>&lt;title&gt;</code> tag is used inside the <code>&lt;head&gt;</code> tag to mention the document title.
<code>&lt;body&gt;</code>	This tag represents the document's body which keeps other HTML tags like <code>&lt;h1&gt;</code> , <code>&lt;div&gt;</code> , <code>&lt;p&gt;</code> etc.
<code>&lt;h1&gt;</code>	This tag represents the heading.
<code>&lt;p&gt;</code>	This tag represents a paragraph.

## HTML Document Structure

Document declaration tag

```
<html>
```

```
<head>
```

```
  header related tags
```

```
</head>
```

```
<body>
```

```
  body related tags
```

```
</body>
```

```
</html>
```

## The `<!DOCTYPE>` Declaration

The `<!DOCTYPE>` declaration tag is used by the web browser to understand the version of the HTML used in the document.

Current version of HTML is 5 and it makes use of the following declaration:

```
<!DOCTYPE html>
```

## Heading Tags :

Any Description starts with a heading.

You can use different sizes for your headings.

HTML also has six levels of headings, which use the elements

`<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`.

`<h1>` tag is large size and `<h6>` is small size.

## Example :

```
<!DOCTYPE html>
<html>

<head>
  <title>Heading Example</title>
</head>

<body>
  <h1>This is heading 1</h1>
  <h2>This is heading 2</h2>
  <h3>This is heading 3</h3>
  <h4>This is heading 4</h4>
  <h5>This is heading 5</h5>
  <h6>This is heading 6</h6>
</body>

</html>
```



# OUTPUT:

This will produce following result:

## This is heading 1

### This is heading 2

#### This is heading 3

##### This is heading 4

##### This is heading 5

###### This is heading 6

## Paragraph Tag

The `<p>` tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening `<p>` and a closing `</p>` tags.

```
<!DOCTYPE html>
<html>
<head>
  <title>Paragraph tag Example</title>
</head>
<body>
  <p>This is the first paragraph of text.</p>
  <p>This is a second paragraph of text.</p>
</body>
</html>
```

Output :

This is the first paragraph of text.

This is a second paragraph of text.

# Line Break Tag

Whenever you use the `<br>` element, anything following it starts from the next line.  
This will produce following result:

```
<!DOCTYPE html>
<html>
<head>
  <title>Line Break tag Example</title>
</head>
<body>
  <p>Hello<br />
    You delivered your assignment ontime Thanks<br />
    Mahnaz</p>
</body>
</html>
```

Output :  
Hello  
You delivered your assignment ontime Thank  
Mahnaz

# Centering Content

You can use `<center>` tag to put any content in the center of the page or any table cell.

```
<!DOCTYPE html>
<html>

<head>
  <title>Centring Content Example</title>
</head>
<body>
  <p>This text is not in the center.</p>
  <center>
    <p>This text is in the center.</p>
  </center>
</body>
</html>
```

Output :  
This will produce following result:  
This text is not in the center.

This text is in the center.

## Horizontal Lines

Horizontal lines are used to visually break up sections of a document.

The `<hr>` tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

```
<!DOCTYPE html>
<html>
<head>
  <title>Horizontal Line Example</title>
</head>
<body>
  <p>This is paragraph one and should be on top</p>
  <hr />
  <p>This is paragraph two and should be at bottom</p>
</body>
</html>
```

Output :

This will produce following result:

This is paragraph one and should be on top

## Preserve Formatting

Sometimes you want your text to follow the exact format of how it is written in the HTML document. In those cases, you can use the preformatted tag `<pre>`.

Any text between the opening `<pre>` tag and the closing `</pre>` tag will preserve the formatting of the source document.

```
<!DOCTYPE html>
<html>
<head>
  <title>Preserve Formatting Example</title>
```

```
</head>
<body>
  <pre>
functiontestFunction( strText ){
alert (strText)
}
</pre>
</body>
</html>
```

Output :

This will produce following result:

```
functiontestFunction( strText ){
alert (strText)
}
```

## Non breaking Spaces

Suppose you want to use the phrase "10 Jan 2000."

Here you would not want a browser to split the "10,Jan" and "2000" across two lines:

An example of this technique appears in the movie "10 Jan 2000."

In cases where you do not want the client browser to break text, you should use a non breaking space entity `&nbsp;` instead of a normal space.

```
<!DOCTYPE html>
<html>

<head>
  <title>Nonbreaking Spaces Example</title>
</head>

<body>
  <p>An example of this technique appears in the movie "12&nbsp;Angry&nbsp;Men."</p>
</body>

</html>
```

# HTML Elements

An HTML element is defined by a starting tag.

If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags:forward slash as shown below with few tags:

Start Tag	Content	End Tag
<code>&lt;p&gt;</code>	This is paragraph content.	<code>&lt;/p&gt;</code>
<code>&lt;h1&gt;</code>	This is heading content.	<code>&lt;/h1&gt;</code>
<code>&lt;div&gt;</code>	This is division content.	<code>&lt;/div&gt;</code>
<code>&lt;br /&gt;</code>		

So here `<p>...</p>` is an HTML element, `<h1>...</h1>` is another HTML element. There are some HTML elements which don't need to be closed,

such as `<img src="">`, `<hr>` and `<br>` elements. These are known as void elements.

## Nested HTML Elements

It is very much allowed to keep one HTML element inside another HTML element:

```
<!DOCTYPE html>
<html>
<head>
  <title>Nested Elements Example</title>
</head>
<body>
  <h1>This is <i>italic</i> heading</h1>
  <p>This is <u> underlined </u> paragraph</p>
</body>
</html>
```

Output :

This will display following result:

This is *italic* heading

This is underlined paragraph

## HTML Attributes

- An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag.
- All attributes are made up of two parts: a name and a value:
  - The name is the property you want to set.
- For example, the paragraph `<p>` element in the example carries an attribute whose name is `align`, which you can use to indicate the alignment of paragraphs on the page.
- The value is what you want the value of the property to be set and always put within quotations.
- The below example shows three possible values of `align` attribute: left, center and right.

Attribute names and attribute values are case-insensitive.

```
<!DOCTYPE html>
<html>
<head>
  <title>Align Attribute Example</title>
</head>
<body>
```

```
<p align="left">This is left aligned</p>
<p align="center">This is center aligned</p>
<p align="right">This is right aligned</p>
</body>
</html>
```

Output :

This will display following result:

This is left aligned

This is center aligned

This is right aligned

# Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are:

- id
- title
- class
- style

## The id Attribute

The id attribute of an HTML tag can be used to uniquely identify any element within an HTML page.

There are two primary reasons that you might want to use an id attribute on an element:

- If an element carries an id attribute as a unique identifier it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

```
<p id="abc">This para explains what is HTML</p>
```

`<p id="lmn">This para explains what is Cascading Style Sheet</p>`

## The title Attribute

The title attribute gives a suggested title for the element.

The syntax for the title attribute is similar as explained for id attribute:

The behavior of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip when the cursor comes over the element.

```
<!DOCTYPE html>
<html>
<head>
  <title>The title Attribute Example</title>
</head>
<body>
  <h3 title="Hello HTML!">Titled Heading Tag Example</h3>
</body>
</html>
```

Output :

This will produce following result:

### Titled Heading Tag Example

Now try to bring your cursor over "Titled Heading Tag Example" and you will see that whatever title you used in your code is coming out as a tooltip of the cursor.

## The class Attribute

The class attribute is used to associate an element with a style sheet, and specifies the class of element.



You will learn more about the use of the class attribute when you will learn Cascading Style Sheets (CSS). So for now you can avoid it.

The value of the attribute may also be a space-separated list of class names.

For example:

```
class="className1 className2 className3"
```

## The style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

```
<!DOCTYPE html>
<html>
<head>
  <title>The style Attribute</title>
</head>
<body>
  <pstyle="font-family:arial; color:#FF0000;">Some text...</p>
</body>
</html>
```

Output :

This will produce following result:

Some text...

At this point of time, we are not learning CSS, so just let's proceed without bothering much about CSS.

Here you need to understand what are HTML attributes and how they can be used while formatting content.

# Internationalization Attributes

There are three internationalization attributes, which are available for most (although not all) XHTML elements.

- dir
- lang
- Generic

# The dir Attribute

The `dir` attribute allows you to indicate to the browser the direction in which the text should flow.

The `dir` attribute can take one of two values, as you can see in the table that follows:

Value	Meaning
ltr	Left to right (the default value)
rtl	Right to left

Example :

```
<!DOCTYPE html>
<html dir="rtl">
  <head>
    <title>Display Directions</title>
  </head>
  <body>
    This is how to render right-to-left directed text.
  </body>
</html>
```

- When *dir* attribute is used within the `<html>` tag, it determines how text will be presented within the entire document.
- When used within another tag, it controls the text's direction for just the content of that tag.

# The lang Attribute

The lang attribute allows you to indicate the main language used in a document,

but this attribute was kept in HTML only for backwards compatibility with earlier versions of HTML.

The values of the *lang* attribute are ISO-639 standard two-character language

codes. Check [HTML Language Codes: ISO 639](#) for a complete list of language codes.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>English Language Page</title>
  </head>
  <body>
    This page is using English Language
  </body>
</html>
```

## Generic Attributes

Here's a table of some other attributes that are readily usable with many of the HTML tags.

Attribute	Options	Function
align	right, left, center	Horizontally aligns tags
V align	top, middle, bottom	Vertically aligns tags within an HTML element.

<b>bbgcolor</b>	<b>numeric, hexadecimal, RGB values</b>	<b>Places a background color behind an element</b>
<b>background</b>	<b>URL</b>	<b>Places a background image behind an element</b>
<b>id</b>	<b>User Defined</b>	<b>Names an element for use with Cascading Style Sheets.</b>
<b>class</b>	<b>User Defined</b>	<b>Classifies an element for use with Cascading Style Sheets.</b>
<b>width</b>	<b>Numeric Value</b>	<b>Specifies the width of tables, images, or table cells.</b>
<b>height</b>	<b>Numeric Value</b>	<b>Specifies the height of tables, images, or table cells.</b>
<b>title</b>	<b>User Defined</b>	<b>"Pop-up" title of the elements.</b>

```

<!DOCTYPE html>
<html>
  <head>
    <title>HTML Alignment</title>
  </head>
  <body>
    <h1>HTML Alignments</h1>
    <table border="1" width="500">

```

```

<tr>
  <th>Name</th>
  <th>ID</th>
  <th>Age</th>
</tr>
<tr style="height: 50px;">
  <td valign="top">Nithin</td>
  <td valign="center">1</td>
  <td valign="bottom">22</td>
</tr>
<tr style="height: 50px;">
  <td valign="bottom">Dilip</td>
  <td valign="center">2</td>
  <td valign="top">23</td>
</tr>
</table>
</body>
</html>

```

## HTML Formatting

If you use a word processor, you must be familiar with the ability to make text bold, italicized, or underlined; these are just three of the ten options available to indicate how text can appear in HTML and XHTML.

### Bold Text

- Anything that appears within <b>...</b> element, is displayed in bold as shown below:
- 

```

<!DOCTYPE html>
<html>

```

```
<head>
  <title>Bold Text Example</title>
</head>
<body>
  <p>The following word uses a <b>bold</b> typeface.</p>
</body>
</html>
```

This will produce following result:  
The following word uses a bold typeface.

## Italic Text

Anything that appears within `<i>...</i>` element is displayed in italicized as shown below:

## Underlined Text

Anything that appears within `<u>...</u>` element, is displayed with underline as shown below:

## Strike Text

Anything that appears within `<strike>...</strike>` element is displayed with strikethrough, which is a thin line through the text as shown below:

## Superscript Text

The content of a `<sup>...</sup>` element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

## Subscript Text

The content of a `<sub>...</sub>` element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

# Inserted Text

*Anything that appears within `<ins>...</ins>` element is displayed as inserted text with underline.*

# Deleted Text

Anything that appears within `<del>...</del>` element, is displayed as deleted text with striking.

# Larger Text

The content of the `<big>...</big>` element is displayed one font size larger than the rest of the text surrounding it as shown below:

# Smaller Text

The content of the `<small>...</small>` element is displayed one font size smaller than the rest of the text surrounding it as shown below:

```
<!DOCTYPE html>
<html>

<head>
  <title> Text Example</title>
</head>

<body>
  <p>The following word uses a <i>italicized</i> typeface.</p>
  <p>The following word uses a <u>underlined</u> typeface.</p>
  <p>The following word uses a <strike>striketrough</strike> typeface.</p>
  <p>The following word uses a <sup>superscript</sup> typeface.</p>
  <p>The following word uses a <sub>subscript</sub> typeface.</p>
  <p>I want to drink <del>cola</del><ins>wine</ins></p>
  <p>I want to drink <del>cola</del><ins>wine</ins></p>
  <p>The following word uses a <big>big</big> typeface.</p>
  <p>The following word uses a <small>small</small> typeface.</p>
</body>

</html>
```

Output 👍

The following word uses a *italicized* typeface.

The following word uses a underlined typeface.

The following word uses a ~~striketrough~~ typeface.

The following word uses a <sup>superscript</sup> typeface.

The following word uses a <sub>subscript</sub> typeface.

I want to drink colawine

I want to drink colawine

The following word uses a big typeface.

The following word uses a small typeface.

## Grouping Content

The <div> and <span> elements allow you to group together several elements to create sections or subsections of a page.

```
<!DOCTYPE html>
<html>

<head>
  <title>Div Tag Example</title>
</head>

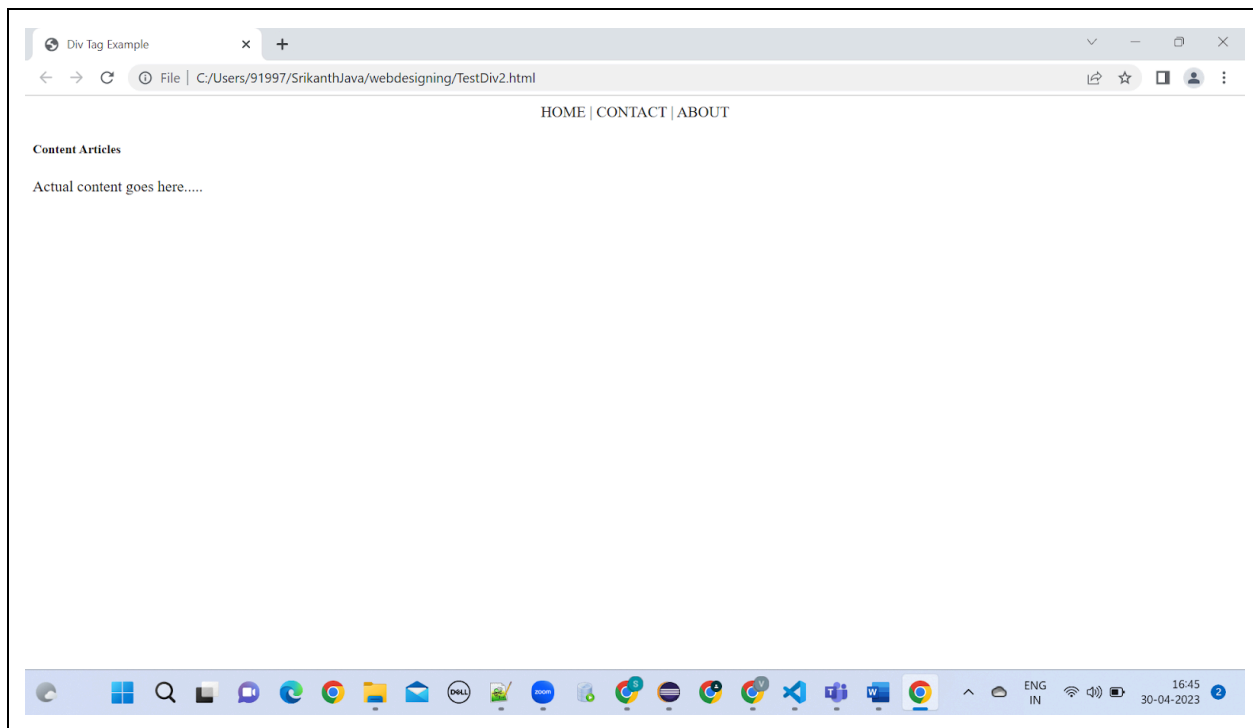
<body>
  <div id="menu" align="middle">
    <a href=" /index.htm">HOME</a> |
    <a href=" /about/contact_us.htm">CONTACT</a> |
    <a href=" /about/index.htm">ABOUT</a>
  </div>

  <div id="content" align="left" bgcolor="white">
    <h5>Content Articles</h5>
    <p>Actual content goes here.....</p>
  </div>
```



```
</body>
```

```
</html>
```



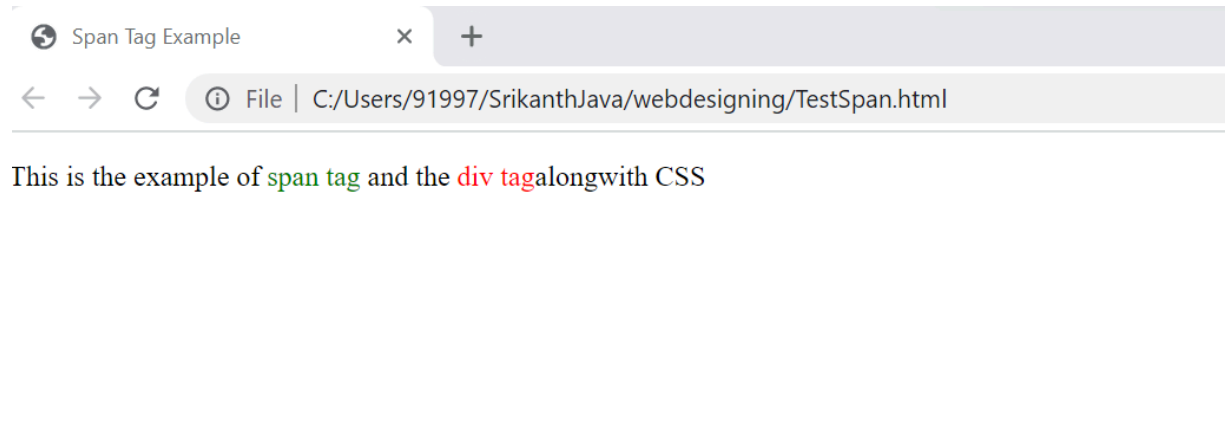
The `<span>` element, on the other hand, can be used to group inline elements only.

So, if you have a part of a sentence or paragraph which you want to group together, you could use the `<span>` element as follows.

```
<!DOCTYPE html>
<html>
<head>
  <title>Span Tag Example</title>
</head>
<body>
  <p>This is the example of <span style="color:green">span tag</span> and the <span style="color:red">div
    tag</span>along with CSS</p>
```

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

OUTPUT :



## HTML Phrase Tags

The phrase tags have been designed for specific purposes, though they are displayed in a similar way as other basic tags like `<b>`, `<i>`, `<pre>`, and `<tt>`..

## Emphasized Text

Anything that appears within `<em>...</em>` element is displayed as emphasized text.

## Marked Text

Anything that appears with-in `<mark>...</mark>` element, is displayed as marked with yellow ink.

## Strong Text

Anything that appears within `<strong>...</strong>` element is displayed as important text.

## Text Abbreviation

You can abbreviate a text by putting it inside opening `<abbr>` and closing `</abbr>` tags.

If present, the title attribute must contain this full description and nothing else.

## Quoting Text

When you want to quote a passage from another source, you should put it in between `<blockquote>...</blockquote>` tags.

Text inside a `<blockquote>` element is usually indented from the left and right edges of the surrounding text, and sometimes uses an italicized font.

## Short Quotations

The `<q>...</q>` element is used when you want to add a double quote within a sentence.

## Keyboard Text

When you are talking about computers, if you want to tell a reader to enter some text, you can use the `<kbd>...</kbd>` element to indicate what should be typed in, as in this example.

## Samp Tag

The `<samp>...</samp>` element indicates sample output from a program, and script etc.

Again, it is mainly used when documenting programming or coding concepts.

## Address Text

The `<address>...</address>` element is used to contain any address.

```
<!DOCTYPE html>
<html>
<head>
  <title>Emphasized Text Example</title>
</head>

<body>
  <!-- emphasized -->
  <p>The following word uses a <em>emphasized</em> typeface.</p>
```

```

<!--mark -->
<p>The following word has been <mark>marked</mark> with yellow</p>

<!--blockquote -->
<p>The following description of XHTML is taken from the W3C Web site:</p>
<blockquote>XHTML 1.0 is the W3C's first Recommendation for XHTML, following on from earlier work on
HTML 4.01, HTML
    4.0, HTML 3.2 and HTML 2.0.</blockquote>

<!-- Short Quotations -->
<p>Amit is in Spain, <q>I think I am wrong</q>.</p>

<!--Keyboard Text-->
<p>Regular text. <kbd>This is inside kbd element</kbd> Regular text.</p>

<!--Samp Tag-->
<p>Result produced by the program is <samp>Hello World!</samp></p>

<!-- address -->
<address>388A, Road No 22, Jubilee Hills - Hyderabad</address>

</body>
</html>

```

## Output :



# HTML Comments

Comment is a piece of code which is ignored by any web browser.

It is a good practice to add comments into your HTML code, especially in complex documents, to indicate sections of a document and any other notes to anyone looking at the code.

Comments help you and others understand your code and increase code readability.

HTML comments are placed in between `<!-- ... -->` tags. So any content placed with-in `<!-- ... -->` tags will be treated as comment and will be completely ignored by the browser.

## Valid vs Invalid Comments

Comments do not nest which means a comment cannot be put inside another comment.

Second, the double-dash sequence `--` may not appear inside a comment except as part of the closing `-->` tag.

You must also make sure that there are no spaces in the start-of-comment string.

Here the given comment is a valid comment and will be wiped off by the browser.

## Multiline Comments

So far we have seen single line comments, but HTML supports multi-line comments as well.

You can comment multiple lines by the special beginning tag `<!--` and ending tag `-->` placed before the first line and end of the last line as shown in the given example below.

```
<!DOCTYPE html>
<html>

<head><!-- Document Header Starts -->
  <title>This is document title</title>
</head><!-- Document Header Ends -->

<body>
  <p>Document content goes here.....</p>

  <!-- This is valid comment -->
  <p>Document content goes here.....</p>

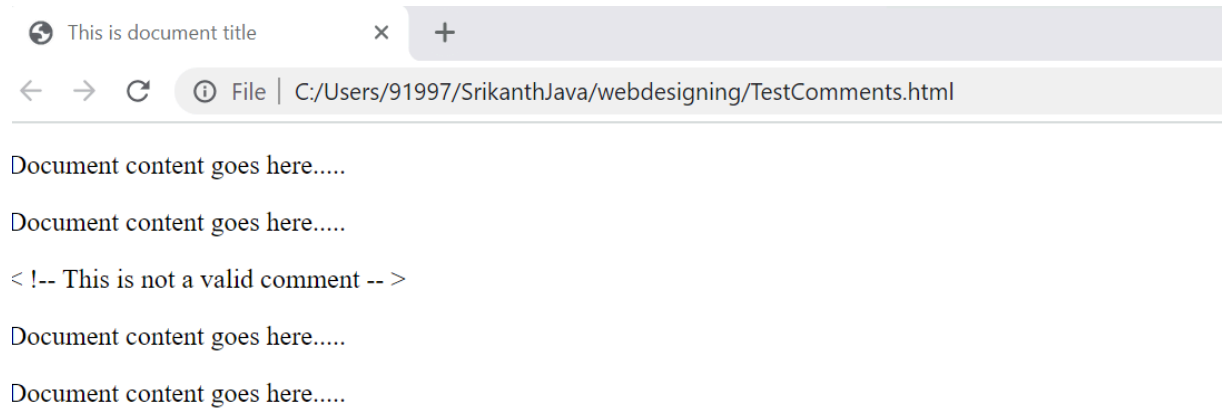
  <!-- This is not a valid comment -- >
  <p>Document content goes here.....</p>
```

```
<!--  
    This is a multiline comment and it can  
    span through as many as lines you like.  
-->  
<p>Document content goes here.....</p>
```

</body>

</html>

**Output :**



# HTML Images

Images are very important to beautify as well as to depict many complex concepts in a simple way on your web page.

This tutorial will take you through simple steps to use images in your web pages.

## Insert Image

You can insert any image in your web page by using `<img>` tag.

Following is the simple syntax to use this tag.

```

```

The `<img>` tag is an empty tag, which means that it can contain only a list of attributes and it has no closing tag.

You can use a PNG, JPEG or GIF image file based on your comfort but make sure you specify the correct image file name in the `src` attribute.

Image names are always case sensitive.

The `alt` attribute is a non-mandatory attribute which specifies an alternate text for an image, if the image cannot be displayed.

## Set Image Width/Height

You can set image width and height based on your requirement using `width` and `height` attributes.

You can specify width and height of the image in terms of either pixels or percentage of its actual size.

## Set Image Border

By default the image will have a border around it, you can specify border thickness in terms of pixels using the `border` attribute. A thickness of 0 means, no border around the picture.

## Set Image Alignment

By default the image will align at the left side of the page, but you can use `align` attribute to set it in the center or right.

```
<!DOCTYPE html>
<html>

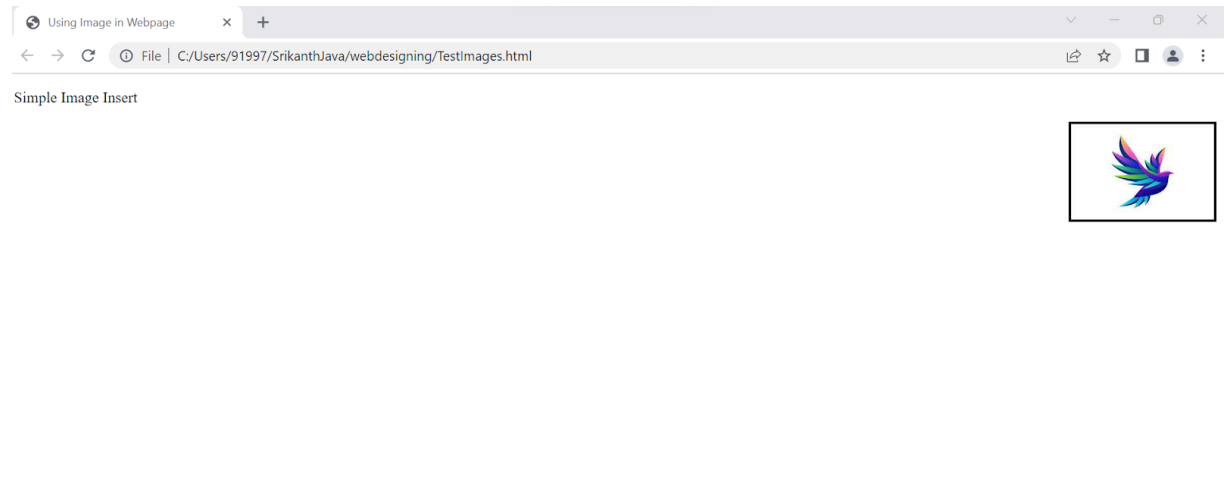
<head>
  <title>Using Image in Webpage</title>
</head>

<body>
  <p>Simple Image Insert</p>
```

```

</body>
</html>
```

#### Output:



## HTML Tables

The HTML tables allow to arrange data like text, images, links, etc. into rows and columns of cells.

The HTML tables are created using the `<table>` tag in which the

`<tr>` tag is used to create table rows and

`<td>` tag is used to create data cells.

## Table Heading

Table heading can be defined using `<th>` tag.

This tag will be put to replace `<td>` tag, which is used to represent actual data cell.

Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element in any row.



# Cellpadding and Cellspacing Attributes

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells.

The cellspacing attribute defines the **width of the border**, while

cellpadding represents the **distance between cell borders and the content** within a cell.

# Colspan and Rowspan Attributes

You will use the colspan attribute if you want **to merge two or more columns** into a single column.

Similarly you will use rowspan if you want to merge **two or more rows**.

# Tables Backgrounds

You can set table background using one of the following two ways:

- bgcolor attribute - You can set background color for the whole table or just for one cell.
- background attribute - You can set a background image for the whole table or just for one cell.

You can also set border color using bordercolor attribute.

# Table Caption

The caption tag will serve as a title or explanation for the table and it shows up at the top of the table.

This tag is deprecated in newer versions of HTML/XHTML.

# Table Header, Body, and Footer

Tables can be divided into three portions: a header, a body, and a foot.

The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are:

- `<thead>` - to create a separate table header.
- `<tbody>` - to indicate the main body of the table.
- `<tfoot>` - to create a separate table footer.

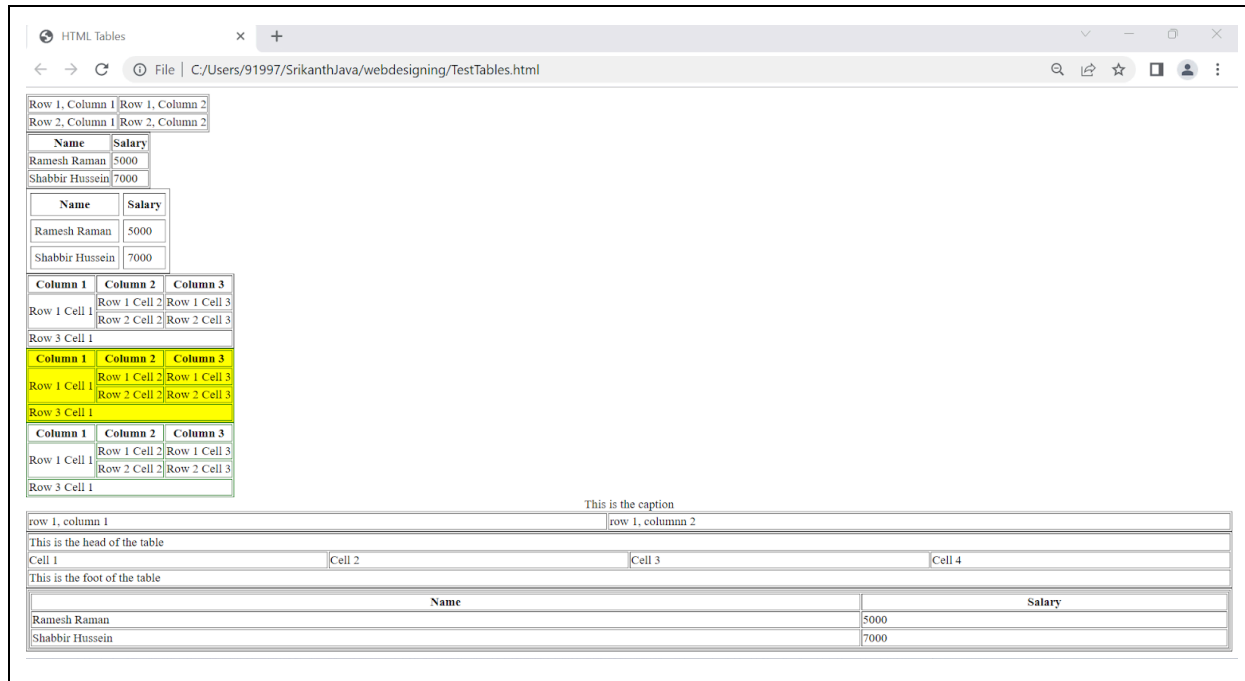
A table may contain several `<tbody>` elements to indicate different *pages* or groups of data.

But it is notable that `<thead>` and `<tfoot>` tags should appear before `<tbody>`

## Nested Tables

You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag `<td>`.

Following is the example of using another table and other tags inside a table cell.



# HTML Lists

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain:

- **<ul>** - An unordered list. This will list items using plain bullets.
- **<ol>** - An ordered list. This will use different schemes of numbers to list your items.
- **<dl>** - A definition list. This arranges your items in the same way as they are arranged in a dictionary.

# HTML Unordered Lists

An unordered list is a collection of related items that have no special order or sequence.

This list is created by using the HTML `<ul>` tag.

Each item in the list is marked with a bullet.

## The type Attribute

You can use type attribute for `<ul>` tag to specify the type of bullet you like.

By default it is a disc. Following are the possible options:

```
<ul type="square">
```

```
<ul type="disc">
```

```
<ul type="circle">
```

Following is an example where we used `<ul type="square">`

## HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted then an HTML ordered list will be used.

This list is created by using `<ol>` tag.

The numbering starts at one and is incremented by one for each successive ordered list element tagged with `<li>`.

## The type Attribute

You can use type attribute for `<ol>` tag to specify the type of numbering you like. By default it is a number. Following are the possible options:

```
<ol type="1"> - Default-Case Numerals.
```

```
<ol type="I"> - Upper-Case Numerals.
```

```
<ol type="i"> - Lower-Case Numerals.
```

`<ol type="a">` - Lower-Case Letters.

`<ol type="A">` - Upper-Case Letters.

Following is an example where we used `<ol type="1">`

## The start Attribute

You can use start attribute for `<ol>` tag to specify the starting point of numbering you need.

Following are the possible options:

`<ol type="1"start="4">` - Numerals starts with 4.

`<ol type="I"start="4">` - Numerals starts with IV.

`<ol type="i"start="4">` - Numerals starts with iv.

`<ol type="a"start="4">` - Letters starts with d.

`<ol type="A"start="4">` - Letters starts with D.

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML Unordered List</title>
</head>

<body>
  <ul type="square">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ul>
  <ul type="disc">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ul>
  <ul type="circle">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
  </ul>
</body>
</html>
```

```
</li>Radish</li>
</ul>

<ol>
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>

<ol type="1">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>

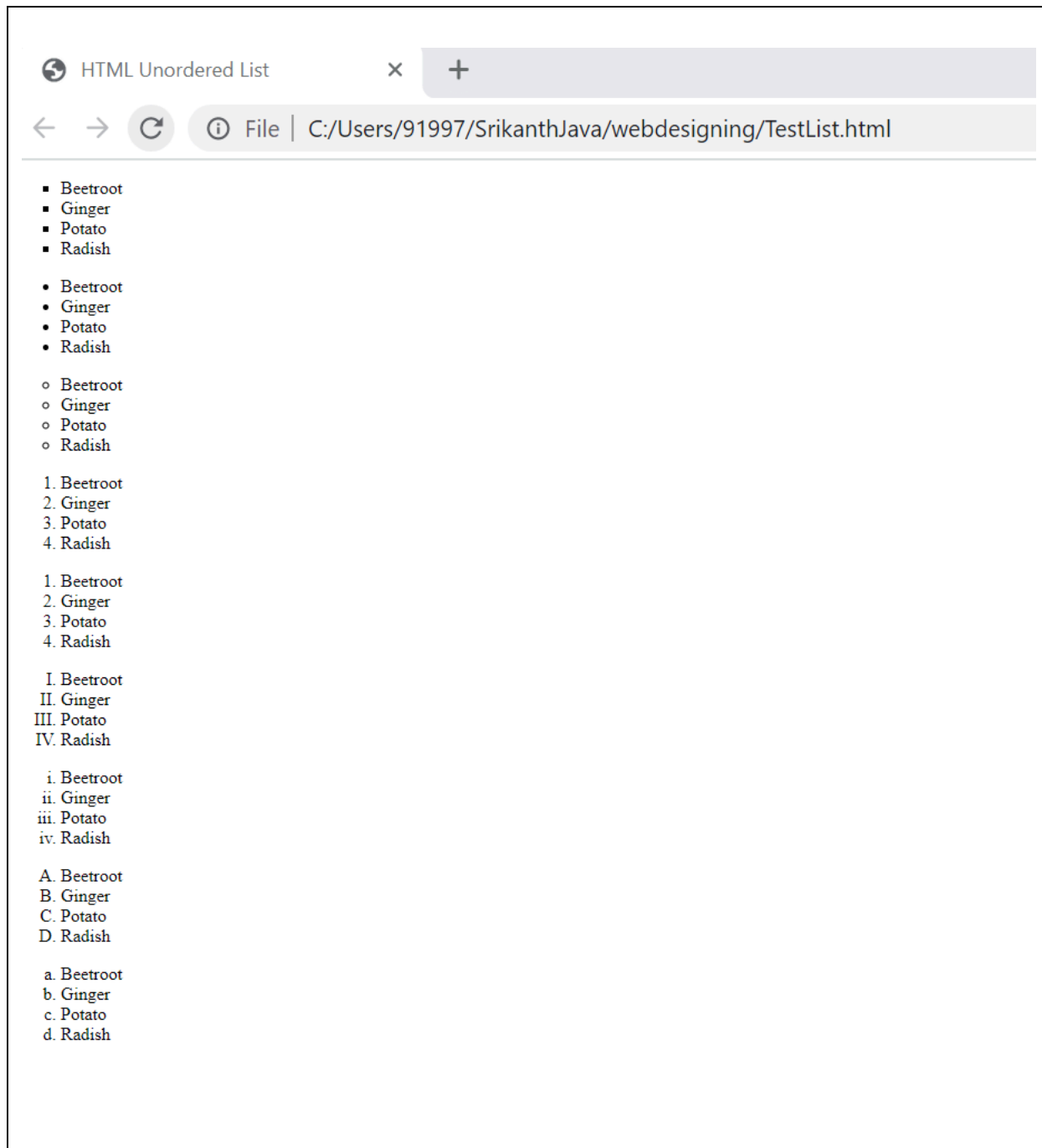
<ol type="I">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>

<ol type="i">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>

<ol type="A">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>

<ol type="a">
  <li>Beetroot</li>
  <li>Ginger</li>
  <li>Potato</li>
  <li>Radish</li>
</ol>
</body>
</html>
```

**Output :**



## HTML Definition Lists

HTML and XHTML support a list style which is called definition lists where entries are listed like in a dictionary or encyclopedia.

The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

The Definition List makes use of the following three tags.

- `<dl>` - Defines the start of the list
- `<dt>` - A term
- `<dd>` - Term definition
- `</dl>` - Defines the end of the list

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML Definition List</title>
</head>
<body>
  <dl>
    <dt><b>HTML</b></dt>
    <dd>This stands for Hyper Text Markup Language</dd>
    <dt><b>HTTP</b></dt>
    <dd>This stands for Hyper Text Transfer Protocol</dd>
  </dl>
</body>
</html>
```

# HTML Text Links

A webpage can contain various links that take you directly to other pages and even specific parts of a given page.

These links are known as hyperlinks.

Hyperlinks allow visitors to navigate between Web sites by clicking on words and images.

Thus you can create hyperlinks using text or images available on a webpage.



## Linking Documents

A link is specified using HTML tag <a>.

This tag is called anchor tag and anything between the opening <a> tag and the closing </a> tag becomes part of the link and a user can click that part to reach the linked document.

Following is the simple syntax to use <a> tag.

```
<a href="Document URL" ... attributes-list>Link Text</a>
```

Let's try following example which links <http://www.kosmiktech.com> at your page:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hyperlink Example</title>
</head>
<body>
  <p>Click following link</p>
  <a href="https://www.kosmiktechnologies.com/" target="_self">Kosmik</a>
</body>
</html>
```

## The target Attribute

We have used the target attribute in our previous example.

This attribute is used to specify the location where the linked document is opened.

Following are possible options:

Option	Description
<code>_blank</code>	Opens the linked document in a new window or tab.
<code>_self</code>	Opens the linked document in the same frame.

Try following example to understand basic difference in few options given for target attribute.

## Linking to a Page Section

You can create a link to a particular section of a given webpage by using name attribute.

This is a two step process.

First create a link to the place where you want to reach with-in a webpage and name it using `<a...>` tag as follows:

```
<h1>HTML Text Links <a name="top"></a></h1>
```

Second step is to create a hyperlink to link the document and place where you want to reach:

```
<a href="/html/html_text_links.htm#top">Go to the Top</a>
```

This will produce the following link, where you can click on the link generated Go to the Top to reach to the top of the HTML Text Link tutorial.

## Setting Link Colors

You can set colors of your links, active links and visited links using `link`, `alink` and `vlink` attributes of `<body>` tag.

```
<!DOCTYPE html>
<html>
<head>
  <title>Hyperlink Example</title>
```

```
</head>

<body>
  <p>Click following link</p>
  <a href="https://www.kosmiktechnologies.com/" target="_self">Kosmik</a>

  <p>Click any of the following links</p>
  <a href="https://www.kosmiktechnologies.com/" target="_blank">Opens in New</a> |
  <a href="https://www.kosmiktechnologies.com/" target="_self">Opens in Self</a> |
  <a href="https://www.kosmiktechnologies.com/" target="_parent">Opens in Parent</a> |
  <a href="https://www.kosmiktechnologies.com/" target="_top">Opens in Body</a>
</body>
</html>
```

## HTML Image Links

We have seen how to create hypertext links using text and we also learnt how to use images in our web pages.

Now we will learn how to use images to create hyperlinks.

It's simple to use an image as a hyperlink.

We just need to use an image inside hyperlink at the place of text as shown below:

## HTML Email Links

Its not difficult to put an HTML email link on your webpage but it can cause unnecessary spamming problems for your email account.

There are people who can run programs to harvest these types of emails and later use them for spamming in various ways.

You can have other options to facilitate people to send you emails.

One option could be to use HTML forms to collect user data and then use PHP or ASP script to send an email.

# HTML Email Tag

HTML <a> tag provides you the option to specify an email address to send an email.

While using <a> tag as an email tag, you will use `mailto:email address` along with *href* attribute. Following is the syntax of using `mailto` instead of using `http`.

```
<a href="mailto:abc@example.com">Send Email</a>
```

## Default Settings

You can specify a default *email subject* and *email body* along with your email address.

Following is the example to use default subject and body.

```
<a href="mailto:abc@example.com?subject=Feedback&body=Message">
```

Send Feedback

```
</a>
```

This code will generate the following link which you can use to send email.

```
<!DOCTYPE html>
<html>

<head>
  <title>Image Hyperlink Example</title>
</head>

<body>
  <p>Click following Images for Kosmik</p>
  <br>
  <a href="https://www.kosmiktechnologies.com" target="_self">
    
  </a>
  <br>
  <!--Email tag -->
  <a href="mailto:srkntjva@gmail.com">Send Email</a>
  <br>
```

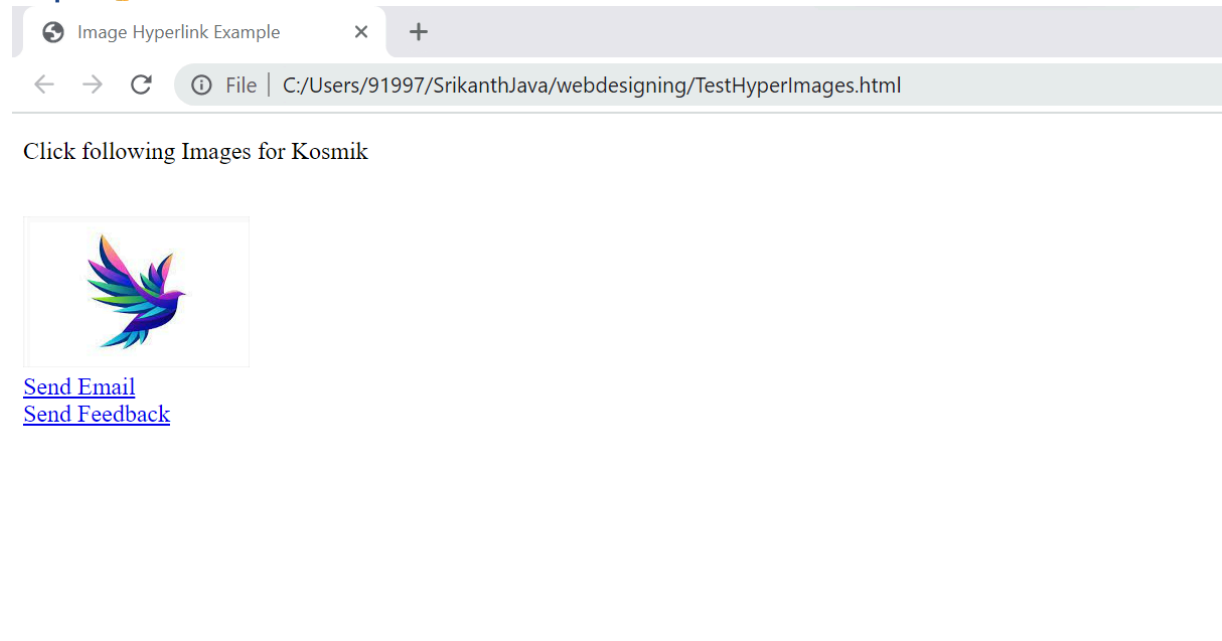
```
<!--Default settings -->
```

```
<a href="mailto:srkntjva@gmail.com?subject=Feedback&body=Message">Send Feedback  
</a>
```

```
</body>
```

```
</html>
```

## Output 👍



## HTML Iframes

You can define an inline frame with HTML tag `<iframe>`.

The `<iframe>` tag is not somehow related to `<frameset>` tag, instead, it can appear anywhere in your document.

The `<iframe>` tag defines a rectangular region within the document in which the browser can display a separate document, including scrollbars and borders.

The `src` attribute is used to specify the URL of the document that occupies the inline frame.

Following is the example to show how to use the `<iframe>`:

### The `<iframe>` Tag Attributes

Most of the attributes of the <iframe> tag, including *name*, *class*, *frameborder*, *id*, *longdesc*, *marginheight*, *marginwidth*, *name*, *scrolling*, *style*, and *title* behave exactly like the corresponding attributes for the <frame> tag.

Attribute	Description
src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src="/html/top_frame.htm" will load an HTML file available in html directory.
name	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).

## HTML Blocks

All the HTML elements can be categorized into two categories

(a) Block Level Elements (b) Inline Elements

## Block Elements

Block elements appear on the screen as if they have a line break before and after them.

For example the <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <ul>, <ol>, <dl>, <pre>, <hr />, <blockquote>, and <address> elements are all block level elements.

They all start on their own new line, and anything that follows them appears on its own new line.

## Inline Elements

Inline elements, on the other hand, can appear within sentences and do not have to appear on a new line of their own.

The `<b>`, `<i>`, `<u>`, `<em>`, `<strong>`, `<sup>`, `<sub>`, `<big>`, `<small>`, `<li>`, `<ins>`, `<del>`, `<code>`, `<cite>`, `<dfn>`, `<kbd>`, and `<var>` elements are all inline elements.

## Grouping HTML Elements

There are two important tags which we use very frequently to group various other HTML tags (i) `<div>` tag and (ii) `<span>` tag

### The `<div>` tag

This is the very important block level tag which plays a big role in grouping various other HTML tags and applying CSS on a group of elements. Even now `<div>` tag can be used to create a web page layout where we define different parts ( Left, Right, Top etc) of the page using `<div>` tag. This tag does not provide any visual change on the block but this has more meaning when it is used with CSS.

Following is a simple example of `<div>` tag. We will learn Cascading Style Sheet (CSS) in a separate chapter but we used it here to show the usage of `<div>` tag:

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML div Tag</title>
</head>

<body>
  <!-- First group of tags -->
  <div style="color:red">
    <h4>This is first group</h4>
    <p>Following is a list of vegetables</p>
    <ul>
```

```
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ul>
</div>

<!-- Second group of tags -->
<div style="color:green">
  <h4>This is second group</h4>
  <p>Following is a list of fruits</p>
  <ul>
    <li>Apple</li>
    <li>Banana</li>
    <li>Mango</li>
    <li>Strawberry</li>
  </ul>
</div>
</body>
</html>
```

## The <span> tag

The HTML <span> is an inline element and it can be used to group inline-elements in an HTML document.

This tag also does not provide any visual change on the block but has more meaning when it is used with CSS.

The difference between the <span> tag and the <div> tag is that the <span> tag is used with inline elements whereas the <div> tag is used with block-level elements.

Following is a simple example of <span> tag. We will learn Cascading Style Sheet (CSS) in a separate chapter but we used it here to show the usage of <span> tag:

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML span Tag</title>
</head>

<body>
```



```
<p>This is <span style="color:red">red</span> and this is <span style="color:green">green</span></p>

</body>

</html>
```

## HTML Backgrounds

HTML provides you following two good ways to decorate your webpage background.

- Html Background with Colors
- Html Background with Images

### Html Background with Colors

The bgcolor attribute is used to control the background of an HTML element, specifically page body and table backgrounds. Following is the syntax to use bgcolor attribute with any HTML tag.

```
<tagnamebgcolor="color_value"...>
```

This color\_value can be given in any of the following formats:

```
<!-- Format 1 - Use color name -->
```

```
<table bgcolor="lime">
```

```
<!-- Format 2 - Use hex value -->
```

```
<table bgcolor="#f1f1f1">
```

```
<!-- Format 3 - Use color value in RGB terms -->
```

```
<table bgcolor="rgb(0,0,120)">
```

Here are the examples to set background of an HTML tag:

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Background Colors</title>
</head>
<body>
  <!-- Format 1 - Use color name -->
  <table bgcolor="yellow" width="100%">
```

```
<tr>
  <td>
    This background is yellow
  </td>
</tr>
</table>

<!-- Format 2 - Use hex value -->
<table bgcolor="#6666FF" width="100%">
  <tr>
    <td>
      This background is sky blue
    </td>
  </tr>
</table>

<!-- Format 3 - Use color value in RGB terms -->
<table bgcolor="rgb(255,0,255)" width="100%">
  <tr>
    <td>
      This background is green
    </td>
  </tr>
</table>
</body>
</html>
```

## Html Background with Images

The `background` attribute can also be used to control the background of an HTML element, specifically page body and table backgrounds.

You can specify an image to set the background of your HTML page or table.

Following is the syntax to use background attributes with any HTML tag.

**Note:** The *background* attribute is deprecated and it is recommended to use Style Sheet for background setting.

```
<tagname background="Image URL" ...>
```

The most frequently used image formats are JPEG, GIF and PNG images.

Here are the examples to set background images of a table.

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML Background Images</title>
</head>
<body>
  <!-- Set table background -->
  <table background="images/Image3.jpg" width="100%" height="100">
    <tr>
      <td>
        This background is filled up with HTML image.
      </td>
    </tr>
  </table>

</body>
</html>
```

## Transparent Backgrounds

You might have seen many patterns or transparent backgrounds on various websites.

This simply can be achieved by using a patterned image or transparent image in the background.

It is suggested that while creating patterns or transparent GIF or PNG images, use the smallest dimensions possible even as small as 1x1 to avoid slow loading.

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML Background Images</title>
</head>
<body>
```

```
<!-- Set a table background using pattern -->
<table background="/images/Image3.jpg" width="100%" height="100">
  <tr>
    <td>
      This background is filled up with a pattern image.
    </td>
  </tr>
</table>

<!-- Another on table background using pattern -->
<table background="/images/Image3.jpg" width="100%" height="100">
  <tr>
    <td>
      This background is filled up with a pattern image.
    </td>
  </tr>
</table>
</body>
</html>
```

# HTML Colors

Colors are very important to give a good look and feel to your website.

You can specify colors on page level using <body> tag or you can set colors for individual tags using bgcolor attribute.

The <body> tag has following attributes which can be used to set different colors:

- **bgcolor** - sets a color for the background of the page.
- **text** - sets a color for the body text.
- **alink** - sets a color for active links or selected links.
- **link** - sets a color for linked text.
- **vlink** - sets a color for *visited links* - that is, for linked text that you have already clicked on.

## HTML Color Coding Methods

There are following three different methods to set colors in your web page:

- Color names - You can specify color names directly like green, blue or red.
- Hex codes - A six-digit code representing the amount of red, green, and blue that makes up the color.
- Color decimal or percentage values - This value is specified using the `rgb( )` property.

Now we will see these coloring schemes one by one.

## HTML Colors - Color Names

You can sepecify direct a color name to set text or background color.

W3C has listed 16 basic color names that will validate with an HTML validator but there are over 200 different color names supported by major browsers.

## W3C Standard 16 Colors

Here is the list of W3C Standard 16 Colors names and it is recommended to use them.

	Black		Gray		Silver		White
	Yellow		Lime		Aqua		Fuchsia
	Red		Green		Blue		Purple
	Maroon		Olive		Navy		Teal

Here are the s to set background of an HTML tag by color name:

## HTML Colors - Hex Codes

A hexadecimal is a 6 digit representation of a color.

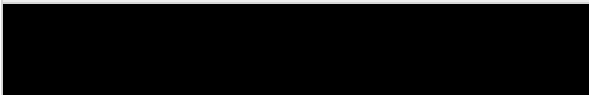








The first two digits(RR) represent a red value,

the next two are a green value(GG),

and the last are the blue value(BB).

A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Paint shop Pro or MS Paint.

Each hexadecimal code will be preceded by a pound or hash sign #. Following is a list of few colors using hexadecimal notation.

Color	Color HEX
	#000000
	#FF0000
	#00FF00
	#0000FF
	#FFFF00
	#00FFFF
	#FF00FF
	#C0C0C0
	#FFFFFF

Here are the s to set background of an HTML tag by color code in hexadecimal:

# HTML Fonts

Fonts play a very important role in making a website more user friendly and increasing content readability.

Font face and color depends entirely on the computer and browser that is being used to view your page but you can use HTML `<font>` tag to add style, size, and color to the text on your website.

The font tag has three attributes called size, color, and face to customize your fonts.

To change any of the font attributes at any time within your webpage, simply use the `<font>` tag. The text that follows will remain changed until you close with the `</font>` tag.

## Set Font Size

You can set content font size using size attribute.

The range of accepted values is from 1(smallest) to 7(largest). The default size of a font is 3.

```
<!DOCTYPE html>
<html>

<head>
  <title>Setting Font Size</title>
</head>

<body>
  <font size="1">Font size="1"</font><br />
  <font size="2">Font size="2"</font><br />
  <font size="3">Font size="3"</font><br />
  <font size="4">Font size="4"</font><br />
  <font size="5">Font size="5"</font><br />
  <font size="6">Font size="6"</font><br />
  <font size="7">Font size="7"</font>
```

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

## Setting Font Face

You can set the font face using *face* attribute but be aware that if the user viewing the page doesn't have the font installed, they will not be able to see it.

Instead user will see the default font face applicable to the user's computer.

```
<!DOCTYPE html>
<html>

<head>
  <title>Font Face</title>
</head>

<body>
  <font face="Times New Roman" size="5">Times New Roman</font><br />
  <font face="Verdana" size="5">Verdana</font><br />
  <font face="Comic sans MS" size="5">Comic Sans MS</font><br />
  <font face="WildWest" size="5">WildWest</font><br />
  <font face="Bedrock" size="5">Bedrock</font><br />
</body>

</html>
```

## Specify alternate font faces

A visitor will only be able to see your font if they have that font installed on their computer.

So, it is possible to specify two or more font face alternatives by listing the font face names, separated by a comma.

```
<font face="arial,helvetica">
```

```
<font face="Lucida Calligraphy,Comic Sans MS,Lucida Console">
```



When your page is loaded, their browser will display the first font face available.

If none of the given fonts are installed, then it will display the default font face

*Times New Roman.*

## Setting Font Color

You can set any font color you like using *color* attribute.

You can specify the color that you want by either the color name or hexadecimal code for that color.

```
<!DOCTYPE html>
<html>

<head>
  <title>Setting Font Color</title>
</head>
<body>
  <font color="#FF00FF">This text is in pink</font><br />
  <font color="red">This text is red</font>
</body>
</html>
```

## HTML Forms

HTML Forms are required when you want to collect some data from the site visitor.

For during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as ASP Script or PHP script etc.

The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like

text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML <form> tag is used to create an HTML form and it has following syntax:

```
<form action="Script URL"method="GET|POST">
```

form elements like input, textarea etc.

```
</form>
```

# Form Attributes

Apart from common attributes, following is a list of the most frequently used form attributes:

Attribute	Description
action	Backend script ready to process your passed data.
method	Method to be used to upload data.  The most frequently used are GET and POST methods.

## HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Text Input Controls
- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls

- Clickable Buttons
- Submit and Reset Button

# Text Input Controls

There are three types of text input used on forms:

- Single-line text input controls - This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML `<input>` tag.
- Password input controls - This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML `<input>` tag.
- Multi-line text input controls - This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML `<textarea>` tag.

## Single-line text input controls

This control is used for items that require only one line of user input, such as search boxes or names. They are created using the HTML `<input>` tag.

## Password input controls

This is also a single-line text input but it masks the character as soon as a user enters it.

They are also created using HTML `<input>` tag but type attribute is set to password.

```
<!DOCTYPE html>
<html>

<head>
  <title>Text Input Control</title>
</head>

<body>
  <form>
```

```
First name:
<input type="text" name="first_name" />
<br>
Last name:
<input type="text" name="last_name" />
</form>
</body>
</html>

<!DOCTYPE html>
<html>
<head>
  <title>Password Input Control</title>
</head>

<body>
  <form>
    User ID :
    <input type="text" name="user_id" />
    <br>
    Password:
    <input type="password" name="password" />
  </form>
</body>

</html>
```

## Attributes

Following is the list of attributes for <input> tag for creating password field.

Attribute	Description
type	Indicates the type of input control and for password input control it will be set to password.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.

value	This can be used to provide an initial value inside the control.
size	Allows to specify the width of the text-input control in terms of characters.
maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

## Multiple-Line Text Input Controls

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML `<textarea>` tag.

```
<!DOCTYPE html>
<html>

<head>
  <title>Multiple-Line Input Control</title>
</head>

<body>
  <form>
    Description :<br />
    <textarea rows="5" cols="50" name="description">
      Enter description here...
    </textarea>
  </form>
</body>

</html>
```

Description :

## Attributes

Following is the list of attributes for `<textarea>` tag.

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
rows	Indicates the number of rows of text area box.
cols	Indicates the number of columns of text area box

# Checkbox Control

Checkboxes are used when more than one option is required to be selected.

They are also created using HTML `<input>` tag but type attribute is set to checkbox.

Here is an HTML code for a form with two checkboxes:

## Attributes

Following is the list of attributes for `<checkbox>` tag.

Attribute	Description
Type	Indicates the type of input control and for checkbox input control it will be set to checkbox.

Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	The value that will be used if the checkbox is selected.
checked	Set to <i>checked</i> if you want to select it by default.

```

<!DOCTYPE html>
<html>
<head>
  <title>Checkbox Control</title>
</head>
<body>
  <form>
    <input type="checkbox" name="maths" value="on">Maths
    <input type="checkbox" name="physics" value="on"> Physics
  </form>
</body>
</html>

```

## Radio Button Control

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to radio.

### Attributes

Following is the list of attributes for radio button.

Attribute	Description
Type	Indicates the type of input control and for checkbox input control it will be set to radio.

Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Value	The value that will be used if the radio box is selected.
checked	Set to <i>checked</i> if you want to select it by default.

```
<!DOCTYPE html>
<html>
<head>
  <title>Radio Box Control</title>
</head>
<body>
  <form>
    <input type="radio" name="subject" value="maths">Maths
    <input type="radio" name="subject" value="physics"> Physics
  </form>
</body>
</html>
```

# Select Box Control

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

## Attributes

Following is the list of important attributes of `<select>` tag:



Attribute	Description
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Size	This can be used to present a scrolling list box.
Multiple	If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag:

Attribute	Description
Value	The value that will be used if an option in the select box box is selected.
Selected	Specifies that this option should be the initially selected value when the page loads.
Label	An alternative way of labeling options

```

<!DOCTYPE html>
<html>

<head>
  <title>Select Box Control</title>
</head>

<body>
  <form>
    <select name="dropdown">
      <option value="Maths" selected>Maths</option>

```

```
<option value="Physics">Physics</option>
</select>
</form>
</body>

</html>
```

# File Upload Box

If you want to allow a user to upload a file to your website, you will need to use a file upload box, also known as a file select box.

This is also created using the `<input>` element but type attribute is set to file.

## Attributes

Following is the list of important attributes of file upload box:

Attribute	Description
Name	Used to give a name to the control which is sent to the server to be recognized and get the value.
Accept	Specifies the types of files that the server accepts.

```
<!DOCTYPE html>
<html>
<head>
  <title>File Upload Box</title>
</head>
<body>
  <form>
    <input type="file" name="fileupload" accept="image/*" />
```

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

# Button Controls

There are various ways in HTML to create clickable buttons.

You can also create a clickable button using `<input>` tag by setting its type attribute to button. The type attribute can take the following values:

Type	Description
submit	This creates a button that automatically submits a form.
reset	This creates a button that automatically resets form controls to their initial values.
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

```
<!DOCTYPE html>
<html>
<head>
  <title>File Upload Box</title>
</head>
<body>
  <form>
```

```
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" />
<input type="button" name="ok" value="OK" />
<input type="image" name="imagebutton" src="/html/images/logo.png" />
</form>
</body>
</html>
```

# Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server.

This control hides inside the code and does not appear on the actual page.

For , the following hidden form is being used to keep the current page number.

When a user clicks the next page then the value of hidden control will be sent to the web server and there it will decide which page has been displayed next based on the passed current page.

```
<!DOCTYPE html>
<html>
<head>
  <title>Hidden Forms</title>
</head>
<body>
  <form>
    <p>This is page 10</p>
    <input type="hidden" name="pagename" value="10" />
    <input type="submit" name="submit" value="Submit" />
    <input type="reset" name="reset" value="Reset" />
  </form>
</body>
</html>
```

# HTML Marquees

An HTML marquee is a scrolling piece of text displayed either horizontally across or vertically down your webpage depending on the settings.

This is created by using the HTML `<marquees>` tag.

## Syntax

A simple syntax to use HTML `<marquee>` tag is as follows:

```
<marquee attribute_name="attribute_value".....moreattributes>
```

One or more lines or text message or image

```
</marquee>
```

## The `<marquee>` Tag Attributes

Following is the list of important attributes which can be used with `<marquee>` tag.

Attribute	Description
Width	This specifies the width of the marquee. This can be a value like 10 or 20% etc.
Height	This specifies the height of the marquee. This can be a value like 10 or 20% etc.
Direction	This specifies the direction in which marquee should scroll. This can be a value like <i>up</i> , <i>down</i> , <i>left</i> or <i>right</i> .

Behavior	This specifies the type of scrolling of the marquee. This can have a value like <i>scroll</i> , <i>slide</i> and <i>alternate</i> .
scrolldelay	This specifies how long to delay between each jump. This will have a value like 10 etc.
scrollamount	This specifies the speed of marquee text. This can have a value like 10 etc.
Loop	This specifies how many times to loop. The default value is INFINITE, which means that the marquee loops endlessly.
Bgcolor	This specifies background color in terms of color name or color hex value.

Below are a few s to demonstrate the usage of marquee tag.

```
<!DOCTYPE html>
<html>
<head>
  <title>HTML marquee Tag</title>
</head>
<body>
  <marquee> This is basic of marquee</marquee>
  <marquee width="50%">This will take only 50% width</marquee>
  <marquee direction="right">This text will scroll from left to right</marquee>
  <marquee direction="left">This text will scroll from right to left</marquee>
</body>
</html>
```

## HTML Layouts

A web page layout is very important to give a better look to your website.

It takes considerable time to design a website's layout with great look and feel.

Nowadays, all modern websites are using CSS and Javascript based frameworks to come up with responsive and dynamic websites but you can create a good layout using simple HTML tables or division tags in combination with other formatting tags.

This chapter will give you a few s on how to create a simple but working layout for your webpage using pure HTML and its attributes.

## HTML Layouts - Using DIV, SPAN

The <div> element is a block level element used for grouping HTML elements. While the <div> tag is a block-level element, the HTML <span> element is used for grouping elements at an inline level.

Although we can achieve pretty nice layouts with HTML tables, but tables weren't really designed as a layout tool. Tables are more suited to presenting tabular data.

TestLayout1.html

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML Layouts using DIV, SPAN</title>
</head>

<body>
  <div style="width:100%">
    <div style="background-color:#b5dcb3; width:100%">
      <h1>This is Web Page Main title</h1>
    </div>

    <div style="background-color:#aaa; height:200px;width:100px;float:left;">
      <div><b>Main Menu</b></div>
      HTML<br />
      PHP<br />
      PERL...
    </div>

    <div style="background-color:#eee; height:200px;width:350px;float:left;">
      <p>Technical and Managerial Tutorials</p>
    </div>
```

```

<div style="background-color:#aaa; height:200px;width:100px;float:right;">
  <div><b>Right Menu</b></div>
  HTML<br />
  PHP<br />
  PERL...
</div>

<div style="background-color:#b5dcb3;clear:both">
  <center>
    Copyright © 2023 kosmiktechnologies.com
  </center>
</div>
</div>
</body>

</html>

```

## TesLayout2.html

```

<html>

<head>
  <title>this is sample</title>
</head>

<body>

  <div style="width:100%;height:40px;background-color:red">
    <div style="width:50%;float:left">
      <p>+91 9999999999</p>
    </div>
    <div style="width:50%;float:right">
      <p style="float:right">kosmiktechnologies.com</p>
    </div>
  </div>

  <!--navbar starts here -->

  <div style="width:100%;height:100px">
    <div style="width:30%;float:left">
      <imgsrc="logo1.png" />
    </div>
    <div style="width:70%;float:right">

```





## What is CSS?

CSS stands for Cascading Style Sheets, is a simple design language **used to control the style of a web page in a simple and easy way.**

CSS handles the look and feel part of a web page.

Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs,

how columns are sized and laid out,

what background images or colors are used,

and layout designs, in display for different devices and screen sizes as well as a variety of other effects.

## Advantages of CSS

- **CSS saves time** – you can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it.
- **Pages load faster** – If you are using CSS, you do not need to write HTML tag attributes every time.
- Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- **Easy maintenance** – To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- **Multiple Device Compatibility** – Style sheets allow content to be optimized for more than one type of device.
- **Offline Browsing** – CSS can store web applications locally with the help of an offline cache. Using this, we can view offline websites. The cache also ensures faster loading and better overall performance of the website.
- **Platform Independence** – The Script offers consistent platform independence and can support the latest browsers as well.

## Who Creates and Maintains CSS?

CSS was invented by Håkon Wium Lie on October 10, 1994 and maintained through a group of people within the W3C called the CSS Working Group.

## CSS - Syntax

A CSS comprises style rules that are interpreted by the browser and then applied to the corresponding elements in your document.

A style rule is made of three parts –

- **Selector** – A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
- **Property** - A property is a type of attribute of an HTML tag.

Put simply, all the HTML attributes are converted into CSS properties.

They could be *color*, *border* etc.

- **Value** - Values are assigned to properties.

For example, *color* property can have value either *red* or *#F1F1F1* etc.

You can put CSS Style Rule Syntax as follows –

```
selector{property: value }  
table{ border :1px solid #C00; color :red}
```

Here table is a selector and border is a property and given value *1px solid #C00* is the value of that property.

You can define selectors in various simple ways based on your comfort.

Let me put these selectors one by one.

## The Type Selectors

This is the same selector we have seen above.

Again, one more example to give a color to all level 1 headings:

```
h1 {  
color:#36CFFF;  
}
```

## The Universal Selectors

Rather than selecting elements of a specific type,

the universal selector quite simply matches the name of any element type –

```
*{  
color:#000000;  
}
```

## The Descendant Selectors

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element.

As given in the following example, style rule will apply to <li> element only when it lies inside <ul> tag.

```
ul li{  
color:#000000;  
}
```

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Page Title</title>  
    <!-- <link  
      rel="stylesheet"  
      type="text/css"  
      media="screen"  
      href="main.css"  
    > -->  
    <style type="text/css">  
      /* type selectors */  
      h1{
```

```
        color: brown;
    }

    p{
        font-size: large;
        color: blueviolet;
    }

    /* Universal selector */
    *{
        color: green;
    }

    ul li{
        color: rgb(11, 99, 140);
        font-family: 'Gill Sans', 'Gill Sans MT', Calibri,
'Trebuchet MS', sans-serif;
    }
</style>
</head>
<body>
    <h1>Welcome to CSS</h1>
    <p> Hello guys, Good morning !! Welcome to CSS</p>
    <ul>
        <li>Apple</li>
        <li>Orange</li>
        <li>Grapes</li>
        <li>Banana</li>
    </ul>
    <ol>
        <li>Apple</li>
        <li>Orange</li>
        <li>Grapes</li>
        <li>Banana</li>
    </ol>
</body>
</html>
```

## The Class Selectors

You can define style rules based on the class attribute of the elements.

All the elements having that class will be formatted according to the defined rule.

```
.black {  
color:#000000;  
}
```

This rule renders the content in black for every element with class attribute set to *black* in our document.

You can make it a bit more particular. For example:

```
<p class="black">  
  This para will be styled by the classes center and bold.  
</p>
```

## The ID Selectors

You can define style rules based on the *id* attribute of the elements.

All the elements having that *id* will be formatted according to the defined rule.

```
#black {  
color:#000000;  
}
```

## The Child Selectors

You have seen the descendant selectors.

There is one more type of selector, which is very similar to descendants but has different functionality.

Consider the following example –

```
Body > p {  
color:#000000;
```

```
}
```

## The Attribute Selectors

You can also apply styles to HTML elements with particular attributes.

The style rule below will match all the input elements having a type attribute with a value of *text* –

```
input[type = "text"]{  
color: #000000;  
}
```

The advantage to this method is that the `<input type = "submit" />` element is unaffected, and the color applied only to the desired text fields.

## Multiple Style Rules

You may need to define multiple style rules for a single element.

You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example –

```
h1 {  
color: #36C;  
font-weight: normal;  
letter-spacing: .4em;  
margin-bottom: 1em;  
text-transform: lowercase;  
}
```

Here all the property and value pairs are separated by a semi colon (;). You can keep them in a single line or multiple lines. For better readability we keep them into separate lines.

## Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma, as given in the following example –

```
h1, h2, h3 {  
color: #36C;
```

```
font-weight: normal;
letter-spacing:.4em;
margin-bottom:1em;
text-transform: lowercase;
}
```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

You can combine the various *id* selectors together as shown below –

```
#content, #footer, #supplement {
position: absolute;
left:510px;
width:200px;
}
```

## CSS - Inclusion

---

There are four ways to associate styles with your HTML document.  
Most commonly used methods are inline CSS and External CSS.

### 1) Embedded CSS - The <style> Element

You can put your CSS rules into an HTML document using the <style> element.

This tag is placed inside <head>...</head> tags.

Rules defined using this syntax will be applied to all the elements available in the document.

Here is the generic syntax –

Following is the example of embed CSS based on the above syntax –

```
<!DOCTYPE html>
<html>
<head>
```



```
<style type="text/css" media="all">
```

```
body{
```

```
background-color: linen;
```

```
}
```

```
    h1 {
```

```
color: maroon;
```

```
margin-left:40px;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>This is a heading</h1>
```

```
<p>This is a paragraph.</p>
```

```
</body>
```

```
</html>
```

## Attributes

Attributes associated with <style> elements are –

Attribute	Value	Description
Type	text/css	Specifies the style sheet language as a content-type (MIME type). This is a required attribute.

## 2) Inline CSS - The *style* Attribute

You can use the style attribute of any HTML element to define style rules.

These rules will be applied to that element only. Here is the generic syntax –

`<element style="...style rules....">`

## Attributes

Attribute	Value	Description
Style	style rules	The value of <i>style</i> attribute is a combination of style declarations separated by semicolon (;).

Following is the example of inline CSS based on the above syntax –

```
<html>
<head>
</head>
<body>
<h1 style="color:#36C; font:arial"> This is inline CSS </h1>
</body>
</html>
```

### 3) External CSS - The <link> Element

The <link> element can be used to include an external stylesheet file

in your HTML document.

An external style sheet is a separate text file with .css extension.

You define all the Style rules within this text file and then you can include this file in any HTML document using <link> element.

Here is the generic syntax of including external CSS file –

```
<head>
<link type = "text/css" href = "..." media = "..." />
</head>
```

## Attributes

Attributes associated with <style> elements are –

Attribute	Value	Description
Type	text/css	Specifies the style sheet language as a content-type (MIME type). This attribute is required.
Href	URL	Specifies the style sheet file having Style rules.  This attribute is required.

```
h1, h2, h3 {
color:#36C;
font-weight: normal;
letter-spacing:.4em;
margin-bottom:1em;
text-transform: lowercase;
}
```

Now you can include this file *mystyle.css* in any HTML document as follows –

```
<head>
<linktype="text/css"href="mystyle.css"media=" all"/>
</head>
```

## CSS Rules Overriding

We have discussed three ways to include stylesheet rules in an HTML document.

Here is the rule to override any Style Sheet Rule.

- Any inline style sheet takes the highest priority.
- So, it will override any rule defined in `<style>...</style>` tags or rules defined in any external style sheet file.
- Any rule defined in `<style>...</style>` tags will override rules defined in any external style sheet file.

- Any rule defined in an external style sheet file takes lowest priority, and rules defined in this file will be applied only when the above two rules are not applicable.

## CSS Comments

Many times, you may need to put additional comments in your stylesheet blocks.

So, it is very easy to comment on any part of the style sheet.

You can simple put your comments inside `/*.....this is a comment in the style sheet.....*/`.

You can use `/* ....*/` to comment multi-line blocks in a similar way you do in C and C++ programming languages.

```
<!DOCTYPE html>
<html>
<head>
<style>
p{
color: red;
/* This is a single-line comment */
text-align: center;
}
/* This is a multi-line comment */
</style>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
```

## CSS - Measurement Units

CSS supports a number of measurements including

absolute units such as inches, centimeters, points, and so on, as well as

relative measures such as percentages and em units.

You need these values while specifying various measurements in your Style rules e.g border = "1px solid red".

We have listed out all the CSS Measurement Units along with proper Examples –

Unit	Description	Example
% or pt	Defines a measurement as a percentage relative to another value, typically an enclosing element.	<code>p {font-size: 16pt; line-height: 125%;}</code>
cm	Defines a measurement in centimeters.	<code>div {margin-bottom: 2cm;}</code>
em	<p>A relative measurement for the height of a font in em spaces.</p> <p>Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 16px; thus, 2em would be 32px.</p> <p>(ephemeral unit)</p>	<code>p {letter-spacing: 7em;}</code>

<b>px</b>	<b>Defines a measurement in screen pixels.</b>	<b>p {padding: 25px;}</b>
-----------	--	---------------------------

## CSS - Colors

CSS uses color values to specify a color.

Typically, these are used to set a color either for the foreground of an element (i.e., its text) or else for the background of the element.

They can also be used to affect the color of borders and other decorative effects.

You can specify your color values in various formats. Following table lists all the possible formats –

Format	Syntax	Example
Hex Code	#RRGGBB	p{color:#FF0000;}
Short Hex Code	#RGB	p{color:#6A7;}
RGB %	rgb(rrr%,ggg%,bbb%)	p{color:rgb(50%,50%,50%);}

RGB Absolute	<code>rgb(rrr,ggg,bbb)</code>	<code>p{color:rgb(0,0,255);}</code>
--------------	-------------------------------	-------------------------------------

These formats are explained in more detail in the following sections –

## CSS Colors - Hex Codes

A hexadecimal is a 6 digit representation of a color.

The first two digits(RR) represent a red value,

the next two are a green value(GG),and the

Last are the blue value(BB).

Each hexadecimal code will be preceded by a pound or hash sign '#'.

Following are the examples to use Hexadecimal notation.

## CSS Colors - Short Hex Codes

This is a shorter form of the six-digit notation.

In this format, each digit is replicated to arrive at an equivalent six-digit value.

For example: #6A7 becomes #66AA77.

Each hexadecimal code will be preceded by a pound or hash sign '#'.

Following are the examples to use Hexadecimal notation.

## CSS Colors - RGB Values

This color value is specified using the `rgb( )` property.

This property takes three values, one each for red, green, and blue.

The value can be an integer between 0 and 255 or a percentage.

Following is the example to show a few colors using RGB values.

## CSS - Background

How to set backgrounds of various HTML elements.

You can set the following background properties of an element –

- The **background-color** property is used to set the background color of an element.
- The **background-image** property is used to set the background image of an element.
- The **background-repeat** property is used to control the repetition of an image in the background.
- The **background-position** property is used to control the position of an image in the background.

## Set the Background Color

Set the background color for an element using background-color property.

```
<html>
```



```
<head>
<body>
<p style="background-color:yellow;">
    This text has a yellow background color.</p>
</body>
</head>
</html>
```

## Set the Background Image

We can set the background image by calling local stored images as shown below

```
<html>
<head>
<style>
body {
background-image:url("/css/images/css.jpg");
background-color:#cccccc;
}
</style>
<body>
<h1>Hello World!</h1>
</body>
</head>
</html>
```

## Repeat the Background Image

The following example demonstrates how to repeat the background image if an image is small.

You can use *no-repeat* value for *background-repeat* property if you don't want to repeat an image, in this case the image will display only once.

By default *background-repeat* property will have *repeat* value.

```
<html>
<head>
<style>
body{
background-image:url("/css/images/css.jpg");
background-repeat: repeat;
}
</style>
</head>
<body>
<p>Kosmik</p>
</body>
</html>
```

The following example demonstrates how to repeat the background image vertically.

```
<html>
<head>
<style>
body{
background-image:url("/css/images/css.jpg");
```

```
background-repeat: repeat-y;
}
</style>
</head>
<body>
<p>Kosmik</>
</body>
</html>
```

The following example demonstrates how to repeat the background image horizontally.

```
<html>
<head>
<style>
body{
background-image:url("/css/images/css.jpg");
background-repeat: repeat-x;
}
</style>
</head>
<body>
<p>Kosmik</>
</body>
</html>
```

## Set the Background Image Position

The following example demonstrates how to set the background image position 100 pixels away from the left side.

```
<html>
<head>
<style>
body{
background-image:url("/css/images/css.jpg");
background-position:100px;
}
</style>
</head>
<body>
<p>Kosmik</p>
</body>
</html>
```

The following example demonstrates how to set the background image position

100 pixels away from the left side and

200 pixels down from the top.

```
<html>
<head>
<style>
body{
```

```
background-image:url("/css/images/css.jpg");  
Background-position:100px 200px;  
}  
</style>  
</head>  
<body>  
<p>Kosmik</p>  
</body>  
</html>
```

## Set the Background Attachment

Background attachment determines whether a background image is fixed or scrolls with the rest of the page.

The following example demonstrates how to set the fixed background image.

```
<!DOCTYPE html>  
<html>  
<head>  
  
<style>  
body {  
background-image:url("/css/images/css.jpg");  
background-repeat:no-repeat;  
background-attachment:fixed;  
}  
</style>  
</head>  
<body>
```

```
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
</body>
</html>
```

The following example demonstrates how to set the scrolling background image.

```
<!DOCTYPE html>
<html>
<head>

<style>
body {
background-image:url('/css/images/css.jpg');
background-repeat:no-repeat;
background-attachment:fixed;
background-attachment:scroll;
}.
</style>

</head>
<body>

<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
```

```
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>
<p>The background-image is fixed. Try to scroll down the page.</p>

</body>
</html>
```

## CSS - Fonts

This Concepts teaches you how to set fonts of a content, available in an HTML element.

You can set following font properties of an element –

- The **font-family** property is used to change the face of a font.
- The **font-style** property is used to make a font italic or oblique.
- The **font-variant** property is used to create a small-caps effect.
- The **font-weight** property is used to increase or decrease how bold or light a font appears.
- The **font-size** property is used to increase or decrease the size of a font.
- The **font** property is used as shorthand to specify a number of other font properties.

## Set the Font Family

Following is the example, which demonstrates how to set the font family of an element.

Possible value could be any font family name.

```
<html>
<head>
</head>
<body>
<p style="font-family:georgia,garamond,serif;">
    This text is rendered in either georgia, garamond, or the default serif font
    depending on which font you have at your system.
</p>
</body>
</html>
```

## Set the Font Style

Following is the example, which demonstrates how to set the font style of an element. Possible values are *normal*, *italic* and *oblique*.

```
<html>
<head>
</head>
<body>
<p style="font-style:italic;">
    This text will be rendered in italic style
</p>
</body>
</html>
```

It will produce the following result –



## Set the Font Variant

The following example demonstrates how to set the font variant of an element. Possible values are *normal* and *small-caps*.

```
<html>
<head>
</head>
<body>
<p style="font-variant:small-caps;">
    This text will be rendered as small caps
</p>
</body>
</html>
```

It will produce the following result –

## Set the Font Weight

The following example demonstrates how to set the font weight of an element.

The font-weight property provides the functionality to specify how bold a font is.

Possible values could be *normal*, *bold*, *bolder*, *lighter*, *100*, *200*, *300*, *400*, *500*, *600*, *700*, *800*, *900*.

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

```
<p style="font-weight:bold;">This font is bold.</p>
<p style="font-weight:bolder;">This font is bolder.</p>
<p style="font-weight:500;">This font is 500 weight.</p>
</body>
</html>
```

## Set the Font Size

The following example demonstrates how to set the font size of an element.

The font-size property is used to control the size of fonts.

Possible values could be *xx-small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*, *smaller*, *larger*, *size in pixels* or *in %*.

```
<html>
<head>
</head>
<body>
<p style="font-size:20px;">This font size is 20 pixels</p>
<p style="font-size:small;">This font size is small</p>
<p style="font-size:large;">This font size is large</p>
</body>
</html>
```

## CSS - Text

This chapter teaches you how to manipulate text using CSS properties.

You can set following text properties of an element –

- The **color** property is used to set the color of a text.

- The **direction** property is used to set the text direction.
- The **letter-spacing** property is used to add or subtract space between the letters that make up a word.
- The **word-spacing** property is used to add or subtract space between the words of a sentence.
- The **text-align** property is used to align the text of a document.
- The **text-decoration** property is used to underline, overline, and strikethrough text.
- The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.

The **text-shadow** property is used to set the text shadow around a text.

## Set the Text Color

The following example shows how to set the text color. Possible value could be any color name in any valid format.

```
<html>
<head>
</head>
<body>
<p style="color:red;">
    This text will be written in red.
</p>
</body>
```

```
</html>
```

It will produce the following result –

## Set the Text Direction

The following example demonstrates how to set the direction of a text.

Possible values are *ltr* or *rtl*.

```
<html>
<head>
</head>
<body>
<p style="direction:rtl;">
    This text will be rendered from right to left
</p>
</body>
</html>
```

It will produce the following result –

## Set the Space between Characters

The following example demonstrates how to set the space between characters.

Possible values are *normal* or a number specifying space..

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

```
</head>
<body>
<p style="letter-spacing:5px;">
    This text is having space between letters.
</p>
</body>
</html>
```

It will produce the following result –

## Set the Space between Words

The following example demonstrates how to set the space between words.

Possible values are *normal* or a *number specifying space*.

```
<html>
<head>
</head>
<body>
<p style="word-spacing:5px;">
    This text is having space between words.
</p>
</body>
</html>
```

## Set the Text Alignment

The following example demonstrates how to align a text.

Possible values are *left*, *right*, *center*, *justify*.

```
<html>
<head>
</head>
<body>
<p style="text-align:right;">
    This will be right aligned.
</p>
<p style="text-align:center;">
    This will be center aligned.
</p>

<p style="text-align:left;">
    This will be left aligned.
</p>

</body>
</html>
```

## Decorating the Text

The following example demonstrates how to decorate a text.

Possible values are *none*, *underline*, *overline*, *line-through*.

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

```
</head>
<body>
<p style="text-decoration:underline;">
    This will be underlined
</p>

<p style="text-decoration:line-through;">
    This will be striked through.
</p>

<p style="text-decoration:overline;">
    This will have a over line.
</p>

<p style="text-decoration:blink;">
    This text will have blinking effect
</p>
</body>
</html>
```

## Set the Text Cases

*The following example demonstrates how to set the cases for a text.*

*Possible values are none, capitalize, uppercase, lowercase.*

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

```
<p style="text-transform:capitalize;">
```

This will be capitalized

```
</p>
```

```
<pstyle="text-transform:uppercase;">
```

This will be in uppercase

```
</p>
```

```
<p style="text-transform:lowercase;">
```

This will be in lowercase

```
</p>
```

```
</body>
```

```
</html>
```

## Set the White Space between Text

The following example demonstrates how white space inside an element is handled.

Possible values are *normal*, *pre*, *nowrap*.

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
<p style="white-space:pre;">
```

This text has a line break and the white-space pre setting tells the browser to honor it just like the HTML pre tag.</p>

```
</body>
```

```
</html>
```



## Set the Text Shadow

The following example demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

```
<html>
<head>
</head>
<body>
<p style="text-shadow:4px4px8px blue;">
    If your browser supports the CSS text-shadow property, this text will have a blue shadow.
</p>
</body>
</html>
```

## CSS - Using Images

Images play an important role in any webpage.

Though it is not recommended to include a lot of images, it is still important to use good images wherever required.

CSS plays a good role to control image display. You can set the following image properties using CSS.

- The **border** property is used to set the width of an image border.
- The **height** property is used to set the height of an image.
- The **width** property is used to set the width of an image.
- The **-moz-opacity** property is used to set the opacity of an image.

## The Image Border Property

The *border* property of an image is used to set the width of an image border.

This property can have a value in length or in %.

A width of zero pixels means no border.

Here is the example –

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

<br/>

</body>
</html>
```

## The Image Height Property

The *height* property of an image is used to set the height of an image.

This property can have a value in length or in %.

While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

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

<br/>

</body>
</html>
```

## The Image Width Property

The *width* property of an image is used to set the width of an image.

This property can have a value in length or in %.

While giving value in %, it applies it in respect of the box in which an image is available.

Here is an example –

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

<br/>

</body>
</html>
```

## CSS - Links

This chapter teaches you how to set different properties of a hyperlink using CSS.

You can set following properties of a hyperlink –

We will revisit the same properties when we discuss Pseudo-Classes of CSS.

- The **:link** signifies unvisited hyperlinks.
- The **:visited** signifies visited hyperlinks.
- The **:hover** signifies an element that currently has the user's mouse pointer hovering over it.
- The **:active** signifies an element on which the user is currently clicking.

```
<style type="text/css">
a:link {color: #000000}
a:visited {color: #006600}
a:hover {color: #FFCC00}
a:active {color: #FF00CC}
</style>
```

## CSS - Tables

This tutorial will teach you how to set different properties of an HTML table using CSS.

You can set following properties of a table –

- The **border-collapse** specifies whether the browser should control the appearance of the adjacent borders that touch each other or whether each cell should maintain its style.
- The **border-spacing** specifies the width that should appear between table cells.

- The **empty-cells** specify whether the border should be shown if a cell is empty.
- The **table-layout** allows browsers to speed up layout of a table by using the first width properties it comes across for the rest of a column rather than having to load the whole table before rendering it.

## The border-spacing Property

The border-spacing property specifies the distance that separates adjacent cells'.

borders. It can take either one or two values; these should be units of length.

If you provide one value, it will apply to both vertical and horizontal borders. Or you can specify two values, in which case, the first refers to the horizontal spacing and the second to the vertical spacing –

```
<style type="text/css">
  /* If you provide one value */
  table.example {border-spacing:10px;}
  /* This is how you can provide two values */
  table.example {border-spacing:10px; 15px;}
</style>
```

Now let's modify the previous example and see the effect –

## The table-layout Property

The table-layout property is supposed to help you control how a browser should render or lay out a table.

This property can have one of the three values: *fixed*, *auto* or *inherit*.

The following example shows the difference between these properties.

```
<html>
```

```
<head>
<style type="text/css">
table.auto{
table-layout:auto
}
table.fixed{
table-layout:fixed
}
</style>

</head>
<body>

<table class="auto" border="1"width="100%">
<tr>
<td width="20%">10000000000000000000000000000000</td>
<td width="40%">10000000</td>
<td width="40%">100</td>
</tr>
</table>
<br/>

<table class="fixed" border="1"width="100%">
<tr>
<td width="20%">10000000000000000000000000000000</td>
<td width="40%">10000000</td>
<td width="40%">100</td>
</tr>
</table>
```

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

## CSS - Borders

The *border* properties allow you to specify how the border of the box representing an element should look. There are three properties of a border you can change:

- The **border-color** specifies the color of a border.
- The **border-style** specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
- The **border-width** specifies the width of a border.

### The border-color Property

The border-color property allows you to change the color of the border surrounding an element. You can individually change the color of the bottom, left, top and right sides of an element's border using the properties –

- **border-bottom-color** changes the color of bottom border.
- **border-top-color** changes the color of top border.
- **border-left-color** changes the color of left border.
- **border-right-color** changes the color of right border.

The following example shows the effect of all these properties –

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

```
<style type="text/css">
p.example1{
border:1px solid;
border-bottom-color:#009900;/* Green */
border-top-color:#FF0000;/* Red */
border-left-color:#330000;/* Black */
border-right-color:#0000CC;/* Blue */
}
p.example2{
border:1px solid;
border-color:#009900;/* Green */
}
</style>
</head>
<body>
<p class="example1">
    This example shows all borders in different colors.
</p>

<p class="example2">
    This example is showing all borders in green color only.
</p>
</body>
</html>
```

## The border-style Property

The border-style property allows you to select one of the following styles of border –

- **none:** No border. (Equivalent of border-width:0;)



- **solid:** Border is a single solid line.
- **dotted:** Border is a series of dots.
- **dashed:** Border is a series of short lines.
- **double:** Border is two solid lines.
- **groove:** Border looks as though it is carved into the page.
- **ridge:** Border looks the opposite of groove.
- **inset:** Border makes the box look like it is embedded in the page.
- **outset:** Border makes the box look like it is coming out of the canvas.
- **hidden:** Same as none, except in terms of border-conflict resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using the following properties –

- **border-bottom-style** changes the style of bottom border.
- **border-top-style** changes the style of top border.
- **border-left-style** changes the style of left border.
- **border-right-style** changes the style of right border.

The following example shows all these border styles –

```
<html>
<head>
</head>
<body>
<p style="border-width:4px;border-style:none;">
    This is a border with none width.
```

</p>

<p style="border-width:4px;border-style:solid;">

This is a solid border.

</p>

<p style="border-width:4px;border-style:dashed;">

This is a dahsed border.

</p>

<p style="border-width:4px;border-style:double;">

This is a double border.

</p>

<p style="border-width:4px;border-style:groove;">

This is a groove border.

</p>

<p style="border-width:4px;border-style:ridge">

This is aridge border.

</p>

<p style="border-width:4px;border-style:inset;">

This is a inset border.

</p>

<p style="border-width:4px;border-style:outset;">

This is a outset border.

</p>

```
<p style="border-width:4px;border-style:hidden;">
    This is a hidden border.
</p>

<p style="border-width:4px;border-top-style:solid;
border-bottom-style:dashed;border-left-style:groove;border-right-style:double;">
    This is aa border with four different styles.
</p>
</body>

</html>
```

## The border-width Property

The border-width property allows you to set the width of an element's borders.

The value of this property could be either a length in px, pt or cm or it should be set to *thin*, *medium* or *thick*.

You can individually change the width of the bottom, top, left, and right borders of an element using the following properties –

- **border-bottom-width** changes the width of bottom border.
- **border-top-width** changes the width of top border.
- **border-left-width** changes the width of left border.
- **border-right-width** changes the width of right border.

The following example shows all these border width –

```
<html>
<head>
</head>
<body>
<p style="border-width:4px;border-style:solid;">
    This is a solid border whose width is 4px.
</p>

<p style="border-width:4pt;border-style:solid;">
    This is a solid border whose width is 4pt.
</p>

<p style="border-width:thin;border-style:solid;">
    This is a solid border whose width is thin.
</p>

<p style="border-width:medium;border-style:solid;">
    This is a solid border whose width is medium;
</p>

<p style="border-width:thick;border-style:solid;">
    This is a solid border whose width is thick.
</p>

<p style="border-bottom-width:4px;border-top-width:10px;
border-left-width:2px;border-right-width:15px;border-style:solid;">
    This is aa border with four different width.
</p>
</body>
```

```
</html>
```

## CSS - Margins

The *margin* property defines the space around an HTML element. It is possible to use negative values to overlap content.

The values of the margin property are not inherited by the child elements. Remember that the adjacent vertical margins (top and bottom margins) will collapse into each other so that the distance between the blocks is not the sum of the margins, but only the greater of the two margins or the same size as one margin if both are equal.

We have the following properties to set an element margin.

- The **margin** specifies a shorthand property for setting the margin properties in one declaration.
- The **margin-bottom** specifies the bottom margin of an element.
- The **margin-top** specifies the top margin of an element.
- The **margin-left** specifies the left margin of an element.
- The **margin-right** specifies the right margin of an element.

Now, we will see how to use these properties with examples.

### The Margin Property

The `margin` property allows you to set all of the properties for the four margins in one declaration. Here is the syntax to set margin around a paragraph –

Here is an example –

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

<body>
<p style="margin:15px; border:1px solid black;">
all four margins will be 15px
</p>

<p style="margin:2%; border:1px solid black;">
top and bottom margin will be 10px, left and right margin will be 2% of the total width of the
document.
</p>

<p style="margin:10px2%-10px; border:1px solid black;">
top margin will be 10px, left and right margin will be 2% of the total width of the document,
bottom margin will be -10px
</p>

<p style="margin:10px2%-10pxauto; border:1px solid black;">
top margin will be 10px, right margin will be 2% of the total width of the document, bottom
margin will be -10px, left margin will be set by the browser
</p>
</body>

</html>
```

## CSS - Lists

Lists are very helpful in conveying a set of either numbered or bullet points.

This chapter teaches you how to control list **type**, **position**, **style**, etc., using CSS.

We have the following five CSS properties, which can be used to control lists:

- The **list-style-type** allows you to control the shape or appearance of the marker.
- The **list-style-position** specifies whether a long point that wraps to a second line should align with the first line or start underneath the start of the marker.
- The **list-style** serves as shorthand for the preceding properties.

Now, we will see how to use these properties with examples.

## The list-style-type Property

The *list-style-type* property allows you to control the shape or style of bullet point (also known as a marker) in the case of unordered lists and the style of numbering characters in ordered lists.

Here are the values which can be used for an unordered list –

Value	Description
none	NA
disc (default)	A filled-in circle
circle	An empty circle

square	A filled-in square
--------	--------------------

Here are the values, which can be used for an ordered list –

Value	Description	Example
decimal	Number	1,2,3,4,5
decimal-leading-zero	0 before the number	01, 02, 03, 04, 05
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V
lower-greek	The marker is lower-greek	alpha, beta, gamma



lower-latin	The marker is lower-latin	a, b, c, d, e
upper-latin	The marker is upper-latin	A, B, C, D, E

Here is an example –

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

<body>
<ul style="list-style-type:circle;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ul style="list-style-type:square;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style-type:decimal;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

```
<ol style="list-style-type:lower-alpha;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>

<ol style="list-style-type:lower-roman;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
</body>

</html>
```

## The list-style Property

The *list-style* allows you to specify all the list properties into a single expression. These properties can appear in any order.

Here is an example –

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

<body>
```

```
<ul style="list-style: inside square;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style: outside upper-alpha;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
</body>

</html>
```

## CSS - Paddings

The *padding* property allows you to specify how much space should appear between the content of an element and its border –

The value of this attribute should be either a length, a percentage, or the word *inherit*.

If the value is *inherit*, it will have the same padding as its parent element.

If a percentage is used, the percentage is of the containing box.

The following CSS properties can be used to control lists.

You can also set different values for the padding on each side of the box using the following properties –

- 

The **padding-bottom** specifies the bottom padding of an element.

- The **padding-top** specifies the top padding of an element.
- The **padding-left** specifies the left padding of an element.
- The **padding-right** specifies the right padding of an element.
- The **padding** serves as shorthand for the preceding properties.

Now, we will see how to use these properties with examples.

## The padding-bottom Property

The *padding-bottom* property sets the bottom padding (space) of an element. This can take a value in terms of length of %.

Here is an example –

```
<html>
<head>
</head>
<body>
<p style="padding-bottom:15px; border:1px solid black;">
    This is a paragraph with a specified bottom padding
</p>

<p style="padding-bottom:5%; border:1px solid black;">
    This is another paragraph with a specified bottom padding in percent
```

```
</p>
</body>

</html>
```

## CSS - Cursors

The *cursor* property of CSS allows you to specify the type of cursor that should be displayed to the user.

One good usage of this property is in using images for submit buttons on forms. By default, when a cursor hovers over a link, the cursor changes from a pointer to a hand.

However, it does not change form for a submit button on a form

The following table shows the possible values for the cursor property –

Value	Description
auto	Shape of the cursor depends on the context area it is over. For example, an 'I' over text, a 'hand' over a link, and so on.
crosshair	A crosshair or plus sign
default	An arrow

<b>pointer</b>	<b>A pointing hand (in IE 4 this value is hand).</b>
<b>move</b>	<b>The 'I' bar</b>
<b>text</b>	<b>The I bar.</b>
<b>wait</b>	<b>An hour glass.</b>
<b>help</b>	<b>A question mark or balloon, ideal for use over help buttons.</b>
<b>&lt;url&gt;</b>	<b>The source of a cursor image file.</b>

```

<html>
<head>
</head>

<body>
<p>Move the mouse over the words to see the cursor change:</p>
<div style="cursor:auto">Auto</div>
<div style="cursor:crosshair">Crosshair</div>
<div style="cursor:default">Default</div>

```

```
<div style="cursor:pointer">Pointer</div>

<div style="cursor:move">Move</div>

<div style="cursor:e-resize">e-resize</div>

<div style="cursor:ne-resize">ne-resize</div>

<div style="cursor:nw-resize">nw-resize</div>

<div style="cursor:n-resize">n-resize</div>

<div style="cursor:se-resize">se-resize</div>

<div style="cursor:sw-resize">sw-resize</div>

<div style="cursor:s-resize">s-resize</div>

<div style="cursor:w-resize">w-resize</div>

<div style="cursor:text">text</div>

<div style="cursor:wait">wait</div>

<div style="cursor:help">help</div>

</body>

</html>
```

## CSS - Outlines

Outlines are very similar to borders, but there are few major differences as well –

- An outline does not take up space.
- Outlines do not have to be rectangular.
- Outline is always the same on all sides; you cannot specify different values for different sides of an element.

You can set the following outline properties using CSS.

- The `outline-width` property is used to set the width of the outline.
- The `outline-style` property is used to set the line style for the outline.
- The `outline-color` property is used to set the color of the outline.
- The `outline` property is used to set all the above three properties in a single statement.

## The outline-width Property

The *outline-width* property specifies the width of the outline to be added to the box. Its value should be a length or one of the values *thin*, *medium*, or *thick*, just like the `border-width` attribute.

A width of zero pixels means no outline.

Here is an example –

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

<body>
<p style="outline-width:thin;outline-style:solid;">
    This text is having thin outline.
</p>
<br/>

<p style="outline-width:thick;outline-style:solid;">
    This text is having thick outline.
</p>
<br/>
```



```
<p style="outline-width:5px;outline-style:solid;">
```

This text is having 5x outline.

```
</p>
```

```
</body>
```

```
</html>
```

## The outline-style Property

The *outline-style* property specifies the style for the line (solid, dotted, or dashed) that goes around an element. It can take one of the following values –

- **none:** No border. (Equivalent of `outline-width:0;`)
- **solid:** Outline is a single solid line.
- **dotted:** Outline is a series of dots.
- **dashed:** Outline is a series of short lines.
- **double:** Outline is two solid lines.
- **groove:** Outline looks as though it is carved into the page.
- **ridge:** Outline looks the opposite of groove.
- **inset:** Outline makes the box look like it is embedded in the page.
- **outset:** Outline makes the box look like it is coming out of the canvas.
- **hidden:** Same as none.

Here is an example –

```
<html>
```

```
<head>
</head>
<body>
<p style="outline-width:thin;outline-style:solid;">
    This text is having thin solid outline.
</p>
<br/>

<p style="outline-width:thick;outline-style:dashed;">
    This text is having thick dashed outline.
</p>
<br/>

<p style="outline-width:5px;outline-style:dotted;">
    This text is having 5x dotted outline.
</p>
</body>

</html>
```

## The outline-color Property

The *outline-color* property allows you to specify the color of the outline. Its value should either be a color name, a hex color, or an RGB value, as with the color and border-color properties.

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

```
</head>

<body>
<p style="outline-width:thin;outline-style:solid;outline-color:red">
    This text is having thin solid red outline.
</p>
<br/>

<p style="outline-width:thick;outline-style:dashed;outline-color:#009900">
    This text is having thick dashed green outline.
</p>
<br/>

<p style="outline-width:5px;outline-style:dotted;outline-color:rgb(13,33,232)">
    This text is having 5x dotted blue outline.
</p>
</body>

</html>
```

## The outline Property

The *outline* property is a shorthand property that allows you to specify values for any of the three properties discussed previously in any order but in a single statement.

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

<body>
```

```
<p style="outline:thin solid red;">
    This text is having thin solid red outline.
</p>
<br/>

<p style="outline:thick dashed #009900;">
    This text is having thick dashed green outline.
</p>
<br/>

<p style="outline:5px dotted rgb(13,33,232);">
    This text is having 5x dotted blue outline.
</p>
</body>

</html>
```

## CSS - Dimension

You have seen the border that surrounds every box ie. element, the padding that can appear inside each box and the margin that can go around them. In this tutorial we will learn how we can change the dimensions of boxes.

We have the following properties that allow you to control the dimensions of a box.

- The height property is used to set the height of a box.
- The width property is used to set the width of a box.
- The line-height property is used to set the height of a line of text.
- The max-height property is used to set a maximum height that a box can be.

- The min-height property is used to set the minimum height that a box can be.
- The max-width property is used to set the maximum width that a box can be.
- The min-width property is used to set the minimum width that a box can be.

## The Height and Width Properties

The *height* and *width* properties allow you to set the height and width for boxes. They can take values of a length, a percentage, or the keyword auto.

```
<html>
<head>
</head>
<body>
<p style="width:400px; height:100px; border:1px solid red; padding:5px; margin:10px;">
    This paragraph is 400pixels wide and 100 pixels high
</p>
</body>
</html>
```

## CSS - Scrollbars

There may be a case when an element's content might be larger than the amount of space allocated to it.

For example, given width and height properties do not allow enough room to accommodate the content of the element.

CSS provides a property called *overflow* which tells the browser what to do if the box's contents is larger than the box itself. This property can take one of the following values .

Value	Description
visible	Allows the content to overflow the borders of its containing element.
hidden	The content of the nested element is simply cut off at the border of the containing element and no scrollbars are visible.
scroll	The size of the containing element does not change, but the scrollbars are added to allow the user to scroll to see the content.

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

<style type="text/css">
.scroll{
display:block;
border:1px solid red;
padding:5px;
```

```
margin-top:5px;
```

```
width:300px;
```

```
height:50px;
```

```
overflow:scroll;
```

```
}
```

```
</style>
```

```
<body>
```

```
<p>Example of scroll value:</p>
```

```
<div class="scroll">
```

I am going to keep lot of content here just to show you how scrollbars works if there is an overflow in an

element box. This provides your horizontal as well as vertical scrollbars.

```
</div>
```

```
<br/>
```

```
<p>Example of auto value:</p>
```

```
</body>
```

```
</html>
```

## CSS - Visibility

A property called *visibility* allows you to hide an element from view. You can use this property along with JavaScript to create very complex menu and very complex webpage layouts.

You may choose to use the visibility property to hide error messages that are only displayed if the user needs to see them, or to hide answers to a quiz until the user selects an option.

The *visibility* property can take the values listed in the table that follows –

Value	Description
Visible	The box and its contents are shown to the user.
Hidden	The box and its content are made invisible, although they still affect the layout of the page.

```
<html>
<head>
</head>
<body>
<p>
    This paragraph should be visible in normal way.
</p>

<p style="visibility:hidden;">
```



**This paragraph should not be visible.**

`</p>`

`</body>`

`</html>`

## CSS - Positioning

CSS helps you to position your HTML element. You can put any HTML element at whatever location you like. You can specify whether you want the element positioned relative to its natural position in the page or absolute based on its parent element.

Now, we will see all the CSS positioning related properties with examples –

### Relative Positioning

Relative positioning changes the position of the HTML element relative to where it normally appears. So "left:20" adds 20 pixels to the element's LEFT position.

You can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for *left*.
- Move Right - Use a positive value for *left*.
- Move Up - Use a negative value for *top*.
- Move Down - Use a positive value for *top*.

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
<div style="position:relative;left:150px;top:2px;background-color:yellow;">
```

This div has relative positioning.

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

## Absolute Positioning

An element with **position: absolute** is positioned at the specified coordinates relative to your screen top-left corner.

You can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for *left*.
- Move Right - Use a positive value for *left*.
- Move Up - Use a negative value for *top*.
- Move Down - Use a positive value for *top*.

```
<html>
<head>
</head>
<body>
<div style="position: absolute; left: 80px; top: 20px; background-color: yellow;">
  This div has absolute positioning.
</div>
</body>
</html>
```

## Fixed Positioning

Fixed positioning allows you to fix the position of an element to a particular spot on the page, regardless of scrolling. Specified coordinates will be relative to the browser window.

You can use two values *top* and *left* along with the *position* property to move an HTML element anywhere in the HTML document.

- Move Left - Use a negative value for *left*.
- Move Right - Use a positive value for *left*.
- Move Up - Use a negative value for *top*.
- Move Down - Use a positive value for *top*.

```
<html>
<head>
</head>
<body>
<div style="position:fixed; left:80px; top:20px;background-color:yellow;">
  This div has fixed positioning.
</div>
</body>
</html>
```

## CSS - Layers

CSS gives you the opportunity to create layers of various divisions.

The CSS layers refer to applying the *z-index* property to elements that overlap with each other.

The *z-index* property is used along with the *position* property to create an effect of layers. You can specify which element should come on top and which element should come at bottom.

A z-index property can help you to create more complex web page layouts. Following is the example which shows how to create layers in CSS.

```
<html>

<head>

</head>

<body>

<div style="background-color:red; width:300px; height:100px;position:relative; top:10px; left:80px;
z-index:2">

</div>

<div style="background-color:yellow; width:300px; height:100px;position:relative; top:-60px; left:35px;
z-index:1;">

</div>

<div style="background-color:green; width:300px; height:100px;position:relative; top:-220px;
left:120px; z-index:3;">

</div>

</body>

</html>
```

# CSS - Pseudo Classes

CSS pseudo-classes are used to add special effects to some selectors. You do not need to use JavaScript or any other script to use those effects. A simple syntax of pseudo-classes is as follows –

```
selector:pseudo-class{property: value}
```

CSS classes can also be used with pseudo-classes –

```
selector.class:pseudo-class{property: value}
```

The most commonly used pseudo-classes are as follows –

Value	Description
-------	-------------

:link	Use this class to add special style to an unvisited link.
-------	---

:visited	Use this class to add special style to a visited link.
----------	--

:hover	Use this class to add special style to an element when you mouse over it.
--------	---

:active	Use this class to add special style to an active element.
---------	---

:focus	Use this class to add special style to an element while the element has focus.
--------	--

:first-child	Use this class to add special style to an element that is the first child of some other element.
--------------	--

:lang	Use this class to specify a language to use in a specified element.
-------	---

While defining pseudo-classes in a <style>...</style> block, following points should be noted –

- a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.
- a:active MUST come after a:hover in the CSS definition in order to be effective.
- Pseudo-class names are not case-sensitive.

- Pseudo-class are different from CSS classes but they can be combined.

## The :link pseudo-class

The following example demonstrates how to use the :link class to set the link color. Possible values could be any color name in any valid format.

```
<html>
<head>
<style type="text/css">
a:link {color:#000000}
</style>
</head>
<body>
<a href="">Black Link</a>
</body>
</html>
```

## The :visited pseudo-class

The following is the example which demonstrates how to use the *:visited* class to set the color of visited links. Possible values could be any color name in any valid format.

```
html>
<head>
<styletype="text/css">
a:visited {color:#006600}
```

```
</style>
</head>
<body>
<a href="">Click this link</a>
</body>
</html>
```

## The :hover pseudo-class

The following example demonstrates how to use the *:hover* class to change the color of links when we bring a mouse pointer over that link. Possible values could be any color name in any valid format.

```
<html>
<head>
<style type="text/css">
a:hover {color:#FFCC00}
</style>
</head>
<body>
<a href="">Bring Mouse Here</a>
</body>
</html>
```

It will produce the following link. Now you bring your mouse over this link and you will see that it changes its color to yellow.

## The :active pseudo-class

The following example demonstrates how to use the *:active* class to change the color of active links. Possible values could be any color name in any valid format.

```
<html>
<head>
<style type="text/css">
a:active {color:#FF00CC}
</style>
</head>
<body>
<a href="">Click This Link</a>
</body>
</html>
```

It will produce the following link. When a user clicks it, the color changes to pink.

## The :focus pseudo-class

The following example demonstrates how to use the *:focus* class to change the color of focused links. Possible values could be any color name in any valid format.

```
<html>
<head>
<style type="text/css">
a:focus {color:#0000FF}
</style>
</head>
<body>
<a href="">Click this Link</a>
</body>
</html>
```



It will produce the following link. When this link gets focused, its color changes to orange. The color changes back when it loses focus.

## The :first-child pseudo-class

The *:first-child* pseudo-class matches a specified element that is the first child of another element and adds special style to that element that is the first child of some other element.

To make :first-child work in IE <!DOCTYPE> must be declared at the top of document.

For example, to indent the first paragraph of all <div> elements, you could use this definition –

```
<html>
<head>

<style type="text/css">
div> p:first-child
{
text-indent:25px;
}
</style>

</head>
<body>

<div>
<p>First paragraph in div. This paragraph will be indented</p>
<p>Second paragraph in div. This paragraph will not be indented</p>
</div>

<p>But it will not match the paragraph in this HTML:</p>
```

```
<div>
<h3>Heading</h3>
<p>The first paragraph inside the div. This paragraph will not be effected.</p>
</div>

</body>
</html>
```

## CSS - Pseudo Elements

CSS pseudo-elements are used to add special effects to some selectors. You do not need to use JavaScript or any other script to use those effects. A simple syntax of pseudo-element is as follows

–

```
selector:pseudo-element{property: value}
```

CSS classes can also be used with pseudo-elements –

```
selector.class:pseudo-element{property: value}
```

The most commonly used pseudo-elements are as follows –

Value	Description
:first-line	Use this element to add special styles to the first line of the text in a selector.
:first-letter	Use this element to add special style to the first letter of the text in a selector.

:before	Use this element to insert some content before an element.
:after	Use this element to insert some content after an element.

## The :first-line pseudo-element

The following example demonstrates how to use the *:first-line* element to add special effects to the first line of elements in the document.

```
<html>
<head>
<style type="text/css">
p:first-line { text-decoration: underline;}
p.noline:first-line{ text-decoration: none;}
</style>
</head>
<body>
<p class="noline"> This line would not have any underline because this belongs to nline
class.</p>

<p>The first line of this paragraph will be underlined as defined in the CSS rule above. Rest of the
lines in this paragraph will remain normal. This example shows how to use :first-line pseduo
element to give effect to the first line of any HTML element.</p>
</body>
</html>
```

## The :first-letter pseudo-element

The following example demonstrates how to use the *:first-letter* element to add special effects to the first letter of elements in the document.

```
<html>

<head>

<style type="text/css">
p:first-letter { font-size:5em;}
p.normal:first-letter{ font-size:10px;}
</style>

</head>

<body>

<p class="normal"> First character of this paragraph will be normal and will have font size 10
px;</p>


<p>The first character of this paragraph will be 5em big as defined in the CSS rule above. Rest
of the characters in this paragraph will remain normal. This example shows how to use :first-letter
pseduo element to give effect to the first characters of any HTML element.</p>

</body>

</html>
```

## The :before pseudo-element

The following example demonstrates how to use the *:before* element to add some content before any element.

```
<html>

<head>

<style type="text/css">
p:before
{
content:url(/images/bullet.gif)

```

```
}
</style>
</head>

<body>
<p> This line will be preceded by a bullet.</p>
<p> This line will be preceded by a bullet.</p>
<p> This line will be preceded by a bullet.</p>
</body>
</html>
```

## The :after pseudo-element

The following example demonstrates how to use the *:after* element to add some content after any element.

```
<html>
<head>
<style type="text/css">
p:after
{
content:url(/images/bullet.gif)
}
</style>
</head>
<body>
<p> This line will be succeeded by a bullet.</p>
<p> This line will be succeeded by a bullet.</p>
<p> This line will be succeeded by a bullet.</p>
</body>
</html>
```

## CSS - Layouts

- Hope you are very comfortable with HTML tables and you are efficient in designing page layouts using HTML Tables.
- But you know CSS also provides plenty of controls for positioning elements in a document. Since CSS is *the wave of the future*, why not learn and use CSS instead of tables for page layout purposes?

The following list collects a few pros and cons of both the technologies –

- Most browsers support tables, while CSS support is being slowly adopted.
- Tables are more forgiving when the browser window size changes - morphing their content and wrapping to accommodate the changes accordingly. CSS positioning tends to be exact and fairly inflexible.
- Tables are much easier to learn and manipulate than CSS rules.

But each of these arguments can be reversed –

- CSS is pivotal to the future of Web documents and will be supported by most browsers.
- CSS is more exact than tables, allowing your document to be viewed as you intended, regardless of the browser window.
- Keeping track of nested tables can be a real pain. CSS rules tend to be well organized, easily read, and easily changed.

Finally, we would suggest you to use whichever technology makes sense to you and use what you know or what presents your documents in the best way.

CSS also provides *table-layout* property to make your tables load much faster. Following is an example –

```
<table style="table-layout:fixed;width:600px;">
<tr height="30">
<td width="150">CSS table layout cell 1</td>
<td width="200">CSS table layout cell 2</td>
<td width="250">CSS table layout cell 3</td>
</tr>
</table>
```

## Sample Column Layout

- Here are the steps to create a simple Column Layout using CSS –
- Set the margin and padding of the complete document as follows –

```
<style style="text/css">
<!--
body {
margin:9px 9px 0 9px;
padding:0;
background:#FFF;
}
-->
</style>
```

Now, we will define a column with yellow color and later, we will attach this rule to a <div>:

```
<style style="text/css">
<!--
#level0 {
```

```
background:#FC0;  
  
}  
-->  
</style>
```

- Upto this point, we will have a document with yellow body, so let us now define another division inside level0 –

```
<style style="text/css">  
<!--  
  #level1 {  
margin-left:143px;  
padding-left:9px;  
background:#FFF;  
  }  
  -->  
</style>
```

Now, we will nest one more division inside level1, and we will change just background color –

```
<style style="text/css">  
<!--  
  #level2 {  
background:#FFF3AC;  
  }  
  -->  
</style>
```

Finally, we will use the same technique, nest a level3 division inside level2 to get the visual layout for the right column –



```
<style style="text/css">
<!--
  #level3 {
margin-right:143px;
padding-right:9px;
background:#FFF;
  }
  #main {
background:#CCC;
  }
-->
</style>
```

Complete the source code as follows –

```
<style style="text/css">
body{
margin:9px9px09px;
padding:0;
background:#FFF;
}

#level0 {background:#FC0;}

#level1 {
margin-left:143px;
padding-left:9px;
```

```
background:#FFF;
}

#level2 {background:#FFF3AC;}

#level3 {
margin-right:143px;
padding-right:9px;
background:#FFF;
}

#main {background:#CCC;}
</style>
<body>
<div id="level0">
<div id="level1">
<div id="level2">
<div id="level3">
<div id="main">
    Final Content goes here...
</div>
</div>
</div>
</div>
</div>
</div>
</body>
```

Similarly, you can add a top navigation bar or an ad bar at the top of the page.

# What is Java Script?

→ JavaScript is lightweight and most commonly used as a part of web pages, whose implementations allow **client-side script** to interact with the user and make dynamic pages.

→ It is an interpreted programming language with object-oriented capabilities.

→ It is open and cross-platform.

→ JavaScript was first known as **Live Script**, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.

→ JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **Live Script**.

JavaScript is a lightweight, interpreted programming language.

- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

## Client-side JavaScript

Client-side JavaScript is the most common form of the language.

The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts.

For example, you might use JavaScript to check if the user has entered a valid email address in a form field.

## Advantages of JavaScript

- **Less server interaction**
  - You can validate user input before sending the page off to the server.
  - This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors**
  - They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity**
  - You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces –**
  - You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## JavaScript - Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The **<script>** tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
```

JavaScript code

```
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like –

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

JavaScript code

```
</script>
```

## First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code.

```
<html>

<body>

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

        document.write("Hello World!")

    </script>
```

```
</body>

</html>
```

## Whitespace and Line Breaks

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

## Semicolons are Optional

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java.
- JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line.
- For example, the following code could be written without semicolons.

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

    var1 = 10

    var2 = 20

</script>
```

But when formatted in a single line as follows, you must use semicolons –

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

    var1 = 10; var2 = 20;

</script>
```

## Case Sensitivity

JavaScript is a case-sensitive language.

This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

## Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">  
  
  <!--  
  
    // This is a comment. It is similar to comments in C++  
  
  /*
```

```
* This is a multiline comment in JavaScript  
  
* It is very similar to comments in C Programming  
  
*/  
  
//-->  
  
</script>
```

## JavaScript - Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document.

However the most preferred ways to include JavaScript in an HTML file are as follows

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then included in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

### JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

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



```
        function sayHello() {  
            alert("Hello World")  
        }  
    </script>  
</head>  
<body>  
    <input type="button" onclick="sayHello()" value="Say Hello" />  
</body>  
</html>
```

This code will produce the following results –

JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

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

```
</script>

<p>This is web page body </p>

</body>
</html>
```

This code will produce the following results –

## JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows

```
<html>

  <head>

    <script type="text/javascript">

      function sayHello() {

        alert("Hello World")

      }

    </script>

  </head>

  <body>

    <script type="text/javascript">

      document.write("Hello World")

    </script>

    <input type="button" onclick="sayHello()" value="Say Hello" />

  </body>
```

```
</html>
```

## JavaScript in External File

- As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.
- You are not restricted to maintaining identical code in multiple HTML files.
- The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.
- Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

```
<html>

<head>

    <script type="text/javascript" src="filename.js" ></script>

</head>

<body>

    .....

</body>

</html>
```

- To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.
- For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

```
function sayHello() {  
    alert("Hello World")  
}
```

## JavaScript - Variables

### JavaScript Data types

- One of the most fundamental characteristics of a programming language is the set of data types it supports.
- These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types –

- Numbers, eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.
- JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value.
- In addition to these primitive data types, JavaScript supports a composite data type known as object.

We will cover objects in detail in a separate chapter.

### JavaScript Variables

- Like many other programming languages, JavaScript has variables.
- Variables can be thought of as named containers.

- You can place data into these containers and then refer to the data simply by naming the container.
- Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

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

```
<!--
```

```
    var money;
```

```
    var name;
```

```
//-->
```

```
</script>
```

You can also declare multiple variables with the same var keyword as follows –

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

```
<!--
```

```
    var money, name;
```

```
//-->
```

```
</script>
```

- Storing a value in a variable is called variable initialization.
- You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.
- For instance, you might create a variable named money and assign the value 2000.50 to it later.
- For another variable, you can assign a value at the time of initialization as follows.

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

```
<!--
```

```
    var name = "Ali";
```

```
    var money;
```

```
money = 2000.50;

//-->

</script>
```

- JavaScript is an untyped language.
- This means that a JavaScript variable can hold a value of any data type. Unlike many other languages.

## JavaScript Variable Scope

- The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.
  - Global Variables – A global variable has global scope which means it can be defined anywhere in your JavaScript code.
  - Local Variables – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.
- Within the body of a function, a local variable takes precedence over a global variable with the same name.
- If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>

<body onload = checkscope();>

  <script type = "text/javascript">

    <!--

    var myVar = "global"; // Declare a global variable

    function checkscope( ) {

      var myVar = "local"; // Declare a local variable

      document.write(myVar);
```

```
    }  
    //-->  
    </script>  
  
</body>  
  
</html>
```

This produces the following result –

local

## JavaScript Variable Names

- While naming your variables in JavaScript, keep the following rules in mind.
- You should not use any of the JavaScript reserved keywords as a variable name.
- These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
  - JavaScript variable names should not start with a numeral (0-9).
  - They must begin with a letter or an underscore character.
  - For example, 123test is an invalid variable name but \_123test is a valid one.
  - JavaScript variable names are case-sensitive.
  - For example, Name and name are two different variables.

## JavaScript Reserved Words

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

<b>abstract</b>	<b>else</b>	<b>instanceof</b>	<b>Switch</b>
<b>boolean</b>	<b>enum</b>	<b>int</b>	<b>synchronized</b>
<b>break</b>	<b>export</b>	<b>interface</b>	<b>this</b>
<b>byte</b>	<b>extends</b>	<b>long</b>	<b>throw</b>
<b>case</b>	<b>false</b>	<b>native</b>	<b>throws</b>
<b>catch</b>	<b>final</b>	<b>new</b>	<b>transient</b>
<b>char</b>	<b>finally</b>	<b>null</b>	<b>true</b>
<b>class</b>	<b>float</b>	<b>package</b>	<b>try</b>
<b>const</b>	<b>for</b>	<b>private</b>	<b>typeof</b>
<b>continue</b>	<b>function</b>	<b>protected</b>	<b>var</b>
<b>debugger</b>	<b>goto</b>	<b>public</b>	<b>void</b>
<b>default</b>	<b>if</b>	<b>return</b>	<b>volatile</b>
<b>delete</b>	<b>implements</b>	<b>short</b>	<b>while</b>
<b>do</b>	<b>import</b>	<b>static</b>	<b>with</b>
<b>double</b>	<b>in</b>	<b>super</b>	



# JavaScript - Operators

## What is an operator?

Let us take a simple expression: 4 + 5 is equal to 9.

Here 4 and 5 are called operands and '+' is called the operator.

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look at all the operators one by one.

## Arithmetic Operators

JavaScript supports the following arithmetic operators –

Assume variable A holds 10 and variable B holds 20, then –

Sr.No	Operator and Description
1	<p><b>+</b> (Addition)</p> <p>Adds two operands</p> <p>Ex: A + B will give 30</p>

2	<p><b>- (Subtraction)</b></p> <p>Subtracts the second operand from the first</p> <p>Ex: A - B will give -10</p>
3	<p><b>* (Multiplication)</b></p> <p>Multiply both operands</p> <p>Ex: A * B will give 200</p>
4	<p><b>/ (Division)</b></p> <p>Divide the numerator by the denominator</p> <p>Ex: B / A will give 2</p>
5	<p><b>% (Modulus)</b></p> <p>Outputs the remainder of an integer division</p> <p>Ex: B % A will give 0</p>
6	<p><b>++ (Increment)</b></p> <p>Increases an integer value by one</p> <p>Ex: A++ will give 11</p>
7	<p><b>-- (Decrement)</b></p> <p>Decreases an integer value by one</p> <p>Ex: A-- will give 9</p>

**Note – Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".**

The following code shows how to use arithmetic operators in JavaScript.

## Output

**a + b = 43**

**a - b = 23**

**a / b = 3.3**

**a % b = 3**

**a + b + c = 43Test**

**++a = 35<html>**

**<body>**

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

```
<!--
```

```
var a = 33;
```

```
var b = 10;
```

```
var c = "Test";
```

```
var linebreak = "<br />";
```

```
document.write("a + b = ");
```

```
result = a + b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a - b = ");
```

```
result = a - b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a / b = ");
```

```
result = a / b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a % b = ");
```

```
result = a % b;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("a + b + c = ");
```

```
result = a + b + c;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
a = ++a;
```

```
document.write(++a = );
```

```
result = ++a;
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
b = --b;
```

```
document.write("--b = ");  
  
result = --b;  
  
document.write(result);  
  
document.write(linebreak);  
  
//-->  
  
</script>
```

Set the variables to different values and then try...

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

--b = 8

Set the variables to different values and then try...

## Comparison Operators

JavaScript supports the following comparison operators –

Assume variable A holds 10 and variable B holds 20, then –

Sr.No	Operator and Description
1	<p><b>== (Equal)</b></p> <p>Checks if the value of two operands are equal or not, if yes, then the condition becomes true.</p> <p>Ex: (A == B) is not true.</p>

2	<p><b>!= (Not Equal)</b></p> <p>Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.</p> <p>Ex: (A != B) is true.</p>
3	<p><b>&gt; (Greater than)</b></p> <p>Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A &gt; B) is not true.</p>
4	<p><b>&lt; (Less than)</b></p> <p>Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A &lt; B) is true.</p>
5	<p><b>&gt;= (Greater than or Equal to)</b></p> <p>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A &gt;= B) is not true.</p>
6	<p><b>&lt;= (Less than or Equal to)</b></p> <p>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A &lt;= B) is true.</p>

The following code shows how to use comparison operators in JavaScript.

```
<html>
```

```
<body>
```

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

```
<!--
```

```
var a = 10;
```

```
var b = 20;
```

```
var linebreak = "<br />";
```

```
document.write("(a == b) => ");
```

```
result = (a == b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a < b) => ");
```

```
result = (a < b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a > b) => ");
```

```
result = (a > b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a != b) ==> ");
```

```
result = (a != b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a >= b) ==> ");
```

```
result = (a >= b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("(a <= b) ==> ");
```

```
result = (a <= b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
//-->
```

```
</script>
```

Set the variables to different values and different operators and then try...

```
</body>
```

```
</html>
```

## Output

```
(a == b) ==> false
```

```
(a < b) ==> true
```

```
(a > b) ==> false
```

```
(a != b) ==> true
```



`(a >= b) => false`

`a <= b) => true`

Set the variables to different values and different operators and then try...

## Logical Operators

JavaScript supports the following logical operators –

Assume variable A holds 10 and variable B holds 20, then –

Sr.No	Operator and Description
1	<p><b>&amp;&amp; (Logical AND)</b></p> <p>If both the operands are non-zero, then the condition becomes true.</p> <p>Ex: (A &amp;&amp; B) is true.</p>
2	<p><b>   (Logical OR)</b></p> <p>If any one of the two operands are non-zero, then the condition becomes true.</p> <p>Ex: (A    B) is true.</p>
3	<p><b>! (Logical NOT)</b></p> <p>Reverses the logical state of its operand.</p> <p>If a condition is true, then the Logical NOT operator will make it false.</p> <p>Ex: ! (A &amp;&amp; B) is false.</p>

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>

<body>


    <script type="text/javascript">

        var a = true;

        var b = false;

        var linebreak = "<br />";


        document.write("(a && b) => ");

        result = (a && b);

        document.write(result);

        document.write(linebreak);


        document.write("(a || b) => ");

        result = (a || b);

        document.write(result);

        document.write(linebreak);


        document.write("(!(a && b) => ");

        result = (!(a && b));

        document.write(result);

        document.write(linebreak);
```

```
</script>

<p>Set the variables to different values and different operators and then try...</p>

</body>
</html>
Output

(a && b) => false
(a || b) => true
!(a && b) => true

Set the variables to different values and different operators and then try...
```

# Assignment Operators

JavaScript supports the following assignment operators –

Sr.No	Operator and Description
1	<b>= (Simple Assignment )</b>  Assigns values from the right side operand to the left side operand  Ex: C = A + B will assign the value of A + B into C
2	<b>+= (Add and Assignment)</b>  Sum += number;  It adds the right operand to the left operand and assigns the result to the left operand.

	<p>Ex: <code>C += A</code> is equivalent to <code>C = C + A</code></p>
3	<p><b><code>--</code> (Subtract and Assignment)</b></p> <p>It subtracts the right operand from the left operand and assigns the result to the left operand.</p> <p>Ex: <code>C -= A</code> is equivalent to <code>C = C - A</code></p>
4	<p><b><code>*=</code> (Multiply and Assignment)</b></p> <p>It multiplies the right operand with the left operand and assigns the result to the left operand.</p> <p>Ex: <code>C *= A</code> is equivalent to <code>C = C * A</code></p>
5	<p><b><code>/=</code> (Divide and Assignment)</b></p> <p>It divides the left operand with the right operand and assigns the result to the left operand.</p> <p>Ex: <code>C /= A</code> is equivalent to <code>C = C / A</code></p>
6	<p><b><code>%=</code> (Modules and Assignment)</b></p> <p>It takes modulus using two operands and assigns the result to the left operand.</p> <p>Ex: <code>C %= A</code> is equivalent to <code>C = C % A</code></p>

**Note** – Same logic applies to Bitwise operators so they will become like `<<=`, `>>=`, `&=`, `|=` and `^=`.

Try the following code to implement assignment operator in JavaScript.

```
<html>
```

```
<body>
```

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

```
<!--
```

```
var a = 33;
```

```
var b = 10;
```

```
var linebreak = "<br />";
```

```
document.write("Value of a => (a = b) => ");
```

```
result = (a = b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("Value of a => (a += b) => ");
```

```
result = (a += b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("Value of a => (a -= b) => ");
```

```
result = (a -= b);
```

```
document.write(result);
```

```
document.write(linebreak);
```

```
document.write("Value of a => (a *= b) => ");
```

```
result = (a *= b);

document.write(result);

document.write(linebreak);


document.write("Value of a => (a /= b) => ");

result = (a /= b);

document.write(result);

document.write(linebreak);


document.write("Value of a => (a %= b) => ");

result = (a %= b);

document.write(result);

document.write(linebreak);

//-->

</script>
```

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

## Output

Value of a => (a = b) => 10

Value of a => (a += b) => 20

Value of a => (a -= b) => 10

Value of a => (a \*= b) => 100

Value of a => (a /= b) => 10

Value of a => (a % b) => 0

Set the variables to different values and different operators and then try...

## Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No	Operator and Description
1	? : (Conditional )  If Condition is true? Then value X : Otherwise value Y

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
```

```
<body>
```

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

```
<!--
```

```
var a = 10;
```

```
var b = 20;
```

```
var linebreak = "<br />";
```

```
document.write ("((a > b) ? 100 : 200) => ");
```

```
result = (a > b) ? 100 : 200;

document.write(result);

document.write(linebreak);


document.write ("((a < b) ? 100 : 200) => ");

result = (a < b) ? 100 : 200;

document.write(result);

document.write(linebreak);

//-->

</script>
```

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

## Output

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100

Set the variables to different values and different operators and then try...

## Type of Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.



The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the typeof Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

The following code shows how to implement the typeof operator.

```
<html>

<body>
```

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

<!--

var a = 10;

var b = "String";

var linebreak = "<br />";


result = (typeof b == "string" ? "B is String" : "B is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);


result = (typeof a == "string" ? "A is String" : "A is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);

//-->

</script>
```

<p>Set the variables to different values and different operators and then try...</p>

</body>

</html>

## Output

Result => B is String

Result => A is Numeric

Set the variables to different values and different operators and then try...

## JavaScript - if...else Statement

- While writing a program, there may be a situation when you need to adopt one out of a given set of paths.
- In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.
- JavaScript supports conditional statements which are used to perform different actions based on different conditions.
- Here we will explain the if..else statement.

### FlowChart of if-else

The following flowchart shows how the if-else statement works.

//TODO if else flowchart

JavaScript supports the following forms of if..else statement –

- if statement
- if...else statement
- if...else if... statement.

### if statement

- The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

### Syntax

The syntax for a basic if statement is as follows –

```
if (expression){
```

Statement(s) to be executed if expression is true

}

- Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed.
- If the expression is false, then no statement would be not executed.
- Most of the time, you will use comparison operators while making decisions.

Try the following example to understand how the if statement works.

```
<html>
```

```
<body>
```

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

```
<!--
```

```
var age = 20;
```

```
if( age > 18 ){
```

```
document.write("<b>Qualifies for driving</b>");
```

```
}
```

```
//-->
```

```
</script>
```

```
<p>Set the variable to different value and then try...</p>
```

```
</body>
```

```
</html>
```

## Output

Qualifies for driving

Set the variable to different value and then try...

## if...else statement

The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

## Syntax

```
if (expression){
```

```
    Statement(s) to be executed if expression is true
```

```
}
```

```
else{
```

```
    Statement(s) to be executed if expression is false
```

```
}
```

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
```

```
    <body>
```

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

```
            <!--
```

```
            var age = 15;
```

```
            if( age > 18 ){
```

```
                document.write("<b>Qualifies for driving</b>");
```

```
            }
```

```
        else{

            document.write("<b>Does not qualify for driving</b>");

        }

        //-->

    </script>

    <p>Set the variable to different value and then try...</p>

</body>

</html>
```

## Output

Does not qualify for driving

Set the variable to different value and then try...

## if...else if... statement

The if...else if... statement is an advanced form of if...else that allows JavaScript to make a correct decision out of several conditions.

## Syntax

The syntax of an if-else-if statement is as follows –

```
if (expression 1){

    Statement(s) to be executed if expression 1 is true

}

else if (expression 2){

    Statement(s) to be executed if expression 2 is true

}
```

```
else if (expression 3){
```

```
    Statement(s) to be executed if expression 3 is true
```

```
}
```

```
else{
```

```
    Statement(s) to be executed if no expression is true
```

```
}
```

- There is nothing special about this code.
- It is just a series of if statements, where each if is a part of the else clause of the previous statement.
- Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
```

```
<body>
```

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

```
        <!--
```

```
        var book = "maths";
```

```
        if( book == "history" ){
```

```
            document.write("<b>History Book</b>");
```

```
        }
```

```

else if( book == "maths" ){

document.write("<b>Maths Book</b>");

}


else if( book == "economics" ){

document.write("<b>Economics Book</b>");

}


else{

document.write("<b>Unknown Book</b>");

}

//-->

</script>


<p>Set the variable to different value and then try...</p>

</body>

<html>

```

## Output

**Maths Book**

Set the variable to different value and then try...

## JavaScript - Switch Case

- You can use multiple if...else...if statements, as in the previous chapter, to perform a multiway branch.



- However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.
- Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

## Flow Chart

The following flow chart explains a switch-case statement works.

## Syntax

- The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression.
- The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

switch (expression)

{

case condition 1: statement(s)

break;

case condition 2: statement(s)

break;

...

case condition n: statement(s)

break;

default: statement(s)

}

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain break statement in *Loop Control* chapter.

Try the following example to implement switch-case statement.

```
<html>

<body>

    <script type="text/javascript">

        <!--

        var grade='A';

        document.write("Entering switch block<br />");

        switch (grade)

        {

            case 'A': document.write("Good job<br />");

            break;

            case 'B': document.write("Pretty good<br />");

            break;

            case 'C': document.write("Passed<br />");

            break;

            case 'D': document.write("Not so good<br />");

            break;
```

```
case 'F': document.write("Failed<br />");  
  
break;  
  
default: document.write("Unknown grade<br />")  
  
}  
  
document.write("Exiting switch block");  
  
!-->  
  
</script>  
  
<p>Set the variable to different value and then try...</p>  
  
</body>  
  
</html>
```

## Output

Entering switch block

Good job

Exiting switch block

Set the variable to different value and then try...

**Break statements play a major role in switch-case statements.**

**Try the following code that uses switch-case statements without any break statement.**

```
<html>  
  
  <body>  
  
  
  
  
  
  
  
  
  
    <script type="text/javascript">  
  
      <!--  
  
      var grade='A';
```

```
document.write("Entering switch block<br />");

switch (grade)

{

case 'A': document.write("Good job<br />");

case 'B': document.write("Pretty good<br />");

case 'C': document.write("Passed<br />");

case 'D': document.write("Not so good<br />");

case 'F': document.write("Failed<br />");

default: document.write("Unknown grade<br />")

}

document.write("Exiting switch block");

//-->

</script>
```

<p>Set the variable to different value and then try...</p>

</body>

</html>

## Output

Entering switch block

Good job

Pretty good

Passed

Not so good

Failed

Unknown grade

Exiting switch block

Set the variable to different value and then try...

## JavaScript - While Loops

- While writing a program, you may encounter a situation where you need to perform an action over and over again.
- In such situations, you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to ease down the pressure of programming.

### The while Loop

The most basic loop in JavaScript is the while loop which would be discussed in this chapter.

The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true.

Once the expression becomes false, the loop terminates.

### Flow Chart

The flow chart of while loop looks as follows –

### Syntax

The syntax of while loop in JavaScript is as follows –

```
while (expression){
```

```
    Statement(s) to be executed if expression is true
```

```
}
```

Try the following example to implement a while loop.

```
<html>
```

```
<body>
```

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

```
<!--
```

```
var count = 0;
```

```
document.write("Starting Loop ");
```

```
while (count < 10){
```

```
document.write("Current Count : " + count + "<br />");
```

```
count++;
```

```
}
```

```
document.write("Loop stopped!");
```

```
//-->
```

```
</script>
```

```
<p>Set the variable to different value and then try...</p>
```

```
</body>
```

```
</html>
```

## Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

## The do...while Loop

- The do...while loop is similar to the while loop except that the condition check happens at the end of the loop.
- This means that the loop will always be executed at least once, even if the condition is false.

### Syntax

The syntax for do-while loop in JavaScript is as follows –

```
do{  
    Statement(s) to be executed;  
} while (expression);
```

Try the following example to learn how to implement a do-while loop in JavaScript.

```
<html>

<body>


<script type="text/javascript">

<!--

var count = 0;


document.write("Starting Loop" + "<br />");

do{

    document.write("Current Count : " + count + "<br />");

    count++;

}

while (count < 5);

document.write ("Loop stopped!");

//-->

</script>


<p>Set the variable to different value and then try...</p>

</body>

</html>
```

## Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2



Current Count : 3

Current Count : 4

Loop Stopped!

Set the variable to different value and then try...

## JavaScript - For Loop

The 'for' loop is the most compact form of looping.

It includes the following three important parts –

- The loop initialization where we initialize our counter to a starting value.

The initialization statement is executed before the loop begins.

- The test statement which will test if a given condition is true or not.

If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

### Syntax

The syntax of for loop in JavaScript is as follows –

```
for (initialization; test condition; iteration statement){  
    Statement(s) to be executed if test condition is true  
}
```

Try the following example to learn how a for loop works in JavaScript.

```
<html>
```

```
    <body>
```

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

<!--

var count;

document.write("Starting Loop" + "<br />");


for(count = 0; count < 10; count++){

document.write("Current Count : " + count );

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

}


document.write("Loop stopped!");

//-->

</script>


<p>Set the variable to different value and then try...</p>

</body>

</html>
```

## Output

Starting Loop

Current Count : 0

Current Count : 1

Current Count : 2

Current Count : 3

Current Count : 4

Current Count : 5

Current Count : 6

Current Count : 7

Current Count : 8

Current Count : 9

Loop stopped!

Set the variable to different value and then try...

## JavaScript - Loop Control

- JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching its bottom.
- There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.
- To handle all such situations, JavaScript provides break and continue statements.
- These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

### The break Statement

- The break statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

## Flow Chart

The flow chart of a break statement would look as follows –

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace –

```
<html>

<body>


    <script type="text/javascript">

        <!--

        var x = 1;

        document.write("Entering the loop<br /> ");

        while (x < 20)

        {

            if (x == 5){

                break; // breaks out of loop completely

            }

            x = x + 1;

            document.write( x + "<br />");

        }

        document.write("Exiting the loop!<br /> ");

        //-->

    </script>
```

<p>Set the variable to different value and then try...</p>

</body>

</html>

## Output

Entering the loop

2

3

4

5

Exiting the loop!

Set the variable to different value and then try...

**We already have seen the usage of break statement inside a switchstatement.**

## The continue Statement

- The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.
- When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.
- This example illustrates the use of a continue statement with a while loop.
- Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 –

<html>

<body>

<script type="text/javascript">

<!--

var x = 1;

```
document.write("Entering the loop<br /> ");

while (x < 10)

{

x = x + 1;

if (x == 5){

    continue; // skip rest of the loop body

}

document.write( x + "<br />");

}

document.write("Exiting the loop!<br /> ");

//-->

</script>

<p>Set the variable to different value and then try...</p>

</body>

</html>
```

## Output

Entering the loop

2

3

4

6

7

8

9

10

Exiting the loop!

## JavaScript - Functions

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.
- It helps programmers in writing modular codes
- Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions.
- You must have seen functions like alert() and write() in the earlier chapters.
- JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

### Function Definition

- Before we use a function, we need to define it.
- The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

### Syntax

The basic syntax is shown here.

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

```
<!--
```

```
function functionname(parameter-list)
```

```
    {  
      statements  
    }  
  }  
  </script>
```

- Try the following example. It defines a function called sayHello that takes no parameters –

```
<script type="text/javascript">  
  <!--  
    function sayHello()  
    {  
      alert("Welcome to JS");  
    }  
  </-->  
</script>
```

## Calling a Function

- To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>  
  <head>  
  
    <script type="text/javascript">  
      function sayHello()  
      {  
        document.write ("Hello there!");  
      }  
    </script>  
  </head>  
</html>
```



```
    }  
    </script>  
  
</head>  
<body>  
    <p>Click the following button to call the function</p>  
  
    <form>  
        <input type="button" onclick="sayHello()" value="Say Hello">  
    </form>  
  
    <p>Use different text in write method and then try...</p>  
</body>  
</html>
```

## Function Parameters

- Till now, we have seen functions without parameters.
- But there is a facility to pass different parameters while calling a function.
- These passed parameters can be captured inside the function and any manipulation can be done over those parameters.
- A function can take multiple parameters separated by comma.

Try the following example. We have modified our sayHello function here.

Now it takes two parameters.

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

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

function sayHello(name, age)

{

document.write (name + " is " + age + " years old.");

}

</script>


</head>

<body>

<p>Click the following button to call the function</p>


<form>

<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">

</form>


<p>Use different parameters inside the function and then try...</p>

</body>

</html>
```

## The return Statement

- A JavaScript function can have an optional return statement.
- This is required if you want to return a value from a function.
- This statement should be the last statement in a function.
- For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

- Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
```

```
<head>
```

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

```
function concatenate(first, last)
```

```
{
```

```
var full;
```

```
full = first + last;
```

```
return full;
```

```
}
```

```
function secondFunction()
```

```
{
```

```
var result;
```

```
result = concatenate('Zara', 'Ali');
```

```
document.write (result );
```

```
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<p>Click the following button to call the function</p>
```

```
<form>

<input type="button" onclick="secondFunction()" value="Call Function">

</form>


<p>Use different parameters inside the function and then try...</p>


</body>

</html>
```

## JavaScript – Events

### What is an Event?

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event.
- When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

### onClick Event

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Try the following example.

```
<html>

<head>

    <script type="text/javascript">

        <!--

        function sayHello() {

            alert("Hello World")

        }

        //-->

    </script>

</head>

<body>

    <p>Click the following button and see result</p>

    <form>

        <input type="button" onclick="sayHello()" value="Say Hello" />

    </form>

</body>

</html>
```

## onsubmit Event

- onsubmit is an event that occurs when you try to submit a form.
- You can put your form validation against this event type.
- The following example shows how to use onsubmit.
- Here we are calling a validate() function before submitting a form data to the web server.
- If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```
<html>
<head>

    <script type="text/javascript">

        function validateform() {
            var name = document.loginform.name.value;
            var password = document.loginform.pass.value;

            if(name==null || name==""){
                alert("Name should not be empty or null !!");
                return false;
            }else if(password.length<8){
                alert("password should contain minimum 8 charcaters !!");
                return false;
            }

        }

    </script>

</head>
<body>

    <form name="loginform" method="POST" action="welcome.jsp" onsubmit="return
validateform()">

        <label>Username</label><input type="text" name="name">
        <label>Password</label><input type="password" name="pass">
        <input type="submit" value="login" />
    </form>
```

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

## onmouseover and onmouseout

- These two event types will help you create nice effects with images or even with text as well.
- The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element. Try the following example.

```
<html>
```

```
<head>
```

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

```
<!--
```

```
function over() {
```

```
document.write ("Mouse Over");
```

```
}
```

```
function out() {
```

```
document.write ("Mouse Out");
```

```
}
```

```
//-->
```

```
</script>
```

```
</head>
```

```
<body>
```

```
  <p>Bring your mouse inside the division to see the result:</p>
```

```
  <div onmouseover="over()" onmouseout="out()">
```

```
    <h2> This is inside the division </h2>
```

```
  </div>
```

```
</body>
```

```
</html>
```

## HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference.

Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Onabort	script	Triggers on an abort event
Onblur	script	Triggers when the window loses focus
onchange	script	Triggers when an element changes
Onclick	script	Triggers on a mouse click



<b>ondblclick</b>	<b>script</b>	<b>Triggers on a mouse double-click</b>
<b>Ondrag</b>	<b>script</b>	<b>Triggers when an element is dragged</b>
<b>onkeydown</b>	<b>script</b>	<b>Triggers when a key is pressed</b>
<b>onkeypress</b>	<b>script</b>	<b>Triggers when a key is pressed and released</b>
<b>Onkeyup</b>	<b>script</b>	<b>Triggers when a key is released</b>
<b>Onload</b>	<b>script</b>	<b>Triggers when the document loads</b>
<b>onmousedown</b>	<b>script</b>	<b>Triggers when a mouse button is pressed</b>
<b>onmousemove</b>	<b>script</b>	<b>Triggers when the mouse pointer moves</b>
<b>onmouseout</b>	<b>script</b>	<b>Triggers when the mouse pointer moves out of an element</b>
<b>onmouseover</b>	<b>script</b>	<b>Triggers when the mouse pointer moves over an element</b>
<b>onmouseup</b>	<b>script</b>	<b>Triggers when a mouse button is released</b>

<b>onmousewheel</b>	<b>script</b>	<b>Triggers when the mouse wheel is being rotated</b>
<b>onsubmit</b>	<b>script</b>	<b>Triggers when a form is submitted</b>

## JavaScript - Page Redirection

### What is Page Redirection?

- You might have encountered a situation where you clicked a URL to reach page X but internally you were directed to another page Y.
- It happens due to page redirection.
- There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons –

- You did not like the name of your domain and you are moving to a new one.

In such a scenario, you may want to direct all your visitors to the new site.

Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.

- You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
- The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search

engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

## How Page Re-direction Works ?

The implementations of Page-Redirection are as follows.

It is quite simple to do a page redirect using JavaScript at client side.

To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>

<head>

    <script type="text/javascript">

        function Redirect() {

            window.location="http://www.tutorialspoint.com";

        }

    </script>

</head>

<body>

    <p>Click the following button, you will be redirected to home page.</p>

    <form>

        <input type="button" value="Redirect Me" onclick="Redirect();" />
```

```
</form>
```

```
</body>
```

```
</html>
```

## Example 2

```
<html>
```

```
<head>
```

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

```
<!--
```

```
function Redirect() {
```

```
    window.location="http://www.tutorialspoint.com";
```

```
}
```

```
document.write("You will be redirected to main page in 10 sec.");
```

```
setTimeout('Redirect()', 10000);
```

```
//-->
```

```
</script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

## Output

You will be redirected to tutorialspoint.com main page in 10 seconds!

## Example

The following example shows how to redirect your site visitors onto a different page based on their browsers.

```
<html>

<head>

    <script type="text/javascript">

        <!--

        var browsername=navigator.appName;

        if( browsername == "Netscape" )

        {

            window.location="http://www.location.com/ns.htm";

        }

        else if ( browsername == "Microsoft Internet Explorer")

        {

            window.location="http://www.location.com/ie.htm";

        }

        else

        {

            window.location="http://www.location.com/other.htm";

        }

        //-->

    </script>

</head>
```

```
<body>

</body>

</html>
```

## JavaScript - Dialog Boxes

JavaScript supports three important types of dialog boxes.

These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

### Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires entering some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages.

Alert box gives only one button "OK" to select and proceed.

```
<html>

<head>
```

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

<!--

function Warn() {

alert ("This is a warning message!");

document.write ("This is a warning message!");

}

//-->

</script>

</head>

<body>

<p>Click the following button to see the result: </p>

<form>

<input type="button" value="Click Me" onclick="Warn();" />

</form>

</body>

</html>
```

## Confirmation Dialog Box

A confirmation dialog box is mostly used to take the user's consent on any option. It displays a dialog box with two buttons: Cancel.

If the user clicks on the OK button, the window method `confirm()` will return true. If the user clicks on the Cancel button, then `confirm()` returns false. You can use a confirmation dialog box as follows.

```
<html>

<head>

    <script type="text/javascript">

        <!--

        function getConfirmation(){

            var retVal = confirm("Do you want to continue ?");

            if( retVal == true ){

                document.write ("User wants to continue!");

                return true;

            }

            else{

                document.write ("User does not want to continue!");

                return false;

            }

        }

        //-->

    </script>

</head>

<body>
```



```
<p>Click the following button to see the result: </p>

<form>

<input type="button" value="Click Me" onclick="getConfirmation();" />

</form>

</body>
</html>
```

## Prompt Dialog Box

- The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.
- This dialog box is displayed using a method called `prompt()` which takes two parameters:
  - (i) a label which you want to display in the text box and
  - (ii) a default string to display in the text box.
- This dialog box has two buttons: OK and Cancel.
- If the user clicks the OK button, the window method `prompt()` will return the entered value from the text box. If the user clicks the Cancel button, the window method `prompt()` returns null.

The following example shows how to use a prompt dialog box –

```
<html>

<head>

    <script type="text/javascript">

        <!--

        function getValue(){

            var retVal = prompt("Enter your name : ", "your name here");

            document.write("You have entered : " + retVal);

        }

        //-->

    </script>

</head>

<body>

    <p>Click the following button to see the result: </p>

    <form>

        <input type="button" value="Click Me" onclick="getValue();" />

    </form>

</body>

</html>
```

## JavaScript - Page Printing

Many times you would like to place a button on your webpage to print the content of that web page via an actual printer.

JavaScript helps you to implement this functionality using the print function of window object.

The JavaScript print function `window.print()` prints the current web page when executed. You can call this function directly using the onclick event as shown in the following example.

Try the following example.

```
<html>

  <head>

    <script type="text/javascript">

    </script>

  </head>

  <body>

    <form>

      <input type="button" value="Print" onclick="window.print()" />

    </form>

  </body>

</html>
```

## Object Methods

**Methods are the functions that let the object do something or let something be done to it.**

**There is a small difference between a function and a method**

- a function is a standalone unit of statements and
- a method is attached to an object and can be referenced by this keyword.
- Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example – Following is a simple example to show how to use the write() method of document object to write any content on the document.**

```
document.write("This is test");
```

## **User-Defined Objects**

**All user-defined objects and built-in objects are descendants of an object called Object.**

## **The new Operator**

**The new operator is used to create an instance of an object.**

**To create an object, the new operator is followed by the constructor method.**

**In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.**

```
var employee = new Object();
```

```
var books = new Array("C++", "Perl", "Java");
```

```
var day = new Date("August 15, 1947");
```

## **JavaScript Native Objects**

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects –

- [JavaScript String Object](#)
- [JavaScript Array Object](#)
- [JavaScript Date Object](#)
- [JavaScript Math Object](#)
- [JavaScript RegExp Object](#)

## JavaScript - The Number Object

The Number object represents numerical data, either integers or floating-point numbers.

In general, you do not need to worry about Number objects because the browser automatically converts number literals to instances of the number class.

### Syntax

The syntax for creating a number object is as follows –

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

## JavaScript - The Strings Object

- The String object lets you work with a series of characters;

- it wraps Javascript's string primitive data type with a number of helper methods.
- As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

## Syntax

Use the following syntax to create a String object –

```
var val = new String(string);
```

The String parameter is a series of characters that has been properly encoded.

## String Methods

- Here is a list of the methods available in String objects along with their description.
-

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Method	Description
<a href="#"><u>charAt()</u></a>	Returns the character at the specified index.
<a href="#"><u>charCodeAt()</u></a>	Returns a number indicating the Unicode value of the character at the given index.
<a href="#"><u>concat()</u></a>	Combines the text of two strings and returns a new string.
<a href="#"><u>indexOf()</u></a>	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.

<u><a href="#">lastIndexOf()</a></u>	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
<u><a href="#">localeCompare()</a></u>	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
<u><a href="#">match()</a></u>	Used to match a regular expression against a string.
<u><a href="#">replace()</a></u>	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
<u><a href="#">search()</a></u>	Executes the search for a match between a regular expression and a specified string.
<u><a href="#">slice()</a></u>	Extracts a section of a string and returns a new string.
<u><a href="#">split()</a></u>	Splits a String object into an array of strings by separating the string into substrings.
<u><a href="#">substr()</a></u>	Returns the characters in a string beginning at the specified location through the specified number of characters.
<u><a href="#">substring()</a></u>	Returns the characters in a string between two indexes into the string.



<u><a href="#">toLocaleLowerCase()</a></u>	The characters within a string are converted to lower case while respecting the current locale.
<u><a href="#">toLocaleUpperCase()</a></u>	The characters within a string are converted to upper case while respecting the current locale.
<u><a href="#">toLowerCase()</a></u>	Returns the calling string value converted to lower case.
<u><a href="#">toString()</a></u>	Returns a string representing the specified object.
<u><a href="#">toUpperCase()</a></u>	Returns the calling string value converted to uppercase.
<u><a href="#">valueOf()</a></u>	Returns the primitive value of the specified object.

```

<html>
  <head>
    <script type="text/javascript">
function fun1(){
document.write('Helloooooo');
  const a = 'Hello';
  const b = new String('Sreenu');

  document.write(a);
  document.write(b);
  document.write(typeof a);
  document.write(typeof b);
    </script>
  </head>
</html>

```



```
document.write(linebreak);
//toLocaleLowerCase two Strings
const result2 = var1.toLocaleLowerCase();
document.write(result2);
document.write(linebreak);

//toLocaleUpperCase two Strings
const result3 = var1.toLocaleUpperCase();
document.write(result3);
document.write(linebreak);

//toLowerCase two Strings
const result4 = var1.toLowerCase();
document.write(result4);
document.write(linebreak);

//toUpperCase two Strings
const result5 = var1.toUpperCase();
document.write(result5);
document.write(linebreak);

//Removing whitespace from the starting and ending
const result6 = var3.trim();
document.write(result6);
document.write(linebreak);

//Converting String to an Array
const result7 = var2.split();
document.write(result7);
console.log(result7);
document.write(linebreak);

//Slicing the String
const result8 = var2.slice(1,4);
document.write(result8);
document.write(linebreak);
```

```

    //converting into String
    const a = 100;
    const b = true;

    const result9 = String(a);
    const result11 = result9+100;
    const result10 = String(b);
    console.log(result9);
    console.log(result10);
    console.log(result11);

    //escape charcater
    const name = 'My Name is \'Rahul\'';
    console.log(name);

}
</script>
</head>
<body>
    <form>
        <input type="button" value="hello" onclick="fun1();">
    </form>
</body>
</html>

```

## Arrays Object

- The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

## Syntax

Use the following syntax to create an Array object –

```
var fruits = new Array( "apple", "orange", "mango" );
```

The Array parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows –

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element

## Array Methods

Here is a list of the methods of the Array object along with their description.

Method	Description
<u><a href="#">concat()</a></u>	Returns a new array comprised of this array joined with other array(s) and/or value(s).

<u><a href="#">every()</a></u>	Returns true if every element in this array satisfies the provided testing function.
<u><a href="#">filter()</a></u>	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
<u><a href="#">forEach()</a></u>	Calls a function for each element in the array.
<u><a href="#">indexOf()</a></u>	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
<u><a href="#">join()</a></u>	Joins all elements of an array into a string.
<u><a href="#">lastIndexOf()</a></u>	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
<u><a href="#">map()</a></u>	Creates a new array with the results of calling a provided function on every element in this array.
<u><a href="#">pop()</a></u>	Removes the last element from an array and returns that element.
<u><a href="#">push()</a></u>	Adds one or more elements to the end of an array and returns the new length of the array.
<u><a href="#">reduce()</a></u>	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
<u><a href="#">reduceRight()</a></u>	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

<u><a href="#">reverse()</a></u>	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
<u><a href="#">shift()</a></u>	Removes the first element from an array and returns that element.
<u><a href="#">slice()</a></u>	Extracts a section of an array and returns a new array.
<u><a href="#">some()</a></u>	Returns true if at least one element in this array satisfies the provided testing function.
<u><a href="#">toSource()</a></u>	Represents the source code of an object
<u><a href="#">sort()</a></u>	Sorts the elements of an array
<u><a href="#">splice()</a></u>	Adds and/or removes elements from an array.
<u><a href="#">toString()</a></u>	Returns a string representing the array and its elements.
<u><a href="#">unshift()</a></u>	Adds one or more elements to the front of an array and returns the new length of the array.

In the following sections, we will have a few examples to demonstrate the usage of Array methods.

## The Date Object

The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date( ) as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

### Syntax

You can use any of the following syntaxes to create a Date object using Date() constructor.

`new Date( )`

`new Date(milliseconds)`

`new Date(datestring)`

`new Date(year,month,date[,hour,minute,second,millisecond ])`

**Note – Parameters in the brackets are always optional.**

Here is a description of the parameters –

- **No Argument –** With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds –** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring –** When one string argument is passed, it is a string representation of a date, in the format accepted by the Date.parse()method.
- **7 arguments –** To use the last form of the constructor shown above. Here is a description of each argument:



- **year** – Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
- **month** – Integer value representing the month, beginning with 0 for January to 11 for December.
- **date** – Integer value representing the day of the month.
- **hour** – Integer value representing the hour of the day (24-hour scale).
- **minute** – Integer value representing the minute segment of a time reading.
- **second** – Integer value representing the second segment of a time reading.
- **millisecond** – Integer value representing the millisecond segment of a time reading.

## Date Methods

Here is a list of the methods used with Date and their description.

Method	Description
<u><a href="#">Date()</a></u>	Returns today's date and time
<u><a href="#">getDate()</a></u>	Returns the day of the month for the specified date according to local time.
<u><a href="#">getDay()</a></u>	Returns the day of the week for the specified date according to local time.

<u><a href="#">getFullYear()</a></u>	Returns the year of the specified date according to local time.
<u><a href="#">getHours()</a></u>	Returns the hour in the specified date according to local time.
<u><a href="#">getMilliseconds()</a></u>	Returns the milliseconds in the specified date according to local time.
<u><a href="#">getMinutes()</a></u>	Returns the minutes in the specified date according to local time.
<u><a href="#">getMonth()</a></u>	Returns the month in the specified date according to local time.
<u><a href="#">getSeconds()</a></u>	Returns the seconds in the specified date according to local time.
<u><a href="#">getTime()</a></u>	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
<u><a href="#">getYear()</a></u>	Deprecated - Returns the year in the specified date according to local time. Use <code>getFullYear</code> instead.
<u><a href="#">setDate()</a></u>	Sets the day of the month for a specified date according to local time.
<u><a href="#">setFullYear()</a></u>	Sets the full year for a specified date according to local time.

<u><a href="#">setHours()</a></u>	Sets the hours for a specified date according to local time.
<u><a href="#">setMilliseconds()</a></u>	Sets the milliseconds for a specified date according to local time.
<u><a href="#">setMinutes()</a></u>	Sets the minutes for a specified date according to local time.
<u><a href="#">setMonth()</a></u>	Sets the month for a specified date according to local time.
<u><a href="#">setSeconds()</a></u>	Sets the seconds for a specified date according to local time.
<u><a href="#">setTime()</a></u>	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
<u><a href="#">setYear()</a></u>	Deprecated - Sets the year for a specified date according to local time. Use setFullYear instead.
<u><a href="#">toDateString()</a></u>	Returns the "date" portion of the Date as a human-readable string.
<u><a href="#">toString()</a></u>	Returns a string representing the specified Date object.
<u><a href="#">toTimeString()</a></u>	Returns the "time" portion of the Date as a human-readable string.

Converts a date to a string, using the universal time convention.

## The Math Object

The `math` object provides you properties and methods for mathematical constants and functions. Unlike other global objects, `Math` is not a constructor. All the properties and methods of `Math` are static and can be called by using `Math` as an object without creating it.

Thus, you refer to the constant `pi` as `Math.PI` and you call the *sine* function as `Math.sin(x)`, where `x` is the method's argument.

## Syntax

The syntax to call the properties and methods of `Math` are as follows

```
var pi_val = Math.PI;
```

```
var sine_val = Math.sin(30);
```

In the following sections, we will have a few examples to demonstrate the usage of `Math` properties.

## Math Methods

Here is a list of the methods associated with `Math` object and their description

Method	Description
<u><a href="#">abs()</a></u>	Returns the absolute value of a number.

<a href="#"><u>acos()</u></a>	Returns the arccosine (in radians) of a number.
<a href="#"><u>asin()</u></a>	Returns the arcsine (in radians) of a number.
<a href="#"><u>atan()</u></a>	Returns the arctangent (in radians) of a number.
<a href="#"><u>atan2()</u></a>	Returns the arctangent of the quotient of its arguments.
<a href="#"><u>ceil()</u></a>	Returns the smallest integer greater than or equal to a number.
<a href="#"><u>cos()</u></a>	Returns the cosine of a number.
<a href="#"><u>exp()</u></a>	Returns $E^N$ , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
<a href="#"><u>floor()</u></a>	Returns the largest integer less than or equal to a number.
<a href="#"><u>log()</u></a>	Returns the natural logarithm (base E) of a number.
<a href="#"><u>max()</u></a>	Returns the largest of zero or more numbers.
<a href="#"><u>min()</u></a>	Returns the smallest of zero or more numbers.
<a href="#"><u>pow()</u></a>	Returns base to the exponent power, that is, base exponent.

<u><a href="#">random()</a></u>	Returns a pseudo-random number between 0 and 1.
<u><a href="#">round()</a></u>	Returns the value of a number rounded to the nearest integer.
<u><a href="#">sin()</a></u>	Returns the sine of a number.
<u><a href="#">sqrt()</a></u>	Returns the square root of a number.
<u><a href="#">tan()</a></u>	Returns the tangent of a number.

In the following sections, we will have a few examples to demonstrate the usage of the methods associated with Math.

## Regular Expressions and RegExp Object

A regular expression is an object that describes a pattern of characters. The JavaScript RegExp class represents regular expressions, and both String and RegExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

### Syntax

A regular expression could be defined with the RegExp () constructor, as follows

–

```
var pattern = new RegExp(pattern, attributes);
```

or simply

```
var pattern = /pattern/attributes;
```

Here is the description of the parameters –

- **pattern** – A string that specifies the pattern of the regular expression or another regular expression.
- **attributes** – An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

## Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range `[0-3]` to match any decimal digit ranging from 0 through 3, or the range `[b-v]` to match any lowercase character ranging from b through v.

## Literal characters

Character	Description
<b>Alphanumeric</b>	<b>Itself</b>
<code>\0</code>	<b>The NUL character (\u0000)</b>
<code>\t</code>	<b>Tab (\u0009)</b>
<code>\n</code>	<b>Newline (\u000A)</b>
<code>\v</code>	<b>Vertical tab (\u000B)</b>
<code>\f</code>	<b>Form feed (\u000C)</b>



<code>\r</code>	Carriage return ( <code>\u000D</code> )
-----------------	---

# JavaScript - Errors & Exceptions

## Handling

- There are three types of errors in programming:
- (a) Syntax Errors
- (b) Runtime Errors, and
- (c) Logical Errors.

### Syntax Errors

Syntax errors, also called parsing errors, occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

```
<script type="text/javascript">  
    window.print();  
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

## Runtime Errors

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

```
<script type="text/javascript">  
    window.printme();  
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

## Logical Errors

Logic errors can be the most difficult type of errors to track down.

These errors are not the result of a syntax or runtime error.

Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

## The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try...catch...finally construct as well as the throw operator to handle exceptions.

You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

Here is the try...catch...finally block syntax –

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

  <!--

    try {

      // Code to run

      [break;]

    } catch ( e ) {

      // Code to run if an exception occurs

      [break;]

    } finally {

      // Code that is always executed regardless of

      // an exception occurring

    }

  //→

</script>
```

The try block must be followed by either exactly one catch block or one finally block (or one of both).

When an exception occurs in the try block, the exception is placed in e and the catch block is executed.

The optional finally block executes unconditionally after try/catch.

## Examples

Here is an example where we are trying to call a non-existing function which in turn is raising an exception.

Let us see how it behaves without try...catch–

```
<html>

<head>

  <script type="text/javascript">

    <!--

    function myFunc()

    {

      var a = 100;

      alert("Value of variable a is : " + a );

    }

    //-->

  </script>

</head>

<body>

  <p>Click the following to see the result:</p>

  <form>

    <input type="button" value="Click Me" onclick="myFunc();" />

  </form>

</body>

</html>
```

Now let us try to catch this exception using try...catch and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

```
<!DOCTYPE html>

<html>

  <head>

    <title>Page Title</title>

    <script>

      try{

        var x = ["10","20","30","40","50"];

        document.write(x);

        document.write(b);

      }catch(e){

        alert("The Exception is :" + e.message);

      }finally{

        document.write('Hello Finally !!');

      }

    </script>

  </head>

  <body></body>

</html>
```

```
<html>
```

```
<head>
```

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

```
<!--
```

```
function myFunc()
```

```
{
```

```
var a = 100;
```

```
try {
```

```
    alert("Value of variable a is : " + a );
```

```
}
```

```
catch ( e ) {
```

```
    alert("Error: " + e.description );
```

```
}
```

```
finally {
```

```
    alert("Finally block will always execute!" );
```

```
}
```

```
}
```

```
//-->
```

```
</script>
```

```
</head>

<body>

    <p>Click the following to see the result:</p>

    <form>

        <input type="button" value="Click Me" onclick="myFunc();" />

    </form>

</body>
</html>
```

## The throw Statement

- You can use a throw statement to raise your built-in exceptions or your customized exceptions.
- Later these exceptions can be captured and you can take appropriate action.
- The following example demonstrates how to use a throw statement.

```
<html>

    <head>
```

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

```
function myFunc()
```

```
{
```

```
var a = 100;
```

```
var b = 0;
```

```
var c = 0;
```

```
try{
```

```
if ( b == 0 ){
```

```
    throw( "Divide by zero error. B value should not be zero !" );
```

```
}
```

```
else
```

```
{
```

```
    c = a / b;
```

```
    document.write(c);
```

```
}
```

```
}
```

```
catch ( e ) {
```

```
    alert("Error: " + e );
```



```
    }  
    }  
  
</script>  
  
</head>  
  
<body>  
  
    <p>Click the following to see the result:</p>  
  
    <form>  
  
        <input type="button" value="Click Me" onclick="myFunc();">  
  
    </form>  
  
</body>  
  
</html>
```

- You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a try...catch block.

## JavaScript - Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button.

If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information.

This was really a lengthy process which used to put a lot of burden on the server.

- JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.
  - Basic Validation – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
  - Data Format Validation – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>

<head>

    <title>Form Validation</title>

    <script type="text/javascript">

        <!--

        // Form validation code will come here.

        //-->

    </script>

</head>

<body>
```

```
<form action="/cgi-bin/test.cgi" name="myForm" onsubmit="return(validate());">
```

```
<table cellspacing="2" cellpadding="2" border="1">
```

```
<tr>
```

```
<td align="right">Name</td>
```

```
<td><input type="text" name="Name" /></td>
```

```
</tr>
```

```
<tr>
```

```
<td align="right">EMail</td>
```

```
<td><input type="text" name="EMail" /></td>
```

```
</tr>
```

```
<tr>
```

```
<td align="right">Zip Code</td>
```

```
<td><input type="text" name="Zip" /></td>
```

```
</tr>
```

```
<tr>
```

```
<td align="right">Country</td>
```

```
<td>
```

```
<select name="Country">
```

```
<option value="-1" selected>[choose yours]</option>
```

```
<option value="1">USA</option>
```

```
<option value="2">UK</option>
```

```
        <option value="3">INDIA</option>

        </select>

    </td>

</tr>

<tr>

    <td align="right"></td>

    <td><input type="submit" value="Submit" /></td>

</tr>

</table>

</form>

</body>

</html>
```

## Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling `validate()` to validate data when `onsubmit` event is occurring. The following code shows the implementation of this `validate()` function.

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

    <!--

        // Form validation code will come here.

        function validate()

        {
```

```
if( document.myForm.Name.value == "" )  
  
{  
  
alert( "Please provide your name!" );  
  
document.myForm.Name.focus() ;  
  
return false;  
  
}
```

```
if( document.myForm.EMail.value == "" )  
  
{  
  
alert( "Please provide your Email!" );  
  
document.myForm.EMail.focus() ;  
  
return false;  
  
}
```

```
if( document.myForm.Zip.value == "" ||  
  
isNaN( document.myForm.Zip.value ) ||  
  
document.myForm.Zip.value.length != 5 )  
  
{  
  
alert( "Please provide a zip in the format #####" );  
  
document.myForm.Zip.focus() ;  
  
return false;  
  
}
```

```
if( document.myForm.Country.value == "-1" )  
  
{
```

```
        alert( "Please provide your country!" );

        return false;

    }

    return( true );

}

//-->
</script>
```

## Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address.

An email address must contain at least a '@' sign and a dot (.).

Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Try the following code for email validation.

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

    <!--

        function validateEmail()

        {

            var emailID = document.myForm.EMail.value;
```

```
        atpos = emailID.indexOf("@");

        dotpos = emailID.lastIndexOf(".");

        if (atpos < 1 || ( dotpos - atpos < 2 ))
        {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;

            return false;
        }

        return( true );
    }

    //-->
</script>
```

```
<script>

function validate(){

    var fname=document.f1.fname.value;

    var lname=document.f1.lname.value;
```

```
var status=false;

if(fname=="")
{
    document.getElementById('fnamenote').innerHTML="please enter u r first name";
    document.getElementById('fnamenote').style.fontSize=30px;

    status=false;
}
else
{
    status=true;
}

if(lname=="")
{
    document.getElementById('lnamenote').innerHTML="please enter u r last name";
    status=false;
}
else
{
    status=true;
}

return status;
```



```
}  
  
</script>  
  
<form name="f1" onsubmit="return validate()" method="post" action="abc.html">  
  
<input type='text' name="fname"/>  
  
<p id="fnamenote" style="color:red"></p>  
  
<input type='text' name="lname"/>  
  
<p id="lnamenote" style="color:red"></p>  
  
<input type='submit' value='Check Field' />  
  
</form>
```

