SYLLABUS

Subject: - Web Designing

Unit	Contents
UNIT – I	**Introduction to Internet-** World Wide Web, Internet Addressing. Browser, URL. Web server, website, homepage, Domain Name. Basic concepts. **Softwares for Web Designing** - Notepad/Notepad++, Dreamweaver, Blue Griffon, Net beans, Sea Monkey, Word press, Sublime. **Introduction to HTML:** HTML Tags and Attributes, HTML Basic Tags, Formatting Tags, HTML Color Coding, Div and Span Tags for Grouping Lists: Unordered Lists, Ordered Lists, Definition list. Images: Image and Image Mapping **Hyperlink: URL** - Uniform Resource Locator, URL Encoding. Table: <table><th>, <tr>, <td>, <caption>, <thead>, <tbody>, <tfoot>, <colgroup>, <col> Attributes Using Iframe as the Target **Form** <input>, <textarea>, <button>, <select>, <label> **Headers:** Title, Base, Link, Styles, Script HTML Meta Tag, XHTML, HTML Deprecated Tags & Attributes
UNIT – II	**CSS:** Introduction, Features and benefits of CSS, CSS Syntax, External Style Sheet using <link>, Multiple Style Sheets, Value Lengths and Percentages. **Selectors:** ID Selectors. Class Selectors, Grouping Selectors, Universal Selector, Descendant / Child Selectors, Attribute Selectors, CSS-Pseudo Classes. **Color Background Cursor:** background-image, background-repeat, background position, CSS Cursor **Text Fonts:** color, background-color, text-decoration, text-align, vertical-align, text-indent, text-transform, white-space, letter-spacing, word-spacing, line-height, font-family, font-size, font-style, font-variant, font-weight.
UNIT – III	**Lists Tables:** List-style-type, list-style-position, list-style-image, list-style, CSS Tables (border, width & height, text-align, vertical-align, padding, color) **Box Model:** Borders & Outline, Margin & Padding, Height and width, CSS Dimensions. **Display Positioning:** CSS Visibility, CSS Display, CSS Scrollbars, CSS Positioning (Static Positioning, Fixed Positioning, Relative Positioning, Absolute Positioning), CSS Layers with Z-Index. **Floats:** The float Property. The clear Property, The clearfix Hack.
UNIT – IV	**The JavaScript:** Nature of JavaScript, Script Writing Basics, Enhancing HTML Documents with JavaScript, The Building Blocks. Introduction to JavaScript, JavaScript Engines, Values, Variables and Operators, Variable Mutation, Basic Operators, Operator Precedence, JavaScript Types. Types Definition, Types in JavaScript, Objects, Type Conversion and Coercion, Static vs Dynamic Type Checking. **JavaScript Conditionals:** Introduction to Conditionals, Conditionals in JavaScript, Ternary Operators and Conditionals. Conditional Ladder & Switch statement. **JavaScript Arrays:** Introduction to Arrays, Declaring and Mutating Arrays, Array Methods and Properties, Replication with Array Methods, Multi-dimensional Arrays.
UNIT –V	**JavaScript Loops:** Introduction to Loops, Loops in JavaScript, While and Do/While Loops, For Loops, Break and Continue in Loops, Iterating Arrays, Iterating Objects. **JavaScript Functions:** Introduction to Functions, Functions in JavaScript, Nested Functions in JavaScript, Arrow Functions in JavaScript, Function as an Argument, Function as the Returned Object, **JavaScript Scope:** Scope Introduction, Scope in JavaScript, Lexical Scope. Module

	Scope. Method of Adding Interactivity to a Web Page, Creating Dynamic Web Pages, Concept of Java Scripting the Forms. Java Scripting the Forms, Basic Script Construction, Talking to the Form Objects. Organizing the Objects and Scripts, Field-Level Validation, Check Required Fields like Validating Zip Code, Automated Formatting, Format Phone, For Money, Automatic Calculation, Calculate Expiration Date, Calculate Amount etc.

UNIT -1

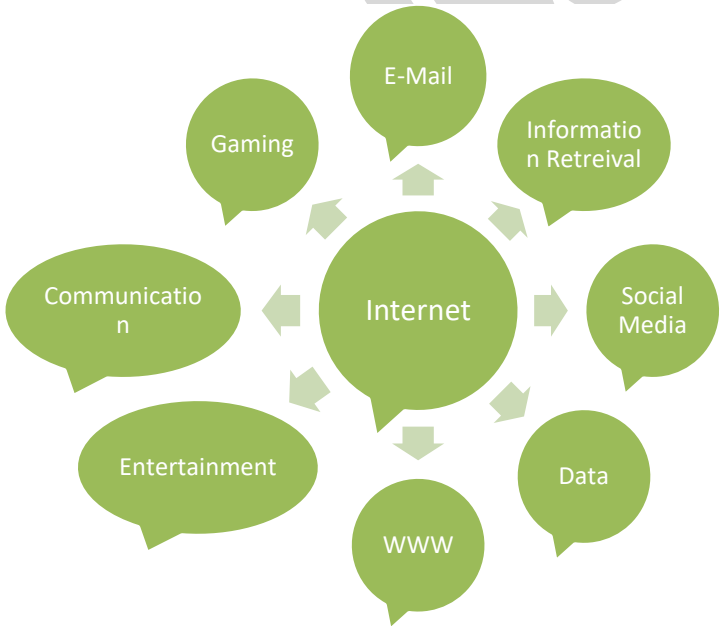
Introduction to Internet

The Internet, also known as the Net, is a worldwide network of computers or network of network that are interlinked. Over Internet information is spread over web page. The Internet provides many online services like information retrieval, email, gaming, chat etc.

Definition

Federal Networking Council (FNC) in October 1995 defined the term "Internet". According to FNC "Internet" refers to the global information system that -

I. It Is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions
II. Is able to support communications using the Transmission Control Protocol/ Internet Protocol (TCP/IP) suite or its subsequent extensions and/ or other IP-compatible protocols; and
III. It provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.



WEB PAGE

A **web page** is collection of text, graphics, video and audio. It contains hyperlink that connects to next page of website. A web page is commonly written in HTML (Hypertext Markup Language) that is accessible through the Internet or other networks using an Internet browser (Google Chrome,

Mozilla, opera etc.). The name "web page" in a website is similar to paper pages that are bound together into a book.

WEBSITE

A **website** represents a centrally managed collection of web pages, containing group of text images and all types of multi-media files presented to the attention of the Internet users in a visual and easily accessible way. The Website gives users access to a vast array of documents that are connected to each other by means of hypertext or hypermedia links—i.e., hyperlinks, electronic connections that link related pieces of information in order to allow a user easy access to them.

STATIC WEBSITE

Static, or 'fixed', websites are the most simplistic. Their content does not change depending on the user, and is not regularly updated. Static websites are built using simple HTML code, and typically provide information.

DYNAMIC WEBSITE

A dynamic website or web page will display different content each time it is visited. Examples include blogs and eCommerce sites, or generally any site that is updated regularly. Dynamic websites can also be set up to show different content to different users, at different times of the day etc. Dynamic websites make for a more personal and interactive experience for the user, although they can be a little more complex to develop and may load slightly slower than static ones.

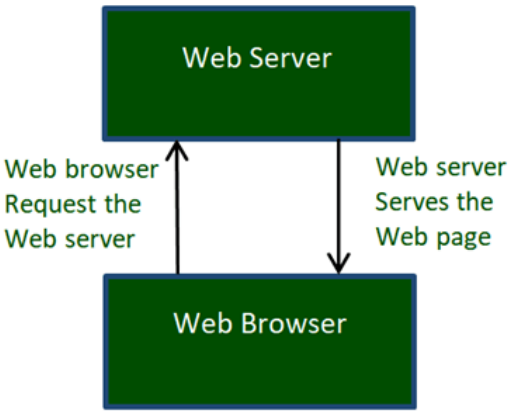
WORLD WIDE WEB ("WWW'

The **World Wide Web** ("WWW', "Web" or "W3") was begun in 1989 by Tim Berners-Lee and his colleagues at CERN, an international scientific organization based in Geneva, Switzerland. They created a protocol, Hyper Text Transfer Protocol (HTTP), which standardized communication between servers and clients. Their text-based Web browser was made available for general release in January 1992.

World Wide Web, which is also known as a Web, is **a collection of websites or web pages stored in web servers and connected to local computers through the internet**. Users can access the content of these sites from any part of the world over the internet using their devices such as computers, laptops, cell phones, etc.

WEB BROWSER

Web browser is a software application used to locate, retrieve, and display content on the World Wide Web, including web pages, images, videos, and other files. It means that the web browser asks for data from the website and the web server sends the information back to the browser, which then displays the results on the Internet-enabled device. Some common web browsers are Google Chrome, Mozilla Firefox, Internet Explorer, Safari, etc



WEB SERVER

A **WEB SERVER** is a software and hardware that is used to store and deliver the website content. Content could be anything from images, texts, application data, videos, and many more as per the request of the web browser. Basically, when a browser asks for information, then the process goes through many steps. The person specifies the URL in the address bar of the web browser, and then the web browser gets the IP address of the domain name. This is done by either translating the URL using the Domain Name System or DNS, or it is done by searching through the cache. This brings the browser to the web server. The web server then responds and sends the requested page to the browser.

Some uses of web server

- Send and receive emails.
- Download the file transfer protocol or FTP request
- Build and publish web pages.

WEBSITE ADDRESS

A website address, also known as a URL (uniform resource locator), is an Internet or intranet name that points to to a location where a file, directory or website page is hosted

ABSOLUTE AND RELATIVE ADDRESS

It always begins with the protocol name. An absolute reference is the complete address to a web page including http://www, just as you'd use in the browser's address bar Absolute references are used to refer to a site somewhere else on the Internet.

A relative URL is any URL that doesn't explicitly specify the protocol (e.g., “http:// " or " https:// ") and/or domain (www.example.com), which forces the visitor's web browser (or the search engine bots) to assume it refers to the same site on which the URL appears

Hyper Text Markup Language (HTML)

HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the

Web. HTML uses "markup" to annotate text, images, and other content for display in a Web browser.

An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case insensitive. That is, it can be written in uppercase, lowercase, or a mixture. For example, the <title> tag can be written as <Title>, <TITLE>, or in any other way.

<html>

This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.

<head>

This tag represents the document's header which can keep other HTML tags like <title>, <link> etc

<title>

The <title> tag is used inside the <head> tag to mention the document title

<body>

This tag represents the document's body which keeps other HTML tags like <h1>, , etc

The <body> tag has 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.

Heading Tags

Any document 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>. While displaying any heading, browser adds one line before and one line after that heading.

Paragraph Tag

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag.

Line Break Tag

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

Centering Content

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

Bold Text

Anything that appears within ... element, is displayed in bold

Italic Text

Anything that appears within ... element is displayed in italicized

Underlined Text

Anything that appears within <u>...</u> element, is displayed with underline

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 list of attributes and it has no closing tag.

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 HTML <ul> tag. Each item in the list is marked with a bullet.

HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted, then 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>.

HTML Definition Lists

HTML supports 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.

Definition List makes use of following three tags.

- <dl> − Defines the start of the list
- <dt> − A term

- <dd> − Term definition
- </dl> − Defines the end of the list

HTML - Tables

The HTML tables allow web authors to arrange data like text, images, links, other tables, 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. The elements under <td> are regular and left aligned by default

Table Tags

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. Headings, which are defined in <th> tag are centered and bold by default.

The <**caption**> tag defines a table caption. The <caption> tag must be inserted immediately after the <**table**> tag.

The <**tbody**> tag is used to group the body content in an HTML table. The <tbody> element is generally used with the <thead> and <tfoot>elements to specify each part of a table (body, header, footer).

The <**tfoot**> tag is used to group footer content in an HTML table.

The <**thead**> tag is used to group header content in an HTML table.

The **<colgroup>** tag used for formatting columns in a table, The <colgroup> tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row. <colgroup> tag must be a child of a <table> element, after any <caption> elements and before any <thead>, <tbody>, <tfoot>, and <tr> elements.

The **<col>** tag specifies column properties for each column within a <colgroup> element. The <col> tag is useful for applying styles to entire columns, instead of repeating the styles for each cell, for each row.

<iframe>

The <iframe> tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document.

<style>

The <style> tag is used to define style information (CSS) for a document. Inside the <style> element we specify how HTML elements should render in a browser.When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet. If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used.

<form>

The HTML <form> tag is **used for creating a form for user input**. A form can contain textfields, checkboxes, radio-buttons and more. Forms are used to pass user-data to a specified URL.

- The <input> element is the most important form element. It specifies an input field where the user can enter data. The <input> element can be displayed in several ways, depending on the type attribute. E.g. <input type="button">, <input type="checkbox">
- The <label> tag defines a label for several elements: The <label> tag is used to specify a label for an <input> element of a form. It adds a label to a form control such as text, email, password, textarea etc. It toggles the control when a user clicks on a text within the <label> element.
- The <button> HTML element represents **a clickable button, used to submit forms or anywhere in a document for accessible, standard button functionality**.
- <textarea>: The Textarea element. The <textarea> HTML element represents **a multi-line plain-text editing control**, useful when you want to allow users to enter a sizeable amount of free-form text, for example a comment on a review or feedback form. A text area can hold an unlimited number of characters. The size of a text area is specified by the <cols> and <rows>
- The <select> element is used to create a drop-down list. The <select> element is most often used in a form, to collect user input. The <option> tags inside the <select> element define the available options in the drop-down list.

UNIT –2

CSS Introduction

- CSS treats each HTML element as if it appears inside its own box and uses rules to indicate how that element should look.
- Rules are made up of selectors (that specify the elements the rule applies to) and declarations (that indicate what these elements should look like).
- Different types of selectors allow you to target your rules at different elements.
- Declarations are made up of two parts: the properties of the element that you want to change, and the values of those properties. For example, the font-family property sets the choice of font, and the value arial specifies Arial as the preferred typeface.
- CSS rules usually appear in a separate document, although they may appear within an HTML page.

CSS features,

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

CSS works by associating rules with HTML elements. These rules govern how the content of specified elements should be displayed. A CSS rule contains two parts: a selector and a declaration. CSS Associates Style rules with HTML elements

P (Selector)

{ font-family: Arial;} (Declaration)

This rule indicates that all elements should be shown in the Arial typeface
Selectors indicate which element the rule applies to. The same rule can apply to more than one element if you separate the element names with commas.

Declarations indicate how the elements referred to in the selector should be styled. Declarations are split into two parts (a property and a value), and are separated by a colon.

CSS declarations sit inside curly brackets and each is made up of two parts: a property and a value, separated by a colon. You can specify several properties in one declaration, each separated by a semi-colon. CSS Properties Affect How Elements Are Displayed

h1, h2, h3⟶ (selector)

{font-family: Arial; color: yellow;}

(Property) (Value)

This rule indicates that all <h>, <h2>and <h3> elements should be shown in the Arial typeface, in a yellow color.

Properties indicate the aspects of the element you want to change. For example, color, font, width, height and border.

Values specify the settings you want to use for the chosen properties. For example, if you want to specify a color property then the value is the color you want the text in these elements to be.

External CSS

<link> element in HTML document to tell the browser where to find the CSS file used to style the page. It is an empty element (meaning it does not need a closing tag), and it lives inside the <head> element. It should use three attributes:

```
<head>
  <link rel="stylesheet" href="styles.css" type=”text/css”>
</head>
```

href This specifies the path to the CSS file (which is often placed in a folder called css or styles).

type This attribute specifies the type of document being linked to. The value should be text/css.

rel This specifies the relationship between the HTML page and the file it is linked to. The value should be stylesheet when linking to a CSS file.

An HTML page can use more than one CSS style sheet. To do this it could have a <link> element for every CSS file it uses. For example, some authors use one CSS file to control the presentation (such as fonts and colors) and a second to control the layout.

Internal CSS

<style>

You can also include CSS rules within an HTML page by placing hem inside a <style> element, which usually sits inside the <head> element of the page. The <style> element should use the type attribute to indicate that the styles are specified in CSS. The value should be text/css.

- When building a site with more than one page, you should use an external CSS style sheet. This:
- Allows all pages to use the same style rules (rather than repeating them in each page).
- Keeps the content separate from how the page looks.
- Means you can change the styles used across all pages by altering just one file (rather than each individual page).

Selectors

Selector	Meaning	Example
Universal Selector	Applies to all elements in the document	*** {}** Targets all elements on the page
Type Selector	Matches element names	**h1, h2, h3 {}** Targets the **<h1>**, **<h2>** and **<h3>** elements
Class Selector	Matches an element whose class attribute has a value that matches the one specified after the period (or full stop) symbol	.note {} Targets any element whose class attribute has a value of note p.note {} Targets only <p> elements whose class attribute has a value of note
ID selector	Matches an element whose id attribute has a value that matches the one specified after the pound or hash symbol	**#introduction {}** Targets the element whose **id** attribute has a value of **introduction**
Child Selector	Matches an element that is a direct child of another	**li>a {}** Targets any **<a>** elements that are children of an ** ** element (but not other **<a>** elements in the page)
Descendant Selector	Matches an element that is a descendent of another specified element (not just a direct child of that element)	**p a {}** Targets any **<a>** elements that sit inside a **<p>** element, even if there are other elements nested between them
Adjacent Sibling Selector	Matches an element that is the nest sibling of another	**h1+p {}** Targets the first **<p>** element after any **<h1>** element (but not other **<p>** elements)
General Sibling Selector	Matches an element that is a sibling of another, although it does not have to be the directly	**h1~p {}**

	preceding element	If you had two <**p**> elements that are siblings of an <**h1**> element, this rule would apply to both

Advantages to placing your CSS rules in a separate style sheet

- All of your web pages can share the same style sheet. This is achieved by using the **<link>** element on each HTML page of your site to link to the same CSS document. This means that the same code does not need to be repeated in every page (which results in less code and smaller HTML pages).
- Therefore, once the user has downloaded the CSS stylesheet, the rest of the site will load faster. If you want to make a change to how your site appears, you only need to edit the one CSS file and all of your pages will be updated. For example, you can change the style of every **<h1>** element by altering
- The HTML code will be easier to read and edit because it does not have lots of CSS rules in the same document. It is generally considered good practice to have the content of the site separated from the rules that determine how it appears

Advantages to placing CSS rules in the same page as your HTML code

- If you are just creating a single page, you might decide to put the rules in the same file to keep everything in one place. (However, many authors would consider it better practice to keep the CSS in a separate file.)
- If you have one page which requires a few extra rules (that are not used by the rest of the site), you might consider using CSS in the same page. (Again, most authors consider it better practice to keep all CSS rules in a separate file.)
- Most of the examples in this book place the CSS rules in the **<head>** of the document (using the **<style>** element) rather than a separate document. This is simply to save you opening two files to see how the CSS examples work

CSS Properties

The background-image property sets one or more background images for an element. By default, a background-image is placed at the top-left corner of an element, and repeated both vertically and horizontally.

The background-repeat property sets if/how a background image will be repeated. By default, background-image is repeated both vertically and horizontally.

repeat-x	The background image is repeated only horizontally
repeat-y	The background image is repeated only vertically
no-repeat	The background-image is not repeated. The image will only be shown once
repeat	The background image is repeated both vertically and horizontally. The last image will be clipped if it does not fit. This is default

The background-position property sets the starting position of a background image.

Positions can be

left top| left center| left bottom |right top|right center|right bottom|center top|center center|center bottom

```
body {
  background-image: url('RCCM.gif');
  background-repeat: no-repeat;
  background-position: center;
}
```

CSS cursor

The cursor property specifies the mouse cursor to be displayed when pointing over an element.

```
cursor example
<html>
<head>
<style>
.auto {cursor: auto;}
.cell {cursor: cell;}
.col-resize {cursor: col-resize;}
.help {cursor: help;}
</style>
</head>
<body>
<p class="auto">auto</p>
<p class="cell">cell</p>
<p class="col-resize">col-resize</p>
<p class="help">help</p>
</body>
</html>
```

The background-color property sets the background color of an element.
The text-align property specifies the horizontal alignment of text in an element.

Syntax

text-align: left|right|center|justify;

The text-transform property controls the capitalization of text.

Syntax

text-transform: none|capitalize|uppercase|lowercase;

The vertical-align property sets the vertical alignment of an element.

Syntax

vertical-align: baseline|*length*|sub|super|top|text-top|middle|bottom|text-bottom;

The text-indent property specifies the indentation of the first line in a text-block.

The text-decoration property specifies the decoration added to text, and is a shorthand property for:

- text-decoration-line (required)
  - Sets the kind of text decoration to use (like underline, overline, line-through)
- text-decoration-color
  - Sets the color of the text decoration
- text-decoration-style
  - Sets the style of the text decoration (like solid, wavy, dotted, dashed, double)

The font property is a shorthand property for:

- font-style
- font-variant
- font-weight
- font-size/line-height
- font-family

The font-size and font-family values are required. If one of the other values is missing, their default value are used.

The font-family property specifies the font for an element.

Syntax

font-family: *family-name*|*generic-family*

The font-size property sets the size of a font.

Syntax

font-size:medium|xx-small|x-small|small|large|x-large|xx-large|smaller|larger|*length*|

The font-style property specifies the font style for a text.

Syntax

font-style: normal|italic|

The font-variant property specifies whether or not a text should be displayed in a small-caps font.

Syntax

font-variant: normal|small-caps|

The font-weight property sets how thick or thin characters in text should be displayed.

Syntax

font-weight: normal|bold|bolder|lighter|*number*

The white-space property specifies how white-space inside an element is handled.

Syntax

white-space: normal|nowrap|pre|pre-line|pre-wrap

The letter-spacing property increases or decreases the space between characters in a text.

Syntax

letter-spacing: normal|*length*

The word-spacing property increases or decreases the white space between words.

Syntax

word-spacing: normal|*length*|

The line-height property specifies the height of a line.

Syntax

line-height: normal|*number*|*length*

UNIT -3

List Tables

The list-style-type property specifies the type of list item marker.

The list-style-type: none property can also be used to remove the markers/bullets.

The list-style-image property specifies an image as the list item marker:

The list-style-position property specifies the position of the list-item markers (bullet points).

"list-style-position: outside;" means that the bullet points will be outside the list item. The start of each line of a list item will be aligned vertically. This is default:

"list-style-position: inside;" means that the bullet points will be inside the list item. As it is part of the list item, it will be part of the text and push the text at the start:

The list-style property is a shorthand property. It is used to set all the list properties in one declaration:

When using the shorthand property, the orders of the property values are:

- list-style-type (if a list-style-image is specified, the value of this property will be displayed if the image for some reason cannot be displayed)
- list-style-position (specifies whether the list-item markers should appear inside or outside the content flow)
- list-style-image (specifies an image as the list item marker)

If one of the property values above are missing, the default value for the missing property will be inserted, if any.

CSS Tables

To specify table borders in CSS, use the border property.

The border-collapse property sets whether the table borders should be collapsed into a single border:

The width and height of a table are defined by the width and height properties.

The text-align property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>. By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.

To center-align the content of <td> elements as well, use text-align: center:

To left-align the content, force the alignment of <th> elements to be left-aligned, with the text-align: left property:

The vertical-align property sets the vertical alignment (like top, bottom, or middle) of the content in <th> or <td>. By default, the vertical alignment of the content in a table is middle (for both <th> and <td> elements).

To control the space between the border and the content in a table, use the padding property on <td> and <th> elements:

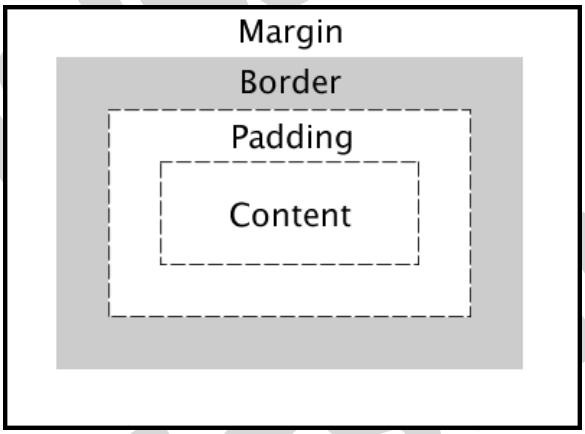
Use the :hover selector on <tr> to highlight table rows on mouse over:

To set background color set background-color property in table To set foreground color set color: property in table;

Box Model

The CSS Box Model is used **to create a definition for the way the HTML elements are organized on the screen**. In CSS, the term "box model" is used when talking about design and layout. The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and all the properties that manipulate them. Each element can be thought of as having its own box.

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent



CSS has two box-models. **content-box and border-box** . content-box is the default.

Height and Width

When you set the width and height properties of an element with CSS, you just set the width and height of the content area. To calculate the full size of an element, you must also add padding, borders and margins.

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

You can use the visibility property to control whether an element is visible or not. This property can take one of the following values listed in the table below:

Value	Description
visible	Default value. The box and its contents are visible.
hidden	The box and its content are invisible, but still affect the layout of the page.
collapse	This value causes the entire row or column to be removed from the display. This value is used for row, row group, column, and column group elements.
inherit	Specifies that the value of the visibility property should be inherited from the parent element i.e. takes the same visibility value as specified for its parent.

Display Positioning

The display property specifies the display behavior (the type of rendering box) of an element.

In HTML, the default display property value is taken from the HTML specifications or from the browser/user default style sheet.

The position property specifies the type of positioning method used for an element.

position: static| relative |fixed |absolute |sticky

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

position: static;

HTML elements are positioned static by default. Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

Position: relative;

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element. This element has position: relative;

Position: fixed;

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Position: absolute;

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Position: sticky;

An element with position: sticky; is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

CSS Scrollbars

They can be styled

-webkit-scrollbar

webkit-scrollbar-track — the track (**progress bar**) of the scrollbar

```
webkit-scrollbar-track {
  background: black;
}
 /* Handle */
::-webkit-scrollbar-thumb {
  background: pink;
}
/* Handle on hover */
```

```
::-webkit-scrollbar-thumb:hover {
  background: red;
}
```

CSS LAYERS with Z-index

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 webpage 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>
```

Float Property

The float property specifies whether an element should float to the left, right, or not at all.

float: none|left|right

```
e.g.img  {
  float: right;
}
```

clear property

The clear property controls the flow next to floated elements.
The clear property specifies what should happen with the element that is next to a floating element.

```
img {
  float: left;
}
p.clear {
  clear: left;
}
```

The <p> element is pushed below left floated elements (the <p> element do not allow floating elements on the left side):

clearfix

Elements after a floating element will flow around it. Use the "clearfix" hack to fix the problem. If an element is taller than the element containing it, and it is floated, it will overflow outside of its container.

Then we can add overflow: auto; to the containing element to fix this problem:
The overflow: auto clearfix works well as long as you are able to keep control of your margins and padding

UNIT-4 and 5

JavaScript Introduction

Javascript is a dynamic computer programming language. It 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.

JavaScript is a **cross-platform, object-oriented scripting language used to make webpages interactive** (e.g., having complex animations, clickable buttons, popup menus, etc.).can be used both on the client-side **and server-side**

- 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.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

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 e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are:

- **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.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

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 you 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 syntax will look as follows.

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

   JavaScript code

</script>
```

Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-- >". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

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

This code will produce the following result:

Hello World!

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. But when formatted in a single line as follows, you must use semicolons:

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 //-->.

Example

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>
```

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 include 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>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This code will produce the following results:

Click here for the result

Say Hello

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>
<p>This is web page body </p>
</body>
</html>
```

This code will produce the following results:

```
Hello World
This is web page body
```

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>
```

This code will produce the following result.

HelloWorld

Say Hello

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 be 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 Datatypes

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**, e.g., 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>
```

Note: Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

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.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable function
checkscope( ) {
var myVar = "local"; // Declare a local variable
document.write(myVar);
}
//-->
```

</script>

It will produce 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 are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	Instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

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:

S. No.	Operator and Description
1	**+ (Addition)** Adds two operands **Ex:** A + B will give 30
2	**- (Subtraction)** Subtracts the second operand from the first **Ex:** A - B will give -10
3	*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
4	**/ (Division)** Divide the numerator by the denominator **Ex:** B / A will give 2
5	Outputs the remainder of an integer division **Ex:** B % A will give 0
6	**++ (Increment)** Increases an integer value by one **Ex:** A++ will give 11
7	**-- (Decrement)** Decreases an integer value by one **Ex:** A-- will give 9

Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

The following code shows how to use arithmetic operators in JavaScript.

```
<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>
```

```
<p>Set the variables to different values and then try...</p>
</body>
</html>
```

Output

a + b = 43

a - b = 23

a / b = 3.3

a % b = 3

a + b + c = 43Test

a++ = 33

b-- = 10

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:

S.No	Operator and Description
1	**== (Equal)** Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
2	**!= (Not Equal)** Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
3	**> (Greater than)** Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true. **< (Less than)**
4	Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
	>= (Greater than or Equal to)
5	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. **Ex:** (A >= B) is not true. **<= (Less than or Equal to)**
6	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.

	Ex: (A <= B) is true.

Example

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>
<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) => 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:

S.No	Operator and Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example

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...

Bitwise Operators

JavaScript supports the following bitwise operators:

Assume variable A holds 2 and variable B holds 3, then:

S.No	Operator and Description
1	**& (Bitwise AND)** It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.
2	**| (BitWise OR)** It performs a Boolean OR operation on each bit of its integer arguments. **Ex:** (A | B) is 3.
3	**^ (Bitwise XOR)** It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both. **Ex:** (A ^ B) is 1.
4	**~ (Bitwise Not)** It is a unary operator and operates by reversing all the bits in the operand. **Ex:** (~B) is -4.
5	**<< (Left Shift)** It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. **Ex:** (A << 1) is 4.
6	**>> (Right Shift)** Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. **Ex:** (A >> 1) is 1.

>>> (Right shift with Zero)

7 This operator is just like the >> operator, except that the bits shifted in on the left are always zero.
Ex: (A >>> 1) is 1.

Example

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
<body>
<script type="text/javascript">
<!--
var a = 2;        // Bit presentation 10
var b = 3;        // Bit presentation 11
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("(~b) => ");
result = (~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) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and then try...
```

Assignment Operators

JavaScript supports the following assignment operators:

S.No	Operator and Description
1	**= (Simple Assignment)** 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
	+= (Add and Assignment)

2	It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**-= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note: Same logic applies to Bitwise operators, so they will become <<=, >>=, >>=, &=, |= and ^=.

Example

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...
```

Miscellaneous Operators

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator (? :)** and the **typeof operator**.

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.

S.No	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

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.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.

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 times, you will use comparison operators while making decisions.

Example

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

The syntax of an **if-else** statement is as follows:

```
if (expression){
Statement(s) to be executed if expression is true
}else{
Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

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.

Example

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...

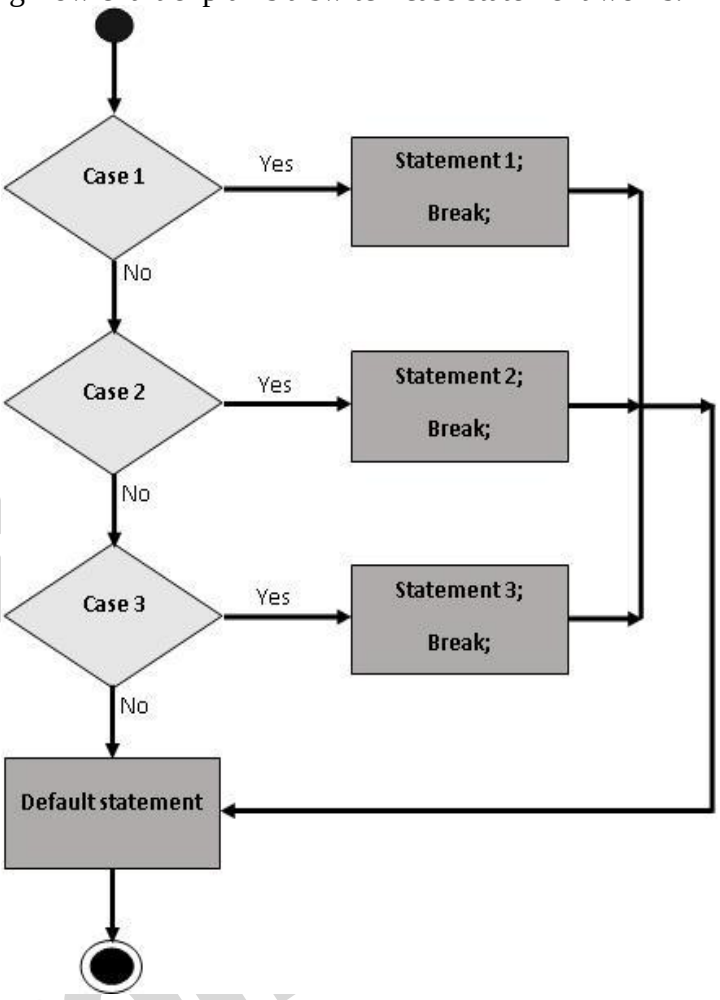
You can use multiple **if...else…if** statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Switch

Switch statement handles multiway branching 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)

}

Break statements play a major role in switch-case statements. 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.

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.

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.

Syntax

The syntax of **while loop** in JavaScript is as follows:

while (expression)

{

Statement(s) to be executed if expression is true

}

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**.

The syntax for **do-while** loop in JavaScript is as follows:

do{

Statement(s) to be executed;

} while (expression);

Note: Don't miss the semicolon used at the end of the **do...while** loop.

The 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 is JavaScript is as follows:

```
for (initialization; test condition; iteration statement){ Statement(s) to be
executed if test condition is true

}
```

for...in

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

The syntax of 'for..in' loop is:

```
for (variablename in object){

statement or block to execute

}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Break and continue

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 at 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 look.

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, used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

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.

Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely. A **label** is simply an identifier followed by a colon (:) that is applied to a statement or a block of code.

Note: Line breaks are not allowed between the '**continue**' or '**break**' statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Function

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. Functions allow a programmer to divide a big program into a number of small and manageable functions.

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. We were using these functions again and again, but they had been written in core JavaScript only once.

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 [redacted]
JavaScript is b[redacted]
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>
```

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>
<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>
```

Output

Click the following button to call the function

Say Hello

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.

Example

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>
```

Output

Click the following button to call the function

Say Hello

Use different parameters inside the function and then try...

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.

Example

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>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

There is a lot to learn about JavaScript functions, however we have covered the most important concepts in this tutorial.

Nested Functions

Earlier function definition was allowed only in top level global code, but now JavaScript allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the function statement.

Example

Try the following example to learn how to implement nested functions.

```
<html>
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
function square(x) { return x*x; }
return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
var result;
result = hypotenuse(1,2);
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>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

Function () Constructor

The *function* statement is not the only way to define a new function; you can define your function dynamically using **Function()** constructor along with the **new** operator.

Syntax

Following is the syntax to create a function using **Function()** constructor along with the **new** operator.

```
<script type="text/javascript">
 <!--
 var variablename = new Function(Arg1, Arg2..., "Function Body");
 //-->
 </script>
```

The **Function()** constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the **Function()** constructor is not passed any argument that specifies a name for the function it creates. The **unnamed** functions created with the **Function()** constructor are called **anonymous** functions.

Function Literals

The concept of **function literals** is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

The syntax for a **function literal** is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.

```
<script type="text/javascript">
<!--
var variablename = function(Argument List){
Function Body
};
//-->
</script>
```

Syntactically, you can specify a function name while creating a literal function as follows.

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

var variablename = function FunctionName(Argument List){Function Body};

//-->

</script>

But this name does not have any significance, so it is not worthwhile.

ARRAY

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 Properties

Here is a list of the properties of the Array object along with their description.

Property	Description
constructor	Returns a reference to the array function that created the object.
index	The property represents the zero-based index of the match in the string
input	This property is only present in arrays created by regular expression matches.

length	Reflects the number of elements in an array.
prototype	The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

Java Script Scope

Scope determines the accessibility (visibility) of variables, objects, and functions from different parts of the code.

JavaScript has 3 types of scope:

- Block scope
- Function scope
- Global scope

Block Scope

These two keywords let and const provide Block Scope in JavaScript. Variables declared inside a { } block cannot be accessed from outside the block:

Variables declared with the var keyword can NOT have block scope. Variables declared inside a { } block can be accessed from outside the block.

Local Scope

Variables declared within a JavaScript function, become LOCAL to the function.

Local variables have Function Scope: They can only be accessed from within the function. Since local variables are only recognized inside their functions, variables with the same name can be used in different functions. Local variables are created when a function starts, and deleted when the function is completed.

Function Scope

JavaScript has function scope: Each function creates a new scope. Variables defined inside a function are not accessible (visible) from outside the function. Variables declared with var, let and const are quite similar when declared inside a function. They all have Function Scope:

Global JavaScript Variables

A variable declared outside a function, becomes GLOBAL.

A global variable has Global Scope: All scripts and functions on a web page can access it.

Global Scope

Variables declared Globally (outside any function) have Global Scope. Global variables can be accessed from anywhere in a JavaScript program. Variables declared with var, let and const are quite similar when declared outside a block.

They all have Global Scope:

```
var x = 2;       // Global scope
```

```
let x = 2;       // Global scope

const x = 2;       // Global scope
```

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.