UI Technologies

(HTML, CSS & JavaScript)

Including ES6 Features

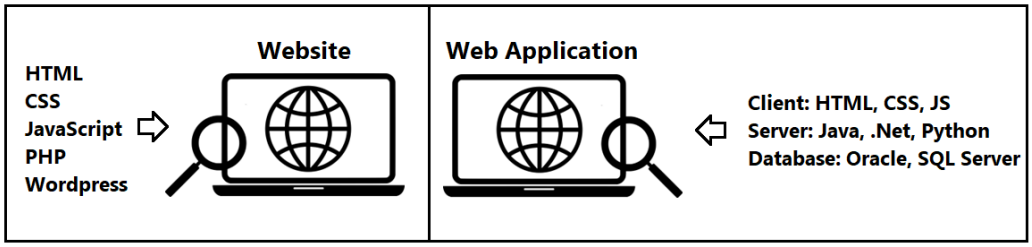
CoderStar

Contents

Web Technologies

Web technologies:

- Every Web developer or Full-Stack developer must know about HTML, CSS and JavaScript.
- Web technologies are used to develop
  - Complete static website
  - User Interface of Web application

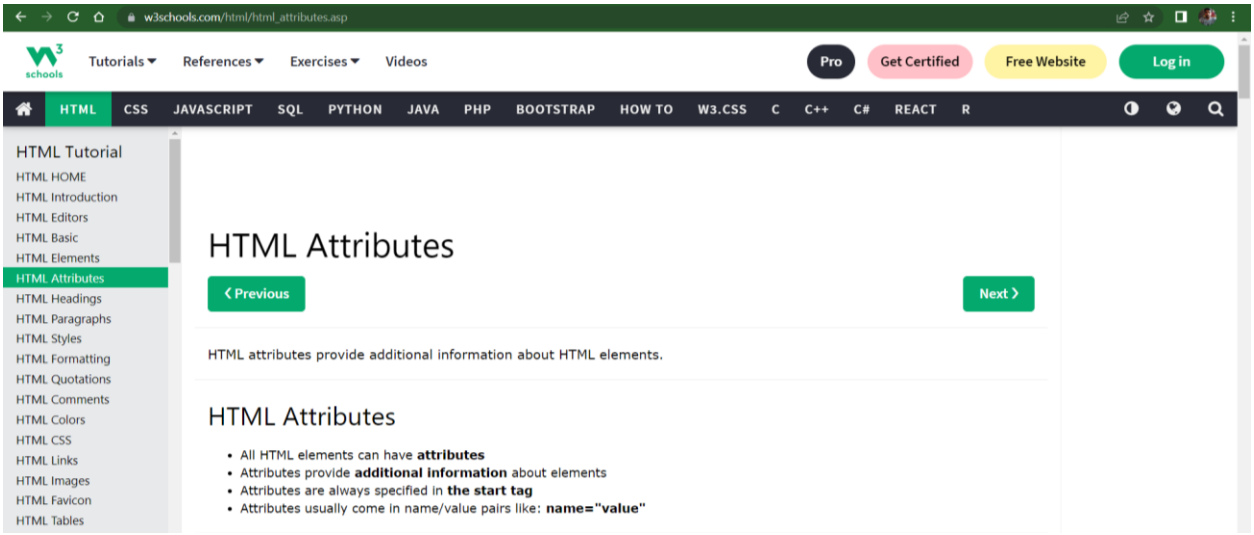


Static HTML Page:

- Static web page has fixed content to all users(customers).

Static Website:

- Static websites contain all static pages only.
- We create static websites using HTML, CSS and JavaScript only.
- We cannot use any server-side programming to create static website
- Examples: tutorial websites like
  - www.html.com
  - www.w3schools.com
  - www.javatpoint.com

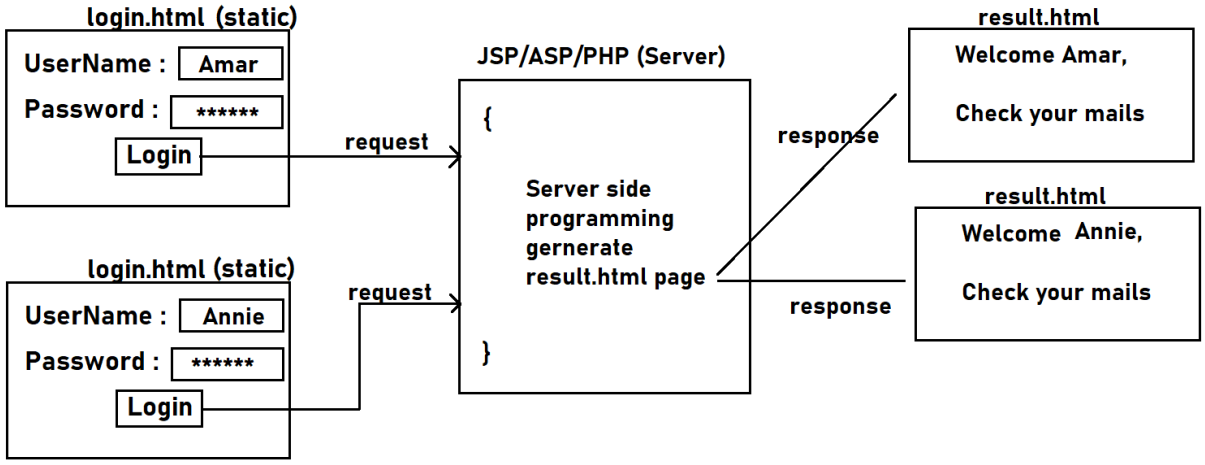


Dynamic HTML Page:

- Dynamic web page content different from customer to customer.
- Server-side programming generates dynamic web page.

Web application:

- An application deployed in server and run from multiple clients in network.
- Web application generates dynamic HTML pages
- **For example:** In Gmail application Login page is Static. When different user's login with their details, inbox page appears for differently for users.



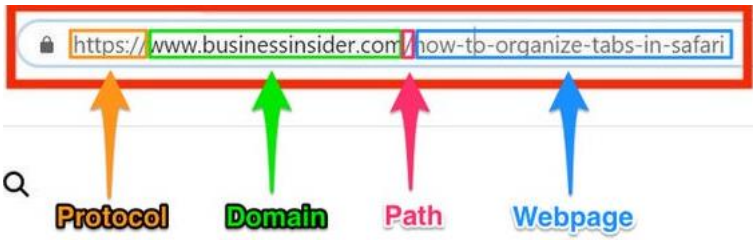
Define Client?

- A user system connected to network.
- Client system must install with Web browser to browse web pages.

Define Web browser?

- **Web browser** is a software by which we can access any web page or web application in World Wide Web.
- **Examples,** Google Chrome, Mozilla etc.

URL(Uniform resource location**):** A unique reference of Web application or Web page



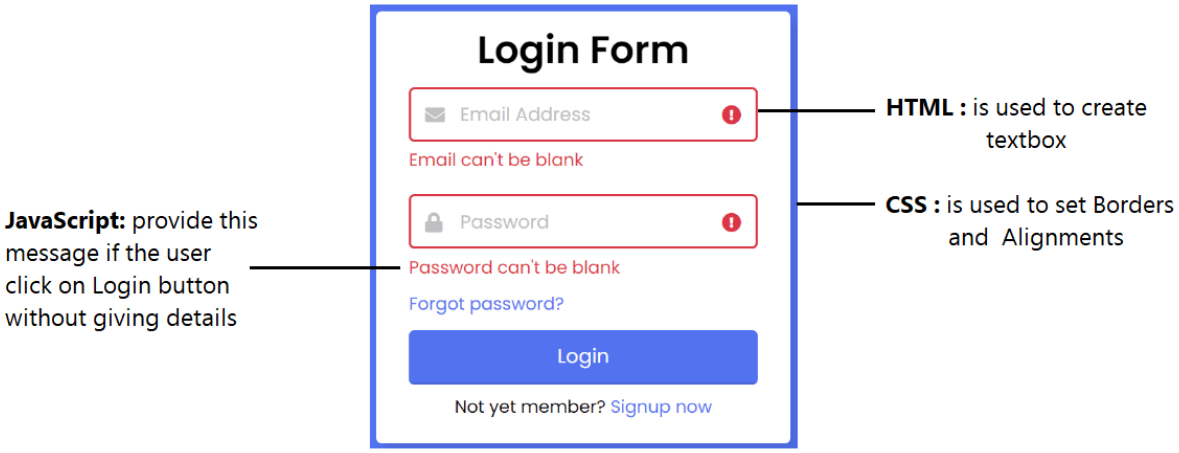
Client-Side Technologies and Frameworks

Client-side technologies:

1. **HTML:** HTML is used to create plain web pages using components such as labels, text boxes, buttons, radio button, check boxes, images, hyperlinks etc.

2. **CSS:** CSS is used apply styles to web pages such as borders, text colors, backgrounds, layouts etc.

3. **JavaScript:** JavaScript is used to validate web forms as well as providing dynamic functionality to web pages.



Client-Side Frameworks: The client-side framework is usually a JavaScript library and runs in a Web browser. Every client-side framework is different from the other in its use and its function.

Angular: It was designed by Google by which developers can build front-end-based apps without using any other plugins or frameworks.

React: developed by Facebook helping web developers build user interfaces specifically for single-page applications.

Bootstrap: an HTML, CSS and JavaScript framework that developers can use to build responsive and native mobile websites.

HTML

HTML introduction:

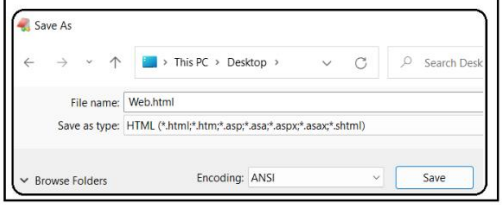
- HTML – Hyper Text Mark Up Language
- HTML language providing elements to create web pages
- HTML Elements are HTML, Head, Body, Img, Button, Input etc.

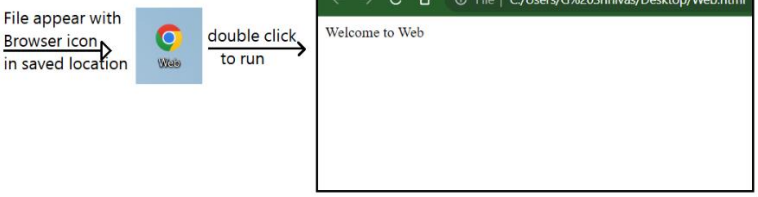
Note: Every HTML element is a program

Steps to write First HTML Code:

1. Open Notepad
2. Write HTML code
3. Save the file with .html extension
4. File will be shown with browser icon in the saved location.
5. Run the file by Double click on that ICON.

```
<html>
    <body>
      Welcome to Web
    </body>
</html>
```





What are HTML IDEs?

- IDE – Integrated Development Environment
- A software by which we develop and run HTML code easily
- Examples:
    - Visual Studio
    - Sublime text
    - Notepad++

Advantage of IDE: Code intelligence – Giving ideas to programmer to develop the code more easily. For example, if we write start tag – IDE will write end tag automatically.

Case-sensitivity: Html is purely case insensitive; we can use lowercase, uppercase or mixed-case while we are writing HTML tags.

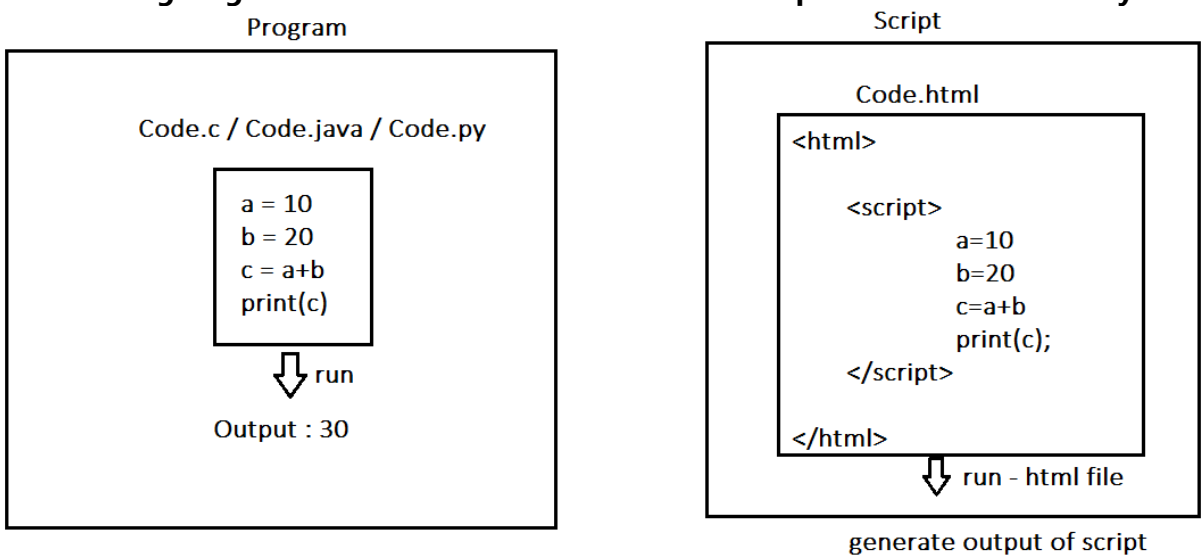
Errorless Language:

- Though HTML document contains errors, it runs and gives output.
- The only way to understand the errors with unexpected results.
- It is complex to identity the errors in HTML document.

Program v/s Script:

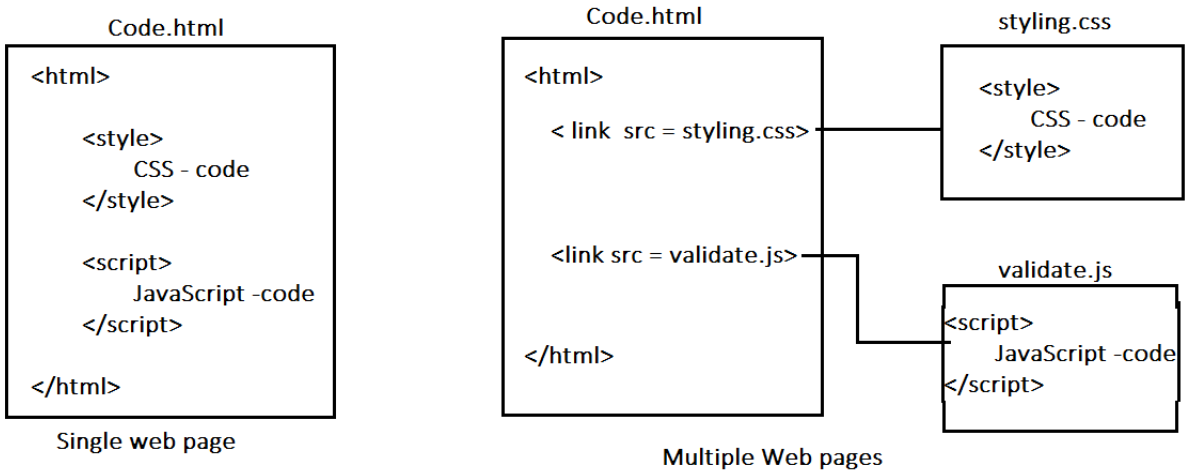
- **Program:** Run alone to generate output. For example, C, C++, Java etc.
- **Script:** Script always run with the help of another program to generate output hence Script is called Sub-Program.

The following diagram describes that we must run JavaScript from HTML code only:



We define HTML, CSS and JavaScript code in 2 ways:

1. Writing CSS and JavaScript code inside the HTML file only.
2. Writing CSS and JavaScript file separately and linking them from HTML page.



HTML Tags

HTML Tags:

- HTML language providing elements to prepare web pages
- We represent HTML elements with angular brackets(<>) called HTML Tags.
- HTML Tags are 2 types
    1) Non empty tags (container tags)
    2) Empty tags

Paired Tags (Non empty tags): Contains data. It is a combination of start tag & end tag

Syntax:

```
<start> data </end>
```

Example:

```
<title> data </title>
```

Unpaired Tags (Empty tags): It contains only start tag and we must end in the same tag

Syntax:

```
<start/>
```

Example:

```
<br/>
<img src=" "/>
<input type="button" />
```

Note: HTML is errorless language – hence it accepts the tags without syntax such as

Writing first HTML program:

- **<doctype>** tag represents which version of HTML is using in the document by which browser will understand and execute the code.
- HTML document starts with HTML tag and end with HTML tag

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

- Every HTML program has two locations
    1) Head location
    2) Body location

Head Location:

- The information which is not visible on web page.
- <meta> information: By which Search engine recognize our web site.
- CSS code
- JavaScript code
- Links to other documents(files)

Body location: The information visible on webpage

- Headings
- Paragraphs
- Form tags
- Images
- Hyperlinks….

HTML document format:

```
<!doctype html>
<html>
        <head>
                Head location information will not display on web page such as
                1. Page title (title tag)
                2. page information to search engine (meta tags)
                3. CSS code and JavaScript Code (link tags)
        </head>
        <body>
                Information display on web page
                1. Headings, Paragraphs, Lists, Tables
                2. Form tags
                3. Image, Audio files, Video files
                4. Hyperlinks
        </body>
</html>
```

Example HTML Program:

```
<!doctype html>
<html>
        <head>
                <title>My First Web page</title>
                <meta name="Keywords" content="HTML, CSS, JavaScript tutorial">
                <meta name="Description" content="Web Technologies tutorial site">
                <link type="text/css" rel="stylesheet" href="/styling.css">
        </head>
        <body>
                <p>I am planning to develop tutorials website</p>
        </body>
</html>
```

HTML Attributes

HTML attributes:

- Attributes also called Properties.
- Attributes giving extra functionality to HTML elements.
- Attributes must be placed inside <start> tag.
- Attribute name and value separated by equal to (=) sign
- Attribute values must be placed either in double quotes or single quotes

For example:

<start attribute=value> content </end>

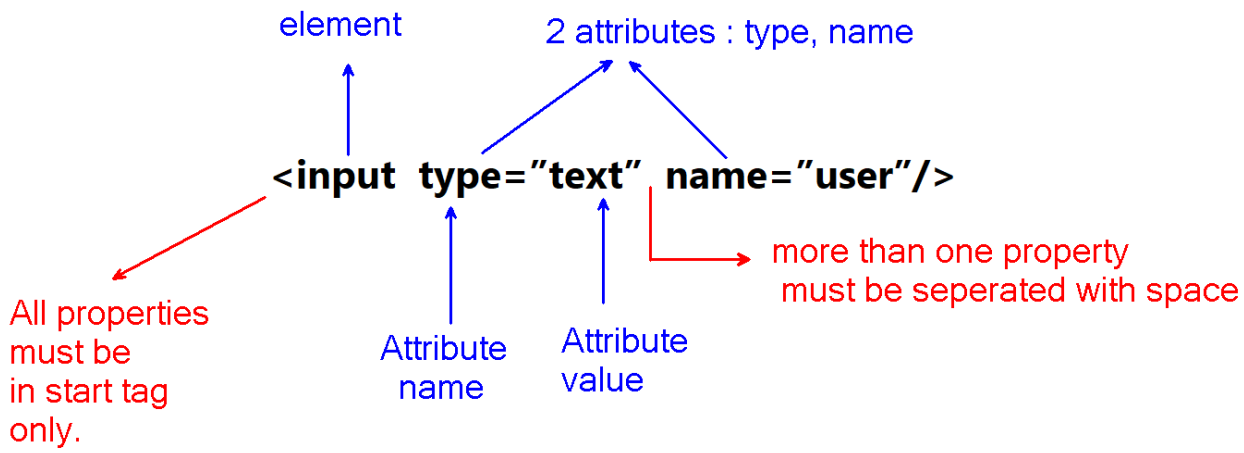
Note: Multiple attributes must be separated using space character.

Attributes in Non-empty tag:

<input type="text" name="user"/>

Attributes in Empty tag:

The following diagram explains attributes more clearly:



CSS Introduction

CSS Introduction:

- CSS stands for Cascading Style Sheets.
- CSS is used to apply styles to HTML elements
- We can apply styles to HTML elements in 3 ways
    1) Using style attribute
    2) Using <style> tag in head location
    3) Using external CSS file

1. **Using Style Attribute:** To apply different styles to different elements in single HTML document.

```
<!doctype html>
<html>
    <body>
        <p style="color:red"> First Paragraph </p>
        <p style="color:blue"> Second Paragraph </p>
        <p style="font-size:50px"> Third Paragraph </p>
    </body>
</html>
```

2. **Using <style> tag in head location:** When we want to apply same styles to multiple HTML elements in Single page.

```
<!doctype html>
<html>
    <head>
        <style>
            p{
color : red ;
}
            h1{
color : blue ;
}
        </style>
    </head>
    <body>
        <p> First Paragraph </p>
        <p> Second Paragraph </p>

        <h1> First Heading </h1>
        <h1> Second Heading </h1>
    </body>
</html>
```

3. **External CSS files:** When we want to apply same styles to all elements in different web pages**.** This process is called "Code centralization"

CSS file:

- Write CSS code into text file.
- Save with .css extension.
- Connect CSS file from HTML using <link> tag in head location

Mystyle.css:

```
p{
        color : blue ;
}

h1{
        color : red ;
}
```

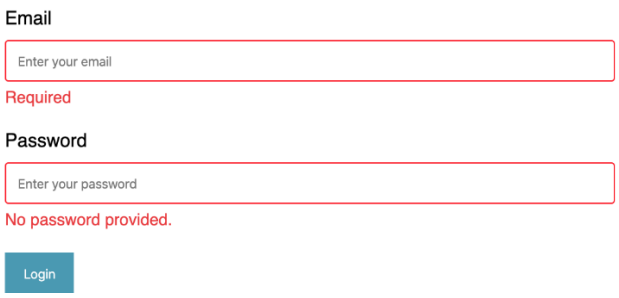
Web.html:

```
<!doctype html>
<html>
        <head>
                        <link rel="stylesheet" href="Mystyle.css"/>
        </head>
        <body>
                <p> First Paragraph </p>
                <p> Second Paragraph </p>
                <p> Third Paragraph </p>

                <h1> First Heading </h1>
                <h1> Second Heading </h1>
                <h1> Third Heading </h1>
        </body>
</html>
```

JavaScript Introduction

JavaScript Introduction:

- JavaScript performs Client-side validations
- JavaScript provides Functionality to web pages.



- JavaScript executes with HTML code.
- JavaScript is case-sensitive language

We can Write JavaScript code in places of HTML document

1) In Body Location
2) In Head Location

Body Location: Execute Script when web page is loading.

```
<html>
        <head>
        </head>
        <body>
                <script>
                        ....
                </script>
        </body>
</html>
```

Head Location: Execute Script on action (for example click on button).

```
<html>
        <head>
                <script>
                        ....
                </script>
        </head>
        <body>
        </body>
</html>
```

Heading tags:

- Heading tags are used to provide heading of page.
- Web page recognized by the search engine using the heading tags and sub heading tags.
- There is total 6 levels of heading tags h1, h2, h3…… h6

```
<!doctype html>
<html>
    <head>
        <title> Heading tags </title>
    </head>
    <body>
        <h1> Heading1 </h1>
        <h2> Heading2 </h2>
        <h3> Heading3 </h3>
        <h4> Heading4 </h4>
        <h5> Heading5 </h5>
        <h6> Heading6 </h6>
    </body>
</html>
```

HTML Paragraph tag:

- HTML paragraph is used to define a paragraph in a webpage.
- If we are using various <p> tags in one HTML file then browser automatically adds a single blank line between the two paragraphs.

```
<!DOCTYPE html>
<html>
    <body>
        <p>This is first paragraph.</p>
        <p>This is second paragraph.</p>
        <p>This is third paragraph.</p>
    </body>
</html>
```

Abbreviation tag: To abbreviate a text, write text between <abbr> and </abbr> tag.

```
<!DOCTYPE html>
<html>
    <body>
        <p>An <abbr title = "Hypertext Markup language">HTML </abbr>language is used to create web pages.
        </p>
    </body>
</html>
```

Marquee tag: The <marquee> tag is a container tag of HTML is implemented for creating scrollable text or images within a web page

```
<html>
        <body>
           <marquee width="40%" direction="up" height="30%">
                This is sample scrolling text.
           </marquee>
        </body>
</html>
```

Anchor tag:

- It is used to define a hyperlink that links to another page or files or location.
- The "href" attribute is used to provide the address of link.

```
<!DOCTYPE>
<html>
        <body>
                <a href="https://www.gmail.com">Gmail</a>
                <a href="https://www.google.com">Google</a>
                <a href="https://www.youtube.com">Youtube</a>
        </body>
</html>
```

<img>:

- HTML img tag is used to display image on the web page.
- It is an empty tag that contains attributes only.
- "src" attribute is necessary that describes the path of the image.

```
<!DOCTYPE>
<html>
        <body>
                <h2>HTML Image Example</h2>
                <img src="smile.jpg"/>
        </body>
</html>
```

Attributes <img> tag:

1) **alt:** This attribute defines an alternate text for the image, if it can't be displayed.
2) **width:** It is an optional attribute to specify the width of image.
3) **height:** It is used to specify height of image.

```
<!DOCTYPE html>
<html>
        <body>
                <img src="animal.jpg" height="180" width="300" alt="deer">
        </body>
</html>
```

HTML Comments: The text placed in between <!-- ... --> tags. Comment text will not execute by the browser. Comments are used to explain a statement or block of code.

```
<!DOCTYPE html>
<html>
        <body>
                <p>Welcome to Web</p>  <!-- <p> indicates paragraph -->
        </body>
</html>
```

HTML Lists: Representing List of things. There are 3 types of lists

1. Ordered List or Numbered List (ol)
2. Unordered List or Bulleted List (ul)
3. Description List or Definition List (dl)

Note: Defining a List inside another List is called Nested List

1. **Ordered List:** Representing the List with numbers.

```
<!DOCTYPE>
<html>
    <body>
            <h3>WebTechnologies</h3>
            <ol>
                    <li>HTML</li>
                    <li>CSS</li>
                    <li>JavaScript</li>
            </ol>
    </body>
</html>
```

2. **Unordered Lists:** Representing the List with Bullets or other symbols

```
<!DOCTYPE>
<html>
    <body>
        <h3>WebTechnologies</h3>
        <ul>
            <li>HTML</li>
            <li>CSS</li>
            <li>JavaScript</li>
        </ul>
    </body>
</html>
```

3. **Description List:** Representing each item in list with its description

```
<!DOCTYPE>
<html>
    <body>
        <h3>WebTechnologies</h3>
            <dl>
            <dt>HTML</dt>
            <dd>is used to created Web pages</dd>
            <dt>CSS</dt>
            <dd>is used to apply styles to web pages</dd>
            <dt>JavaScript</dt>
            <dd>is used to perform client side validations</dd>
            </dl>
    </body>
</html>
```

Type attribute in Lists: Type attribute is used to specify the type of Number or Symbol to show while printing the list. Type attribute can apply to both ordered lists and unordered lists.

Unordered List types:

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

Ordered List types:

```
<ol type = "1"> - Default-Case Numerals.
<ol type = "I"> - Upper-Case Numerals.
<ol type = "i"> - Lower-Case Numerals.
<ol type = "A"> - Upper-Case Letters.
<ol type = "a"> - Lower-Case Letters.
```

Start attribute: Start attribute represents the starting number of ordered list

```
<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.
```

Nested Lists: Defining a List inside another List

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaFullStack</title>
	</head>
	<body>
		<p>FullStack Concepts</p>
		<ol>
			<li>Web Technologies
				<ul>
					<li>HTML</li>
					<li>CSS</li>
					<li>JavaScript</li>
				</ul>
			</li>
			<li>ServerSide Technologies
				<ul>
					<li>Servlets</li>
					<li>JSP</li>
					<li>JDBC</li>
				</ul>
			</li>
			<li>Database
				<ul>
					<li>Oracle</li>
				</ul>
```

```
                    </li>
                    <li>UI Frameworks
                        <ul>
                            <li>ReactJS</li>
                        </ul>
                    </li>
                    <li>Server Side Frameworks</li>
                        <ul>
                            <li>Hibernate</li>
                            <li>Spring</li>
                            <li>SpringBoot</li>
                        </ul>
                    </li>
            </ol>
        </body>
</html>
```

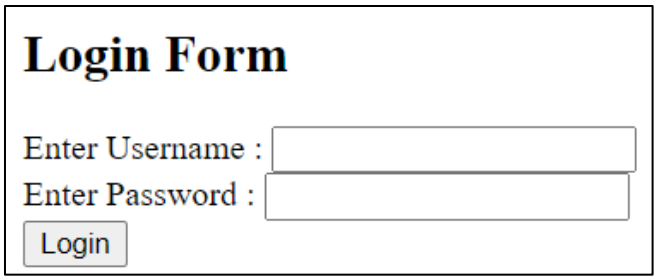
HTML Forms

HTML Forms:

- An HTML form is used to collect user input.
- HTML form data
    - Either sent to a server for processing.
    - Or send to JavaScript code for validation before submit to server.

Send form information directly to Server:

- Below example shows how to submit login form information to Java Servlet program.
- The data will submit to LoginServlet program.
- "name" attribute is used to collect the form values in Servlet program

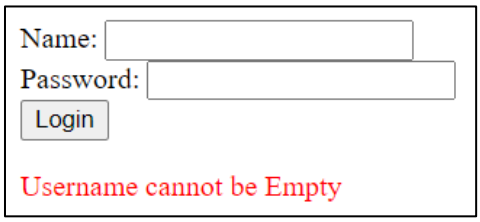


Code:

```
<!DOCTYPE html>
<html>
	<body>
		<h2>Login Form</h2>
		<form action="LoginServlet" method="post">
			<label>Enter Username :</label>
			<input type="text" name="user"/>
			<br/>
			<label>Enter Password :</label>
			<input type="password" name="pass">
			<br/>
			<input type="submit" value="Login"/>
		</form>
	</body>
</html>
```

Send form information to JavaScript for Validations before submission:

- It is recommended to perform minimum validations at client before submit to server.
- **Validation:** If the user clicks on submit button without input any data, we must display an error message from JavaScript code.




```
<html>
	<head>
		<script>
			function validateform()
			{
				var name=document.login.name.value;
				var pass=document.login.pass.value;
				if (name==null || name=="")
				{
		 document.getElementById("err").innerHTML = "Username cannot be Empty";
					return false;
				}
				else if(pass==null || pass=="")
				{
		 document.getElementById("err").innerHTML = "Password cannot be empty";
					return false;
				}
			}
		</script>
	</head>

	<body>
		<form name="login" method="post" action="LoginServlet" onsubmit="return
validateform()" >
			Name: <input type="text" name="name"/><br/>
			Password: <input type="password" name="pass"/><br/>
			<input type="submit" value="Login"/> <br/>
			<p id="err" style="color:red"></p>
		</form>
	</body>
</html>
```

Form elements: HTML providing form elements to take any type of input from the user. The elements are:

1. Labels
2. Text boxes
3. Password fields
4. Check boxes
5. Radio button
6. Drop down boxes
7. Submit buttons…..

Form syntax: All form controls must be placed inside the <form> …. </form> tags

```
<form action = "login.php">
        // form controls
</form>
```

HTML <input>: <input> tag is used to create different form controls such as Text fields, Password fields, Radio buttons, Check boxes etc.

Create Text Filed using <input>: we use type="text" to create text box

```
<!doctype html>
<html>
        <body>
                Enter UserName : <input type="text" name="uname"/>
        </body>
</html>
```

Note: "Name" attribute is used to collect the input value of text box from script or from server.

Create Password field using <input>: we use type="password" to create password field

```
<!doctype html>
<html>
        <body>
                Enter UserName : <input type="text" name="uname"/> <br/>
                Enter Password : <input type="password" name="pwd"/>
        </body>
</html>
```

Radio Button control:

- Radio buttons are used to select one option from multiple options.
- For example, Gender or Quiz etc….

Note: If we use one name for all the radio buttons, only one radio button can be selected.

```
<!doctype html>
```

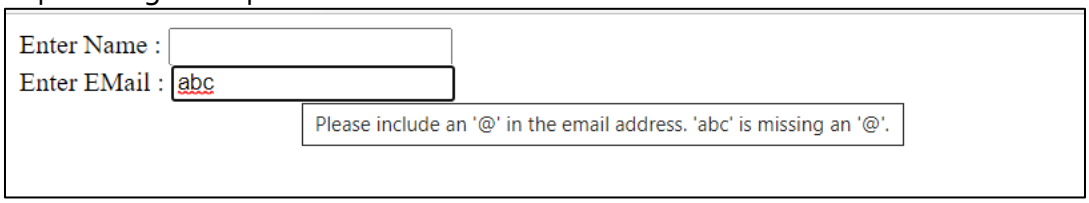
```
<html>
	<body>
		<input type="radio" name="gender"/> Male <br/>
		<input type="radio" name="gender"/> Female <br/>
		<input type="radio" name="gender"/> Other
	</body>
</html>
```

Email Field control:

- Email field is introduced int HTML 5
- It validates the text for correct email address
- You must use @ and . in this field

```
<!doctype html>
<html>
	<body>
		Enter Name : <input type="text" name="name"/> <br/>
		Enter EMail : <input type="email" name="mail"/> <br/>
	</body>
</html>
```

Output: providing tool tip to enter the valid email address



Checkbox control:

- We can select more than one option at a time from the given options
- Examples: Languages known, Hobbies, Skills etc.

```
<!doctype html>
<html>
	<body>
		Languages Known : <br/>
		<input type="checkbox" name="Telugu"/> Telugu<br/>
		<input type="checkbox" name="English"/> English<br/>
		<input type="checkbox" name="Hindi"/> Hindi<br/>
	</body>
</html>
```

Submit button: we use type="submit" to create submit button. We can submit form data to JavaScript code as well as to server-side program.

```
<!doctype html>
<html>
	<body>
		<form action="login.php">
			Enter Username : <input type="text" name="user"/> <br/>
			Enter Password : <input type="password" name="pwd"/> <br/>
			<input type="submit" value="LOGIN" />
		</form>
	</body>
</html>
```

Drop Down Control:

- <select> element is used to create Drop Down list in HTML
- <option> tag is used to add values to drop down list.

```
<!doctype html>
<html>
    <body>
        <form action="login.php">
            <label> Select Cars </label>
            <select>
                <option> BMW </option>
                <option> Volvo </option>
                <option> Audi </option>
                <option> Benz </option>
            </select>
        </form>
    </body>
</html>
```

HTML Tables

HTML Table: Tables are used to represent the record type information.

- We create the table using <table> tag.
- <tr> represents table-row
- <td> represents table-column or data
- <th> is used to define headings.

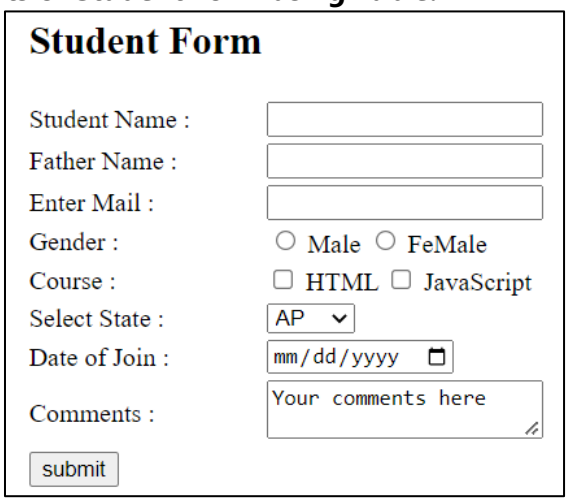
Note: It is recommended to display the table with borders

Displaying Full Stack Coursed Information:

Course	Fees	Duration
UI	25000	3 Months
Java	30000	4 Months
Python	25000	3 Months

```
<html>
        <head> <title>FullStack Courses</title> </head>
        <body>
                <table border=1>
                        <tr>
                                <th>Course</th>
                                <th>Fees</th>
                                <th>Duration</th>
                        </tr>
                        <tr>
                                <td>UI</td>
                                <td>25000</td>
                                <td>3 Months</td>
                        </tr>
                        <tr>
                                <td>Java</td>
                                <td>30000</td>
                                <td>4 Months</td>
                        </tr>
                        <tr>
                                <td>Python</td>
                                <td>25000</td>
                                <td>3 Months</td>
                        </tr>
                </table>
        </body
</html>
```

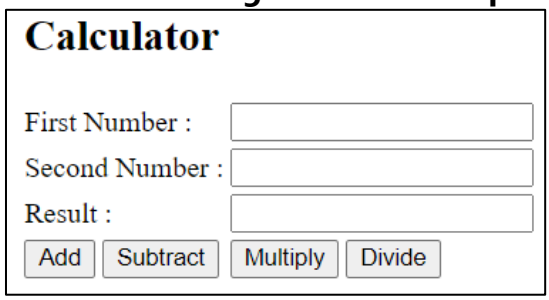
Displaying All components of Student Form using Table:




```
<!doctype html>
<html>
	<body>
		<table align="center">
			<tr>
				<td><h2> Student Form </h2></td>
			</tr>

			<tr>
				<td> Student Name : </td>
				<td> <input type="text" name="sname" /></td>
			</tr>

			<tr>
				<td> Father Name : </td>
				<td> <input type="text" name="fname" /></td>
			</tr>

			<tr>
				<td> Enter Mail : </td>
				<td> <input type="email" name="mail" /></td>
			</tr>

			<tr>
				<td> Gender : </td>
				<td> <input type="radio" name="gender"/> Male
					<input type="radio" name="gender"/> FeMale</td>
			</tr>
```

```
                    <tr>
                        <td> Course : </td>
                        <td> <input type="checkbox" name="HTML"/> HTML
                            <input type="checkbox" name="JavaScript"/>
JavaScript</td>
                    </tr>

                    <tr>
                        <td> Select State : </td>
                        <td>
                            <select name="state">
                                <option> AP </option>
                                <option> TS </option>
                                <option> GOA </option>
                            </select>
                        </td>
                    </tr>

                    <tr>
                        <td> Date of Join :</td>
                        <td> <input type="date" /> </td>
                    </tr>

                    <tr>
                        <td> Comments : </td>
                        <td>
                            <textarea row="2" columns="20">Your comments
here</textarea>
                        </td>
                    </tr>

                    <tr>
                        <td> <input type="submit" value="submit"/> </td>
                    </tr>
                </table>
        </body>
</html>
```

Calculator Program in JavaScript




```
<!doctype html>
<html>
	<head>
		<script>
			function add()
			{
				var a = parseInt(document.getElementById("first").value);
				var b = parseInt(document.getElementById("second").value);
				var c = a+b;
				document.getElementById("result").value = c;
			}
			function subtract()
			{
				var a = parseInt(document.getElementById("first").value);
				var b = parseInt(document.getElementById("second").value);
				var c = a-b;
				document.getElementById("result").value = c;
			}
			function multiply()
			{
				var a = parseInt(document.getElementById("first").value);
				var b = parseInt(document.getElementById("second").value);
				var c = a*b;
				document.getElementById("result").value = c;
			}
			function divide()
			{
				var a = parseInt(document.getElementById("first").value);
				var b = parseInt(document.getElementById("second").value);
				var c = a/b;
				document.getElementById("result").value = c;
			}
		</script>
	</head>
```

```
	<body>
		<table align="center">
			<tr>
				<td><h2> Calculator </h2></td>
			</tr>

			<tr>
				<td> First Number : </td>
				<td> <input type="text" id="first" /></td>
			</tr>

			<tr>
				<td> Second Number : </td>
				<td> <input type="text" id="second" /></td>
			</tr>

			<tr>
				<td> Result : </td>
				<td> <input type="text" id="result" /></td>
			</tr>

			<tr>
				<td>
					<button onclick="add()">Add</button>
					<button onclick="subtract()">Subtract</button>
				</td>
				<td>
					<button onclick="multiply()">Multiply</button>
					<button onclick="divide()">Divide</button>
				</td>
			</tr>
		</table>
	</body>
</html>
```

CSS – Cascading Style Sheets

Introduction:

- Using HTML, we can create only plain web pages.
- We use CSS to apply styles to HTML pages such as text-colors, background-colors, borders, alignments and many more.

We can apply styles to HTML elements in 3 ways

1. Using style attribute
2. Using <style> tag in head location
3. Using external CSS file

1. **Using Style Attribute:** To apply different styles to different elements in single HTML document.

```
<html>
    <head>
        <title> Headings </title>
    </head>
    <body>
        <h1 style="color:red"> Web Technologies </h1>
        <h2 style="color:blue"> HTML </h2>
        <p style="color:green"> HTML is used to create Web pages </p>

        <h2 style="color:blue"> CSS </h2>
        <p style="color:green"> CSS is used to apply styles </p>

        <h2 style="color:blue"> JavaScript </h2>
        <p style="color:green"> JavaScript is used to validate Web pages </p>
    </body>
</html>
```

2. **Using <style> tag in head location:** When we want to apply same styles to multiple HTML elements in Single page.

```
<html>
    <head>
        <title> Headings </title>
        <style>
            h1{
                color:red;
            }
            h2{
                color:blue;
            }
```

```
                p{
                        color:green;
                }
            </style>
        </head>
        <body>
            <h1 style="color:red"> Web Technologies </h1>
            <h2 style="color:blue"> HTML </h2>
            <p style="color:green"> HTML is used to create Web pages </p>

            <h2 style="color:blue"> CSS </h2>
            <p style="color:green"> CSS is used to apply styles </p>

            <h2 style="color:blue"> JavaScript </h2>
            <p style="color:green"> JavaScript is used to validate Web pages </p>
        </body>
    </html>
```

3. **External CSS files:** When we want to apply same styles to all elements in different web pages**.** This process is called "Code centralization"

CSS file:

- Write CSS code into text file.
- Save with .css extension.
- Connect CSS file from HTML using <link> tag in head location

Mystyle.css:

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

Web.html:

```
<!doctype html>
<html>
        <head>
                <link rel="stylesheet" href="Mystyle.css"/>
        </head>
```

```
        <body>
                <h1 style="color:red"> Web Technologies </h1>
                <h2 style="color:blue"> HTML </h2>
                <p style="color:green"> HTML is used to create Web pages </p>

                <h2 style="color:blue"> CSS </h2>
                <p style="color:green"> CSS is used to apply styles </p>

                <h2 style="color:blue"> JavaScript </h2>
                <p style="color:green"> JavaScript is used to validate Web pages </p>
        </body>
</html>
```

Change the background colors by click on the buttons:

```
<html>
        <head>
                <script>
                        function blue()
                        {
                                document.body.style.backgroundColor="blue";
                        }
                        function green()
                        {
                                document.body.style.backgroundColor="green";
                        }
                        function red()
                        {
                                document.body.style.backgroundColor="red";
                        }
                        function yellow()
                        {
                                document.body.style.backgroundColor="yellow";
                        }
                </script>
        </head>

        <body>
                <button onclick="blue()"> Blue </button>
                <button onclick="green()"> Green </button>
                <button onclick="red()"> Red </button>
                <button onclick="yellow()"> Yellow </button>
        </body>
</html>
```

Class Selector in CSS

CSS .class Selector: .class selector select elements with specific class attribute.

CSS Borders: We can set borders to HTML elements using border-style attribute in CSS

```
<html>
	<head>
		<style>
			p{
				border-style : dotted ;
			}
		</style>
	</head>
	<body>
		<p> No border </p>
		<p> A dotted border </p>
		<p> A dashed border </p>
		<p> A solid border </p>
		<p> A double border </p>
	</body>
</html>
```

Note: In above example, same border is applied to all paragraphs.

To select particular element in HTML from Style – we use Class Selector

```
<html>
	<head>
		<style>
			p.none { border-style : none ; }
			p.dotted { border-style : dotted ; }
			p.dashed { border-style : dashed ; }
			p.solid { border-style : solid ; }
			p.double { border-style : double ; }
		</style>
	</head>
	<body>
		<p class="none"> No border </p>
		<p class="dotted"> A dotted border </p>
		<p class="dashed"> A dashed border </p>
		<p class="solid"> A solid border </p>
		<p class="double"> A double border </p>
	</body>
</html>
```

CSS ID Selection

CSS id Selector:

- HTML page consists number of HTML elements.
- The id is used to access an element uniquely with in the page.
- We use hash(#) character to access.

```
<!doctype html>
<html>
    <head>
        <style>
            p{
                color : blue ;
                text-align : center ;
            }
        </style>
    </head>
    <body>
        <p> Paragraph1 </p>
        <p> Paragraph2 </p>
        <p id="p1"> Paragraph with ID </p>
    </body>
</html>
```

Note: In the above example, Styles applied to all paragraph elements as we didn't select particular paragraph element with ID

```
<!doctype html>
<html>
    <head>
        <style>
            #p1{
                color : blue ;
                text-align : center ;
            }
        </style>
    </head>
    <body>
        <p> Paragraph1 </p>
        <p> Paragraph2 </p>
        <p id="p1"> Paragraph with ID </p>
    </body>
</html>
```

We can apply id selector for particular element and remaining element with different style.

```
<!doctype html>
<html>
	<head>
		<style>
			#p1{
				color : blue ;
				text-align : center ;
			}

			p{
				color : red ;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<p> Paragraph1 </p>
		<p> Paragraph2 </p>
		<p id="p1"> Paragraph with ID </p>
	</body>
</html>
```

We can apply same id to multiple HTML elements: When we select ID to apply styles all related elements affects

```
<!doctype html>
<html>
	<head>
		<style>
			#p1{
				color : blue ;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<p> Paragraph1 </p>
		<p id="p1"> Paragraph with ID </p>
		<p> Paragraph2 </p>
		<p id="p1"> Paragraph with ID </p>
	</body>
</html>
```

CSS Class Selector v/s ID Selector:

Class	Id
We can apply a class to various elements.	We can only apply it to one specific element.
Starts with "." followed by the name of the class.	Starts with "#" symbol followed by a unique id name.
We can attach multiple class selectors to an element.	We can attach only one ID selector to an element.

Syntax:
.class{
// declarations of CSS
}
	Syntax:
	#id{
	// declarations of CSS
	}

Application Class Selector , ID Selector and No Selector:

```
<html>
	<head>
		<style>
			.center{
				color : blue ;
				text-align : center ;
			}
			#para1{
				color : red ;
				text-align : left;
			}
			#para2{
				color : orange;
				text-align : center;
			}
			p{
				color : green ;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<p class="center"> Paragraph1</p>
		<p class="center"> Paragraph2</p>
		<p id="para1"> Paragraph3 </p>
		<p id="para2"> Paragraph4 </p>
		<p> Paragraph5 </p>
		<p> Paragraph6 </p>
	</body>
</html>
```

CSS Universal Selector: Selects All HTML elements on the page

```
<!doctype html>
<html>
	<head>
		<style>
			*{
				color : blue ;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<p> Paragraph1 </p>
		<p id="p1"> Paragraph with ID p1 </p>
		<p> Paragraph2 </p>
		<p id="p2"> Paragraph with ID p2 </p>
	</body>
</html>
```

CSS Grouping Selector: Selects all HTML elements with same style definitions.

```
<!doctype html>
<html>
	<head>
		<style>
			h1, h2, p{
				color : blue ;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<h1> Heading1 </h1>
		<h1> Heading2 </h1>
		<p> Paragraph1 </p>
		<p> Paragraph2 </p>
		<h2> Heading3 </h2>
		<h2> Heading4 </h2>
	</body>
</html>
```

CSS Backgrounds: used to apply Background colors and images

- background-color
- background-image
- background-repeat
- background-position

Background-color: We apply background color to entire body hence we use

```
<!doctype html>
<html>
	<head>
		<style>
			h1, h2, p{
				color : red ;
				text-align : center ;
			}
			body{
				background-color : lightblue ;
			}
		</style>
	</head>
	<body>
		<h1> Heading1 </h1>
		<p> Paragraph1 </p>
		<h2> Heading4 </h2>
	</body>
</html>
```

Background- image: By default, the image is applicable to entire page.

url():

- functions is used to specify the location of image
- If the image and our HTML file are in the same location – no need to specify the entire path of image.
  - **url("abc.jpg")**
- If the image and HTML file are in different locations – we need to specify the complete path of image
  - **url("d:/src/myimages/abc/jpg");**

```
<!doctype html>
<html>
	<head>
		<style>
			h1, h2, p{
```

```
                        color : red ;
                        text-align : center ;
                }
                body{
                        background-image : url("download.jpg") ;
                }
        </style>
    </head>
    <body>
        <h1> Heading1 </h1>
        <p> Paragraph1 </p>
        <h2> Heading4 </h2>
    </body>
</html>
```

Background-repeat:

- We can set the direction of image only to
  - X – Axis (background-repeat : repeat-x)
  - Y – Axis (background-repeat : repeat-y)
  - No repeat (background-repeat : no-repeat)

```
<!doctype html>
<html>
    <head>
        <style>
            h1, h2, p{
                color : red ;
                text-align : center ;
            }
            body{
                background-image : url("download.jpg") ;
                background-repeat : repeat-x;
            }
        </style>
    </head>
    <body>
        <h1> Heading1 </h1>
        <p> Paragraph1 </p>
        <h2> Heading4 </h2>
    </body>
</html>
```

CSS - Borders

CSS Borders:

- We can set CSS borders to all elements or individual elements.
- We can apply borders to elements ID selector, Class selector…..

```
<!doctype html>
<html>
	<head>
		<style>
			p.one{
				color : red ;
				text-align : center ;
				border-color : blue ;
				border-style : solid ;
			}
			p.two{
				color : blue ;
				text-align : center ;
				border-color : green ;
				border-style : dotted ;
			}
			p.three{
				color : green ;
				text-align : center ;
				border-color : red ;
				border-style : dashed ;
			}
		</style>
	</head>
	<body>
		<p class="one"> Paragraph1 </p>
		<p class="two"> Paragraph2 </p>
		<p class="three"> Paragraph3 </p>
		<p class="four"> Paragraph4 </p>
	</body>
</html>
```

DIV tag in HTML:

- tag means division or section in HTML page
- section works like container (separate set of HTML tags)
- is used to apply styles to specific set of HTML tags using class selector

CSS Padding: It is used to create space around the HTML element content.

```
<!doctype html>
<html>
	<head>
		<style>
			div {
				padding : 70px ;
				border : 1px solid blue ;
			}
		</style>
	</head>
	<body>
		<h1> HTML Div - CSS Padding </h1>
		<div> This div element with padding </div>
	</body>
</html>
```

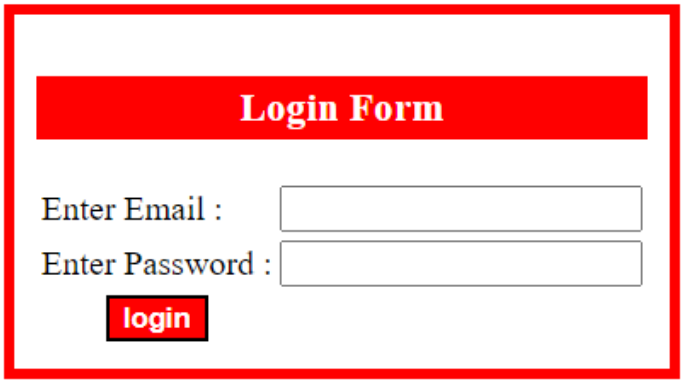
Note: border specification is important for every tag

Apply CSS styles to each tag in tag:

```
<!doctype html>
<html>
	<head>
		<style>
			div {
				padding : 10px ;
				border : 2px solid blue ;
			}

			h1 {
				text-align : center;
				text-transform : uppercase;
				color : red ;
			}

			p {
				text-align : justify ;
				letter-spacing : 3px ;
```

```
                }

                a {
                        text-decoration : none ;
                        color : green ;
                }
        </style>
    </head>
    <body>
        <div>
            <h1> Heading text </h1>
            <p> The div tag is generally used by web developers to group HTML elements together and apply CSS styles to many elements at once. For example: If you wrap a set of paragraph elements into a div element so you can take the advantage of CSS styles and apply font style to all paragraphs at once instead of coding the same style for each paragraph element.</p>
            <a href="www.google.com"> Google </a>
        </div
    </body>
</html>
```

** tags mostly used to represent the form in a style:**




```
<!doctype html>
<html>
    <head>
        <style>
            .login{
                padding : 10px;
                border : 5px solid red ;
                float : left;
            }
            .head{
                background-color : red;
```

```
                        color : white ;
                        padding : 4px;
                        text-align : center;
                }

                .sub{
                        background-color : red;
                        color : white ;
                        font-weight : bold ;
                }
        </style>
    </head>
    <body>
        <div class="login">
            <h3 class="head"> Login Form </h3>
            <form action="Login.jsp" method="post">
                <table>
                    <tr>
                        <td> Enter Email : </td>
                        <td> <input type="email"
name="email"/></td>
                    </tr>

                    <tr>
                        <td> Enter Password : </td>
                        <td> <input type="password"
name="pwd"/></td>
                    </tr>

                    <tr>
                        <td style="text-align:center"><input class="sub"
type="submit" value="login"/></td>
                    </tr>
                </table>
            </form>
        </div
    </body>
</html>
```

Simple with background and fore-ground color text:

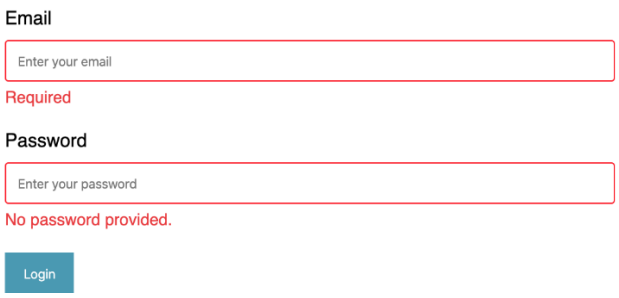
Code:

```
<!doctype html>
<html>
	<head>
		<style>
			.myDiv{
				border : 5px solid red ;
				background-color : lightblue;
				text-align : center ;
			}
		</style>
	</head>
	<body>
		<h1> Div Tag Example </h1>
		<div class="myDiv">
			<h1> This is Heading tag </h1>
			<p> This is paragraph tag </p>
		</div>
		<p> Note : This text is outside to div element </p>
	</body>
</html>
```

JavaScript

JavaScript Introduction:

- JavaScript performs Client-side validations
- JavaScript provides Functionality to web pages.



- JavaScript executes with HTML code.
- JavaScript is case-sensitive language

We can Write JavaScript code in places of HTML document

3) In Body Location
4) In Head Location

Body Location: Execute Script when web page is loading.

```
<html>
        <head>
        </head>
        <body>
                <script>
                        ....
                </script>
        </body>
</html>
```

Head Location: Execute Script on action(for example click on button).

```
<html>
        <head>
                <script>
                        ....
                </script>
        </head>
        <body>
        </body>
</html>
```

JavaScript – Comments

Comments: Comments are used to describe the code or document. These are 2 types

1. Single-line Comment
2. Multi-line Comment

Single line Comment: It is represented by double forward slashes (//).

```
<!doctype html>
<html>
	<body>
		<script>
			var a=10;
			var b=20;
			var c=a+b; // + operator add values of a & b and store result into c
			document.write(c); // write() - display information of web page
		</script>
	</body>
</html>
```

Multi line Comment: It can be used describe the entire document.

```
/* This program is used to perform
Addition operation of 2 numbers
And display result on Web page
Using document.write() method */
<!doctype html>
<html>
	<body>
		<script>
			var a=10;
			var b=20;
			var c=a+b;
			document.write(c);
		</script>
	</body>
</html>
```

Variables and Functions

Variable:

- Variable is an identity of memory location.
- Variable is used to store the information.

Create variable:

- In JavaScript, we represent the variables either by using var keyword or let keyword
- No need to specify the data type in variable declaration.

Syntax:	**Example**

var identity;
or
let identity;
	var name = "amar";
	var age = 23;
	var salary = 35000.00;

Rules to create variable:

1. Variable name contains Alphabets (a-z or A-Z) or Digits(0-9) or Symbols ($ or _)
2. Variable name should not start with digit
    a. var 9a = 10 ; (error)
    b. var a9 = 10 ; (no error)
3. Only two symbols allowed in variable name ($, _)
    a. var acc-num = 1234; (error)
    b. var acc_num = 1234; (no error)
4. Variables are case sensitive. Hence we can define multiple variables with same name with different case.
    a. var a=10, A=20; (no error)
5. Variable name should not match with keywords
    a. var for=10 ;

Variable Declaration, Assignment, Modify and Initialization:

Declaration: A variable without value is called declaration

```
var a;
let b;
```

Assignment: Assigning value to variable which is already declared.

```
var a;  // declaration
a = 10;  // assignment
a = 20;  // assignment
```

Modify: Updating the existing value of variable.

```
var balance; // declaration
balance = 3000 ; // assignment
balance = balance – 1400 ;  // modify or update
```

Initialization: Assigning a value at the time of declaration

```
var x = 10 ; // initialization
```

Function:

- Variable stores the information
- Function performs operation on information.
- Function is a block of instructions – taking input, perform operations on input and returns result.
- We define functions in JavaScript using **"function"** keyword

Syntax:	**Example:**

function identity()
{
.....
Logic
.....
}
	function add()
	{
	var a=10, b=20;
	var c = a+b;
	print(c);
	}

Note: We always perform operations when action performed. For example, display alert message when user clicks on button

```
<!doctype html>
<html>
	<head>
		<script>
			function clickMe()
			{
				alert("Hello");
			}
		</script>
	</head>

	<body>
		<h2> JavaScript Alert Box </h2>
		<button onclick="clickMe()">Click Me</button>
	</body>
</html>
```

document.write():

- "document" is a pre-defined object in JavaScript.
- "document" represents "webpage".
- write() is a pre-defined method to display information.
- If we invoke write() method on document object then it writes the information on webpage

Local and Global Variables

Local variables: Defining a variable inside the function or block is called Local variable. We can access them only from the same block or function.

```
<html>
	<head>
		<script>
			function add(){
				var a=10;
				var b=20;
				var c=a+b;
				document.write("Sum is : " + c);
			}
		</script>
	</head>
	<body>
		<button onclick="add()">add</button>
	</body>
</html>
```

Global variables: Defining variables outside to all functions – we can them from all functions.

```
<html>
	<head>
		<script>
			var a=5;
			var b=3;
			function add(){
				document.write("Sum is : " + (a+b));
			}
			function subtract(){
				document.write("Difference is : " + (a-b));
			}
			function multiply(){
				document.write("Product is : " + (a*b));
			}
		</script>
	</head>
	<body>
		<button onclick="add()">add</button>
		<button onclick="subtract()">subtract</button>
		<button onclick="multiply()">multiply</button>
	</body>
</html>
```

Var, Let and Const

Var	Let	Const
Variable can be either Global Scope or Local Scope	Variable is Local scope	Variable is Local Scope
Re-Declaration or Updating of variable is possible	Only Updating possible	Nothing is possible.
Hoisting allowed – Variables initializes with "undefined"	Hoisting allowed – Variables not initializes with values.	Hoisting allowed – Variables not initializes with values.
Can be declared without being initialized	Can be declared without being initialized	Must be initialized with value at the time of declaration.

Clear explanation about Var, Let and Const:

Var: var declarations are globally scoped or function/locally scoped.

```
var x = 10;
function myFun() {
        var y = 20;
}
```

var variables can be re-declared:

```
var x = 10;
var x = 20;
```

Display the value of re-declared variable as follows:

```
var x = 10;
var times = 4;
if (times > 3) {
        var x = 20;
}
console.log(x) // display 20
```

Hoisting of var: variables and function declarations are moved to the top of their scope before code execution.

if we do this:

```
console.log (x);
var x = 10;
```

it is interpreted as this: (initialized with a value of undefined)

```
var x;
console.log(x); // x is undefined
x = 10
```

Let: let is block scope. So a variable declared in a block with let is only available within block.

```
let x = 10;
let times = 4;
if (times > 3) {
        let y = 20;
        console.log(y); // 20
}
console.log(y) // y is not defined
```

let can be updated:

```
let x = 10;
x = 20;
```

but not re-declared:

```
let x = 10;
let x = 20; // error: already declared
```

However, if the same variable is defined in different scopes, there will be no error:

```
let x = "say Hi";
if (true) {
        let x = "say Hello instead";
        console.log(x); // "say Hello instead"
}
console.log(x); // "say Hi"
```

Hoisting of let: Just like var, let declarations are hoisted to the top. Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error.

Const:

- Variables declared with the const maintain constant values.
- const declarations are block scoped
- const cannot be updated or re-declared

```
const x = "say Hi";
x = "say Hello instead";// error: Assignment to constant variable.
```

Note: Every const declaration, therefore, must be initialized at the time of declaration.

Hoisting of const: Just like let, const declarations are hoisted to the top but are not initialized.

Classification of Functions

Classifications: Depends on taking input parameters and return values, functions classified into four types.

No Parameters & No Return values	With Parameters & No Return values	With Parameters & With Return values	No Parameters & With Return values

function say()
{
write("Hello");
}
	function add(a, b)
	{
	var c = a+b;
	write(c);
	}
		function add(a, b)
		{
		var c = a+b;
		return c;
		}
			function getPI()
			{
			const PI = 3.142;
			return PI;
			}

No Parameters and No Return values code: this type of function is used to perform operations like clear(), removeAll(), displayAll() etc.

```
<!doctype html>
<html>
	<head>
		<script>
			function sayHello(){
				document.write("Hello All, Welcome");
			}
		</script>
	</head>
	<body>
		<button onclick="sayHello()">Say Hello to All</button>
	</body>
</html>
```

With Parameters and No return values: This type of function takes input, performs operation and display results from the function itself.

```
<!doctype html>
<html>
	<head>
		<script>
			function add(a, b){
				var c = a+b;
				document.write("Sum is : " + c);
			}
		</script>
	</head>

	<body>
```

```
            <button onclick="add(10, 20)">Add</button>
        </body>
</html>
```

Function with Parameters and with Return values:

```
<!doctype html>
<html>
        <head>
                <script>
                        function add(a, b){
                                var c = a+b;
                                return "Sum = "+c;
                        }
                        function calculate(){
                                document.getElementById("sum").innerHTML = add(10, 20);
                        }
                </script>
        </head>
        <body>
                <button onclick="calculate()">Add</button>
                <p id="sum"> </p>
        </body>
</html>
```

Function Without Parameters with Return Values:

```
<!doctype html>
<html>
        <head>
                <script>
                        function PI(){
                                return "PI value is = 3.142";
                        }
                        function getPI(){
                                document.getElementById("pi").innerHTML = PI();
                        }
                </script>
        </head>
        <body>
                <button onclick="getPI()">PI Value</button>
                <p id="pi"> </p>
        </body>
</html>
```

Formatting the Output

Format Output:

- It is important to format the results before display to the user.
- Write() method display information on web page.
- We display information by concatenating with proper messages to make the user more readable.

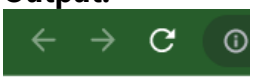
The table giving the examples to concatenate the messages with values:

Syntax	Example
int + int -> int	10 + 20 = 30
String + String -> String	"Hello" + "World" -> HelloWorld "10" + "20" -> 1020
String + int -> String	"Sum = " + 10 -> Sum = 10
String + int + int -> String	"Sum = " + 10 + 20 -> Sum = 1020
String + (int + int) -> String	"Sum = " + (10+20) -> Sum = 30
String + double -> String	"PI = " + 3.142 -> PI = 3.142
String – int -> Error	"Sum = " – 10 -> Error

Code:

```
<!doctype html>
<html>
	<body>
		<script>
			document.write(10 + 20 + "<br/>");
			document.write("Hello" + "World" + "<br/>");
			document.write("Sum = " + 10 + "<br/>");
			document.write("PI value = " + 3.142 + "<br/>");
			document.write("Sum = " + 10 + 20 + "<br/>");
			document.write("Sum = " + (10 + 20) + "<br/>");
		</script>
	</body>
</html>
```

Output:



30
HelloWorld
Sum = 10
PI value = 3.142
Sum = 1020
Sum = 30

Operators

Operator: Operator is a symbol that performs operation on data.

Arithmetic Operators:

- These operators are used to perform all arithmetic operations.
- Operators are +, - , *, / , %
- Division(/) operator returns the quotient.
- Mod(%) operator returns the remainder.

```
<!doctype html>
<html>
	<body>
		<script>
			var a=5, b=3;
			document.write(a + " + " + b + " = " + (a+b) + "<br/>");
			document.write(a + " - " + b + " = " + (a-b) + "<br/>");
			document.write(a + " * " + b + " = " + (a*b) + "<br/>");
			document.write(a + " / " + b + " = " + (a/b) + "<br/>");
			document.write(a + " % " + b + " = " + (a%b) + "<br/>");
		</script>
	</body>
</html>
```

Program to display the last digit of given number:

```
<!doctype html>
<html>
	<head>
		<script>
			function lastDigit()
			{
				var num = parseInt(document.getElementById("num").value);
				var x = num%10;
				document.getElementById("last").innerHTML="Last Digit is : "+x;
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="lastDigit()"> Last Digit </button>
		<p id="last"> </p>
	</body>
</html>
```

Program to remove the last digit of given number:

```
<!doctype html>
<html>
	<head>
		<script>
			function removeLastDigit()
			{
				var num = parseInt(document.getElementById("num").value);
				var n = Math.floor(num/10);
				document.getElementById("last").innerHTML = "Now n is : " + n
;
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="removeLastDigit()"> Remove Last Digit </button>
		<p id="last"> </p>
	</body>
</html>
```

Calculate the Total Salary based on Basic Salary of Employee:

```
<!doctype html>
<html>
	<head>
		<script>
			function totalSalary()
			{
				var basic = parseFloat(document.getElementById("sal").value);
				var hra = 0.15 * basic;
				var da = 0.2 * basic;
				var ta = 0.25 * basic ;
				var total = basic + hra + da + ta ;
				document.getElementById("total").innerHTML= "Total :" + total ;
			}
		</script>
	</head>
	<body>
		Enter Basic Salary : <input type="text" id="sal"/>
		<button onclick="totalSalary()"> Find Total Salary </button>
		<p id="total"> </p>
	</body>
</html>
```

Relational operators: These operators return a boolean values by validating the relation between data. Operators are >, < , >=, <=, ==, !=

```
<html>
	<body>
		<script>
			var a=5, b=3;
			document.write(a + " > " + b + " = " + (a>b) + "<br/>");
			document.write(a + " < " + b + " = " + (a<b) + "<br/>");
			document.write(a + " == " + b + " = " + (a==b) + "<br/>");
			document.write(a + " != " + b + " = " + (a!=b) + "<br/>");
		</script>
	</body>
</html>
```

Logical operators: These operators return a boolean value by validating more than one expression. Operators are &&, ||, !

Expression1	Expression2	&&	||	!Expression1	!Expression2
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

```
<html>
	<body>
		<script>
			document.write("true && true = " + (true && true) + "<br/>");
			document.write("true && false = " + (true && false) + "<br/>");
			document.write("true || true = " + (true || true) + "<br/>");
			document.write("true || false = " + (true || false) + "<br/>");
		</script>
	</body>
</html>
```

```
<html>
	<body>
		<script>
			document.write("5>3 && 5!=3 = " + (5>3 && 5!=3) + "<br/>");
			document.write("5>3 || 5==3 = " + (5>3 || 5==3) + "<br/>");
			document.write("!(5>3) = " + (!(5>3)) + "<br/>");
		</script>
	</body>
</html>
```

Conditions using Arithmetic, Relational and Logical operators

Condition to check the number is Positive or Negative:

Variables:

var a=?;

Condition:

a >= 0

Condition to check the number is equal to zero or not:

Variables:

var a=?;

Condition:

a == 0

Condition to check the 2 numbers equal or not:

Variables:

var a=?, b=?;

Condition:

a ==b

Condition to check First Num greater than Second Num or Not:

Variables:

var a=?, b=?;

Condition:

a > b

Square of First Number equals to Second Number or Not :

Variables:

var a=?, b=?;

Condition:

a*a == b

Condition to check sum of 2 numbers equal to 10 or not:

Variables:

var a=?, b=? ;

Condition:

a+b == 10 ;

Condition to check the number divisible by 3 or not:

Variables:

var n=?;

Condition:

n % 3 == 0

Condition to check the number is Even or not:

Variables:

var n=?;

Condition:

n % 2 == 0

Condition to check the last digit of Number is 0 or not:

Variables:

var n=?;

Condition:

n % 10 == 0

Condition to check the multiplication of 2 numbers not equals to 3rd number or not:

Variables:

var a=?, b=?, c=?;

Condition:

a * b == c

Condition to check average of 4 subjects marks greater than 60 or not:

Variables:

var m1=?, m2=?, m3=? , m4=? ;

Condition:

(m1 + m2+ m3+ m4)/4 >= 60

Condition to check the sum of First 2 numbers equals to last digit of 3rd num or not:

Variables:

var a=?, b=? , c=? ;

Condition:

a + b == c% 10

Condition to check the given quantity of fruits exactly in dozens or not:

```
Variables:
        var quantity = ?;

Condition:
        quantity%12 == 0
```

Condition to person is eligible for Vote or Not:

```
Variables:
        var age=?;

Condition:
        age >= 18
```

Condition to check First Num is greater than both Second & Third Nums or Not:

```
Variables:
        var a=?, b=?, c=?;

Condition:
        a > b && a > c
```

Condition to check the number divisible by both 3 and 5 or not:

```
Variables:
        var n=?;

Condition:
        n % 3 == 0 && n % 5 == 0
```

Condition to check the number is in between 30 and 50 or not:

```
Variables:
        var n=?;

Condition:
        n >= 30 && n <= 50
```

Condition to check the student passed in all 5 subjects or Not:

```
Variables:
        var n=?;

Condition:
        n >= 30 && n <= 50
```

Condition to check the character is Vowel or Not:

Variables:

```
var ch = '?';
```

Condition:

```
ch == 'a' || ch == 'e' || ch== 'i' || ch=='o' || ch=='u'
```

Condition to check the character is Upper case alphabet or not:

Variables:

```
var ch='?';
```

Condition:

```
ch >= 'A' && ch <= 'Z'
```

Condition to check the character is Alphabet or not:

Variables:

```
var ch='?';
```

Condition:

```
(ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z')
```

Condition to check the 3 numbers equal or not:

Variables:

```
var a=?, b=?, c=?;
```

Condition:

```
a == b && b == c
```

Condition to check any 2 numbers equal or not among the 3 numbers:

Variables:

```
var a=? , b=? , c=?;
```

Condition:

```
a==b || b==c || c=a
```

Condition to check the 3 numbers are unique or not:

Variables:

```
var a=? , b=? , c=?;
```

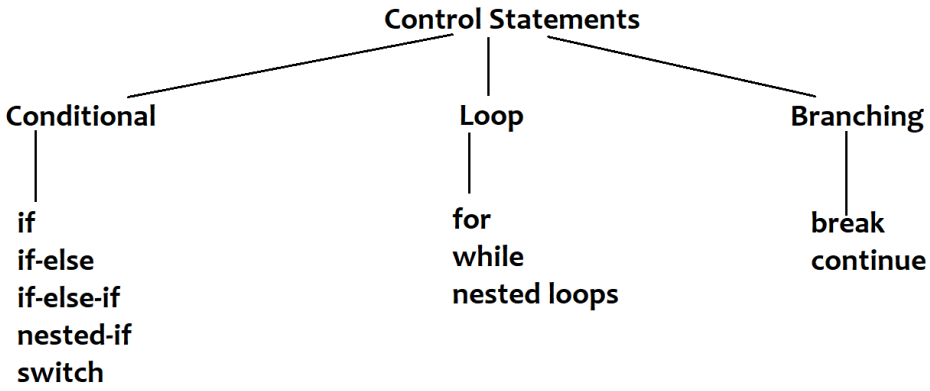
Condition:

```
a!=b && b!=c && c!=a
```

Control Statements

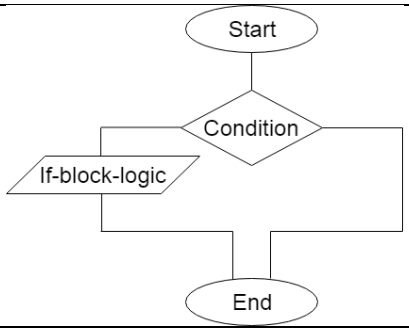
Control Statements:

- Statement is a line of code.
- Sequential Statements execute one by one from top to bottom.
- Control statements execute repeatedly and randomly based on conditions



If Block: Executes a block of instructions if the given condition is valid.

Syntax:	**Flow:**

if(condition)
{
Logic;
}
	

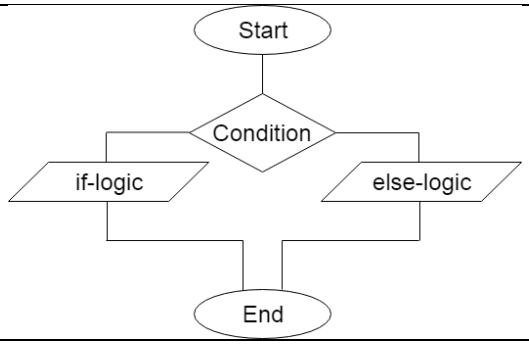
Program: Provide 15% discount to customer on bill amount if the bill amount is >5000

```
<html>
	<head>
		<script>
			function discount() {
				var bill = parseFloat(document.getElementById("bill").value);
				if(bill>5000) {
					var disc = 15/100 * bill ;
					bill = bill-disc ;
				}
				document.write("Final bill to pay : " + bill);
			}
		</script>
	</head>
```

```
        <body>
                Enter Bill Amount :     <input type="text" id="bill"/>
                                                <br/>
                <button onclick="discount()"> Final Bill to Pay </button>
        </body>
</html>
```

if - else block: Else block can be used to execute an optional logic if the given condition has failed.

Syntax:	**Flow:**

if (condition)
{
Logic….
}
else
{
Logic….
}
	

Program to check the person is eligible for Vote or Not:

```
<html>
        <head>
                <script>
                        function canVote(){
                                var age = paseInt(document.getElementById("age").value);
                                if(age>=18)
                                        document.write("you can vote");
                                else
                                        document.write("wait " + (18-age) + " years to vote");
                        }
                </script>
        </head>

        <body>
                Enter your Age :        <input type="text" id="age"/>
                                                <br/>
                <button onclick="canVote()"> can vote </button>
        </body>
</html>
```

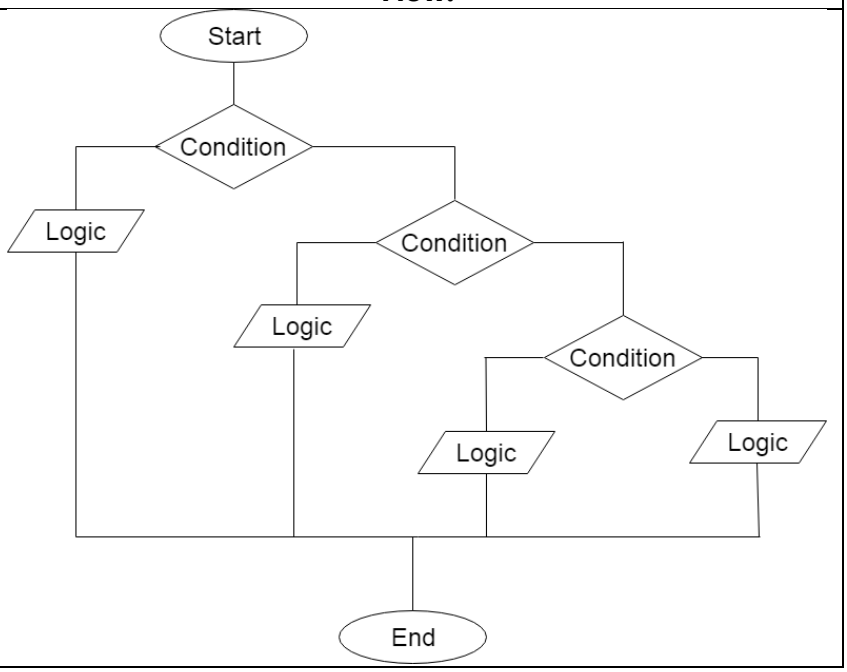
Program to check the number is Even or Not:

```
<html>
	<head>
		<script>
			function isEven(){
				var num = parseInt(document.getElementById("num").value);
				if(num%2==0)
					document.write(num + " is even");
				else
					document.write(num + " is not even");
			}
		</script>
	</head>
	<body>
		Enter number :<input type="text" id="num"/><br/>
		<button onclick="isEven()"> Even or Not </button>
	</body>
</html>
```

If-else-if ladder:

- It is allowed to defined multiple if blocks sequentially.
- It executes only one block among the multiple blocks defined.

Syntax:	Flow:

if (condition){
Logic....
}
else if (condition){
Logic....
}
else if (condition){
Logic....
}
else{
Logic....
}
	

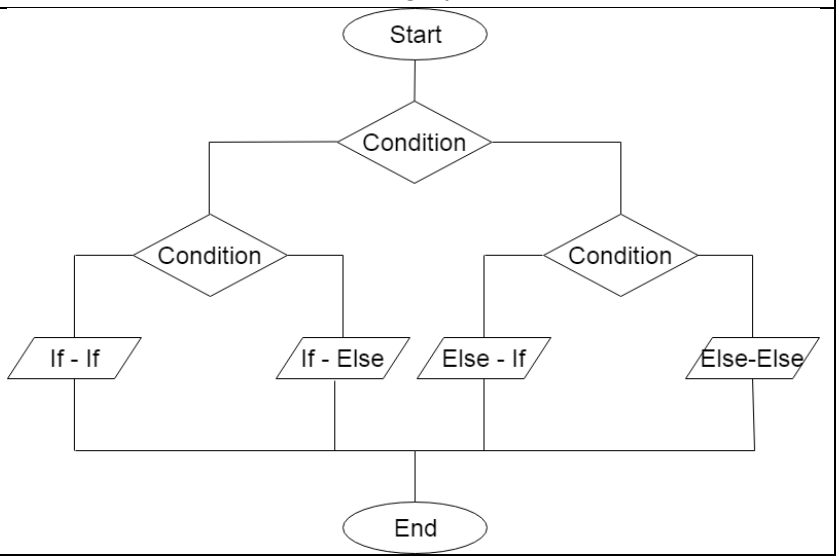
Program to check biggest among two numbers:

```
<html>
	<head>
		<script>
			function big(){
				var a = parseInt(document.getElementById("first").value);
				var b = parseInt(document.getElementById("second").value);
				var c = parseInt(document.getElementById("third").value);
				if(a>b && a>c)
					document.write("a is big");
				if(b>c)
					document.write("b is big");
				else
					document.write("c is big");
			}
		</script>
	</head>
	<body>
		First Number : <input type="text" id="first"/>br/>
		Second Number : <input type="text" id="second"/><br/>
		Third Number : <input type="text" id="third"/>br/>
		<button onclick="big()"> Biggest of 3 </button>
	</body>
</html>
```

Nested-if: Defining if block inside another if block. Inner block condition evaluates only when outer condition is valid.

Syntax:	**Flow:**

```
if (condition){
	if (condition)
		If-If
	else
		If-Else
}
else {
	if (condition)
		Else-If
	else
		Else-Else
}
```



Program to check the number is even or not only if it is positive.

```
<html>
	<head>
		<script>
			function big()
			{
				var n = paseInt(document.getElementById("num").value);
				if(n>0){
					if(n%2==0)
						document.write("Even number");
					else
						document.write("Odd number");
				}
				else
					document.write("Input number is negative");
			}
		</script>
	</head>
	<body>
		Enter Number : input type="text" id="num"/>br/>
		<button onclick="even()"> Even or Not </button>
	</body>
</html>
```

Loop Control Statements

Loop control statements:

- Loop control statements execute a block of instructions repeatedly as long as the condition is valid.
- Loops classified into
    - **For loop** – we use for loop when we know the number of iterations
        - Print 1 – 10 numbers
        - Print all elements of Array
    - **While Loop** – we use when we don't know the number of iterations
        - Print file text – we don't know how many lines are present
        - Print Database table details – we don't know how many records are present
    - **Do-While Loop** – When we want to execute loop at least once. Execute the Block and the check the condition from the next iteration.

Program to print numbers from 1 to 10 and 10 to 1:

```
<html>
	<head>
		<script>
			function print1to10(){
				for(var i=1 ; i<=10 ; i++){
					document.write("i val : " + i + "<br/>");
				}
			}
			function print10to1(){
				for(var i=10 ; i>=1 ; i--){
					document.write("i val : " + i + "<br/>");
				}
			}
		</script>
	</head>
	<body>
		<button onclick="print1to10()"> Print 1 to 10 </button>
		<button onclick="print10to1()"> Print 10 to 1 </button>
	</body>
</html>
```

Display the multiplication table of given input number:

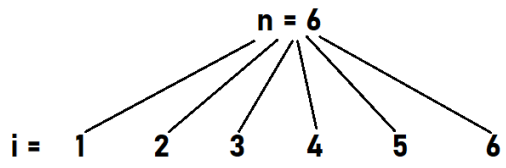
Enter table number: 6
6x1=6
6x2=12
..

```
<html>
	<head>
		<script>
			function printTable(){
				var n = parseInt(document.getElementById("num").value);
				for(var i=1 ; i<=10 ; i++){
					document.write(n + "x" + i + "=" + (n*i) + "<br/>");
				}
			}
		</script>
	</head>
	<body>
		Enter table number : <input type="text" id="num"/>
		<button onclick="printTable()"> print table </button>
	</body>
</html>
```

Print only even numbers in the given range:

```
<html>
	<head>
		<script>
			function evens(){
				var n = parseInt(document.getElementById("range").value);
				for(var i=1 ; i<=n ; i++){
					if(i%2==0)
						document.write(i + "<br/>");
				}
			}
		</script>
	</head>
	<body>
		Enter Limit : <input type="text" id="range"/>
		<button onclick="evens()"> Even Numbers </button>
	</body>
</html>
```

Display Factors for the given number:




```
for(i=1 ; i<=n ; i++)
{
    if(n%i == 0)
      print(i is a factor);
}
```

```
<html>
	<head>
		<script>
			function factors(){
				var n = parseInt(document.getElementById("num").value);
				for(var i=1 ; i<=n ; i++){
					if(n%i==0)
						document.write(i + " is a factor" + "<br/>");
				}
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="factors()"> Find factors </button>
	</body>
</html>
```

Program Count the number of factors for the given number:

```
<!doctype html>
<html>
	<head>
		<script>
			function factors(){
				var n = parseInt(document.getElementById("num").value);
				var count=0;
				for(var i=1 ; i<=n ; i++){
					if(n%i==0)
						count++;
				}
				document.write("Factors count is : " + count);
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="factors()"> Count Factors </button>
	</body>
</html>
```

Program to Check the input number is prime number or Not:

Prime number: The number which is having 2 factors

```
<html>
	<head>
		<script>
			function prime(){
				var n = parseInt(document.getElementById("num").value);
				var count=0;
				for(var i=1 ; i<=n ; i++){
					if(n%i==0)
						count++;
				}
				if(count==2)
					document.write(n + " is prime number");
				else
					document.write(n + " is not prime number");
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="prime()"> Check Prime or Not </button>
	</body>
</html>
```

Print sum of factors for the given number:

```
<html>
	<head>
		<script>
			function factorsSum(){
				var n = parseInt(document.getElementById("num").value);
				var sum=0;
				for(var i=1 ; i<=n ; i++){
					if(n%i==0){
						document.write(i + " is a factor" + "<br/>");
						sum=sum+i;
					}
				}
				document.write("Factors Sum is : " + sum);
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="factorsSum()"> Factors Sum </button>
```

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

Program to check the input number is Perfect or not:

Perfect number: Sum of factors except itself is equals to the same number is called perfect number

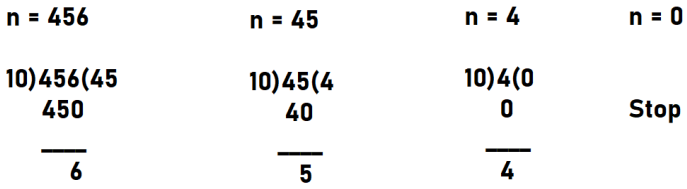
```
<html>
	<head>
		<script>
			function perfect(){
				var n = parseInt(document.getElementById("num").value);
				var sum=0;
				for(var i=1 ; i<n ; i++){
					if(n%i==0)
						sum=sum+i;
				}
				if(n==sum)
					document.write(n + " is perfect");
				else
					document.write(n + " is not perfect");
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="perfect()"> Perfect Or Not </button>
	</body>
</html>
```

break: It is a branching statement that terminates the flow of loop or switch

```
<html>
	<body>
		<script>
			for(var i=1 ; i<=10 ; i++){
				if(i>5){
					break;
				}
				document.write("i val : " + i + "<br/>");
			}
		</script>
	</body>
</html>
```

While loop: We use while loop when we don't know the number of iterations of loop.

Program to count the number of digits in the given number:




```
<html>
	<head>
		<script>
			function digitsCount(){
				var n = parseInt(document.getElementById("num").value);
				var count=0;
				while(n>0){
					n = parseInt(n/10);
					count++;
				}
				document.write("Number of Digits : " + count);
			}
		</script>
	</head>
	<body>
		Enter Number : <input type="text" id="num"/>
		<button onclick="digitsCount()"> Digits Count </button>
	</body>
</html>
```

Program to find the Sum of digits in the given number:

```
<html>
	<head>
		<script>
			function digitsSum(){
				var n = parseInt(document.getElementById("num").value);
				var sum=0;
				while(n>0){
					var rem = n%10;
					sum = sum + rem;
					n = parseInt(n/10);
				}
				document.write("Sum of digits : " + sum);
```

```
                }
            </script>
        </head>
        <body>
            Enter Number : <input type="text" id="num"/>
            <button onclick="digitsSum()"> Digits Sum </button>
        </body>
</html>
```

Reverse Number Program:

```
<!doctype html>
<html>
        <head>
            <script>
                function reverse()
                {
                    var n = parseInt(document.getElementById("num").value);
                    var rev=0;
                    while(n>0)
                    {
                        var rem = n%10;
                        rev = rev*10 + rem;
                        n = parseInt(n/10);
                    }
                    document.write("Reverse number is : " + rev);
                }
            </script>
        </head>

        <body>
            Enter Number : <input type="text" id="num"/>
            <button onclick="reverse()"> Reverse Number </button>
        </body>
</html>
```

JavaScript - Arrays

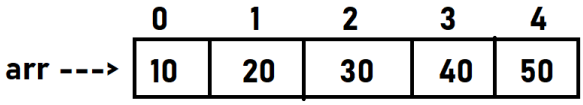
Array: Array can store more than one element but of same type

```
var arr = [10, 20, 30, 40, 50];
var marks = [56, 78, 34, 90, 67];
```

Memory Allocation:

- All array elements store in consecutive memory locations.
- We access element using their index.
- Index starts with 0 to length-1

var arr = {10, 20, 30, 40, 50};



Display the elements of Array:

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write(arr[0] + " <br/> ");
			document.write(arr[1] + " <br/> ");
			document.write(arr[2] + " <br/> ");
			document.write(arr[3] + " <br/> ");
			document.write(arr[4] + " <br/> ");
		</script>
	</body>
</html>
```

Note: It is recommended to use loops to process information of array

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Array elements are : <br/>");
			for (var i=0 ; i<5 ; i++){
				document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

length: It is a pre-defined variable that returns the length of array

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Array elements are : <br/>");
			for (var i=0 ; i<arr.length ; i++){
				document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

Program to display array elements in reverse order:

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Reverse Array elements are : <br/>");
			for (var i=arr.length-1 ; i>=0 ; i--){
				document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

Program to print sum of Array elements:

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			var sum = 0;
			for (var i=0 ; i<arr.length ; i++)
			{
				sum = sum + arr[i];
			}
			document.write("Sum of elements : " + sum);
		</script>
	</body>
</html>
```

Program to print only even numbers in the given array:

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [5, 9, 2, 8, 7, 1, 6, 4, 3];
			document.write("Even elements in array : <br/>");
			for (var i=0 ; i<arr.length ; i++){
				if(arr[i]%2==0)
					document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

Program to display the biggest element in the array:

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [5, 7, 2, 8, 6, 1, 9, 4, 3];
			var big = arr[0];
			for (var i=1 ; i<arr.length ; i++)
			{
				if(arr[i] > big)
					big = arr[i];
			}
			document.write("Biggest element : " + big);
		</script>
	</body>
</html>
```

Java Script - Strings

String:

- One dimensional character array.
- String is an object.
- String object providing functionality to perform string operations.

Syntax: var name = "amar" ;

Program to define a string and display:

```
<!doctype html>
<html>
	<body>
		<script>
			var name = "amar" ;
			document.write("Hello " + name);
		</script>
	</body>
</html>
```

Find the length of String: 'length' property is used to find the length of the string.

```
<!doctype html>
<html>
	<body>
		<script>
			var name = "amar" ;
			document.write("Length is : " + name.length);
		</script>
	</body>
</html>
```

JavaScript providing pre-defined function to process String objects:

Function	Description
charAt()	returns the character of specified index
concat()	merge 2 strings
indexOf()	it returns index value of specified character.
substr()	it is used to fetch sub string from the given string
toLowerCase()	converts upper case characters into lower case
to UpperCase()	converts lower case characters into upper case
toString()	Converts any object into String.
valueOf()	provides primitive value of String object.
split()	split string into substring array.
trim()	it trims white spaces from the left and right side of the string.

Display first and last character of specified string:

```
<html>
	<body>
		<script>
			var name = "amar" ;
			document.write("String : " + name + "<br/>");
			document.write("First char: " + name.charAt(0) + "<br/>");
			document.write("Last char: " + name.charAt(name.length-1));
		</script>
	</body>
</html>
```

Display all characters in the String one by one:

```
<html>
	<body>
		<script>
			var name = "amar" ;
			for(var i=0 ; i<= name.length-1 ; i++) {
				document.write(name.charAt(i) + "<br/>");
			}
		</script>
	</body>
</html>
```

Display String elements in reverse order:

```
<html>
	<body>
		<script>
			var name = "amar" ;
			for(var i=name.length-1 ; i>=0 ; i--){
				document.write(name.charAt(i) + "<br/>");
			}
		</script>
	</body>
</html>
```

Display only digits in the given string:

```
<html>
	<body>
		<script>
			var name = "amar@123" ;
			for(var i=0 ; i<name.length ; i++){
				var ch = name.charAt(i);
```

```
                        if(ch>='0' && ch<='9'){
                                document.write(name.charAt(i) + "<br/>");
                        }
                }
        </script>
    </body>
</html>
```

Display the count of Upper-case characters, lower case characters, digits and symbols:

```
<html>
    <body>
        <script>
            var name = "Amar@123" ;
            var c1=0, c2=0, c3=0, c4=0;
            for(var i=0 ; i<name.length ; i++){
                var ch = name.charAt(i);
                if(ch>='A' && ch<='Z')
                    c1++;
                else if(ch>='a' && ch<='z')
                    c2++;
                else if(ch>='0' && ch<='9')
                    c3++;
                else
                    c4++;
            }
            document.write("Upper case characters : " + c1 + "<br/>");
            document.write("Lower case characters : " + c2 + "<br/>");
            document.write("Digits case characters : " + c3 + "<br/>");
            document.write("Symbols case characters : " + c4 + "<br/>");
        </script>
    </body>
</html>
```

Display Strings in the given array:

```
<html>
    <body>
        <script>
            var arr = ["HTML", "CSS" , "JavaScript", "Angular" , "React" , "VueJS"];
            document.write("Array items are : <br/>");
            for(var i=0 ; i<arr.length ; i++){
                document.write(arr[i] + "<br/>");
            }
        </script>
```

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

Display word count in the given sentence(String):

```
<html>
	<body>
		<script>
			var line = "This is online web technologies session" ;
			var arr = line.split(" ");
			document.write("Words count : " + arr.length);
		</script>
	</body>
</html>
```

Display words in the given string:

```
<!doctype html>
<html>
	<body>
		<script>
			var line = "This is online web technologies session" ;
			var arr = line.split(" ");
			for(var i=0 ; i<arr.length ; i++)
			{
				document.write(arr[i] + " <br/>");
			}
		</script>
	</body>
</html>
```

Regular Expressions

Regular Expression:

- Regular expression is a set of characters called Pattern.
- These Patterns are used to search required information from the given text.
- JavaScript providing RegExp object with Properties and Functions.

Some of the Regular Expressions for Alphabets and Numerical Validations

[abc] – find any character in the brackets a, b or c
[a-z] – find characters a to z lowercase.
[0-9] – Find all digits

[^xyz] – find any character other than specified in the brackets.
[^abc] Find any character NOT between the brackets
[^0-9] Find any character NOT between the brackets (any non-digit)

(word) – find the "word" specified in the round brackets
[abc|xyz] – find either the characters a,b,c or x,y,z

Non-Empty Text Field validation: In the program, we are not using any regular expression. We check the length of input to specify Error message if the user not entered any value.

```
<html>
	<head>
		<script>
			function valid(){
				var x = document.getElementById('val').value;
				if(x.length == 0){
					alert("Please Enter Name");
					document.getElementById('val').focus();
					return false;
				}
				return true;
			}
		</script>
	</head>
	<body>
		<form>
			Enter Name : <input type='text' id='val'/>
			<input type='button' onclick="valid()" value='Submit' />
		</form>
	</body>
</html>
```

Program to Validate only Numerical input:

```
<html>
	<head>
		<script>
			function valid(){
				var x = document.getElementById('val').value;
				var expr = new RegExp("^[0-9]");
				if(expr.test(x)){
					alert("Thank you for your valid input");
				}
				else{
					alert("Please eneter numbers only");
					document.getElementById('val').focus();
				}
			}
		</script>
	</head>
	<body>
		<form>
			Enter Age : <input type='text' id='val'/>
			<input type='button' onclick="valid()" value='Submit' />
		</form>
	</body>
</html>
```

Program to Validate only Alphabetical Input:

```
<!doctype html>
<html>
	<head>
		<script>
			function valid(){
				var x = document.getElementById('val').value;
				var expr = new RegExp("^[a-zA-Z]");
				if(expr.test(x)){
					alert("Thank you for your valid input");
				}
				else{
					alert("Name should not contain numbers");
					document.getElementById('val').focus();
				}
			}
		</script>
	</head>
```

```
        <body>
                <form>
                        Enter Name : <input type='text' id='val'/>
                        <input type='button' onclick="valid()" value='Submit' />
                </form>
        </body>
</html>
```

[ ] - Square brackets : Square brackets specify a set of characters you wish to match.

Expression	String	Matched?
[abc]	a	1 match
	ac	2 matches
	Hey Jude	No match
	abc de ca	5 matches

- Here, [abc] will match if the string you are trying to match contains any of the a, b or c.
- You can also specify a range of characters using - inside square brackets.
- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].
- You can complement (invert) the character set by using caret ^ symbol at the start of a square-bracket.
- [^abc] means any character except a or b or c.
- [^0-9] means any non-digit character.

. **– Period :** A period matches any single character (except newline '\n').

Expression	String	Matched?
..	a	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

---

^ - Caret : The caret symbol ^ is used to check if a string **starts with** a certain character.

Expression	String	Matched?
^a	a	1 match
	abc	1 match
	bac	No match
^ab	abc	1 match
	acb	No match (starts with followed by b)a but not

$ - Dollar : The dollar symbol $ is used to check if a string **ends with** a certain character.

Expression	String	Matched?
a$	a	1 match
	formula	1 match
	cab	No match

*** - Star : T**he star symbol * matches **zero or more occurrences** of the pattern left to it.

Expression	String	Matched?
ma*n	mn	1 match
	man	1 match
	mann	1 match
	main	No match (a is not followed by n)
	woman	1 match

+ - Plus : The plus symbol + matches one or more occurrences of the pattern left to it.

Expression	String	Matched?
ma+n	mn	No match (no a character)
	man	1 match
	mann	1 match
	main	No match (a is not followed by n)
	woman	1 match

? - Question Mark : The question mark symbol ? matches **zero or one occurrence** of the pattern left to it.

Expression	String	Matched?
ma?n	mn	1 match
	man	1 match
	maan	No match (more than one a character)
	main	No match (a is not followed by n)
	woman	1 match

{} – Braces: Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.

Expression	String	Matched?
a{2,3}	abc dat	No match
	abc daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

This RegEx [0-9]{2, 4} matches at least 2 digits but not more than 4 digits.

Expression	String	Matched?
[0-9]{2,4}	ab123csde	1 match (match at ab123csde)
	12 and 345673	3 matches (12, 3456, 73)
	1 and 2	No match

| - Alternation : Vertical bar | is used for alternation (or operator).

Expression	String	Matched?
a|b	cde	No match
	ade	1 match (match at ade)
	acdbea	3 matches (at acdbea)

Here, a|b match any string that contains either a or b

() – Group : Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

Expression	String	Matched?
(a|b|c)xz	ab xz	No match
	abxz	1 match (match at abxz)
	axz cabxz	2 matches (at axzbc cabxz)

\ - Backslash

- Backslash \ is used to escape various characters including all metacharacters. For example,
- \$a match if a string contains $ followed by a. Here, $ is not interpreted by a RegEx engine in a special way.
- If you are unsure if a character has special meaning or not, you can put \ in front of it. This makes sure the character is not treated in a special way.

Special Sequences : Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

\A - Matches if the specified characters are at the start of a string.

Expression	String	Matched?
\Athe	the sun	Match
	In the sun	No match

\b - Matches if the specified characters are at the beginning or end of a word.

Expression	String	Matched?
\bfoo	football	Match
	a football	Match

foo\b	a football	No match
	the foo	Match
	the afoo test	Match
	the afootest	No match

\B - Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.

Expression	String	Matched?
\Bfoo	football	No match
	a football	No match
foo\B	a football	Match
	the foo	No match
	the afoo test	No match
	the afootest	Match

\d - Matches any decimal digit. Equivalent to [0-9]

Expression	String	Matched?
\d	12abc3	3 matches (at 12abc3)
	JavaScript	No match

\D - Matches any non-decimal digit. Equivalent to [^0-9]

Expression	String	Matched?
\D	1ab34"50	3 matches (at 1ab34"50)
	1345	No match

\s - Matches where a string contains any whitespace character.

Expression	String	Matched?
\s	JavaScript RegEx	1 match
	JavaScriptRegEx	No match

\S - Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v].

Expression	String	Matched?
\S	a b	2 matches (at a b)
		No match

ES6 - Features

ES6 Features

ES6 Features: ES6 brought significant changes to the JavaScript language. It introduces several new features such as, block-scoped variables, new loop for iterating over arrays and objects, template literals, and many other enhancements

ES6 features are:

1. The let keyword
2. The const keyword
3. Arrow Functions
4. For/of
5. Map Objects
6. Set Objects
7. Classes
8. Function Rest Parameter
9. String.includes()
10. String.startsWith()
11. String.endsWith()
12. Array.from()
13. Array keys()
14. Array find()
15. Array findIndex()
16. New Math Methods
17. New Number Properties
18. New Number Methods
19. New Global Methods
20. Object entries
21. JavaScript Modules

Var, Let and Const

Var	Let	Const
Variable can be either Global Scope or Local Scope	Variable is Local scope	Variable is Local Scope
Re-Declaration or Updating of variable is possible	Only Updating possible	Nothing is possible.
Hoisting allowed – Variables initializes with "undefined"	Hoisting allowed – Variables not initializes with values.	Hoisting allowed – Variables not initializes with values.
Can be declared without being initialized	Can be declared without being initialized	Must be initialized with value at the time of declaration.

Clear explanation about Var, Let and Const:

Var: var declarations are globally scoped or function/locally scoped.

```
var x = 10;
function myFun() {
        var y = 20;
}
```

var variables can be re-declared:

```
var x = 10;
var x = 20;
```

Display the value of re-declared variable as follows:

```
var x = 10;
var times = 4;
if (times > 3) {
        var x = 20;
}
console.log(x) // display 20
```

Hoisting of var: variables and function declarations are moved to the top of their scope before code execution.

if we do this:

```
console.log (x);
var x = 10;
```

it is interpreted as this: (initialized with a value of undefined)

```
var x;
console.log(x); // x is undefined
x = 10
```

Let: let is block scope. So a variable declared in a block with let is only available within block.

```
let x = 10;
let times = 4;
if (times > 3) {
        let y = 20;
        console.log(y); // 20
}
console.log(y) // y is not defined
```

let can be updated:	**but not re-declared:**
let x = 10; x = 20;	let x = 10; let x = 20; // error: already declared

However, if the same variable is defined in different scopes, there will be no error:

```
let x = "say Hi";
if (true) {
        let x = "say Hello instead";
        console.log(x); // "say Hello instead"
}
console.log(x); // "say Hi"
```

Hoisting of let: Just like var, let declarations are hoisted to the top. Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error.

Const:

- Variables declared with the const maintain constant values.
- const declarations are block scoped
- const cannot be updated or re-declared

```
const x = "say Hi";
x = "say Hello instead";// error: Assignment to constant variable.
```

Note: Every const declaration, therefore, must be initialized at the time of declaration.

Hoisting of const: Just like let, const declarations are hoisted to the top but are not initialized.

Lambda Expressions

Lambda expression:

- Writing a function in a short format.
- **Function:** become single line expression.

Anonymous functions:

- Anonymous – No name
- Function without name is called "Lambda expression"

Method example:

```
function increment(x)
{
        x=x+1;
        return x;
}
```

Lambda expression:

```
x => x+1
```

Lambda expression with zero arguments:

```
() => {
            document.write("Hello");
        };
```

Lambda expression with input values:

```
(x, y) => {
            document.write(x+y);
         };
```

Lambda expression with return values:

```
(x, y) => {
            return x+y;
         };
```

Can be defined simply like:

```
(x,y) => x+y ;
```

Program to define Arrow function to perform multiplication operation:

```
<html>
    <body>
        <h1> Lambda expression / Arrow Function </h1>
        <p> Defining a function  as an expression in single line </p>
        <p> Name not required - function keyword not required </p>
```

```
            <p> Return type not required </p>
            <p id="res"> </p>
            <script>
                var multiply= (x, y) => x*y ;
                document.getElementById("res").innerHTML = multiply(3,5);
            </script>
        </body>
</html>
```

Arrow function can be define in more than one statement using { }

```
<!doctype html>
<html>
        <body>
            <h1> Lambda expression / Arrow Function </h1>

            <p> Defining a function  as an expression in single line </p>
            <p> Name not required - function keyword not required </p>
            <p> Return type not required </p>

            <p id="res"> </p>

            <script>
                var z = (x, y) => {
                    var sum = x*y ;
                    return sum
                }
                document.getElementById("res").innerHTML = z(5,5);
            </script>
        </body>
</html>
```

For/Of Loop

For/Of Loop:

- It is called enhanced for loop
- It is used to iterate Array or Collection easily.

Note: Generally, we iterate an array by specifying their index values.

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Array elements are : <br/>");
			for (var i=0 ; i<arr.length ; i++){
				document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

In case of For/Of loop, no need to specify the index to iterate the elements of Array or Collection:

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Array elements are : <br/>");
			for(var x of arr){
				document.write(x + "<br/>");
			}
		</script>
	</body>
</html>
```

We can iterate any type of array using for/of easily:

```
<html>
	<body>
		<script>
			var arr=["HTML", "JavaScript", "CSS", "Angular", "VueJS"];
			document.write("Array elements are : <br/>");
			for(var s of arr){
				document.write(s + "<br/>");
			}
		</script>
	</body>
</html>
```

Limitations: For/of loop can process elements only in forward direction. For/of loop can process elements one by one

Display Array in Reverse Order: We must use for loop only

```
<html>
	<body>
		<script>
			var arr = [10, 20, 30, 40, 50];
			document.write("Array elements are : <br/>");
			for(var i=arr.length-1 ; i>=0 ; i--){
				document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

for/of loop: To process elements of Array/Collection only in forward direction.

Display even number in the given array using for-loop:

```
<html>
	<body>
		<script>
			var arr = [5, 2, 9, 1, 6, 4, 7, 3];
			document.write("Even elements : <br/>");
			for(var i=0 ; i<arr.length ; i++){
				if(arr[i]%2==0)
					document.write(arr[i] + "<br/>");
			}
		</script>
	</body>
</html>
```

The above program we can easily perform using for/of loop:

```
<html>
	<body>
		<script>
			var arr = [5, 2, 9, 1, 6, 4, 7, 3];
			document.write("Even elements : <br/>");
			for(var x of arr){
				if(x%2==0)
					document.write(x + "<br/>");
			}
		</script>
	</body>
</html>
```

JavaScript Maps

Map:

- Map store elements using keys. (Key=Value pairs)
- Map working like dictionary.
- Keys must be unique (not duplicated)
- Values can be duplicated

For example, we process accounts using account numbers because names can be duplicated but account numbers must be unique.

var accounts = {101 : 'Amar' , 102 : 'Sathya', 103 : 'Amar'};

Map object providing pre-defined methods to process:

Clear()	removes all elements from map object
Delete(key)	removes element associated with specified key.
Get(key)	returns the element associated with the key
Has(key)	returns true if value associated with the key, else returns false
Keys()	returns set of keys in this map object
Values()	returns set of Values in this map object
Set(key, value)	set the value of existing key in the Map

Create Map and display element with key:

```
<!doctype html>
<html>
	<body>
		<script>
			var accounts = new Map([
				[101, 'Amar'],
				[102, 'Annie'],
				[103, 'Sathya']]);
			document.write(accounts.get(102));
		</script>
	</body>
</html>
```

Display all keys from the Map:

```
<!doctype html>
<html>
	<body>
		<script>
			var accounts = new Map([
				[101, 'Amar'],
				[102, 'Annie'],
```

```
                        [103, 'Sathya'],
                        [104, 'Harsha']]);
                var arr = accounts.keys();
                document.write("keys : <br/>");
                for(var k of arr)
                {
                        document.write(k + "<br/>");
                }
        </script>
    </body>
</html>
```

Display all elements of Map:

```
<!doctype html>
<html>
    <body>
        <script>
            var accounts = new Map([
                [101, 'Amar'],
                [102, 'Annie'],
                [103, 'Sathya'],
                [104, 'Harsha']]);
            var arr = accounts.keys();
            document.write("keys & Values : <br/>");
            for(var k of arr)
            {
                var v = accounts.get(k);
                document.write(k + " : " + v + "<br/>");
            }
        </script>
    </body>
</html>
```

JavaScript Sets

Array: Array stores duplicate values but same type of elements.
For example: marks = [56, 78, 56, 78, 90];

Map: Map stores Values with Keys. Keys must be unique where as values can be duplicated.
For example: accounts = [[101 : 'amar'], [102 : 'annie'], [103 : 'amar']];

Set: Set doesn't allow duplicate values.
For example: emp_ids = [101, 102, 103, 104]

Set object methods:

new Set()	Creates a new set
add()	Adds element to set
delete()	removes specified element from set
hash()	returns true if the value exists
clear()	remove all elements
forEach()	invokes each element
values()	returns an iterator with all values
keys()	same as values()
entries()	returns [key, value] pairs in Map but [value, value] pairs in Set

Create new set and add elements:

```
<!doctype html>
<html>
	<body>
		<script>
			var colors = new Set();

			colors.add("black");
			colors.add("blue");
			colors.add("green");
			colors.add("cyan");
			colors.add("red");

			document.write(colors);
		</script>
	</body>
</html>
```

Output : [Object Set]

We can display values of any collection using for-each loop:

```
<html>
	<body>
		<script>
			var colors = new Set();
			colors.add("black");
			colors.add("blue");
			colors.add("green");
			colors.add("cyan");
			document.write("Colors are : <br/>");
			for(var col of colors){
				document.write(col + "<br/>");
			}
		</script>
	</body>
</html>
```

Construct Set from the Array directly:

```
<html>
	<body>
		<script>
			var arr = ["black", "blue", "green", "cyan", "red"];
			var colors = new Set(arr);
			document.write("Colors are : <br/>");
			for(var col of colors){
				document.write(col + "<br/>");
			}
		</script>
	</body>
</html>
```

Note: If we create Set from Array – all duplicates will be removed automatically.

```
<html>
	<body>
		<script>
			var arr = ["a", "e", "i", "o", "u", "i", "e", "u", "a", "i"];
			var vowels = new Set(arr);
			document.write("Vowels are : <br/>");
			for(var v of vowels){
				document.write(v + "<br/>");
			}
		</script>
	</body>
</html>
```

Collect value from set and pass to the function to perform any operation in function:

```
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			for(var n of nums){
				even(n);
			}
			function even(x){
				if(x%2==0)
					document.write(x + "<br/>");
			}
		</script>
	</body>
</html>
```

forEach() method: invokes function for each set element

```
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			nums.forEach(function(n){
				if(n%2==0){
					document.write(n);
				}
			})
		</script>
	</body>
</html>
```

forEach() with lambda expression:

```
<!doctype html>
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			nums.forEach(n => {document.write(n);})
		</script>
	</body>
</html>
```

Sum of elements in Set using forEach() method and lambda expression:

```
<!doctype html>
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			var sum=0;
			nums.forEach(n => sum+=n)
			document.write("Sum : " + sum);
		</script>
	</body>
</html>
```

values(): this method returns an iterator with set elements:

```
<!doctype html>
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			var itr = nums.values();
			document.write(itr);
		</script>
	</body>
</html>
```

Output: [Object Set Iterator]

Iterate elements using for/of loop:

```
<!doctype html>
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			var itr = nums.values();

			for(var x of itr)
			{
				document.write(x);
			}
		</script>
	</body>
</html>
```

Iterate elements using next() method:

```
<!doctype html>
<html>
	<body>
		<script>
			var nums = new Set([7, 4, 2, 9, 3, 6, 1, 8]);
			var itr = nums.values();

			var x = itr.next();
			while(!x.done)
			{
				document.write(x.value);
				x = itr.next();
			}
		</script>
	</body>
</html>
```

Classes and Objects

Class:

- Class - is a collection of variables and methods
- Variables – Store information
- Methods – Process information

Object:

- Real world entity (Human, House, Car, Laptop, Mobile….)
- Object – memory allocation of class.

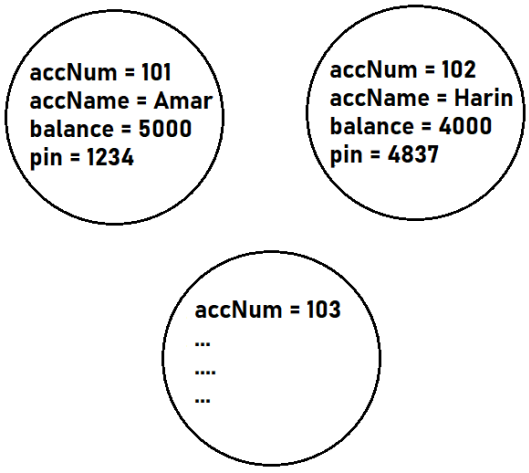
Banking Application:

Account class – Number of Account Holders (objects)

Class

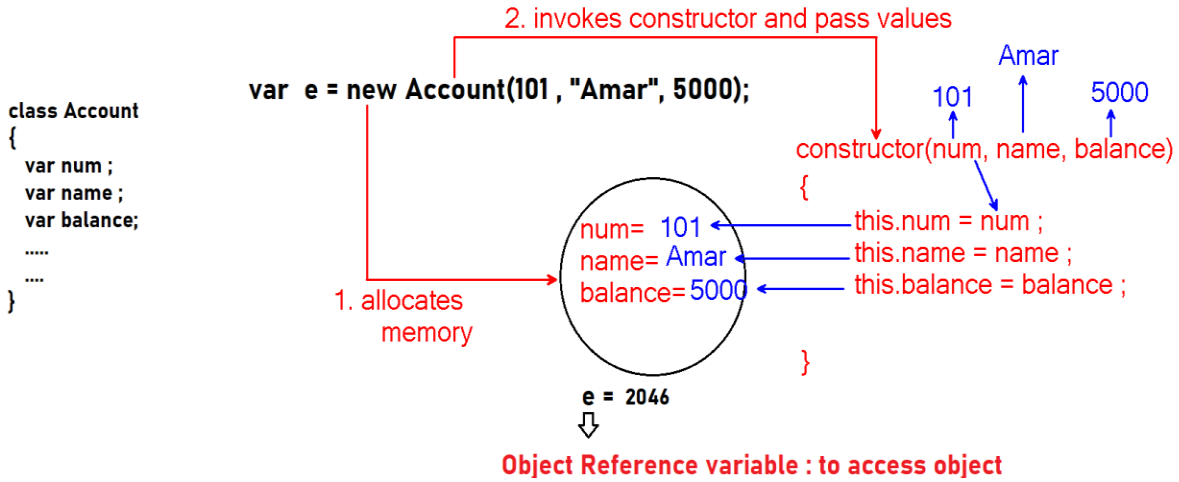
```
class Account
{
        var accNum;
        var accName;
        var balance ;
        var pin ;

        withdraw()
        {
            Logic...
        }
}
```

Objects



Constructor():

- We create object to class using "new" operator
- "new" operator allocates memory to variables.
- To provide initial values to variables we use constructor.
- "this" is used to point an object.



Create Object and Print details:

```
<!doctype html>
<html>
	<body>
		<script>
			class Account{
				constructor(num, name, balance){
					this.num = num;
					this.name = name;
					this.balance = balance;
				}
			}
			var e = new Account(101, "Amar", 5000);
			document.write("Num : " + e.num + "<br/>");
			document.write("Name : " + e.name + "<br/>");
			document.write("Balance : " + e.balance + "<br/>");
		</script>
	</body>
</html>
```

Create multiple objects and print details using method:

```
<!doctype html>
<html>
	<body>
		<script>
			class Account{
				constructor(num, name, balance){
					this.num = num;
					this.name = name;
					this.balance = balance;
```

```
                        }
                        details(){
                                document.write("Num : " + this.num + "<br/>");
                                document.write("Name : " + this.name + "<br/>");
                                document.write("Balance : " + this.balance + "<br/>");
                        }
                }
                var e1 = new Account(101, "Amar", 5000);
                var e2 = new Account(102, "Harin", 7000);
                var e3 = new Account(103, "Annie", 9000);
                e1.details();
                e2.details();
                e3.details();
        </script>
    </body>
</html>
```

Function with Rest parameter: The rest parameter(...) allows a function to collect any number of arguments.

Before rest feature (we use method overloading:
Overloading: Defining multiple functions with same name but with different arguments(parameters)

```
function add(x, y){
}
function add(x, y, z){
}
function add(x, y, z, w){
}
```

With rest feature, we can simply define as:

```
function add(...args){
}
```

Passing different length of arguments to function using rest parameter:

```
<!doctype html>
<html>
	<body>
		<script>
			function display(...args)
			{
				document.write("Input values are : <br/>");
				for(var x of args)
				{
					document.write(x + "<br/>");
				}
			}

			display(10,20);
			display(10,20,30,40,50);
			display("abc", "xyz", "lmn");
		</script>
	</body>
</html>
```

Adding different length of numbers using rest parameter:

```
<!doctype html>
<html>
	<body>
		<script>
			function add(...args)
			{
				var sum=0;
				for(var x of args)
				{
					sum=sum+x;
				}
				document.write("Sum is : " + sum + "<br/>");
			}

			add(10,20);
			add(10,20,30,40,50);
			add(10,20,30);
		</script>
	</body>
</html>
```

JavaScript - Date Object

Date object: It is providing methods to display system date and time.

```
<html>
	<body>
		<script>
			var d = new Date();
			document.write("Date : " + d);
		</script>
	</body>
</html>
```

Formatting Date: Date object providing methods to get values of month, week, day, hour etc..

```
<html>
	<body>
		<script>
			var date = new Date();
			var day = date.getDate();
			var month = date.getMonth()+1;
			var year = date.getFullYear();
			document.write(day + "/" + month + "/"+year);
		</script>
	</body>
</html>
```

Note: we need to concatenate if the value is single digit

```
<html>
	<body>
		<script>
			var date = new Date();
			var day = date.getDate();
			if(day<10){
				day = "0"+day;
			}
			var month = date.getMonth()+1;
			if(month<10){
				month = "0"+month;
			}
			var year = date.getFullYear();
			document.write(day + "/" + month + "/"+year);
		</script>
	</body>
</html>
```

Program to display Date and Time:

```
<html>
	<body>
		<script>
			var date = new Date();

			var day = date.getDate();
			var month = date.getMonth()+1;
			var year = date.getFullYear();
			document.write("Date : " + day + "/" + month + "/"+year);

			var h = date.getHours();
			var m = date.getMinutes();
			var s = date.getSeconds();
			document.write("<br/>");
			document.write("Time : " + h + "-" + m + "-" + s);
		</script>
	</body>
</html>
```

JavaScript – Math Object

Math object: Providing number of pre-defined methods to perform simple mathematical operations on data.

Math.sqrt() method:

```
<html>
	<body>
		<script>
			var x = 144;
			var r = Math.sqrt(x);
			document.write("Square root of " + x + " is : " + r);
		</script>
	</body>
</html>
```

Math.random() method: Generate random numbers

```
<html>
	<body>
		<script>
			var x = Math.random();
			var y = Math.random();
			document.write("x val : " + x + "<br/>");
			document.write("y val : " + y);
		</script>
	</body>
</html>
```

Math.pow(m, n): find m^n value

```
<html>
	<body>
		<script>
			var x = Math.pow(2, 3);
			var y = Math.pow(5, 2);
			document.write("2 power 3 is : " + x + "<br/>");
			document.write("5 power 2 is : " + y);
		</script>
	</body>
</html>
```

Math.floor() :

- returns the lowest integer for the given number.
- For example,
    - 3 for 3.7
    - 5 for 5.9

Math.ceil():

- return the largest integer for the given number.
- For example,
    - 4 for 3.2
    - 6 for 5.1
    - 8 for 8.9

Math.round():

- returns the rounded integer near to the given number.
- If the fractional part is equal or greater than 0.5, it goes upper value as 1
- If the fractional part is less than 0.5, it goes to lower value as 0

```
<html>
        <body>
                <script>
                        document.write("floor of 3.2  is : " + Math.floor(3.2) + "<br/>");
                        document.write("floor of 3.8  is : " + Math.floor(3.8) + "<br/>");

                        document.write("ceil of 3.2  is : " + Math.ceil(3.2) + "<br/>");
                        document.write("ceil of 3.8  is : " + Math.ceil(3.8) + "<br/>");

                document.write("round of 3.2  is : " + Math.round(3.2) + "<br/>");
                document.write("round of 3.8  is : " + Math.round(3.8) + "<br/>");
                </script>
        </body>
</html>
```

Math.trunc(): it returns only Integer part of specified decimal value

Math.trunc(4.9); // returns 4

Math.sign():

returns -1 if input value is negative
returns 0 if input value is zero
returns +1 if input value is positive

Example:

Math.sign(-4) : returns -1

Math.cbrt() : returns cube root of x

String Methods

String.startsWith(): This method returns if the string begins with specified value.

```
<!doctype html>
<html>
	<body>
		<script>
			var line = "Online web technologies session";
			var found = line.startsWith("Online");
			if(found)
				document.write("Start with Online");
			else
				document.write("Not starts with Online");
		</script>
	</body>
</html>
```

String.endsWith(): Check the given string ends with specified value or not.
If ends with value, it returns true else it returns false.

```
<!doctype html>
<html>
	<body>
		<script>
			var line = "Online web technologies session";
			var found = line.endsWith("Online");
			if(found)
				document.write("ends with Online");
			else
				document.write("Not ends with Online");
		</script>
	</body>
</html>
```

Array Methods

Array.from(): This method returns an Array object from given object with length property.

```
<!doctype html>
<html>
	<body>
		<script>
			var str = "ABCDEFGHIJ";
			var arr = Array.from(str);

			document.write("Array elements are : <br/>");
			for(var x of arr)
			{
				document.write(x + "<br/>");
			}
		</script>
	</body>
</html>
```

Array find(): The find() method returns the value of first array element that passes a test function.

Program to display the element larger than 18 in the given array:

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [4, 9, 13, 16, 19, 21, 31, 15];

			for(var x of arr)
			{
				if(x > 18)
				{
					document.write(x);
					break;
				}
			}
		</script>
	</body>
</html>
```

Above example using Array find():

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [4, 9, 13, 16, 20, 21, 31, 15];
			var res = arr.find(myFunction);
			document.write(res);

			function myFunction(value, index, array)
			{
				return value>18;
			}
		</script>
	</body>
</html>
```

Note: Function name can be anything but we cannot change parameters list.

Array findIndex(): It returns the index of first array element that passes to the function.

```
<!doctype html>
<html>
	<body>
		<script>
			var arr = [4, 9, 13, 16, 20, 21, 31, 15];
			var val = arr.find(myFunction);
			var index = arr.findIndex(myFunction);
			document.write(val + " index @ : " + index);

			function myFunction(value, index, array)
			{
				return value>18;
			}
		</script>
	</body>
</html>
```