**INDUSTRIAL TRAINING**

**An Industrial Training report submitted for partial fulfillment of requirements for the award of**

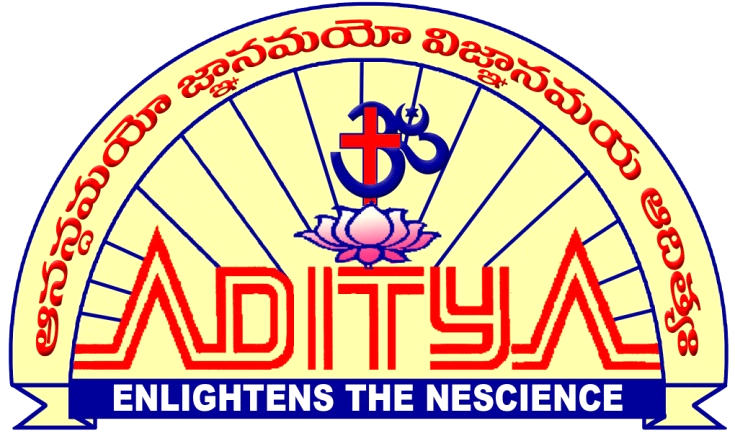
**DIPLOMA**

**IN**

**ELECTRONIC AND COMMUNICATION ENGINEERING**

SUBMITTED BY

**M.SAI NARMADA 21255-EC-035**



**DEPARTMENT OF ELECTRONIC AND COMMUNICATION ENGINEERING**

**ADITYA ENGINEERING COLLEGE -255**

**(II Shift Polytechnic)**

**(Approved by AICTE, Affiliated to SBTET)**

ADITYA NAGAR, ADB ROAD, SURAMPALEM-533437

[**www.aec.edu.in**](http://www.aec.edu.in)

(2021-2024)

**ADITYA ENGINEERING COLLEGE -255**

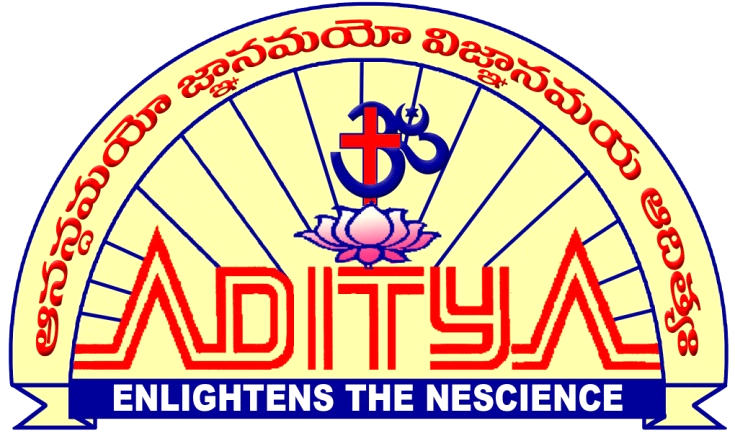
**(II Shift Polytechnic)**

**(Approved by AICTE, Affiliated to SBTET)**

**DEPARTMENT**

**OF**

**ELECTRONIC AND COMMUNICATION ENGINEERING**



**CERTIFICATE**

This is to certify that the industrial training report being submitted by **M.SAI NARMADA** bearing roll no.**21255-EC-081** in partial fulfillment for the award of the **Diploma in Electronic and Communication Engineering.** It is record of bonafied work carried out by me under the esteemed guidance and supervision of Mr. **SRI SURYA.**

**SRI SURYA R.V.V SRINIVAS**

**TRANING GUIDE HEAD OF THE DEPT.**

**LECTURER E.C.E ENGINEERING**

**EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

I would like to take this opportunity to express my profound sense of gratitude to Principal **Mr. S. T. V. S KUMAR**, Aditya polytechnic colleges for his refining comments and critical judgments of the industrial training.

I have great pleasure in expressing my deep sense of gratitude to our Head of the Department **Mr. R.V.V Srinivas,** Department of **Electronics and Communications** Engineering, Aditya Engineering College (II shift polytechnic) for providing all necessary support for successful completion of our training.

I would like to express my special thanks of gratitude to my trainer **Mr. J. Durga Prasad**, who gave me the golden opportunity to do this Industrial Training in **TECHNICAL HUB** which helped me in learn so many new things, Knowledge and Hands- on experience.

I wish to convey my sincere gratitude to my guide **Mr**.**Sri Surya** Department of **Electronics and Communications** Engineering, Aditya Engineering College (II shift polytechnic), Surampalem. We are highly Indebted to him for his guidance, timely suggestions at every stage and encouragement to complete this training successfully.

I thank all the staff members of our department & the college administration and all my friends who helped me directly and indirectly in carrying out this training successfully.

**Sincerely,**

**G.S.V.HARI BHARGAV,**

**21255-EC-035.**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Topic Name** | **Page No.** |
| **1** | **HTML** |  |
|  | Introduction to HTML | 6 |
|  | Basic HTML Document | 6 |
|  | Basic Tags | 6-7 |
|  | Heading Tags | 7-8 |
|  | Paragraph Tag | 8 |
|  | HTML Anchor Tag | 8-9 |
|  | HTML Image Tag | 9 |
|  | Presentation Formatting Tags | 9-13 |
|  | HTML Table Tag | 13-15 |
|  | HTML Form Tag | 16-17 |
|  | HTML Background Image | 17-19 |
| **2** | **CSS** |  |
|  | Introduction to CSS | 20 |
|  | Types of StyleSheets | 20-22 |
|  | CSS Selectors | 23-25 |
|  | CSS Background Properties | 25 |
|  | CSS Box Model | 25-26 |
|  | Block Level & Inline Level | 27-28 |
|  | CSS Position Property | 28-29 |
|  | CSS Pseudo Class | 30-31 |
|  | Z-index Property | 31-32 |
|  | CSS Overflow Property | 32-36 |
|  | Opacity and Box-Shadow | 37-38 |
|  | CSS Flex properties | 38-40 |
| **3** | **BOOTSTRAP** |  |
|  | Introduction to Bootstrap | 41 |
|  | Bootstrap CDN & Offline Link Inserting | 41-42 |
|  | Bootstrap Grid System | 42-46 |
|  | Bootstrap Images | 46-48 |
|  | Bootstrap Jumbotron | 49-50 |
|  | Bootstrap Page Header | 50-51 |
|  | Bootstrap Table | 51-53 |
|  | Bootstrap Forms | 53-64 |
|  | Bootstrap Modal | 64-66 |
|  | Bootstrap Alerts | 66-68 |
| **4** | **JAVASCRIPT** |  |
|  | Introduction to Javascript | 69 |
|  | Javascript Function | 69-70 |
|  | Javascript Arrays | 70-72 |
|  | Javascript Objects | 72-73 |
|  | Javascript Array of Objects | 73-74 |
|  | Javascript Array Properties | 75-76 |
|  | Javascript Operators | 79-85 |
|  | Javascript Looping Statement | 85 |
|  | Javascript DOM Elements | 86-88 |
|  | Javascript DOM Elements Manipulation | 88-89 |
|  | Javascript Events | 89-92 |
|  | Javascript String Methods | 92-94 |
|  | Javascript Array Methods | 95-96 |
|  | Javascript Form Validation | 96-101 |

**HTML**

**Introduction to HTML:**

HTML stands for Hypertext Markup Language, and it is the most widely used language to write Web Pages.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

**Basic Html Document :**

In its simplest form, following is an example of an HTML document:

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

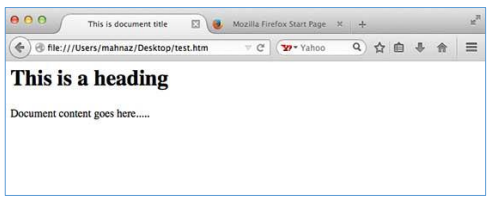
<h1>This is heading</h1>

<p>Document content goes here….</p>

</body>

</html>

Finally open it using a web browser like Internet Explorer or Google Chrome, or Firefox etc. It must show the following output:

****

**Basic Tags:**

HTML makes use of various tags to format the content. These tags are enclosed within angle braces . Except few tags, most of the tags have their corresponding closing tags.

There are Basic HTML tags shown below:

|  |  |
| --- | --- |
| **Tag** | **Description** |
| <!DOCTYPE html> | This tag defines the document type and HTML version. |
| <html> | This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>…</head> and document body which is represented by  <body>…</body> tags. |
| <head> | This tag represents the document's header which can keep other HTML tags like <title> , <link> ..etc. |
| <title> | The <title> tag is used inside the tag to mention the document title. |
| <body> | This tag represents the document's body which keeps other HTML tags like <h1> , <div> , <p> etc. |

**Heading Tags:**

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>, <h6>** While displaying any heading, browser adds one line before and one line after that heading.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>

<h4>Heading 4</h4>

<h5>Heading 5</h5>

<h6>Heading 6</h6>

</body>

</html>

**Output:**

**This is heading 1**

**This is heading 2**

**This is heading 3**

**This is heading 4**

**This is heading 5**

This is heading 6

**Paragraph Tag:**

The**<p>**tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening <p>and a closing </p> tag as shown below in the example:

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>This is document title</title>

</head>

<body>

<p>Here is first paragraph</p>

<p>Here is second paragraph</p>

<p>Here is third paragraph</p>

</body>

</html>

**Output**:

Here is a first paragraph of text.

Here is a second paragraph of text.

Here is a third paragraph of text.

**Html Anchor Tag:**

The <a> tag (anchor tag) in HTML is used to create a hyperlink on the webpage. This hyperlink is used to link the webpage to other web pages or some section of the same web page. It’s either used to provide an absolute reference or a relative reference as its “href” value**.**

**Syntax:**

<a href = "link"> Link Name </a>

**Attributes:**

The anchor tag contains many attributes which are listed below.

* [HTML <a> name Attribute](https://www.geeksforgeeks.org/html-a-name-attribute/): It is used to specify the anchor name. It is not supported by HTML 5 you can use the global id attribute instead.
* [HTML <a> rel Attribute](https://www.geeksforgeeks.org/html-a-rel-attribute/): It is used to specify the relation between the current document and the linked document.
* [HTML <a> shape Attribute](https://www.geeksforgeeks.org/html-a-shape-attribute/): It is used to specify the shape of the link. It is not supported by HTML 5.
* [HTML <a> type Attribute](https://www.geeksforgeeks.org/html-a-type-attribute/): It is used to specify the type of links.
* [HTML <a> target Attribute](https://www.geeksforgeeks.org/html-a-target-attribute/): It specifies the target link.
* [HTML <a> rev Attribute](https://www.geeksforgeeks.org/html-a-rev-attribute/): It is used to specify the relation between the linked document and the current document. It is not supported by HTML 5.

**HTML Image Tag:**

HTML ***<*img*>*** tag is used to add image inside webpage/website. Nowadays website does not directly add images to a web page, as the images are linked to web pages by using the <img> tag which holds space for the image.

**Syntax:**

<img src="" alt="" width="" height="">

**Attributes:**

The **<img>**tag has following attributes.

* [src](https://www.geeksforgeeks.org/html-img-src-attribute/): It is used to specify the path to the image.
* [alt](https://www.geeksforgeeks.org/html-img-alt-attribute/): It is used to specify an alternate text for the image. It is useful as it informs the user about what the image means and also due to any network issue if the image cannot be displayed then this alternate text will be displayed.
* crossorigin: It is used to import images from third-party sites that allow cross-origin access to be used with canvas.
* [height](https://www.geeksforgeeks.org/html-img-height-attribute/): It is used to specify the height of the image.
* [width](https://www.geeksforgeeks.org/html-img-width-attribute/): It is used to specify the width of the image.

**Presentation Formatting Tags:**

**Bold Text:**

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

**Example:**

< ! DOCTYPE html >

< html>

< head>

<title>Bold Text Example</title>

</head>

<body>

<p>The following word uses a <b> bold</b>typeface. </p>

</body>

</html>

**Output**:

The following word uses a **bold** typeface

**Italic Text:**

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

**Example:**

< ! DOCTYPE html

< html>

< head>

<title> Italic Text Example</title>

</head>

<body>

<p>The following word uses a < i > italicized< /i >typeface. </p>

</body>

</html>

**Output**:

The following word uses an *italicized* typeface.

**Underlined Text:**

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

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Underlined Text Example</title>

</head>

<body>

<p>The following word uses a <u>underlined</u> typeface.</p>

</body>

</html>

**Output:**

The following word uses an underlined typeface.

**Strike Text:**

Anything that appears within <strike>...</strike> element is displayed with strikethrough,

which is a thin line through the text as shown below:

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Strike Text Example</title>

</head>

<body>

<p>The following word uses a <strike>strikethrough</strike> typeface.</p>

</body>

</html>

**Output:**

The following word uses a strikethrough typeface.

**Superscript Text:**

The content of a <sup>...</sup> element is written in superscript; the font size used is the

Same size as the characters surrounding it but is displayed half a character's height above

the other characters.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Superscript Text Example</title>

</head>

<body>

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

</body>

</html>

**Output**:

The following word uses a superscript typeface.

**Subscript Text:**

The content of a <sub>...</sub> element is written in subscript; the font size used is the

same as the characters surrounding it, but is displayed half a character's height beneath the

other characters.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Subscript Text Example</title>

</head>

<body>

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

</body>

</html>

**Output**:

The following word uses a subscript typeface.

**Inserted Text:**

Anything that appears within <ins>...</ins> element is displayed as inserted text.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Inserted Text Example</title>

</head>

<body>

<p>I want to drink <del>cola</del><ins>wine</ins></p>

</body>

</html>

**Output**:

I want to drink ~~cola~~wine

**Deleted Text:**

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

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>Deleted Text Example</title>

</head>

<body

<p>I want to drink <del>cola</del><ins>wine</ins></p>

</body>

</html>

**Output**:

I want to drink ~~cola~~ wine

**HTML Table Tags:**

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

The HTML tables are created using the **<table>**tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>HTML Table Header</title>

</head>

<body>

<table border="1">

<tr>

<th>Name</th>

<th>Salary</th>

</tr>

<tr>

<td>Ramesh Raman</td>

<td>5000</td>

</tr>

<tr>

<td>Shabbir Hussein</td>

<td>7000</td>

</tr>

</table>

</body>

</html>

**Output:**

|  |  |
| --- | --- |
| **Name** | **Salary** |
| Ramesh Raman | 5000 |
| Shabbir Hussein | 7000 |

**Cellpadding and Cellspacing Attributes:**

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

Similar way you will use rowspan if you want to merge two or more rows

**Example:**

<!DOCTYPE html>

<html>

<table border="1">

<tr>

<th>Column 1</th>

<th>Column 2</th>

<th>Column 3</th>

</tr>

<tr>

<td rowspan="2">Row 1 Cell 1</td>

<td>Row 1 Cell 2</td>

<td>Row 1 Cell 3</td>

</tr>

<tr>

<td>Row 2 Cell 2</td>

<td>Row 2 Cell 3</td>

</tr>

<tr>

<td colspan="3">Row 3 Cell 1</td>

</tr>

</table>

</body>

</html>

**output:**

|  |  |  |
| --- | --- | --- |
| **Column 1** | **Column 2** | **Column 3** |
| Row 1 Cell 1 | Row 1 Cell 2 | Row 1 Cell 3 |
| Row 2 Cell 2 | Row 2 Cell 3 |
| Row 3 Cell 1 | | |

**Table Header, Body, and Footer:**

Tables can be divided into three portions: a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

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

* to create a separate table header.
* to indicate the main body of the table.
* to create a separate table footer.

A table may contain several elements to indicate different pages or groups of data. But it is notable that and tags should appear before **.**

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>HTML Table</title>

</head>

<body>

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

<thead>

<tr>

<td colspan="4">This is the head of the table</td>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="4">This is the foot of the table</td>

</tr>

</tfoot>

<tbody>

<tr>

<td>Cell 1</td>

<td>Cell 2</td>

<td>Cell 3</td>

<td>Cell 4</td>

</tr>

</tbody>

</table>

</body>

</html>

**Output:**

| This is the head of the table | | | |
| --- | --- | --- | --- |
| This is the foot of the table | | | |
| Cell 1 | Cell 2 | Cell 3 | Cell 4 |

**Html Form Tag:**

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

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

Top of Form

Top of Form

<form action =”scriptURL” method=” GET|POST”>

Form elements like input,textarea etc

</form>

Bottom of Form

**Form Attributes:**

**Action -**Backend script ready to process your passed data

**method -** Method to be used to upload data. The most frequently used are GET and POST

methods.

**HTML Form Controls:**

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

* Text Input Controls
* Checkboxes Controls
* Radio Box Controls
* Select Box Controls
* File Select boxes
* Hidden controls
* Clickable Buttons
* Submit and Reset Button

**Text Input Controls:**

There are three types of text input used on forms:

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

**<textarea>** tag.

**Attributes:**

Following is the list of attributes for **<input>**tag for creating text field

**type**  - Indicates the type of input control and for text input control it will be set to

**name**- Used to give a name to the control which is sent to the server to be recognized

**value** - This can be used to provide an initial value inside the control.

**Size**  - Allows to specify the width of the text-input control in terms of characters.

**Maxlength**-Allows to specify maximum number of characters a user can enter into text box.

**Html Backgrounds:**

By default, your webpage background is white in color. You may not like it, but no worries. HTML provides you following two good ways to decorate your webpage background.

* Html Background with Colors
* Html Background with Images

Now let's see both the approaches one by one using appropriate examples.

**Html Background with Colors:**Thebgcolor attribute is used to control the background of an HTML element, specifically page body and table backgrounds. Following is the syntax to use bgcolor attribute with any HTML tag.

<tagname bgcolor="color\_value"...>

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

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

<table bgcolor="lime" >

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

<table bgcolor="#f1f1f1" >

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

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

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>HTML Background Colors</title>

</head>

<body>

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

<table bgcolor="yellow" width="100%">

<tr>

<td> This background is yellow </td>

</tr>

</table>

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

<table bgcolor="#6666FF" width="100%">

<tr>

<td> This background is sky blue </td>

</tr>

</table>

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

<table bgcolor="rgb(255,0,255)" width="100%">

<tr>

<td> This background is green </td>

</tr>

</table>

</body>

</html>

**Output:**

This background is yellow

This background is skyblue

This background is green

**Html Background with Images:**

The background attribute can also be used to control the background of an HTML element, specifically page body and table backgrounds. You can specify an image to set background of your HTML page or table. Following is the syntax to use background attribute with any HTML tag.

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

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

The most frequently used image formats are JPEG, GIF and PNG images. Example Here are the examples to set background images of a table.

<!DOCTYPE html>

<html>

<head>

<title>HTML Background Images</title>

</head>

<body>

<!-- Set table background --><table background="/images/html.gif" width="100%" height="100">

<tr>

<td> This background is filled up with HTML image. </td>

</tr>

</table>

</body>

</html>

**Output:**

This background is filled up with HTML image.

**CSS**

**Introduction to CSS:**

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, as well as a variety of other effects.

CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

**Advantages of CSS:**

* CSS saves time - You can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
* Pages load faster - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So, less code means faster download times.
* Easy maintenance - To make a global change, simply change the style, and all the elements in all the web pages will be updated automatically.
* Superior styles to HTML - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* Multiple Device Compatibility - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
* Global web standards – Now HTML attributes are being deprecated and it is being recommended to use CSS. So it’s a good idea to start using CSS in all the HTML pages to make them compatible with future browsers.

**Types Of Style Sheets:**

Cascading Style Sheet(CSS) is used to set the style in web pages that contain HTML elements. It sets the background color, font-size,font-family,color,…etc property of elements on a web page.There are three types of CSS which are given below:-

* Inline CSS
* Internal or Embedded CSS
* External CSS

**Inline CSS:**

Inline CSS contains the CSS property in the body section attached with element is known as inline CSS. This kind of style is specified within an HTML tag using the style attribute.

**Example:**

<!DOCTYPE html>

<html>

<body>

<h1 style="color:blue;">A Blue Heading</h1>

<p style="color:red;"> A red paragraph.</p>

</body>

</html>

**Output:**

**A Blue Heading**

A red paragraph.

**Internal CSS Or Embedded CSS:**

This can be used when a single HTML document must be styled uniquely. The CSS rule set should be within the HTML file in the head section i.e., the CSS is embedded within the HTML file.

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

body{background-color: powderblue;}

h1{color: blue;}

p {color: red;}

</style>

</head>

<body>

<h1>This is a heading</h1>

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

</body>

</html>

**Output:**

**This is a heading**

This is a paragraph.

**External CSS:**

 External CSS contains separate CSS file which contains only style property with the help of tag attributes (For example class, id, heading, … etc). CSS property written in a separate file with .css extension and should be linked to the HTML document using **link** tag. This means that for each element, style can be set only once and that will be applied across web pages.

**Example:**

<!DOCTYPE html>  
<html>  
<head>  
<link rel="stylesheet" href="styles.css">  
</head>  
<body>  
<h1>This is a heading</h1>  
<p>This is a paragraph.</p>  
</body>  
</html>

**Style.css:**

h1 {  
  color: blue;  
}  
p {  
  color: red;  
}

**Output:**

**This is a heading**

This is a paragraph.

**Css Selectors:**

**CSS selectors** define the pattern to select elements to which a set of CSS rules are then applied.

**Basic Selectors:**

* **Universal Selector**
* The universal selector (\*) selects all HTML elements on the page.

**Example:**

<style>

\* {

text-align: center;

color: blue;

}

</style>

* **Element Selector:**
* The element selector selects HTML elements based on the element name.

**Example:**

<style>

p {  
  text-align: center;  
  color: red;  
}

</style>

* **Id Selector:**
* The id selector uses the id attribute of an HTML element to select a specific element.The id of an element is unique within a page, so the id selector is used to select unique element.
* To select an element with a specific id, write a hash (#) character, followed by the id of the element

**Example:**

#para1 {

text-align: center;

color: red;

}

* **Class Selector:**
* The class selector selects HTML elements with a specific class attribute.
* To select elements with a specific class, write a period (.) character, followed by the class name.

**Example:**

**.**center {  
  text-align: center;  
  color: red;  
}

* **Grouping Selectors:**
* The grouping selector selects all the HTML elements with the same style definitions.It will be better to group the selectors, to minimize the code.To group selectors, separate each selector with a comma.

**Example:**

h1,h2,p {

text-align: center;

color: red;  
}

* **Combinators:**

[**Descendant Combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Descendant_combinator)**:**

The " " (space) combinator selects nodes that are descendants of the first element.

**Syntax:** A B

**Example:** div span will match all [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) elements that are inside a [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) element.

[**Child Combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Child_combinator)**:**

The > combinator selects nodes that are direct children of the first element.

**Syntax:** A > B

**Example:** ul > li will match all [<li>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li) elements that are nested directly inside a [<ul>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul) element.

[**General Sibling Combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/General_sibling_combinator)**:**

The ~ combinator selects siblings. This means that the second element follows the first (though not necessarily immediately), and both share the same parent.

**Syntax:** A ~ B

**Example:** p ~ span will match all [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) elements that follow a [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p), immediately or not.

[**Column Combinator**](https://developer.mozilla.org/en-US/docs/Web/CSS/Column_combinator)**:**

The || combinator selects nodes which belong to a column.

**Syntax:** A || B

**Example:** col || td will match all [<td>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/td) elements that belong to the scope of the [<col>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/col).

**Css Background Properties:**

The CSS background properties are used to define the background effects for elements. There are lots of properties to design the background.   
CSS background properties are as follows:

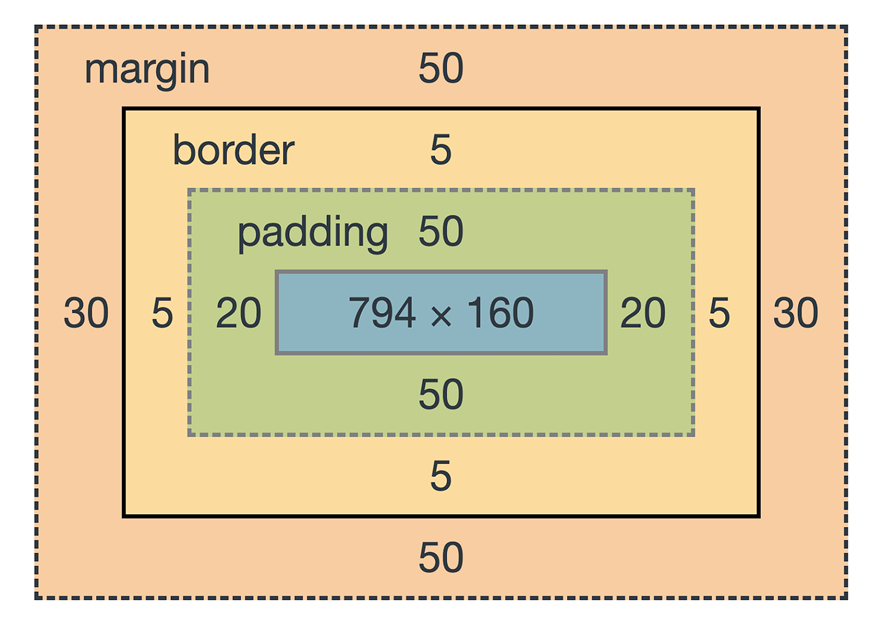
* [**background-color**](https://www.geeksforgeeks.org/css-background-color-property/#:~:text=The%20background%2Dcolor%20property%20in,to%20read%20for%20the%20user.)**:** The background-color property in CSS is used to specify the background color of an element.
* [**background-image**](https://www.geeksforgeeks.org/css-background-image-property/)**:**The background-image property is used to set one or more background images to an element.
* [**background-repeat**](https://www.geeksforgeeks.org/css-background-repeat-property/#:~:text=The%20background%2Drepeat%20property%20in,will%20be%20repeated%20or%20not.&text=The%20last%20image%20will%20be,fit%20in%20the%20browser%20window.): The background-repeat property in CSS is used to repeat the background image both horizontally and vertically.
* [**background-attachment**](https://www.geeksforgeeks.org/css-background-attachment-property/#:~:text=The%20property%20background%2Dattachment%20property,applied%20to%20all%20HTML%20elements.): The property background-attachment property in CSS is used to specify the kind of attachment of the background image with respect to its container.
* [**background-position**](https://www.geeksforgeeks.org/css-background-position-property/): In CSS body-position property is mainly used to set an image at a certain position.
* [**background-origin**](https://www.geeksforgeeks.org/css-background-origin-property/)**:** The background-origin is a property defined in CSS which helps in adjusting the background image of the webpage.
* [**background-clip**](https://www.geeksforgeeks.org/css-background-clip-property/)**:** The background-clip property in CSS is used to define how to extend background (color or image) within an element.
* **background-size**: The background-size [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) property sets the size of the element's background image. The image can be left to its natural size, stretched, or constrained to fit the available space.

**Css Box Model:**

[CSS](https://www.simplilearn.com/tutorials/css-tutorial/what-is-css) is the style sheet language for web page presentation and design, including colors, fonts, and layouts. CSS is a popular language used in various electronic devices, devices -from printers to large or small screens. This article will explore the CSS Box Model, which is the basics of every element on the web page.

The CSS box model is a container that contains multiple properties including borders, margin, padding, and the content itself. It is used to create the design and layout of web pages. According to the CSS box model, the web browser supplies each element as a square prism.

The following diagram illustrates the box model:-



The [CSS](https://www.javatpoint.com/css-tutorial) box model contains the different properties in CSS. These are listed below.

* Border
* Margin
* Padding
* Content

**Border Field**

* It is a region between the padding-box and the margin. Its proportions are determined by the width and height of the boundary.

**Margin Field**

* This segment consists of the area between the boundary and the edge of the border.
* The proportion of the margin region is equal to the margin-box width and height. It is better to separate the product from its neighbor nodes.

**Padding Field**

* This field requires the padding of the component. In essence, this area is the space around the subject area and inside the border-box. The height and the width of the padding box decide its proportions.

**Content Field**

* Material such as text, photographs, or other digital media is included in this area.
* It is constrained by the information edge, and its proportions are dictated by the width and height of the content enclosure.

**Block Level& Inline Level:**

* **Block Level Elements:**

Block level elements take up as much space as possible by default. Each block level element will start a new line on the page, stacking down the page. In addition to stacking vertically, block level elements will also take up as much horizontal space as possible.

The p element is an example of a block level element. Each new paragraph tag will appear on its own line vertically. Paragraphs with longer content will stretch all the way to the edge of the page.

**Examples of block level elements:**

* **<**p**>**
* <ul>,<ol><dl>
* <h1>to<h6>
* <article>,<section>

**Example:**

<html>

<body>

<p>This is paragraph tag</p>

<h6>This is heading tag</p>

</body>

<html>

**Output:**

This is paragraph tag

**This is heading tag**

**Inline Level Elements:**

Inline elements display in a line. They do not force text after them to a new line. An anchor (or link) is an example of an inline element. You can put several links in a row, and they will display in a line.

**Examples of inline elements:**

* <a>
* <strong>
* <em>
* <b>,<q>
* <mark>
* <span>

**Example:**

<!DOCTYPE html>

<html>

<body>

<p>This is an inline span

<span style="border: 1px solid black">Hello World</span>

element inside a paragraph</p>

<p>The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary</p>

</body>

</html>

**Output:**

This is an inline span Hello World element inside a paragraph.

The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary.

**CSS Position Property:**

* The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).
* There are five different position values:
* static
* relative
* fixed
* absolute

**position: static**

* HTML elements are positioned static by default.
* Static positioned elements are not affected by the top, bottom, left, and right properties.
* An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

div.static

{  
  position: static;  
  border: 3px solid #73AD21;  
}

**position: relative**

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

div.relative

{  
  position: relative;  
  left: 30px;  
  border: 3px solid #73AD21;  
}

**position: fixed**

* An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.
* A fixed element does not leave a gap in the page where it would normally have been located.

div.fixed

{

position: fixed;

bottom: 0;

right: 0;

width: 300px;

border: 3px solid #73AD21;

}

**position: absolute**

* An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
* If an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling**.**  
   div.absolute

**{**  
   position**:** absolute**;**  
  top**:** 80px**;**  
  right**:** 0**;**  
  width**:** 200px**;**  
  height**:** 100px**;**  
  **}**

**CSS Pseudo-classes**

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

* Style an element when a user mouses over it
* Style visited and unvisited links differently
* Style an element when it gets focus

**Syntax:**

Selector:pseudo-class {  
  property: value;  
}

**Anchor Pseudo-classes:**

Links can be displayed in different ways:

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

a:link /\* unvisited link \*/

{

color: red;

}

a:visited /\* visited link \*/

{

color: green;

}

a:hover /\* mouse over link \*/

{

color: hotpink;

}

a:active /\* selected link \*/

{

color: blue;

}

</style>

</head>

<body>

<h2>Styling a link depending on state</h2>

<p><b><a href="default.asp" target="\_blank">This is a link</a></b></p>

<p><b>Note:</b> a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.</p>

<p><b>Note:</b> a:active MUST come after a:hover in the CSS definition in order to be effective.</p>

</body>

</html>

**Output:**

**Styling a link depending on state**

[**This is a link**](https://www.w3schools.com/css/default.asp)

**Note:** a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.

**Note:** a:active MUST come after a:hover in the CSS definition in order to be effective.

**z-index property**:

* The z-index property specifies the stack order of an element.
* When elements are positioned, they can overlap other elements.
* The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
* An element can have a positive or negative stack order

**Example**:

<!DOCTYPE html>

<html>

<head>

<style>

.container {

position: relative;

}

.black-box {

position: relative;

z-index: 1;

border: 2px solid black;

height: 100px;

margin: 30px;

}

.gray-box {

position: absolute;

z-index: 3; /\* gray box will be above both green and black box \*/

background: lightgray;

height: 60px;

width: 70%;

left: 50px;

top: 50px;

}

.green-box {

position: absolute;

z-index: 2; /\* green box will be above black box \*/

background: lightgreen;

width: 35%;

left: 270px;

top: -15px;

height: 100px;

}

</style>

</head>

<body>

<div class="container">

<div class="black-box">Black box (z-index: 1)</div>

<div class="gray-box">Gray box (z-index: 3)</div>

<div class="green-box">Green box (z-index: 2)</div>

</div>

</body>

</html>

**Output**:

Green box (z-index: 2)

Black box (z-index: 1)

Gray box (z-index: 3)

**Overflow**:

The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

* Visible
* Hidden
* Scroll
* Auto

**Overflow-visible:**

By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box.

**Example**:

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-color: coral;

width: 200px;

height: 65px;

border: 1px solid;

overflow: visible;

}

</style>

</head>

<body>

<h2>Overflow: visible</h2>

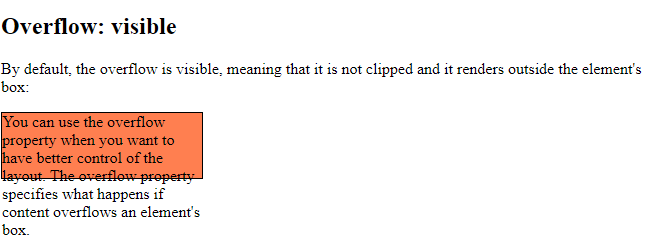
<p>By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:</p>

<div>You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.</div>

</body>

</html>

**Output**:



**Overflow-hidden:**

 The hidden value, the overflow is clipped, and the rest of the content is hidden.

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-color: coral;

width: 200px;

height: 65px;

border: 1px solid black;

overflow: hidden;

}

</style>

</head>

<body>

<h2>Overflow: hidden</h2>

<p>With the hidden value, the overflow is clipped, and the rest of the content is hidden:</p>

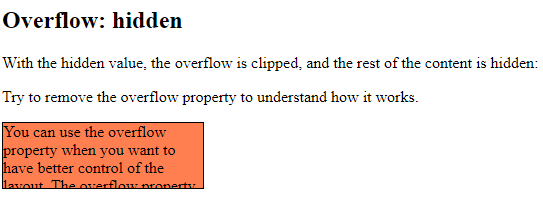
<p>Try to remove the overflow property to understand how it works.</p>

<div>You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.</div>

</body>

</html>

**Output**:



**Overflow-scroll:**

Setting the value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it)

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-color: coral;

width: 200px;

height: 100px;

border: 1px solid black;

overflow: scroll;

}

</style>

</head>

<body>

<h2>Overflow: scroll</h2>

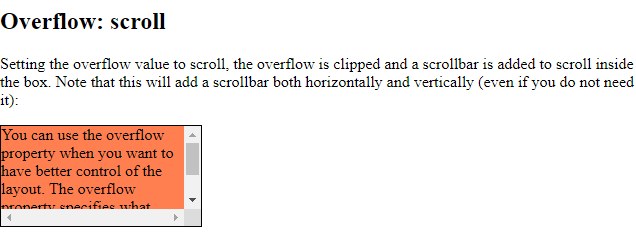
<p>Setting the overflow value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):</p>

<div>You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.</div>

</body>

</html>

**Output:**

****

**Overflow-auto:**

The auto value is similar to scroll, but it adds scrollbars only when necessary

**Example:**

<!DOCTYPE html>

<html>

<head>

<style>

div {

background-color: coral;

width: 200px;

height: 65px;

border: 1px solid black;

overflow: auto;

}

</style>

</head>

<body>

<h2>Overflow: auto</h2>

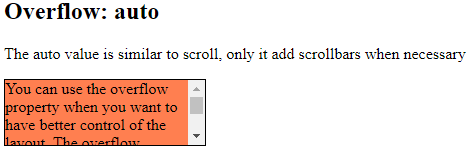
<p>The auto value is similar to scroll, only it add scrollbars when necessary:</p>

<div>You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.</div>

</body>

</html>

**Output:**

****

**Opacity And Box Shadow:**

**Opacity:**

The opacity property sets the opacity level for an element.

The opacity-level describes the transparency-level, where 1 is not transparent at all, 0.5 is 50% see-through, and 0 is completely transparent.

**Note:** When using the opacity property to add transparency to the background of an element, all of its child elements become transparent as well. This can make the text inside a fully transparent element hard to read. If you do not want to apply opacity to child elements, use RGBA color values instead.



**Syntax:**opacity: number|initial|inherit|;

|  |  |
| --- | --- |
| Value | Description |
| Number | Specifies the opacity. From 0.0(fully transparent) to 1.0 (fully opaque) |
| Initial | Sets this property to its default value. |
| Inherit | Inherits this property from its parent element. |

**Box Shadow:**

The box-shadow property attaches one or more shadows to an element.

**Syntax:**

box-shadow : h-offset v-offset blur color;

**Example:**

<html>

<head>

<style>

#example{

border : 1px solid;

padding : 10px;

box-shadow : 5px 10px 18px red;

}

</style>

</head>

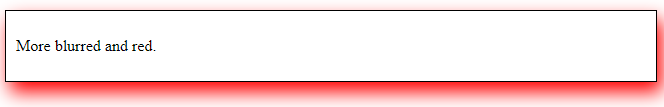
<body>

<div id=”example”><p>More blurred and red. </p></.div>

</body>

</html>

**Output:-**



**CSS flex Properties:**

The flex CSS shorthand property sets how a flex item will grow or shrink to fit the space available in its flex container.

**Display Flex Property:**

The flex property sets the flexible length on flexible items.

**syntax:**

display:flex;

**Example:**

#main {

width: 300px;

height: 300px;

border: 1px solid black;

display: flex;

}

**flex none:**

It sizes the item according to its width / height properties, but makes it fully inflexible.Same as 0 0 auto.

**Example:**

.h{

flex:none;

}

**flex auto:**

It sizes the item based on its width / height properties, but makes it fully flexible so that they absorb any extra space along the main axis.Same as 1 1 auto.

**Example:**

p{

flex: auto;

}

**flex-direction property:**

The flex-direction CSS property sets how flex items are placed in the flex container defining the main axis and the direction (normal or reversed).The default value of flex-direction is row. It is used to specify that the item has normal text direction.

The flex direction has the following properties:

* row
* row-reverse
* column
* column-reverse

**syntax:**

p{

flex: row-reverse;

}

**flex-grow property:**

The flex-grow property specifies how much the item will grow relative to the rest of the flexible items inside the same container.

**syntax:**

flex-grow: number| initial| inherit;

**Example:**

div {

display:flex;

flex-grow: 1;

}

**flex-shrink Property:**

The flex-shrink property specifies how the item will shrink relative to the rest of the flexible items inside the same container.

**Syntax:**

flex-shrink: number| initial| inherit;

**Example:-**

div

{

display:flex;

flex-shrink: 4;

}

**flex-wrap property:**

The flex-wrap CSS property sets whether flex items are forced onto one line or can wrap onto multiple lines. If wrapping is allowed, it sets the direction that lines are stacked.

**Syntax:**

flex-wrap: property;

**Example:**

div {

display: flex;

flex-wrap: wrap;

}

**flex initial:**

The item is sized according to its width and height properties. It shrinks to its minimum size to fit the container, but does not grow to absorb any extra free space in the flex container. This is equivalent to setting " flex: 0 1 auto ".

**Example:**

div{

flex: initial;

}

**flex inherit:**

Inherits this property from its parent element.

**Example:**

div{

flex: inherit;

}

**BOOTSTRAP**

**Introduction To Bootstrap:**

Bootstrap mainly includes CSS (Cascading Style Sheets) and an optional JavaScript-supported design template (plug-ins) that deals with typography, buttons, forms, and other user interface components. This Bootstrap framework helps rapid web development and supports developers in creating responsive web pages.

**History Of Bootstrap:**

Twitter Blueprint was the first name for Bootstrap and was developed on Twitter by Mr. Mark Otto and Jacob Thornton. It was released as an open-source product on GitHub in August 2011.

**Benefits Of Bootstrap:**

* It produces fewer cross-browser bugs.
* It is a consistent framework supported by all web browsers and CSS-based compatibility.
* It is a lightweight and hence widely used framework for creating responsive sites.
* It's easily customizable.
* It has a simple and effective grid system.

**Bootstrap CDNAnd Offline Line Link Inserting:**

**Bootstrap CDN:**

* If you don't want to download and host Bootstrap yourself, you can include it from a CDN (Content Delivery Network).
* MaxCDN provides CDN support for Bootstrap's CSS and JavaScript. You must also include jQuery:
* <!--Latestcompiled and minified CSS -->  
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
* <!-- jQuery library -->  
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>
* <!-- Latest compiled JavaScript -->  
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js">

</script>

**Downloading files locally:**

Another way of importing Bootstrap to HTML is to directly download the files locally to your HTML project folder. The files can be downloaded from the following links:

* Bootstrap 4: <https://getbootstrap.com/docs/4.3/getting-started/download/>
* Bootstrap 5: <https://v5.getbootstrap.com/docs/5.0/getting-started/download/>

**For CSS:**

Include a link to the bootstrap.min.css file in the <head> portion of your HTML file. Doing this enables you to use the Bootstrap CSS components as per your need.

**For JS:**

Add a link to the bootstrap.min.js file before the end of the *<body>*portion of your HTML file. Doing this enables you to use Bootstrap JS components.

**Bootstrap-Grid System:**

* In graphic design, a grid is a structure (usually two-dimensional) made up of a series of intersecting straight (vertical, horizontal) lines used to structure the content. It is widely used to design layout and content structure in print design. In web design, it is a very effective method to create a consistent layout rapidly and effectively using HTML and CSS.
* To put in simple words, grids in web design organize and structure content, makes the websites easy to scan and reduces the cognitive load on users.
* 0
* Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout options, as well as powerful mixins for generating more semantic layouts.
* Let us understand the above statement. Bootstrap 3 is mobile first in the sense that the code for Bootstrap now starts by targeting smaller screens like mobile devices, tablets, and then “expands” components and grids for larger screens such as laptops, desktops.
* **Mobile First Strategy Content**
* Determine what is most important.
* **Layout**
* Design to smaller widths first.
* Base CSS address mobile device first; media queries address for tablet, desktops.
* **Progressive Enhancement**
* Add elements as screen size increases.

**Working Of Bootstrap Grid System:**Grid systems are used for creating page layouts through a series of rows and columns that house your content. Here's how the Bootstrap grid system works:

* Rows must be placed within a **.container** class for proper alignment and padding.
* Use rows to create horizontal groups of columns.
* Content should be placed within the columns, and only columns may be the immediate children of rows.
* Predefined grid classes like **.row and .col-xs-4** are available for quickly making grid layouts. LESS mixins can also be used for more semantic layouts.
* Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and the last column via negative margin on **.rows.**
* Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three **.col-xs-4.**

**Grid Options:**

The following table summarizes aspects of how Bootstrap grid system works across multiple devices:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Extra small devices Phones (<768px) | Small devices Tablets (≥768px) | Medium devices Desktops (≥992px) | Large devices Desktops (≥1200px) |
| Grid behavior | Horizontal at all times | Collapsed to start, horizontal above breakpoints | Collapsed to start, horizontal above breakpoints | Collapsed to start, horizontal above breakpoints |
| Max container width | None (auto) | 750px | 970px | 1170px |
| Class prefix | .col-xs- | .col-xs- | .col-md- . | .col-lg- |
| # of columns | 12 | 12 | 12 | 12 |
| Max column width | Auto | Auto | Auto | Auto |
| Gutter width | 30px (15px on each side of a column) | 30px (15px on each side of a column) | 30px (15px on each side of a column) | 30px (15px on each side of  a column) |
| Nestable | Yes | Yes | Yes | Yes |
| Offsets | Yes | Yes | Yes | Yes |
| Column ordering | Yes | Yes | Yes | Yes |

* **Basic Grid Structure:**

Following is basic structure of Bootstrap grid:

|  |
| --- |
| <div class=”container”>  <div class=”row”>  <div class=”col-\*-\*”></div>  <div class=”col-\*-\*”></div>  </div>  <div class=”row”>……</div>  </div>  <div class=”container”>……. |

**Responsive Column Resets:**

* With the four tiers of grids available, you are bound to run into issues where at certain breakpoints, the columns don't clear quite right as one is taller than the other. To fix that, use a combination of a class .**clearfix** and the responsive utility classes as shown in the **Example:**

<div class="container">

<div class="row" >

<div class="col-xs-6 col-sm-3"

style="background-color: #dedef8;

box-shadow: inset 1px -1px 1px #444, inset -1px 1px1px #444;"><p>Lorem

ipsum dolor sit amet, consecteturadipisicingelit.</p>

</div>

<div class="col-xs-6 col-sm-3"

style="background-color: #dedef8;box-shadow: inset 1px -1px 1px #444, inset -1px 1px1px #444;">

<p>Loremipsum dolor sitamet, consecteturadipisicingelit, sed do eiusmod

temporincididuntutlabore et dolore magna aliqua. Utenim ad minim veniam, quisnostrud exercitation ullamcolaboris

nisiutaliquip ex eacommodoconsequat.</p>

<p>Loremipsum dolor sitamet, consecteturadipisicingelit, sed do

eiusmodtemporincididuntut.</p>

</div>

<div class="clearfix visible-xs"></div>

<div class="col-xs-6 col-sm-3" style="background-color: #dedef8; box-shadow:inset 1px -1px 1px #444, inset -1px 1px1px #444;">

<p>Utenim ad minim veniam, quisnostrud exercitation ullamco

laboris nisi utaliquip ex eacommodoconsequat. </p>

</div>

<div class="col-xs-6 col-sm-3"

style="background-color: #dedef8;box-shadow:

inset 1px -1px 1px #444, inset -1px 1px1px #444;">

<p>Loremipsum dolor sitamet, consecteturadipisicingelit, sed do eiusmodtemporincididuntutlabore et doloremagna aliqua. Utenim ad minim </p>

</div>

</div>

</div>

**Offset Columns:**

* Offsets are a useful feature for more specialized layouts. They can be used to push columns over for more spacing (for example). The **.col-xs=\*** classes don’t support offsets, but they are easily replicated by using an empty cell.
* To use offsets on large displays, use **the .col-md-offset-\*** classes. These classes increase the left margin of a column by \* columns where \* range from **1** to **11**.
* In the following example, we have <div class="col-md-6">..</div>. We will center this using class **.col-md-offset-3**.

|  |
| --- |
| <div class="container">  <h1>Hello, world!</h1>  <div class="row" >  <div class="col-xs-6 col-md-offset-3"  style="background-color: #dedef8;box-shadow:  inset 1px -1px 1px #444, inset -1px 1px1px #444;">  <p>Loremipsum dolor sitamet, consecteturadipisicingelit. </p></div>  </div>  </div> |

**Column Ordering:**Another nice feature of Bootstrap grid system is that you can easily write the columns in anorder,and show them in another one. You can easily change the order of built-in gridcolumns with **.col-md-push-\*** and .**col-md-pull-\*** modifier classes where \* range from 1 to11.

In the following example we have two columns layout with left column being the narrowest and acting as a sidebar. We will swap the order of these columns using **.col-md-push-\*** and**.col-md-pull-\*** classes.

|  |
| --- |
| <div class="container">  <h1>Hello, world!</h1>  <div class="row">  <p>Before Ordering</p>  <div class="col-md-4" style="background-color: #dedef8;  box-shadow: inset 1px -1px 1px #444, inset -1px 1px1px #444;"> I am on  left </div>  <div class="col-md-8" style="background-color: #dedef8;  box-shadow: inset 1px -1px 1px #444, inset -1px 1px1px #444;"> I am on  right </div>  </div><br>  <div class="row"><p>After Ordering</p>  <div class="col-md-4 col-md-push-8" style="background-color: #dedef8; box-shadow:  inset 1px -1px 1px #444, inset -1px 1px1px  #444;">I was on left </div>  <div class="col-md-8 col-md-pull-4" style="background-color: #dedef8; box-shadow: inset 1px -1px 1px #444, inset -1px 1px1px #444;">  I was on right </div>  </div>  </div> |

**Bootstrap Images:**

* Images in Bootstrap are made responsive with img-fluid . This applies max-width :100% and height: auto; to the image so that it scales with the parent element.
* <imgsrc=” ”class=” img-fluid” alt=” ” >



**Thumbnails:**

* In addition to our border-radius utilities, you can use .img-thumbnail to give an image a rounded 1px border appearance.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.1/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js">

</script>

</head>

<body>

<div class="container">

<p>The .img-thumbnail class creates a thumbnail of the image:</p>

<imgsrc="cinqueterre.jpg" class="img-thumbnail" alt="Cinque Terre" width="304" height="236">

</div>

</body>

</html>

**Output:**

The .img-thumbnail class creates a thumbnail of the image:



**Bootstrap Image Shapes:**

**Rounded Corners:**

* The .img-rounded class adds rounded corners to an image.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<h6 class=”text-center text-hotpink”>.rounded</h6>

<imgsrc="cinqueterre.jpg" class="img-rounded" alt="Cinque Terre" width="304" height="236">

</div>

</body>

</html>

**Output:**

****

**Circle:**

The .img-circle class shapes the image to a circle .

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<h2>Circle</h2>

<p>The .img-circle class shapes the image to a circle (not available in IE8):</p>

<imgsrc="cinqueterre.jpg" class="img-circle" alt="Cinque Terre" width="304" height="236">

</div>

</body>

</html>

**Output:**

****

**Bootstrap Jumbotron :**

* A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.
* A jumbotron indicates a big box for calling extra attention to some special content or information.
* A jumbotron is displayed as a grey box with rounded corners. It also enlarges the font sizes of the text inside it.
* Class .jumbotron to create a jumbotron.

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<linkrel=”stylesheet”href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<div class="jumbotron">

<h1>Bootstrap Tutorial</h1>

<p>Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile-first projects on the web.</p>

</div>

</div>

</body>

</html>

**Output:**



**Fluid Jumbotron:**

* To make the jumbotron full width, and without rounded corners, add the .jumbotron-fluid modifier class and add a .container or .container-fluid within.

**Example:**

<div class="jumbotron jumbotron-fluid">

<div class="container">

<h1 class="display-4">Fluid jumbotron</h1>

<p class="lead">This is a modified jumbotron that occupies the entire horizontal space of its parent.</p>

</div>

</div>

**Output:**

****

**Bootstrap -Page Header:**

**Creating a Page Header:**

* A page header is like a section divider.
* The .page-header class adds a horizontal line under the heading (+ adds some extra space around the element):

**Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<link rel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

</head>

<body>

<div class="container">

<div class="page-header">

<h1>Example Page Header</h1>

</div>

<p>This is some text.</p>

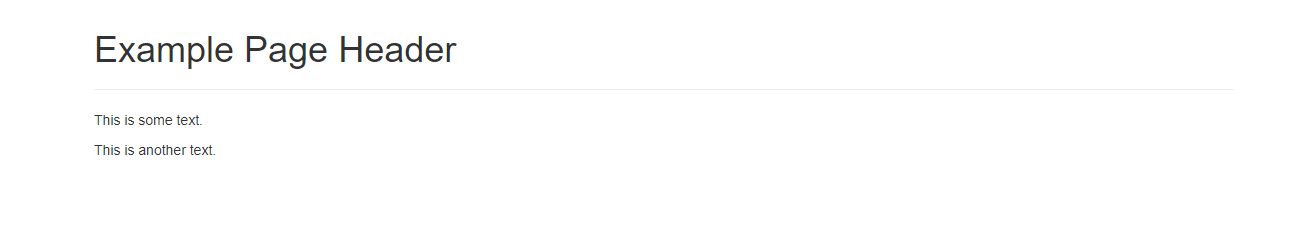
<p>This is another text.</p>

</div>

</body>

</html>

**Output:**



**Bootstrap Tables:**

Due to the widespread use of tables across third-party widgets like calendars and date pickers, we’ve designed our tables to be **opt-in**. Just add the base .table class to any <table>, then extend with custom styles or our various included modifier classes.

Using the most basic table markup, here’s how.table-based tables look in Bootstrap. **All table styles are inherited in Bootstrap**, meaning any nested tables will be styled in the same manner as the parent.

A basic Bootstrap table has a light padding and only horizontal dividers.

The **.table** class adds basic styling to a table:

**Syntax:**<table class=”table”>

**Example:**

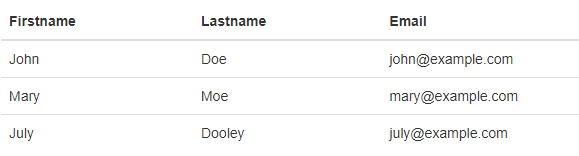
<tr>

<th>Firstname</th>

<th>Lastname</th>

<th>Email</th>

</tr>



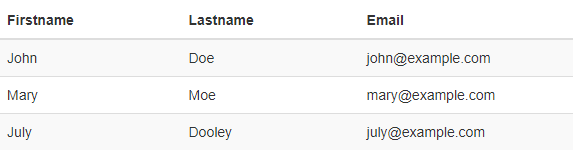
**Striped Table:**

The**.table-striped** class adds zebra-stripes to a table:

**Syntax:**

**<**table class=”table table-striped”>

**Example:**

****

**Bordered Table:**

The **.table-bordered** class adds borders on all sides of the table and cells:

**Syntax:**

**<**table class=”table table-bordered”>

**Example:**

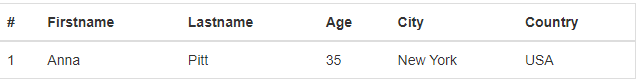


**Responsive Tables:**

The  .table-responsive class creates a responsive table. The table will then scroll horizontally on small devices (under 768px). When viewing on anything larger than 768px wide, there is no difference:

**Syntax:**

**<**table class=”table table-responsive”>



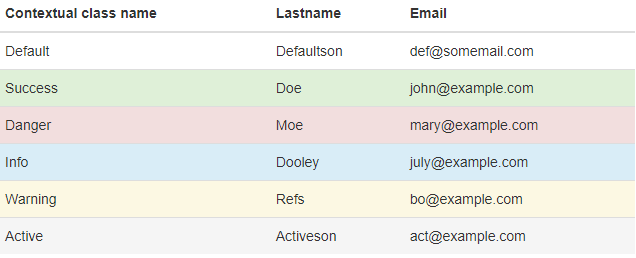
**Contextual Classes:**

Contextual classes can be used to color table rows (<tr>) or table cells (<td>):

The classes that can be used are: .active, .success, .info, .warning, and .danger.

**Syntax:<**tr class=”contextual class name ”>

**Example**:<tr class=”success”>



**Bootstrap forms:**

Bootstrap provides three types of form layouts:

* Vertical form (this is default)
* Horizontal form
* Inline form

Standard rules for all three form layouts:

* Wrap labels and form controls in <div class="form-group"> (needed for optimum spacing)
* Add class .form-control to all textual <input>, <textarea>, and <select> elements.

Vertical form:

* This is the default Bootstrap form layout in which all input fields and labels are vertically aligned. There is no need any extra Bootstrap class applied on form and their child elements.

**Example:**

<form action="/action\_page.php">

<div class="form-group">

<label for="email">Email:</label>

<input type="email" class="form-control" id="email" placeholder="Enter email" name="email"></div>

</form>

**Output:**



**Horizontal form:**

* A horizontal form means that the labels are aligned next to the input field (horizontal) on large and medium screens.

**Example:**

<form class="form-horizontal" action="/action\_page.php">  
  <div class="form-group">  
    <label class="control-label col-sm-2" for="email">Email:</label>  
    <div class="col-sm-10">  
      <input type="email" class="form-control" id="email" placeholder="Enter email">  
    </div>  
  </div>

</form>

**Output:**



**Inline form:**

* In an inline form, all of the elements are inline, left-aligned, and the labels are alongside.
* Additional rule for an inline form:
* Add class .form-inline to the <form> element

**Example:**

<form class="form-inline" action="/action\_page.php">  
  <div class="form-group">  
    <label for="email">Email address:</label>  
    <input type="email" class="form-control" id="email">  
  </div>

</form>

**Output:**

Screenshot (12).png

**Bootstrap inputs:**

Bootstrap supports the following form controls:

* input
* textarea
* checkbox
* radio
* select

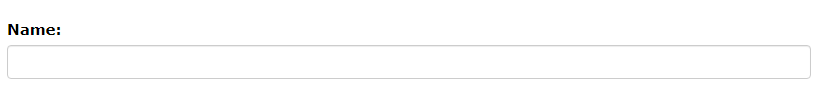
**Bootstrap Input:**

* Bootstrap supports all the HTML5 input types: text, password, datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color.

**Example:**

<div class="form-group">  
  <label for="usr">Name:</label>  
  <input type="text" class="form-control" id="usr">  
</div>

**Output:**



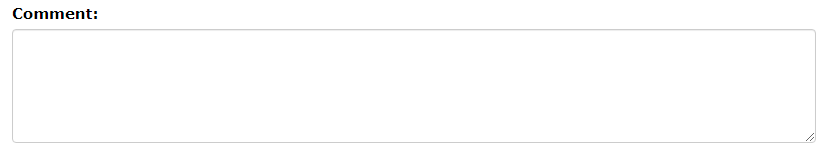
**Bootstrap Textarea:**

* Bootstrap textarea is an input dedicated for a large volume of text.

**Example:**

<div class="form-group">  
  <label for="comment">Comment:</label>  
  <textarea class="form-control" rows="5" id="comment"></textarea>  
</div>

**Output:**



**Default input:**

* input class to set default input field in bootstrap.

**Example:**

<div class="form-group">  
    <label for="inputdefault">Defaultinput</label>  
    <input class="form-control" id="inputdefault" type="text">  
  </div>

**Large input:**

* Input-lg class to set large input field in bootstrap.

**Example:**  
  <div class="form-group">  
    <label for="inputlg">Large input</label>  
    <input class="form-control input-lg" id="inputlg" type="text">  
  </div>

**Output:**



**Column Sizing:**

* col-md-3 which means take 3 columns on the medium-sized screens.
* col-md-4 which means take 4 columns on the medium-sized screens.

**Example:**

<div class="form-group row">  
  <div class="col-xs-2">  
    <label for="ex1">col-xs-2</label>  
    <input class="form-control" id="ex1" type="text">  
  </div>  
  <div class="col-xs-3">  
    <label for="ex2">col-xs-3</label>  
    <input class="form-control" id="ex2" type="text">  
  </div>  
  <div class="col-xs-4">  
    <label for="ex3">col-xs-4</label>  
    <input class="form-control" id="ex3" type="text">  
  </div>  
</div>

**Output:**



**Form Layout:**

Bootstrap provides you with following types of form layouts −

* Vertical (default) form
* In-line form
* Horizontal form

**Vertical Or BasicForm:**

The basic form structure comes with Bootstrap; individual form controls automatically receive some global styling. To create a basic form do the following −

* Add a role *form* to the parent <form> element.
* Wrap labels and controls in a <div> with class *.form-group*. This is needed for optimum spacing.
* Add a class of *.form-control* to all textual <input>,<textarea>, and <select> element

<form role = "form">

<div class = "form-group">

<label for = "name">Name</label>

<input type = "text" class = "form-control" id = "name" placeholder = "EnterName">

</div>

<div class = "form-group">

<label for = "inputfile">File input</label>

<input type = "file" id = "inputfile">

<p class = "help-block">Example block-level help text here.</p>

</div>

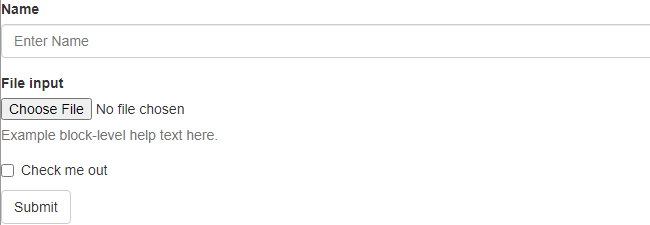
<div class = "checkbox">

<label><input type = "checkbox"> Check me out</label>

</div>

<button type = "submit" class = "btnbtn-default">Submit</button>

</form>



**Inline Form:**

To create a form where all of the elements are inline, left aligned and labels are alongside, add the class *.form-inline* to the <form> tag.

<form class="form-inline" role="form">

<div class="form-group">

<label class="sr-only" for="name">Name</label>

<input type="text"class="form-control" id="name" placeholder="Enter Name">

</div>

<div class="form-group">

<label class="sr-only" for="inputfile">File input</label>

<input type="file" id="inputfile">

</div>

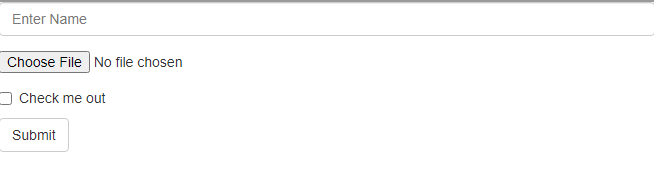
<div class="checkbox">

<label><input type="checkbox"> Check me out</label>

</div>

<button type="submit" class="btn btn-default">Submit</button>

</form>



* By default inputs, selects, and textareas have 100% width in Bootstrap. You need to set a width on the form controls when using inline form.
* Using the class *.sr-only* you can hide the labels of the inline forms.

**Horizontal Form:**

Horizontal forms stands apart from the others not only in the amount of markup, but also in the presentation of the form. To create a form that uses the horizontal layout, do the following −

* Add a class of *.form-horizontal* to the parent <form> element.
* Wrap labels and controls in a <div> with class *.form-group*.
* Add a class of *.control-label* to the labels.

<form class="form-horizontal" role="form">

<div class="form-group">

<label for="firstname" class="col-sm-2 control-label">First Name</label>

<div class="col-sm-10">

<input type="text" class="form-control" id="firstname" placeholder="Enter First Name">

</div>

</div>

<div class="form-group">

<label for="lastname" class="col-sm-2 control-label">Last Name</label>

<div class="col-sm-10">

<input type="text" class="form-control" id="lastname" placeholder="Enter Last Name">

</div>

</div>

<div class="form-group">

<div class="col-sm-offset-2 col-sm-10">

<div class="checkbox">

<label><inputtype="checkbox"> Remember me</label>

</div>

</div>

</div>

<div class="form-group">

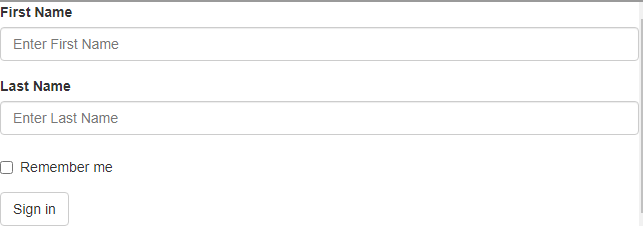
<div class="col-sm-offset-2 col-sm-10">

<button type="submit"class="btn btn-default">Sign in</button>

</div>

</div>

</form>



**Supported Form Controls:**

Bootstrap natively supports the most common form controls mainly input, textarea, checkbox,radio, and select.

**Inputs:**

The most common form text field is the input field. This is where users will enter most of the essential form data. Bootstrap offers support for all native HTML5 input types: *text, password,* datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color. Proper type declaration is required to make Inputs fully styled.

**Example:**

<form role="form">

<div class="form-group">

<label for="name">Label</label>

<input type="text" class="form-control" placeholder="Text input">

</div>

</form>

**Textarea:**

The textarea is used when you need multiple lines of input. Change *rows* attribute as necessary (fewer rows = smaller box, more rows = bigger box).

<form role="form">

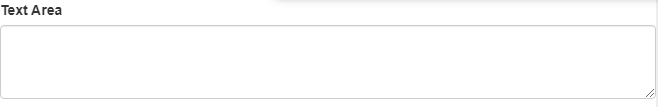
<div class="form-group">

<label for="name">Text Area</label>

<textarea class="form-control" rows="3"></textarea>

</div>

</form>



**Checkboxes And Radio Buttons:**

Checkboxes and radio buttons are great when you want users to choose from a list of preset options.

* When building a form, use *checkbox* if you want the user to select any number of options from a list. Use *radio* if you want to limit the user to just one selection.
* Use *.checkbox-inline* or *.radio-inline* class to a series of checkboxes or radios for controls appear on the same line.

The following example demonstrates both (default and inline) types -

<label for="name">Example of Default Checkbox and radio button </label>

<div class="checkbox">

<label>

<input type="checkbox" value="">Option 1

</label>

</div>

<div class="checkbox">

<label>

<input type="checkbox" value="">Option 2

</label>

</div>

<div class="radio">

<label>

<input type="radio" name="optionsRadios" id="optionsRadios1" value="option1" checked> Option 1

</label>

</div>

<div class="radio">

<label>

<input type="radio" name="optionsRadios" id="optionsRadios2" value="option2"> Option 2 - selecting it will deselect option 1

</label>

</div>

<label for="name">Example of Inline Checkbox and radio button </label>

<div>

<label class="checkbox-inline">

<input type="checkbox" id="inlineCheckbox1" value="option1"> Option 1

</label>

<label class="checkbox-inline">

<input type="checkbox" id="inlineCheckbox2" value="option2"> Option 2

</label>

<label class="checkbox-inline">

<input type="checkbox" id="inlineCheckbox3" value="option3"> Option 3

</label>

<label class="checkbox-inline">

<input type="radio" name="optionsRadiosinline" id="optionsRadios3" value="option1" checked> Option 1

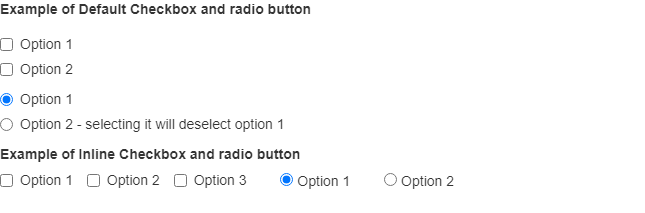
</label>

<label class="checkbox-inline">

<input type="radio" name="optionsRadiosinline" id="optionsRadios4" value="option2"> Option 2

</label>

</div>



**Selects:**

A select is used when you want to allow the user to pick from multiple options, but by default it only allows one.

* Use <select> for list options with which the user is familiar, such as states or numbers.
* Use *multiple = "multiple"* to allow the users to select more than one option.

The following example demonstrates both (select and multiple) types

**Static Control:**

Use the class *.form-control-static* on a <p>, when you need to place plain text next to a form label within a horizontal form.

<form class = "form-horizontal" role = "form">

<div class = "form-group">

<label class = "col-sm-2 control-label">Email</label>

<div class = "col-sm-10">

<p class = "form-control-static">email@example.com</p>

</div>

</div>

<div class = "form-group">

<label for = "inputPassword" class = "col-sm-2 control-label">Password</label>

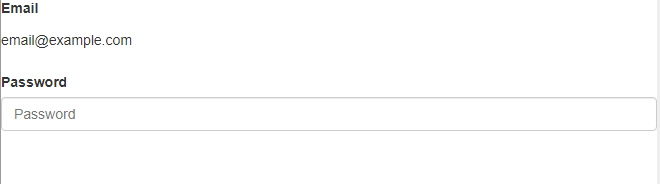
<div class = "col-sm-10">

<input type = "password" class = "form-control" id = "inputPassword" placeholder = "Password">

</div>

</div>

</form>



**Form Control States:**

In addition to the *:focus* (i.e., a user clicks into the input or tabs onto it) state, Bootstrap offers styling for disabled inputs and classes for form validation.

Input Focus

When an input receives *:focus*, the outline of the input is removed and a *box-shadow* is applied.

**Disabled lnputs:**

If you need to disable an input, simply adding the *disabled* attribute will not only disable it; it will also change the styling and the mouse cursor when the cursor hovers over the element.

**Disabled Fieldsets:**

Add the disabled attribute to a <fieldset> to disable all the controls within the <fieldset> at once.

Validation States

Bootstrap includes validation styles for errors, warnings, and success messages. To use, simply add the appropriate class (*.has-warning, .has-error, or .has-success*) to the parent element.

The following example demonstrates all the form control states

**Form Control Sizing:**

You can set heights and widths of forms using classes like *.input-lg* and *.col-lg-\** respectively. The following example demonstrates this

<form role = "form">

<div class = "form-group">

<input class = "form-control input-lg" type = "text" placeholder =".input-lg">

</div>

<div class = "form-group">

<input class = "form-control" type = "text" placeholder = "Default input">

</div>

<div class = "form-group">

<input class = "form-control input-sm" type = "text" placeholder = ".input">

</div>

<div class = "form-group"></div>

<div class = "form-group">

<select class = "form-control input-lg">

<option value = "">.input-lg</option>

</select>

</div>

<div class = "form-group">

<select class = "form-control">

<option value = "">Default select</option>

</select>

</div>

<div class = "form-group">

<select class = "form-control input-sm">

<option value = "">.input-sm</option>

</select>

</div>

<div class = "row">

<div class = "col-lg-2">

<input type = "text" class = "form-control" placeholder = ".col-lg-2">

</div>

<div class = "col-lg-3">

<input type = "text" class = "form-control" placeholder = ".col-lg-3">

</div>

<div class = "col-lg-4">

<input type = "text" class = "form-control" placeholder = ".col-lg-4">

</div>

</div>

</form>



**Bootstrap Modal:**

**The Modal Plugin:**

Use Bootstrap’s JavaScript modal plugin to add dialogs to your site for lightboxes, user notifications, or completely custom content.

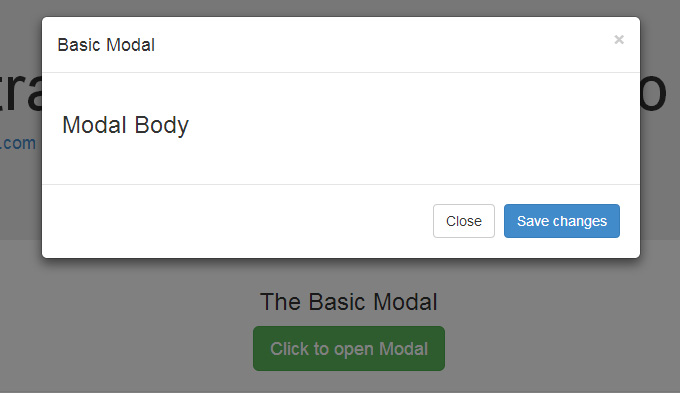
**Usage:**

The modal plugin toggles your hidden content on demand, via data attributes or JavaScript. It also adds .modal-open to the <body> to override default scrolling behavior and generates a .modal-backdrop to provide a click area for dismissing shown modals when clicking outside the modal.

**How it works:**

Before getting started with Bootstrap’s modal component, be sure to read the following as our menu options have recently changed.

* Modals are built with HTML, CSS, and JavaScript. They’re positioned over everything else in the document and remove scroll from the <body> so that modal content scrolls instead.
* Clicking on the modal “backdrop” will automatically close the modal.
* Bootstrap only supports one modal window at a time. Nested modals aren’t supported as we believe them to be poor user experiences.
* Modals use position: fixed, which can sometimes be a bit particular about its rendering. Whenever possible, place your modal HTML in a top-level position to avoid potential interference from other elements. You’ll likely run into issues when nesting a .modal within another fixed element.
* Once again, due to position: fixed, there are some caveats with using modals on mobile devices.



**Modal components:**

Below is a static modal example (meaning its position and display have been overridden). Included are the modal header, modal body (required for padding), and modal footer (optional). We ask that you include modal headers with dismiss actions whenever possible, or provide another explicit dismiss action.

* Scrolling long content
  + When modals become too long for the user’s viewport or device, they scroll independent of the page itself.
* Vertically centered
  + Add .modal-dialog-centered to .modal-dialog to vertically center the modal.
* Tooltips and popovers
  + [Tooltips](https://getbootstrap.com/docs/4.0/components/tooltips/) and [popovers](https://getbootstrap.com/docs/4.0/components/popovers/) can be placed within modals as needed. When modals are closed, any tooltips and popovers within are also automatically dismissed.
* Using the grid
  + Utilize the Bootstrap grid system within a modal by nesting .container-fluid within the .modal-body. Then, use the normal grid system classes as you would anywhere else.
* **Via data attributes:**
* **Toggle:**
  + Activate a modal without writing JavaScript. Set data-coreui-toggle="modal" on a controller element, like a button, along with a data-coreui-target="#foo" or href="#foo" to target a specific modal to toggle.
  + <buttontype="button"data-coreui-toggle="modal"data-coreui target="#myModal">Launch modal </button>
* **Dismiss:**
  + Dismissal can be achieved with the data attribute on a button **within the modal** as demonstrated below:
  + <button type="button" class="btn-close" data-coreui-dismiss="modal" aria-label="Close"></button>
* **Via JavaScript:**

Create a modal with a single line of JavaScript:

const myModal=newcoreui.Modal(document.getElementById('myModal'),options)

(or)

const myModalAlternative=newcoreui.Modal('#myModal',options)

**Positions:**

To change the position of the modal add one of the following classes to the .modal-dialog

**Top right:**.modal-side + .modal-top-right

**Top left:**.modal-side + .modal-top-left

**Bottom right:**.modal-side + .modal-bottom-right

**Bottom left:**.modal-side + .modal-bottom-right

**Bootstrap Alerts:**

Bootstrap Alerts are used to provide an easy way to create predefined alert messages. Alert adds a style to your messages to make it more appealing to the users.

**Dissimisal Alerts:**

When an alert message is dismissed, its message is removed from the Alerts Inbox page and from the Alerts table on the Dashboard page. Some alerts are dismissed automatically when the alert condition is resolved; others must be dismissed manually.

Alerts on events are triggered only one time at the time the event is detected. Therefore, if an alert on an event is dismissed manually, it is not redisplayed.

To close the alert message, add a .alert-dismissible class to the alert container. Then add class="close" and data-dismiss="alert" to a link or a button element (when you click on this the alert box will disappear).

**Example:**

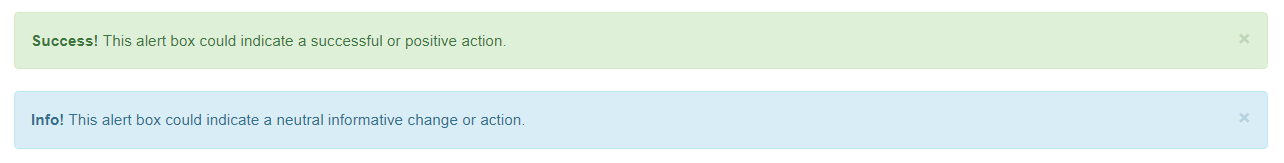
<body>

<div class="alert alert-success alert-dismissible">  
  <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>  
  <b >Success!</b > Indicates a successful or positive action.  
</div>

<div class="alert alert-info alert-dismissible">  
  <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>  
  <b >info!</b > Indicates a successful or positive action.  
</div>

< /body >

**Output:**

****

**The Aria-\* Attribute And&Times:**

* To help improve accessibility for people using screen readers, you should include the aria-label="close" attribute, when creating a close button.

**Links In Alerts:**

* Bootstrap Alerts are used to provide an easy way to create predefined alert messages. Alert adds a style to your messages to make it more appealing to the users.
* Alerts are created with the .alert class, followed by one of the four contextual classes
* .alert-success - Indicates a successful or positive action.
* .alert-info - Indicates a neutral informative change or action.
* .alert-warning - Indicates a warning that might need attention.
* .alert-danger - Indicates a dangerous or potentially negative action.

**Example:**

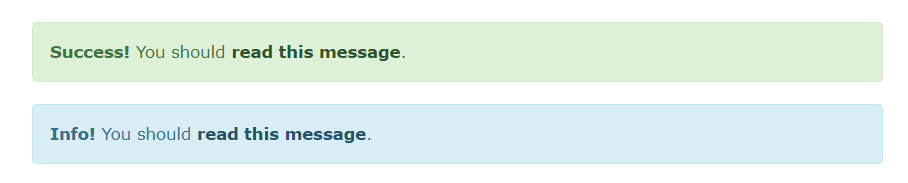
<body><div class="alert alert-success">  
 <strong>Success!</strong> You should

<a href="#" class="alert-link">read this message</a>.  
</div>

<div class="alert alert-iNFO">  
 <strong>Info!</strong> You should

 <a href="#" class="alert-link">read this message</a>.  
</div>

</body>

**Output:**

**JAVASCRIPT**

**Introduction to Javascript:**

* JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive
* There are also more advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website than downloading files (such as realtime collaboration between multiple computers).
* JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.

**Example:** Complex animations, clickable buttons, popup menus, etc.

**Javascript Functions:**

* A JavaScript function is a block of code designed to perform a particular task.
* A JavaScript function is executed when "something" invokes it (calls it).

**Example:-**

<!DOCTYPE html>

<html>

<body>

<p id="demo"></p>

<script>

Function myFunction(p1, p2) {

return p1 \* p2;

}

document.getElementById("demo").innerHTML = myFunction(4, 3);

</script>

</body>

</html>

**Output:**

12

**JavaScript Function Syntax:**

* A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **( )**.
* Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
* The parentheses may include parameter names separated by commas:  
  **(*parameter1, parameter2, ...*)**
* The code to be executed, by the function, is placed inside curly brackets: **{}**

**Syntax:**

function *name*(*parameter1, parameter2, parameter3*) { *// code to be executed*  
 }

* Function **parameters** are listed inside the parentheses ( ) in the function definition.
* Function **arguments** are the **values** received by the function when it is invoked.
* Inside the function, the arguments (the parameters) behave as local variables.

**Function Invocation:**

The code inside the function will execute when "something" invokes (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked).

**Function Return:**

* When JavaScript reaches a return statement, the function will stop executing.
* If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
* Functions often compute a **return value**. The return value is "returned" back to the "caller":

**Example:**

let x = myFunction(4, 3);   // Function is called, return value will end up in x  
function myFunction(a, b) {  
  return a \* b;             // Function returns the product of a and b  
}

**Output:**

12

**Uses of Functions:**

* Code is reusable : Define the code once, and use it many times.
* You can use the same code many times with different arguments, to produce different results.

**JavaScript Array:**

* An Array can hold many values under a single name, and you can access the values by referring to an index number.
* **JavaScript array** is an object that represents a collection of similar type of elements.

**Creating an Array:**

Usingan array literal is the easiest way to create a JavaScript Array.

**Syntax:** const *array\_name* = [*item1*, *item2*, ...];

There are 3 ways to construct array in JavaScript

* By array literal
* By creating instance of Array directly (using new keyword)
* By using an Array constructor (using new keyword)

**JavaScript array literal:**

**Syntax:-** var arrayname=[value1,value2.....valueN];

**JavaScript Array directly (new keyword):**

**Syntax:-**var arrayname=new Array();

Here, newkeyword is used to create instance of array.

**JavaScriptarray constructor (new keyword):**

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

**Accessing Array elements:**

Arrays are indexed from 0. The array name followed by the subscript is used for referring an array element.

**Syntax:**array\_name[subscript];

JavaScript supports the following categories of arrays.

* Multidimensional array
* Passing arrays to functions
* Return array from functions

**Multidimensional Arrays:**

* A multidimensional array can be defined as an array reference to another array for its value.
* Multidimensional arrays are not directly provided in JavaScript. If you need to create a multidimensional array, you have to do it by using the one-dimensional array.

**Syntax:** var array\_name = [[value1,value2,value3],[val1,val2,val3]];

**Passing Array to function:**

Passing array as an argument to a function, you have to specify the array name (a reference to an array) without brackets.

**Syntax :**

var array\_name = **new** Array["element1”,”element2”,”element3”…];

function function\_name(array\_name)

{

//code;

}

**Return Array from function:**

It allows a function to return an array.

**Syntax :**

function function\_array()

{

    return new Array("element1”,”element2”,”element3”…);

 }

**JavaScript Objects:**

* A javascript object is an entity having state and behavior (properties and method).
* JavaScript is an object-based language. Everything is an object in JavaScript.

**Creating Objects in JavaScript:**

There are 3 ways to create objects.

* By object literal
* By creating instance of Object directly (using new keyword)
* By using an object constructor (using new keyword)

**JavaScript Object by object literal:**

The syntax of creating object using object literal is given below:

**Syntax:**object={property1:value1,property2:value2.....propertyN:valueN}

**By creating instance of Object:**

The syntax of creating object directly is given below:

**Syntax:**var objectname=new Object();

Here, **new keyword** is used to create object.

**By using an Object constructor:**

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

**Accessing properties:**

To access a property of an object, you use one of two notations: the dot notation and array-like notation.

* The dot notation (.)

**Syntax:** objectName.propertyName

Array-like notation ([])

**Syntax:**objectName['propertyName']

To change the value of a property, you use the assignment operator (=)

person.firstName = 'Jane';

**Nested Objects:**

* The nested objects are utilized to store the object properties with another object .
* The dot and square bracket notation are employed to access the properties of objectsin javacsript.

**Syntax:**

object={{property1:value1,property2:value2.....propertyN:valueN},{ property1:value1,property2:value2.....propertyN:valueN },…,{} }

**Deleting an object:**

­­­­To delete a property of an object , we can use the delete operator

**Syntax:**

delete object\_name.property\_name;

**Javascript Array Of Objects:**

JavaScript variables can be objects. Arrays are special kinds of objects. Because of this, you can have variables of different types in the same Array.

**Array of objects:**Itstores multiple values in a **single**variable. The object can contain anything in the real world such as person names, cars, and game characters. Objects are very easy to use in some situations if you know where the data is being processed. The character set of objects is known as **Properties**. The properties of an object can be called by using**DOT notation**and **[ ] notation**.

**What is an Array of Objects in JavaScript?**

An array in java script is the collection of multiple types of data at a single place in order, and an array of objects in java script is a collection of homogenous data, that stores a sequence of numbered objects at a single place.

**Example:**

let myArr =

[

{

StudentName: "jack";

StudentClass: 6,

StudentSection: "a"

},

{

StudentName: "tom",

StudentClass: 6,

StudentSection: "a"

}

];

console.log (myArr [1]);

**Output:**

{

StudentName: "tom",

StudentClass: 6,

StudentSection: "a"

}

Above, there is an array of objects myArr and we are accessing an object from the array by using array index myArr[1], also printing the value on the console. As result, we are printing an object from an array of objects. This prints all the key-value pairs present at that particular index.

**Creating an Array of Objects:**

To create an array of objects in java script, we have to declare an empty array first and then Initialize it with the objects separated by a comma.

let mySchoolArr = [

{

School Name: “childrens school",

School Address: "random address"

},

{

NumberOfStudens: 500,

numberOfTeachers: 10,

}

];

**Array Properties:**

There are three types of properties in array Object:

**1.Constructor:-**The constructor properties return a function that creates the Array prototype. This return function is a native code that is provided by the java script engine.

**Example:-**

let myArr = [

{

studentName: "jack",

studentClass: 6,

studentSection: "a"

},

{

studentName: "tom",

studentClass: 6,

studentSection: "a"

}

];

console.log(myArr.constructor);

**Output:**

function Array()

{

[native code]

}

Above, we have an array of objects myArr and we are printing its constructor properties on the console. As result, we print the function Array() { [native code] } on the console.Here, we get the native code as a reference which is a predefined function inside the javascript engine responsible for creating the Array prototype.

**2. Length:-**It returns the length of elements in an array i.e. the number of objects present in the array.

**Example:-**

let myArr = [

{

StudentName: "jack",

studentClass: 6,

studentSection: "a"

},

{

StudentName: "tom",

StudentClass: 6,

StudentSection: "a"

}

];

console.log(myArr.length);

**Output**:

2.

Above, we use the length properties on the myArr array of objects and also print it on the console. As result, we print 2 on the console.

**3. Prototype:** Array Prototype property helps to add new methods into the Array, for example- adding all values into an array. It also helps to add properties to the Array Object.

**Example:**

let myArr = [1,2,3,4];

Array.prototype.add = function()

{

let result = 0;

for(let i = 0; i < this.length; i++)

{

result += this[i];

}

return result;

}

console.log(myArr.add());

**Output:**

10

Above, we are adding a user-defined method into the Array Object. To do that, we accessed the Array prototype property using dot notation and added the user-defined function add into Array Object like this Array.prototype.add. Then, assigned an anonymous function to add, which is the definition of add method.

**Javascript Operators:**

**What is an Operator ?**

Let us take a simple expression **4 + 5 is equal to 9.** Here 4 and 5 are called **operands** and ‘+’ is called the **operator**. JavaScript supports the following types of operators.

* Arithmetic Operators
* Comparison Operators
* Logical (or Relational) Operators
* Assignment Operators
* Conditional (or ternary) Operators

**Arithmetic Operators**

JavaScript supports the following arithmetic operators:

Assume variable A holds 10 and variable B holds 20, then:

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

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

**Example:**

<html>

<body>

<script type="text/javascript">

var a = 33,b = 10;

var c = "Test";

varlinebreak = "<br />";

document.write("a + b =” a + b ,linebreak);

document.write("a - b = “a - b ,linebreak);

document.write("a / b = " a / b,linebreak);

document.write("a % b = " a % b,linebreak);

document.write("a + b + c = " a + b + c,linebreak);

document.write("a++ = " a++,linebreak);

document.write("b-- = " b--,linebreak);

</script>

</body>

</html>

**Output:**

a + b = 43

a - b = 23

a / b = 3.3

a % b = 3

a + b + c = 43Test

a++ = 33

b-- = 10

**Comparison Operators**

JavaScript supports the following comparison operators:

Assume variable A holds 10 and variable B holds 20, then:

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

**Example:-**

<html>

<body>

<script type="text/javascript">

var a = 10,b = 20;

varlinebreak = "<br />";

document.write("(a == b) => ",(a == b),linebreak);

document.write("(a < b) => ",(a < b),linebreak);

document.write("(a > b) => ",(a > b),linebreak);

document.write("(a != b) => ",(a != b),linebreak);

document.write("(a >= b) => ",(a >= b),linebreak);

document.write("(a <= b) => ",(a <= b),linebreak);

</script>

</body>

</html>

**Output:-**

(a == b) => false

(a < b) => true

(a > b) => false

(a != b) => true

(a >= b) => false

(a <= b) => true

**Logical Operators:**

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

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

**Example:**

<html>

<body>

<script type="text/javascript">

var a = true;

var b = false;

varlinebreak = "<br />";

document.write("(a && b) => "(a && b),linebreak);

document.write("(a || b) => "(a || b),linebreak);

document.write("!(a && b) => “,(!(a && b)),linebreak);

</script>

</body>

</html>

**Output:**

(a && b) => false

(a || b) => true

!(a && b) => true

**Bitwise Operators:**

JavaScript supports the following bitwise operators:

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

|  |  |
| --- | --- |
| **S.No** | **Operator and Description** |
| 1. | **& (Bitwise AND)**  It performs a Boolean AND operation on each bit of its integer arguments.  **Ex**: (A & B) is 2. |
| 2. | **| (BitWise OR)**  It performs a Boolean OR operation on each bit of its integer arguments.  **Ex:** (A | B) is 3. |
| 3. | **^ (Bitwise XOR)**  It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.  **Ex:** (A ^ B) is 1. |
| 4. | **~ (Bitwise Not)**  It is a unary operator and operates by reversing all the bits in the operand.  **Ex:** (~B) is -4. |
| 5. | **<< (Left Shift)**  It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.  **Ex:** (A << 1) is 4 |
| 6. | **>> (Right Shift)**  Binary Right Shift Operator. The left operand’s value is moved right by the number of bits specified by the right operand.  **Ex:** (A >> 1) is 1 |
| 7. | **>>> (Right shift with Zero)**  This operator is just like the >> operator, except that the bits shifted in on the left are always zero.  **Ex:** (A >>> 1) is 1. |

**Example:**

<html>

<body>

<script type="text/javascript">

var a = 2; // Bit presentation 10

var b = 3; // Bit presentation 11

varlinebreak = "<br />";

document.write("(a & b) => ", (a & b),linebreak);

document.write("(a | b) => "(a | b),linebreak);

document.write("(a ^ b) => "(a ^ b),linebreak);

document.write("(~b) => ",~b,linebreak);

document.write("(a << b) => ",(a << b),linebreak);

document.write("(a >> b) => "(a ^ b),linebreak);

</script>

</body>

</html>

**Output:**

(a & b) => 2

(a | b) => 3

(a ^ b) => 1

(~b) => -4

(a << b) => 16

(a >> b) => 0

**Assignment Operators:**

JavaScript supports the following assignment operators:

|  |  |
| --- | --- |
| **S.No** | **Operator and Description** |
| 1. | **= (Simple Assignment )**  Assigns values from the right side operand to the left side operand  **Ex:** C = A + B will assign the value of A + B into C |
| 2. | **+= (Add and Assignment)**  It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A |
| 3. | **-= (Subtract and Assignment)**  It subtracts the right operand from the left operand and assigns the result to the left operand.  **Ex:** C -= A is equivalent to C = C - A |
| 4. | **\*= (Multiply and Assignment)**  It multiplies the right operand with the left operand and assigns the result to the left operand.  **Ex:** C \*= A is equivalent to C = C \* A |
| 5. | **/= (Divide and Assignment)**  It divides the left operand with the right operand and assigns the result to the left operand.  **Ex:** C /= A is equivalent to C = C / A |
| 6. | **%= (Modules and Assignment)**  It takes modulus using two operands and assigns the result to the left operand.  **Ex:** C %= A is equivalent to C = C % A |

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

**Example :**

<html>

<body>

<script type="text/javascript">

var a = 33,b = 10,linebreak = "<br />";

document.write("Value of a => (a = b) => ",(a = b),linebreak);

document.write("Value of a => (a += b) => "(a += b),linebreak);

document.write("Value of a => (a -= b) => "(a += b),linebreak);

document.write("Value of a => (a \*= b) => "(a \*= b),linebreak);

document.write("Value of a => (a /= b) => "(a /= b),linebreak);

document.write("Value of a => (a %= b) => "(a %= b),linebreak);

</script>

</body>

</html>

**Output:-**

Value of a => (a = b) => 10

Value of a => (a += b) => 20

Value of a => (a -= b) => 10

Value of a => (a \*= b) => 100

Value of a => (a /= b) => 10

Value of a => (a %= b) => 0

**Miscellaneous Operators:**

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

**Conditional Operator (? :)**

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

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

**Example :**

<html>

<body>

<script type="text/javascript">

var a = 10,b = 20,linebreak = "<br />";

document.write ("((a > b) ?100 : 200) => ",(a > b) ? 100 : 200,linebreak);

document.write ("((a < b) ?100 : 200) => ",(a < b) ? 100 : 200,linebreak);

</script>

</body>

</html>

**Output:-**

((a > b) ? 100 : 200) => 200

((a < b) ? 100 : 200) => 100

**typeof Operator:**

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand. The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation. Here is a list of the return values for the typeof Operator.

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| Number | "number" |
| String | "string" |
| Boolean | "boolean" |
| Object | "object" |
| Function | "function" |
| Undefined | "undefined" |
| Null | "object" |

**Example:-**

<html>

<body>

<script type="text/javascript">

var a = 10;

var b = "String";

varlinebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");

document.write("Result => ", result,linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");

document.write("Result => " ,result,linebreak);

</script>

</body>

</html>

**Output:**

Result => B is String

Result => A is Numeric

**Javascript Looping Statements:**

The Javascript loops are used to iterate the piece of code using for,while,do whle or for –in loop.It makes the code compact .It is mostly used in array.

There are three types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop

**JavaScript for loop:**

The JavaScript for loop iterates the elements for the fixed number of times.It should be used if number of iteration is known.

**The syntax of for loop is :**

for (initialization; condition; increment)

{

 //   code to be executed

}

**JavaScript while loop:**

The JavaScript while loop iterates the elements for the infinite number of times.It should be used if number of iteration is not known.

**The syntax of while loop is :**

while (condition)

{

//code to be executed

}

**JavaScript do while loop:**

The JavaScript do while loop iterates the elements for the infinite number of times like while loop .But, code is executed at least once whether condition is true or false .

**The syntax of do while loop is :**

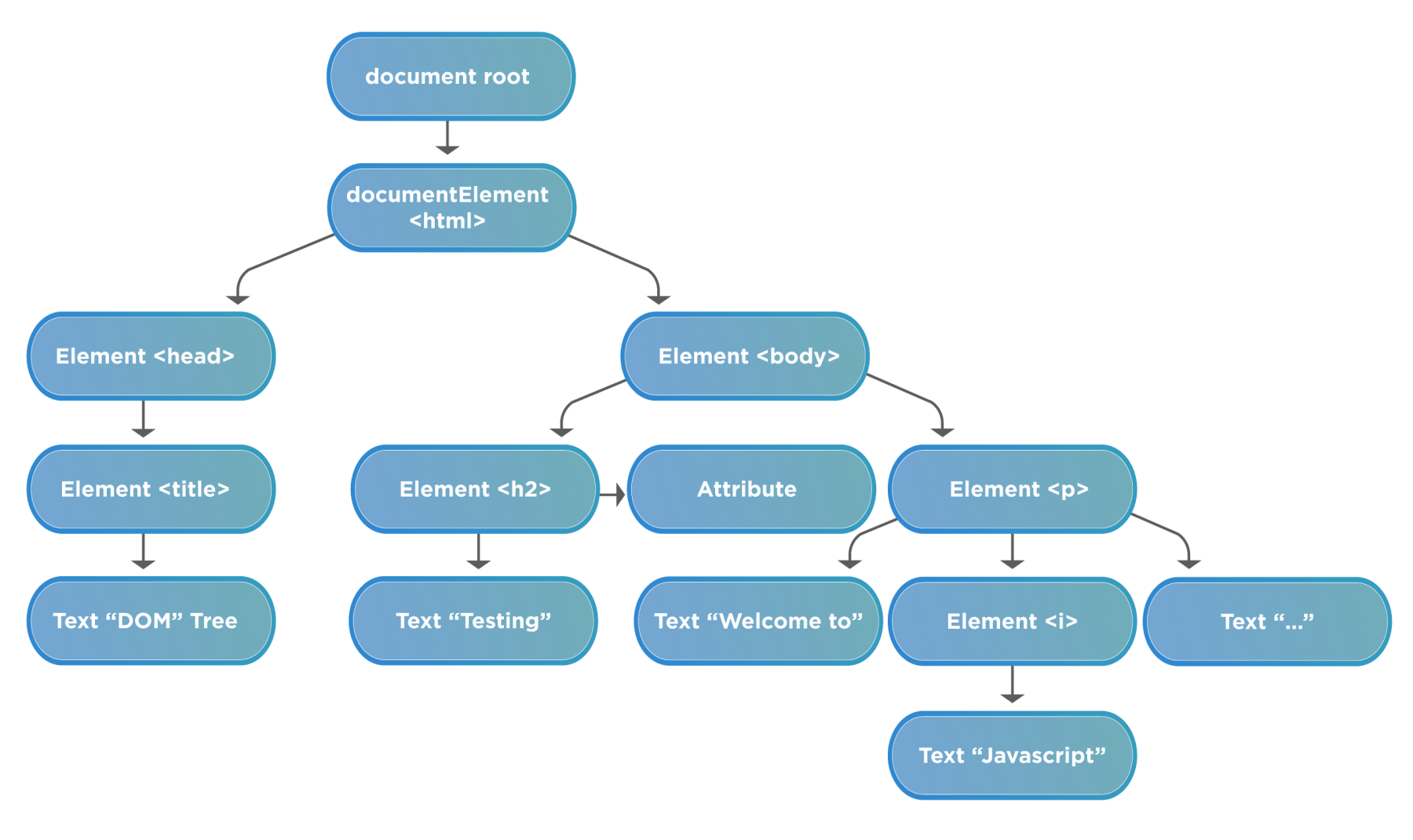
do{

 //  code to be executed

}while (condition);

**Javascript DOM Elements:**

[**DOM**](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) is a data representation of the objects in the HTML and XML pages. The document loaded in your browser is represented by a **document object model.** Moreover, it is a "**tree structure**" representation created by the browser that enables the HTML structure to be easily accessed by programming languages. Additionally, the DOM represents the document as nodes and objects. In this way, programming languages can connect to the page. Furthermore, a simple structure of a web page DOM will look like below:



**How to access DOM elements using JavaScript?**

A webpage in JavaScript is a document, and JavaScript provides an object "document", which designates the complete webpage. Moreover, the document object provides various properties and methods to access and manipulate the web elements loaded on the page. To identify and access the DOM elements, JavaScript uses three ways:

* Accessing elements By ID.
* Accessing elements By TagName.
* Accessing elements By ClassName.

**Accessing a DOM element By ID:**

JavaScript can find HTML elements in the DOM based on the "**id**" of the element. The document object provides a method "**getElementById()**" to accomplish this task. Moreover, its syntax looks like below:

**Syntax:**

document.getElementById(“IDName”);

**Example:**

<html>

<body>

<h6>Demonstrating getElementById in javascript:</h6><br>

<b id="bold">Industrial Training</b>

<script type="text/javascript">

document.getElementById("bold").innerHTML = " WEB TECHNOLOGY ";

</script>

</body>

</html>

In the above example, we can see that element by id "bold" has been found. In addition to that, we also changed its attribute innerHTML to WEB TECHNOLOGY.

**Accessing a DOM element By TagName:**

JavaScript can find the elements in the HTML based on the "TagName" and return an array of matching nodes. The inbuilt function, of document object, available for this is getElementByTagName(). Additionally, its syntax looks like below:

**Syntax:-** document.getElementByTagName(“tagName”);

**Example:**

<html>

<body>

<h6>Demonstrating getElementByTag in javascript:</h6><br>

<b>Industrial Traning</b>

<script type="text/javascript">

document.getElementsByTagName("b")[0].innerHTML ="WEB TECHNOLOGY ";

</script>

</body>

</html>

In the above example, we can see that the element by HTML tag"<b>" has been found. Additionally, we changed its attribute innerHTML to WEB TECHNOLOGY.

**Accessing a DOM element By ClassName:**

JavaScript can find the element in the HTML based on the className attribute of the element and returns an array of matching nodes. The inbuilt function available in this operation is getElementByClassName(). Additionally, its syntax looks like below:

**Syntax:**

document.getElementByClassName(“ClassName”);

**Example:**

<html>

<body>

<h6>Demonstrating getElementsByClassName in javascript:</h6></br>

<b class="bold">Industrial Traning </b>

<script type="text/javascript">

document.getElementsByClassName("bold")[0].innerHTML = "WEB TECNOLOGY";

</script>

</body>

</html>

In the above example, we can see that element by className "bold" has been found. Additionally, we changed its attribute innerHTML to WEB TECHNOLOGY.

**How to manipulate DOM elements by using JavaScript?**

Apart from accessing the elements, JavaScript provides some methods and properties which can manipulate or change the values of the DOM elements. Few of those methods and properties are:

* + write
  + innerHTML
  + attributeName
  + Style.property
  + setAttribute
  + createElement
  + appendChild
  + removeChild
  + replaceChild
* **write:**

This method writes new elements or text to the HTML page. Additionally, its syntax looks like below:

**Syntax:**

document.write(“data”);

* **innerHTML:**

It is a property that we use to get or set the HTML or XML markup contained within the element. Also, its syntax looks like below:

**Syntax:**

node.innerhtml = “changingText”;

* **attributeName:**

We use his property is used to get and update the value of an attribute of an HTML element. Additionally, its syntax looks like below:

**Syntax:**

node.atrributeName = value;

* **Style.property**

We use this property to set or edit the existing style properties of an HTML tag. Also, its syntax looks like below:

**Syntax:**

node.Style.attribute = value;

* **setAttribute**

We use this function to create or update an attribute for the existing HTML element. Additionally, its syntax looks like below:

**Syntax:**

node.setAttribute(attributeName, attributeValue);

* **createElement and appendChild**

This createElement() method is used to create a new element in the HTML DOM. Once the creation of element happens, it can append to a parent element using the appendChild() method. Moreover, its syntax looks like below:

**Syntax:**

var node = document.createElement(tagName);

document.parentTag.appendChild(node);

* **removeChild**

This function removes an HTML element from the document. Also, its syntax looks like below:

**Syntax:**

node.removechild(childNode);

* **replaceChild**

The replaceChild() method replaces a child node with a new node. The new node could be an existing node in the document, or you can create a new node. Additionally, its syntax looks like below:

**Syntax:**

node.replaceChild(newnode, oldnode);

**JavaScript Events:**

Events are a part of the Document Object Model(DOM) and every HTML element contains a set of events which can trigger JavaScript Code. There are some JavaScript Events those are as follows:

**onclick Event:**

The ***onclick event*** occurs when the user clicks on an HTML element. It allows the programmer to execute a JavaScript's function when an element gets clicked.

**Example:**

<html>

<body>

<button onclick="myFunction()">Click me</button>

<p id="demo"></p>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Hello World";

}

</script>

</body>

</html>

**Output:**

Before Clicking:

click.png

After Clicking:



**onsubmit Event:**

The **onsubmit event** is an event that occurs when you try to submit a form. When a form is submitted the **onsubmit Event** is triggered.

**Example:**

<html>

<body>

<form onsubmit="myFunction()">

Enter name: <input type="text" name="fname">

<input type="submit" value="Submit">

<span id="s1"></span>

</form>

<script>

function myFunction()

{

document.getElementById('s1').innerHTML="Form is Submitted Successfully"

}

</script>

</body>

</html>

**Output:**

Before submitting:

submit.png

After submitting:

C:\Users\Admin\Desktop\Annotation 2022-12-20 230734.png

**onmouseover Event:**

The **onmouseover event** occurs when the mouse pointer is over (enters) on to the selected element.

**Example:**

<html>

<body>

<h1 id="demo" onmouseover="mouseOver()" onmouseout="mouseOut()">Mouse over me</h1>

<script>

function mouseOver() {

document.getElementById("demo").style.color = "red";

}

</script>

</body>

</html>

**Output:**

Before Mouse Over:

out1.png

After Mouse Over:

**over.png**

**Javascript String Methods:**

**1.**[**charAt(indexOfCharacter)**](https://www.geeksforgeeks.org/javascript-string-charat-method/)**:** This method returns the character at the specifiedindex.String in JavaScript has zero-based indexing.

**Parameters:** This method accepts single parameter **indexOfCharacter** that holds the Index of the character of any string.

**Example:**

let text= "HELLOWORLD";  
 let char = text.charAt(0);

**2.**[**concat( objectOfString )**](https://www.geeksforgeeks.org/javascript-string-prototype-concat-function/)**:** This method combines the text of two strings and returns a new combined or joined string. To concatenate two strings, we use **concat()** method on one object of string and send another object of string as a parameter. This method accepts one argument. The variable contains text in double quotes or single quotes.

**Example:**

let text1 = "Hello";  
 let text2 = "World";  
 let text3 = text1.concat(" ", text2);

**3.**[**endsWith(queryString,length):**](https://www.geeksforgeeks.org/javascript-string-endswith-method/) This methods checks whether a string ends with a specified string or characters. This method returns “true” if string ends with the provided string, else it returns “false”. This method is case sensitive. This method accepts two arguments.

* **queryString:** The string to be searched for.
* **length:** The default value is the length of the string you provided

**Example:**

let text= "Helloworld";  
 let result = text.endsWith("world");

**4.**[**includes(queryString):**](https://www.geeksforgeeks.org/javascript-string-includes-method/) This method checks whether the string variable contains specific string or not. This method returns “true” if the string is present in the variable string variable, else it returns “false”. This method is case-sensitive. This method accepts a single parameter **queryString** that holds the string that you want to check if it is present or not.

**Example:**

let text= "Helloworld,welcometotheuniverse.";  
 let result = text.includes("world");

**5**.[**indexOf(queryString)**](https://www.geeksforgeeks.org/javascript-string-prototype-indexof-function/) **:** This method returns the index of first occurrence of given query string. This method returns -1 if a given character or string is not present in the string variable. This method is case-sensitive. This method accepts a single parameter **queryString** that holds the Character or string for getting the index of that string.

**Example:**

let text= "Helloworld,welcometotheuniverse.";  
let result = text.indexOf("welcome");

**6.**[**replace(replaceValue, replaceWithValue)**](https://www.geeksforgeeks.org/javascript-string-replace-method/) **:** This method returns string with the changes. This method is case-sensitive.  
This method accepts two parameters as mentioned above and described below:

* + **replacedValue:** This parameter holds the character that you want to replace.
  + **replaceWithValue:** This parameter holds the character that you want to replace with.

**Example:**

let text= "VisitMicrosoft!";  
 let result = text.replace("Microsoft", "W3Schools");

**7.**[**search(queryString)**](https://www.geeksforgeeks.org/javascript-string-search/) **:** This method search for specified value or regular expression. This method returns the position of the match if it is found and if it is not found, it returns -1. This method is case-sensitive. This method accepts a single parameter **queryString** that holds the string that you want to get the position.

**Example:**

let text= "Mr.Bluehasabluehouse";  
let position = text.search("Blue");

**8.**[**slice(startIndex, endIndex)**](https://www.geeksforgeeks.org/javascript-string-slice/) **:**This method extract a part of string and returns a new string.

This method accepts two arguments.

* + **startIndex:** This parameter holds the index from where you want to extract. It is included.
  + **endIndex:** This parameter holds the index till where you want to extract. It is excluded.

**Example:**

let text= "Helloworld!";  
 let result = text.slice(0, 5);

**9.**[**split(character):**](https://www.geeksforgeeks.org/javascript-string-prototype-split-function/) This method splits the string into an array of sub-strings. This method returns an array. This method accepts single parameter **character** on which you want to split the string.

**Example:**

let text= "Howareyoudoingtoday?";  
const myArray = text.split(" ");

**10.**[**startsWith(queryString)**](https://www.geeksforgeeks.org/javascript-string-startswith/)**:** This method checks if a string starts with a given query string or not. This method returns “true” if string starts with provided query string else it returns “false”. This method accepts single parameter **queryString** that you want to check existing string start with it or not.

**Example:**

let text= "Helloworld,welcometotheuniverse.";  
 text.startsWith("Hello");

**11.**[**toLowerCase(stringVariable)**](https://www.geeksforgeeks.org/javascript-string-prototype-tolowercase/)**:** This method converts all the character present in string to lower case and returns a new string with all the characters in lower case. This method accepts single parameter **stringVariable** string that you want to convert in lower case.

**Example:**

let text= "HelloWorld!";  
let result = text.toLowerCase();

**12.**[**toUpperCase(stringVariable)**](https://www.geeksforgeeks.org/javascript-string-touppercase/)**:** This method converts all the character present in String to upper case and returns a new String with all character in upper case. This method accepts single parameter **stringVariable** string that you want to convert in upper case.

**Example:**

let text= "HelloWorld!";  
let result = text.toUpperCase();

**13.**[**trim()**](https://www.geeksforgeeks.org/javascript-string-prototype-trim-function/)**:** This method is used to remove either white spaces from the given string. This method returns a new string with removed white spaces. This method called on a String object. This method doesn’t accept any parameter.

**Example:**

let text= "      HelloWorld!       ";  
let result = text.trim();

**Javascript Array Methods :**

* **Pop():**

The  pop()  method removes the last element from an array:

**Example:**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.pop();

* **Push():**

The push() method adds a new element to an array (at the end):

**Example :**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.push("Kiwi");

* **shift():**

The shift() method removes the first array element and "shifts" all other elements to a lower index**.**

**Example :**

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.shift();

* **unshift():**

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

**Example :**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.unshift("Lemon");

* **splice():**

The splice() method can be used to add new items to an array:

**Example :**

const fruits = ["Banana", "Orange", "Apple", "Mango"];  
 fruits.splice(2, 0, "Lemon", "Kiwi");

* **slice():**

The slice() method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

**Example :**

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
 const citrus = fruits.slice(1);

**Javascript Form Validation:**

* JavaScript provides a way to validate form's data on the client's computer before sending it to thewebserver. Form validation generally performs two functions. Basic Validation − First of all, the form must be checked to make sure all the mandatory fields are filled in.

**Why form validation:**

* Form validation is required to prevent online form abuse by malicious users. Improper validation of form data is one of the main causes of security vulnerabilities. It exposes your website to attacks such as header injections, cross-site scripting, and SQL injections.

**How to validate form:**

* Validating the form is nothing but validating the each input field in the form.
* There are different types of input fields in form like checkboxes, radio buttons.
* If we validate the input each input field then form validation is completed.

**Some types of input fields with validations:**

**text box:**

defines a single-line text field. The default width of the text field is 20 characters.

**Syntax:**

function validateForm() {  
  let x = document.forms["myForm"]["fname"].value;  
  if (x == "") {  
    alert("Name must be filled out");  
    return false;  
  }  
}

**password:**

The **<input type="password">** defines a password field (characters are masked).

**Syntax:**

function verifyPassword() {

   var pw = document.getElementById("pswd").value;

  //check empty password field

  if(pw == "") {

     document.getElementById("message").innerHTML = "\*\*Fill the password please!";

     return false;

  }

 //minimum password length validation

  if(pw.length **<** **8**)

{

     document.getElementById("message").innerHTML = "\*\*Password length must be atleast 8 characters";

     return false;

  }

//maximum length of password validation

  if(pw.length **>** 15)

 {

     document.getElementById("message").innerHTML = "\*\*Password length must not exceed 15 characters";

     return false;

  }

 else

 {

     alert("Password is correct");

  }

}

**Checkbox:**

The <input type="checkbox"> defines a checkbox. The checkbox is shown as a square box that is ticked (checked) when activated

**Syntax:**

function validateForm(form)

{

console.log("checkbox checked is ", form.agree.checked);

if(!form.agree.checked)

{

document.getElementById('agree\_chk\_error').style.visibility='visible';

return false;

}

else

{

document.getElementById('agree\_chk\_error').style.visibility='hidden';

return true;

}

}

**Number:**

The <input type="number"> defines a field for entering a number

**Syntax:**

function phonenumber(inputtxt)

{

var phoneno = /^\d{10}$/;

if((inputtxt.value.match(phoneno)){

return true;

}

else{

alert("message");

return false;

}

}

**Radio buttons:**

Radio buttons are normally presented in radio groups (a collection of radio buttons describing a set of related options). Only one radio button in a group can be selected at the same time.

**Syntax:**

var getSelectedValue = document.querySelector( 'input[name="season"]:checked');

if(getSelectedValue != null) {

document.write("Radio button is selected");

}

else {

document.write("Nothing has been selected");

}

**Select:**

Select is used to insert the dropdown list in the form.

**Syntax:**

function Dropdown\_Validation(ddlId) {

var empty = document.getElementById(ddlId).value;

if (empty == "0") {

alert('Please select an item');

return false;

}

return true;

}

**Example:**

<html>

<head>

<script>

function SLJR() {

var name =document.forms.RegForm.Name.value;

var email =document.forms.RegForm.EMail.value;

var phone =document.forms.RegForm.Telephone.value;

var what =document.forms.RegForm.Subject.value;

var password =document.forms.RegForm.Password.value;

var address = document.forms.RegForm.Address.value;

var regEmail=/^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/g;

var regPhone=/^\d{10}$/; // Javascript reGex for Phone Number validation.

var regName = /\d+$/g; // Javascript reGex for Name validation

if (name == "" || regName.test(name)) {

window.alert("Please enter your name properly.");

name.focus();

return false;

}

if (address == "") {

window.alert("Please enter your address.");

address.focus();

return false;

}

if (email == "" || !regEmail.test(email)) {

window.alert("Please enter a valid e-mail address.");

email.focus();

return false;

}

if (password == "") {

alert("Please enter your password");

password.focus();

return false;

}

if(password.length <6){

alert("Password should be atleast 6 character long");

password.focus();

return false;

}

if (phone == "" || !regPhone.test(phone)) {

alert("Please enter valid phone number.");

phone.focus();

return false;

}

if (what.selectedIndex == -1) {

alert("Please enter your course.");

what.focus();

return false;

}

return true;

}

</script>

<style>

div {

box-sizing: border-box;

width: 100%;

border: 100px solid black;

float: left;

align-content: center;

align-items: center;

}

form {

margin: 0 auto;

width: 600px;

}

</style>

</head>

<body>

<h1 style="text-align: center;">REGISTRATION FORM</h1>

<form name="RegForm" onsubmit="return SLJR()" method="post">

<div>Name: <input type="text" size="65" name="Name" /><div>

<div>Address: <input type="text" size="65" name="Address" /><div>

<div>E-mail Address: <input type="text" size="65" name="EMail" /><div>

<div>Password: <input type="text"size="65" name="Password" /></div>

<div>Telephone: <input type="text" size="65" name="Telephone" /></div>

<div>

SELECT YOUR COURSE

<select type="text" value="" name="Subject">

<option>BTECH</option>

<option>BBA</option>

<option>BCA</option>

</select>

</div>

<div>Comments: <textarea cols="55” name="Comment"></textarea></div>

<div>

<input type="submit" value="send" name="Submit" />

<input type="reset" value="Reset" name="Reset" />

</div>

</form>

</body>

</html>

**Output:**