

# **INDUSTRIAL TRAINING**

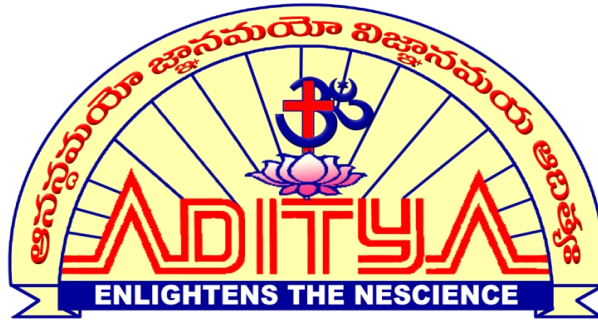
**An Industrial Training report submitted for partial fulfillment of  
requirements for the award of**

## **DIPLOMA IN COMMUNICATION AND COMPUTER NETWORKING**

**SUBMITTED BY**

**M. SURYA CHAKRA VENKAT**

**22249-CCN-038**



**DEPARTMENT OF COMMUNICATION AND COMPUTER ENGINEERING**

**ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY -249**

**(II Shift Polytechnic)**

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

**ADITYA NAGAR, ADB ROAD, SURAMPALEM-533437**

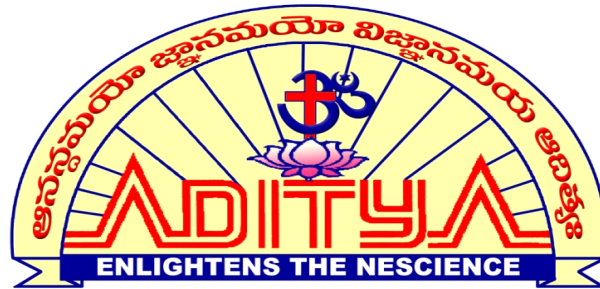
**[www.acet.edu.in](http://www.acet.edu.in)**

**(2022-2025)**

# ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY-249

(II Shift Polytechnic)  
(Approved by AICTE, Affiliated to SBTET)

## DEPARTMENT OF COMMUNICATION AND COMPUTER NETWORKING



### CERTIFICATE

This is to certify that the industrial training report being submitted by **M. SURYA CHAKRA VENKAT** bearing roll no. **22249-CCN-038** in partial fulfillment for the award of the **Diploma in Communication And Computer Networking**. It is record of bonafied work carried out by me under the esteemed guidance and supervision of Mrs. **P. KEERTHANA**.

Mrs. P. KEETHANA

TRAINING GUIDE

LECTURER

Mrs. M. MANI RATNAM

HEAD OF THE DEPT.

COMMUNICATION AND  
COMPUTER ENGINEERING

EXTERNAL EXAMINER

## ACKNOWLEDGEMENT

I would like to take this opportunity to express my profound sense of gratitude to Principal **Mr. A. V. MadhavRao**, 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 **Mrs. M. Mani Ratnam**, Department of Computer 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. D. Vara 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 **Mrs. P. KEERTHANA** Department of Computer 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,  
**M. SURYA CHAKRA VENKAT**  
**22249-CCN-038.**

## **DECLARATION**

We declare that the project entitled “FRONTEND WEB PAGE DEVLEOPMENT” is a genuine project done by us and thirds work has been submitted to the ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY (II SHIFT POLYTECHNIC COLLEGE)

Surampalem Affiliated to the STATE BOARD OF TECHNICAL EDUCATION & TRAINING in partial fulfillment of the degree Diploma. We further declare that this project work has not been submitted in partial fulfillment of the award of any degree of this or any other educational institutions.

# ABSTRACT

Web development is the work involved in developing a website for the Internet (World Wide Web) or an intranet (a private network). Web development can range from developing a simple single static page of plain text to complex web applications, electronic businesses, and social network services.

Therefore, Abstract web style is an approach which is against the conventional web design methods where the elements have definable figurative appearances. This flexibility and freedom make Abstract web design very unique and diverse.

The front end of a website is everything the user either sees or interacts with when they visit the website. It is responsible for the total look and feel of an online experience. While the term may sound a little technical, website front ends are really an everyday encounter for almost all of us

We develop a simple website by using html, css, bootstrap, javascript in web development. We are referenced by some other websites for developing this frontend web page.

Several elements can be used that elements can be reused.

Information presentation of web front-end development technology in network operation process exerts a key function. It cannot influence user experience and applications.

# INDEX

Sl. No.	Topic Name	Page No.
<b>1</b>	<b>HTML</b>	
	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
<b>2</b>	<b>CSS</b>	
	Introduction to CSS	20
	Types of StyleSheets	20-22
	CSS Selectors	23-25
	CSS Background Properties	25
	CSS Box Model	25-26
	CSS Position property	27-28
	Z-index Property	28-29
	CSS Overflow Property	30-34
	Opacity and Box-Shadow	34-35
<b>3</b>	<b>BOOTSTRAP</b>	
	Introduction to Bootstrap	36
	Bootstrap CDN & Offline Link Inserting	36-37
	Bootstrap Grid System	37-41
	Bootstrap Images	41-43
	Bootstrap Table	44-46
	Bootstrap Modal	46-48
	Bootstrap Alerts	48-49
<b>4</b>	<b>JAVASCRIPT</b>	
	Introduction to Javascript	50

	Javascript Function	50-51
	Javascript Arrays	51-53
	Javascript Objects	53-54
	Javascript Operators	54-62
	Javascript DOM Elements	62-65
	Javascript DOM Elements Manipulation	65-66
	Javascript Events	66-68
	Javascript Array Methods	69-70
	Javascript Form Validation	70-75

<b>4</b>	<b>ReactJs</b>	
	Introduction to ReactJs	76
	React Render Html	77-78
	React Events	78-79
	React lists	79-80
	React Forms	80-81
	React Routers	81-84
	React CSS styling	84-85
	React Sass styling	85

<b>5</b>	<b>Industrial Training Project</b>	86-94
----------	------------------------------------	-------

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

### 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: 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: It is used to specify the relation between the current document and the linked document.
- 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: It is used to specify the type of links.
- HTML <a> target Attribute: It specifies the target link.
- 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: It is used to specify the path to the image.
- alt: 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: It is used to specify the height of the image.
- width: 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 <sup>superscript</sup> 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 syntax:

```
<form action = "scriptURL" method=" GET|POST">
```

Form elements like input,textarea etc

```
</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 **text**
- 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:**The bgcolor attribute is used to control the background of an HTML element, specifically page body and table backgrounds. Following is the syntax to use bgcolor attribute with any HTML tag.

```
<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:**

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

**Syntax:** A B

**Example:** div span will match all <span> elements that are inside a <div> element.

**Child Combinator:**

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

**Syntax:** A > B

**Example:** ul > li will match all <li> elements that are nested directly inside a <ul> element.

**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> elements that follow a <p>, immediately or not.

### **Column Combinator:**

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

**Syntax:** A || B

**Example:** col || td will match all <td> elements that belong to the scope of the <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:

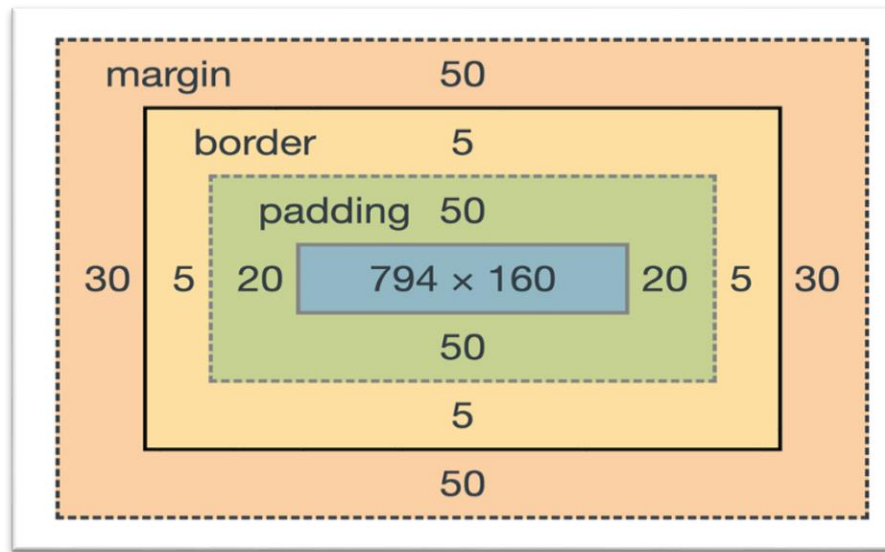
- **background-color:** The background-color property in CSS is used to specify the background color of an element.
- **background-image:** The background-image property is used to set one or more background images to an element.
- **background-repeat:** The background-repeat property in CSS is used to repeat the background image both horizontally and vertically.
- **background-attachment:** 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:** In CSS body-position property is mainly used to set an image at a certain position.
- **background-origin:** The background-origin is a property defined in CSS which helps in adjusting the background image of the webpage.
- **background-clip:** 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 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 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 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.

## 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;
}
```

## **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:**

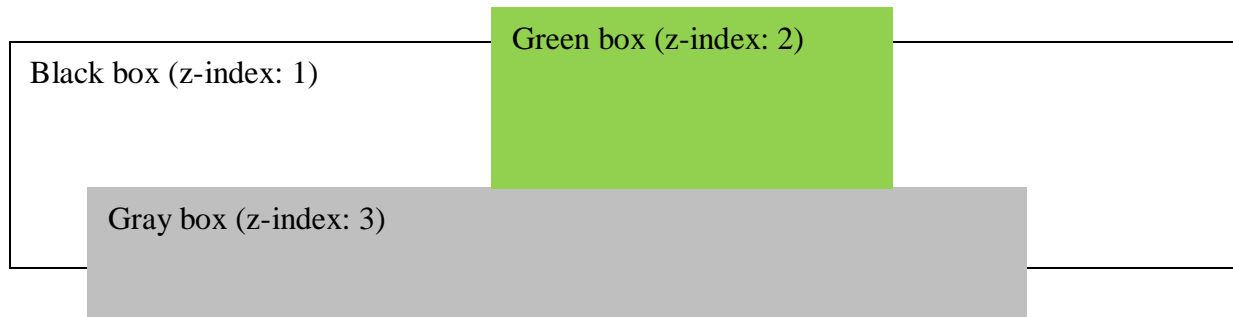
```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
    position: relative;
}
.black-box {
    position: relative;
    z-index: 1;
    border: 2px solid black;
}
```

```

    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:**



## 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: visible

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

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.

## 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: hidden

With the hidden value, the overflow is clipped, and the rest of the content is hidden:

Try to remove the overflow property to understand how it works.

You can use the overflow property when you want to have better control of the layout. The overflow property

### 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)

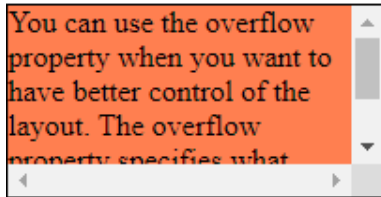
**Example:**

```
<!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: scroll

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):



## Overflow-auto:

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

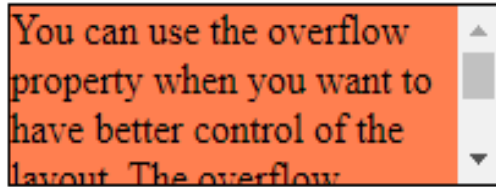
### Example:

```
<!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:

## Overflow: auto

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



## 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.



opacity 0.2



opacity 0.5



opacity 1  
(default)

**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:-



More blurred and red.

# 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 CDN And 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 1px 1px #444;"><p>Lorem
ipsum dolor sit amet, consectetur adipisicing elit.</p>
</div>
<div class="col-xs-6 col-sm-3"
style="background-color: #dedef8;box-shadow: inset 1px -1px 1px #444, inset -
1px 1px 1px #444;">
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.</p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut.</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 1px 1px #444;">
<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. </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>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim 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>Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p></div>
</div>
</div>

```

**Column Ordering:** Another nice feature of Bootstrap grid system is that you can easily write the columns in an order, and show them in another one. You can easily change the order of built-in grid columns with **.col-md-push-\*** and **.col-md-pull-\*** modifier classes where \* range from 1 to 11.

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 1px 1px
#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 1px 1px #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 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>
```

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

## Striped Table:

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

**Syntax:**

```
<table class="table table-striped">
```

**Example:**

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

## Bordered Table:

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

### Syntax:

```
<table class="table table-bordered">
```

### Example:

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

## 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">
```

#	Firstname	Lastname	Age	City	Country
1	Anna	Pitt	35	New York	USA

## 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">

Contextual class name	Lastname	Email
Default	Defaultson	def@somemail.com
Success	Doe	john@example.com
Danger	Moe	mary@example.com
Info	Dooley	july@example.com
Warning	Refs	bo@example.com
Active	Activeson	act@example.com

## 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:

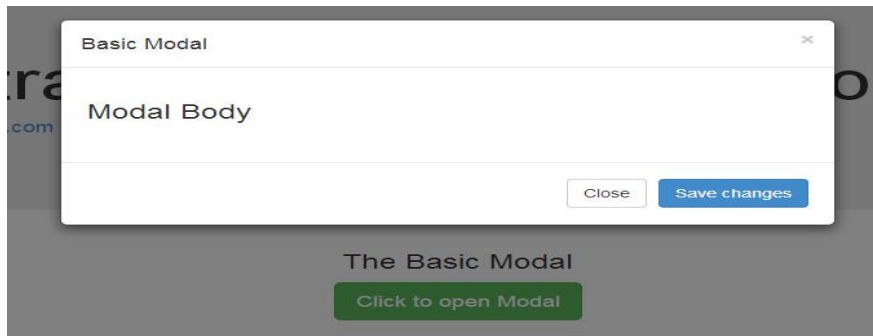
The modal plugin toggles your hidden content on demand, via data attributes or JavaScript. It also adds .modal-open to the <body> to override default scrolling behavior and generates a .modal-backdrop to provide a click area for dismissing shown modals when clicking outside the modal.

### 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 and 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.
  - `<button type="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.

### **Dismissal 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:

**Success!** This alert box could indicate a successful or positive action.

**Info!** This alert box could indicate a neutral informative change or action.

### 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>

```

# 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:

Using an 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 objects in javascript.

**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 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";
var linebreak = "<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;
var linebreak = "<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.	<b>&amp; (Bitwise AND)</b> It performs a Boolean AND operation on each bit of its integer arguments. <b>Ex:</b> (A & B) is 2.
2.	<b>  (Bitwise OR)</b> It performs a Boolean OR operation on each bit of its integer arguments. <b>Ex:</b> (A   B) is 3.
3.	<b>^ (Bitwise XOR)</b> 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. <b>Ex:</b> (A ^ B) is 1.
4.	<b>~ (Bitwise Not)</b> It is a unary operator and operates by reversing all the bits in the operand. <b>Ex:</b> (~B) is -4.

5.	<b>&lt;&lt; (Left Shift)</b> 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. <b>Ex:</b> (A << 1) is 4
6.	<b>&gt;&gt; (Right Shift)</b> Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand. <b>Ex:</b> (A >> 1) is 1
7.	<b>&gt;&gt;&gt; (Right shift with Zero)</b> This operator is just like the >> operator, except that the bits shifted in on the left are always zero. <b>Ex:</b> (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.	<b>= (Simple Assignment )</b> Assigns values from the right side operand to the left side operand <b>Ex:</b> C = A + B will assign the value of A + B into C
2.	<b>+= (Add and Assignment)</b> It adds the right operand to the left operand and assigns the result to the left operand. <b>Ex:</b> C += A is equivalent to C = C + A
3.	<b>-= (Subtract and Assignment)</b> It subtracts the right operand from the left operand and assigns the result to the left operand. <b>Ex:</b> C -= A is equivalent to C = C - A
4.	<b>*= (Multiply and Assignment)</b> It multiplies the right operand with the left operand and assigns the result to the left operand. <b>Ex:</b> C *= A is equivalent to C = C * A
5.	<b>/= (Divide and Assignment)</b> It divides the left operand with the right operand and assigns the result to the left operand. <b>Ex:</b> C /= A is equivalent to C = C / A
6.	<b>%= (Modules and Assignment)</b> It takes modulus using two operands and assigns the result to the left operand. <b>Ex:</b> 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.	<b>? : (Conditional)</b> 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";
var linebreak = "<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>
```

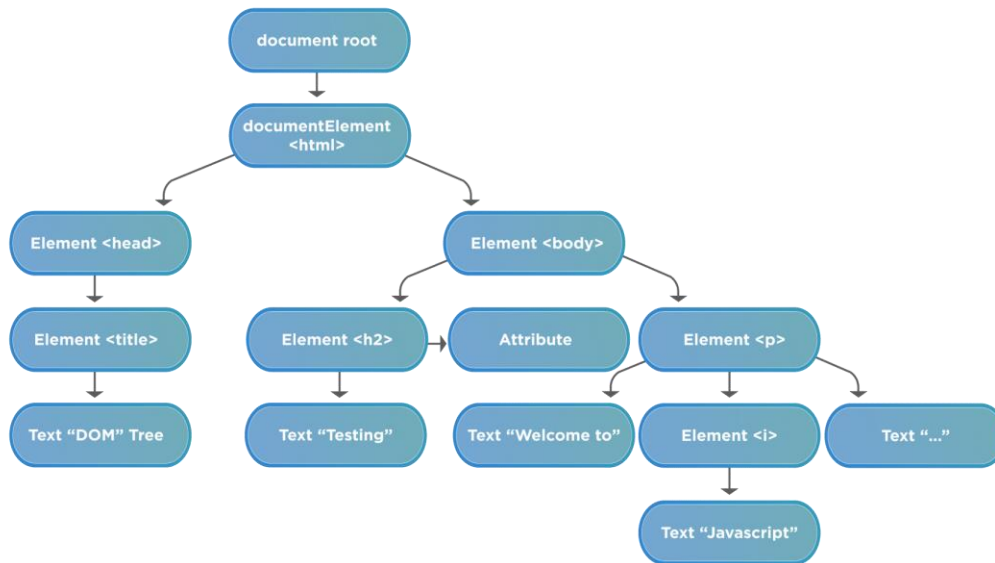
#### Output:

```
Result => B is String
Result => A is Numeric
```

### Javascript DOM Elements:

**DOM** 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.getElementsByTagName("tagName");

**Example:**

```
<html>  
<body>  
<h6>Demonstrating getElementByTag in javascript:</h6><br>  
<b>Industrial Training</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.getElementsByClassName("ClassName");
```

**Example:**

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

```

<h6>Demonstrating getElementByClassName in javascript:</h6></br>
<b class="bold">Industrial Traning </b>
<script type="text/javascript">
    document.getElementByClassName("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.attributeName = 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 me

After Clicking:

Click me

Hello World

**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:

Enter name:

After submitting:

Enter name:

Form is Submitted Successfully

### 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:

**Hai**

After Mouse Over:

**Hai**

## 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 the webserver. 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 at least 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:**

## REGISTRATION FORM

**Name:**

**Address:**

**E-mail Address:**

**Password:**

**Telephone:**

**SELECT YOUR COURSE**

BTECH  
 BBA  
 BCA  
 B.COM  
 GEEKFORGEEKS

[Translate](#)

[Maps](#)

This page says  
 Please enter valid phone number.

[Sign In](#) or [Sign Up](#)

**Name:**

**Address:**

**E-mail Address:**

**Password:**

**Telephone:**

# React JS

## What is React?

- React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.
- React is a tool for building UI components.
- ReactJS, commonly referred to as React, is an open-source JavaScript library for building user interfaces, particularly for single-page applications. It is maintained by Facebook and a community of individual developers and companies. React allows developers to create large web applications that can change data, without reloading the page.

## How does React Work?

- Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.
- **Component-Based Architecture:** ReactJS allows developers to build reusable UI components, each managing its own state, making it easier to develop dynamic and interactive web applications.
- **JSX and Virtual DOM:** React uses JSX, a syntax similar to HTML, and maintains a virtual DOM to optimize performance by efficiently updating the actual DOM only when necessary.
- **Efficient State Management:** With concepts like state and props, React ensures efficient data handling and rendering, while lifecycle methods provide hooks for different phases of a component's lifecycle.

## React Render HTML:

### The render Method

- The `render()` method is then called to define the React component that should be rendered.
- But render where?
- There is another folder in the root directory of your React project, named "public". In this folder, there is an `index.html` file.
- You'll notice a single `<div>` in the body of this file. This is where our React application will be rendered.

### Example: -

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<p>Hello</p>);
```

### OUTPUT



## React Components

- Components are like functions that return HTML elements.
- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.
- A class component must include the `extends React.Component` statement. This statement creates an inheritance.

to `React.Component`, and gives your component access to `React.Component`'s functions.

## React Class Components

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a `render()` function.
- Components come in two types, Class components and Function components, in this chapter you will learn about Class components.
- When creating a React component, the component's name must start with an upper case letter.
- The component has to include the `extends React.Component` statement, this statement creates an inheritance to `React.Component`, and gives your component access to `React.Component`'s functions.
- The component also requires a `render()` method, this method returns HTML.

## React Events

- Just like HTML DOM events, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.
- React events are written in camelCase syntax:
- `onClick` instead of `onclick`.
- React event handlers are written inside curly braces:
- `onClick={shoot}` instead of `onclick="shoot()"`.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

function Football() {
  const shoot = () => {
    alert("Great Shot!");
  }

  return (
    <button onClick={shoot}>Take the shot!</button>
  );
}
```



```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Football />);
```

## ReactJS Lists

- **Lists** are very useful when it comes to developing the UI of any website. **React Lists** are mainly used for **displaying menus** on a website, for example, the **navbar menu**. In regular JavaScript, we can use **arrays** for creating lists.
- We can create lists in React just like we do in regular [JavaScript](#) i.e. by storing the list in an [array](#). To traverse a list we will use the [map\(\)](#) function. To create a React list, follow these given steps:
- **Step 1:** Create a list of elements in React in the form of an array and store it in a variable. We will render this list as an unordered list element in the browser.
- **Step 2:** We will then traverse the list using the JavaScript `map()` function and update elements to be enclosed between `<li>` `</li>` elements.
- **Step 3:** Finally we will wrap this new list within `<ul>` `</ul>` elements and render it to the DOM.

This example implements a simple list in ReactJS :

```
import React from 'react';
import ReactDOM from 'react-dom';

const numbers = [1,2,3,4,5];

const updatedNums = numbers.map((number)=>{
  return <li>{number}</li>;
});

ReactDOM.render(
  <ul>
    {updatedNums}
  </ul>,
  document.getElementById('root')
);
```

**Output:** The above code will render an unordered list as shown below

- 1
- 2
- 3
- 4
- 5

## React Forms

- **React forms** are the way to collect the user data in a React application. React typically utilize controlled components to manage form state and handle user input changes efficiently. It provides additional functionality such as preventing the default behavior of the form which refreshes the browser after the form is submitted.
- In React Forms, all the form data is stored in the React's component state, so it can handle the form submission and retrieve data that the user entered. To do this we use controlled components.

Example :

```
import React from 'react';
import ReactDOM from 'react-dom';

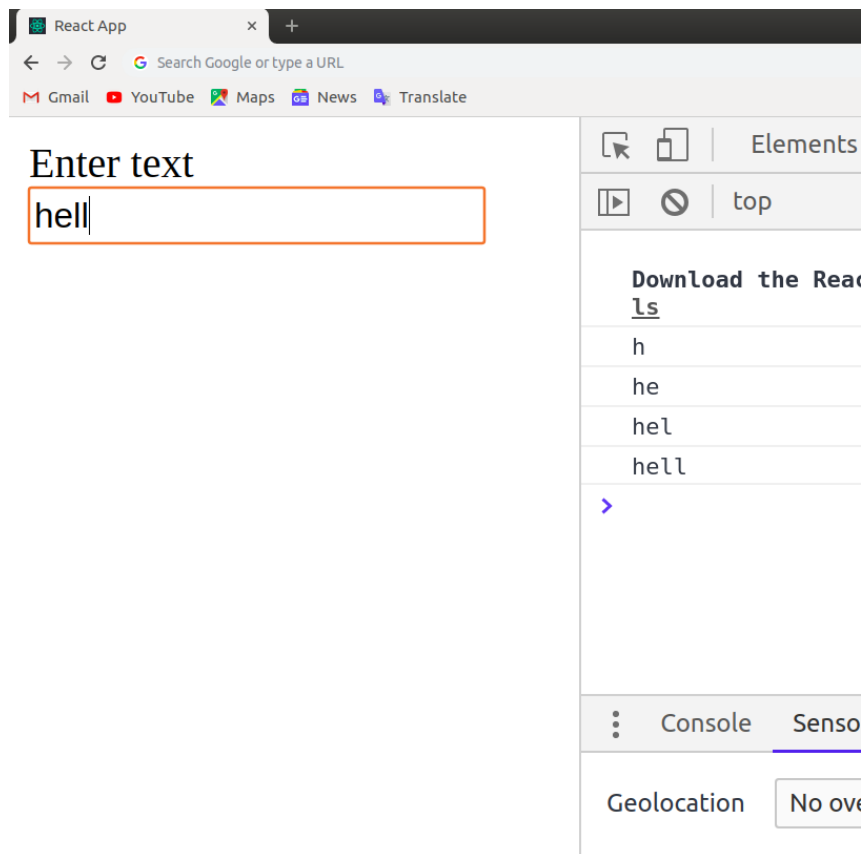
class App extends React.Component {

  onInputChange(event) {
    console.log(event.target.value);
  }

  render() {
    return (
      <div>
        <form>
          <label>Enter text</label>
          <input type="text"
            onChange={this.onInputChange}/>
        </form>
      </div>
    );
  }
}

ReactDOM.render(<App />,
  document.querySelector('#root'));
```

**Output:**



## React Router

- **React Router**, is your essential tool for building single-page applications (SPAs). Imagine users effortlessly transitioning between sections, experiencing your website like a fluid app and React Router makes it possible, paving the way for a delightful user experience and a modern web presence.
- A React website shouldn't mean a Large page reloads every time users navigate.
- As there is no inbuilt routing in React, the React JS Router enables routing support in React and navigation to different components in multi-page applications. It renders components for corresponding routes and assigned URLs.

### Steps to Use React Router

**Step 1:** Initialize a react project. Check this post for [setting up React app](#)

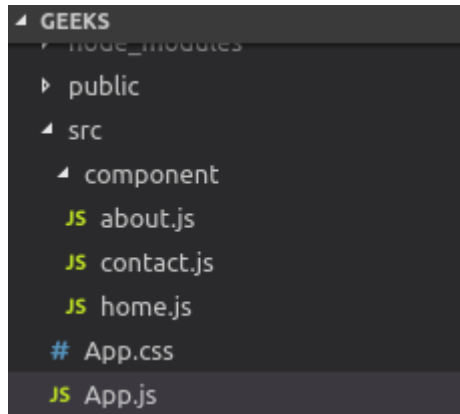
**Step 2:** Install react-router in your application write the following command in your terminal

```
npm i react-router-dom
```

### Step 3: Importing React Router

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
```

#### Folder Structure:



Example :

// Filename - App.js

```
import React, { Component } from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Link,
} from "react-router-dom";
import Home from "./component/home";
import About from "./component/about";
import Contact from "./component/contact";
import "./App.css";

class App extends Component {
  render() {
    return (
      <Router>
        <div className="App">
          <ul className="App-header">
```

```

    <li>
      <Link to="/">Home</Link>
    </li>
    <li>
      <Link to="/about">
        About Us
      </Link>
    </li>
    <li>
      <Link to="/contact">
        Contact Us
      </Link>
    </li>
  </ul>
  <Routes>
    <Route
      path="/"
      element={<Home />}
    ></Route>
    <Route
      path="/about"
      element={<About />}
    ></Route>
    <Route
      path="/contact"
      element={<Contact />}
    ></Route>
  </Routes>
</div>
</Router>
);
}

```

- [Home](#)
- [About Us](#)
- [Contact Us](#)

OUTPUT :

```
}
```

```
export default App;
```

## React CSS

- CSS in React is used to style the React App or Component. The **style** attribute is the most used attribute for styling in React applications, which adds dynamically-computed styles at render time. It accepts a JavaScript object in **camelCased** properties rather than a CSS string.
- There are many ways available to add styling to your React App or Component with CSS. Here, we are going to discuss mainly **four** ways to style React Components, which are given below:

1. Inline Styling
2. CSS Stylesheet
3. CSS Module
4. Styled Components

### Example:

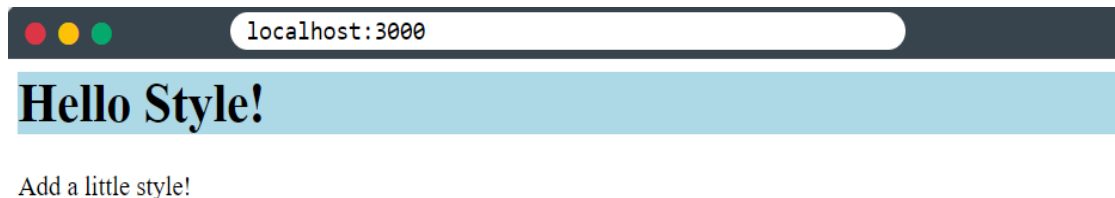
```
import React from 'react';
import ReactDOM from 'react-dom/client';

const Header = () => {
  return (
    <>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

## Styling React Using Sass

- Generally, we recommend that you don't reuse the same CSS classes across different components.
- For example, instead of using a `.Button` CSS class in `<AcceptButton>` and `<RejectButton>` components, we recommend creating a `<Button>` component with its own `.Button` styles, that both `<AcceptButton>` and `<RejectButton>` can render (but [not inherit](#)).
- Following this rule often makes CSS preprocessors less useful, as features like mixins and nesting are replaced by component composition. You can, however, integrate a CSS preprocessor if

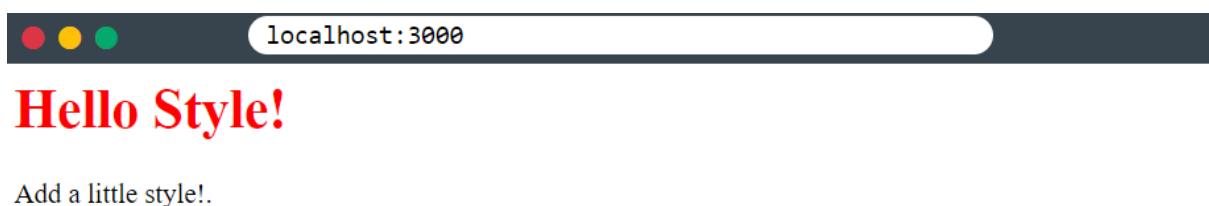


you find it valuable.

- To use Sass, first install `sass`:

```
$ npm install sass
# or
$ yarn add sass
```

Example :



## Industrial Project Work

### Introduction:

The **YouTube Clone Project** is an ambitious endeavor aimed at replicating the core functionalities and content delivery mechanisms of the renowned YouTube platform. This project is developed using **React.js**, a powerful JavaScript library for building user interfaces. By undertaking this project, the goal is to gain a deeper understanding of modern web development practices, enhance proficiency in React.js, and demonstrate the ability to create a fully functional, interactive, and user-friendly video-sharing platform.

**YouTube** is known for its comprehensive video hosting, sharing, and streaming services. By cloning YouTube, this project seeks to create a similar platform that offers structured and easily accessible video content. The project aims to include a variety of features such as video uploads, playback, user authentication, comments, likes, and a responsive design that ensures a seamless user experience across different devices.

The choice of **React.js** for this project is motivated by its component-based architecture, which allows for efficient development and maintenance of complex user interfaces. React.js also offers excellent performance and a rich ecosystem of libraries and tools that enhance the development process. By leveraging React.js, the project aims to implement a modular, scalable, and maintainable codebase.

### **Purpose of the Project:**

The primary purpose of this project is to create a functional and aesthetically similar version of YouTube, providing users with a familiar environment for accessing and sharing video content. By utilizing React.js, the project aims to demonstrate the capabilities of this powerful library in building dynamic, responsive, and component-based web applications. The project will include features such as video uploads, playback, user authentication, comments, likes, and a responsive design to ensure a seamless user experience across different devices.

### **Why React is used for Project?**

**React.js** is chosen for this project due to its component-based architecture, which promotes reusability and maintainability of code. Its virtual DOM feature ensures high performance and smooth updates, making it ideal for building a dynamic video-sharing platform like YouTube. Additionally, the large ecosystem and community support around React.js provide a wealth of resources and tools that facilitate rapid development and troubleshooting.

By cloning YouTube with React.js, this project not only serves as a practical learning tool but also showcases the strengths and best practices of using React.js in modern web development. Through this documentation, we aim to provide a



comprehensive guide to the development process, from initial setup to deployment, ensuring that others can learn from and build upon our work.

## Source Code {website}:

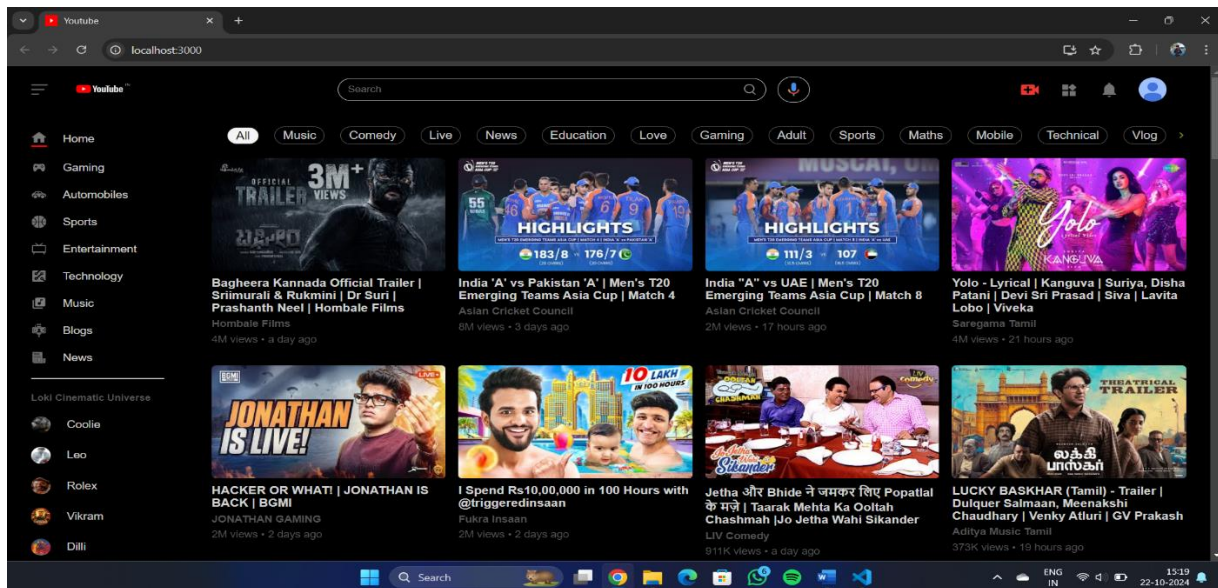
### App.Js:

```
import React, { useState } from 'react'
import Navbar from './Components/Navbar/Navbar'
import Home from './pages/Home/Home'
import Video from './pages/Video/Video'
import {Routes,Route} from 'react-router-dom'

const App = () => {
  const [sidebar,setSidebar] = useState(true);
  return (
    <div>
      <Navbar setSidebar={setSidebar}/>
      <Routes>
        <Route path="/" element={<Home sidebar={sidebar}/>}/>
        <Route path="/video/:categoryId/:videoId" element={<Video/>}/>
      </Routes>
    </div>
  )
}

export default App
```

## Output:



## Pages;

Pages is a folder for develop the Home and Video pages created and stored in Pages folder

## HOME :

If click on the Html page and display the all HTML concepts and all concepts prepare the using pages and created same like as w3 schools website and pages link the “Allroutes “ and also path is mentioned

## Example:

Create the page:

```
import React, { useState } from 'react'
```

```
import './Home.css'
```

```
import Sidebar from '../Components/Sidebar/Sidebar'
```

```
import Feed from '../Components/Feed/Feed'
```

```
import Tags from '../Components/Tags/Tags'
```

```
const Home = ({sidebar}) => {
```

```
  const [category , setCategory] = useState(0);
```

```
  return (
```

```
    <
```

```
      <Sidebar sidebar={sidebar} category={category} setCategory={setCategory}/>
```

```
      <div className={`container ${sidebar?"":"large-container"} `>
```

```

    <Tags className='tags'/>
    <Feed category={category}/>
  </div>
</>
)
}

```

## export default Home

Import the page:

```
import Home from "pages/Home/Home"
```

mentioned the path:

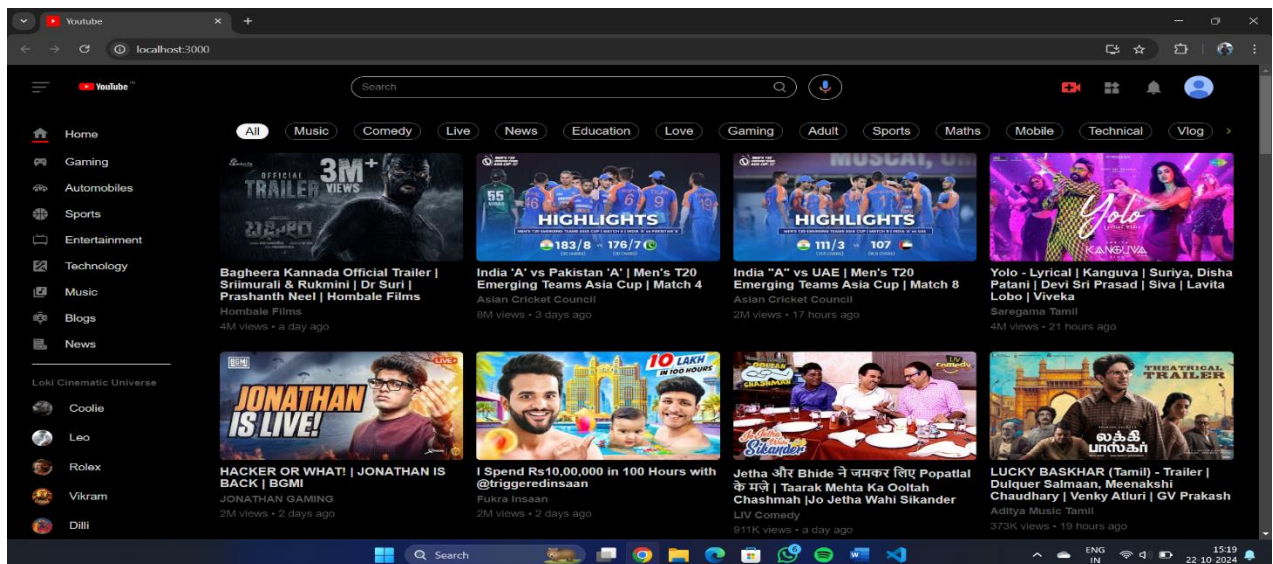
```
{ path: "/", component: <Home /> },
```

Link the page:

```
<Route path="/" element={<Home sidebar={sidebar}/>}/>
```

```
<Route path="/video/:categoryId/:videoId" element={<Video/>}/>
```

## Output:



## Conclusion

**Recreating a YouTube clone using React is a testament to your dedication to mastering web development and a valuable portfolio piece that demonstrates a wide range of skills. This project showcases your advanced proficiency in modern web technologies and highlights your ability to innovate and apply unique design aesthetics. By building a responsive, mobile-friendly video-sharing platform with reusable, scalable components, you not only gain practical experience directly applicable to real-world scenarios but also create a compelling addition to your portfolio. This experience equips you for future challenges in your career, serving as a strong foundation for further exploration and specialization in various aspects of web development.**