



**TOPS TECHNOLOGIES**

Training | Outsourcing | Placement | Study Abroad

# Frontend Development

# All Modules

**Module 1 - [Foundation]**

**Module 2 - [Fundamentals of World Wide Web ]**

**Module 3 - [Fundamentals of IT ]**

**Module 4 - [HTML]**

**Module 5 - [CSS & CSS3]**

**Module 6 - [HTML5]**

**Module 7 - [Bootstrap Basic & Advanced ]**

**Module 8 - [JavaScript Essentials And Advanced ]**

**Module 9 - [React - Components, State, Props ]**

**Module 10 - [ Lists , Hooks , Localstorage , Api Project ]**

**Module 11 - [React -Advanced React- Styling , Routing]**

**Module 12 - [React – JSON-server and Firebase Real Time Database]**

**Module 13 - [React - Applying Redux]**

**Module 14 - [Fetch Data using GraphQL ]**

**Module 15 - [Next JS]**

**Module 16 - [Introduction –Vue.js, D3.js, Chart.js ]**



**TOPS TECHNOLOGIES**  
Training | Outsourcing | Placement | Study Abroad

# **Module - 1**

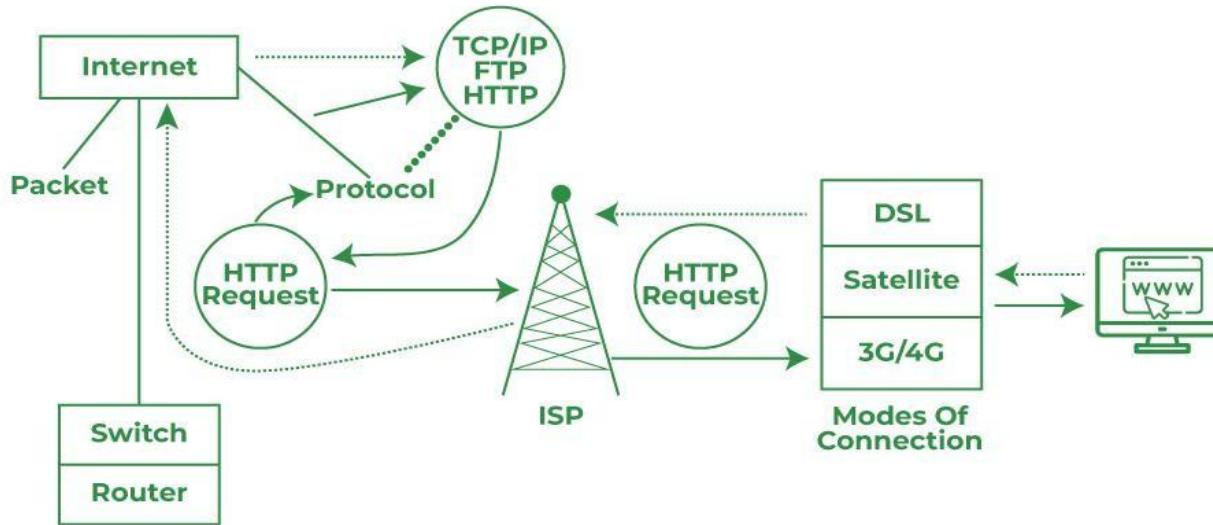
## **[Foundation]**

- ✓ How does the Internet Work
- ✓ DNS and how it works
- ✓ What is HTTP
- ✓ Browsers and how they work?
- ✓ What is Domain Name?
- ✓ What is hosting?
- ✓ Git and GITHUB Training

# How Does the Internet Work?

The Internet is the world's most fascinating invention to date. The journey started back in 1969 as a part of a research program and by the time of the '90s, it became a sensation among everyone. Today, if you're reading this, you should be thankful for the Internet. But have you ever thought about How Simple Internet Works?

# How Does the Internet Work



# DNS and how it works ?

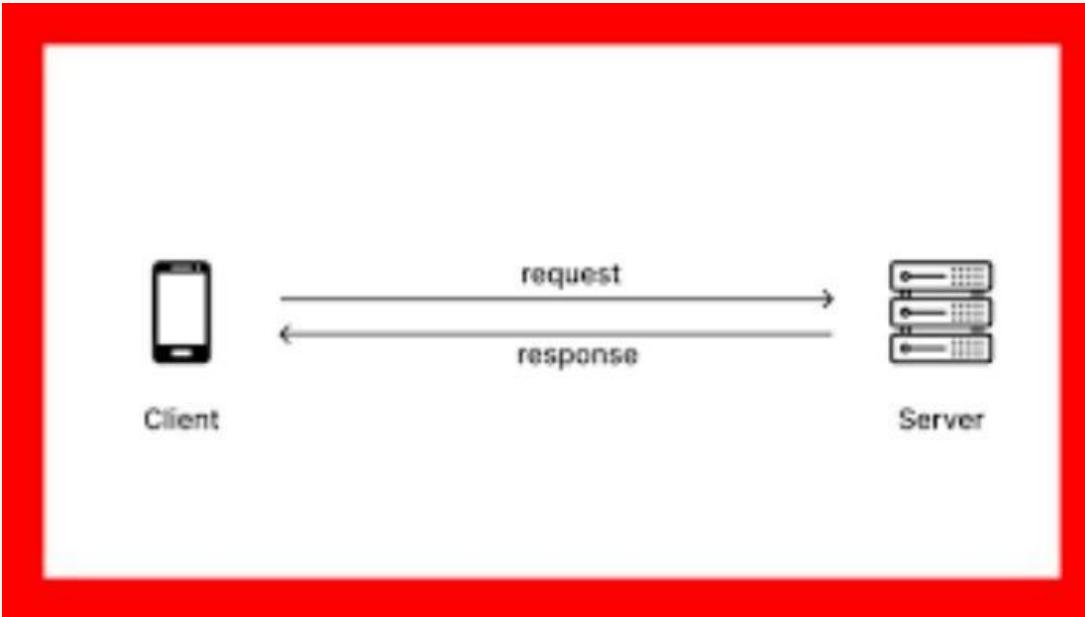
The Domain Name System (DNS) turns domain names into IP addresses, which browsers use to load internet pages. Every device connected to the internet has its own IP address, which is used by other devices to locate the device.

# What is a DNS Server?

A DNS server is a computer with a database containing the public IP addresses associated with the names of the websites an IP address brings a user to. DNS acts like a phonebook for the internet. Whenever people type domain names, like Fortinet.com or Yahoo.com, into the address bar of web browsers, the DNS finds the right IP address. The site's IP address is what directs the device to go to the correct place to access the site's data..

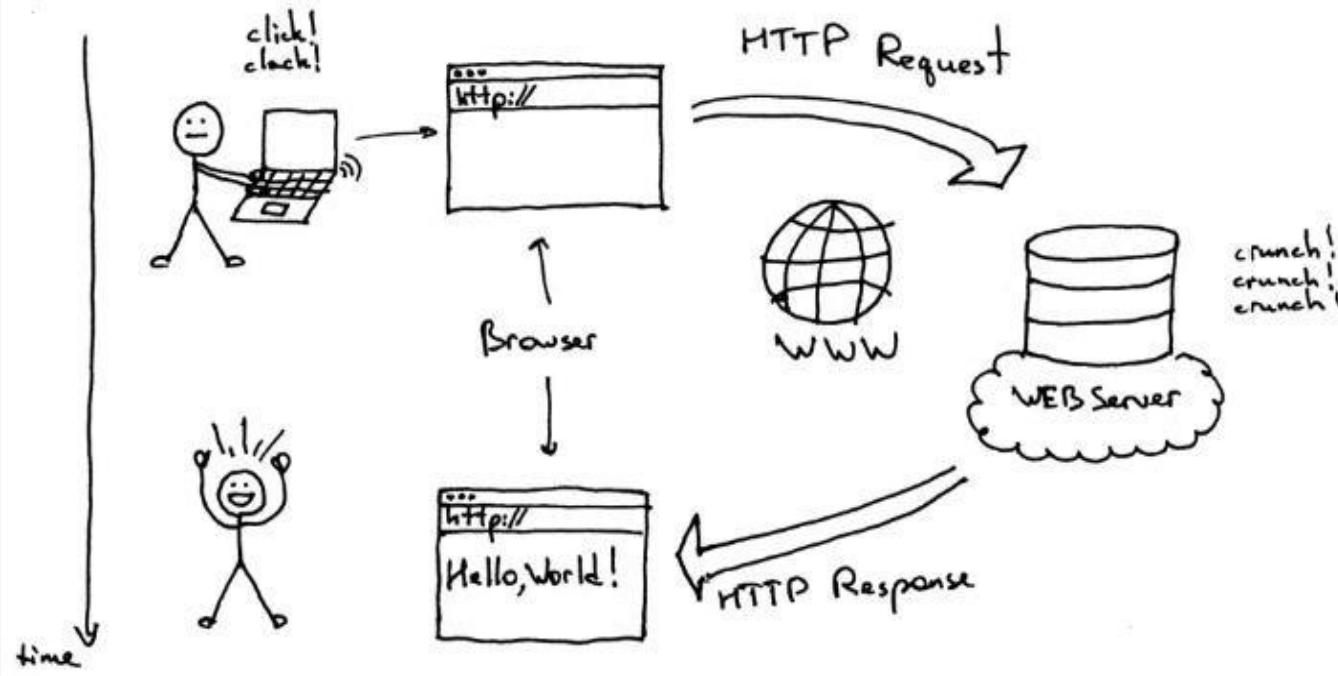
# What is http ?

**HTTP** stands for **H**yper **T**ext **T**ransfer **P**rotocol  
**WWW** is about communication between web  
**clients** and **servers** Communication between  
client computers and web servers is done by  
sending **HTTP Requests** and receiving **HTTP**  
**Responses**



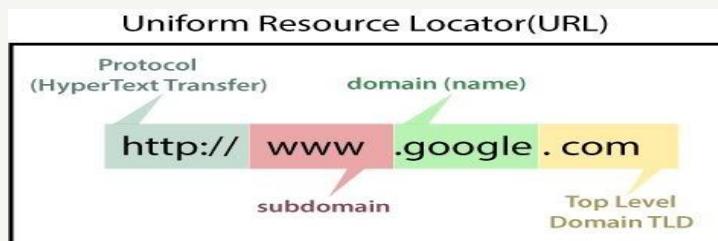
# What is Browser and how it work ?

Web Browser Definition: **A software application used to access information on the World Wide Web** is called a Web Browser. When a user requests some information, the web browser fetches the data from a web server and then displays the webpage on the user's screen.



# What is Domain Name ?

A domain name is a **string of text that maps to an alphanumeric IP address, used to access a website from client software**. In plain English, a domain name is the text that a user types into a browser window to reach a particular website. For instance, the domain name for Google is 'google.com'.



# What is Hosting ?

Web Hosting is like renting space on the Internet or the **web browser**, its equivalent to allocating server space on the **World Wide Web**.

Which secures your dedicated environment for your web domain. **Web hosting** provides a space to keep your website's **data on a server**. When someone enters your domain name into their browser, this server promptly displays your site to them.

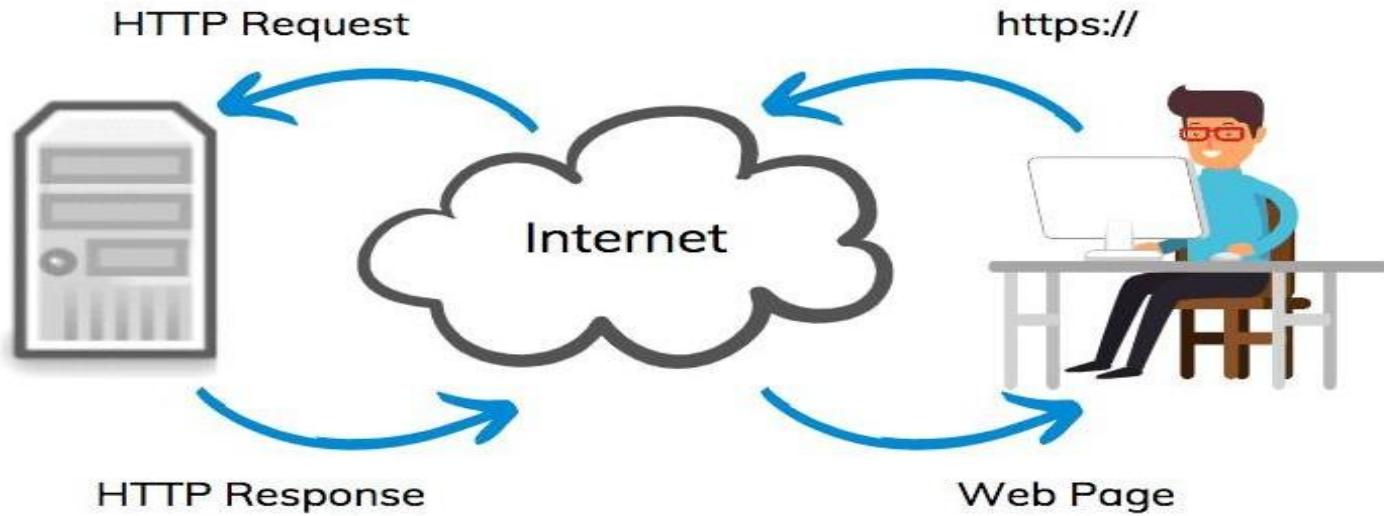
# Hosting

Web hosting is a service that allows organizations and individuals to post a website or web page onto the Internet. A web host, or web hosting service provider, is a business that provides the technologies and services needed for the website or webpage to be viewed in the Internet. Websites are hosted, or stored, on special computers called servers. When Internet users want to view your website, all they need to do is type your website address or domain into their browser.

# How web hosting works?

Web hosting is a service that allows users to store and access their websites online. When a user requests a website, a web hosting server retrieves the website's files and sends them back to the user's browser. The browser then displays the website on the user's screen.

# How Web Hosting Works



# What is Git and Github Training ?

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

## **It is used for:**

Tracking code

changes Tracking who

made changes Coding

collaboration

# What does Git do?

Manage projects with **Repositories**

**Clone** a project to work on a local copy

Control and track changes with **Staging** and **Committing**

**Branch** and **Merge** to allow for work on different parts and versions of a project

**Pull** the latest version of the project to a local copy

**Push** local updates to the main project

# What is Github ?

Git is not the same as GitHub.

GitHub makes tools that use Git.

GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018

**Git Getting Started <https://github.com/>**

# Git all commands

*Create a repository [Shubham-2122/Tops-data](#)*

```
echo "# Tops-data" >> README.md
```

```
git init
```

```
git add README.md
```

```
git commit -m "first commit"
```

```
git branch -M main
```

```
git remote add origin
```

```
https://github.com/Shubham-2122/Tops-data.git
```

```
git push -u origin main
```

# Module - 2

## Fundamentals of World Wide Web

- ✓ Careers in Web Technologies and Job Roles
- ✓ Difference between Web Designer and Web Developer
- ✓ How the Website Works?
- ✓ Client and Server Scripting Languages
- ✓ Domains and Hosting
- ✓ Types of Websites (Static and Dynamic Websites)
- ✓ Web Standards and W3C recommendations
- ✓ Responsive Web Designing
- ✓ Protocol
- ✓ Basics of SEO
- ✓ Basic of html
- ✓ SDLC

# Careers in Web Technologies and Job Roles

The average base salary of a web developer in India is around Rs 3,08,000 per annum that includes around Rs 30,000 in bonuses and Rs 20,000 on a profit-sharing basis. This figure can go up to a maximum of 7,80,000 per annum or even beyond that depending on your experience, skillset, certifications, location, and employer.

India / Job / Web Developer

## Average Web Developer Salary in India

₹308,040

Avg. Salary

Show Hourly Rate

₹29,700  
BONUS

₹22,500  
COMMISSION

₹20,067  
PROFIT SHARING

What am I worth?

Get pay report

How should I pay?

Price a job

The average salary for a Web Developer in India is ₹308,040.



A Web Developer typically makes between ₹123k - ₹777k.

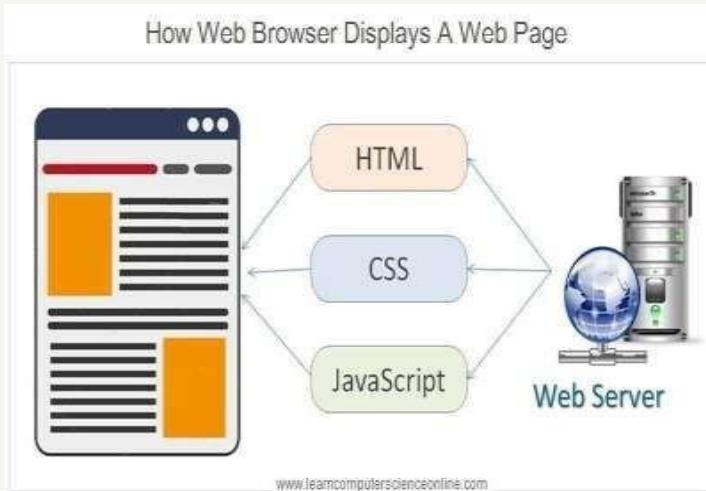


[Click here for practical](#)

# How the Website Works?

**How the web works** provides a simplified view of what happens when you view a webpage in a web browser on your computer or phone.

This theory is not essential to writing web code in the short term, but before long you'll really start to benefit from understanding.



# What is website?

A website is a collection of many web pages, and web pages are digital files that are written using HTML(Hypertext Markup Language). To make your website available to every person in the world, it must be stored or hosted on a computer connected to the Internet round a clock. Such computers are known as a **Web Server**.



# Types of websites Static website

In Static Websites, Web pages are returned by the server which are prebuilt source code files built using simple languages such as HTML, CSS, or JavaScript. There is no processing of content on the server (according to the user) in Static Websites.

Web pages are returned by the server with no change therefore, static Websites are fast .

# Dynamic website

In Dynamic Websites, Web pages are returned by the server which is processed during runtime means they are not pre built web pages, but they are built during runtime according to the user's demand with the help of server-side scripting languages such as PHP, Node.js, ASP.NET and many more supported by the server.

**[Click here for practical](#)**

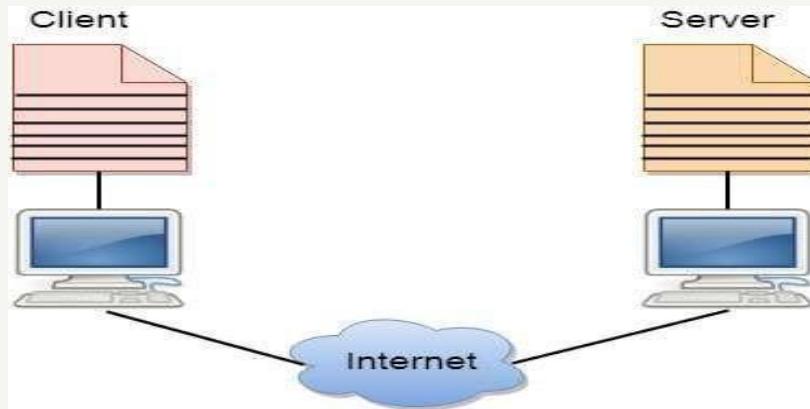
# Client and Server

## Client

A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the service started by the user and terminates when the service is completed.

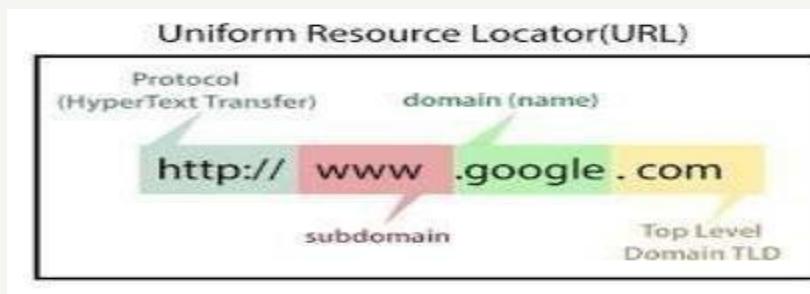
# Server

A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.



# Domains and Hosting Domain

A Domain name is an address where one can find the website by typing the web address in the browser URL bar to visit a website. When you enter the domain name Of the website in the search box, a powerful engine searches the web's largest pool of names and takes us to the website



# W3C standard

The World Wide Web Consortium (W3C) develops international Web standards: HTML, CSS, and many more. W3C's Web standards are called *W3C Recommendations*. All W3C standards are reviewed for accessibility support by the Accessible Platform Architectures ([APA](#)) Working Group.

The W3C standards and Working Group Notes introduced below are particularly relevant to accessibility.



# W3C Recommendation

The W3C Recommendation Track process is designed to maximize consensus about the content of a technical report, to ensure high technical and editorial quality, and to earn endorsement by W3C and the broader community.

**[Click here for practical](#)**



# Responsive web designing

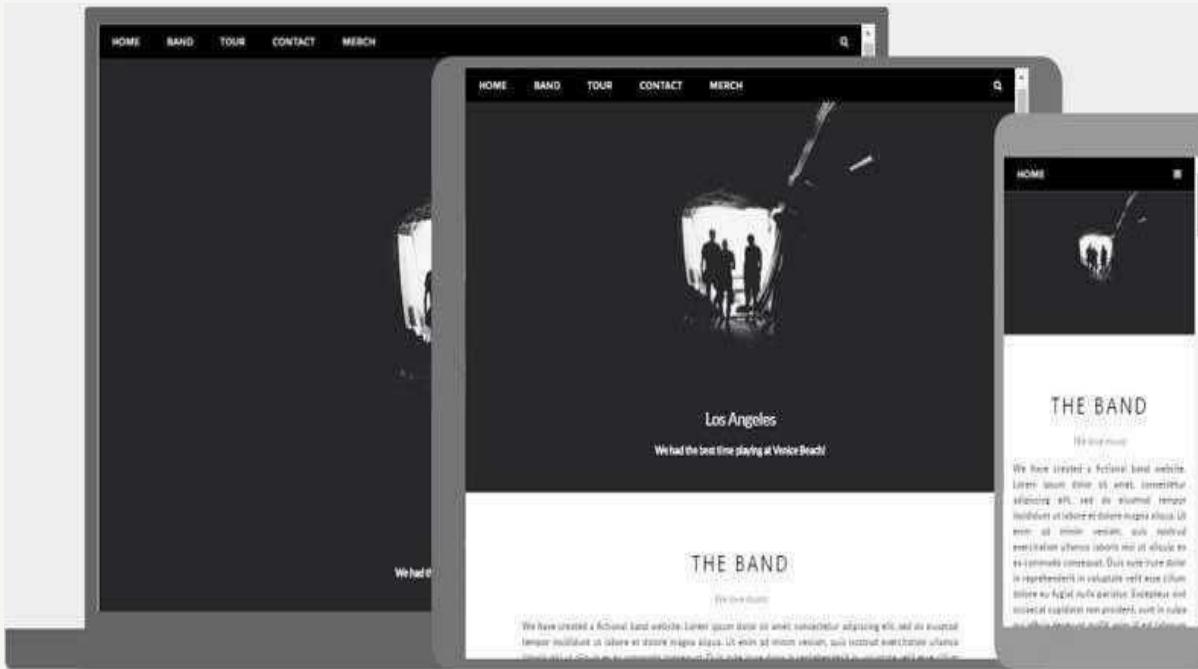
Responsive web design is about creating web pages that look good on all devices!

A responsive web design will automatically adjust for different screen sizes and viewports.

## Setting The Viewport

To create a responsive website, add the following <meta> tag to all your web pages:

Syntax :    `<meta name="viewport" content="width=device-width,  
initial-scale=1.0">`



[Click here for practical](#)

# Facebook Pixel

The Facebook Pixel is a piece of code provided by Facebook that you can add to your website to track user activity and measure the effectiveness of your ads. It helps you understand how users interact with your site after seeing your Facebook or Instagram ads. This tool allows you to gather data about actions taken on your website, such as purchases, sign-ups, or page views.

## Main functions of the Facebook Pixel:

1. **Tracking Conversions:** It helps you track actions on your site that were triggered by users clicking on your ads. For example, if someone clicks on an ad and then makes a purchase, the Pixel records that as a conversion.
2. **Retargeting:** You can use the Pixel to retarget people who visited your site but didn't take a desired action, like completing a purchase or signing up. This allows you to serve them ads that encourage them to return and finish what they started.
3. **Creating Lookalike Audiences:** The Pixel allows you to build lookalike audiences based on the people who have already visited your website. Facebook then finds other users who resemble your existing audience, making it easier to target new customers.
4. **Optimizing Ads:** By collecting data from Pixel events, Facebook's algorithms can optimize your ad delivery, ensuring your ads are shown to the people most likely to take the desired action.
5. **Analytics:** It provides insights into your ad performance, helping you understand how users interact with your website after seeing your ads.

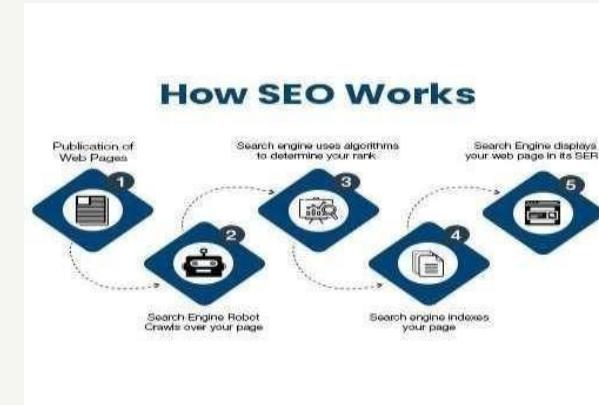
# Protocol

It is a digital language through which we communicate with others on the Internet. protocol meaning is that it a set of mutually accepted and implemented rules at both ends of the communications channel for the proper exchange of information.

# What is SEO

Search engine optimization is the process of improving the quality and quantity of website traffic to a website or a web page from search engines. SEO targets unpaid traffic rather than direct traffic or paid traffic.

[\*\*Click here for practical\*\*](#)



Well, SEO stands for 'Search Engine Optimization', which is **the process of getting traffic from free, organic, editorial, or natural search results in search engines**. It aims to improve your website's position in search results pages. Remember, the higher the website is listed; the more people will see it.



# What is SDLC

SDLC stands for **Software Development Life Cycle**, which is a structured approach used by software developers to design, develop, test, and deploy software applications. It provides a systematic process to ensure the creation of high-quality software within a certain timeframe and budget.

The SDLC is divided into several stages, which typically include:

**Planning/Requirements Gathering:** This phase involves understanding and documenting the business needs and technical requirements. Stakeholders discuss what features and functionalities the software should have.

**System Design:** Based on the requirements, the system's architecture and design are created. This could include the high-level structure of the software, database design, and user interfaces.

**Implementation (Coding):** The actual code is written in this phase. Developers start building the software based on the design specifications.

**Testing:** Once the software is built, it goes through various tests to find and fix any bugs or issues. This ensures the software meets the required quality standards.

**Deployment:** After testing, the software is deployed to the production environment, where users can start using it.

**Maintenance:** Once the software is live, it enters the maintenance phase, where updates, bug fixes, and improvements are made as required by the users.

# **Module – 3**

# **Fundamental of IT**

# Web development tools and environments

## Code Editors & IDEs (Integrated Development Environments)

These tools are used for writing and editing code:

- **Visual Studio Code (VS Code):** A popular, lightweight, and customizable code editor with a rich set of extensions for web development, including support for JavaScript, HTML, CSS, and frameworks like React or Angular.
- **Sublime Text:** A fast, minimalistic code editor known for its speed and powerful features like multiple cursors and customizable key bindings.
- **Atom:** An open-source text editor built by GitHub that is highly customizable and supports a wide variety of plugins.
- **WebStorm:** A commercial IDE designed specifically for JavaScript and web development with great support for frameworks like React, Angular, and Node.js.
- **Brackets:** An open-source code editor with a focus on web development, featuring live preview and preprocessor support.

# Visual Studio

Visual Studio Code, commonly referred to as VS Code is an integrated development environment developed by Microsoft for Windows, Linux, macOS and web browsers. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded version control with Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.



# sublime

Sublime Text is a text and source code editor featuring a minimal interface, syntax highlighting and code folding with native support for numerous programming and markup languages, search and replace with support for regular expressions, an integrated terminal/console window, and customizable themes. Available for Windows, macOS, and Linux, its functionality can be expanded with plugins written in Python.



# Version Control Systems

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

Enhances the project development speed by providing efficient collaboration,

Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,

Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change.,

# Package Managers - NPM, Yarn, composer

Yarn is used for handling dependencies within JavaScript applications. It serves as both a package manager and a project manager. Whether you work on basic projects or complex industry-level monolithic repositories, whether you contribute to open-source initiatives or are part of an enterprise environment, Yarn always provides reliable support.

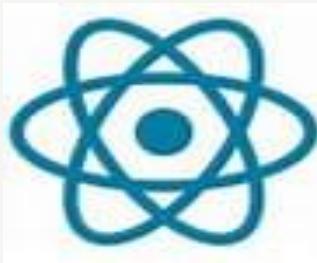
NPM, short for Node Package Manager, is the default package manager for NodeJS. It is a command-line utility that allows you to install, manage, and share packages or modules of JavaScript code. These packages can range from small utility libraries to large frameworks, and they can be easily integrated into Node JS projects to extend their functionality.

# Grunt, Gulp, Webpack, Parcel



JavaScript task runners: Grunt vs Gulp major difference		
	Grunt	Gulp
Concept	Configuration over coding	Coding over configuration
Process execution	Temporary files on a local disk	Memory
Format of source data	JSON	JavaScript
Number of simultaneous tasks	One	Multiple
Requires to be proficient in Node.js	No	Yes
Able to set up task dependencies	No	Yes

# Frameworks and Libraries



# Development and Testing Servers



# Database management systems



# Front End Build Tools



BABEL



less

# Development Environments XAMPP, MAMP



# Design and Prototyping Tools - Figma, Adobe XD, Sketch



# **Continuous Integration and Deployment (CI/CD) Tool - Jenkins, Travis, CircleCI, Github Actions**

Continuous Integration and Continuous Deployment (CI/CD) are practices that allow development teams to deliver software updates quickly and reliably. CI/CD tools help automate the processes of integrating code changes, testing, and deploying the application, ensuring that software is always in a deployable state. Here's an overview of the most popular CI/CD tools: **Jenkins**, **Travis CI**, **CircleCI**, and **GitHub Actions**.

## 1. Jenkins

**Jenkins** is one of the most widely used open-source automation servers, primarily designed to help with continuous integration and continuous delivery.

### Key Features:

- **Open Source:** Free to use, with a large community of plugins and contributors.
- **Extensive Plugin Ecosystem:** Jenkins has an extensive plugin library for almost any tool you can think of, making it highly customizable.
- **Pipeline as Code:** Jenkins supports defining pipelines in a domain-specific language (DSL), allowing you to define CI/CD workflows as code ([Jenkinsfile](#)).
- **Integration with Version Control Systems:** It integrates seamlessly with Git, Subversion, and other version control systems, triggering builds on code commits.
- **Distributed Builds:** Jenkins allows distributed builds, meaning you can run builds on multiple machines in parallel, reducing build times.

## 2. Travis CI

**Travis CI** is a cloud-based CI/CD tool that integrates directly with GitHub repositories, automating the process of building, testing, and deploying applications.

### Key Features:

- **GitHub Integration:** Travis CI integrates seamlessly with GitHub repositories, automatically triggering builds on push or pull requests.
- **Build Matrix:** Allows running tests in parallel on different environments (e.g., different versions of programming languages, operating systems).
- **Easy Configuration:** Configuration is done through a `.travis.yml` file, which defines the build steps and environment settings.
- **Cloud and Self-hosted:** Travis CI offers a cloud-hosted service and also allows you to host your own Travis CI instance for private repositories.

### 3. CircleCI

CircleCI is a modern CI/CD tool that offers cloud-based and on-premise solutions to automate testing, building, and deployment pipelines.

#### Key Features:

- **Cloud and Self-hosted Options:** CircleCI provides both cloud-hosted and self-hosted solutions, giving teams flexibility in how they deploy and manage their CI/CD pipelines.
- **Docker Support:** CircleCI has strong support for Docker, allowing you to build and test in isolated environments (containers).
- **Fast Parallelism:** CircleCI offers a highly efficient pipeline that runs tests and builds in parallel, significantly reducing build times.
- **Flexible Configuration:** CircleCI uses YAML configuration files ([.circleci/config.yml](#)), which define pipelines, workflows, and build steps.
- **Integration with GitHub and Bitbucket:** CircleCI integrates easily with GitHub and Bitbucket for version control and source code management.

## 4. GitHub Actions

**GitHub Actions** is GitHub's native CI/CD tool, tightly integrated into the GitHub platform. It allows you to automate your software workflows, such as testing and deployment, directly from your GitHub repository.

### Key Features:

- **Native Integration with GitHub:** GitHub Actions is built into GitHub, so there's no need for external CI/CD services like Jenkins, Travis, or CircleCI. It directly integrates into GitHub workflows.
- **Customizable Workflows:** Workflows are defined in `.yaml` files and can trigger on various GitHub events (e.g., pushes, pull requests, releases).
- **Matrix Builds:** GitHub Actions allows you to define matrix builds, where the same set of tests can be run on multiple environments or versions of a language.
- **Free for Public Repositories:** GitHub Actions is free for public repositories and offers generous free usage for private repositories with GitHub plans.
- **Container Support:** GitHub Actions works well with Docker and other containerized environments, allowing you to define workflows within containers.

# Module – 4

# HTML

- ✓ Introduction to HTML
- ✓ Tags in HTML
- ✓ HTML Attribute
- ✓ HTML Elements
- ✓ Text Formatting in HTML
- ✓ IFRAME And File Path in HTML
- ✓ HTML Tables
- ✓ HTML List
- ✓ HTML FORMS
- ✓ HTML Head, ID, Class and Layout
- ✓ HTML Entities
- ✓ HTML Events

- ✓ HTML Layout
- ✓ HTML Head
- ✓ HTML Meta
- ✓ HTML Elements
- ✓ HTML Scripts
- ✓ HTML Doctypes

- ✓ Advance HTML
- ✓ HTML Audio and Video

Tag

- ✓ HTML SVG
- ✓ Scalable Vector Graphics
- ✓ Canvas and URL in HTML
- ✓ URL Encode
- ✓ XHTML
- ✓ API in HTML5

# Browsers

Google Chrome

Internet

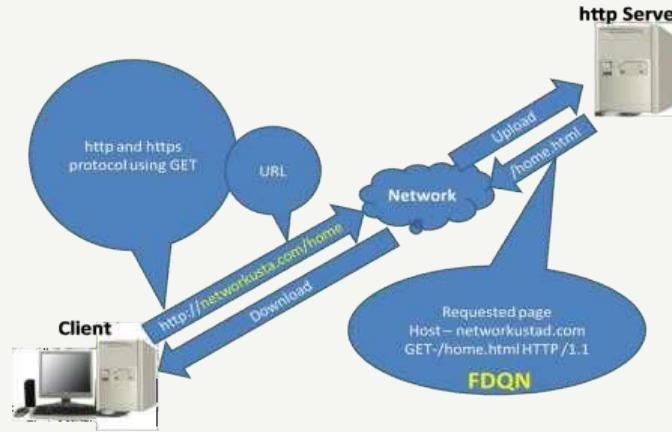
Explorer Firefox

Safari

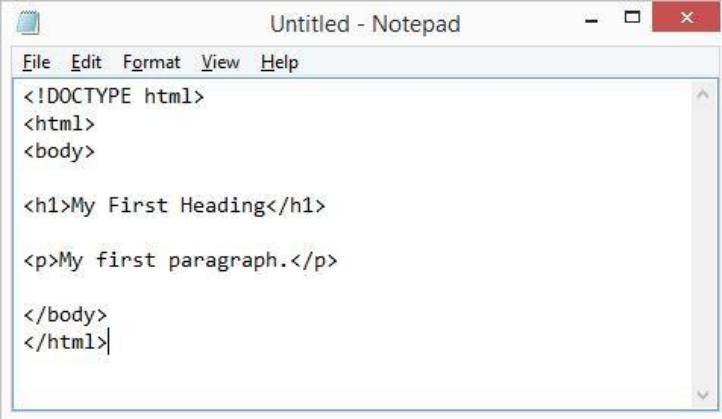
Edge

# Hypertext Transfer Protocol

The Hypertext Transfer Protocol is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems



# Text Editor



Untitled - Notepad

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

**Notepad**



untitled • Sublime Text (UNREGISTERED)

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <h1>My First Heading</h1>
6
7  <p>My first paragraph.</p>
8
9  </body>
10 </html>
```

Line 10, Column 8      Tab Size: 4      HTML

**Sublime Text**

# What is HTML

HTML stands for Hypertext Markup Language.  
HTML is used to create web pages and web applications. HTML is widely used language on the web.

We can create a static website by HTML only.  
Technically, HTML is a Markup language rather than a programming language.

**[Click here for practical](#)**

# Introduction of html

HTML is the standard markup language for creating Web pages.

HTML stands for Hyper Text Markup Language

HTML is the standard markup language for creating Web pages

HTML describes the structure of a Web page

HTML consists of a series of elements

HTML elements tell the browser how to display the content

HTML elements label pieces of content such as "this is a heading",  
"This is a paragraph", "this is a link", etc.

**[Click here for practical](#)**

The <!DOCTYPE html> declaration defines that this document is an HTML5 document  
The <html> element is the root element of an HTML page

The <head> element contains meta information about the HTML page  
The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)

The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists,etc.

The <h1> element defines a large heading  
The <p> element defines a paragraph

# HTML Tags

HTML tags are like keywords which defines that how web browser will format and display the content.

With the help of tags, a web browser can distinguish between an HTML content and a simple content.

HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags.

**[Click here for practical](#)**

# Rules for Tags

All HTML tags must enclosed within < > these brackets.

Every tag in HTML perform different tasks.

If you have used an open tag <tag>, then you must use a close tag </tag> (except some tags)

Ex. <p> </p> , <strong> </strong>

**[Click here for practical](#)**

# HTML Meta Tags

```
<!DOCTYPE html>
<title>
<link>
<meta>
<style>
```

[\*\*\*Click here for practical\*\*\*](#)

# HTML Text Tags

paragraph tag :- <p> </p>

heading tag :- <h1>, <h2>, <h3>, <h4>, <h5>, <h6>

semantic tag :-

<strong>

<em>

<cite>

[\*\*\*Click here for practical\*\*\*](#)

# HTML Unclosed Tags

<br>  
<hr>  
<img>

**[Click here for practical](#)**

# HTML Element

These elements are responsible for creating web pages and define content in that webpage.

An element is a collection of start tag, attributes, end tag, content between them.

**[Click here for practical](#)**

# Void Element

Void element: All the elements in HTML do not require to have start tag and end tag, some elements does not have content and end tag such elements are known as Void elements or empty elements. These elements are also called as unpaired tag.

Ex.

```
<br>  
<hr>
```

[Click here for practical](#)

# Block-level element

These are the elements, which structure main part of webpage, by dividing a page into coherent blocks.

A block-level element always start with new line and takes the full width of web page, from left to right.

These elements can contain block-level as well as inline elements

[\*\*Click here for practical\*\*](#)

# Inline elements:

Inline elements are those elements, which differentiate the part of a give text and provide it a particular function.

These elements does not start with new line and take width as per requirement.

The Inline elements are mostly used with other elements.

[\*\*Click here for practical\*\*](#)

# Useful HTML Elements

HTML Link Tags

HTML Image HTML

List HTML Table

HTML Form

# Attributes

HTML attributes are special words which provide additional information about the elements or attributes are the modifier of the HTML element.

Each element or tag can have attributes, which defines the behaviour of that element.

# Attributes

Attributes should always be applied with start tag.

The Attribute should always be applied with its name and value pair.

The Attributes name and values are case sensitive, and it is recommended by W3C that it should be written in Lowercase only.

You can add multiple attributes in one HTML element, but need to give space between two attributes.

**[Click here for practical](#)**

# **Text Formatting**

# What is Formatting

HTML Formatting is a process of formatting text for better look and feel. HTML provides us ability to format text without using CSS.

Categories:

- Physical tag: These tags are used to provide the visual appearance to the text.
- Logical tag: These tags are used to add some logical or semantic value to the text.

# Formatting Text

1. Bold Text: **<b>** and **<strong>**
2. Italic Text: *<i>* and *<em>*
3. Marked Formatting: **<mark>**
4. Underlined Text: **<u>** and  
**<ins>**
5. Strike Text: **<strike>** and **<del>**
6. Monospaced Font: **<tt>**
7. Superscript Text: **<sup>**
8. Subscript Text: **<sub>**
9. Larger Text: **<big>**
10. Smaller Text: **<small>**

[\*\*Click here for practical\*\*](#)

# HTML Phrase tag

The HTML phrase tags are special purpose tags, which defines the structural meaning of a block of text or semantics of text.

# Phrase Tags Ex.

1. Abbreviation: <abbr title = "">  
</abbr>
2. Marked: <mark>
3. Strong: <strong>
4. Emphasized: <em>
5. Definition: <dfn>
6. Quoting: <blockquote cite="">, <cite>
7. Short: <q>
8. Code: <code>
9. Keyboard: <kbd>
10. Address: <address>

[Click here for practical](#)

# HTML Comments

Syntax:

```
<!--
```

Write commented text  
here

```
-->
```

```
<html lang="en"> [event]
  > <head> ... </head>
  > <body>
    > <style> ... </style>
    > <!--
        The text in here will be invisible on the website! Here's
        another line of the comment. You can have as many lines as you
        want! 😊
      > -->
    > <div class="content">
      >   And here's that regular HTML content again.
    > </div>
  > </body>
</html>
```

# HTML HEAD

# Head

The HTML <head> element is used as a container for metadata (data about data). It is used between <html> tag and <body> tag.

The head of an HTML document is a part whose content is not displayed in the browser on page loading. It just contains metadata about the HTML document which specifies data about the HTML document.

[\*\*\*Click here for practical\*\*\*](#)

# Tags

Following is a list of tags used in metadata:

- <title>
- <link>
- <style>
- <script>
- <meta>
- <base>

[\*\*\*Click here for practical\*\*\*](#)

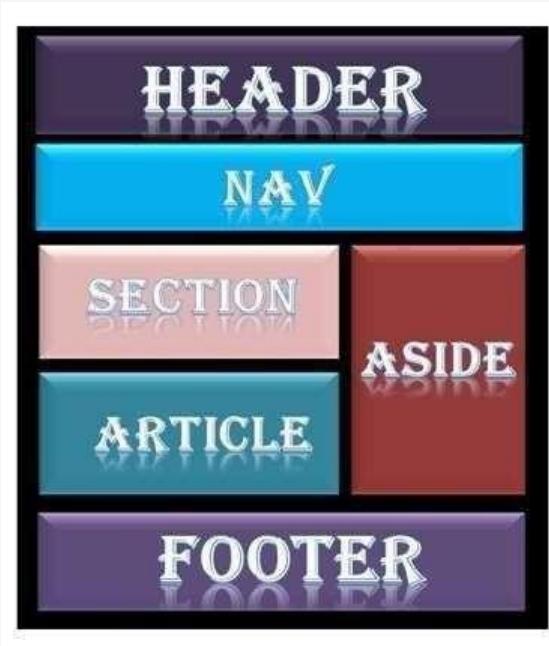
# <meta>

```
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta http-equiv="refresh" content="30">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

**[Click here for practical](#)**

# HTML Layout

# Layout



# Tags

**<header>**: It is used to define a header for a document or a section.

**<nav>**: It is used to define a container for navigation links

**<section>**: It is used to define a section in a document

**<article>**: It is used to define an independent self-contained article

# Tags

**<aside>**: It is used to define content aside from the content (like a sidebar)

**<footer>**: It is used to define a footer for a document or a section

**<details>**: It is used to define additional details

**<summary>**: It is used to define a heading for the **<details>** element

**[Click here for practical](#)**

# HTML Head

The HTML `<head>` element is a container for the following elements: `<title>`, `<style>`, `<meta>`, `<link>`, `<script>`, and `<base>`.

The `<head>` element is a container for metadata (data about data) and is placed between the `<html>` tag and the `<body>` tag.

HTML metadata is data about the HTML document. Metadata is not displayed on the page.

Metadata typically define the document title, character set, styles, scripts, and other meta information

[Click here for practical](#)

# HTML scripts

The `<script>` tag is used to embed a client-side script (JavaScript).

The `<script>` element either contains scripting statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

**[Click here for practical](#)**

# HTML Doctypes

The **DOCTYPE** declaration is the first line of code in an HTML or XHTML document. It informs the web browser about the version of HTML the page is written in, ensuring consistent rendering across different browsers.

**<!DOCTYPE html>**

[Click here for practical](#)

# **HTML Entities**

# Entities

HTML character entities are used as a replacement of reserved characters in HTML. You can also replace characters that are not present on your keyboard by entities.

For example: if you use less than (<) or greater than (>) symbols in your text, the browser can mix them with tags that's why character entities are used in HTML to display reserved characters.

[Click here for practical](#)

<	non-breaking space	&nbsp;
>	less than	&lt;
&	greater than	&gt;
&	ampersand	&amp;
"	double quotation mark	&quot;
'	single quotation mark (apostrophe)	&apos;
¢	cent	&cent;
£	pound	&pound;
¥	yen	&yen;
€	Euro	&euro;
©	copyright	&copy;
®	registered trademark	&reg;

# Symbols

There are many mathematical, technical and currency symbols which are not present on a normal keyboard. We have to use HTML entity names to add such symbols to an HTML page.

If there no entity name exists, you can use an entity number, a decimal, or hexadecimal reference.

[\*\*\*Click here for practical\*\*\*](#)

A	&#913;	&Alpha;	GREEK CAPITAL LETTER ALPHA
B	&#914;	&Beta;	GREEK CAPITAL LETTER BETA
Γ	&#915;	&Gamma;	GREEK CAPITAL LETTER GAMMA
Δ	&#916;	&Delta;	GREEK CAPITAL LETTER DELTA
E	&#917;	&Epsilon;	GREEK CAPITAL LETTER EPSILON
Z	&#918;	&Zeta;	GREEK CAPITAL LETTER ZETA
←	&#8592;	&larr;	LEFTWARDS ARROW
↑	&#8593;	&uarr;	UPWARDS ARROW
→	&#8594;	&rarr;	RIGHTWARDS ARROW
↓	&#8595;	&darr;	DOWNWARDS ARROW

# HTML Table

# HTML Table

HTML table tag is used to display data in tabular form (row \* column). There can be many columns in a row.

HTML tables are used to manage the layout of the page e.g. header section, navigation bar, body content, footer section etc. But it is recommended to use div tag over table to manage the layout of the page .

[\*\*Click here for practical\*\*](#)

[\*\*Click here for practical\*\*](#)

# HTML Table

## What is an HTML Table?

An HTML table is created using the `<table>` tag. Inside this tag, you use

- `<tr>` to define table rows,
- `<th>` for table headers, and
- `<td>` for table data cells

[Click here for practical](#)

# **HTML**

## **List**

# HTML Lists

HTML Lists are used to specify lists of information. All lists may contain one or more list elements. There are three different types of HTML lists:

- Ordered List or Numbered List (ol)
- Unordered List or Bulleted List (ul)
- Description List or Definition List (dl)

[\*\*\*Click here for practical\*\*\*](#)

# Ordered List or Numbered List

All the list items are marked with numbers by default.

**Syntax :**

```
<ol>
    <li>Aries</li>
    <li>Bingo</li>
</ol>
```

Types:

1, I, i, A, a

[\*\*Click here for practical\*\*](#)

# Unordered List or Bulleted List

All the list items are marked with bullets.

Syntax :

```
<ul>
  <li>Aries</li>
  .
</ul>
```

Types:

disc, circle, square, none

[Click here for practical](#)

# Description List or Definition List

Entries are listed like a dictionary or encyclopedia.

## Syntax :

```
<dl>
  <dt>HTML</dt>
  <dd>is a markup language</dd>
  <dt>Java</dt>
  <dd>is a programming language and platform</dd>
</dl>
```

[\*\*\*Click here for practical\*\*\*](#)

# **HTML File Path and Iframe Tag**

# File Paths

An HTML file path is used to describe the location of a file in a website folder. Used to link images, file, CSS file, JS file, video, etc.

Attributes:

1. src

Syntax :

1. 
2. 
3. 
4. 

[\*\*Click here for practical\*\*](#)

# Types of File Paths

## 1. Absolute File Paths:

Absolute file path specifies full

## 2. Relative File Paths

### Syntax :

```

```

[Click here for practical](#)

# IFrames

HTML Iframe is used to display a nested webpage (a webpage within a webpage). Used to embed Webpage or a YouTube video.

Syntax:<iframe src="URL"></iframe>

Attributes:

1. src
2. width
3. height
4. frameborder
5. allowfullscreen

[Click here for practical](#)

# HTML Form

# HTML Form

An HTML form is a section of a document which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing such as name, email address, password, phone number, etc.

HTML forms are required if you want to collect some data from of the site visitor.

Ex. Online Shopping Website

**[Click here for practical](#)**

# HTML Form Elements

```
<form>
<input>
<textarea>
<label>
<fieldset>
<legend>
```

[Click here for practical](#)

# Input Types

```
<input type="text" name="username">
<input type="password" name="password">
<input type="email" name="email">
<input type="radio" name="password">
<input type="checkbox" id="cricket" name="cricket" value="cricket"/>
<input type="submit" value="submit">
<input type="button" value="button">
```

[\*\*Click here for practical\*\*](#)

# Form Inputs Types

# Input Types

text:	Defines a one-line text input field
password:	Defines a one-line password input
submit :	field
reset:	Defines a submit button to submit the form to server
radio:	Defines a reset button to reset all values in the form.
checkbox:	Defines a radio button which allows select one option.
button:	Defines checkboxes which allow select multiple options form.
file:	Defines a simple push button, which can be programmed to perform a task on an event.
image:	Defines to select the file from device storage. Defines a graphical submit button.

# HTML5 Added Input Types

- color: Defines an input field with a specific color.
- date: Defines an input field for selection of date.
  
- datetime-local: Defines an input field for entering a date without time zone.
- email: Defines an input field for entering an email address.
- month: Defines a control with month and year, without time zone.
- number: Defines an input field to enter a number.
- url: Defines a field for entering URL
- week: Defines a field to enter the date with week-year, without time zone.
  
- search: Defines a single line text field for entering a search string.
- tel: Defines a single line text field for entering a telephone number.

# HTML Form Attributes

# Form Attributes

**action:** The action attribute value defines the web page where information proceed. It can be .php, .jsp, .asp, etc. or any URL

## **method:** Defines the HTTP method

- post: We can use the post value of method attribute when we want to process the sensitive data as it does not display the submitted data in URL.
- get: The get value of method attribute is default value while submitting the form. But this is not secure as it displays data in URL after submitting the form.  
**target:** Where to open the response after submitting the form
  - \_self: The response will display in current page only.
  - \_blank: Load the response in a new page.

**autocomplete:** Enables an input field to complete automatically.

**enctype:** Defines the encoding type

- application/x-www-form-urlencoded: Default
- multipart/form-data: It does not encode any character. It is used when our form contains file-upload controls.
- text/plain (HTML5): only space are encoded into + symbol

**novalidate:** Does not perform any type of validation and submit the form.

# Input Attributes

1. name
2. value
3. required
4. autofocus
5. placeholder
6. disabled
7. size
8. form

[\*\*\*Click here for practical\*\*\*](#)

# Class and Id

# Class

The HTML class attribute is used to specify a single or multiple class names for an HTML element.

The class name can be used by CSS and JavaScript to do some tasks for HTML elements.

You can use this class in CSS with a specific class, write a period (.) character, followed by the name of the class for selecting elements.

[Click here for practical](#)

# ID

The id attribute is used to specify the unique ID for an element of the HTML document.

It allocates the unique identifier which is used by the CSS and the JavaScript for performing certain tasks.

**[Click here for practical](#)**

# **HTML Form and Keyboard Event Attributes**

# What is an Event?

When a browser reacts on user action, then it is called as an event.

# Form Event Attributes

<b>Attribute</b>	<b>Description</b>
onblur	Executed the script when form element loses the focus. Executed
onchange	the script when the value of the element is changed.
onfocus	Trigger an event when the element gets focused. Executed the
oninput	script when the user enters input to the element.
oninvalid	Executed the script when the element does not satisfy its predefined constraints.
onreset	Triggers the event when user reset the form element values. Triggers
onsubmit	the event when a form is submitted.

# **HTML Keyboard and Mouse Event Attributes**

# Keyboard Event Attributes

**Attribute**

onkeydown

onkeypress

onkeyup

**Description**

Triggers the event when the user presses down a key on the keyboard.

Trigger the event when the user presses the key which displays some character.

Trigger the event when the user releases the currently pressed key.

# Mouse Event Attributes

Attribute	Description
onclick	Trigger the event when the mouse clicks on the element.
ondblclick	Trigger the event when mouse double-click occurs on the element.
onmousedown	Trigger the event when the mouse button is pressed on the Element.
onmousemove	Trigger the event when the mouse pointer moves over the element.
onmouseout	Trigger the event when the mouse moves outside the element.
onmouseover	Trigger the event when the mouse moves onto the element.
onmouseup	

# **Module - 6**

# **HTML5**

# HTML Audio Tag

# HTML Audio Tag

HTML audio tag is used to define sounds such as music and other audio clips. Currently there are three supported file format for HTML 5 audio tag.

- mp3
- wav
- ogg

HTML5 supports <video> and <audio> controls. The Flash, Silverlight and similar technologies are used to play the multimedia items.

[\*\*\*Click here for practical\*\*\*](#)

# HTML Video Tag

# HTML Video Tag

HTML 5 supports <video> tag also. The HTML video tag is used for streaming video files such as a movie clip, song clip on the web page.

Currently, there are three video formats supported for HTML video tag:

- mp4
- webM
- Ogg

**[Click here for practical](#)**

# Video Attributes

<b>Attribute</b>	<b>Description</b>
controls	It defines the video controls which is displayed with play/pause buttons.
height	It is used to set the height of the video player.
width	It is used to set the width of the video player.
poster	It specifies the image which is displayed on the screen when the video is not played.
autoplay	It specifies that the video will start playing as soon as it is ready.
loop	It specifies that the video file will start over again, every time when it is completed.
muted	It is used to mute the video output.
preload	
src	

# **HTML SVG**

# What is SVG?

The HTML SVG is an acronym which stands for Scalable Vector Graphics.

HTML SVG is a modularized language which is used to describe graphics in XML. It describes two-dimensional vector and mixed vector/raster graphics in XML.

SVG is mostly used for vector type diagrams like pie charts, 2-Dimensional graphs in an X,Y coordinate system etc.

**[Click here for practical](#)**

# Ellipse

The cx attribute defines the x coordinate of the center of the ellipse  
The cy attribute defines the y coordinate of the center of the ellipse  
The rx attribute defines the horizontal radius  
The ry attribute defines the vertical radius

**[Click here for practical](#)**

# Rectangle

cx, cy and r are attributes of circle tag. These attributes can't be used with svg other tag.

**[Click here for practical](#)**

# **HTML SVG Line, Polygon, Polyline**

# SVG Line

[Click here for practical](#)

# SVG Polygon

The `<polygon>` element is used to create a graphic that contains at least three sides.

Polygons are made of straight lines, and the shape is "closed" (all the lines connect up).

[Click here for practical](#)

# SVG Polyline

The <polyline> element is used to create any shape that consists of only straight lines (that is connected at several points)

[\*\*\*Click here for practical\*\*\*](#)

# SVG Path, Text and Stroking

# SVG Path

The <path> element is used to define a path.

The following commands are available for path

data: M = moveto L = lineto

H = horizontal lineto V = vertical

lineto

Z = closepath C = curveto

S = smooth curveto

[Click here for practical](#)

# SVG Text

The <text> element is used to define a text.

**[Click here for practical](#)**

# SVG Stroke

SVG offers a wide range of stroke properties. In this chapter we will look at the following:

- stroke
- stroke-width
- stroke-linecap
- stroke-dasharray

[Click here for practical](#)

# SVG Gradient

# SVG Gradient

A gradient is a smooth transition from one color to another. In addition, several color transitions can be applied to the same element.

There are two main types of gradients in SVG:

- Linear
- Radial

# Linear Gradient

The `<linearGradient>` element is used to define a linear gradient.

The `<linearGradient>` element must be nested within a `<defs>` tag. The `<defs>` tag is short for definitions and contains definition of special elements (such as gradients).

Linear gradients can be defined as horizontal, vertical or angular gradients:

Horizontal gradients are created when  $y_1$  and  $y_2$  are equal and  $x_1$  and  $x_2$  differ

Vertical gradients are created when  $x_1$  and  $x_2$  are equal and  $y_1$  and  $y_2$  differ

Angular gradients are created when  $x_1$  and  $x_2$  differ and  $y_1$  and  $y_2$  differ

[Click here for practical](#)

# Radial Gradient

The <radialGradient> element is used to define a radial gradient.

**[Click here for practical](#)**

# HTML Canvas

# HTML Canvas

The HTML canvas element provides HTML a bitmapped surface to work with. It is used to draw graphics on the web page.

The HTML 5 <canvas> tag is used to draw graphics using scripting language like JavaScript.

The <canvas> element is only a container for graphics, you must need a scripting language to draw the graphics. The <canvas> element allows for dynamic and scriptable rendering of 2D shapes and bitmap images.

# Create a Canvas syntax

```
<canvas id="myCanvas1" width="300" height="100" style="border:2px solid;">  
Your browser does not support the HTML5 canvas tag.  
</canvas>
```

[\*\*\*Click here for practical\*\*\*](#)

# Canvas Tag with Javascript

```
<script>  
    var c =  
        document.getElementById("myCanvas"); var ctx =  
        c.getContext("2d");  
        ctx.fillStyle = "#FF0000";  
        ctx.fillRect(0,0,200,100);  
</script>
```

[Click here for practical](#)

# Line on Canvas

[Click here for practical](#)

# Circle on Canvas

[Click here for practical](#)

# Text on Canvas

[Click here for practical](#)

# URL Encode

# What is URL

URL stands for Uniform Resource Locator. It is actually a web address. A URL can contain words i.e. (learnvern.com) or an Internet Protocol (IP) address i.e. 195.201.68.81. But most of the user use URL in the form of words because it is easy to remember than numbers.

## Syntax:

scheme://prefix.domain:port/path/filename

# What is URL

**scheme** is used to define the type of Internet service (most common is http or https).

**prefix domain** is used to define a domain prefix (default for http is

**port** www). is used to define the Internet domain name (like lernvern.com).

**path** is used to define the port number at the host (default for http is 80).

**filename** is used to define a path at the server (If omitted: the root directory of the site).

is used to define the name of a document or resource.

# URL Encode

URL encoding is used to convert non-ASCII characters into a format that can be used over the Internet because a URL is sent over the Internet by using the ASCII character-set only. If a URL contains characters outside the ASCII set, the URL has to be converted.

In URL encoding, the non-ASCII characters are replaced with a "%" followed by hexadecimal digits.

URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign, or %20.

Ex. © will be replaced with %C2%A9

# Difference between HTML and XHTML

# What is XHTML?

XHTML is a stricter, more XML-based version of HTML.

- XHTML stands for EXtensible HyperText Markup Language
- XHTML is a stricter, more XML-based version of HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers

XML is a markup language where all documents must be marked up correctly (be "well-formed").

# Difference between HTML and XHTML

1. <!DOCTYPE> is mandatory:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

2. The xmlns attribute in <html> is mandatory

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

3. <html>, <head>, <title>, and <body> are mandatory

4. Elements must always be properly nested

5. Elements must always be closed

# Difference between HTML and XHTML

6. Elements must always be in lowercase

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

7. Attribute names must always be in lowercase

```
<a href="https://learnvern.com">Visit our HTML tutorial</a>
```

8. Attribute values must always be quoted

```
<a href="https://learnvern.com">Visit our HTML tutorial</a>
```

9. Attribute minimization is forbidden

```
<input type="checkbox" name="vehicle" value="car" checked="checked" />
```

# **API's in HTML**

# What is an API

## **What is Web API?**

- API stands for Application Programming Interface.
- A Web API is an application programming interface for the Web.
- A Browser API can extend the functionality of a web browser.
- A Server API can extend the functionality of a web server.

API's help to access data.

# What is an API

The HTML Geolocation API is used to locate a user's position.

The HTML Geolocation API is used to get the geographical position of a user.

Since this can compromise privacy, the position is not available unless the user approves it.

# HTML Drag and Drop API

# Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

# **HTML Web Storage API**

# Web Storage

With web storage, web applications can store data locally within the user's browser.

Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.

# Types of Web Storage

HTML web storage provides two objects for storing data on the client:

`window.localStorage` - stores data with no expiration date

`window.sessionStorage` - stores data for one session (data is lost when the browser tab is closed)

# **Module - 5**

# **CSS and CSS3**

# CSS

- ✓ Introduction to CSS
- ✓ How to insert CSS
- ✓ Inline
- ✓ Internal
- ✓ External
- ✓ Comments in CSS
- ✓ CSS Selectors
- ✓ CSS Pseudo Selector
- ✓ CSS Specificity
- ✓ CSS Text
- ✓ CSS Fonts
- ✓ CSS Background and Border Properties

- ✓ CSS Colors
- ✓ CSS Important
- ✓ Line height, Padding and Margin
- ✓ CSS Filters
- ✓ CSS Images
- ✓ CSS Overflow
- ✓ CSS Position Property
- ✓ CSS Vertical Align, White Space and Word Wrap

- ✓ CSS Width and Height
- ✓ CSS Box-shadow and Text-shadow
- ✓ CSS Text Transform
- ✓ CSS Visibility
- ✓ CSS Icons
- ✓ Justify, Text Decoration and Text-Align

- ✓ CSS List
- ✓ CSS Selectors
- ✓ CSS Specificity
- ✓ Text Indent and Text Stroke
- ✓ CSS Calc ()
- ✓ CSS Print Properties
- ✓ CSS Columns
- ✓ CSS hyphens
- ✓ CSS Positions
- ✓ CSS Transform and Resize
- ✓ Transition Delay
- ✓ CSS Buttons
- ✓ CSS Positioning

# Advance CSS

- ✓ CSS Animation
- ✓ @keyframe
- ✓ CSS Pseudo elements
- ✓ CSS Gradient
- ✓ CSS z-index
- ✓ CSS Combinators
- ✓ Masking
- ✓ CSS Media Query
- ✓ 2D and 3D Transforms
- ✓ CSS Flex
- ✓ CSS Grid

# What is CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

# What is CSS

- CSS stands for Cascading Style Sheet.
- CSS is used to design HTML tags.
- CSS is a widely used language on the web.
- HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

[\*\*\*Click here for practical\*\*\*](#)

# Advantages of CSS

- Solves a big problem
- Saves a lot of time
- Provide more attributes

# CSS Syntax

declaration  
↓  
selector → p{color: black}  
↑↑  
property      value

[Click here for practical](#)

Selector{Property1: value1; Property2:  
value2; .....;}

# CSS Syntax

**Selector:** Selector indicates the HTML element you want to style.  
It could be any tag like <h1>, <title> etc.

**Declaration Block:** The declaration block can contain one or more declarations separated by a semicolon.

**Property:** A Property is a type of attribute of HTML element. It could be color, border etc.

**Value:** Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

# How to add CSS inline css

```
<h1 style="color:red">Write Your First CSS Example</h1>
```

This our inline css use

[\*\*\*Click here for practical\*\*\*](#)

# How to add CSS internal css syntax:

```
<head>
  <style>
    h1{
      color:white;
      padding:5px;
    }
  </style>
</head>
```

This is our internal css [Click here for practical](#)

# How to add CSS external css

**Syntax :**

```
<head>
  <link rel="stylesheet" href="style.css">
</head>
```

This is our External css  
File name save as style.css

**[Click here for practical](#)**

# CSS Selector

# CSS Selector

CSS selectors are used to select the content you want to style.

Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

Types of CSS Selector:

- CSS Universal Selector
- CSS Element Selector
- CSS Id Selector
- CSS Class Selector
- CSS Group Selector
- Css Descendant Selector

# CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

## Syntax :

```
<style>
*
{
    color: green;
    font-size: 20px;
}
```

[Click here for practical](#)

# CSS Element Selector

The element selector selects the HTML element by name.

## Syntax :

```
<style> p{  
    text-align: center;  
    color: blue;  
}  
</style>
```

<p>This style will be applied on every paragraph.</p>

[Click here for practical](#)

# CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

## syntax

```
#para1 {  
    text-align:  
    center; color:  
    blue;
```

} [Click here for practical](#)

# CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

**Syntax :**

```
<style>
p.center {
    text-align:
    center; color:
    blue;
}
</style>
<p class="center">This paragraph is blue and center-aligned.</p>
```

**[Click here for practical](#)**

# CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.**Ex.**

```
h1, h2, p  
{  
    text-align: center; color:  
    blue;  
}
```

[\*\*\*Click here for practical\*\*\*](#)

# CSS Combinators

# CSS Combinators

CSS Combinators clarifies the relationship between two selectors, whereas the selectors in CSS are used to select the content for styling.

There can be more than one simple selector in a CSS selector, and between these selectors, we can include a combinator. Combinators combine the selectors to provide them a useful relationship and the position of content in the document.

There are four types of combinators in CSS that are listed as follows:

1. General sibling selector (~)
2. Adjacent sibling selector (+)
3. Child selector (>)
4. Descendant selector (space)

# General Sibling Selector

elements that follow the elements of first selector, and both of them are the children of the same parent. It can be used for selecting the group of elements that share the common parent element.

It is useful when we have to select the siblings of an element even if they are not adjacent directly.

## Syntax:

```
element ~ element {  
    /*style properties*/  
}
```

[Click here for practical](#)

# Adjacent Sibling Selector (+)

It uses the plus (+) sign as the separator between the elements. It matches the second element only when the element immediately follows the first element, and both of them are the children of the same parent. This sibling selector selects the adjacent element, or we can say that the element which is next to the specified tag. It only selects the element which is just next to the specified first element.

## Syntax:

```
element + element  
{  
    /*style properties*/  
}
```

[Click here for practical](#)

# Child Selector (>)

It uses the greater than (>) sign as the separator between the elements. It selects the direct descendant of the parent. This combinator only matches the elements that are the immediate child in the document tree. It is stricter as compared to the descendant selector because it selects the second selector only when the first selector is its parent.

## Syntax:

```
element > element {  
    /*style properties*/  
}
```

[Click here for practical](#)

# Descendant Selector (space)

It combines two selectors in which the first selector represents an ancestor (parent, parent's parent, etc.), and the second selector represents descendants. The elements matched by the second selector are selected if they have an ancestor element that matches the first selector.

## Syntax:

```
element element {  
    /*style properties*/  
}
```

[Click here for practical](#)

# CSS Psudo Selector

# CSS pseudo-classes

A pseudo-class can be defined as a keyword which is combined to a selector that defines the special state of the selected elements. It is added to the selector for adding an effect to the existing elements based on their states. For example, The ":hover" is used for adding special effects to an element when the user moves the cursor over the element.

## Syntax:

selector: pseudo-class

{ property: value;

} [\*\*\*Click here for practical\*\*\*](#)

# :hover

This pseudo-class adds a special style to an element when the user moves the cursor over it.

**Ex.**

```
h1:hover{  
    color: red;  
}
```

[Click here for practical](#)

# :active

It applies when the elements are clicked or activated. It selects the activated element.

**Ex.**

```
a:active{  
    color: yellow;  
}
```

[Click here for practical](#)

# :visited

It selects the visited links and adds special styles to them.

**Ex.**

```
a:visited{  
    color: red;  
}
```

**Click here for practical**

# :lang

It is helpful in documents that require multiple languages.

**Ex.**

```
p:lang(fr)
{
    font-family:Verdana; color:blue;
```

```
}
```

<p lang="fr">With :lang pseudo class with the value fr</p>

[\*\*Click here for practical\*\*](#)

# :focus

It selects the elements that are currently focused on by the user.

**Ex.**

```
input:focus{  
    border:5px solid lightblue;  
    box-shadow:10px 10px 10px  
    black; color: blue; width:300px;
```

}

[\*\*\*Click here for practical\*\*\*](#)

# :first-child

It matches a particular element, which is the first child of another element and adds a special effect to the corresponding element.

**Ex.**

```
h1:first-child {  
    text-indent: 200px;  
    color:blue;  
}
```

[\*\*Click here for practical\*\*](#)

# :nth-child(n)

This selector is used for matching the elements based on their position regardless of the type of its parent. The n can either be a keyword, formula, or a number. It is used to match the elements based on their position within a group of siblings. It matches each element, which is the nth-child.

**Ex.**

```
p:nth-child(2n+1) {  
    background:  
        yellow; color:  
        black;  
    font-size:30px;
```

} [Click here for practical](#)

# CSS Pseudo Element

# CSS Pseudo-elements

A pseudo-class can be defined as a keyword which is combined to a selector that defines the special state of the selected elements. Unlike the pseudo-classes, the pseudo-elements are used to style the specific part of an element, whereas the pseudo-classes are used to style the element.

## Syntax:

```
selector::pseudo-element  
{ property: value;  
}
```

We have used the double colon notation (::pseudo-element) in the syntax. In CSS3, the double colon replaced the single colon notation for pseudo-elements.

# Psuedo Elements

Psuedo Element	Description
::first-letter (:first-letter)	It selects the first letter of the text.
::first-line (:first-line)	It styles the first line of the text.
::before (:before)	It is used to add something before the element's content.
::after (:after)	It is used to add something after the element's content.
::selection	It is used to select the area of an element that is selected by the user.

# ::first-letter

it affects the first letter of the text. It can be applied only to block-level elements. Instead of supporting all CSS properties, it supports some of the CSS properties that are given below.

- Color properties (such as color)
- Font properties (such as font-style, font-family, font-size, font-color, and many more).
- Margin properties (such as margin-top, margin-right, margin-bottom, and margin-left).
- Border properties (like border-top, border-right, border-bottom, border-left, border-color, border-width, and many more).

[\*\*Click here for practical\*\*](#)

# ::first-line

It is similar to the ::first-letter pseudo-element, but it affects the entire line. It adds the special effects to the first line of the text. It supports the following CSS properties:

- Color properties (such as color)
- Font properties (such as font-style, font-family, font-size, font-color, and many more).
- Background properties (such as background-color, background-repeat, background-image, and background-position).
- Other properties are word-spacing, letter-spacing, line-height, vertical-align, text-transform, text-decoration.

[\*\*\*Click here for practical\*\*\*](#)

# ::before

It allows us to add something before the element's content. It is used to add something before the specific part of an element. Generally, it is used with the content property.

**Ex:**

```
h1::before {  
    content: "Hello World.";  
}
```

[\*\*Click here for practical\*\*](#)

# ::after

It works similar to ::before pseudo-element, but it inserts the content after the content of the element. It is used to add something after the specific part of an element. Generally, it is used with the content property.

**Ex:**

```
h1::after {  
    content: "Welcome to the LearnVern";  
}
```

[\*\*Click here for practical\*\*](#)

# ::selection

It is used to style the part of an element that is selected by the user. We can use the following CSS properties with it:

- color.
- background-color.
- Other properties include cursor, outline, etc.

**Ex.**

```
h1::selection  
{ color: red;  
}
```

[\*\*\*Click here for practical\*\*\*](#)

# How to add CSS

# How to add CSS

There are three ways to insert CSS in HTML documents.

1. Inline CSS
2. Internal CSS
3. External CSS

# Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

**[Click here for practical](#)**

# Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

# Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in `<head>` section of the HTML page inside the `<style>` tag.

[\*\*\*Click here for practical\*\*\*](#)

# External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Ex.

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

[\*\*Click here for practical\*\*](#)

# CSS Comments

# Need of the comments

CSS comments are generally written to explain your code. It is very helpful for the users who reads your code so that they can easily understand the code.

Comments are ignored by browsers.

**[Click here for practical](#)**

# Specificity

# Specificity

Specificity is the way which helps the browsers to decide which property value is most relevant for the element. It determines which style declaration is applied to an element.

- The CSS specificity is important only when various selectors are affecting the same element. In this case, the browser needs a way to identify the style to be applied to the matching element, and CSS specificity is the way of doing it.

# Specificity

- When two or more selectors have equal specificity value, then the latest one considers.
- Universal selectors (\*) and the inherited values have lower specificity, i.e., 0 specificity.
- The style property has a greater specificity value compare to the selectors (except the !important in the stylesheet selector).
- The !important alter the selector specificity. When two selectors have equal specificity, then the selector having !important

# hierarchy

**Inline styles:** It is directly attached to the element which is to be styled. For example: `<p style="color: red;">`. It has the highest priority.

**IDs:** It is a unique identifier for the elements of a page that has the second- highest priority. For example: `#para`.

**Classes, attributes, and pseudo-classes:** It includes classes, attributes, and pseudo- classes (like `:focus`, `:hover`, etc.).

**Elements and pseudo-elements:** It includes the name of elements (`div`, `h1`) and pseudo-elements (like `:after` and `:before`). They have the lowest priority.

# Rules

- The specificity of ID selectors is higher than attribute selectors
- In equal specificity, the latest rule will count
- The specificity of class selector is greater than the element selectors

[\*\*\*Click here for practical\*\*\*](#)

# **CSS Text Transform**

# CSS Text Transform

This CSS property allows us to change the case of the text. It is used to control the text capitalization. This CSS property can be used to make the appearance of text in all-lowercase or all-uppercase or can convert the first character of each word to uppercase.

## **Syntax:**

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

[Click here for practical](#)

# 1. capitalize

It transforms the first character of each word to uppercase. It will not capitalize the first letter after the number. It only affects the first letters of the words instead of changing the rest of the letters in the word.

If we apply the capitalize property on a word that already has capital letters, then the letters of that word will not switch to lowercase.

## Syntax:

text-transform: capitalize;

[Click here for practical](#)

## 2. uppercase

As its name implies, it transforms all characters of the word into uppercase.

### Syntax:

text-transform: uppercase;

[Click here for practical](#)

## 3. lowercase

It transforms all characters of the word into lowercase.

### Syntax:

text-transform: lowercase;

**[Click here for practical](#)**

# 4. none

It is the default value that has no capitalization. It renders the text as it is.

## Syntax:

text-transform: none;

**[Click here for practical](#)**

# CSS text-overflow

This property specifies the representation of overflowed text, which is not visible to the user. It signals the user about the content that is not visible. This property helps us to decide whether the text should be clipped, show some dots (ellipsis), or display a custom string.

This property does not work on its own. We have to use white-space: nowrap; and overflow: hidden; with this property

## Syntax:

text-overflow: clip | ellipsis | string

# 1. clip

**clip:** It is the default value that clips the overflowed text. It truncates the text at the limit of the content area, so that it can truncate the text in the middle of the character.

## Syntax:

text-overflow: clip;

[Click here for practical](#)

## 2. ellipsis

**ellipsis:** This value displays an ellipsis (?) or three dots to show the clipped text. It is displayed within the area, decreasing the amount of text.

### Syntax:

text-overflow: ellipsis;

[Click here for practical](#)

# CSS text-orientation

This CSS property specifies the orientation of characters in the line of content. It only applies to the vertical mode of content. This property does not affect elements with horizontal writing mode.

It helps us to control the display of languages that use a vertical script. This property has five **values: mixed, sideways, upright, sideways-right, and use-glyph-orientation**. Its default value is mixed. In latest browsers only mixed and upright are the working property values.

This property depends upon the writing-mode property. It works only when the writing-mode is not set to horizontal-tb.

## Syntax:

text-orientation: mixed | upright

# **Text Indent and Text Stroke**

# Text Indent

This CSS property sets the indentation of the first line in a block of text. It specifies the amount of horizontal space that puts before the lines of text.

It allows the negative values, and if any negative value is defined, then the indentation of the first line will be towards left.

## Syntax:

`text-indent: length`

# Values

**length:** This value sets the fix indentation with the units cm, pt, em, px, and others. Its default value is 0. It allows negative values. The indentation of the first line is on the left when its value is negative.

**percentage:** It specifies the amount in space in the percentage of the width of the containing block.

[Click here for practical](#)

# Text Stroke

This CSS property adds a stroke to the text and also provides decoration options for them. It defines the color and width of strokes for text characters.

This CSS property is the shorthand of the following two properties:

**text-stroke-width:** It describes the thickness of the stroke effect and takes the unit value.

**text-stroke-color:** It takes the value of a color.

**-webkit-text-fill-color:** It fills color inside the text.

The text-stroke can only be used with the -webkit- prefix.

# CSS Fonts

# CSS Fonts

CSS Font property is used to control the look of texts. By the use of CSS font property you can change the text size, color, style and more.

- CSS Font color
- CSS Font family
- CSS Font size
- CSS Font style
- CSS Font variant
- CSS Font weight

# CSS Font Color

It is used to change the color of the text. There are three different formats to define a color:

By a color name By hexadecimal value By RGB

**Ex.**

[Click here for practical](#)

# CSS Font Family

Generic family: It includes Serif, Sans-serif, and Monospace.

Font family: It specifies the font family name like Arial, New Times Roman etc.

Serif: Serif fonts include small lines at the end of characters. Example of serif:  
Times new roman, Georgia etc.

Sans-serif: A sans-serif font doesn't include the small lines at the end of  
characters. Example of Sans-serif: Arial, Verdana etc.

**[Click here for practical](#)**

# CSS Font Size

CSS font size property is used to change the size of the font.

font-size:xx-small;  
font-size:x-small;  
font-size:small; font-size:medium; font-size:large;  
font-size:x-large;

font-size:xx-large; font-size:smaller;  
font-size:larger; font-size:200%;  
font-size:20px;

[Click here for practical](#)

# CSS Font Size

## **1. Absolute-size:**

It is used to set the text to a definite size. Using absolute-size, it is not possible to change the size of the text in all browsers. It is advantageous when we know the physical size of the output.

Ex. Font-size with em

## **2. Relative-size:**

It is used to set the size of the text relative to its neighboring elements.

With relative-size, it is possible to change the size of the text in browsers.

# CSS Font Size

### **3. Responsive font size:**

We can set the size of the text by using a vw unit, which stands for the 'viewport width'. The viewport is the size of the browser window.

**Ex.** 1vw = 1% of viewport width.

### **4. Font-size with the length property:**

It is used to set the size of the font in length. The length can be in cm, px, pt, etc.

**Ex.** font-size: 5cm;

# CSS Font Style

CSS Font style property defines what type of font you want to display. It may be italic, oblique, or normal.

**Ex.**

```
h2 { font-style: italic;  
} h3 { font-style:  
oblique;}  
  
h4 { font-style: normal; }
```

[Click here for practical](#)

# CSS Font Variant

CSS font variant property specifies how to set font variant of an element. It may be normal and small-caps.

**[Click here for practical](#)**

# CSS Font Weight

CSS font weight property defines the weight of the font and specify that how bold a font is. The possible values of font weight may be normal, bold, bolder, lighter or number (100, 200..... upto 900).

[Click here for practical](#)

# CSS Font Stretch

The font-stretch property in CSS allows us to select a normal, expanded, or condensed face from the font's family. This property sets the text wider or narrower compare to the default width of the font. It will not work on any font but only works on the font-family that has a width-variant face.

[\*\*\*Click here for practical\*\*\*](#)

# CSS Background

# CSS Background

CSS background property is used to define the background effects on element. There are 5 CSS background properties that affects the HTML elements:

1. background-color
2. background-image
3. background-repeat
4. background-attachment
5. background-position

# 1. CSS background-color

The background-color property is used to specify the background color of the element.

**Ex.**

[\*Click here for practical\*](#)

## 2. CSS background-image

The background-image property is used to set an image as a background of an element. By default the image covers the entire element

**Ex.**

[Click here for practical](#)

# 3. CSS background-repeat

By default, the background-image property repeats the background image horizontally and vertically. Some images are repeated only horizontally or vertically.

**Ex.**

[Click here for practical](#)

## 4. CSS background-attachment

The background-attachment property is used to specify if the background image is fixed or scroll with the rest of the page in browser window. If you set fixed the background image then the image will not move during scrolling in the browser.

**Ex.**

[\*\*Click here for practical\*\*](#)

# 5. CSS background-position

The background-position property is used to define the initial position of the background image. By default, the background image is placed on the top-left of the webpage. You can set the positions as:center,top,bottom,left,right

**Ex.**

[Click here for practical](#)

# CSS Border Property

# CSS Border

The CSS border is a shorthand property used to set the border on an element.

The CSS border properties are used to specify the style, color and size of the border of an element. The CSS border properties are given below

- border-style
- border-color
- border-width
- border-radius

# 1. CSS Border Style

The Border style property is used to specify the border type which you want to display on the web page.

Ex.

[\*Click here for practical\*](#)

## 2. CSS Border Width

The border-width property is used to set the border's width. It is set in pixels. You can also use the one of the three pre-defined values, thin, medium or thick to set the width of the border.

**Ex.**

[\*Click here for practical\*](#)

# 3. CSS Border Color

There are three methods to set the color of the border. Name: It specifies the color name. For example: "red".

RGB: It specifies the RGB value of the color. For example: "rgb(255,0,0)". Hex: It specifies the hex value of the color. For example: "#ff0000".

**Ex.**

[Click here for practical](#)

# 4. CSS Border Radius

This CSS property sets the rounded borders and provides the rounded corners around an element, tags, or div. It defines the radius of the corners of an element. The values of this property can be defined in percentage or length units. It is shorthand for border top-left-radius, border-top-right-radius, border-bottom-right-radius and border-bottom-left-radius.

**Ex.**

[Click here for practical](#)

# 5. CSS Border Collapse

This CSS property is used to set the border of the table cells and specifies whether the table cells share the separate or common border.

This property has two main values that are separate and collapse. When it is set to the value separate, the distance between the cells can be defined using the border-spacing property. When the border-collapse is set to the value collapse, then the inset value of border-style property behaves like groove, and the outset value behaves like ridge.

**Ex.**

[Click here for practical](#)

# 6. CSS Border Spacing

This CSS property is used to set the distance between the borders of the adjacent cells in the table. It applies only when the border-collapse property is set to separate. There will not be any space between the borders if the border-collapse is set to collapse.

When only one value is specified, then it sets both horizontal and vertical spacing. When we use the two-value syntax, then the first one is used to set the horizontal spacing (i.e., the space between the adjacent columns), and the second value sets the vertical spacing (i.e., the space between the adjacent rows). **Ex.**

[Click here for practical](#)

# 7. CSS Outline

CSS outline is just like CSS border property. It facilitates you to draw an extra border around an element to get visual attention.

Outline offset: The outline offset is used to create a distance between outline and border.

**Ex.**

[\*Click here for practical\*](#)

# 7. CSS Outline...

Outline offset: The outline offset is used to create a distance between outline and border.

[Click here for practical](#)

# 8. CSS Border Image

This CSS property defines an image to be used as the element's border. It draws an image outside the element and replaces the element's border with the corresponding image. It is an interesting task to replace the border of an element with the image.

It is the shorthand property for border-image-source, border-image-slice, border- image-width, border-image-outset, and border-image-repeat.

**Ex.**

[\*\*\*Click here for practical\*\*\*](#)

# 8. CSS Border

Value	Description
border-image-source	It specifies the source of the border-image.
border-image-slice	It is used to divide or slice the image, which is specified by the border- image-source property.
border-image-width	It sets the width of the border-image.
border-image-outset	It sets the amount of space by which the border image is set out from its border box.
border-image-repeat	It controls the repetition of the image to fill the area of the border. stretch  repeat  round  space

# CSS List

# Lists

There are various CSS properties that can be used to control lists. Lists can be classified as ordered lists and unordered lists. In ordered lists, marking of the list items is with alphabet and numbers, whereas in unordered lists, the list items are marked using bullets.

We can style the lists using CSS. CSS list properties allow us to:

# List

- S** Set the distance between the text and the marker in the list.
2. Specify an image for the marker instead of using the number or bullet point.
  3. Control the marker appearance and shape.
  4. Place the marker outside or inside the box that contains the list items.
  5. Set the background colors to list items and lists.

# Properties

**list-style-type:** This property is responsible for controlling the appearance and shape of the marker.

**list-style-image:** It sets an image for the marker instead of the number or a bullet point.

**list-style-position:** It specifies the position of the marker.

**list-style:** It is the shorthand property of the above properties.

**marker-offset:** It is used to specify the distance between the text and the marker. It is unsupported in IE6 or Netscape 7.

# list-style-type

It allows us to change the default list type of marker to any other type such as square, circle, roman numerals, Latin letters, and many more. By default, the ordered list items are numbered with Arabic numerals (1, 2, 3, etc.), and the items in an unordered list are marked with round bullets (•).

If we set its value to none, it will remove the markers/bullets.

### Possible values:

1. decimal
2. lower-alpha
3. lower-roman
4. circle
5. square
6. Disc

**[Click here for practical](#)**

# list-style-position

It represents whether the appearing of the marker is inside or outside of the box containing the bullet points. It includes two values.

inside: It means that the bullet points will be in the list item. In this, if the text goes on the second line, then the text will be wrap under the marker.

outside: It represents that the bullet points will be outside the list item. It is the default value.

[Click here for practical](#)

# list-style-image

It specifies an image as the marker. Using this property, we can set the image bullets. Its syntax is similar to the background-image property. If it does not find the corresponding image, the default bullets will be used.

## Syntax:

```
list-style-image: url(img.png);
```

[Click here for practical](#)

# list-style

It is the shorthand property that is used to set all list properties in one expression. The order of the values of this property is type, position, and image. But if any property value is missing, then the default value will be inserted.

## Syntax:

list-style: lower-alpha inside url(img.png);

[\*\*Click here for practical\*\*](#)

# CSS Display Property

# CSS Display

CSS display is the most important property of CSS which is used to control the layout of the element. It specifies how the element is displayed.

CSS Display Value:

- display: inline;
- display: inline-block;
- display: block;
- display: none;
- flex

# 1. Inline

The inline element takes the required width only. It doesn't force the line break so the flow of text doesn't break in inline example.

The inline elements are:

<span>  
<a>  
<em>  
<b> etc.

**Ex.**

[Click here for practical](#)

## 2. inline-block

The CSS display inline-block element is very similar to inline element but the difference is that you are able to set the width and height.

**Ex.**

[\*Click here for practical\*](#)

# 3. block

The CSS display block element takes as much as horizontal space as they can. Means the block element takes the full available width. They make a line break before and after them.

**Ex.**

[Click here for practical](#)

## 4. none

The "none" value totally removes the element from the page. It will not take any space.

**Ex.**

[Click here for practical](#)

# 5. flex

It is used to display an element as an block-level flex container. It is new in css3.

**Ex.**

[Click here for practical](#)

# CSS Flex

## CSS Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

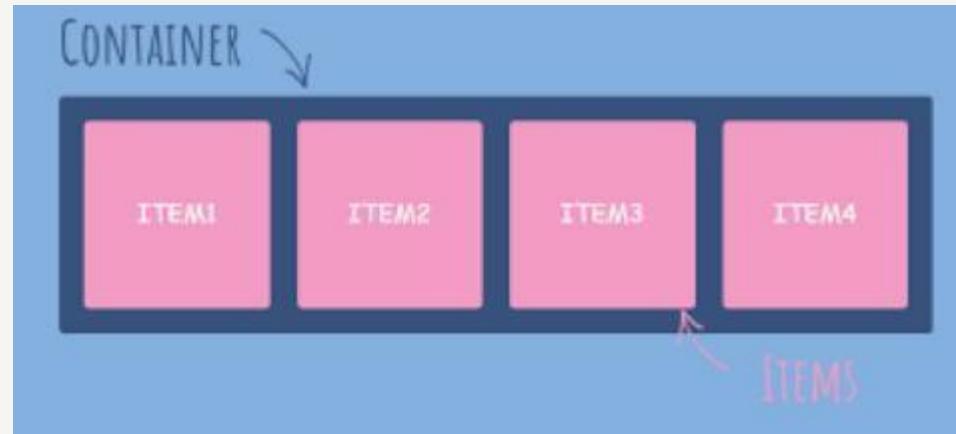
- ✓ Block, for sections in a webpage
- ✓ Inline, for text
- ✓ Table, for two-dimensional table data
- ✓ Positioned, for explicit position of an element
- ✓ The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

[Click here for practical](#)

# CSS Flex Items

The direct child elements of a flex container automatically becomes flexible (flex) items.

[\*\*\*Click here for practical\*\*\*](#)



[Click here for practical](#)

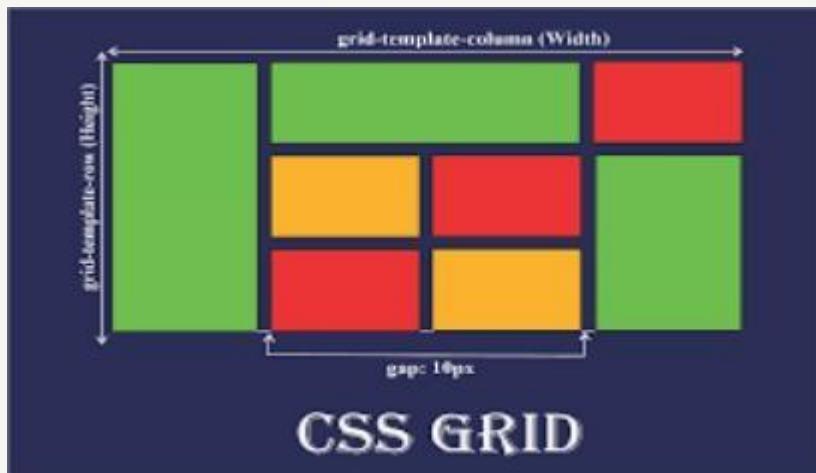


# CSS Grid

## Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

[Click here for practical](#)



# CSS Grid Container

To make an HTML element behave as a grid container, you have to set the display property to grid or inline-grid.

**[Click here for practical](#)**

# CSS Grid Item

A grid *container* contains grid *items*.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

**Note:** The grid-column property is a shorthand property for the grid-column- start and the grid-column-end properties.

## Example

Make "item1" start on column 1 and end before column 5:

[Click here for practical](#)

# CSS Position Property

# CSS Position

The CSS position property is used to set position for an element. it is also used to place an element behind another and also useful for scripted animation effect.

You can position an element using the top, bottom, left and right properties. These properties can be used only after position property is set first. A position element's computed position property is relative, absolute, fixed or sticky.

1. CSS Static Positioning
2. CSS Fixed Positioning
3. CSS Relative Positioning
4. CSS Absolute Positioning

# 1. CSS Static Positioning

This is a by default position for HTML elements. It always positions an element according to the normal flow of the page. It is not affected by the top, bottom, left and right properties.

**Ex.**

```
position: static;
```

[Click here for practical](#)

## 2. CSS Fixed Positioning

The fixed positioning property helps to put the text fixed on the browser. This fixed test is positioned relative to the browser window, and doesn't move even you scroll the window.

**Ex.**

```
position: fixed;
```

[Click here for practical](#)

# 3. CSS Relative Positioning

The relative positioning property is used to set the element relative to its normal position.

**Ex.**

position: relative;

**[Click here for practical](#)**

# 4. CSS Absolute Positioning

The absolute positioning is used to position an element relative to the first parent element that has a position other than static. If no such element is found, the containing block is HTML.

**Ex.**

```
position: absolute;
```

[Click here for practical](#)

# 5. CSS Sticky Property

The CSS position property is used to set the position for an element. It is also used to place an item behind another element and also useful for the scripted animation effect. The "position: sticky;" is used to position the element based on the scroll position of the user. This CSS property allows the elements to stick when the scroll reaches to a certain point. Depends upon the scroll position, a sticky element toggles in between fixed and relative. The element will be positioned relative until the specified position of offset is met in the viewport. Then, similar to position: fixed, the element sticks in one place.

**Ex.**

position: sticky;      [\*\*\*Click here for practical\*\*\*](#)

# CSS Positions

# CSS Positions

property is used to control the positioning of an element on the page. It defines how an element is positioned relative to its normal position or to its containing element. Here's a breakdown of the different values you can use with the **position** property

[Click here for practical](#)

# How the property works

The effects of this property on positioned elements other than the value static are listed as follows:

- When the element is absolutely or fixed positioned (i.e., position: absolute; and position: fixed;), the left property specifies the distance between the element's left edge and the left edge of its containing block (ancestor to which the element is relatively positioned).
- If the element is relatively positioned (i.e., position: relative;), the left property sets the element's left edge to the left/right from its normal position.
- If the position is set to sticky, e., position: sticky; then the positioning context is the viewport. When the element is inside the viewport, the left property behaves like its position is relative. When the element is outside, the left property behaves like its position is fixed.

# left, right, top and bottom

This CSS property specifies the left, right, top and bottom offset for the horizontal positioned elements and does not affect the non-positioned elements. When both left and right properties are defined, the right value has a preference if the container is right-to-left, and the left value has preference if the container is left-to-right.

[\*\*Click here for practical\*\*](#)

## Syntax:

left/right/top/bottom : auto | length | percentage

[\*\*Click here for practical\*\*](#)

# Values

Value	Description
auto	This is the default value.
length	This value defines the position of the property in px, cm, pt, etc. It allows negative values.
percentage	This value defines the position of the property in percentage (%). It is calculated to the width of the element's containing block. It also allows negative values.

# CSS Cursor Property

# CSS Cursor

It is used to define the type of mouse cursor when the mouse pointer is on the element.

1. cursor:alias
2. cursor:auto
3. cursor:all-scroll
4. cursor:col-resize
5. cursor:crosshair
6. cursor:default
7. cursor:copy
8. cursor:pointer
9. Cursor:move [Click here for practical](#)

# CSS Cursor...

- 10. cursor:e-resize
- 11. cursor:ew-resize
- 12. cursor:ne-resize
- 13. cursor:nw-resize
- 14. cursor:n-resize
- 15. cursor:se-resize
- 16. cursor:sw-resize
- 17. cursor:s-resize
- 18. cursor:w-resize
- 19. cursor:text
- 20. Cursor:wait
- 21. cursor:help
- 22. cursor:progress
- 23. cursor:no-drop
- 24. cursor:not-allowed
- 25. cursor:vertical-text
- 26. cursor:zoom-in
- 27. cursor:zoom-out

[\*\*\*Click here for practical\*\*\*](#)

# CSS Buttons

# CSS Button

In HTML, we use the button tag to create a button, but by using CSS properties, we can style the buttons. Buttons help us to create user interaction and event processing.

[\*\*\*Click here for practical\*\*\*](#)

# background-color

This property is used for setting the background color of the button element.

**Ex.**

[\*Click here for practical\*](#)

# border

It is used to set the border of the button. It is the shorthand property for border-width, border-color, and border-style.

**[Click here for practical](#)**

# border-radius

It is used to make the rounded corners of the button. It sets the border radius of the button.

[\*Click here for practical\*](#)

# box-shadow

As its name implies, it is used to create the shadow of the button box. It is used to add the shadow to the button. We can also create a shadow during the hover on the button.

`box-shadow: [horizontal offset] [vertical offset] [blur radius] [optional spread radius] [color];`

**Ex.**

[\*\*Click here for practical\*\*](#)

# padding

It is used to set the button padding.

box-shadow: [horizontal offset] [vertical offset] [blur radius] [optional spread radius] [color];

**Ex.**

[Click here for practical](#)

# CSS Positioning

# CSS Float

The CSS float property is a positioning property. It is used to push an element to the left or right, allowing other element to wrap around it. It is generally used with images and layouts.

Center Aligned

Left  
Aligned



Right  
Aligned

# Concept of Float

A floated element may be moved as far to the left or the right as possible. Simply, it means that a floated element can display at extreme left or extreme right.

The elements after the floating element will flow around it.

The elements before the floating element will not be affected.

If the image floated to the right, the texts flow around it, to the left and if the image floated to the left, the text flows around it, to the right.

[\*\*Click here for practical\*\*](#)

# Float Syntax :

```
<style>
img {
    float: right;
}
</style>
```

[\*\*Click here for practical\*\*](#)

# CSS clearfix

A clear float (or clearfix) is a way for an element to fix or clear the child elements so that we do not require to add additional markup. It resolves the error which occurs when more than one floated elements are stacked next to each other.

**Ex.**

[Click here for practical](#)

# CSS Colors

# CSS Colors

The color property in CSS is used to set the color of HTML elements. Typically, this property is used to set the background color or the font color of an element.

Ways to show colors in CSS:

1. RGB format.
2. RGBA format.
3. Hexadecimal notation.
4. HSL.
5. HSLA.
6. Built-in color.

[\*\*Click here for practical\*\*](#)

# RGB Format

RGB format is the short form of 'RED GREEN and BLUE' that is used for defining the color of an HTML element simply by specifying the values of R, G, B that are in the range of 0 to 255.

The color values in this format are specified by using the `rgb()` property. This property allows three values that can either be in percentage or integer (range from 0 to 255).

This property is not supported in all browsers;

**Syntax:**

`color: rgb(R, G, B);` [Click here for practical](#)

# RGBA Format

It is almost similar to RGB format except that RGBA contains A (Alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 is for fully transparent, and 1.0 is for not transparent.

## Syntax:

```
color:rgba(R, G, B, A);
```

[\*\*\*Click here for practical\*\*\*](#)

# Hexadecimal notation

Hexadecimal can be defined as a six-digit color representation. This notation starts with the # symbol followed by six characters ranges from 0 to F. In hexadecimal notation, the first two digits represent the red (RR) color value, the next two digits represent the green (GG) color value, and the last two digits represent the blue (BB) color value.

## Syntax:

color:#(0-F)(0-F)(0-F)(0-F)(0-F)(0-F);

## Shorthand Hex-Code:

It is a short form of hexadecimal notation in which every digit is recreated.

[Click here for practical](#)

# HSL

- **Hue:** A degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- **Saturation:** A percentage value. 0% means a shade of gray, and 100% is the full color.
- **Lightness:** Also a percentage, representing how light or dark the color is.

[Click here for practical](#)

# HSLA

It is entirely similar to HSL property, except that it contains A (alpha) that specifies the element's transparency. The value of alpha is in the range 0.0 to 1.0, in which 0.0 indicates fully transparent, and 1.0 indicates not transparent.

## Syntax:

color:hsla(H, S, L, A);

[Click here for practical](#)

# Built-in Color

As its name implies, built-in color means the collection of previously defined colors that are used by using a name such as red, blue, green, etc.

## Syntax:

color: color-name;

[Click here for practical](#)

# CSS Important

# CSS Important

This property in CSS is used to give more importance compare to normal property. The !important means 'this is important'. This rule provides a way of making the Cascade in CSS.

If a rule is defined with this attribute, it will reject the normal concern in which the later used rule overrides the previous ones. If we use more than one declaration marked !important, then the normal cascade takes it over again. That means the new marked !important will replace the previous one.

[\*\*Click here for practical\*\*](#)

# **Line Height, Padding and Margin**

# Line Height

The CSS line height property is used to define the minimal height of line boxes within the element. It sets the differences between two lines of your content.

It defines the amount of space above and below inline elements. It allows you to set the height of a line of independently from the font size.

**Ex.**

[\*Click here for practical\*](#)

# CSS Margin

CSS Margin property is used to define the space around elements. It is completely transparent and doesn't have any background color. It clears an area around the element.

Top, bottom, left and right margin can be changed independently using separate properties. You can also change all properties at once by using shorthand margin property.

**Margin Properties:**  
[Click here for practical](#)

# CSS Margin

margin: value auto;

margin: value length in px,pt cm; margin: value %;

**Margin Shorthand properties:** margin: 50px 100px 150px 200px; margin: 50px 100px 150px; margin: 50px 100px; margin 50px;

**[Click here for practical](#)**

# CSS Padding

CSS Padding property is used to define the space between the element content and the element border.

It is different from CSS margin in the way that CSS margin defines the space around elements. CSS padding is affected by the background colors. It clears an area around the content.

Top, bottom, left and right padding can be changed independently using separate properties. You can also change all properties at once by using shorthand padding property.

[\*\*Click here for practical\*\*](#)

# CSS Padding Properties

padding  
padding-left  
padding-right  
padding-top  
padding-bottom

## **Values:**

padding: value in %;  
padding: value in length as in px, pt,  
cm;

[Click here for practical](#)

## **Margin Shorthand**

**properties:** padding: 50px  
100px 150px 200px; padding:  
50px 100px 150px; padding:  
50px 100px; padding: 50px;

# CSS Filters

# What is CSS Filters

CSS filters are used to set visual effects to text, images, and other aspects of a webpage. The CSS filter property allows us to access the effects such as color or blur, shifting on the rendering of an element before the element gets displayed.

## Syntax:

filter: none filter: invert()

filter: drop-shadow() filter:

brightness() filter: saturate() filter:

blur()

filter: hue-rotate()

filter: contrast() filter:  
opacity() filter: grayscale()  
filter: sepia() filter: url();

**[Click here for practical](#)**

# brightness()

As its name implies, it is used to set the brightness of an element. If the brightness is 0%, then it represents completely black, whereas 100% brightness represents the original one. It can also accept values above 100% that provide brighter results.

## Syntax:

filter: brightness( value in %);

[Click here for practical](#)

# blur()

It is used to apply the blur effect to the element. If the blur value is not specified, then the value 0 is used as a default value. The parameter in blur() property does not accept the percentage values. A larger value of it creates more blur.

## Syntax:

filter: blur( value in px);

[\*\*Click here for practical\*\*](#)

# invert()

It is used to invert the samples in the input image. Its 100% value represents completely inverted, and 0% values leave the unchanged input. Negative values are not allowed in it.

## Syntax:

filter: invert(value);

[Click here for practical](#)

# saturate()

It sets the saturation of an element. The 0% saturation represents the completely unsaturated element, whereas the 100% saturation represents the original one. The values greater than 100% are allowed that provides super-saturated results. We cannot use negative values with this property.

## Syntax:

filter: saturate(40);

[Click here for practical](#)

# drop-shadow()

It applies the drop-shadow effect to the input image.

The values it accepts are h-shadow, v-shadow, blur, spread, and color.

## Syntax:

filter:            drop-shadow(10px 20px 30px  
                      yellow);

[Click here for practical](#)

# contrast()

It adjusts the contrast of the input. Its 0% value will create a completely black image, whereas the 100% values leave the unchanged input, i.e., represents the original one. Values greater than 100% are allowed that provides results with less contrast.

## Syntax:

filter: contrast(50%);

[Click here for practical](#)

# opacity()

It is used to apply transparency to the input image. Its 0% value indicates completely transparent, whereas the 100% value represents the original image, i.e., fully opaque.

## Syntax:

`filter: opacity(40%);`

[Click here for practical](#)

# hue-rotate()

It applies a hue-rotation on the input image. Its perimeter value defines the number of degrees around the color circle; the image will be adjusted. Its default value is 0 degree, which represents the original image. Its maximum value is 360 degrees.

## Syntax:

filter: hue-rotate(240deg);

[Click here for practical](#)

# grayscale()

It converts the input image into black and white. 0% grayscale represents the original one, whereas 100% represents completely grayscale. It converts the object colors into 256 shades of gray.

## Syntax:

```
grayscale(80%);
```

[Click here for practical](#)

# sepia()

It is used to transform the image into a sepia image. 0% value represents the original image, whereas the 100% value indicates the completely sepia.

## Syntax:

filter:            sepia(90%);

[\*\*\*Click here for practical\*\*\*](#)

# CSS Images

# CSS Images

Images are an important part of any web application. Including a lot of images in a web application is generally not recommended, but it is important to use the images wherever they required. CSS helps us to control the display of images in web applications.

The styling of an image in CSS is similar to the styling of an element by using the borders and margins. There are multiple CSS properties such as border property, height property, width property, etc. that helps us to style an image.

# Thumbnail Image

The border property is used to make a thumbnail image.

**Ex.**

```
img{  
border: 2px solid red;  
border-radius:5px;  
padding:10px;  
}
```

[Click here for practical](#)

# Transparent image

To make an image transparent, we have to use the opacity property. The value of this property lies between 0.0 to 1.0, respectively.

**Ex.**

```
img{  
border: 2px solid red;  
border-radius:5p  
x; padding:10px;  
opacity:0.3;  
}
```

[Click here for practical](#)

# Rounded image

The border-radius property sets the radius of the bordered image. It is used to create the rounded images.

**Ex.**

```
#img1{  
border-radius:10px;  
}
```

```
#img2{  
border-radius:50%;  
}
```

[\*\*\*Click here for practical\*\*\*](#)

# Responsive Image

It automatically adjusts to fit on the screen size. It is used to adjust the image to the specified box automatically.

**Ex.**

```
#img1{  
max-width:100  
%; height:auto;  
}
```

[\*\*\*Click here for practical\*\*\*](#)

# Center an Image

We can center an image by using the left-margin and right-margin property. We have to set these properties to auto in order to make a block element.

**Ex.**

```
#img1{  
margin-left:auto;  
margin-right:aut  
o; display:block;
```

{[Click here for practical](#)

# CSS Overflow

# CSS Overflow

The CSS overflow property specifies how to handle the content when it overflows its block level container.

We know that every single element on a page is a rectangular box and the size, positioning and behavior of these boxes are controlled via CSS.

Let's take an example: If you don't set the height of the box, it will grow as large as the content. But if you set a specific height or width of the box and the content inside cannot fit then what will happen. The CSS overflow property is used to overcome this problem. It specifies whether to clip content, render scroll bars, or just display content.

# Overflow Values

1.visible: It specifies that overflow is not clipped. it renders outside the element's box.this is a default value.

1.hidden: It specifies that the overflow is clipped, and rest of the content will be invisible.

1.scroll: It specifies that the overflow is clipped, and a scroll bar is used to see the rest of the content.

**[Click here for practical](#)**

1.auto: It specifies that if overflow is clipped, a scroll bar is needed to see the rest of the content.

**[Click here for practical](#)**

# CSS Vertical Align, White Space and Word Wrap

# Vertical Align

The CSS vertical align property is used to define the vertical alignment of an inline or table-cell box.

1. It is applied to inline or inline-block elements.
2. It affects the alignment of the element, not its content. (except table cells)
3. When it applied to the table cells, it affect the cell contents, not the cell itself.

# Vertical Align Values

vertical-align: baseline  
vertical-align: length;  
vertical-align: %;  
vertical-align: sub;  
vertical-align: super;

vertical-align: top;  
vertical-align: bottom;  
vertical-align: text-top;  
vertical-align: middle;  
vertical-align: text-bottom;

[Click here for practical](#)

# White Space

The CSS white space property is used to specify how to display the content within an element. It is used to handle the white spaces inside an element.

## **Values:**

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

[Click here for practical](#)

# Word Wrap

CSS word wrap property is used to break the long words and wrap onto the next line. This property is used to prevent overflow when an unbreakable string is too long to fit in the containing box.

## **Values:**

word-wrap: normal;  
word-wrap:  
break-word;  
  
word-wrap: break-all;

[Click here for practical](#)

# CSS Width and Height

# CSS Width Property

The CSS width property is used to set the width of the content area of an element.

It does not include padding borders or margins. It sets width of the area inside the padding, border, and margin of the element.

## Values:

We can use **max-width** and **min-width** properties to set the maximum and minimum width of

[Click here for practical](#)

# CSS Height Property

This CSS property sets the height of an element. It is used to set the height of content area of an element.

It does not include padding borders or margins, whereas it sets the height of the area inside the padding, border, and margin of the element. It can accept the length and percentage values. But it does not allow negative values.

If we set the height to a numeric value (like in px, %, etc.), the content can be overflow if it does not fit in the given height. We can manage the overflowing content by defining the overflow property.

# CSS Height Values

## Values:

width: auto;

width:

length;

We can use **max-height** and **min-height** properties to set the maximum and minimum height of the element

[Click here for practical](#)

# CSS Box-shadow and Text-shadow

# Box-shadow CSS

It is used to add shadow-like effects around the frame of an element.

## Syntax:

```
box-shadow: h-offset v-offset blur spread color  
|inset|inherit|initial|none;
```

[Click here for practical](#)

# Text-shadow CSS

As its name implies, this CSS property adds shadows to the text. It accepts the comma-separated list of shadows that applied to the text. Its default property is none. It applies one or more than one text-shadow effect on the element's text content.

## Syntax:

```
text-shadow: h-shadow v-shadow blur-radius color|  
none | initial | inherit;
```

[Click here for practical](#)

# Box-shadow/Text Shadow Values

**h-offset:** It horizontally sets the shadow position. Its positive value will set the shadow to the right side of the box. Its negative value is used to set the shadow on the left side of the box.

**v-offset:** Unlike the h-offset, it is used to set the shadow position vertically. The positive value in it sets the shadow below the box, and the negative value sets the shadow above of the box.

**blur:** As its name implies, it is used to blur the box-shadow. This attribute is optional.

**spread:** It sets the shadow size. The spread size depends upon the spread value.

**color:** As its name implies, this attribute is used to set the color of the shadow. It is an optional attribute. [Click here for practical](#)

# Box-shadow/Text Shadow Values

inset: Normally, the shadow generates outside of the box, but by using inset, the shadow can be created within the box.

initial: It is used to set the property of the box-shadow to its default value.  
inherit: it is inherited from its parent.

none: It is the default value that does not include any shadow property.  
[Click here for practical](#)

# Box-shadow

h-offset, v-offset and blur attributes  
box-shadow: 5px 10px 10px;

spread attribute  
box-shadow: 5px 10px 10px 10px;

color attribute  
box-shadow: 5px 10px 10px 10px  
orange;

[\*\*Click here for practical\*\*](#)

inset attribute  
box-shadow: 5px 10px 10px 10px  
orange inset;

initial attribute  
box-shadow: initial;

default attribute  
box-shadow:  
none;

# Box-shadow

initial attribute  
box-shadow: initial;

default attribute box-shadow: none;

**[Click here for practical](#)**

# Text-shadow

Simple Shadow

text-shadow: 3px 3px red;

Fuzzy Shadow

text-shadow: 3px 3px 3px violet;

Multiple Shadows

text-shadow: -3px -3px 3px blue, 3px 3px 3px red;

Glow Effect

text-shadow: 0 0 .1em cyan;

[Click here for practical](#)

# CSS Visibility

# CSS Visibility

The CSS visibility property is used to specify whether an element is visible or not.

Note: An invisible element also take up the space on the page.  
By using display property you can create invisible elements that don't take up space.

## Syntax:

visibility: visible|hidden

[Click here for practical](#)

# Ex. with CSS

h1.visible

```
{ visibility: visible
```

```
}
```

h1.hidden

```
{ visibility: hidden
```

```
}
```

[\*\*\*Click here for practical\*\*\*](#)

# CSS Icons

# CSS Icons

Icons can be defined as the images or symbols used in any computer interface refer to an element. It is a graphical representation of a file or program that helps the user to identify about the type of file quickly.

Using the icon library is the easiest way to add icons to our HTML page. It is possible to format the library icons by using CSS. We can customize the icons according to their color, shadow, size, etc.

There are given some of the icon libraries such as Bootstrap icons, Font Awesome icons, and Google icons that can be used in CSS easily. There is no need to install or download the libraries mentioned above.

# Font Awesome Icons

To use this library in our HTML page, we need to add the following link within the `<head></head>` section.

```
<i class="fa fa-cloud"></i>
<i class="fa fa-file"></i>
<i class="fa fa-heart"></i>
<i class="fa fa-bars"></i>
<i class="fa fa-car"></i>
```

[Click here for practical](#)

# Bootstrap Icons

As similar to the font awesome library, we can add the bootstrap icons in our HTML page using the link given below in the `<head></head>` section.

```
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-file"></i>
<i class="glyphicon glyphicon-heart"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-thumbs-up"></i>
```

[\*\*\*Click here for practical\*\*\*](#)

# Google Icons

As similar to the above libraries, we can add it in our HTML page simply by adding the link given below in the `<head></head>` section.

```
<link rel="stylesheet"  
      href="https://fonts.googleapis.com/icon?family=Material+Icons">
```

```
<i class="material-icons">cloud</i>  
<i class="material-icons">attachment</i>  
<i class="material-icons">computer</i>  
<i class="material-icons">favorite</i>  
<i class="material-icons">traffic</i>
```

[Click here for practical](#)

# **Justify, Text Decoration and Text-align**

# Justify

This CSS property is used to align the items of the flexible box container when the items do not use all available space on the main-axis (horizontally). It defines how the browser distributes the space around and between the content items.

This CSS property can't be used to describe containers or items along the vertical axis. To align the items vertically, we have to use the align-items property.

## Syntax:

justify-content: center | flex-start | flex-end | space-around | space-evenly | space-between

[Click here for practical](#)

# Property Values

**center:** As its name implies, it set the position of items at the center of the container.

**flex-start:** It is the default value that positioned the items at the beginning of the container.

**flex-end:** It set the position of items at the end of the container.

**space-around:** It positioned the items with equal spacing from each other. It evenly distributes the items in the line along with the same space around them.

**space-between:** Items are evenly spaced in which the first item is at the beginning, and the last item is at the end.

**space-evenly:** It also positioned the items with equal space, but the spacing from the corners differs.

[\*\*Click here for practical\*\*](#)

# Text Decoration

It is a CSS property that decorates the content of the text. It adds lines under, above, and through the text. It sets the appearance of decorative lines on text. This CSS property decorates the text with several kinds of lines. This is shorthand for text-decoration-line, text-decoration-color, and text-decoration-style.

Text Decoration can be done in below mentioned ways:

1. text-decoration-line
2. text-decoration-color
3. text-decoration-style

[\*\*\*Click here for practical\*\*\*](#)

# text-decoration-line

It sets the kind of text-decoration like overline, underline, or line-through.  
It can be used to add a combination of lines to the selected text.

## Syntax:

text-decoration: underline | line-through |

overline text-decoration: overline underline

line- through;

[Click here for practical](#)

# text-decoration-style

This property is used to set the style of the line. Its values are solid, dotted, wavy, double, and dashed.

## Syntax:

text-decoration: underline double| line-through dashed|  
overline overline text-decoration: lightblue wavy overline  
underline line-through;

[Click here for practical](#)

# text-decoration-color

This property is used to provide color to the decoration. Its value is any color in a valid format.

## Syntax:

text-decoration: underline double cyan| line-through dashed  
green| dotted overline blue

text-decoration: lightblue wavy overline underline line-through;

[Click here for practical](#)

# Text Align

This CSS property is used to set the horizontal alignment of a table-cell box or the block element. It is similar to the vertical-align property but in the horizontal direction.

## Syntax:

text-align: justify | center | right | left

[Click here for practical](#)

# Property Values

**justify:** It is generally used in newspapers and magazines. It stretches the element's content in order to display the equal width of every line.

**center:** It centers the inline text.

**right:** It is used to align the text to the right.

**left:** It is used to align the text to the left.

[Click here for practical](#)

# **:nth-child(n) selector and :first-child selector**

# **nth-child(n) selector**

This selector is used for matching the elements based on their position regardless of the type of its parent. The n can either be a keyword, formula, or a number. It is used to match the elements based on their position within a group of siblings. It matches each element, which is the nth-child.

## **Syntax:**

[Click here for practical](#)

The "n" in the parentheses is the argument that represents the pattern for matching elements. It can be a functional notation, even or odd.

# first-child and last-child Selector

The :first-child selector is used to select the specified selector, only if it is the first child of its parent.

**Syntax:**

```
:first-child {
```

```
}
```

The :last-child selector matches every element that is the last child of its parent.

**Syntax:**

[\*\*\*Click here for practical\*\*\*](#)

# CSS Calc()

# calc()

It is an inbuilt CSS function which allows us to perform calculations. It can be used to calculate length, percentage, time, number, integer frequency, or angle. It uses the four simple arithmetic operators add (+), multiply (\*), subtract (-), and divide (/).

It is a powerful CSS concept because it allows us to mix any units, such as percentage and pixel.

## Syntax:

calc( Expression ); [Click here for practical](#)

# Values

This CSS function takes a single parameter expression, and the result of this mathematical expression will be used as a value. It can be any simple expression using the four arithmetic operators (+, -, \*, /). The expression is mandatory to define.

Ex.

```
.div {  
    width: calc(150% - 75%);  
    height: calc(350px - 75px);  
}
```

**[Click here for practical](#)**

```
.div {  
    width: calc(40% +  
    10em);  
    height:  
    calc(350px +  
    75px);  
}
```

# CSS Print Properties

# CSS Print

While we need to print the current webpage, css prints page related properties comes into the picture.

There are total of 3 types of print properties available.

1. page-break-before
2. page-break-inside
3. page-break-after

# page-break-before

As the name implies, this CSS property is used to add the page break before the element, when printing the document. It inserts a page break before the specified element during printing the document. We cannot use this CSS property on absolutely positioned elements or an empty <div> element that does not generate a box.

## **Syntax:**

page-break-before: auto | always | left | right | avoid

# Values

Value	Description
auto	It is the default value that inserts a page break before the element, if necessary.
always	This value always forces a page break before the specified element.
avoid	It is used to avoid a page break before the element whenever possible.
left	This value forces either one or two page breaks before the element so that the next page will be depicted as the left-hand page.
right	The right value forces either one or two page breaks before the element so that the next page will be depicted as the right-hand page.

# page-break-inside

this CSS property is used to specify the page break inside the element, when printing the document. This CSS property cannot be used on absolutely positioned elements or an empty <div> element that does not generate a box. It inserts or avoids the page break inside the specified element during printing the document.

## Syntax:

page-break-inside:      auto | avoid

# Values

Value	Description
auto	It is the default value that inserts a page break inside the element, if necessary.
always	It is used to avoid a page break inside the element whenever possible.

# page-break-after

This CSS property is used to adjust the page break after the element when printing the document. It inserts a page break after the specified element during printing.

## Syntax:

page-break-after: auto | always | left | right | avoid

# Values

Value	Description
auto	It is the default value that inserts a page break after the element, if necessary.
always	It always forces a page break after the specified element.
avoid	It is used to avoid a page break after the element whenever possible.
left	It forces either one or two page breaks after the specified element so that the next page will be depicted as the left-hand page.
right	It forces either one or two page breaks after the specified element so that the next page will be depicted as the right-hand page.

# CSS Columns

# CSS Columns

The columns property in CSS sets the number and width of columns in a single declaration. It is a shorthand property that can take several values at a time.

It is used to set both column-count and column-width properties at the same time. Both of these properties are used to control how many columns will appear. The column-count property is used to set the number of columns, whereas the column-width property specifies the width of the columns.

# CSS Columns

Together using column-count and column-width properties creates a multi- column layout that breaks automatically into a single column at narrow browser widths without using the media queries. It is not always helpful to use both of them because it can restrict the responsiveness and flexibility of the layout.

# Values

## Syntax:

columns: auto | column-width

column-count| [\*\*Click here for practical\*\*](#)

Value	Description
Auto	It is the default value which sets the values of column-count and column-width to the default browser values.
column-width	It is used to set the minimum width for columns. However, the actual width of the column may be narrower or wider based on the available space.
column-count	It specifies the maximum number of columns.

# CSS Hyphens

# CSS Hyphens

This CSS property is used to control the hyphenation of the text in the block-level elements. It defines how the word is hyphenated if it is too long or when the text wraps across multiple lines.

This property allows us to split the word into two lines to improve the text layout.

## Syntax:

hyphens: none | manual | auto

[Click here for practical](#)

# Values

Value	Description
none	This value does not hyphenate the words. It never hyphenates the words at line breaks or even if the word is too long.
manual	It is the default value that hyphenates the word only when the characters in the word suggest hyphenation opportunities. The two Unicode characters are defined below, which can be used manually to specify the possible line breakpoints in the text.
auto	In this value, the algorithm decides where the words are hyphenated.

[\*\*Click here for practical\*\*](#)

# CSS Transform and Resize

# CSS transform-origin property

This CSS property is used to change the position of transformed elements. It is a point around which the transformation is applied. It establishes the origin of the transformation of an element. It can be applied to both 2D and 3D rotations.

The transform-origin property must be used with the transform property. **The 2d transformation can change the x-axis and y-axis of the element, whereas the 3D transformation can change the z-axis along with the x-axis and y-axis.**

# CSS transform-origin property

This property can be specified by using one, two, or three values. The first value affects the horizontal position, the second value affects the vertical position, and the third value indicates the position of the z-axis. The third value can also work on 3D transformations and cannot be defined using a percentage.

## Syntax:

transform-origin: x-axis y-axis z-axis

Ex:

for 2D Transformation: transform: rotate(35deg);  
transform-origin: 5% 2%;

for 3D Transformation:

transform: rotate3d(3, 2, 1, 75deg);  
transform-origin: 70% 10% 150px;

**[Click here for practical](#)**

# CSS Resize property

This CSS property allows the user to control the resizing of an element just by clicking or dragging the bottom right corner of the element

This CSS property is used to define how an element is resizable by the user. It doesn't apply on the block or inline elements where overflow is set to visible. So, to control the resizing of an element, we have to set the overflow other than visible like (overflow: hidden or overflow: scroll).

## **Resize:**

resize: none | horizontal | vertical | both [Click here for practical](#)

# Values

Value	Descriptions
none	It is the default value of this property, which does not allow resizing the element.
horizontal	This value allows the user to resize the element's width. It resizes the element in a horizontal direction. There is a unidirectional horizontal mechanism for controlling the width of an element.
vertical	It allows the user to resize the height of an element. It resizes the element in a vertical direction. There is a unidirectional vertical mechanism for controlling the height of an element.
both	It allows the user to resize the width and height of an element. It resizes the element in both horizontal and vertical directions.

[\*\*Click here for practical\*\*](#)

# Transition Delay

# CSS Transition Delay Property

This CSS property specifies the duration to start the transition effect. The value of this property is defined as either **seconds (s) or milliseconds (ms)**. The CSS transitions are effects that are added to change the element gradually from one style to another, without using flash or JavaScript. Without using the **transition-delay**, the animation will start with the hover on the element, but using this CSS property, we can delay the animation by an amount of time. The default value of the transition-delay property is 0, which means that the transition will start to occur immediately without any delay.

## Syntax:

`transition-delay: time`

[Click here for practical](#)

# Values

Value	Description
time	It specifies the amount of time (in seconds or milliseconds) to wait before the transition starts.
initial	It sets this property to its default value.
inherit	It inherits this property from its parent element.

[\*\*\*Click here for practical\*\*\*](#)

# Note

The negative value of the transition-delay property will immediately start the transition effect i.e., the effect will be animated as though it had already begun. The positive value of this property causes the transition effect to start for the given time.

[\*\*\*Click here for practical\*\*\*](#)

# CSS Animation

# CSS Animation

CSS Animation property is used to create animation on the webpage. It can be used as a replacement of animation created by Flash and JavaScript. An animation makes an element change gradually from one style to another. You can add as many as properties you want to add. You can also specify the changes in percentage. 0% specify the start of the animation and 100% specify its completion.

## **CSS3 @keyframes Rule:**

The animation is created in the @keyframe rule. It is used to control the intermediate steps in a CSS animation sequence.

# How does it work?

When the animation is created in the @keyframe rule, it must be bound with selector; otherwise the animation will have no effect.

The animation could be bound to the selector by specifying at least these two properties:

- The name of the animation
- The duration of the animation

[Click here for practical](#)

# Animation

Property	Description
@keyframes	It is used to specify the animation.
animation	This is a shorthand property, used for setting all the properties, except the animation-play-state and the animation-fill-mode property.
animation-delay	It specifies when the animation will start.
animation-direction	It specifies if or not the animation should play in reserve on alternate cycle.
animation-duration	It specifies the time duration taken by the animation to complete one cycle.
animation-fill-mode	It specifies the static style of the element. (when the animation is not playing)
animation-iteration-count	It specifies the number of times the animation should be played.
animation-play-state	It specifies if the animation is running or paused.
animation-name	It specifies the name of @keyframes animation.
animation-timing-function	It specifies the speed curve of the animation.

# Ex. 1 with ‘from’ and ‘to’

**change background color of rectangle from RED  
to BLACK.**

[Click here for practical](#)

# Ex. 2 with percentage values

[Click here for practical](#)

# @keyframe

# CSS @keyframes rule

The CSS @keyframe specifies the animation rule that defines the CSS properties for the elements at each state with a timeline.

We can create complex animation properties by using the @keyframe. An animation is created with the changeable CSS styles that can be repeated indefinitely or a finite number of times. A simple animation can just have two keyframes, while the complex animation has several keyframes.

To use keyframes, we need to create a @keyframes rule along with the animation- name property for matching an animation with its keyframe declaration.

# Property Values

Values	Description
animation - name	It is the required value that defines the name of the animation.
keyframes-selector	It specifies the percentage along with the animation when the keyframe occurs. Its value lies between 0% to 100%, from (same as 0%), to (same as 100%). Keyframe percentages can be written in any order because they will be handled in the order they occur.
css-styles	It defines one or more than one CSS style properties.

# @keyframe timings

Values	Description
linear	It means that the transition rate will be constant from start to end.
ease	It means that the animation starts slowly, and then after a time period, the speed increases, and before end speed will again slow down.
ease-in	It is similar to ease, but the animation ends quickly.
ease-out	It is also similar to ease, but the animation starts fast.

[\*\*Click here for practical\*\*](#)

# CSS Gradient

# CSS Gradient and its uses

CSS gradient is used to display smooth transition within two or more specified colors.

- You don't have to use images to display transition effects.
- The download time and bandwidth usage can also be reduced.
- It provides better look to the element when zoomed, because the gradient is generated by the browser.

There are two types of gradients in CSS3

1. Linear gradients
2. Radial gradients

# CSS Linear Gradient

The CSS3 linear gradient goes up/down/left/right and diagonally. To create a CSS3 linear gradient, you must have to define two or more color stops. The color stops are the colors which are used to create a smooth transition. Starting point and direction can also be added along with the gradient effect.

## Syntax:

`background: linear-gradient (direction, color-stop1, color-stop2.....);`

[Click here for practical](#)

# CSS Linear Gradient: (Top to Bottom)

Top to Bottom Linear Gradient is the default linear gradient. Let's take an example of linear gradient that starts from top. It starts red and transitions to green.

[\*\*\*Click here for practical\*\*\*](#)

# CSS Linear Gradient: (Left to Right)

The following example shows the linear gradient that starts from left and goes to right. It starts red from left side and transitioning to green.

**Ex.**

[Click here for practical](#)

# CSS Linear Gradient: (Diagonal)

If you specify both the horizontal and vertical starting positions, you can make a linear gradient diagonal.

**Ex.**

[Click here for practical](#)

# CSS Radial Gradient

You must have to define at least two color stops to create a radial gradient. It is defined by its center.

## Syntax:

background: radial-gradient(shape size at position,  
start-color, ..., last-color);

[Click here for practical](#)

# CSS Radial Gradient: (Evenly Spaced Color Stops)

Evenly spaced color stops is a by default radial gradient. Its by default shape is ellipse, size is farthest- carner, and position is center.

**Ex.**

[Click here for practical](#)

# Radial Gradient: (Differently Spaced Color Stops)

[\*Click here for practical\*](#)

# Radial Gradient: (From Corners)

[Click here for practical](#)

# CSS z-index

# z-index

The z-index in CSS allows us to define a visual hierarchy on the 3-dimensional plane, i.e., the z-axis. It is used to specify the stacking order of the positioned elements (elements whose position value is either fixed, absolute, relative, or sticky). The stacking order means that the element's position along the z- axis, which is perpendicular to the screen.

Syntax:

z-index: number|auto

[\*\*Click here for practical\*\*](#)

# Masking

# CSS Masking

The mask property in CSS is used to hide an element using the clipping or masking the image at specific points. Masking defines how to use an image or the graphical element as a luminance or alpha mask. It is a graphical operation that can fully or partially hide the portions of an element or object.

Using masking, it is possible to show or hide the parts of an image with different opacity levels. In CSS, the masking is achieved by using the `mask-image` property, and we have to provide an image as the mask.

## **Ex. 1: Mask**

```
#mask{  
-webkit-mask-box-image: url(box.png);  
}
```

## **Ex. 2: Gradient**

#[Click here for practical](#)

# CSS Media Query

# What is Media Query?

CSS Media query is a W3C recommendation and a CSS3 module which is used to adapt to conditions such as screen resolution (e.g. Smartphone screen vs. computer screen).

- The media query technique first used in CSS3.
- It became a W3C recommendation in June 2012.
- It is an extension of media dependent stylesheets used in different media types (i.e. screen and print) found in CSS2.
- The most commonly used media feature is "width".
- It uses the @media rule to include a block of CSS properties only if a certain condition is true.

# What is Responsive Web Design?

It facilitates you to use fluid grids, flexible images, and media queries to progressively enhance a web page for different viewing contexts i.e. Desktop, Smartphone, Tablet etc.



[Click here for practical](#)

# 2D Transforms

# CSS Transforms

CSS3 supports transform property. This transform property facilitates you to translate, rotate, scale, and skews elements.

Transformation is an effect that is used to change shape, size and position. There are two type of transformation i.e. 2D and 3D transformation supported in CSS3.

**Ex.:**

```
translate()  
) rotate()  
scale()  
skewX()  
skewY()
```

# translate()

The CSS translate() method is used to move an element from its current position according to the given parameters i.e. X-axis and Y-axis.

**Ex:**

```
transform: translate(100px,80px);
```

[\*\*\*Click here for practical\*\*\*](#)

# rotate()

The CSS rotate() method is used to rotate an element clockwise or anti-clockwise according to the given degree.

**Ex:**

-ms-transform: rotate(30deg);

-webkit-transform:

rotate(30deg); transform:

rotate(30deg);

[\*\*Click here for practical\*\*](#)

# scale()

The CSS scale() method is used to increase or decrease the size of an element according to the given width and height.

**Ex:**

```
transform: scale(2.5,2);
```

[Click here for practical](#)

# skewX()

The CSS skewX() method is used to skew an element along the X axis according to the given angle.

**Ex:**

```
transform: skewX(30deg);
```

[Click here for practical](#)

# skewY()

The CSS skewY() method is used to skew an element along the Y axis according to the given angle.

**Ex:**

```
transform: skewY(30deg);
```

[Click here for practical](#)

# skew()

The CSS skew() method is used to skew an element along with X-axis and Y- axis according to the given angle.

**Ex:**

```
transform: skew(30deg,20deg);
```

[Click here for practical](#)

# matrix()

The CSS matrix() method combines all the six 2D transform methods altogether.

**Syntax:**

matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())

**Ex:**

transform: matrix(1, -0.3, 0, 1, 0, 0);

[Click here for practical](#)

# 3D Transforms

# The 3D rotateX() method (X-axis rotation)

The CSS 3D rotateX() method is used to rotate an element around its X- axis according to the given degree.

**Ex.:**

```
transform: rotateX(150deg); /* Standard syntax */
```

[Click here for practical](#)

# The 3D rotateY() method (Y-axis rotation)

The CSS 3D rotateY() method is used Y-axis according to the given degree.

to rotate an element around its

**Ex.:**

```
transform: rotateY(150deg); /* Standard syntax  
*/
```

[\*\*\*Click here for practical\*\*\*](#)

# The 3D rotateZ() method (Z-axis rotation)

The CSS 3D rotateZ() method is used to rotate an element around its Z- axis according to the given degree.

**Ex.:**

```
transform: rotateZ(90deg); /* Standard syntax */
```

[\*\*\*Click here for practical\*\*\*](#)

# Module - 7

# Bootstrap Basic & Advanced

- 1) Bootstrap Introduction
- 2) Bootstrap Getting Started
- 3) Bootstrap Grid System
- 4) Bootstrap Fixed Layout
- 5) Bootstrap Fluid Layout
- 6) Bootstrap Responsive Layout
- 7) Bootstrap Typography
- 8) Bootstrap Tables
- 9) Bootstrap Lists
- 10) Bootstrap List Groups
- 11) Bootstrap Forms
- 12) Bootstrap Custom Forms
- 13) Bootstrap Input Groups
- 14) Bootstrap Buttons
- 15) Bootstrap Button Groups

- Bootstrap with CSS: Grid System
  - 1) Bootstrap Images
  - 2) Bootstrap Cards
  - 3) Bootstrap Media Objects
  - 4) Bootstrap Icons
  - 5) Bootstrap Navs
  - 6) Bootstrap Navbar
  - 7) Bootstrap Breadcrumbs
  - 8) Bootstrap Pagination
  - 9) Bootstrap Badges
  - 10) Bootstrap Progress Bars
  - 11) Bootstrap Spinners
  - 12) Bootstrap Jumbotron
  - 13) Bootstrap Helper Classes
- Bootstrap with CSS - typography
- Bootstrap Advanced

- 1) Bootstrap Modals
- 2) Bootstrap Dropdowns
- 3) Bootstrap Tabs
- 4) Bootstrap Tooltips
- 5) Bootstrap Popovers
- 6) Bootstrap Alerts
- 7) Bootstrap Stateful  
Buttons
- 8) Bootstrap Accordion
- 9) Bootstrap Carousel
- 10) Bootstrap Typeahead
- 11) Bootstrap ScrollSpy
- 12) Bootstrap Toasts

- Bootstrap Contextual Classes
- Bootstrap with CSS - Responsive Utilities
- Bootstrap Layout - Glyphicon
- bootstrap Layout - Dropdowns
- Bootstrap Dropup
- Bootstrap - Button Group
- Bootstrap Layout - Navigation Elements
- Navbar
- Breadcrumb
- pagination
- Input Group
- Labels
- Badges
- Jumbotron
- Page Headers
- Alerts
- Progress Bar
- List Group
- Panels
- Wells

# Prerequisites for Bootstrap

- HTML (Mandatory)
- CSS (Mandatory)
- JavaScript/jQuery  
(Moderate)

# Bootstrap



# Bootstrap Alternatives

## 1. Skeleton:

Cons – Limited Templates

## 2. Foundation

Cons – Difficult to modify codes

## 3. UIKit

Cons – Limited Utility classes as compared to Bootstrap

## 4. Bulma

Cons – Still in the development phase

## 5. Pure

Cons – Limited CSS selectors

## 6. Powertocss

Cons – No longer Maintained

## 7. Kickstrap

Cons – No longer Maintained

# Setup

- **Through CDN:**

<https://getbootstrap.com/docs/4.6/getting-started/introduction/>

- **Through Installation:**

<https://getbootstrap.com/docs/4.6/getting-started/download/>

Offline:-

[Click here for practical](#)

Online:-

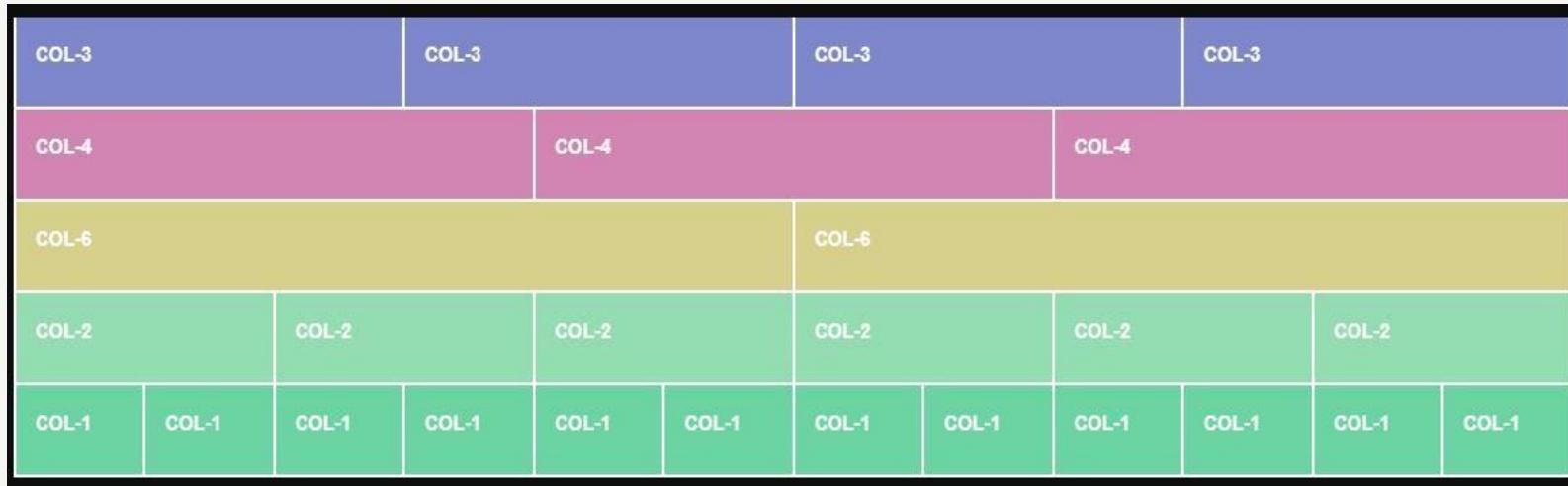
[Click here for practical](#)

# Bootstrap Grid System

# Bootstrap Grid System

- The Bootstrap Grid System allows up to 12 columns across the page. You can use all 12 columns individually or you can group the columns together to create wider columns.
- Bootstrap Grid System is responsive and the columns are rearranged automatically according to the screen size.

# Bootstrap Grid System



# Bootstrap Grid System

There are four classes in Bootstrap Grid System:

- xs (for phones)
- sm (for tablets)
- md (for desktops)
- lg (for larger desktops)

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap Fixed Layout

[Click here for practical](#)

# Bootstrap Fluid Layout

[Click here for practical](#)

# Bootstrap Utilities

# Bootstrap Utilities

- Bootstrap provides some handful helper classes, for faster mobile-friendly development. These can be used for showing and hiding content by device via media query, combined with large, small, and medium devices.

[\*\*Click here for practical\*\*](#)

# Bootstrap Typography

# Bootstrap Typography

Typography provides some utilities to add additional styles to texts.

These utilities are:

1. Text alignment
2. Text transform
3. Font weight and italics

[Click here for practical](#)

[Click here for practical](#)

# Bootstrap Tables

# Bootstrap Tables

- We can create different types of Bootstrap tables by using different classes to style them.

[Click here for practical](#)

[Click here for practical](#)

[Click here for practical](#)

# Bootstrap Forms

# Bootstrap Forms

- In Bootstrap, there are three types of form layouts:
  1. Vertical form (this is default)
  2. Horizontal form
  3. Inline form

[\*\*Click here for practical\*\*](#)

# Bootstrap Rules for Form

- There are three standard rules for these 3 form layouts:
- 1. Always use `<form role="form">` (helps improve accessibility for people using screen readers)
- 2. Wrap labels and form controls in `<div class="form-group">` (needed for optimum spacing)
- 3. Add class `.form-control` to all textual `<input>`, `<textarea>`, and `<select>` elements

**[Click here for practical](#)**

# Input Groups

- The .input-group class is a container to enhance an input by adding an icon, text or a button in front or behind the input field as a "help text".
- Use .input-group-prepend to add the help text in front of the input, and .input-group-append to add it behind the input.
- At last, add the .input-group-text class to style the specified help text.

[\*\*\*Click here for practical\*\*\*](#)

# Sizing

- Add the relative form sizing classes to the .input-group itself and contents within will automatically resize—no need for repeating the form control size classes on each element.

**[Click here for practical](#)**

# Checkboxes and Radios

Place any checkbox or radio option within an input group's addon instead of text.

**[Click here for practical](#)**

# Multiple inputs

While multiple <input>s are supported visually, validation styles are only available for input groups with a single <input>.

**[Click here for practical](#)**

# Multiple addons

Multiple add-ons are supported and can be mixed with checkbox and radio input versions.

[\*\*\*Click here for practical\*\*\*](#)

# Custom forms

- Input groups include support for custom selects and custom file inputs. Browser default versions of these are not supported.

[\*\*Click here for practical\*\*](#)

[\*\*Click here for practical\*\*](#)

# Bootstrap Buttons

There are seven styles to add a button in Bootstrap. Use the following classes to achieve the different button styles:

1. .btn-default
2. .btn-primary
3. .btn-success
4. .btn-info
5. .btn-warning
6. .btn-danger
7. .btn-link

[Click here for practical](#)

# Bootstrap Images

Bootstrap supports for images. There are three classes in Bootstrap that can be used to apply some simple style to the images.

## For Bootstrap 3

1. img-rounded
2. img-circle
3. img-thumbnail

## For Bootstrap 4

1. rounded
2. rounded-circle
3. img-thumbnail

[\*\*Click here for practical\*\*](#)

# Navigation Elements

- Bootstrap provides a few different options for styling navigation elements. All of them share the same markup and base class, **.nav**. Bootstrap also provides a helper class, to share markup and states. Swap modifier classes to switch between each style.
- To create a tabbed navigation menu –
  - Start with a basic unordered list with the base class of **.nav**
  - Add class **.nav-tabs**.

[\*\*Click here for practical\*\*](#)

# Navbar

- The navbar is one of the prominent features of Bootstrap sites.
- Navbars are responsive 'meta' components that serve as navigation headers for your application or site.
- Navbars collapse in mobile views and become horizontal as the available viewport width increases.
- At its core, the navbar includes styling for site names and basic navigation.

[\*\*\*Click here for practical\*\*\*](#)

# Default Navbar

- Add the classes **.navbar**, **.navbar-default** to the `<nav>` tag.
- Add **role = "navigation"** to the above element, to help with accessibility.
- Add a header class **.navbar-header** to the `<div>` element. Include an `<a>` element with class **navbar- brand**. This will give the text a slightly larger size.
- To add links to the navbar, simply add an unordered list with the classes of **.nav**, **.navbar-nav**.

# Responsive Navbar

- To add responsive features to the navbar, the content that you want to be collapsed needs to be wrapped in a <div> with classes **.collapse**, **.navbar-collapse**.
- The collapsing nature is tripped by a button that has the class of **.navbar-toggle** and then features two data- elements.
- The first, **data-toggle**, is used to tell the JavaScript what and the to do with the button, second, **data-target**, indicates which element to toggle.
- Then with a class **.icon-bar** create what we like to call the hamburger button. This will toggle the elements that are in the **.nav-collapse** <div>

[\*\*Click here for practical\*\*](#)

# Navbar Brands

- Adding images to the .navbar-brand will likely always require custom styles or utilities to properly size.

**[Click here for practice](#)**

# Navbar Forms

- To add form elements inside the navbar, add the .navbar-form class to a form element and add an input(s). Note that we have added a .form-group class to the div container holding the input.

[\*\*\*Click here for practical\*\*\*](#)

# Navbar Text

- Use the .navbar-text class to vertical align any elements inside the navbar that are not links (ensures proper padding and text color).

[\*\*\*Click here for practical\*\*\*](#)

# Fixed Navigation Bar

- The navigation bar can also be fixed at the top or at the bottom of the page.
- A fixed navigation bar stays visible in a fixed position (top or bottom) independent of the page scroll.

[Click here for practical](#)

[Click here for practical](#)

# Bootstrap Breadcrumbs

# Breadcrumb

- Breadcrumbs are a great way to show hierarchy-based information for a site.
- In the case of blogs, breadcrumbs can show the dates of publishing, categories, or tags.
- They indicate the current page's location within a navigational hierarchy.

[\*\*Click here for practical\*\*](#)

[\*\*Click here for practical\*\*](#)

# Bootstrap Pagination

[Click here for practical](#)

# Pagination with Icons

With icons pagination

[\*\*Click here for practical\*\*](#)

# Disabled and Active State

[Click here for practical](#)

# Sizing

[Click here for practical](#)

# Alignment

- Change the alignment of pagination components with flexbox utilities.

[Click here for practical](#)

# Bootstrap Spinners

# Spinner

- Spinner is also called a **loading indicator**. It is used to display/indicate the loading state of our projects.  
Bootstrap uses a **.spinner** class to create a Spinner.

[Click here for practical](#)

# Bootstrap Contextual Classes

- Contextual classes are used to color table rows (<tr>) or table cells (<td>)

Class/Context	Color	Description
.active	gray	Used to apply the hover color to the table row or table cell
.success	green	Indicates a successful or positive action
.info	cyan	Indicates a neutral informative change or action
.warning	yellow	Indicates a warning that might need attention
.danger	red	Indicates a dangerous or potentially negative action

[Click here for practical](#)

# Bootstrap Layout: Glyphicon

# Glyphicon

- Glyphicons are easily understandable icons and symbols. They look great and hence are widely used in web projects.
- Bootstrap provides total 260 Glyphicons from the Glyphicons Halflings set.
- **Note:** Glyphicons are only supported up to Bootstrap version- 3:  
<https://getbootstrap.com/docs/3.3/component/>

# Bootstrap Layout: Dropdowns

# Bootstrap Dropdowns

- Dropdown menus are toggleable, contextual menus, used for displaying links in a list format.
- It facilitates users to choose one value from a predefined list.
- You have to wrap dropdown menu within the class .dropdown to create Bootstrap Dropdown.

[\*\*Click here for practical\*\*](#)

# Bootstrap Dropdown Divider

- The **class .divider** is used to separate links inside the dropdown menu

[Click here for practical](#)

# Bootstrap Dropdown Header

- The **class .dropdown-header** is used to add headers inside the dropdown menu.

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap Dropdown Disable an item

- Use the **class .disabled** to disable an item in the dropdown menu.

[Click here for practical](#)

# Bootstrap Dropdown

- If you want to open the menu upwards instead of downwards, you need to change the element with class="dropdown" instead of class="dropdown":

[Click here for practical](#)

# Split Button Dropdowns

- It is used to show dropdowns as split buttons. Here we use all contextual classes.

[Click here for practical](#)

# Bootstrap Layout: Labels

# Labels

- Labels are used to provide additional information about something:
- Use the `.label` class, followed by one of the six contextual classes `.label-default`, `.label-primary`, `.label-success`, `.label-info`, `.label-warning` or `.label-danger`, within a `<span>` element to create a label:

# Bootstrap Layout: Badges

# Badges

- Badges scale to match the size of the immediate parent element by using relative font sizing and em units.
- As of v4, badges no longer have focus or hover styles for links

[\*\*\*Click here for practical\*\*\*](#)

# Button

- Badges can be used as part of links or buttons to provide a counter.

**[Click here for practical](#)**

# Bootstrap Layout: Jumbotron

# Jumbotron

- A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.

**[Click here for practical](#)**

# Fluid Jumbotron

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

```
<div class="mt-4 p-5 bg-primary text-white rounded">
  <h1>Jumbotron Example</h1>
  <p>Lorem ipsum...</p>
</div>
```

[\*\*Click here for practical\*\*](#)

# Bootstrap Layout: Page Headers

# Page Headers

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

# Bootstrap Layout: Alerts

# Alerts

- Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight **required** contextual classes (e.g., .alert-success).

[\*\*Click here for practical\*\*](#)

# Link Color

Use the `.alert-link` utility class to quickly provide matching colored links within any alert.

- [Click here for practical](#)

# Additional content

- Alerts can also contain additional HTML elements like headings, paragraphs and dividers.

**[Click here for practical](#)**

# Dismissing

- Using the alert JavaScript plugin, it's possible to dismiss any alert inline.

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap Layout: Progress Bar

# Progress Bar

- Progress components are built with two HTML elements, some CSS to set the width, and a few attributes.
- Bootstrap doesn't use [the HTML5 <progress> element](#), ensuring you can stack progress bars, animate them, and place text labels over them.

[Click here for practical](#)

# Progress Bar

- Use the .progress as a wrapper to indicate the max value of the progress bar.
- Use the inner .progress-bar to indicate the progress so far.
- The .progress-bar requires an inline style, utility class, or custom CSS to set their width.
- The .progress-bar also requires some role and aria attributes to make it accessible.

[\*\*Click here for practical\*\*](#)

# Labels

- Add labels to your progress bars by placing text within the .progress- bar.

[\*\*\*Click here for practical\*\*\*](#)

# Height

- We only set a height value on the .progress, so if you change that value the inner .progress-bar will automatically resize accordingly.

[\*\*\*Click here for practical\*\*\*](#)

# Backgrounds

- Use background utility classes to change the appearance of individual progress bars.

# Multiple Bars

- Include multiple progress bars in a progress component if you need.

[\*\*\*Click here for practical\*\*\*](#)

# Striped

- Add `.progress-bar-striped` to any `.progress-bar` to apply a stripe via CSS gradient over the progress bar's background color.

[Click here for practical](#)

# Animated Stripes

- The striped gradient can also be animated. Add .progress-bar-animated to .progress-bar to animate the stripes right to left via CSS3 animations.

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap Layout: List Group

# List Group

- List groups are a flexible and powerful component for displaying a series of content.

[\*\*\*Click here for practical\*\*\*](#)

# Active Items

- Add .active to a .list-group-item to indicate the current active selection.

**[Click here for practical](#)**

# Disabled Item

- Add .disabled to a .list-group-item to make it *appear* Disabled.

[Click here for practical](#)

# Links and Buttons

- Use <a>s or <button>s to create *actionable* list group items with hover, disabled, and active states by adding .list-group-item-action.

[Click here for practical](#)

# Flush

- Add .list-group-flush to remove some borders and rounded corners to render list group items edge- to-edge in a parent container (e.g., cards).

[\*\*\*Click here for practical\*\*\*](#)

# Numbered

- Add the .list-group-numbered modifier class (and optionally use an <ol> element) to opt into numbered list group items.

[\*\*\*Click here for practical\*\*\*](#)

# Horizontal

- Add .list-group-horizontal to change the layout of list group items from vertical to horizontal across all breakpoints.

[Click here for practical](#)

# Contextual Class

- Use contextual classes to style list items with a stateful background and color.

[\*\*\*Click here for practical\*\*\*](#)

# Badges

- Add badges to any list group item to show unread counts, activity, and more with the help of some utilities.

**[Click here for practical](#)**

# Bootstrap Layout: Panels

# Panels

- A panel in bootstrap is a bordered box with some padding around its content.

[\*\*\*Click here for practical\*\*\*](#)

# Panel Heading

- The .panel-heading class adds a heading to the panel.

[\*\*Click here for practical\*\*](#)

# Panel Footer

The .panel-footer class adds a footer to the panel

[\*\*\*Click here for practical.\*\*\*](#)

# Panel Group

- To group many panels together, wrap a <div> with class .panel-group around them.
- The .panel-group class clears the bottom-margin of each panel

[Click here for practical](#)

# Panel with contextual classes

- To color the panel, use contextual classes (.panel-default, .panel-primary, .panel-success, .panel-info, .panel-warning, or .panel-danger)

[Click here for practical](#)

# Bootstrap Layout: Wells

# Wells

- The .well class adds a rounded border around an element with a gray background color and some padding

# Well Size

- Change the size of the well by adding the .well-sm class for small wells or .well-lg class for large wells

# Bootstrap Plugins: Modal

# Modal

- The Modal plugin is a dialog box/popup window that is displayed on top of the current page

**[Click here for practical](#)**

# Static Backdrop

- When backdrop is set to static, the modal will not close when clicking outside it. Click the button below to try it.

[Click here for practical](#)

# Scrolling Long Content

- When modals become too long for the user's viewport or device, they scroll independent of the page itself. Try the demo below to see what we mean.
- You can also create a scrollable modal that allows scroll the modal body by adding `.modal-dialog-scrollable` to `.modal-dialog`.

[\*\*Click here for practical\*\*](#)

# Vertically Centered

- Add .modal-dialog-centered to .modal-dialog to vertically center the modal.

**[Click here for practical](#)**

# ToolTips

- ToolTips can be placed within modals as needed. When modals are closed, any tooltip within are also automatically dismissed.

[\*\*\*Click here for practical\*\*\*](#)

# 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.

**[Click here for practical](#)**

# Bootstrap Plugins: Scrollspy

# Scrollspy

- Automatically update Bootstrap navigation or list group components based on scroll position to indicate which link is currently active in the viewport.

**[Click here for practical](#)**

# Bootstrap Plugins: Popover

# Popover

- The Popover component is similar to tooltips; it is a pop-up box that appears when the user clicks on an element. The difference is that the popover can contain much more content.

**[Click here for practical](#)**

# Four Directions

- Four options are available: top, right, bottom, and left aligned.

# Dismiss on next click

- Use the focus trigger to dismiss popovers on the user's next click of a different element than the toggle element.

**[Click here for practical](#)**

# Bootstrap Plugins: Collapse

# Collapse

- Collapsibles are useful when you want to hide and show large amount of content.
- The collapse JavaScript plugin is used to show and hide content. Buttons or anchors are used as triggers that are mapped to specific elements you toggle.

[\*\*\*Click here for practical\*\*\*](#)

# Accordion

- Using the [card](#) component, you can extend the default collapse behavior to create an accordion. To properly achieve the accordion style, be sure to use .accordion as a wrapper.

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap Plugins: Carousel

# Carousel

- It works with a series of images, text or custom markup.
- It also includes support for previous/next controls and indicators.
- The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript.

[\*\*\*Click here for practical\*\*\*](#)

# Bootstrap – Login Page

[Click here for practical](#)

# Bootstrap Project

Please find all GitHub links for projects

Foodies project :- [Click here for practical](#)

Psd to html :- [Click here for practical](#)

# Tailwind

Tailwind CSS is a **utility-first CSS framework** designed to build modern websites quickly and efficiently. Unlike traditional CSS frameworks that provide pre-designed components, Tailwind offers low-level utility classes that can be combined to create custom designs directly in your HTML.

# Why use Tailwind CSS?

Tailwind CSS is a utility-first CSS framework that makes it easy to build custom designs without writing custom CSS. It allows you to apply individual utility classes directly in your HTML, which helps create fully customized layouts with minimal effort.

Tailwind provides many utility classes for building responsive and custom designs.

You can easily customize the framework to suit your needs via its configuration file.

# How to use Tailwind CSS?

**Tailwind CDN:** The link to the Tailwind CSS CDN is included in the <head> section of the HTML document to load the necessary CSS styles.

- Tailwind helps build seamless mobile-first and responsive designs.

**[Click here for practical](#)**

# Typography

- font-family
- Font-style
- font-weight
- text-decoration-line
- text-decoration-style
- text-decoration-color
- text-wrap

**[Click here for practical](#)**

# Flexbox

- flex-wrap
- flex-direction
- flex-basis
- z-index

[Click here for practical](#)

# Border

- border-radius
- border-width
- border-color
- border-style
- outline-width
- outline-color

**[Click here for practical](#)**

# Spacing

- Margin
- padding
- space

**[Click here for practical](#)**

# Grid layout

- grid-template-rows
- grid-row
- grid-auto-columns
- grid-auto-rows
- grid-column
- Grid-template-columns

**[Click here for practical](#)**

# TailWind many more properties

tailwind use for component, utilities and many more to easy understand property.

[\*\*\*Click here for practical\*\*\*](#)

# Module-8

## JavaScript Essentials And Advanced

# An Introduction to JavaScript

- **What is JavaScript?**
- JavaScript was initially created to “make web pages alive”.
- The programs in this language are called scripts. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.

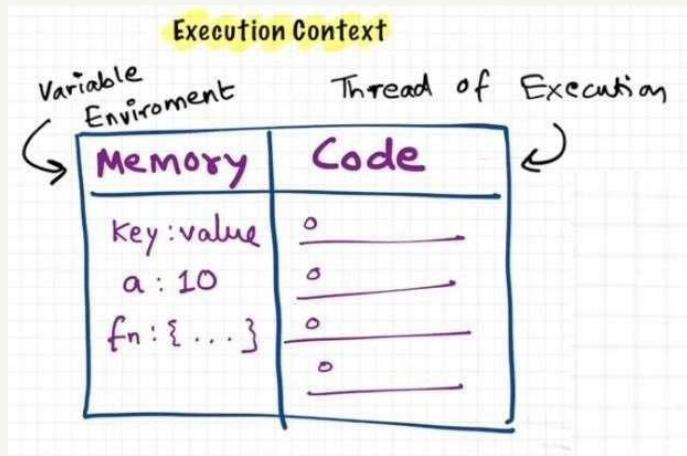
# An Introduction to JavaScript

In this aspect, JavaScript is very different from another language called Java.

- JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

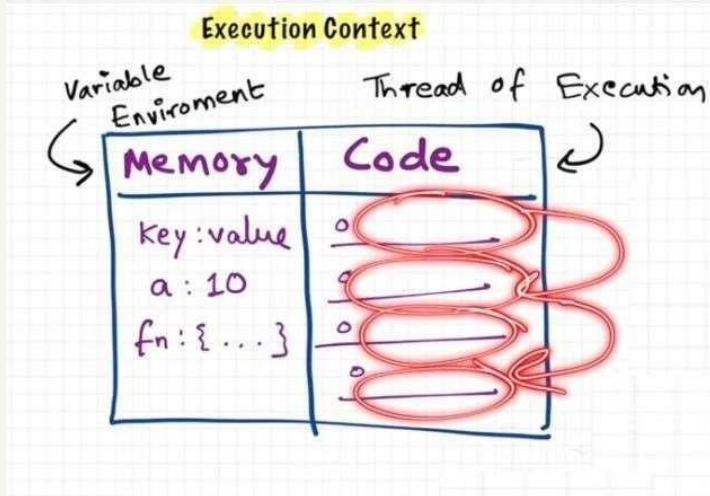
# An Introduction to JavaScript

Everything in JavaScript Happens inside an execution context

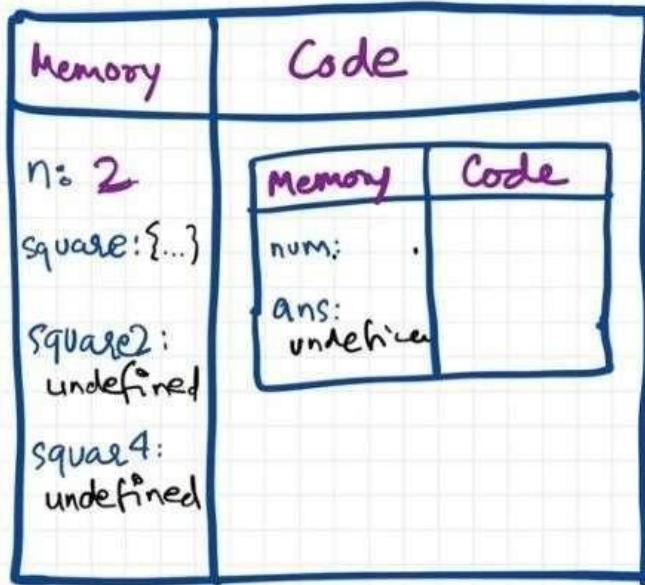


# An Introduction to JavaScript

JavaScript is asynchronous Single-threaded Language



# Execution



```
1 var n =2;
2 function square (num) {
3     var ans = num * num;
4     return ans;
5 }
6 var square2 = square(n);
7 var square4 = square(4);
```

# Why is it called JavaScript?

- When JavaScript was created, it initially had another name: “LiveScript”. But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help.
- But as it evolved, JavaScript became a fully independent language with its own specification called
- **ECMAScript**(European Computer Manufacturers Association.), and now it has no relation to Java at all.

- Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.
- The browser has an embedded engine sometimes called a “JavaScript virtual machine”.
- Different engines have different “codenames”. For example:
- V8 – in Chrome and Opera.
- SpiderMonkey – in Firefox.
- ...There are other codenames like “Trident” and “Chakra” for different versions of IE, “ChakraCore” for Microsoft Edge, “Nitro” and “SquirrelFish” for Safari, etc.
- The terms above are good to remember because they are used in developer articles on the internet. We’ll use them too. For instance, if “a feature X is supported by V8”, then it probably works in Chrome and Opera.

# THE EVOLUTION OF JAVASCRIPT

- JavaScript is currently being used in more than 94% of websites. That means millions of web developers are using JavaScript for web application development. The ease of use and widespread adoption of this programming language is exactly why it continues to become more important to web developers.

# THE EVOLUTION OF JAVASCRIPT

- The Ubiquity of Web Browsers – Can you think of a time or place where you aren't within one click of an internet browser? Nope. The web is available everywhere and this makes JavaScript more important and more commonplace. Plus, the widespread adoption ensures dynamic content gets displayed the way it was intended.

# THE EVOLUTION OF JAVASCRIPT

- **JavaScript Has More Applications** – Thanks to innovations like Node.js and ReactJS, JavaScript is useful for both front and back end web development. By using cross-platform runtime engines to write server-side code, JavaScript has even more applications and increases efficiencies for web developers.

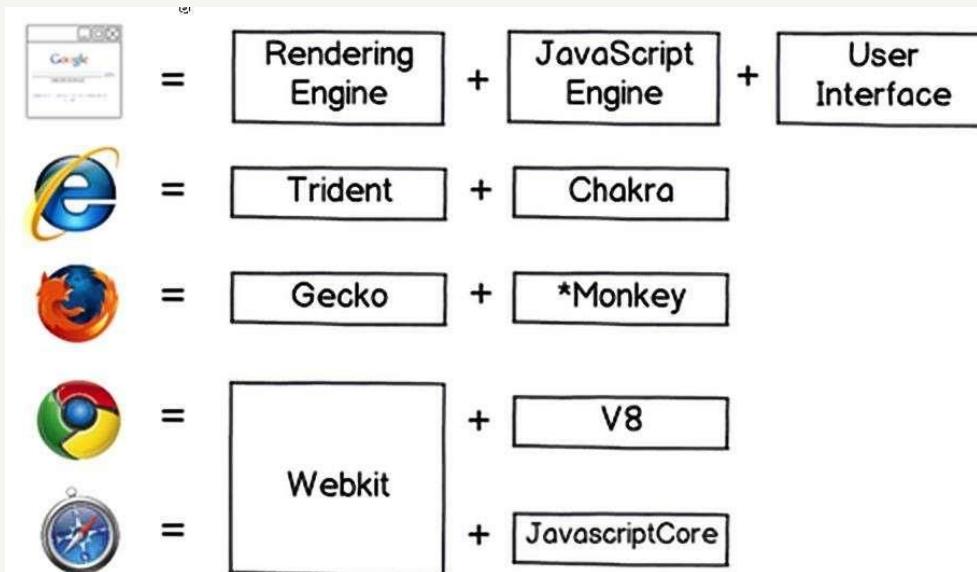
# THE EVOLUTION OF JAVASCRIPT

- **The Ubiquity of Web Browsers** – Can you think of a time or place where you aren't within one click of an internet browser? Nope. The web is available everywhere and this makes JavaScript more important and more commonplace. Plus, the widespread adoption ensures dynamic content gets displayed the way it was intended.

# THE EVOLUTION OF JAVASCRIPT

- **JavaScript Has More Applications** – Thanks to innovations like Node.js and ReactJS, JavaScript is useful for both front and back end web development. By using cross-platform runtime engines to write server-side code, JavaScript has even more applications and increases efficiencies for web developers.

# JS engines



# How do engines work?

- Engines are complicated. But the basics are easy.
- The engine (embedded if it's a browser) reads ("parses") the script.
- Then it converts ("compiles") the script to the machine language.
- And then the machine code runs, pretty fast.
- The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

# What can in-browser JavaScript do?

- JavaScript's capabilities greatly depend on the environment it's running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.
- Add new HTML to the page, change the existing content, modify styles.

# What can in-browser JavaScript do?

- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so- called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

# What CAN'T in-browser JavaScript do?

- JavaScript's abilities in the browser are limited for the sake of the user's safety. The aim is to prevent an evil webpage from accessing private information or harming the user's data.
- Examples of such restrictions include:

# What CAN'T in-browser JavaScript do?

- JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS system functions.
- Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an `<input>` tag.

# What CAN'T in-browser JavaScript do?

- There are ways to interact with camera/microphone and other devices, but they require a user's explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the NSA.

# What makes JavaScript unique?

- There are at least three great things about JavaScript:
  - Full integration with HTML/CSS.
  - Simple things are done simply.
  - Support by all major browsers and enabled by default.
- JavaScript is the only browser technology that combines these three things.

# What makes JavaScript unique?

- That's what makes JavaScript unique. That's why it's the most widespread tool for creating browser interfaces.
- That said, JavaScript also allows to create servers, mobile applications, etc.

# Languages “over” JavaScript

- The syntax of JavaScript does not suit everyone's needs.  
Different people want different features.
- That's to be expected, because projects and requirements  
are different for everyone.
- So recently a plethora of new languages appeared,  
which are transpiled (converted) to JavaScript  
before they run in the browser.
- Modern tools make the transpilation very fast and  
transparent, actually allowing developers to code in  
another language and auto-converting it “under the hood”.

# Languages “over” JavaScript

- Examples of such languages:
- CoffeeScript is a “syntactic sugar” for JavaScript. It introduces shorter syntax, allowing us to write clearer and more precise code. Usually, Ruby devs like it.
- TypeScript is concentrated on adding “strict data typing” to simplify the development and support of complex systems. It is developed by Microsoft.

# Languages “over” JavaScript

- Flow also adds data typing, but in a different way.  
Developed by Facebook.
- Dart is a standalone language that has its own engine that runs in non-browser environments (like mobile apps), but also can be transpiled to JavaScript. Developed by Google.
- There are more. Of course, even if we use one of transpiled languages, we should also know JavaScript to really understand what we’re doing.

# Summary

- JavaScript was initially created as a browser-only language, but is now used in many other environments as well.
- Today, JavaScript has a unique position as the most widely-adopted browser language with full integration with HTML/CSS.
- There are many languages that get “transpiled” to JavaScript and provide certain features. It is recommended to take a look at them, at least briefly, after mastering JavaScript.

# JavaScript Uses in Web Designing

- |    |                     |
|----|---------------------|
| 1  | Websites            |
| 2  | Web Applications    |
| 3  | Presentations       |
| 4  | Server Applications |
| 5  | Web Servers         |
| 6  | Games               |
| 7  | Art                 |
| 8  | Smartwatch Apps     |
| 9  | Mobile Apps         |
| 10 | Flying Robots       |



JavaScript

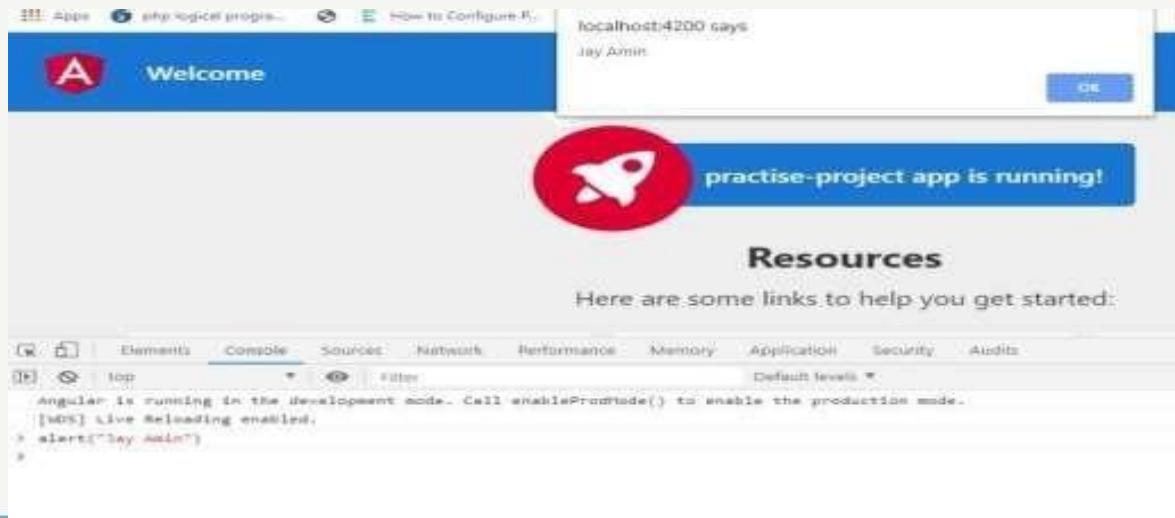
## Application

# Developer console

F12

or

# Right click inspect element



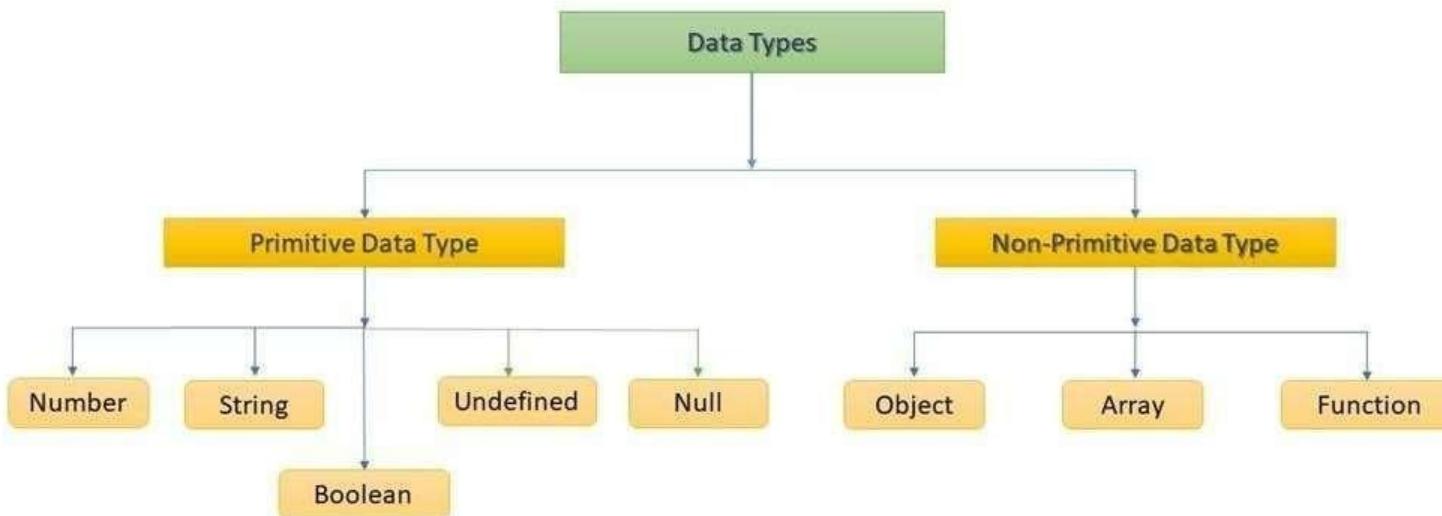
# Developer console

- But in the browser, users don't see errors by default. So, if something goes wrong in the script, we won't see what's broken and can't fix it.
- Most developers lean towards Chrome or Firefox for development because those browsers have the best developer tools. Other browsers also provide developer tools, sometimes with special features, but are usually playing "catch-up" to Chrome or Firefox. So most developers have a "favorite" browser and switch to others if a problem is browser-specific.

# Developer console

- Developer tools are potent; they have many features. To start, we'll learn how to open them, look at errors, and run JavaScript commands.

# Data Types



# Javascript place in Html

## Placing <script> in the <head>:

- **How it works:** The script is loaded before the HTML content, meaning that the browser will download and execute the script before rendering the body content.
- **Drawback:** This can block the rendering of the page because the browser must wait for the script to load and execute before displaying any content. This could lead to slower page load times, especially for larger scripts.
- **When to use:** Use this method when the script needs to execute before any content is displayed (e.g., setting up variables, configurations, or making requests that are required for the initial display).

# Javascript place in Html

## Placing <script> in the <body> :

- **How it works:** The script is loaded after the HTML content is loaded. This means the browser can display the page content first while the script loads in the background.
- **Advantage:** This method improves page load performance since the browser can render the content while the script is still downloading. Once the HTML is rendered, the script is executed.
- **Best practice:** This is the most common approach because it ensures that the page's content appears quickly, and the script executes afterward without blocking rendering.

# Using the **async** and **defer** attributes:

- **async attribute:** If placed in the `<head>`, the script will be fetched in parallel to the HTML parsing, but it will execute as soon as it's downloaded, possibly before the page has finished rendering.
- **defer attribute:** If placed in the `<head>`, it ensures that the script is executed only after the HTML document has been fully parsed (but before the `DOMContentLoaded` event is fired).

```
<script src="script.js" defer></script>
```

# What is a JavaScript Loop?

- **Entry Controlled Loops:** Any loop where we check the test condition before entering the loop is an entry controlled loop. In these loops, the test condition determines whether the program will enter the loop or not. These include for, while, etc.
- **Exit Controlled Loops:** Any loop where we check the test condition after the statements are executed once is an exit controlled loop. In these loops, the test condition determines whether or not the program will exit the loop. This category includes do...while loop.

# JavaScript Loops

A **loop** is a programming construct that repeats a block of code multiple times based on a specific condition or set of conditions. In JavaScript, loops allow you to automate repetitive tasks without needing to write the same code repeatedly.

# Why Do We Use Loops?

Loops are essential for handling tasks that require repetition. They are used for:

1. **Iterating through collections:** You might want to process each item in an array, object, or other iterable collections.
2. **Repeating a task:** When you need to execute the same action multiple times, like summing numbers, applying a function to each element, or making multiple attempts to meet a certain condition.
3. **Efficiency:** Loops reduce the need for repetitive code, making your program cleaner and more maintainable.

# Types of loops

1. For Loop
2. While Loop
3. Do-While Loop
4. For...in Loop
5. For...of Loop

# For loop

The **for loop** is used when you know the exact number of times you want to repeat a block of code.

syntax

```
for (initialization; condition; update) {  
    // Code to be executed  
}
```

# For loop

**Initialization:** You define and initialize a variable (often a counter) that controls the loop.

**Condition:** This is a test expression that is evaluated before each iteration. If the condition is **true**, the loop continues.

**Update:** The loop variable is updated (usually incremented or decremented) after every iteration.

**[Click here for practical](#)**

# while loop

The **while loop** runs as long as the given condition is **true**. It checks the condition **before** each iteration.

Syntax:

```
while (condition)
{
    // Code to be executed
    // updation
}
```

[\*\*\*Click here for practical\*\*\*](#)

# do.. while loop

The **do...while loop** is similar to the **while** loop, but the key difference is that the condition is checked **after** the code is executed. This guarantees that the code will run at least once.

Syntax:

```
do
{
    // Code to be executed
} while (condition);
```

[\*\*Click here for practical\*\*](#)

# For .. in loop

The **for...in loop** is used to loop through the **keys** (or properties) of an **object**. It gives you access to the property names, but not the values directly.

Syntax:

```
for (let key in object) {  
    // Code to be executed  
}
```

# For .. in loop

- This loop is specifically used for iterating over **objects** (not arrays).
- On each iteration, the **key** variable holds the name of the current property

**[Click here for practical](#)**

# For .. of loop

- The **for...of loop** is used to loop through **values** in an **iterable** (like an array, string, or other iterable objects).

Syntax

```
for (let value of iterable) {  
    // Code to be executed  
}
```

# For .. of loop

Unlike `for...in`, which gives you the keys, `for...of` gives you the **values** of the iterable.

This is commonly used to iterate over arrays or strings.

**[Click here for practical](#)**

# Jumping Statement

**jumping statements** control the flow of execution by either stopping the current loop or skipping the current iteration and continuing with the next one. The two primary jumping statements are **break** and **continue**.

These statements are used within loops or switch cases to alter the normal flow of execution.

# Types of Jumping statements

1. Break Statement
2. Continue Statement
3. Labeled Statement
4. return
5. throw

# break Statement

The break statement is used to exit a loop or a switch statement prematurely, meaning it will stop the loop or switch case and transfer control to the next statement after the loop or switch.

## Usage of break:

- In loops: The break statement stops the loop entirely, regardless of whether the loop condition is true or not.
- In switch: The break statement exits the switch case once the matched block of code is executed.

**[Click here for practical](#)**

# Continue Statement

The continue statement is used to skip the current iteration of a loop and move to the next iteration. It doesn't stop the loop entirely, but it prevents the rest of the code inside the loop from executing for the current iteration.

Usage of continue:

- In loops: It can be used inside for, while, or do...while loops to skip certain iterations based on a condition.
- It does not work in switch statements: You cannot use continue to skip to the next case in a switch.

**[Click here for practical](#)**

# Labeled statement

- A labeled statement in JavaScript is a way to assign a name (a label) to a specific statement or block of code. This label can then be used with control flow statements like break or continue to control the flow of execution more specifically, especially in complex loops or nested blocks

**[Click here for practical](#)**

# Return statement

The return statement is used to exit a function and optionally return a value to the caller. It stops the execution of the function and can send a value back to the function's caller.

## Usage of return:

- In functions: The return statement terminates the function's execution and optionally returns a value.
- In loops: A return inside a function (even if inside a loop) causes the function to exit immediately, so the loop will not continue.

[\*\*Click here for practical\*\*](#)

# Throw statement

The `throw` statement is used to **raise an exception** in JavaScript. It allows you to create custom error messages or objects that can be caught by a `try...catch` block for exception handling. When `throw` is executed, the normal flow of the program is interrupted, and control is transferred to the nearest `catch` block.

- **Exception Handling:** `throw` is used to raise exceptions, which can then be handled using `try...catch` blocks.
- You can throw an error message or custom error object.

[\*\*\*Click here for practical\*\*\*](#)

# Function

- A **function** in JavaScript is a block of reusable code designed to perform a specific task. Functions help in organizing code into logical blocks, making it more readable, reusable, and maintainable. JavaScript functions allow you to group a set of instructions together, and you can call these instructions whenever needed.
- In JavaScript, functions can accept inputs, perform some operations, and return a result.

# Function

- Function declaration

```
function functionName(parameter1, parameter2, ...) {  
    // Function body: code to execute  
    // You can access parameters and write logic here  
}  
  
functionName(arg1,arg2)
```

**[Click here for practical](#)**

# Function Expression

- A function expression defines a function as part of an expression. The function can be anonymous (without a name) and assigned to a variable.

```
const functionName = function(parameter1, parameter2, ...) {  
    // Function body  
};
```

**[Click here for practical](#)**

# Arrow Function

## **Arrow Function (ES6)**

Arrow functions, introduced in ECMAScript 6 (ES6), provide a shorter syntax for writing functions. The syntax is:

```
const functionName = (parameter1, parameter2) => {  
    // code to be executed  
};
```

[\*\*\*Click here for practical\*\*\*](#)

# Higher-Order Functions

A higher-order function is a function that takes one or more functions as arguments, returns a function, or both.

These kinds of functions allow for more flexibility and abstraction in your code, and are commonly used in functional programming.

Common examples of higher-order functions in JavaScript include `map()`, `filter()`, `reduce()`, and `forEach()`.

## 1) **Higher-Order Function that Takes a Function as an Argument**

```
function higherOrderFunction(fn) {  
    return fn();  
}  
  
function sayHello() {  
    return "Hello, World!";  
}  
  
console.log(higherOrderFunction(sayHello)); // "Hello, World!"
```

**[Click here for practical](#)**

## 2) Higher-Order Function that Returns Another Function

```
function multiplyBy(factor) {  
    return function(number) {  
        return number * factor;  
    };  
}
```

```
const double = multiplyBy(2);  
console.log(double(5)); // 10
```

```
const triple = multiplyBy(3);  
console.log(triple(5)); // 15
```

[\*\*Click here for practical\*\*](#)

# Function Scope and Closures

**Scope** refers to the visibility of variables inside different parts of a program.

**Closures** are functions that "remember" the scope in which they were created, even after the outer function has finished executing

# Scope

scope refers to the accessibility or visibility of variables, functions, and objects in some particular part of the code.

JavaScript uses lexical scoping, which means that the scope of a variable is determined by its position in the code at the time it's written.

# Types of scope

- 1) Global Scope
- 2) Function Scope
- 3) Block Scope
- 4) Lexical Scope

# Global Scope

Variables declared outside of any function or block are said to have global scope, which means they are accessible throughout the entire script.

**[Click here for practicals](#)**

# Global variables and Memory management

Global variables persist throughout the lifetime of the page or script. This can lead to memory issues if too many global variables are used, as they are never garbage-collected until the script ends. It's a good idea to minimize the use of global variables when possible.

// If you have many global variables, they will stay in memory throughout the page's lifecycle

```
var global1 = "Global variable 1";  
var global2 = "Global variable 2";
```

// and so on...

# Global Scope in Modules (ES6)

Starting from ES6 (ECMAScript 2015), JavaScript introduced **modules**, which allow you to split your code into smaller, reusable pieces. Code inside modules is **not** in the global scope by default.

Each module has its own scope, which prevents polluting the global namespace.

# Global Scope in Modules (ES6)

```
// file1.js
```

```
let moduleVariable = "I am local to this module";
export { moduleVariable };
```

```
// file2.js
```

```
import { moduleVariable } from './file1.js';
console.log(moduleVariable); // Output: I am local to this module
```

# Function Scope

Function scope means that variables declared inside a function are only accessible within that function.

These variables are **local** to the function and are not visible or accessible from outside the function.

[\*\*Click here for practical\*\*](#)

# Function Scope

## Function Scope with **var** (ES5 and below)

Before ES6, the var keyword was used to declare variables.

var has function scope, which means that variables declared with var are scoped to the function, regardless of where in the function they are declared (even if they are declared inside a block like if or for).

[\*\*Click here for example\*\*](#)

# Function Scope

## **let and const (ES6 and beyond)**

In ES6 (ECMAScript 2015) and later, JavaScript introduced let and const keywords to declare variables. Unlike var, both let and const have **block scope**. This means that variables declared with let or const are limited to the block (such as inside an if statement or loop) in which they are declared.

**[Click here for example](#)**

# Block Scope

**block scope** refers to the concept that variables declared within a block (i.e., within curly braces `{}`) are only accessible within that block. This behavior is enforced with the `let` and `const` keywords (introduced in ES6), and it applies to any block, such as loops, conditionals, or even standalone blocks.

**[Click here for example](#)**

# Block Scope

## let and const in block scope

Before ES6, JavaScript only had function scope with `var`.

Variables declared with `var` inside a block (like an `if` statement or a loop) were still accessible outside of the block, which led to potential issues.

ES6 introduced **block-scoping** with `let` and `const`.

[Click here for let example](#)

[Click here for const example](#)

# Block Scope

## In loop statements and conditional statements

One of the most common scenarios where block scoping is important is inside loops and conditionals. With `let` and `const`, you can have **block-scoped variables** within loops or `if` statements, preventing unwanted side effects when variables are reused.

**[Click here for let example](#)**

# Why is Block Scope important ?

Block scope helps solve several issues that arise from function-scoped variables (`var`):

- **Avoiding Variable Leaks:** Block scoping prevents variables from being accidentally accessed outside of their intended scope. This makes your code more predictable and safer by reducing the risk of name conflicts and unintended variable sharing.
- **Improved Control in Loops:** Using `let` and `const` in loops ensures that the loop variable is not accessible outside the loop, making it easier to reason about the code and avoid errors caused by reusing loop variables.
- **Safer Conditional Logic:** Block scoping ensures that variables inside an `if` or `else` block are not accidentally used outside of the block, which is helpful when writing complex conditional logic.

# lexical scope

Lexical scope (also known as static scope) is the scope (visibility) of variables that is determined at the time the code is written, not when it's executed. This means that JavaScript uses the position of a function in the code to determine which variables are accessible to it.

In JavaScript, functions have access to variables that are declared within them and outside them in the scope chain (lexically), based on where the function is defined. Importantly, the scope is determined by where the function is declared, not by where the function is called.

# lexical scope

**Inner functions** can access variables from their **outer functions** (lexically scoped).

**Outer functions** cannot access variables from their **inner functions**.

**Variables** are looked up from the current scope, and if not found, from its outer (parent) scopes

[\*\*\*Click here for example\*\*\*](#)

## lexical scope

# Outer Functions Cannot Access Inner Variables

**innerFunction()** can access **outerVar** because it is inside **outerFunction()**.

But **outerFunction() cannot** access **innerVar** because **innerVar** is declared inside **innerFunction()**, and variables in inner functions are not accessible to outer functions.

[Click here for practical](#)

# closure

## What is a Closure?

A **closure** is a function that **remembers** its **lexical environment**, i.e., it "remembers" the variables that were in scope when the function was created, even if the function is executed outside of that scope.

In simpler terms, a closure is a function that has access to its own scope, the outer function's scope, and the global scope, even after the outer function has finished executing.

# closure

## Key Characteristics of Closures:

1. A closure allows a function to access variables from its outer function even after the outer function has finished executing.
2. A closure can "remember" the environment in which it was created (this includes variables that were in scope at the time of creation).

**[Click here for practical](#)**

# closure

## How Closures Work

When you create a function inside another function, the inner function has access to the variables from the outer function.

If you return the inner function, the inner function retains access to the outer function's variables, even after the outer function has executed.

This is because JavaScript keeps track of the variables in the scope where the function was defined (this is the closure).

**[Click here for practical](#)**

# Array

An **array** in JavaScript is a data structure used to store multiple values in a single variable. Arrays are often used to group related data together.

Unlike traditional variables which hold a single value, arrays allow you to store collections of values, such as numbers, strings, or even other arrays or objects.

# Key Characteristics of Arrays

**Indexed Collection:** Arrays in JavaScript are zero-indexed, meaning the first element is at index 0, the second element at index 1, and so on.

**Ordered:** The elements in an array are ordered, meaning the position of each element matters.

**Dynamic Size:** JavaScript arrays are dynamically sized, meaning the size of an array can change during runtime (i.e., you can add or remove elements).

**Heterogeneous:** An array can contain elements of different types, such as numbers, strings, objects, or even other arrays.

# Creating an Array

Arrays in JavaScript can be created in two main ways:

## 1. Using Array Literals (Recommended)

```
let fruits = ["Apple", "Banana", "Cherry"];
```

This is the most common and preferred way to create an array. You enclose the elements in square brackets ([]), separated by commas.

[Click here for practical](#)

# Creating an Array

## 2. Using the **Array Constructor**

```
let fruits = new Array("Apple", "Banana", "Cherry");
```

This creates an array, but the syntax is less concise and less common than using array literals.

[\*\*Click here for practical\*\*](#)

# Array Methods

`push()` - Add an element to the end

**[Click here for practical](#)**

`pop()` - Remove the last element

**[Click here for practical](#)**

`shift()` - Remove the first element

**[Click here for practical](#)**

`unshift()` - Add an element to the beginning

**[Click here for practical](#)**

# Array Methods

`length` - Get the number of elements

[\*\*Click here for practical\*\*](#)

`indexOf()` - Find the index of an element

[\*\*Click here for practical\*\*](#)

`splice()` - Add or remove elements at specific positions

[\*\*Click here for practical\*\*](#)

`slice()` - Create a shallow copy of a portion of the array

[\*\*Click here for practical\*\*](#)

# Array Methods

forEach() - Loop through each element

**[Click here for practical](#)**

map() - Create a new array by applying a function to each element

**[Click here for practical](#)**

filter() - Filter elements based on a condition

**[Click here for practical](#)**

reduce() - Reduce the array to a single value

**[Click here for practical](#)**

# Multidimensional Arrays

JavaScript arrays can also be multidimensional, meaning an array within an array:

[Click here for practical](#)

# Array Destructuring

JavaScript allows you to use destructuring to unpack values from arrays:

[Click here for practical](#)

# Array.from()

Array.from() is a built-in method in JavaScript that creates a new, shallow-copied array instance from an array-like or iterable object

**Converting a string to an array:** [Click here for practical](#)

**Converting set into array :** [Click here for practical](#)

**Using of mapFunction :** [Click here for practical](#)

**Array-like Objects:** These include objects that have a length property and indexed elements, like arguments, NodeList (from DOM queries), or strings.

[Click here for practical](#)

# Object

An object is a collection of related data and/or functionality. It is one of the most fundamental data structures and allows you to group together various values under a single entity. These values can be of any data type, including strings, numbers, arrays, and even other objects.

# properties

Properties (Key-Value Pairs): Objects consist of properties, which are key-value pairs. The key is a string (also called the property name), and the value can be any data type (string, number, array, function, etc.).

```
const person = {  
    name: "pari",      // key is 'name' and value is pari  
    age: 20,          // key is 'age' and value is 30  
    isEmployed: true  // key is 'isEmployed' and value is true  
};
```

[\*\*Click here for practical\*\*](#)

# methods

In addition to data, objects can also have methods. Methods are functions that are associated with an object and typically operate on the object's properties.

```
const person = {  
    name: "shreya",  
    age: 30,  
    greet: function() {  
        console.log("Hello, " + this.name);  
    }  
};  
person.greet(); // Output: Hello, shreya
```

**[Click here for practical](#)**

# Creating Object

There are 3 ways to create object

## 1) Using Object Literals (Recommended for most cases):

```
const mobile = {  
    brand: "Samsung",  
    model: "s24",  
    year: 2024  
};  
  
console.log(mobile);
```

[Click here for practical](#)

## 2) Using the new Object() Syntax:

```
const mobile = new Object();
mobile.brand = "Samsung";
mobile.model = "S24";
mobile.year = 2024;

console.log(mobile);
```

[\*\*\*Click here for practical\*\*\*](#)

### 3) Using a Constructor Function (typically for creating multiple instances of similar objects):

```
function Mobile(brand, model, year) {  
    this.brand = brand;  
    this.model = model;  
    this.year = year;  
}
```

```
const myMobile = new Mobile("Samsung", "S24", 2024);  
console.log(myMobile);
```

**[Click here for practical](#)**

# JavaScript Objects

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

All JavaScript values, except primitives, are objects.

# Accessing Objects

## Dot Notation

Dot notation is the most common and straightforward way to access an object's properties

e.g. `console.log(person.name);`

## Bracket Notation

Bracket notation is often used when the property name is dynamic (e.g., stored in a variable) or when the property name contains spaces or special characters.

e.g. `console.log(person["name"]);`

**[Click here for practical](#)**

# Add - Delete Properties

## ***Adding Properties***

```
person.country = "USA";
```

```
console.log(person.country); // "USA"
```

## ***Delete Properties***

```
delete person.age;
```

```
console.log(person.age); // undefined
```

# Object Destructuring

Destructuring is a convenient way to extract values from an object into variables.

**[Click here for practical](#)**

# Object Methods

- `Object.keys(obj)` – Returns an array of the object's property names (keys).
- `Object.values(obj)` – Returns an array of the object's property values.
- `Object.entries(obj)` – Returns an array of arrays, each containing a key-value pair.

**[Click here for practical](#)**

# Object.freeze() and Object.seal()

JavaScript provides methods to make objects immutable or prevent new properties from being added.

- **Object.freeze(obj)** – Freezes an object, preventing new properties from being added and existing properties from being modified or deleted.
- **Object.seal(obj)** – Seals an object, preventing new properties from being added but allows existing properties to be modified.

[\*\*\*Click here for practical\*\*\*](#)

# Prototype Chain

Every JavaScript object has an internal property called `[[Prototype]]`, which is used for inheritance. You can access the prototype of an object using `Object.getPrototypeOf()`.

[Click here for practical](#)

# String in JS

A string is a sequence of characters used to represent text. Strings are one of the most commonly used data types in JavaScript and are integral to web development tasks such as manipulating text, processing user input, and working with data formats like JSON, HTML, etc.

# Create String

There are two ways to define string

- 1) ***Using String Literal (Single or Double Quotes)*** - [practical](#)

string by enclosing a sequence of characters in single (' ) or double (" ) quotes.

- 2) ***Using Template Literals (Backticks)*** - [practical](#)

Template literals, introduced in ECMAScript 6 (ES6), allow you to create strings with more flexibility, such as including expressions inside the string. Template literals are enclosed by backticks () .

# String Properties and Methods

**Accessing characters :** can access individual characters in a string using bracket notation.

**length:** Returns the number of characters in the string.

**toUpperCase()** : Converts a string to uppercase:

**toLowerCase()** Converts a string to lowercase:

**substring(start, end)** Extracts a part of a string:

**replace oldValue, newValue**) - Replaces part of a string:

**split(separator)** - Splits a string into an array of substrings:

**trim()** - Removes whitespace from both ends of a string:

**[Click here for practical](#)**

# **DOM**

# **Document Object Model**

## DOM

JavaScript is a programming interface for web documents. It represents the structure of a web page as a tree of objects, where each object corresponds to a part of the page (like elements, attributes, text, etc.).

This allows JavaScript to interact with and manipulate the content, structure, and style of web pages dynamically.

**Document:** The `document` object represents the entire HTML document and serves as the entry point for accessing and manipulating elements within the page.

**Node:** The DOM represents every part of an HTML document as a **node** (an element, attribute, or piece of text). The most common types of nodes are:

- **Element nodes** (e.g., `<div>`, `<p>`)
- **Text nodes** (e.g., the content inside `<p>`)
- **Attribute nodes** (e.g., `class="myClass"` in an HTML element)

**DOM Tree Structure:** The structure of the DOM is hierarchical, and you can think of it as a tree where:

- The **document** is the root.
- Each element is a node branching off from the root or from other nodes.
- Text and attributes are also nodes that are attached to their respective elements.

**DOM Manipulation:** JavaScript can interact with the DOM to modify the page dynamically, such as adding or removing elements, changing styles, handling events, and much more.

**[Click Here for practicals](#)**

## getElementById()

- Purpose: Selects an element by its unique id attribute.
- Description: This is the most efficient method when selecting a single element because id attributes are required to be unique within a page.

### Syntax :

```
var element = document.getElementById("id");
```

- Note: getElementById() returns a single element, even though multiple elements with the same id would be considered invalid according to HTML standards.

## getElementsByClassName( )

- Purpose: Selects all elements with a specified class name.
- Description: This method returns a live HTMLCollection (a live list that automatically updates when the document changes) of all elements with the given class name. Multiple elements can have the same class name.

### Syntax :

```
var elements = document.getElementsByClassName("className");
```

[\*\*Click here for example\*\*](#)

## getElementsByName( )

- Purpose: Selects all elements with a specified tag name (e.g., <div>, <p>, <a>).
- Description: Similar to getElementsByClassName(), this method returns a live HTMLCollection of elements with the specified tag name.

Syntax:

```
var elements = document.getElementsByTagName("tagName")
```

## Promises

JavaScript Promises make handling asynchronous operations like API calls, file loading, or time delays easier. Think of a Promise as a placeholder for a value that will be available in the future. It can be in one of three states

Pending: The task is in the initial state.

Fulfilled: The task was completed successfully, and the result is available.

Rejected: The task failed, and an error is provided.

**[Click here for practical](#)**

## Promises

`resolve(value)`: Marks the promise as fulfilled and provides a result.

`reject(error)`: Marks the promise as rejected with an error.

**[Click here for practical](#)**

## Promises

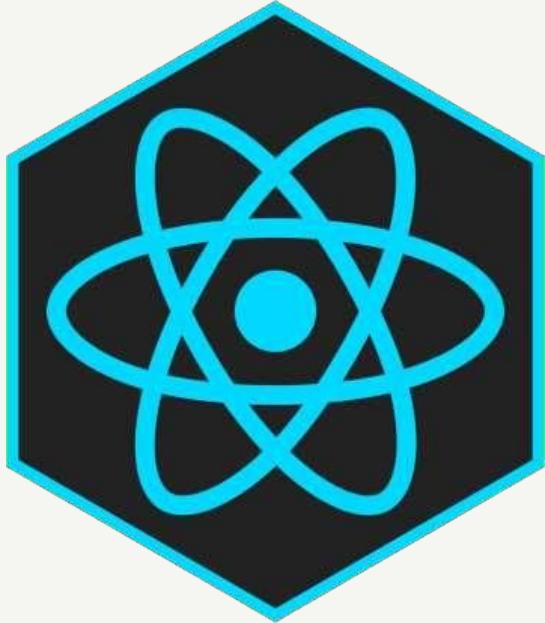
JavaScript Promises make handling asynchronous operations like API calls, file loading, or time delays easier. Think of a Promise as a placeholder for a value that will be available in the future. It can be in one of three states

Pending: The task is in the initial state.

Fulfilled: The task was completed successfully, and the result is available.

Rejected: The task failed, and an error is provided.

**[Click here for practical](#)**



# ReactJS

# Module-9

# React - Components, State, Props

# Introduction

React.js (often just called React) is an open-source JavaScript library used for building user interfaces, particularly for single-page applications where you need fast, dynamic updates without reloading the whole page.

It was developed by Facebook and is widely used for creating interactive and reusable UI components.

# What is ReactJS

ReactJS is an open source JavaScript library used to develop User Interfaces.



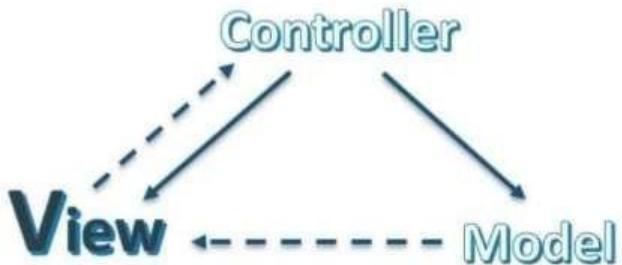
facebook.



ReactJS was introduced by Facebook on May, 2013. It was open sourced in March, 2015.

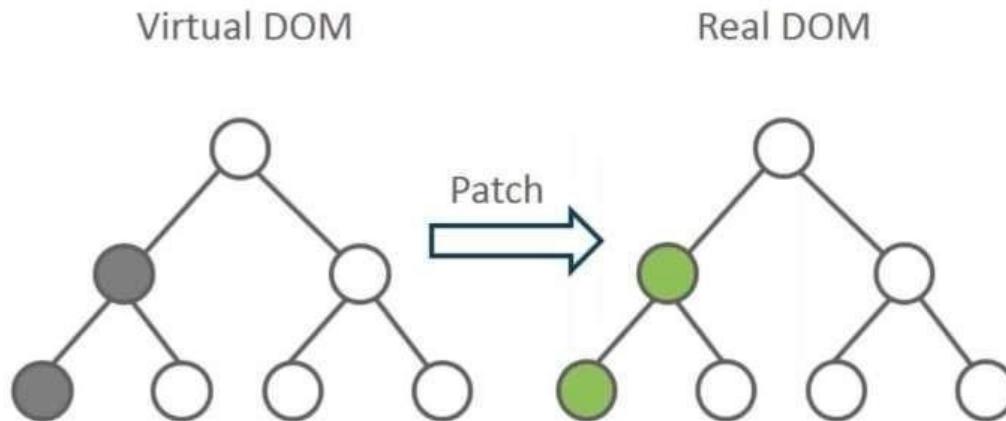
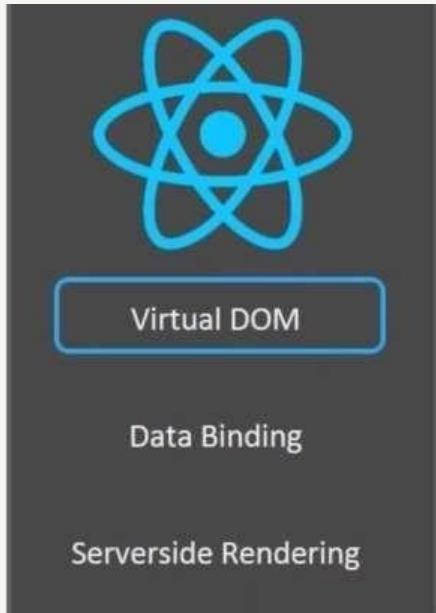
# What is ReactJS

ReactJS is concerned with the components that utilizes the expressiveness of JavaScript with a HTML – like template syntax.

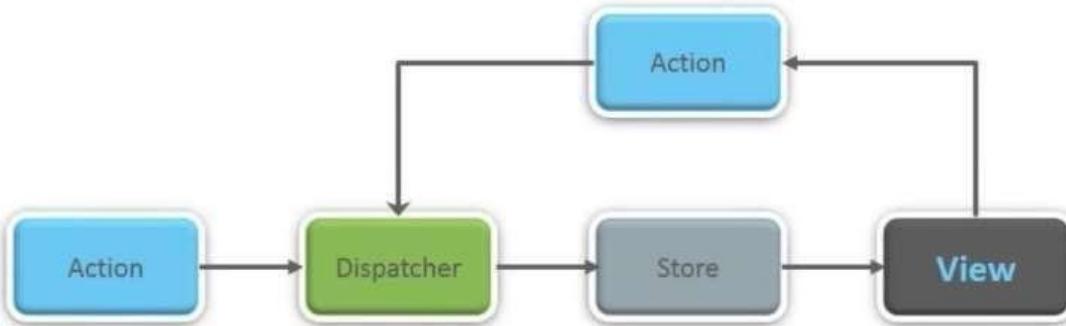
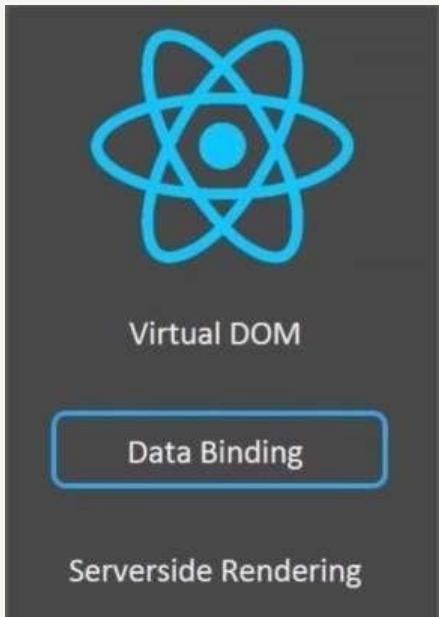


It is basically the View in MVC  
(Model-View-Controller)

# 3D Aspects Of React Virtual Dom

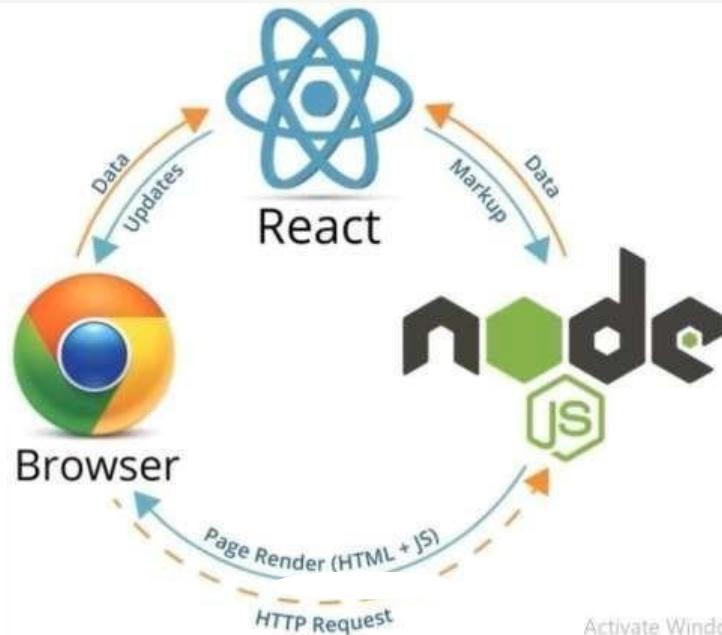
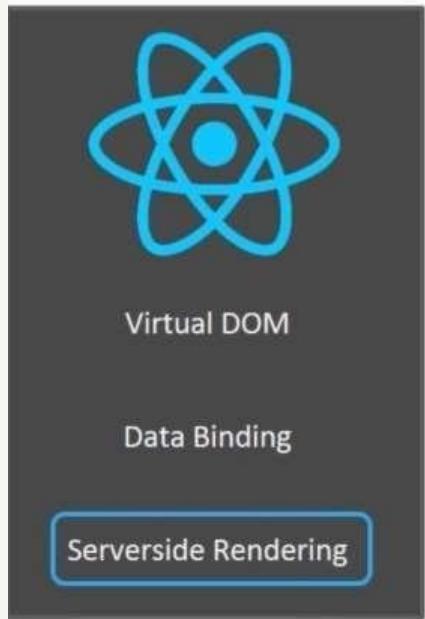


# 3D Aspects Of React Unidirectional Data Binding



One Way Data Binding

# 3D Aspects Of React Server-side Rendering



Activate Windows

# features of React

**Component-Based Architecture:** React allows developers to build UIs as a collection of components, each representing a part of the user interface. Components can be reused and nested within one another, making development more modular and maintainable.

**Virtual DOM:** React uses a Virtual DOM (Document Object Model), which is a lightweight copy of the real DOM. When changes are made to the UI, React updates the Virtual DOM first, then efficiently compares it with the real DOM and only applies the necessary changes, improving performance.

**Declarative Syntax:** React uses a declarative approach to define UI components. Instead of telling the app how to update the UI step by step, you describe how the UI should look based on the current state. React takes care of updating the DOM.

# features of React

**JSX (JavaScript XML):** React uses JSX, a syntax extension that allows you to write HTML-like code within JavaScript. JSX makes it easy to visualize the structure of your UI while using JavaScript logic.

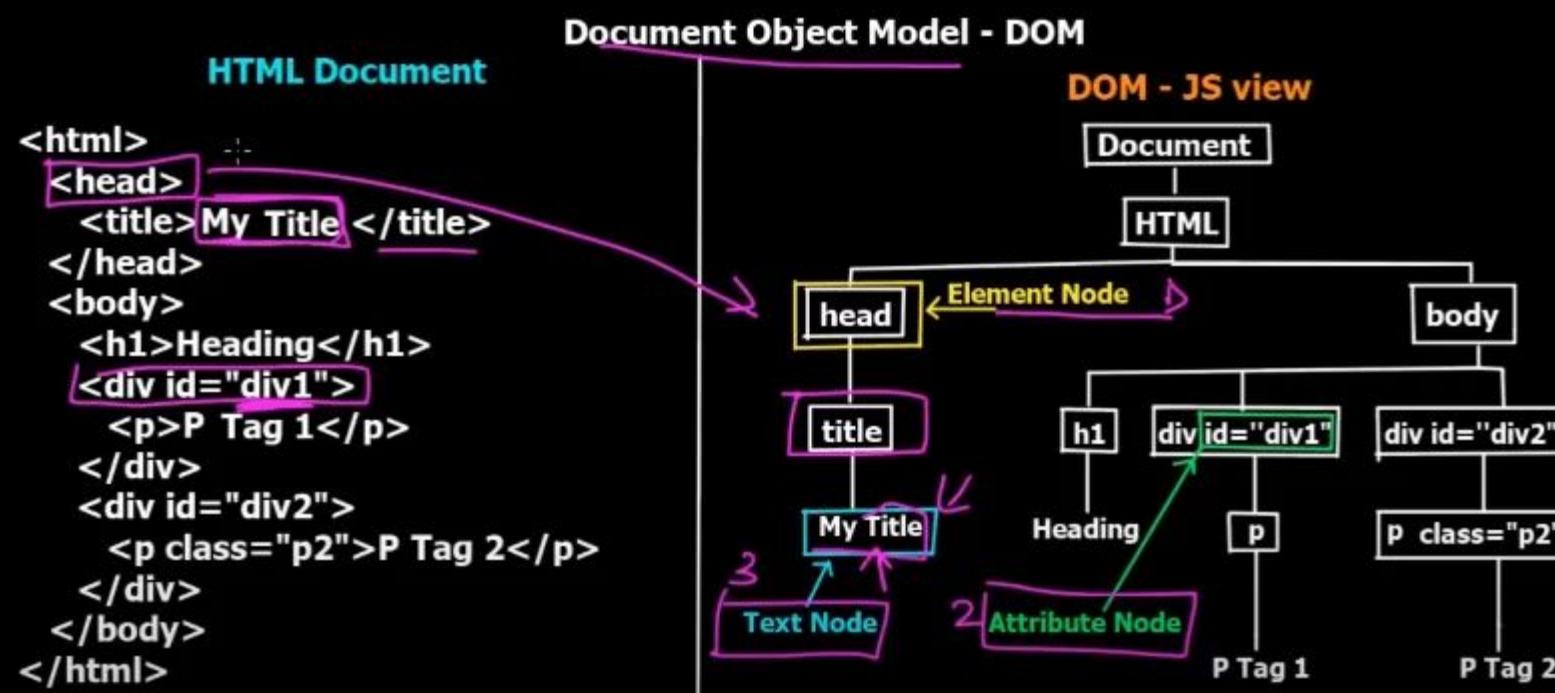
**Unidirectional Data Flow:** React follows a one-way data flow where the data flows from parent components to child components. This makes it easier to understand and debug how data changes within an application.

**State and Props:** Components can have **state** (internal data that can change over time) and **props** (external data passed to components). React automatically updates the UI when the state or props change.

# DOM

In React, the **DOM** (Document Object Model) is a crucial concept because React interacts with the DOM to update the UI.

The DOM is a programming interface for web documents. It represents the page so that programs can manipulate the structure, style, and content. In a web browser, the DOM is a tree-like structure where each node represents part of the web page (e.g., elements, attributes, text).



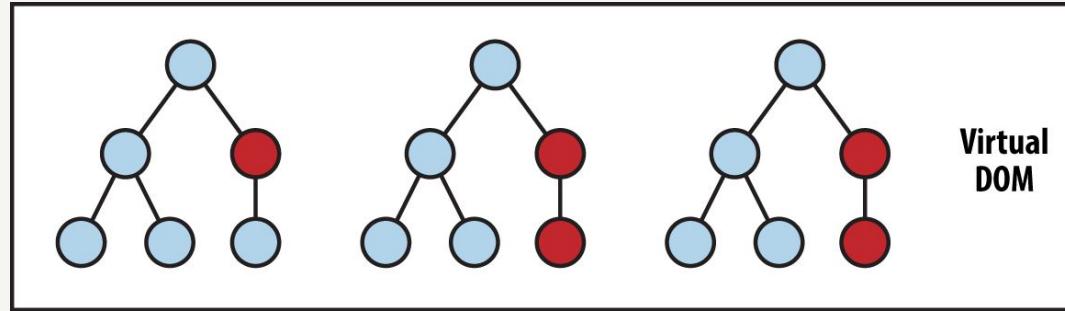
# DOM in React

In traditional web development, when you update a web page (like changing an element's text or style), the browser updates the actual DOM. However, direct manipulation of the DOM can be slow, especially for complex applications with many updates, which leads to performance issues.

To solve this, React introduced the concept of the **Virtual DOM**

# Virtual DOM

- React keeps a virtual representation of the real DOM in memory. This is a lightweight copy of the actual DOM.
- When the state of a component changes, React first updates the virtual DOM instead of the real DOM directly.
- Then, React compares the virtual DOM with a snapshot of the previous state (a process called **reconciliation**).
- It then calculates the minimal set of changes required to update the real DOM, which makes React's rendering much faster.



State Change

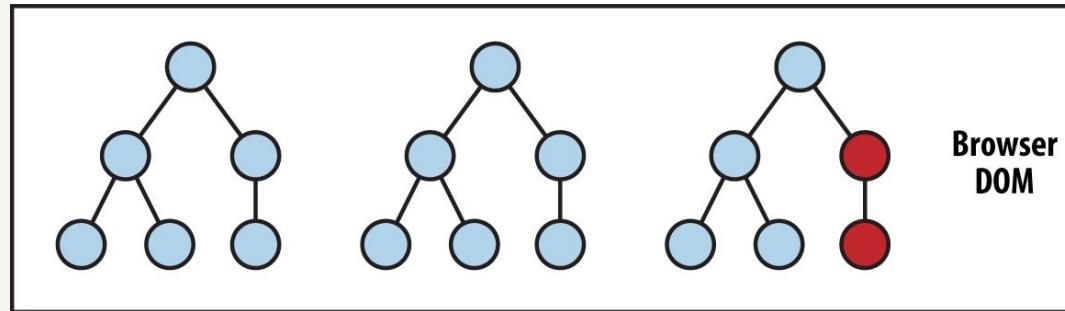


Compute Diff



Re-render

Virtual  
DOM



Browser  
DOM

## **SPA :**

A single-page application is an app that doesn't need to reload the page during its use and works within a browser

When we navigate between multiple page in react it not change page but it change only dom

**Babel :** Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript.

In vs code install extension babel and code runner

# Installation

# Install Node :



Node.js® is an open-source, cross-platform JavaScript runtime environment.

**CYBER MONDAY** **SAVE UP TO 65% OFF!**

Elevate your profile. Improve your skills. Showcase your badges.

**SAVE NOW** Offer ends Dec. 4

OpenJS Foundation

Download Node.js®

**20.10.0 LTS**  
Recommended For Most Users

**21.2.0 Current**  
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)    [Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

**Create React App (CRA)** has long been the go-to tool for most developers to scaffold React projects and set up a dev server. It offers a modern build setup with no configuration.

But it increased development and build time when the project size increases

**Vite is a next-generation**, front-end tool that focuses on speed and performance.

It consists of two major parts:

- 1) A development server
- 2) A Build Command

## Check version of node

### 1) (Traditional installation)

```
#node -v  
#npm install create-react-app  
#npx create-react-app my-app  
#cd my-app  
#npm start
```

**npm :** node package manager npm is a tool which is used to install packages.

**npx : npx is a tool to execute packages.**

**SWC : SWC** (Speedy Web Compiler) is an extensible Rust-based platform for the next generation of fast developer tools.

SWC is used to transform your JavaScript/TypeScript code into a format that the browser can understand. It replaces Babel and TypeScript in the build process

```
○ PS C:\Users\tops\Documents\GitHub\MERN_STACK\REACT> npm create vite@latest
Need to install the following packages:
  create-vite@5.0.0
Ok to proceed? (y) y
✓ Project name: ... myproject
? Select a framework: » - Use arrow-keys. Return to submit.
> Vanilla
  Vue
  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
```

## 2) create new project using vite :

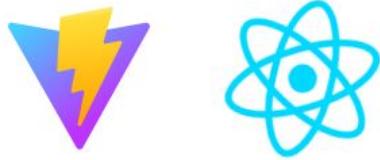
```
#npm init vite@latest my-react-app --template react
```

Ok to proceed ? (y) : y

```
#cd maincontentapp
```

```
#npm install (Install the dependencies required for the project)
```

```
#npm run dev
```



# Vite + React

count is 0

Edit `src/App.jsx` and save to test HMR

Click on the Vite and React logos to learn more

# React Strict Mode

development tool in React that helps developers write better and more reliable code by highlighting potential problems in the app.

It doesn't affect the production build or performance of the app, but it provides additional checks and warnings during development.

it check warning throughout application during development.  
***it prevent deprecated Apis***

# Node Module

node\_modules folder is a directory that contains all the dependencies (packages) required for the application to function.

These dependencies are installed through npm (Node Package Manager) or yarn, and the node\_modules folder is created when you run npm install or yarn install after setting up a new React project or adding a package to the project

The node\_modules folder stores all the third-party libraries, frameworks, and tools that your React application depends on. For example, React itself, react-dom, webpack, babel, and any other libraries you install via npm or yarn will be in this folder.

# Project Structure Overview

```
my-react-app/
|__ public/                      # Static files (favicon, index.html, etc.)
|__ src/                         # Main source code
|   |__ assets/                  # Images, fonts, styles, etc.
|   |__ components/             # Reusable UI components
|   |__ pages/                   # Page-Level components (route-specific)
|   |__ layouts/                 # Layout components (wrappers for pages)
|   |__ hooks/                   # Custom hooks
|   |__ context/                 # React Context API providers
|   |__ utils/                   # Utility functions/helpers
|   |__ services/                # API calls and external services
|   |__ store/                   # State management (Redux, Zustand, etc.)
|   |__ routes/                  # App routing configuration
|   |__ App.jsx                  # Root component
|   |__ main.jsx                 # Entry point (React 18+ uses createRoot)
|__ .env                          # Environment variables
```

# JSX

JSX (JavaScript XML) is a syntax extension for JavaScript used in React to describe UI components.

It allows you to write HTML-like code within JavaScript, making it easier to visualize the UI structure.

# Why Use JSX ?

- It makes UI code **more readable and expressive**.
- It allows **embedding JavaScript expressions** within curly braces `{}`.
- JSX **compiles down to JavaScript** using Babel before rendering in the browser.

[\*\*Click here for practicals\*\*](#)

# JSX Rule ?

- Must have a single parent element
- Use className instead of class
- Self-closing tags for elements without children
- JavaScript expressions inside {}
- Use camelCase for HTML attributes

# JSX Compilation

JSX is not valid JavaScript, so Babel compiles it into regular JavaScript:

jsx

```
const element = <h1>Hello, World!</h1>;
```

Compiles to Js

```
const element = React.createElement("h1", null, "Hello,  
World!");
```

# Components

A **component** is a **reusable** and **independent** piece of UI in React.

It is like a JavaScript function that returns JSX (UI elements).  
Components help build modular and maintainable UIs.

# Types of Components

There are two main types of components in React:

1. **Functional Components** (Modern & Recommended)
2. **Class Components** (Older Approach)

# Function Component

## 1. Functional Components (Recommended)

A **functional component** is a JavaScript function that returns JSX. It is simple and does not require a **class**.

[Click here for practical](#)

# Advantages of Functional Components:

- **Easier to read & write**
- **Better performance** (no extra overhead like class components)
- Supports **React Hooks** (for state and lifecycle methods)

# Class Component

## **2. Class Components (Older Approach)**

Before React Hooks, components were created using ES6 classes. These components extend `React.Component` and have a `render()` method.

[Click here for practical](#)

## Disadvantages of Class Components:

- More **verbose** (requires `this` keyword)
- Harder to manage **state** and **lifecycle methods**
- Not recommended for new projects

- **React Class Component Lifecycle**

In React **class components**, lifecycle methods allow you to control **what happens at different stages** of a component's life (from creation to removal).

These lifecycle methods are categorized into **three phases**:



Phase	Method	Purpose
Mounting	<code>constructor()</code>	Initialize state, bind functions
	<code>getDerivedStateFromProps()</code>	Sync state with props (rarely used)
	<code>render()</code>	Render UI
	<code>componentDidMount()</code>	Perform API calls, subscriptions
Updating	<code>getDerivedStateFromProps()</code>	Sync state with new props
	<code>shouldComponentUpdate()</code>	Optimize rendering (return <code>false</code> to prevent re-render)
	<code>render()</code>	Re-render UI
	<code>getSnapshotBeforeUpdate()</code>	Capture values before update (e.g., scroll position)
Unmounting	<code>componentDidUpdate()</code>	Perform side effects after update
	<code>componentWillUnmount()</code>	Cleanup (remove event listeners, clear timers)

# Import & Export Components in React

components are usually kept in **separate files** for better organization. To use a component in another file, you need to **export** it and then **import** it.

## Exporting Components

There are **two ways** to export a component:

1. **Default Export**
2. **Named Export**

## Default Export (Common Approach)

- Used when exporting **a single component** from a file.
- You can **import it with any name** in another file.

### Export Syntax :

```
const Greeting = () => {  
    return <h1>Hello, Welcome to React!</h1>;  
};  
export default Greeting; // Default Export
```

### Import Syntax :

```
import Greeting from "./Greeting"; // Importing the default export
```

## Named Export (For Multiple Components)

- Used when **exporting multiple components** from a file.
- You **must import using the exact name** in `{}`.

### Export Syntax :

```
export const Welcome = () => <h1>Welcome to React!</h1>;
```

```
export const Msg = () => <h1>Hello everyone</h1>;
```

### Import syntax :

```
import { Welcome, Msg } from "./Messages";
```

# Props in React

**Props (short for "Properties")** are a way to pass **data** from one component to another in React.

They allow components to be **dynamic and reusable** by receiving values from their parent components.

**Props are immutable (read-only)** – a component cannot modify its own props.

**Props flow from parent to child** – data moves one way (top to bottom).

**Props can be of any data type** – string, number, boolean, array, object, function, etc.

- 1) **Passing Props to a Functional Component**

[Click here for practical](#)

- 2) **Passing Props to a Class Component**

[Click here for practical](#)

- 3) **Destructuring props**

[Click here for practical](#)

# Hook Introduction

Hooks were introduced in **React 16.8** to allow functional components to use **state** and other React features **without writing class components**.

# Why Hooks?

Before hooks, React components were either:

Functional components (stateless, no lifecycle methods).

Class components (stateful, with lifecycle methods).

Hooks enable stateful logic inside functional components, making code more readable and reusable.

# State in React

**state** is an object that holds data or information about the component. It determines how the component behaves and renders. When the state changes, the component re-renders to reflect the new data.

# State in React

state is a built-in feature that allows components to keep track of changing data. It represents the internal state of a component, and when the state changes, React re-renders the component to reflect the updated state.

State is mutable and can be modified using the `setState()` method provided by React.

# useState Hook

## What is useState?

The useState hook is a built-in React hook that allows functional components to manage state. It lets components store, update, and re-render based on dynamic data without using class components.

# Syntax of useState

```
const [state, setState] = useState(initialValue);
```

- `state` → The current state value.
- `setState` → A function to update the state.
- `initialValue` → The default value for the state.

[Click here for practical](#)

# State in Function Component

In modern React applications, functional components manage state using the useState hook.

**[Click here for practical](#)**

# Update state

- 1) Direct Update (Simple State)

```
setCount(count + 1);
```

- 2) Functional Update (When New State Depends on Previous State)

State updates are asynchronous, so for updates based on the previous state, use a function:

```
setCount(prevCount => prevCount + 1);
```

# Style

In React, we can style components using multiple approaches, depending on our project needs

- 1) Inline Styles in React
- 2) CSS Stylesheets (.css files)
- 3) CSS Modules (Scoped CSS)
- 4) Styled Components (CSS-in-JS)
- 5) Tailwind CSS (Utility-First CSS Framework)
- 6) Dynamic Styling in React

# Style Uses :

Styling Method	Use Case
Inline Styles	Quick, dynamic styles, but not recommended for complex applications.
CSS Files	Traditional approach, good for global styles.
CSS Modules	Best for component-scoped styles in large projects.
Styled Components	Good for scoped styles, dynamic theming, and modern apps.
Tailwind CSS	Fast development with utility classes, great for responsive design.

## 1) **Inline Styles in React**

React allows us to use inline styles by passing a JavaScript object to the style attribute.

**[Click here for practical](#)**

- Uses a JavaScript object for styles.
- Properties use camelCase (backgroundColor instead of background-color).
- Useful for dynamic styling but not recommended for large projects due to performance concerns

## 2) CSS Stylesheets (.css files)

can write styles in an external CSS file and import it into your component.

**[Click here for practical](#)**

- Keeps styles separate from logic (better maintainability).
- Familiar approach for developers used to traditional CSS.
- Suitable for larger projects with reusable styles.

### 3) CSS Modules (Scoped CSS)

CSS Modules scope styles to a specific component, preventing conflicts.

[\*\*Click here for practical\*\*](#)

- Prevents global CSS conflicts by scoping styles to components.
- Class names are unique (e.g., .button\_abc123).
- Recommended for large applications where styles need isolation.

## 4) Styled Components (CSS-in-JS)

**Latest react version (2024 is "react": "^19.0.0",)**

Styled Components is a popular library that allows you to write CSS directly inside JavaScript.

```
#npm install styled-components
```

**Click here for practical**

- Styles are written inside JavaScript files.
- Uses template literals () for styles.
- Supports dynamic styling with props:

*Note : styled-components is not yet fully compatible with React 19, causing the useContext error*

## Downgrade to the Stable React 18 Version

Ensure your **package.json** looks like this:

```
"dependencies": {  
  "react": "^18.2.0",  
  "react-dom": "^18.2.0",  
  "styled-components": "^6.1.15"  
}
```

Delete **node\_modules** and reinstall dependencies.

```
npm cache clean --force  
npm install
```

**Restart your development server.**

## 5) Tailwind CSS (Utility-First CSS Framework)

Tailwind CSS is a utility-first framework that provides pre-built classes for rapid styling

Click here for practical

```
#npm install -D tailwindcss postcss autoprefixer
```

```
#npx tailwindcss init -p
```

## 6) Dynamic Styling in React

[Click here for practical](#)

- Uses state (useState) to change styles dynamically.
- Inline styles can be modified based on conditions.

# **Module – 10**

## **React Lists , Hooks , Localstorage , Api Project**

## Arrays

An array in JavaScript is an ordered list of values. In React, arrays are often used to store lists of items or to manage state when you need to keep track of multiple similar pieces of data (like a list of users or a collection of objects). Arrays in React are also commonly used for rendering dynamic lists of components using .map()

[\*\*Click here for practical\*\*](#)

## **State with Arrays:**

When using arrays in React for state management, the state will typically be set up using useState. We can add, remove, or update items in an array.

**Immutable Updates:** When you want to update an array in React (like adding or removing elements), you should never directly mutate the array (e.g., using push(), splice()). Instead, create a new array to maintain immutability. React relies on immutability to detect changes and re-render components effectively.

[\*\*Click here for practical\*\*](#)

# Objects

An object in JavaScript is a collection of key-value pairs. In React, objects are commonly used to represent more complex pieces of data, such as the state or props of a component, or for managing individual pieces of data within a component

[\*\*Click here for practical\*\*](#)

## **State with Objects**

When managing an object in the state, React useState hook allows you to store the object. If you need to update a specific property of the object, you should ensure that you maintain immutability.

**[Click here for practical](#)**

Arrays are best for managing ordered collections of similar items and are useful for rendering lists.

Objects are useful for managing more complex data, especially when multiple properties belong together (like a user's information).

React requires immutability, so when modifying arrays or objects in state, you should always create a new copy (e.g., using the spread operator or methods like `.concat()` or `.map()`).

You can combine both arrays and objects to manage more complex data structures, like an array of objects, for more dynamic and complex rendering

**[Click here for practical](#)**

## Keys

React uses the key prop to optimize the re-rendering process by efficiently tracking elements in a list. When a list is updated (such as adding or removing items), React needs to figure out how to apply the changes without unnecessarily re-rendering every single item in the list. This is where keys come in.

Without keys, React will rely on the index of the items in the array (in some cases) to track changes. However, this can cause issues, especially if the list is reordered or if items are added or removed, because indexes might no longer uniquely identify the items.

[\*\*Click here for practical\*\*](#)

## Why Not Use the Index as a Key

Using the index as the key can lead to issues in certain situations. For instance:

- **Reordering:** If the list is reordered (e.g., moving items around), React might mistakenly think the items haven't changed, leading to incorrect or unexpected rendering.
- **Adding or Removing Items:** If items are added or removed, React may misidentify which items have changed.

- Always use a **unique and stable** key (preferably an **id**).
- Avoid using **array index** as a key unless the list is static.
- Keys **help React efficiently update the UI** and prevent unnecessary re-renders.

## React Hooks

React **Hooks** are special functions that let you **use state and other React features** in functional components.

They were introduced in **React 16.8** to replace class components for managing state and side effects.

## 1) useState – Manage State

The useState hook allows you to add state to a functional component.

[\*\*Click here for practical\*\*](#)

## 2) `useEffect` – Handle Side Effects

The `useEffect` hook is used for side effects (e.g., fetching data, updating the DOM, setting timers).

## What is a Side Effect?

Side effects are operations that affect something outside the scope of the function being executed. Some examples of side effects in React are:

- Fetching data from an API
- Subscribing to or cleaning up event listeners
- Interacting with the DOM (e.g., updating the title of the page)
- Setting up timers (e.g., setInterval or setTimeout)

## Syntax of `useEffect`

```
import { useEffect } from "react";  
  
useEffect(() => {  
  // Side effect logic here  
});
```

# Different Use Cases of useEffect

## 1. useEffect Without Dependencies (Runs on Every Render)

If you do not provide a dependency array, `useEffect` runs **after every render**.

[\*\*Click here for practical\*\*](#)

**Use case:** Debugging or logging values on every render.

## Different Use Cases of useEffect

### 2. useEffect With an Empty Dependency Array [] (Runs Only Once - Like componentDidMount)

When you pass an empty dependency array ([]), useEffect will only run once, after the initial render.

[\*\*Click here for practical\*\*](#)

## Different Use Cases of useEffect

### 3. useEffect With Dependencies (Runs Only When Dependencies Change)

When dependencies are provided, useEffect runs only when those values change.

[\*\*Click here for practical\*\*](#)

`useEffect(() => { ... })` → Runs on every render.

`useEffect(() => { ... }, [])` → Runs once when mounted (like `componentDidMount`).

`useEffect(() => { ... }, [dependency])` → Runs when the dependency changes.

`useEffect(() => { return () => { ... } }, [])` → Cleanup function runs when unmounting.

# API

In React, an **API (Application Programming Interface)** allows your application to communicate with a server to **fetch** or **send** data.

The most common use case is fetching data from a REST API and displaying it in your React components.

# 1. Fetching API Data in React

React provides multiple ways to fetch data from an API:

- `fetch()` (built-in JavaScript function)
- `axios` (a popular external library)
- React Query (for advanced use cases)

## Fetching Data Using `fetch()`

React component that fetches data from an API using `fetch()`.

[Click here for practical](#)

## Fetch API using `async/await`

This approach makes the code cleaner and easier to read.

[Click here for practical](#)

# Axios Vs Fetch

**Axios:** More concise and user-friendly.

**Axios** automatically transforms the response data into JSON.

**Axios** automatically rejects the request if the response status is not in the `2xx` range.

**Axios** automatically sets headers like `Content-Type: application/json` for `POST` requests.

---

**Fetch** only rejects requests for network errors; HTTP errors (like 404, 500) must be handled manually.

**Fetch** requires manually calling `.json()` on the response.

**Fetch:** Requires more manual handling of responses and errors.

**Fetch** requires setting headers manually.

## Axios (Easier Alternative to Fetch)

Axios is a popular library for making HTTP requests. First, install it

**npm install axios**

[Click here for practical](#)

## Making Post Request :

To send data to an API, use the POST method.

[Click here for practical](#)

# React Query (Advanced API Handling)

React Query simplifies API calls, caching, and state management.

**npm install @tanstack/react-query**

React Query is a powerful data-fetching and state-management library for React applications. It simplifies handling server-state (API requests) with features like caching, background updates, pagination, automatic retries, and refetching.

## Why Use React Query?

- Eliminates manual state management for API calls.
- Caches API responses for better performance.
- Handles background updates automatically.
- Supports pagination, infinite scrolling, and real-time updates

### 3) **useContext – Use Context API**

Avoid prop drilling by using useContext to share data globally.

The useContext hook in React is used to share state and values globally across components without the need to pass props manually (prop drilling).

It makes state management simpler in small to medium-sized applications.

## Why Use `useContext`?

Normally, to pass data from a parent to a deeply nested child component, you'd need to pass props at every level (a process known as prop drilling):

## Step 1 : Create a Context

create a Context using React.createContext()

### Syntax :

```
const UserContext = createContext(null); // Default value can be null or any object
```

[Click here for practical](#)

## Step 2 : Provide Context at a Higher Level

Wrap the entire component tree (or a part of it) with the Provider, passing the value.

Syntax :

```
const user = { name: "Anjali Patel", subject: "React js" }; // record  
  
return (  
  <UserContext.Provider value={user}>  
    <ChildComponent />  
  </UserContext.Provider>  
);
```

[Click here for practical](#)

## Step 3: Consume Context using useContext

Now, instead of prop drilling, use useContext inside a component to access the shared state.

Syntax :

```
import UserContext from "./UserContext";
const ChildComponent = () => {
  const user = useContext(UserContext); // Get value from Context

  return <h2>Welcome, {user.name}! Age: {user.subject}</h2>;
};
```

[Click here for practical](#)

Final Practical : [Click here](#)

## Theme Change using Global State Management

- 1) Create Context : [Click here](#)
- 2) Provide context : [Click here](#)
- 3) useContext : [Click here](#)

## 4) **useRef**

**useRef** is a built-in React Hook that provides a way to persist a mutable value across renders without causing re-renders when the value changes.

It is commonly used for accessing DOM elements directly and maintaining values across renders without triggering component re-renders.

### Syntax

```
import { useRef } from "react";  
  
const ref = useRef(initialValue);
```

## Common Use Cases of `useRef`

### 1. Accessing DOM Elements

`useRef` is frequently used to get a reference to a DOM element and perform operations on it.

[Click here for practical](#)

### 2. Storing Mutable Values Without Causing Re-Renders

Unlike `useState`, changing `useRef().current` does not cause a component to re-render.

[Click here for practical](#)

## 5) useMemo

`useMemo` is a built-in React Hook that helps optimize performance by **memoizing** the result of a function so that it does not need to be recalculated on every render.

It is particularly useful for expensive computations and preventing unnecessary re-renders of components.

### Syntax

```
import { useMemo } from "react";
```

```
const memoizedValue = useMemo(() => computeExpensiveValue(dep1,  
dep2), [dep1, dep2]);
```

## Why Use `useMemo`?

- Avoid expensive recalculations
- Prevent unnecessary re-renders of child components
- Optimize performance in large applications

## Common Use Cases of `useMemo`

### 1. Optimizing Expensive Computations

If a function performs a **costly calculation**, `useMemo` ensures it runs **only when necessary**.

[Click here for practical](#)

### 2. Memoizing Filtered or Sorted Lists

If a component displays a large list, filtering or sorting it on every render can be inefficient.

[Click here for practical](#)

## 5) useCallback

### What is useCallback?

useCallback is a React Hook that **returns a memoized version of a callback function**, preventing unnecessary re-creation of the function on every render. It is useful when passing functions as props to child components or using them in dependencies of other Hooks like useEffect

## Syntax

```
const memoizedCallback = useCallback(  
() => {  
    // Function logic here  
},  
[dependencies]  
);
```

The first argument is the function to be memoized.

The second argument is the dependency array [dependencies].

If any dependency changes, `useCallback` will return a new function reference.  
Otherwise, it returns the **same function reference** from the previous render.

## Why is `useCallback` Needed?

Every time a component re-renders, all functions inside it are recreated. This can cause unnecessary renders in child components that receive these functions as props

[Click here for practical](#)

[Click here for child Component](#)

### Issue :

The handleClick function is re-created every time the parent re-renders.

Even though ChildComponent is wrapped in React.memo, it still re-renders because handleClick is a new function on every render.

## Solution: Using `useCallback`

We can **memoize** the function using `useCallback`, so it doesn't get re-created unnecessarily.

[Click here for practical](#)

Now, `handleClick` only gets created once and does not change unless its dependencies change.

The child component (`ChildComponent`) does NOT re-render when `ParentComponent` updates unless necessary.

## useCallback with useEffect

When using a function inside `useEffect`, if it's re-created on every render, it can **cause unnecessary re-runs** of the effect.

Problem: Infinite Loop in `useEffect`

[Click here for practical](#)

### issue

The function `fetchData` gets re-created on every render.

Since `useEffect` depends on `fetchData`, it re-runs on every render, causing an infinite loop

Solution: Using `useCallback` to Stabilize the Function

[Click Here for practical](#)

`fetchData` is **memoized** using `useCallback`, so it does not get re-created on every render.

`useEffect` **only runs once** (unless dependencies change), preventing an **infinite loop**.

## Custom Hooks in React.js

A custom hook in React is a reusable function that encapsulates logic using React hooks (useState, useEffect, useRef, etc.). It helps reduce code duplication and improves code organization.

## Why Use Custom Hooks?

Reusability – Write logic once, use it in multiple components.

Readability – Makes components cleaner by separating concerns.

Maintainability – Easier to debug and update logic in one place.

Performance – Prevents unnecessary re-renders by isolating logic.

[Click here for custom hook](#)

[Click here for implementation of Custom Hook](#)

## React Hook Form

React Hook Form (RHF) is a lightweight and efficient form management library for React. It **improves performance** by reducing unnecessary re-renders and keeping form state minimal.

**Lightweight** – Small bundle size (~9KB gzipped)

**Faster Performance** – Reduces unnecessary re-renders

**Easy Integration** – Works well with UI libraries (Material-UI, Ant Design, etc.)

**Built-in Validation** – Uses native HTML validation and custom validation

#npm install react-hook-form

[Click here for practical](#)

# CRUD Operations Using Array and Object

[Click here for practical](#)

# What is LocalStorage?

LocalStorage is a web API provided by the browser that allows developers to store key-value pairs in a user's browser persistently (i.e., the data remains even after the page is reloaded or the browser is closed). It is useful for caching user preferences, authentication tokens, or temporary app data.

## Key Characteristics:

- Data persists even after a page refresh or browser restart.
- Stores data in key-value format (both keys and values are strings).
- 5MB limit per domain (varies by browser).
- Synchronous API (can block execution if used in large amounts).

## **LocalStorage provides three main methods:**

`localStorage.setItem(key, value)` → Stores data

`localStorage.getItem(key)` → Retrieves data

`localStorage.removeItem(key)` → Removes a specific item

`localStorage.clear()` → Clears all data

[\*\*Click here for practical\*\*](#)

# **Module-11**

## **React -Advance React- Styling , Routing**

## **React bootstrap ;**

React Bootstrap is a **React-based** implementation of the popular **Bootstrap** framework. It replaces traditional Bootstrap's **jQuery dependencies** with React components, making it more efficient and compatible with React applications.

### **Why Use React Bootstrap?**

**Fully Compatible with React** – Uses React components instead of jQuery.

**Pre-built UI Components** – Buttons, forms, modals, and more.

**Responsive & Mobile-friendly** – Uses Bootstrap's grid system.

**Customizable** – Can be overridden with CSS or Sass.

**Lightweight** – Only imports required components

## Installation

```
#npm install react-bootstrap bootstrap
```

## Import Bootstrap CSS

```
import "bootstrap/dist/css/bootstrap.min.css";
```

[Click here for practical](#)

## What is Tailwind CSS?

Tailwind CSS is a utility-first CSS framework that allows you to build modern and responsive UI without writing custom CSS. Instead of predefined components (like Bootstrap), Tailwind provides low-level utility classes that let you design directly in your HTML/JSX.

### Why Use Tailwind CSS in React?

Utility-First – No need to write custom CSS files.

Highly Customizable – Tailwind can be extended via tailwind.config.js.

Faster Development – Directly use classes inside JSX.

Responsive Design Built-in – Supports breakpoints (sm, md, lg, etc.).

Optimized with PurgeCSS – Removes unused CSS in production for a smaller bundle size.

## Installation

```
#npm install -D tailwindcss postcss autoprefixer
```

## Initialize Tailwind Run

```
#npx tailwindcss init -p
```

This will generate:

- `tailwind.config.js` (Tailwind configuration)
- `postcss.config.js` (PostCSS configuration)

## Configure Tailwind to Remove Unused CSS

Edit `tailwind.config.js` to include your React file paths:

```
module.exports = {  
  
  content: [  
  
    "./src/**/*.{js,jsx,ts,tsx}", // Include all React files  
  
  ],  
  
  theme: {  
  
    extend: {}, // Extend if needed  
  
  },  
  
  plugins: [],  
  
};
```

## Import Tailwind in Your CSS

In `src/index.css` or `src/globals.css`, add css

`@tailwind base;`

`@tailwind components;`

`@tailwind utilities;`

[Click here for practical](#)

## What is React Router?

React Router is a popular **routing library for React** that allows you to create **single-page applications (SPA)** with multiple views **without refreshing the page**.

### Installing React Router

To use React Router, install it via npm :

```
#npm install react-router-dom
```

## React Router Components Overview

**BrowserRouter** – Wraps your app for routing.

**Routes** – Contains all route definitions.

**Route** – Defines a specific path and component.

**Link** – Used for navigation (similar to `<a>` but without page reload).

**NavLink** – Works like **Link** but adds an active class when the route matches.

**useNavigate** – A hook for programmatic navigation.

[Click here for practical](#)

# Lazy loading (performance optimization)

## What is Lazy Loading?

Lazy loading is a performance optimization technique that **delays the loading of non-essential resources** until they are needed. This reduces the **initial load time** and **improves page speed**, especially in large applications.

## Why Use Lazy Loading?

**Faster Initial Load** – Loads only essential components first.

**Better Performance** – Reduces unnecessary resource usage.

**Optimized Bandwidth** – Loads content only when required.

**Improves Core Web Vitals** – Enhances **Largest Contentful Paint (LCP)** and **First Input Delay (FID)** scores.

[Click here for practical](#)

# Module 12

## React – JSON-server and Firebase Real Time Database

## What is JSON Server?

JSON Server is a **lightweight, mock REST API** that allows you to create a **fake backend** using a simple `db.json` file. It helps **simulate API calls** in development without needing a real backend.

## Why Use JSON Server?

**Quickly create a REST API** without writing backend code.

**Supports CRUD operations** (`GET, POST, PUT, DELETE`).

**Works like a real API** with `fetch()` or Axios.

**Ideal for prototyping and testing React apps.**

## 1. Installation

```
# Global installation
```

```
npm install -g json-server
```

```
# OR install locally in a project
```

```
npm install json-server --save-dev
```

## 2. Create a Fake Database (**db.json**)

Create a file named **db.json** in your project's root folder and add sample data:

**syntax:**

```
{  
  "users": [  
    { "id": 1, "name": "Alice", "email": "alice@example.com" },  
    { "id": 2, "name": "Bob", "email": "bob@example.com" },  
    { "id": 3, "name": "Charlie", "email": "charlie@example.com" }  
  ]  
}
```

### 3. Start the JSON Server

```
json-server --watch db.json --port 5000
```

--watch db.json → Watches db.json for changes.

--port 5000 → Runs the server on port 5000 (default is 3000).

<http://localhost:5000/users>

[Click Here for practical](#)

## Firebase

Firebase is a **backend-as-a-service (BaaS)** provided by Google that helps developers build and scale apps **without managing servers**. It offers various features like **authentication, real-time databases, cloud storage, hosting, and more**.

# What is Firebase?

## **Firebase Features in React.js**

**Authentication** – Login with Email/Password, Google, Facebook, etc.

**Firestore Database** – NoSQL database for real-time and scalable apps.

**Realtime Database** – Live data sync for chat apps and collaborative tools.

**Storage** – Store and serve images, videos, and files.

**Hosting** – Deploy React apps with Firebase Hosting.

**Website:** <https://firebase.google.com>

# Installation :

```
#npm install firebase
```

**Configure Firebase in React : Create a new file `firebaseConfig.js` and paste your Firebase SDK**

**syntax :**

```
import { initializeApp } from "firebase/app";

const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_PROJECT_ID.firebaseio.com",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_PROJECT_ID.appspot.com",
  messagingSenderId: "YOUR_SENDER_ID",
  appId: "YOUR_APP_ID"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export default app;
```

[Click here for practical](#)

# Module-13

## React - Applying Redux

## Redux in React

Redux is a **state management library** used in React applications to manage global state efficiently. It helps keep track of application state **across components** and makes it easier to debug and scale apps

Redux makes state management **organized and scalable**.

It's useful for **large applications** where state sharing is essential.

# What is Redux?

Redux follows the "**Single Source of Truth**" principle, where the entire state of the app is stored in a **single object (store)**.

## Why Use Redux?

**Centralized State** – Manage global state in one place.

**Predictable** – State updates are controlled via actions and reducers.

**Easier Debugging** – Use Redux DevTools to track state changes.

**Efficient** – Optimized state updates prevent unnecessary re-renders.

**Official Redux Website:** <https://redux.js.org/>

# Setting Up Redux in a React App

## Step 1: Install Redux and React-Redux

Run the following command in your React project:

```
#npm install redux react-redux @reduxjs/toolkit
```

**redux** → Core Redux library

**react-redux** → Connects Redux with React

**@reduxjs/toolkit** → Simplifies Redux setup

# Redux Async Actions (Thunk Middleware)

Redux Thunk allows you to make **API calls inside Redux actions**.

## Step 1: Install Redux Thunk

```
npm install redux-thunk
```

## Key Concepts of Redux

Store – The global state of the application.

Actions – Objects that describe what should change in the state.

Reducers – Functions that update state based on actions.

Dispatch – Sends an action to the store to trigger state updates.

Selectors – Get specific pieces of state from the store.

[Click Here for practical](#)

# Module 14

## Fetch Data using GraphQL

## GraphQL

GraphQL is a query language for APIs and a runtime for executing queries against your data.

It allows clients to request only the data they need, improving efficiency and flexibility compared to REST APIs.

## Key Features of GraphQL

**Single Endpoint** – Fetch all required data with one API call.

**Flexible Queries** – Clients specify exactly what data they need.

**Strongly Typed Schema** – Data is structured with a defined schema.

**No Over-fetching or Under-fetching** – Get only the necessary data.

**Real-time Updates** – Uses **subscriptions** for real-time data.

**Official Website:** <https://graphql.org/>

# GraphQL Architecture

GraphQL consists of three main components:

**Schema – Defines data structure and types.**

**Queries – Retrieve data.**

**Mutations – Modify data (Create, Update, Delete).**

# GraphQL Data Types

GraphQL provides **built-in scalar types** and allows the creation of **custom object types**.

## Scalar Types (Basic Data Types)

These are primitive types that hold single values (like numbers, strings, and booleans).  
The most commonly used scalar types are:

- **Int** → Represents a 32-bit integer (e.g., `5`, `-100`, `42`).
- **Float** → Represents a decimal or floating-point number (e.g., `3.14`, `99.99`).
- **String** → Represents a text value (e.g., `"Hello GraphQL"`).
- **Boolean** → Represents `true` or `false` values.
- **ID** → A unique identifier, often used for database records (e.g., `"b1a2c3d4"`).

## Object Types (Custom Types)

While scalar types define basic values, **object types** define structured data models with multiple fields.

For example, a `User` type can have fields like `id`, `name`, and `email`, where `id` is an **ID**, `name` is a **String**, and `email` is a **String**.

```
type User {  
  id: ID!  
  name: String!  
  email: String!  
  age: Int  
}
```

Here:

- `id`, `name`, and `email` are **mandatory fields** (because of `!`).
- `age` is **optional**.

# GraphQL Modifiers

Modifiers help define **whether a field is required or if it should be an array**.

## Non-Null Modifier (!)

By default, GraphQL fields can return `null`. Adding `!` ensures a field **must** have a value.

```
type Product {  
    id: ID!  
    name: String!  
    price: Float  
}
```

- `id` and `name` **must** always have values.
- `price` **can be null**.

## List Modifier ([])

GraphQL allows defining **lists (arrays)** of data.

```
type Query {  
  books: [Book]  
}
```

Here, **books** returns an **array of Book objects**.

Modifiers can be combined:

- **[String]** → A **list** of strings (can be empty or null).
- **[String]!** → A **required list** (list itself cannot be **null**, but can be empty).
- **[String!]!** → A **required list** where **every element is non-null**.

# GraphQL Schema

The **schema** defines the structure of data, the available queries, and mutations.

- ◆ **Defining a Schema**

A GraphQL schema consists of:

1. **Object Types** → Defines structured data models.
2. **Query Type** → Defines how clients fetch data.
3. **Mutation Type** → Defines how clients modify data.

[Click here for practical](#)

# Setting Up a GraphQL Server

## Step 1: Install Dependencies

Create a new Node.js project and install GraphQL and Apollo Server:

```
#npm init -y
```

```
#npm install express graphql express-graphql
```

[Click here for practical](#)

## Apollo Client in React

Apollo Client is a **popular GraphQL client** for fetching data from a GraphQL API in React.

### Install Apollo Client in React

Create a React project and install Apollo Client:

```
#npx create-react-app graphql-client
```

```
#cd graphql-client
```

```
#npm install @apollo/client graphql
```

[Click here for practical](#)

# Module 15

## Next JS

## Introduction to Next.js

Next.js is a **React framework** that enables server-side rendering (SSR), static site generation (SSG), and many other features to build high-performance web applications.

It is developed by **Vercel** and is widely used for building **fast, SEO-friendly, and scalable** applications.

## Why Use Next.js?

Next.js extends React with **powerful features** like:

### Server-Side Rendering (SSR)

- Renders pages **on the server** and sends ready HTML to the client.
- Improves SEO and **faster page loads** compared to client-side rendering.

### Static Site Generation (SSG)

- Generates **HTML at build time**.
- Perfect for blogs, documentation, and landing pages.

## API Routes

- Built-in support for **backend APIs** without needing a separate server.
- You can create **RESTful API endpoints** inside your Next.js app.

## Automatic Code Splitting

- Next.js **only loads the JavaScript needed** for the current page.
- Improves performance and **reduces load time**.

## SEO Optimization

- Unlike React (which renders on the client), Next.js supports **pre-rendering** for better SEO.

# Installation & Setup

To start a Next.js project, run:

```
#npx create-next-app my-next-app
```

```
#cd my-next-app
```

```
#npm run dev
```

Then open **http://localhost:3000** in your browser.

# Pages & Routing

Next.js uses a **file-based routing system**.

Create a new page inside the `/pages` folder, and it automatically becomes a route.

Example:

`/pages/about.js`

Now, you can access this page at <http://localhost:3000/about>.

[Click here for practical](#)

# Module 16

## Introduction

**Vue.js, D3.js, Chart.js**

## Introduction to Vue.js

Vue.js (or simply Vue) is a **progressive JavaScript framework** used to build user interfaces and **single-page applications (SPAs)**.

It was created by **Evan You** and is known for being **lightweight, flexible, and easy to learn.**

Vue.js is popular because of its:

**Simple & Easy to Learn** → Uses HTML, CSS, and JavaScript with a clean syntax.

**Reactive Data Binding** → Automatically updates the UI when data changes.

**Component-Based Architecture** → Breaks the UI into reusable components.

**Virtual DOM** → Improves performance by updating only the necessary parts of the UI.

**Two-Way Data Binding** → Easily syncs data between the UI and the model.

**Lightweight & Fast** → Smaller size compared to React or Angular.

## Installing & Setting Up Vue.js

You can include Vue.js **directly** in an HTML file using a CDN:

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
```

Or, install it using **npm**:

```
npm create vue@latest
cd my-vue-app
npm install
npm run dev
```

[Click here for practical](#)

## **React.js vs Vue.js**

React.js and Vue.js are two of the most popular JavaScript frameworks/libraries for building user interfaces.

Both have their strengths and weaknesses, making them suitable for different types of projects.

## React.js

- Developed by **Facebook (Meta)** in 2013.
- A **JavaScript library** (not a full framework) used for building **component-based UIs**.
- Uses a **Virtual DOM** for fast updates.
- Requires **additional libraries** for routing (React Router) and state management (Redux, Zustand, etc.).

## Vue.js

- Developed by **Evan You** in 2014.
- A **progressive framework** for building user interfaces.
- Has built-in features like **state management (Vuex, Pinia)** and **routing (Vue Router)**.
- Uses a **Virtual DOM** like React but provides better reactivity.

<b>React Js</b>	<b>Vue Js</b>
Moderate (JSX syntax may be unfamiliar)	Easier (uses regular HTML, CSS, and JS)
Well-documented but fragmented	Excellent and beginner-friendly
Faster for larger applications	Faster for smaller applications

## React Js Syntax :

Uses **JSX (JavaScript XML)** to write UI components inside JavaScript.

```
function App() {  
  return <h1>Hello, React!</h1>;  
}  
export default App;
```

## Vue.js (Template-based) Syntax :

- Uses HTML-based templates with special directives.

```
<template>  
  <h1>Hello, Vue!</h1>  
</template>  
<script>  
export default {  
  data() {  
    return { message: "Hello, Vue!" };  
  }  
};  
</script>
```

# When to Use React vs Vue?

## Use React.js if:

- You are working on a **large-scale application** (e.g., Facebook, Instagram).
- You need a **strong job market** with many opportunities.
- You prefer **JSX and JavaScript flexibility** over templates.
- You don't mind **using third-party libraries** for extra features.

## Use Vue.js if:

- You want a **faster development process** with built-in tools.
- You prefer an **easy-to-learn** framework with great documentation.
- You are building a **small-to-medium app** with minimal setup.
- You want **better state management and reactivity** out of the box.

# What is D3.js?

D3.js (Data-Driven Documents) is a powerful JavaScript library for creating dynamic, interactive, and data-driven visualizations in web browsers. It uses SVG, HTML, and CSS to render charts, graphs, and other visual elements.

Developed by: Mike Bostock

Initial Release: 2011

Main Purpose: Data visualization and manipulating the DOM based on data.

Used For: Bar charts, line charts, scatter plots, maps, animations, and more!

## Why Use D3.js?

Highly Flexible – No predefined chart types; you build exactly what you need.

Data-Driven – Dynamically binds data to DOM elements.

Works with Large Datasets – Supports real-time updates and animations.

Built on Web Standards – Uses SVG, Canvas, HTML, and CSS.

Supports Interactivity – Zooming, panning, transitions, and animations.

# Installing D3.js

## Using CDN (Quick Setup)

```
<script src="https://d3js.org/d3.v7.min.js"></script>
```

## Using npm (For Larger Projects)

```
npm install d3
```

**Then import it in your JavaScript file:**

```
import * as d3 from "d3";
```

# Types of Charts we Can Build with D3.js

Bar Charts

Line Charts

Pie & Donut Charts

Scatter Plots

Heatmaps

Tree Maps & Hierarchical Charts

Network Graphs & Force-Directed Graphs

# When to Use D3.js?

## Great for:

- **Custom Data Visualizations** that libraries like Chart.js can't handle.
- **Interactive Dashboards** with real-time updates.
- **Big Data Visualizations** for analytics and reporting.

## Not Ideal for:

- Simple charts (Use Chart.js or Google Charts for easier solutions).
- Projects requiring fast development with minimal customization.

[Click here for practical](#)

# Types of Charts we Can Build with D3.js

Bar Charts

Line Charts

Pie & Donut Charts

Scatter Plots

Heatmaps

Tree Maps & Hierarchical Charts

Network Graphs & Force-Directed Graphs

# Introduction to Chart.js

Chart.js is a simple, flexible, and lightweight JavaScript library used for creating beautiful and interactive charts in web applications. It is based on the HTML5 Canvas API and supports multiple chart types with smooth animations.

Developed by: Nick Downie

Initial Release: 2013

File Size: Lightweight (~60KB minified)

Main Purpose: Creating charts, graphs, and data visualizations easily.

## Why Use Chart.js?

Chart.js is widely used because it is:

**Easy to Use** – Simple API for creating charts with minimal code.

**Lightweight** – Small file size (~60KB minified) compared to libraries like D3.js.

**Highly Customizable** – Supports colors, labels, legends, tooltips, and more.

**Responsive & Interactive** – Charts adjust to different screen sizes.

**Supports Multiple Chart Types** – Bar, Line, Pie, Radar, Doughnut, Bubble, etc.

**Works with JavaScript Frameworks** – Compatible with React, Vue, Angular, etc.

## Installing Chart.js Using CDN (Quick Setup)

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

## Using npm (For Larger Projects)

```
npm install chart.js
```

Then import it in your JavaScript file:

```
import Chart from "chart.js/auto";
```

[Click here for practical](#)

# Advantages of Chart

## Visually Appealing & Customizable

- Supports various chart types (bar, line, pie, radar, etc.).
- Customizable colors, tooltips, legends, and animations.

## Lightweight & Fast

- Small file size compared to other charting libraries.
- Uses the HTML5 Canvas element, making it efficient for rendering.

## Easy to Use & Integrate

- Simple API with clear documentation.
- Works well with React, Vue, Angular, and other frameworks.

## **Responsive & Interactive**

- Automatically adjusts to different screen sizes.
- Supports hover effects, tooltips, and animations.

## **Supports Real-time Data Updates**

- Can update and animate charts dynamically.
- Useful for live dashboards and monitoring tools.

## **Open-source & Community Supported**

- Free to use under the MIT license.
- Active community for support and improvements.