



TOPS TECHNOLOGIES
Training | Outsourcing | Placement | Study Abroad

Frontend Development

Frontend Development

Part 1 - Website Designing

Part 2 - Advance Javascript

Part 3 - Reactjs



TOPS TECHNOLOGIES

Training | Outsourcing | Placement | Study Abroad

Website Designing

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 -Advance 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

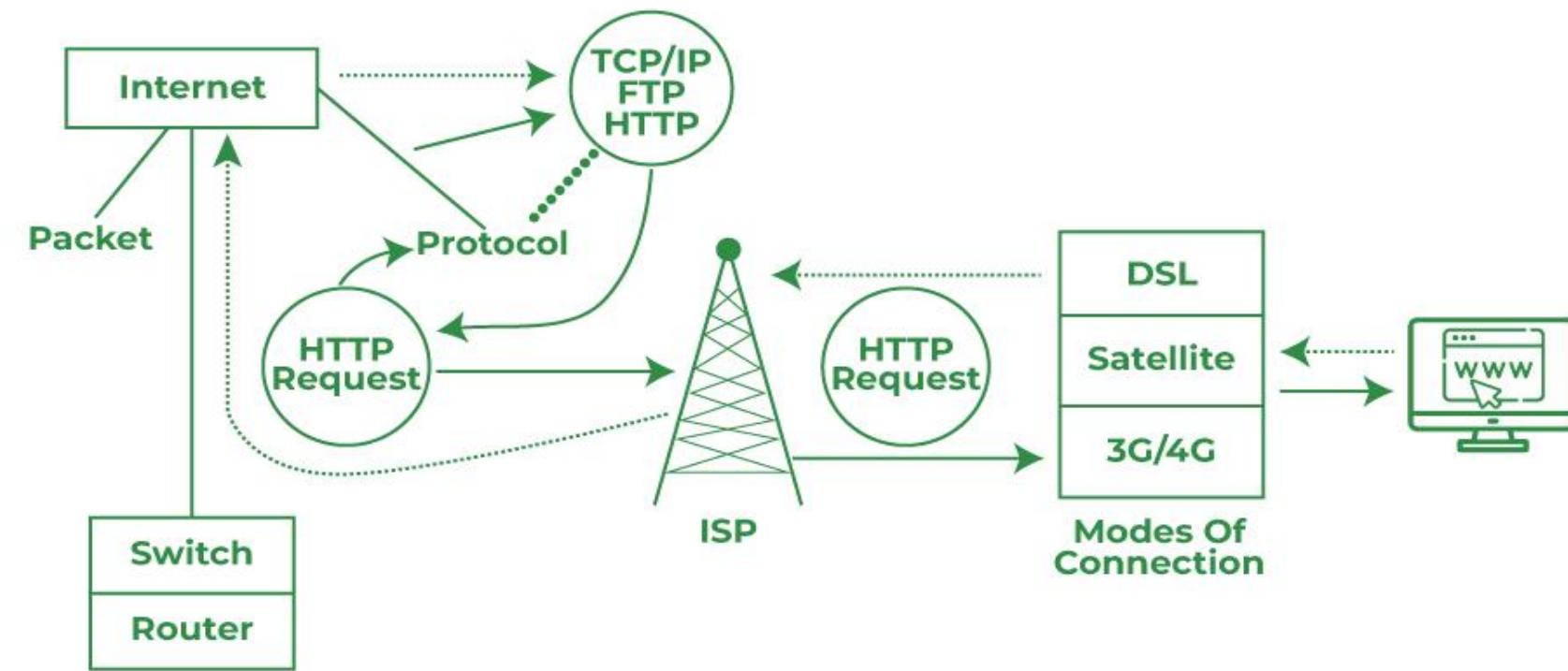
[Foundation]

- ✓ 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?

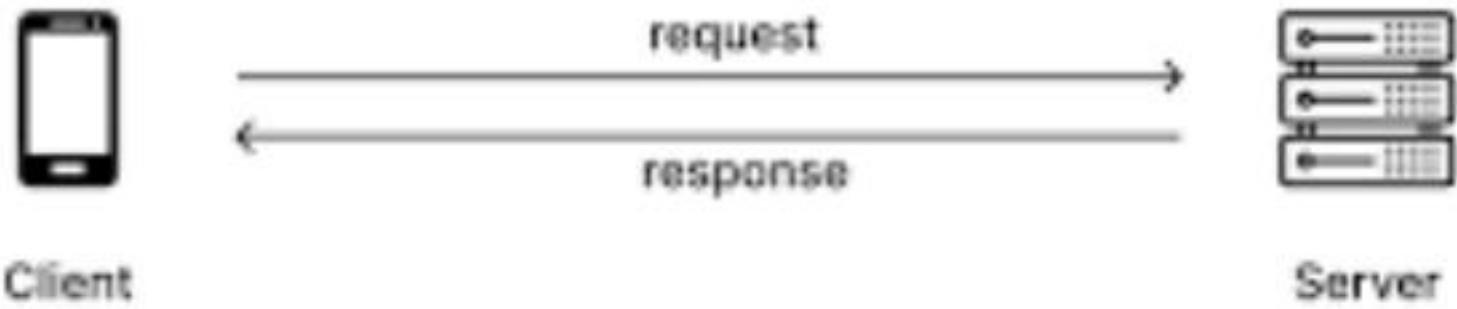
A DNS server is a computer with a database containing the public IP addresses associated with the names of the websites an IP address brings a user to. DNS acts like a phonebook for the internet. Whenever people type domain names, like Fortinet.com or Yahoo.com, into the address bar of web browsers, 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 Hyper Text Transfer Protocol

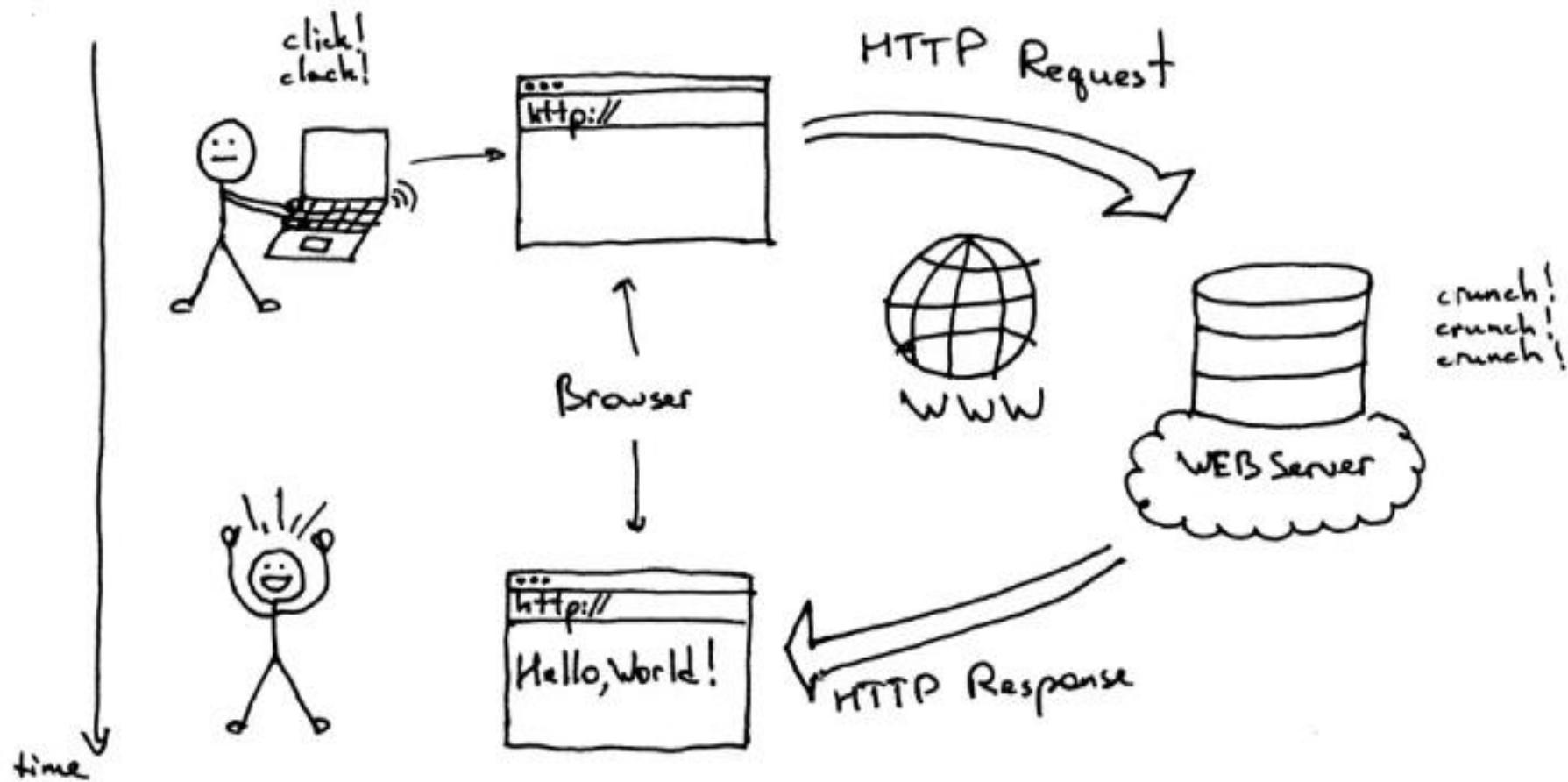
WWW is about communication between web **clients** and **servers**

Communication between client computers and web servers is done by sending **HTTP Requests** and receiving **HTTP Responses**



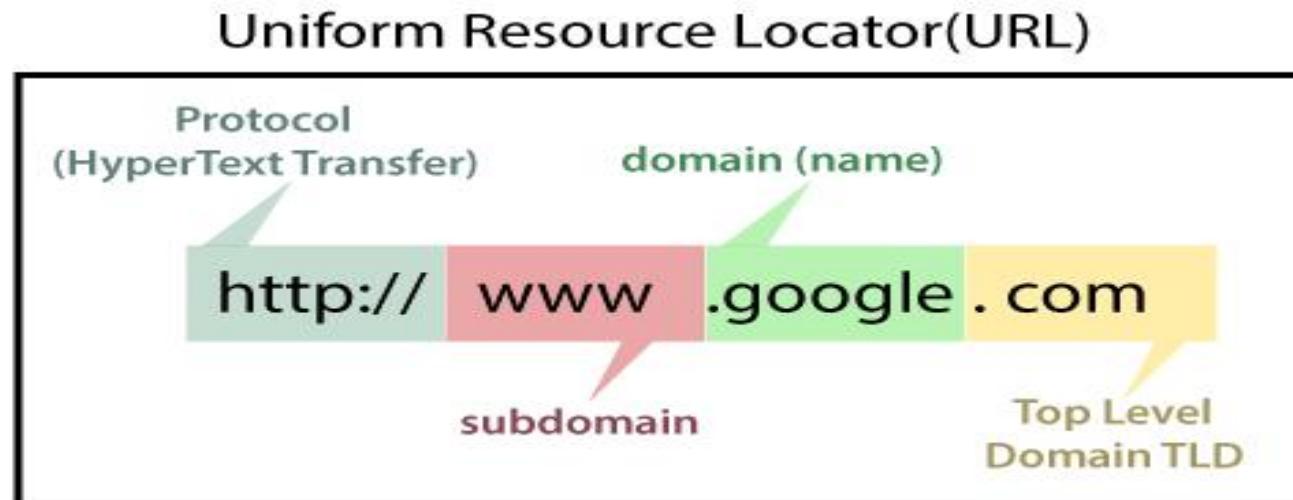
What is Browser and how it work ?

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'.



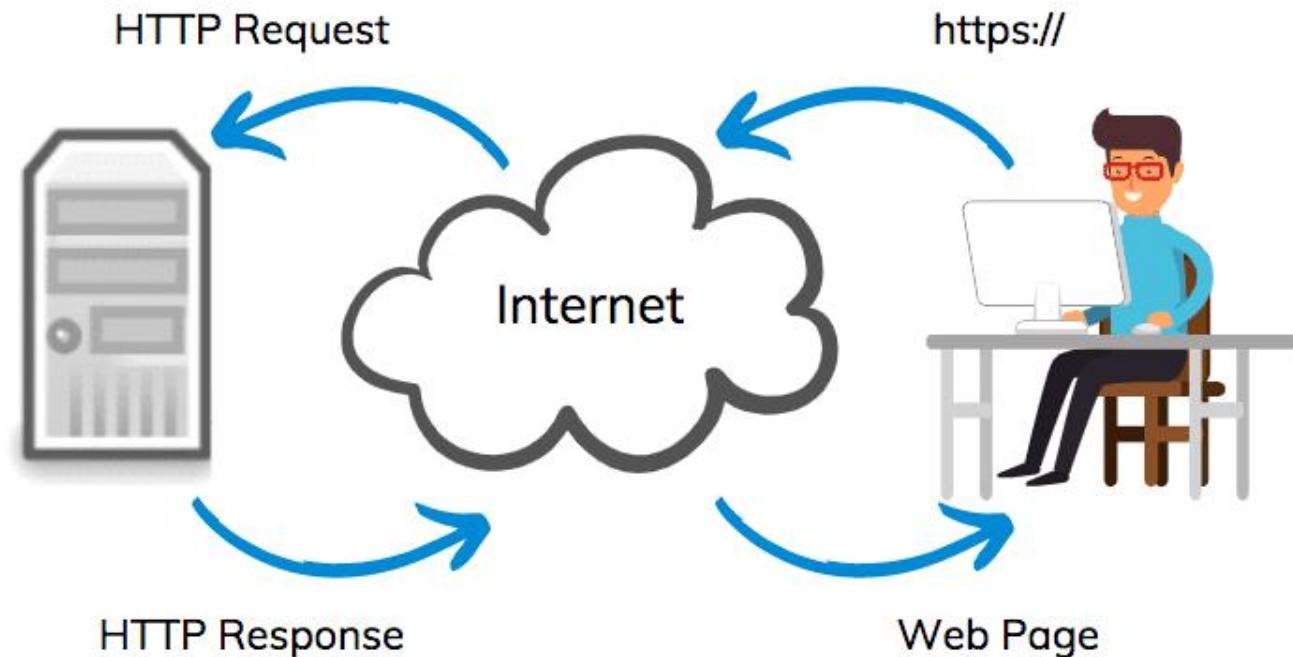
What is Hosting ?

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.

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

<https://github.com/Brijesh1990/brijeshgittraining.git>

```
echo "# brijeshgittraining" >>
README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Brijesh1990/brijeshgittraining.git
git push -u origin main
```

Module - 2

[Fundamentals of World Wide Web]

- ✓ Careers in Web Technologies and Job Roles
- ✓ 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

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.

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.

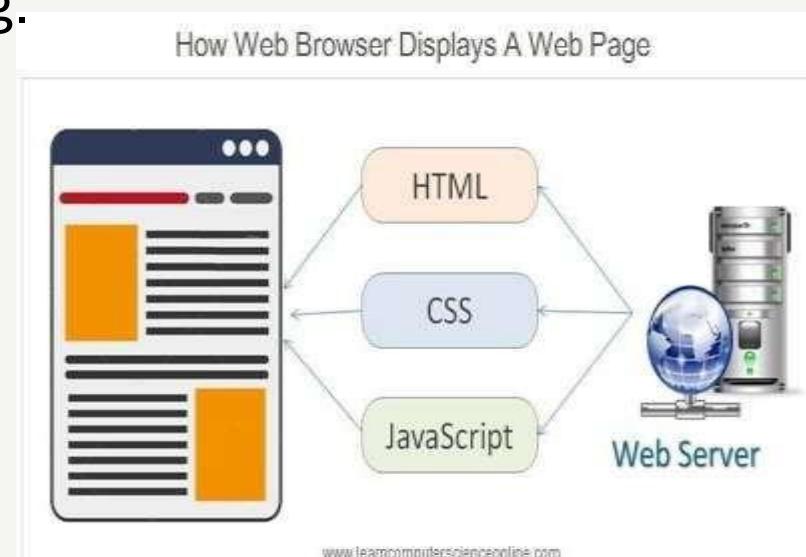


<https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/web%20broswer>

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 prebuilt 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.

<https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/wbesite>

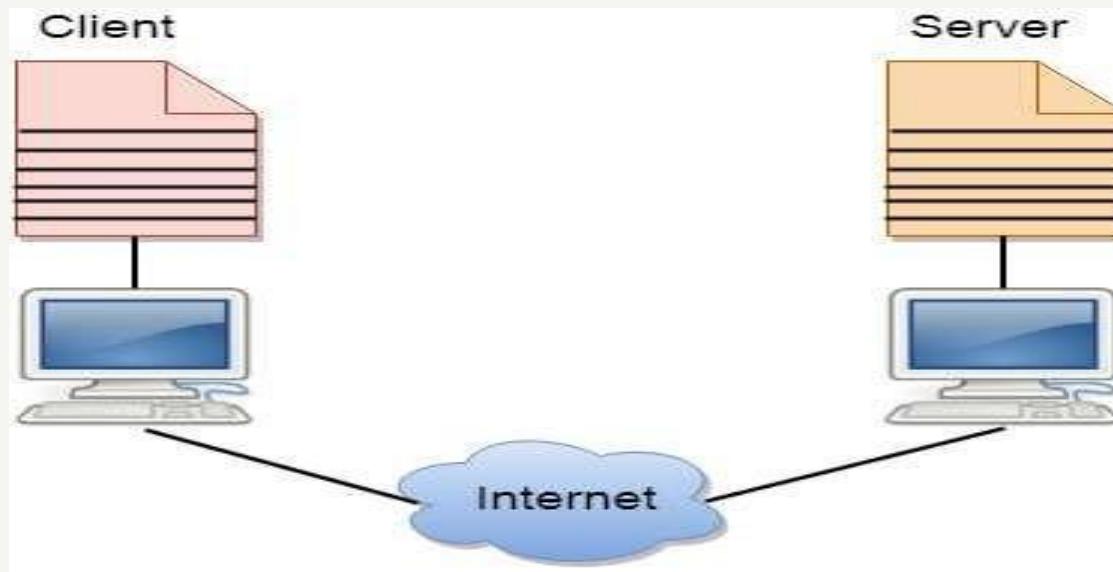
Client and Server

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

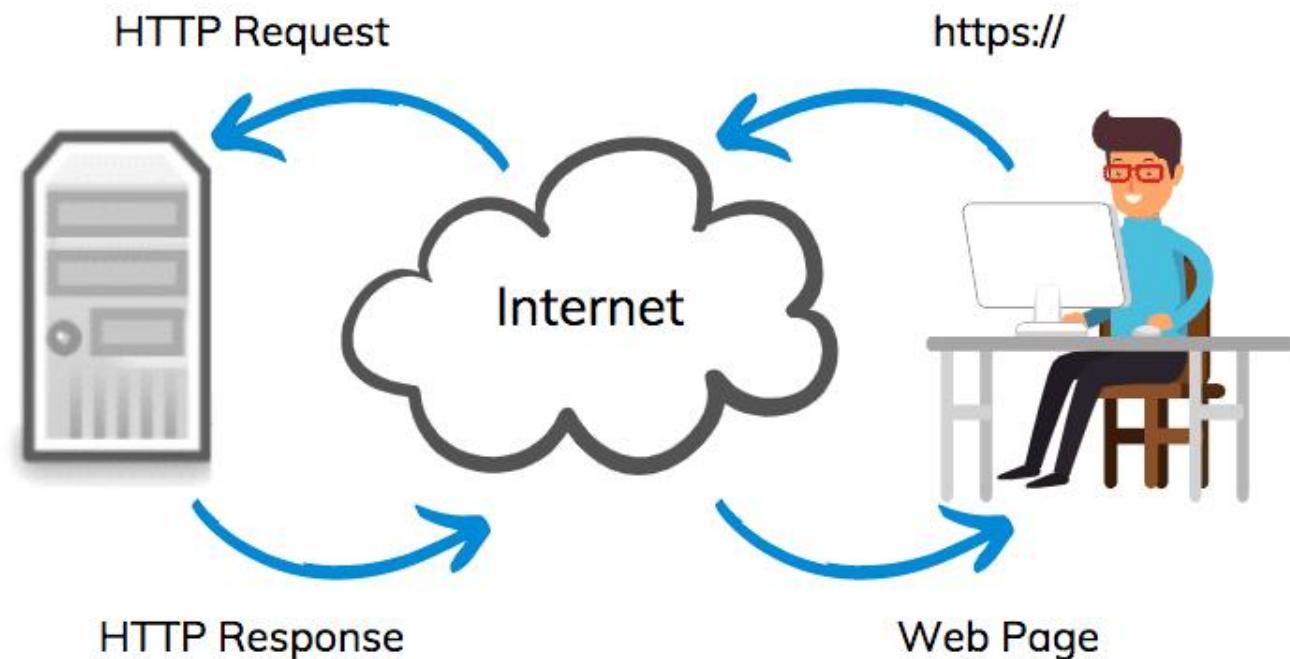
A Domain name is an address where one can find the website by typing the web address in the browser URL bar to visit a website. When you enter the domain name of the website in the search box, a powerful engine searches the web's largest pool of names and takes us to the website.



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



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.

<https://github.com/Brijesh1990/topswebsitedevelopment/tree/master/introductionWD/w3c>



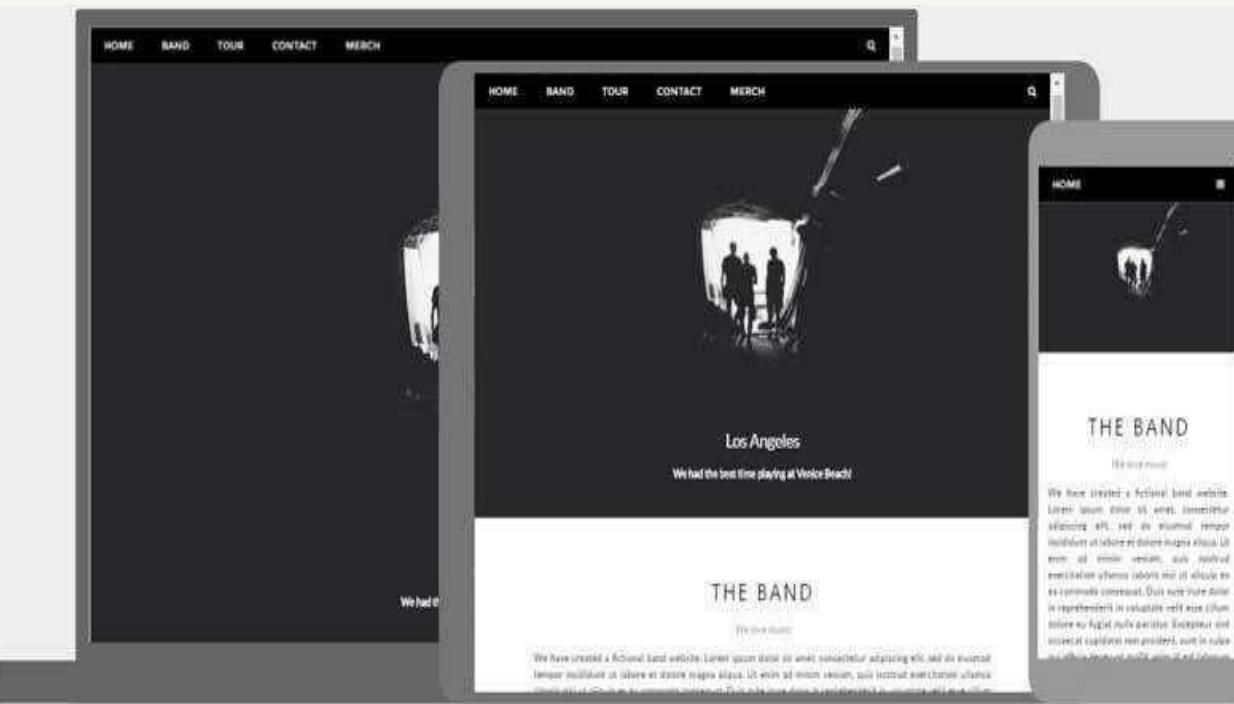
Responsive web designing

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:

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



<https://github.com/Brijesh1990/tops-website-development/tree/master/Live-projectexamples>

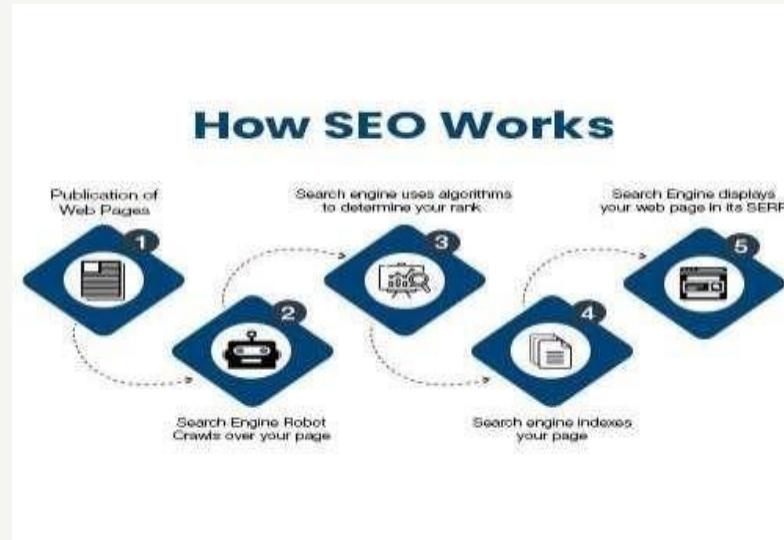
Protocol

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.

<https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/seo>



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.



Module - 3

[Fundamentals of IT]

- ✓ Careers in Web Technologies and Job Roles
- ✓ 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

GitHub Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

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.

GitHub Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

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.

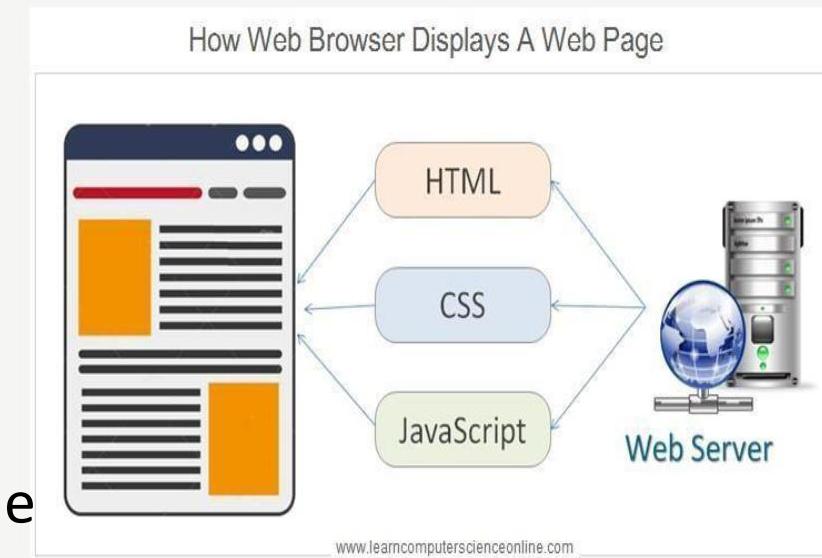


Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

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.



Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

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**.

Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html



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 .

Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

Dynamic website

In Dynamic Websites, Web pages are returned by the server which is processed during runtime means they are not prebuilt 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.

Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

Client and Server

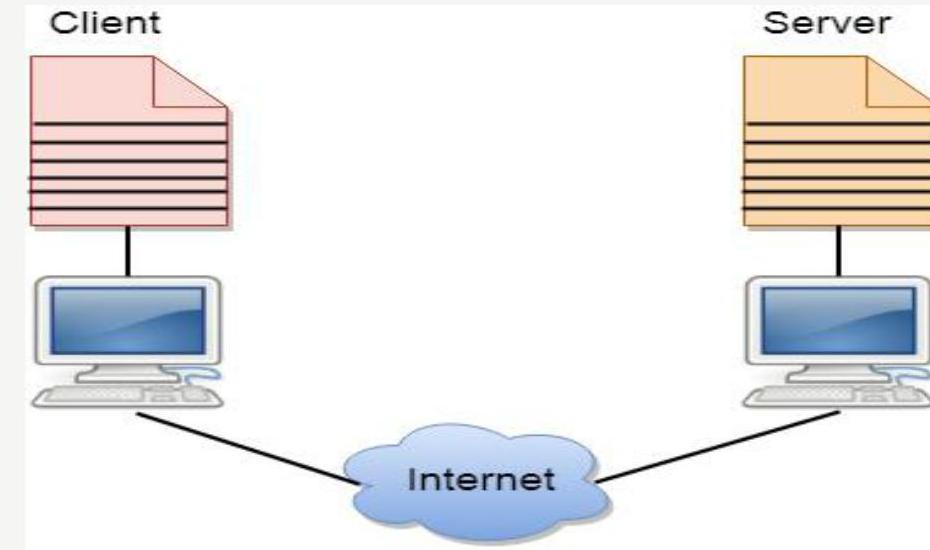
Client

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.

Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

Server

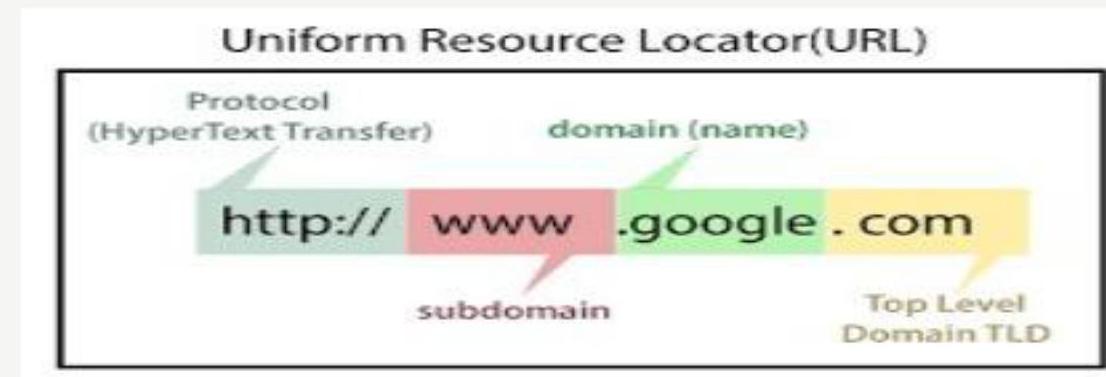
A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.



GitHub Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

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

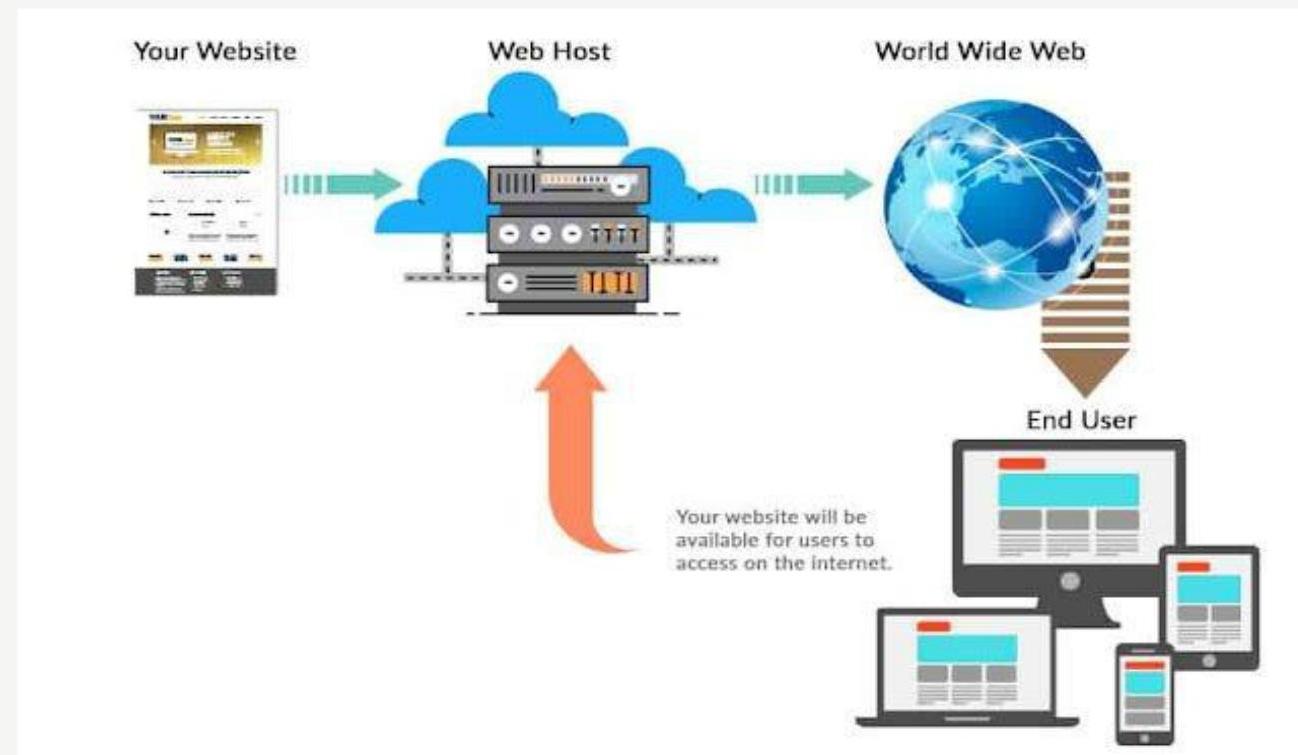


Hosting

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.

Github Link :https://github.com/TopsCode/WEB_DESIGNING/blob/main/Module-1/HTML/01_Intro.html

Hosting



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.



A screenshot of a W3C recommendation page. The page header includes the W3C logo and the title "Media Queries". Below the title, it states "W3C Recommendation 19 June 2012". There are two links: "This Version:" leading to <http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/> and "Latest Version:" leading to <http://www.w3.org/TR/css3-mediaqueries/>.

Responsive web designing

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

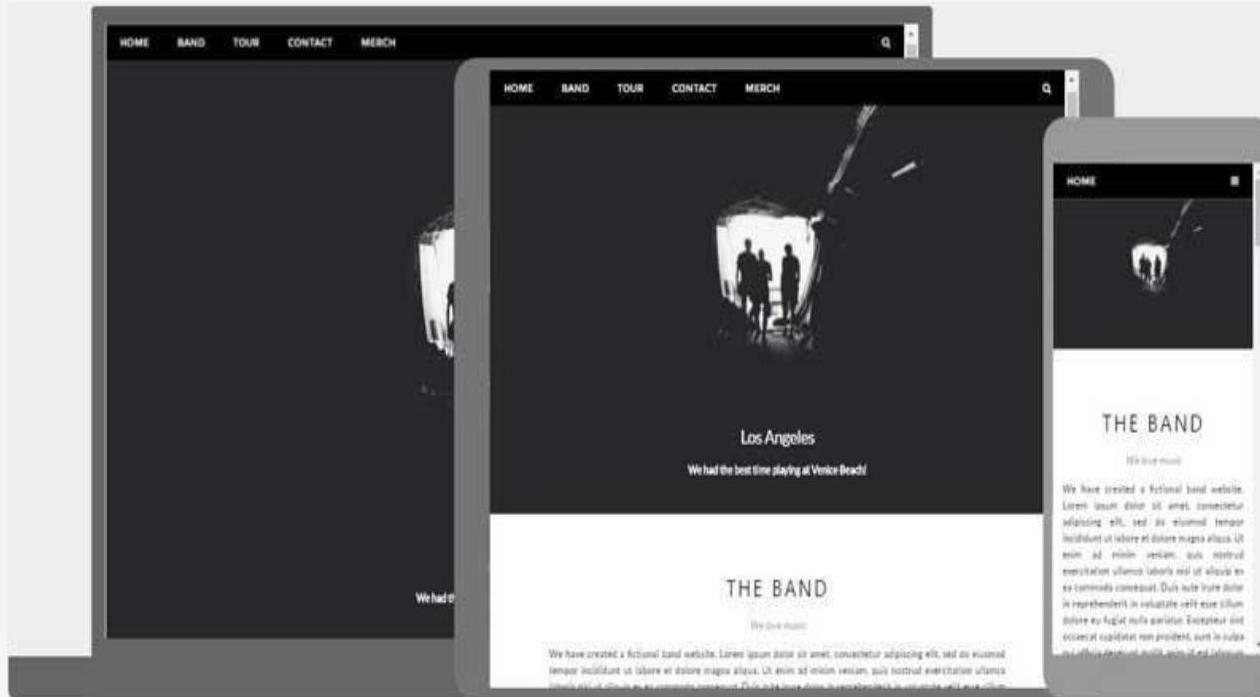
A responsive web design will automatically adjust for different screen sizes and viewports.

Setting The Viewport

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

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

Responsive web designing



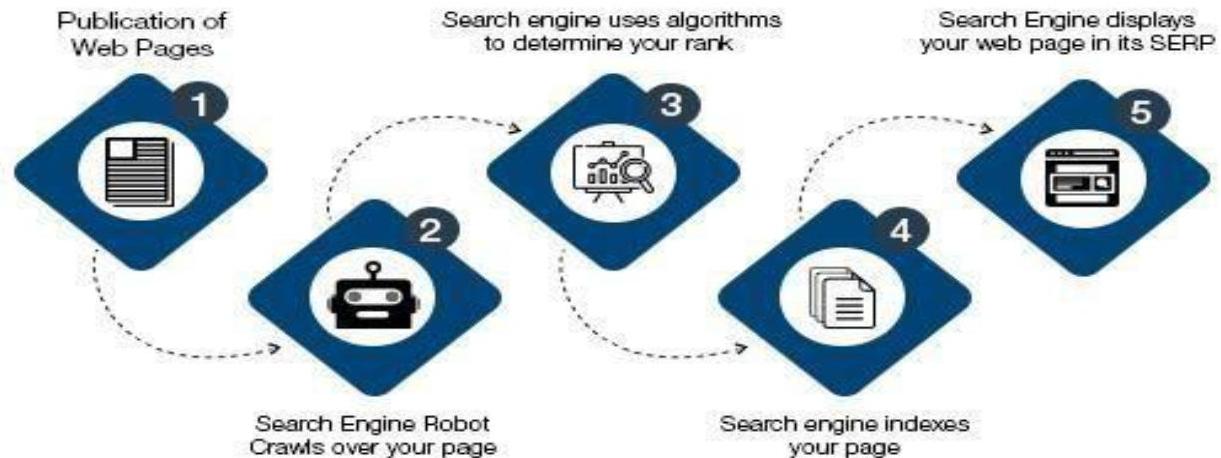
Protocol

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.

How SEO Works



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.



Web Development Tools

- Git and GITHUB Training
- Web development tools and environments are essential for building, testing, and deploying web applications. Here's an overview of some key tools and environments used in web development:
 - Code Editors and Integrated Development Environments (IDEs) - Visual Studio, sublime, atom, jetbrains
 - Version Control Systems - Git, Github, bitbucket
 - Package Managers - NPM, Yarn, composer
 - Task Runners and Build Tools - Grunt, Gulp, Webpack, Parcel

- Frameworks and Libraries - React, angular, vue.js, bootstrap, tailwind
- Development and Testing Servers - Node.js, Apache, Ngnix, Live Server
- Database management systems - MySQL, PostgreSQL, MongoDB, SQLite
- Front End Build Tools - SaSS, LESS, Babel
- Development Environments - XAMPP, MAMP
- Browser Developer Tools - Chrome DevTools, Firefox Developer Tools, Safari Developer Tools
- Collaboration and Communication Tools - trello, slack, Asana
- Continuous Integration and Deployment (CI/CD) Tool - Jenkins, Travis, CirceCI, Github Actionss
- Design and Prototyping Tools - Figma, Adobe XD, Sketch

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

- ✓ 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

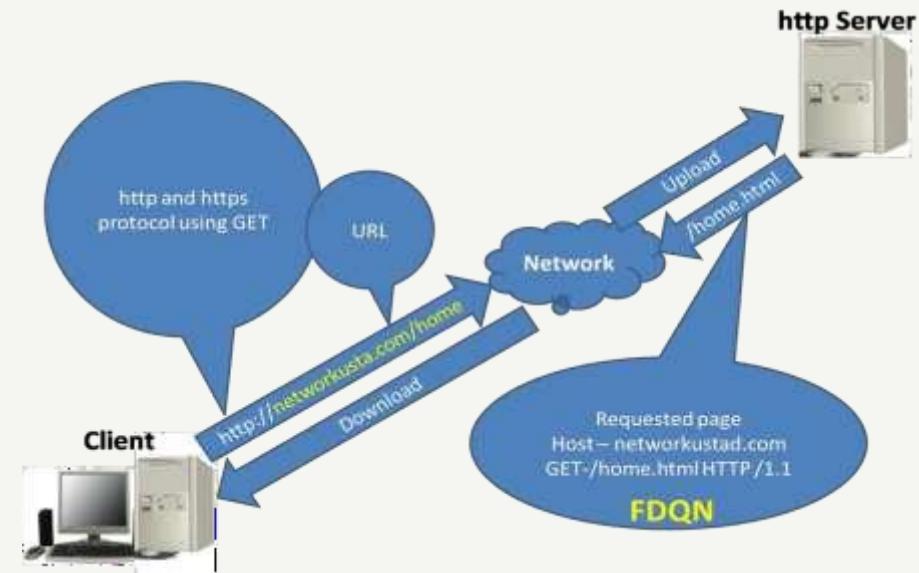
Firefox

Safari

Edge

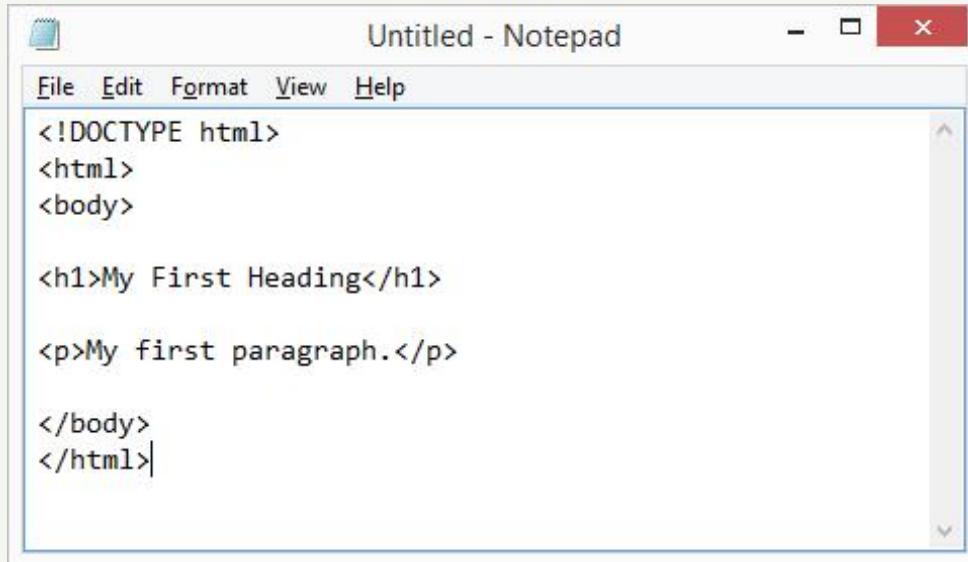
Hypertext Transfer Protocol

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



<https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/protocol/http>

Text Editor



Untitled - Notepad

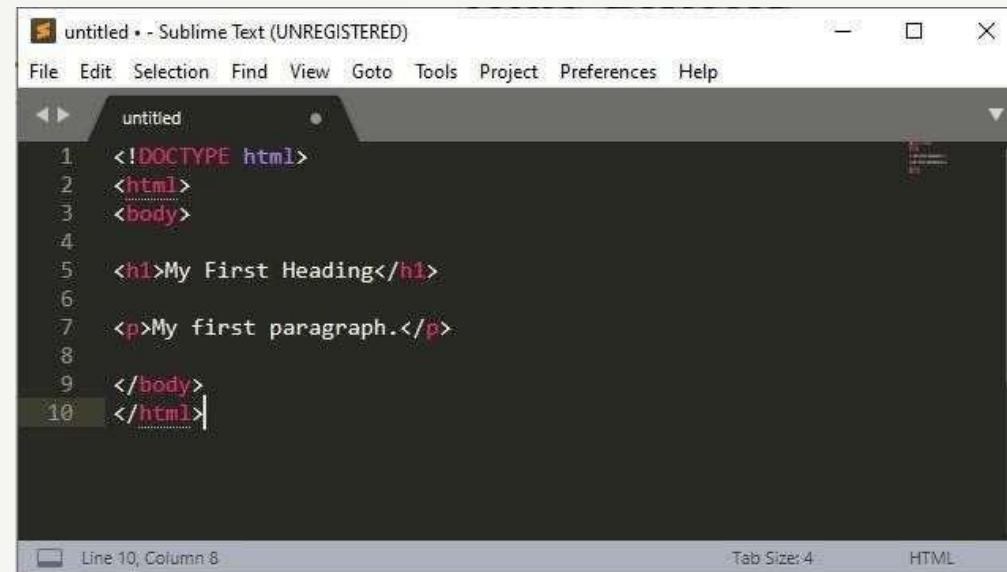
```
<!DOCTYPE html>
<html>
<body>

<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.

<https://github.com/Brijesh1990/tops-website-development/tree/master/introductionWD/introduction-html>

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.

- 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.

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> ,

Example 1

Write an HTML program to print “Hello World”.

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>Hello world.</p>
</body>
</html>
```

HTML Meta Tags

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

HTML Text Tags

```
<p>  
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>  
<strong>  
<em>  
<cite>
```

HTML Unclosed Tags

<hr>

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.

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.

<hr>

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

Ex. <div>
 <h1>-<h6>
 <hr>

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.

Ex. <a>

<label>

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 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.

Syntax

```
<element attribute_name="value">content</element>
```

Ex.

```
<body text="green" bgcolor="orange">
```

Ex.

```
<h1 title="This is heading tag">Example of title  
attribute</h1>
```

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

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:
4. Emphasized:
5. Definition: <dfn>
6. Quoting: <blockquote cite="">, <cite>
7. Short: <q>
8. Code: <code>
9. Keyboard: <kbd>
10. Address: <address>

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.

Tags

Following is a list of tags used in metadata:

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

<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">
```

<base>

```
<base href="https://www.learnvern.com/images/" target="_blank">
```

[https://github.com/Brijesh1990/tops-website-
development/tree/master/module1- html/head-tags](https://github.com/Brijesh1990/tops-website-development/tree/master/module1- html/head-tags)

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

`<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

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.

Syntax

&entity_name;

OR

&#entity_number;

Example

<	non-breaking space	
>	less than	<
&	greater than	>
&	ampersand	&
"	double quotation mark	"
'	single quotation mark (apostrophe)	'
¢	cent	¢
£	pound	£
¥	yen	¥
€	Euro	€
©	copyright	©
®	registered trademark	®

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.

Ex.

A	Α	Α	GREEK CAPITAL LETTER ALPHA
B	Β	Β	GREEK CAPITAL LETTER BETA
Γ	Γ	Γ	GREEK CAPITAL LETTER GAMMA
Δ	Δ	Δ	GREEK CAPITAL LETTER DELTA
E	Ε	Ε	GREEK CAPITAL LETTER EPSILON
Z	Ζ	Ζ	GREEK CAPITAL LETTER ZETA
←	←	←	LEFTWARDS ARROW
↑	↑	↑	UPWARDS ARROW
→	→	→	RIGHTWARDS ARROW
↓	↓	↓	DOWNWARDS ARROW

Find GitHub links

<https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-entity>

<https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-entity>

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 .

<https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/table>

Example:

```
<table>  
  <tr>  
    <th>1 header</th>  
    <th>1 header</th>  
    <th>1 header</th>  
  </tr>  
  <tr>  
    <td>1data</td>  
    <td>1data</td>  
    <td>1data</td>  
  </tr>  
</table>
```

Table Supportive tags

```
<table>  
<tr>  
<th>  
<td>  
<caption>  
<tbody>  
<thead>  
<tfooter>
```

Attributes:

1. rowspan
2. colspan

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)

[https://github.com/Brijesh1990/tops-website-development/tree/master/module1-
html/body-tags/html-list](https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-list)

Ordered List or Numbered List

All the list items are marked with numbers by default.

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

Types:

1, I, i, A, a

Unordered List or Bulleted List

All the list items are marked with bullets.

```
<ul>
<li>Aries</li>
<li>Bingo</li>
<li>Leo</li>
<li>Oracle</li>
</ul>
```

Types:
disc, circle, square, none

Description List or Definition List

Entries are listed like a dictionary or encyclopedia.

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

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

Ex.

1.
2.
3.
4.

Types of File Paths

1. Absolute File Paths:

Absolute file path specifies full URL address. Ex.

```

```

2. Relative File Paths

```

```

<https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/html-iframe>

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

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

Syntax

```
<form action="server url"  
method="get/post">  
  
</form>
```

HTML Form Elements

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

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">
```

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:
- tel: Defines a single line text field for entering a search string. Defines an input field for entering the telephone number.

HTML Form Attributes

Form Attributes

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

1. **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.

1. **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.

Form Attributes

3. **autocomplete**: Enables an input field to complete automatically.
3. **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
5. **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

[https://github.com/Brijesh1990/tops-website-development/tree/master/module1-
html/body-tags/form](https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/form)

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.

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.

<https://github.com/Brijesh1990/tops-website-development/tree/master/module2-css%26css3-basic-advanced/css/Selector>

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

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

HTML Keyboard and Mouse Event Attributes

Keyboard Event Attributes

Attribute	Description
onkeydown	Triggers the event when the user presses down a key on the keyboard.
onkeypress	Trigger the event when the user presses the key which displays some character.
onkeyup	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	Trigger the event when the mouse button is released.

Module-6

HTML 5

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.

Ex.

```
<audio controls>
  <source src="koyal.mp3"
  type="audio/mpeg"> Your browser does
not support the html audio tag.
</audio>
```

[https://github.com/Brijesh1990/tops-website-
development/tree/master/module3- html5/html5/content type tag/audio](https://github.com/Brijesh1990/tops-website-development/tree/master/module3- html5/html5/content_type_tag/audio)

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

Ex.

```
<video controls>
  <source src="movie.mp4" type="video/mp4">
  Your browser does not support the html video
  tag.
</video>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module3-html5/html5/content_type_tag

Video Attributes

Attribute	Description
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.
poster	It is used to set the width of the video player.
autoplay	It specifies the image which is displayed on the screen when the video is not played.
loop	It specifies that the video will start playing as soon as it is ready.
muted	It specifies that the video file will start over again, every time when it is completed.
preload	It is used to mute the video output.
src	It specifies the author view to upload video file when the page loads. It specifies the source URL of the video file.

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.

Ex. Circle

```
<svg width="100" height="100">  
  <circle cx="50" cy="50" r="40" stroke="yellow" stroke-width="4" fill="red" />  
</svg>
```

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

Ex. Ellipse

```
<svg height="140" width="500">
  <ellipse cx="200" cy="80" rx="70" ry="50"
style="fill:yellow;stroke:purple;stroke-width:2" />
Sorry, your browser does not support inline SVG.
</svg>
```

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

Ex. Rectangle

```
<svg width="200" height="100">  
  <rect width="200" height="100" stroke="yellow" stroke-width="4" fill="red" />  
</svg>
```

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

HTML SVG Line, Polygon, Polyline

SVG Line

Ex.

```
<svg height="210" width="500">  
  <line x1="0" y1="0" x2="200" y2="200" style="stroke:blue;stroke-width:2" />  
</svg>
```

https://github.com/Brijesh1990/tops-website-development/tree/master/module3-html5/html5/content_type_tag/SVG

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).

Ex.

```
<svg height="210" width="500">  
  <polygon points="200,10 250,190 160,210"  
  style="fill:lime;stroke:purple;stroke-width:1" />  
</svg>
```

SVG Polyline

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

Ex.

```
<svg height="200" width="500">
  <polyline points="20,20 40,25 60,40 80,120 120,140 200,180"
    style="fill:none;stroke:black;stroke-width:3" />
</svg>
```

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

Ex. Path

Ex.

```
<svg height="210" width="400">  
  <path d="M150 0 L75 200 L225 200 Z" />  
</svg>
```

SVG Text

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

Ex.

```
<svg height="30" width="200">
  <text x="0" y="15" fill="red">I love SVG!</text>
</svg>
```

```
<svg height="60" width="200">
  <text x="0" y="15" fill="red" transform="rotate(30 20,40)">
    I love SVG</text>
</svg>
```

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`

SVG Stroke Ex.

```
<svg height="80" width="300">
  <g fill="none">
    <path stroke="red" d="M5 20 l215 0" />
    <path stroke="black" d="M5 40 l215 0" />
    <path stroke="blue" d="M5 60 l215 0" />
  </g>
</svg>
```

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 $y1$ and $y2$ are equal and $x1$ and $x2$ differ
Vertical gradients are created when $x1$ and $x2$ are equal and $y1$ and $y2$ differ
Angular gradients are created when $x1$ and $x2$ differ and $y1$ and $y2$ differ

Ex. Linear Gradient

```
<svg height="150" width="400">
<defs>
  <linearGradient id="grad1" x1="0%" y1="0%" x2="100%" y2="0%">
    <stop offset="0%" style="stop-color:rgb(255,255,0);stop-opacity:1" />
    <stop offset="100%" style="stop-color:rgb(255,0,0);stop-opacity:1" />
  </linearGradient>
</defs>
<ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

Radial Gradient

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

Ex.

```
<svg height="150" width="500">
  <defs>
    <radialGradient id="grad1" cx="50%" cy="50%" r="50%" fx="50%"
      fy="50%">
      <stop offset="0%" style="stop-
        color:rgb(255,255,255); stop-opacity:0" />
      <stop offset="100%" style="stop-color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55" fill="url(#grad1)" />
</svg>
```

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

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

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>
```

Line on Canvas

```
<canvas id="myCanvasLine" width="200" height="100"  
style="border:1px solid #d3d3d3;">  
Your browser does not support the HTML5 canvas tag.</canvas>  
<script>  
var c =  
document.getElementById("myCanvasLine"); var  
cctx = c.getContext("2d");  
cctx.moveTo(0,0);  
cctx.lineTo(200,100);  
cctx.stroke();  
</script>
```

Circle on Canvas

```
<canvas id="myCanvasCircle" width="200" height="100"  
style="border:1px solid #d3d3d3;">  
Your browser does not support the HTML5 canvas tag.</canvas>  
<script>  
var c =  
document.getElementById("myCanvasCircle"); var  
cctx = c.getContext("2d");  
cctx.beginPath();  
cctx.arc(95,50,40,0,2*Math.PI)  
; cctx.stroke();
```

Text on Canvas

1. Fill Text

```
<canvas id="myCanvasText1" width="300" height="100"  
style="border:1px solid #d3d3d3;">  
Sorry! Your browser does not support the HTML5 canvas tag.</canvas>  
<script>  
var c =  
document.getElementById("myCanvasText1"); var  
cctx = c.getContext("2d");  
cctx.font = "30px Arial";  
cctx.fillText("Hello", 100, 50);</script>
```

Text on Canvas

2. Stroke Text

```
<canvas id="myCanvasText2" width="300" height="100" style="border:1px solid #d3d3d3;">  
Sorry!Upgrade your browser. It does not support the HTML5 canvas tag.</canvas>  
<script>  
var c = document.getElementById("myCanvasText2");  
var cctx = c.getContext("2d");  
cctx.font = "30px Arial";  
cctx.strokeText("Hello JavaTpoint",10,50);  
</script>
```

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 is used to define a domain prefix (default for http is www).
- domain is used to define the Internet domain name (like lernvern.com).
- port is used to define the port number at the host (default for http is 80).
- path is used to define a path at the server (If omitted: the root directory of the site).
- filename 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

3. Elements must always be properly nested

3. 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.

Ex.

```
<p>Click the button to get your coordinates.</p>

<button onclick="getLocation()">Try It</button>

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

<script>
var x = document.getElementById("demo");

function getLocation() {
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
```

Ex.

```
} else {
    x.innerHTML = "Geolocation is not supported by this
browser.";
}
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude
    + "<br>Longitude: " + position.coords.longitude;
}

```

```
</script>
```

Handling Errors

```
function showError(error) {  
    switch(error.code) {  
        case error.PERMISSION_DENIED:  
            x.innerHTML = "User denied the request for Geolocation." break;  
        case error.POSITION_UNAVAILABLE:  
            x.innerHTML = "Location information is unavailable." break;  
        case error.TIMEOUT:  
            x.innerHTML = "The request to get user location timed out." break;  
        case error.UNKNOWN_ERROR:  
            x.innerHTML = "An unknown error occurred." break;  
    }  
}
```

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.

Ex.

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev)
{ ev.preventDefault();
}

function drag(ev)
{ ev.dataTransfer.setData("text",
ev.target.id);
}
```

function drop(ev) { ev.preventDefault()

Ex.

```
var data = ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data))
;
}
</script>
</head>
<body>
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>



</body>
</html>
```

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)

Local Storage Ex.

```
<div id="result"></div>
<script>
// Check browser support
if (typeof(Storage) !== "undefined") {
    // Store
    localStorage.setItem("lastname", "Smith");
    // Retrieve
    document.getElementById("result").innerHTML = localStorage.getItem("lastname");
} else {
    document.getElementById("result").innerHTML = "Sorry, your browser does not
support Web Storage...";
}
</script>
```

Session Storage Ex.

```
<head>
<script>
function clickCounter() {
  if (typeof(Storage) !== "undefined") { if
    (localStorage.clickcount) {
      localStorage.clickcount = Number(localStorage.clickcount)+1;
    } else { localStorage.clickcount =
      1;
    }
    document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + "
time(s).";
  } else {
    document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";
  }
}
</script>
</head>
```

Session Storage Ex.

```
<body>

<p><button onclick="clickCounter()" type="button">Click me!</button></p>
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<p>Close the browser tab (or window), and try again, and the counter will continue
to count (is not reset).</p>

</body>
```

HTML Web Worker API

How it works?

Create a Web worker in external Javascript:

```
var i = 0;

function timedCount()
{ i = i + 1;
  postMessage(i);
  setTimeout("timedCount()",500);
}

timedCount();
```

Note: postMessage() is used to send back the message to the html page

HTML Ex.

```
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
var w;

function startWorker() {
    if (typeof(Worker) !== "undefined")
        { if (typeof(w) == "undefined") {
            w = new Worker("demo_workers.js");
        }
        w.onmessage = function(event)
            { document.getElementById("result").innerHTML =
            event.data;
```

HTML Ex.

```
 } else {
    document.getElementById("result").innerHTML = "Sorry! No Web Worker
    support.";
}
}

function stopWorker()
{
    w.terminate();
    w = undefined;
}
</script>
```

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 Display and Position Properties
- ✓ CSS Buttons
- ✓ CSS Positioning

- ✓ 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

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.

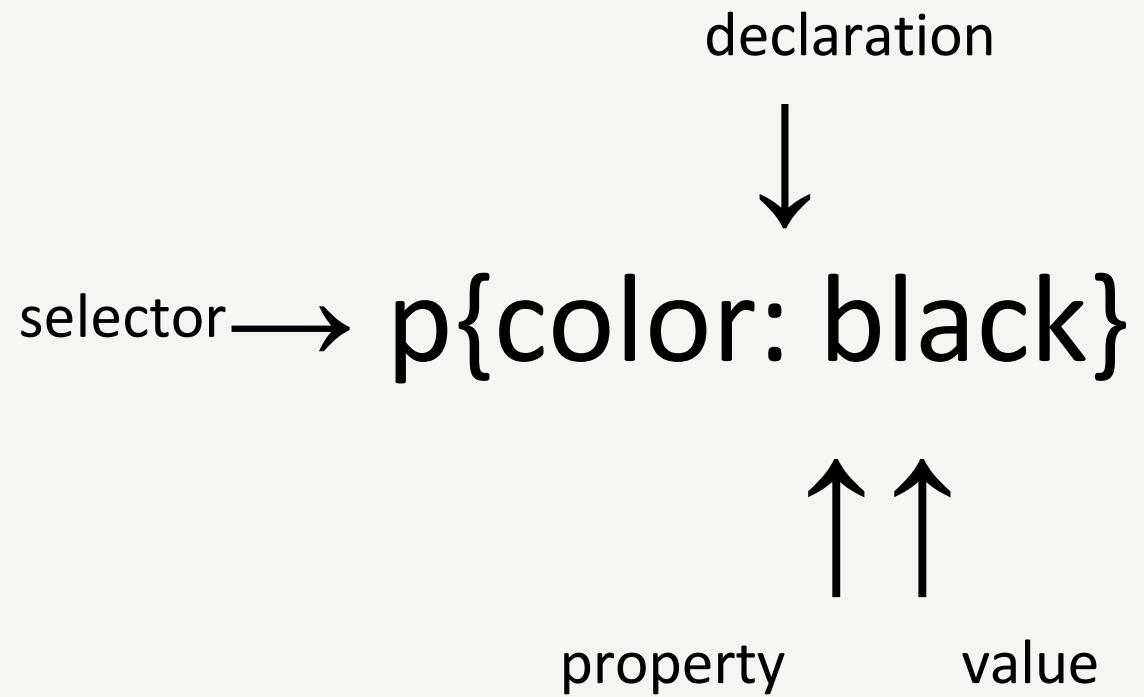
What is CSS

- 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.

Advantages of CSS

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

CSS Syntax



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

```
<body>
<h1>Write Your First CSS Example</h1>
<p>This is Paragraph.</p>
</body>
```

How to add CSS

```
<head>  
<style>  
h1{ color:white;  
background-color:red;  
padding:5px;  
}  
p{ color:blue;  
}  
</style>  
</head>
```

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 Universal Selector

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

Ex.

```
<style>
*
  color: green; font-
  size: 20px;
}
</style>
<h2>This is heading</h2>
<p>This style will be applied on every paragraph.</p>
<p id="para1">Me too!</p>
<p>And me!</p>
```

CSS Element Selector

The element selector selects the HTML element by name.

Ex.

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

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

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.

Ex.

```
<style>
#para1 {
    text-align: center;
    color: blue;
}
</style>
<p id="para1">Hello Javatpoint.com</p>
```

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.

Ex.

```
<style>
p.center {
    text-align: center;
    color: blue;
}
</style>
<h1 class="center">This heading is not affected</h1>
<p class="center">This paragraph is blue and center-aligned.</p>
```

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.

```
<style> h1, h2, p
{
    text-align: center; color:
    blue;
}
</style>
```

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

(\sim) uses the tilde (\sim) sign as the separator between the elements. It selects the 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*/  
}
```

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*/  
}
```

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.

The parent element must always be placed at the left of the ">". If we remove the greater than (>) symbol that designates this as a child combinator, then it will become the descendant selector.

Syntax:

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

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*/  
}
```

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;  
}
```

:hover

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

Ex.

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

:active

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

Ex.

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

:visited

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

Ex.

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

: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>

: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;  
}
```

: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;  
}
```

: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;  
}
```

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).
- Padding properties (such as padding-top, padding-right, padding-bottom, and padding-left).
- Background properties (such as background-color, background-repeat, background-image, and background-position).
- Text related properties (such as text-shadow, text-transform, text-decoration, etc.).
- Other properties are vertical-align (only when the float is 'none') word-spacing, line-height, line-spacing, etc.

::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.

::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.";  
}
```

::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";  
}
```

::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;  
}
```

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.

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.

Syntax

<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>

Ex.

```
<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>
<p>This paragraph is not affected.</p>
```

<https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/1cssInline.html>

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.

Ex.

```
<style>  
body {  
    background-color: linen;  
}  
  
h1 {  
    color: red; margin-left: 80px;  
}  
</style>
```

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>
```

The external style sheet may be written in any text editor but must be savedwith a .css extension. This file should not contain HTML elements.

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.

Comments

Comments are single or multiple lines statement and written within /* */ .

Ex.

```
<style>  
p {  
    color: blue;  
    /* This is a single-line comment  
     */  
    text-align: center;  
}
```

/* This is
a multi-line
comment */
</style>

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.
- 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

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;
```

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;`

2. uppercase

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

Syntax:

text-transform: uppercase;

3. lowercase

It transforms all characters of the word into lowercase.

Syntax:

```
text-transform: lowercase;
```

4. none

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

Syntax:

`text-transform: none;`

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;`

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;

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

Ex.

```
#lr {  
writing-mode: vertical-lr;  
text-orientation: mixed;  
}
```

```
#rl {  
writing-mode: vertical-rl;  
text-orientation: 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.

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.

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

```
h2 { color: #9000A1; }
```

```
p { color:rgb(0, 220, 98); }
```

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.

F

Sans-serif

F

Serif

CSS Font Family

Ex.

```
h1 { font-family: sans-serif;  
} h2 { font-family: serif; }  
p { font-family: monospace;  
}
```

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;

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; }
```

CSS Font Variant

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

Ex.

```
p { font-variant: small-caps;  
} h3 { font-variant: normal; }
```

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).

Ex.

```
font-weight:bold;  
font-  
weight:bolder;  
font-weight:lighter;  
font-weight:100;
```

```
font-weight:200;  
font-weight:300;  
font-weight:900;
```

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.

Ex.

font-stretch: normal / semi-condensed / condensed / extra-condensed / ultra-condensed / semi-expanded/ expanded / extra-expanded / ultra-expanded

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.

```
<style>  
p{  
    background-color: green;  
}  
</style>
```

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.

```
<style>
body {
background-image:
url("image.jpg"); margin-left:100px;
}
</style>
```

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.

```
<style>
body {
    background-image: url("gradient_bg.png");
    background-repeat: repeat-x;      or      background-repeat:
    repeat-y;
}
</style>
```

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.

```
<style>  
background: white  
url('image.jpg'); background-  
repeat: no-repeat;  
background-attachment: fixed;  
</style>
```

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.

```
background: white url('good-morning.jpg');  
background-repeat: no-repeat;  
background-attachment: fixed;  
background-position: center;
```

<https://github.com/Brijesh1990/tops-website-development/tree/master/module2-css%26css3-basic-advanced/css/background>

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.

border-style: none;	border-style: groove;
border-style: dotted;	border-style: ridge;
border-style: dashed;	border-style: inset;
border-style: solid;	border-style:
border-style:	outset; border-style:
double;	hidden
;	

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.

```
<style>
p.one {
    border-style: solid;
    border-width: 5px; / border-width: medium; / border-width: 1px;
}
</style>
```

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.

```
p.one {  
    border-style: solid;  
    border-color: red;  
}  
  
p.two {
```

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.

Variations are as below:

Single Value - border-radius: 30px;

Two Value - border-radius: 20% 10%

;

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.

```
table {  
border-collapse: separate; / border-collapse: collapse;  
}
```

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.

```
table {  
border-collapse:  
separate; border-spacing:
```

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.

Ex.

```
.Outline {  
    background-color: #eee;  
    border: 3px solid  
    Lightgreen; padding: 5px  
    10px;  
    outline-width: 3px;  
    outline-style: solid;  
    outline-color: red;
```

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

7. CSS Outline...

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

.Outline-

```
Offset{ background-
color: #eee; outline:
3px solid red; outline-
offset: 6px; border: 3px
solid
Lightgreen; padding: 5px
10px;
```

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.

```
border-image: url('border.png') 60 / 20px 20px round;
```

<https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/border.html>

8. CSS Border Image...

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:

1. 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

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.

Syntax:

list-style-position:inside;

list-style-position:outside;

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);
```

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);
```

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:

<a>

 etc.

Ex.

```
p {
```

```
display: inline;
```

```
}
```

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.

```
p {  
display: inline-block;  
}
```

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.

```
p {  
display: block;  
}
```

4. none

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

Ex.

```
hidden {  
    display: none;  
}
```

5. flex

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

Ex.

```
p {  
display: flex;  
}
```

<https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/14Advance5Display.html>

<https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/12Advance5.1Display.html>

CSS Flex

CSS Flexbox Layout Module

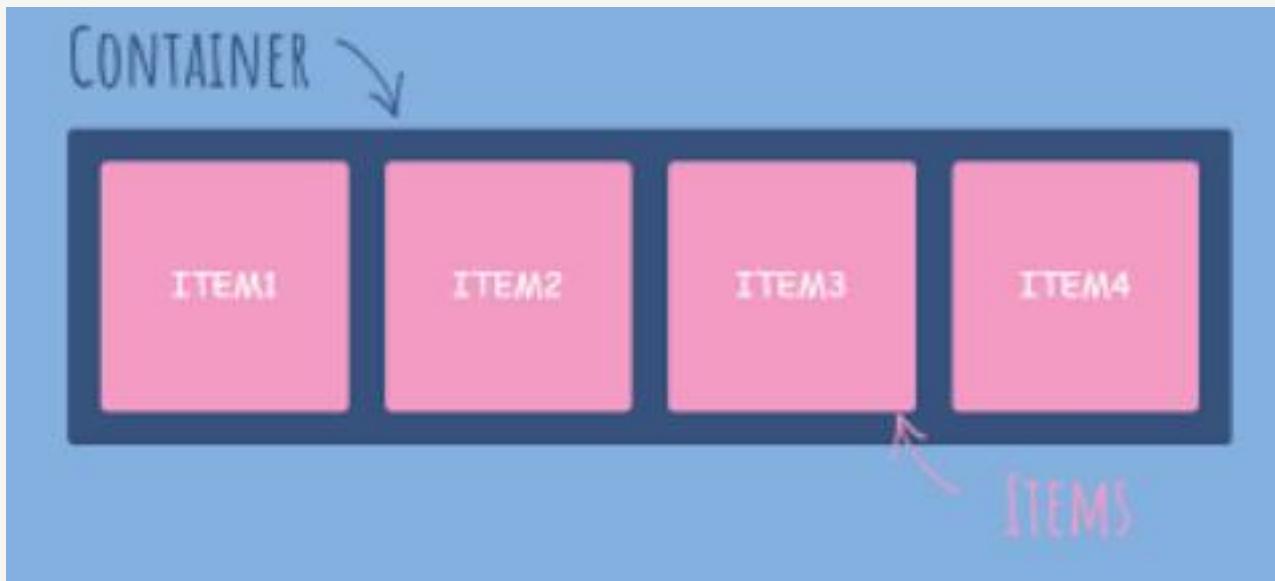
Before the Flexbox Layout module, there were four layout modes:

- ✓ 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.

CSS Flex Items

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

```
<style>
.flex-container {
  display: flex;
  background-color: #f1f1f1;
}
.flex-container > div
{ background-color:
  DodgerBlue; color: white;
width: 100px;
margin: 10px;
text-align: center;
line-height: 75px;
font-size: 30px;
}
</style>
```



```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;}
.flex-container > div
{ background-color:
#f1f1f1; margin: 10px;
padding: 20px;
font-size: 30px;}
</style>
</head>
<body>
<h1>Create a Flex Container</h1>
<div class="flex-container">
<div>1</div>
<div>2</div>
<div>3</div>
</div>
</body>
</html>
```

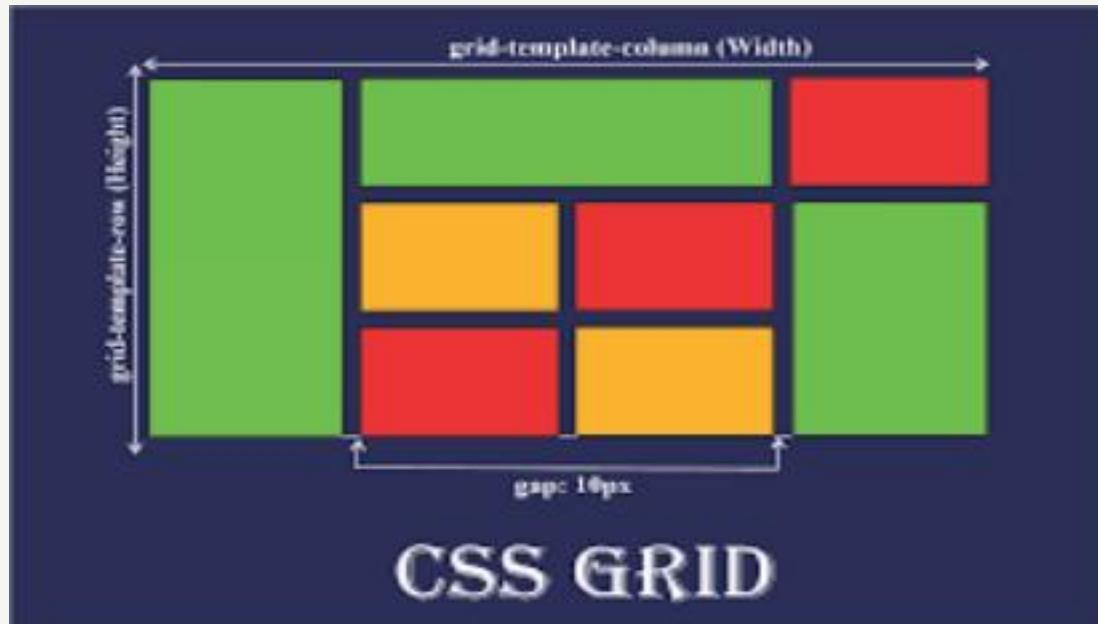


CSS Grid

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.

```
<div class="grid-container">  
  <div class="grid-item">1</div>  
  <div class="grid-item">2</div>  
  <div class="grid-item">3</div>  
  <div class="grid-item">4</div>  
  <div class="grid-item">5</div>  
  <div class="grid-item">6</div>  
  <div class="grid-item">7</div>  
</div>
```



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.

```
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
```

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:

```
.item1 {  
    grid-column: 1 / 5;  
}
```

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;
```

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;
```

3. CSS Relative Positioning

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

Ex.

```
position: relative;
```

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;
```

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;

5. CSS Sticky Ex.

```
.stick{  
    position: sticky;  
    top:50px; padding:  
    10px; font-  
    size:20px; font-  
    weight:bold;  
    background-color:  
    lightblue; border: 1px solid  
    blue;  
}
```

CSS Positions

CSS Positions

CSS Positioning property provides the possible ways by which we can set the element in different positions as shown below:

1. top
2. left
3. right
4. bottom

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.

Syntax:

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

Ex.

```
.len{  
    left/right/top/bottom : length in px;  
}  
.per{  
    left/right/top/bottom : length in %;  
}  
.auto{  
    left/right/top/bottom : auto;  
}
```

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.

Ex.

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

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

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.

background-color

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

Ex.

```
button {  
    color:lightgoldenrodyellow;  
}
```

border

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

Ex.

```
button {  
    border:none; / border:5px brown solid; / border: 5px red dashed;  
    / border: 5px black dotted; / border:5px blue double;  
}
```

border-radius

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

Ex.

```
button {  
    border-radius: value in px or %;  
}
```

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.

```
button {  
}           box-shadow : 0 8px 16px 0 black,      0 6px 20px 0 rgba(0, 0, 0, 0.19);  
button :hover{  
    box-shadow : 0 8px 16px 0           0 6px 20px 0 rgba(0, 0, 0, 0.19);  
    black,  
}
```

padding

It is used to set the button padding.

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

Ex.

```
button {  
    padding: value in px;  
}
```

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.

Float Example

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

<https://github.com/Brijesh1990/tops-website-development/blob/master/module2-css%26css3-basic-advanced/css/advance/15Advance6.1Relativeposition.html>

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.

```
p{  
    clear:right;  
}
```

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.

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);`

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);
```

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 to arrive at an equivalent hexadecimal value.

For example, #7B6 becomes #77BB66 in hexadecimal.

HSL

It is a short form of Hue, Saturation, and Lightness.
Let's understand them individually.

Hue: It can be defined as the degree on the color wheel from 0 to 360. 0 represents red, 120 represents green, 240 represents blue.

Saturation: It takes value in percentage in which 100% represents fully saturated, i.e., no shades of gray, 50% represent 50% gray, but the color is still visible, and 0% represents fully unsaturated, i.e., completely gray, and the color is invisible.

Lightness: The lightness of the color can be defined as the light that we want to provide the color in which 0% represents black (there is no light), 50% represents neither dark nor light, and 100% represents white (full lightness).

Syntax:

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);

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;
```

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.

Ex.

```
element {  
    font-size: 14px  
    !important; color: blue  
        !important;  
    ...  
}
```

Ex.

```
h1 {  
    border-color: red;  
    border: 5px green  
    solid; border-color:  
    black;  
}
```

```
h1 {  
    border-color: red  
    !important;  
    border: 5px green solid;
```

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.

line-height: value in
numbers; line-height: value
in %;

line-height: value in px, pt, cm;

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:

- margin
- margin-left
- margin-right
- margin-top
- margin-bottom

CSS Margin

Ex.

```
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;
```

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.

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;

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();

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 %);`

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);
```

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);
```

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);
```

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);
```

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%);
```

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%);
```

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);
```

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:

```
filter: grayscale(80%);
```

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%);
```

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;  
}
```

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:5px;  
padding:10px;  
opacity:0.3;  
}
```

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%;  
}
```

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;  
}
```

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:auto;  
display:block;  
}
```

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

CSS Overflow Ex

```
<style>

div.scroll {
    background-color:
        #00ffff; width: 100px;
    height: 100px;
    overflow: scroll / visible / hidden /
    auto;
}

</style>
```

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

Ex.

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;

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;

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;

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:

width: auto;

width:

length;

width: %;

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

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

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;
```

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;
```

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.

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.

Box-shadow Ex.

Ex.

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;

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

initial attribute
box-shadow: initial;

default attribute
box-shadow:
none;

Box-shadow Ex.

Ex.

initial attribute

```
box-shadow: initial;
```

default attribute

```
box-shadow:  
none;
```

Text-shadow

Ex.

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;
```

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`

Ex. with CSS

```
h1.visible
{ visibility:
  visible
}
h1.hidden
{ visibility:
  hidden
}
```

Ex. with javascript

```
function myFunction()
{ if(document.getElementById("myDIV").style.visibility ==
"visible"){ document.getElementById("myDIV").style.visibility =
"hidden";
}
else{
document.getElementById("myDIV").style.visibility = "visible";
}
```

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.

```
<link rel="stylesheet"  
      href="https://cdnjs.cloudflare.com/ajax/libs/font-  
      awesome/4.7.0/css/font-awesome.min.css">
```

```
    <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>
```

<https://github.com/Brijesh1990/tops-website-development/tree/master/module1-html/body-tags/font-aswome>

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>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-envelope"></i>
```

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>
```

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

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.

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

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;
```

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;

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;

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`

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.

: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:

```
:nth-child(n) {  
}  
}
```

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

Ex.

1. Using Function:

```
<style>  
    p:nth-child(2n+1) {  
        }  
</style>
```

2. Using even and odd.

```
p:nth-child(even) {  
    }  
p:nth-child(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:

```
p:last-child {  
}
```

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);`

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);  
}
```

```
.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.

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 |

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.

Ex.

```
element name/selector{  
    columns: (length) (column number);  
}
```

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

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.

Ex.

```
.none{ hyphens:  
none;  
}
```

```
.manual{ hyphen  
s: manual;  
}
```

```
.auto{ hyphens:  
auto;  
}
```

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.**

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.

Ex.

Syntax:

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

Ex:

for 2D Transformatio:

transform: rotate(35deg);

transform-origin: 5% 2%;

for 3D Transformatio:

transform: rotate3d(3, 2, 1, 75deg);

transform-origin: 70% 10% 150px;

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

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.

Ex.

```
div{  
    resize: none | horizontal | vertical |  
    both;  
}
```

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`

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.

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.

Ex. 1

- Change the color of the div on hover

```
div{  
width: 100px; height:  
100px;  
background: lightblue;  
transition-property: background-  
color; transition-duration: 1s;  
transition-delay: 0.5s;
```

```
/* For Safari browser */  
-webkit-transition-property:  
background-color;  
-webkit-transition-duration: 1s;  
-webkit-transition-delay: 0.5s;  
}  
div:hover {  
background-color: brown;
```

Ex. 2

- Change the width of div on hover

```
.first{  
width: 150px;  
height: 150px;  
background-color: lightblue;  
transition-property: width;  
transition-duration: 1s;  
transition-delay: initial;
```

```
.first:hover {  
width: 300px;  
}
```

Ex. 3

- Rotate the div on hover

```
padding:15px;  
width: 200px;  
height: 200px;  
background: lightgreen;  
transition: background-color  
           1s, width 2s, height 2s,  
transform 2s; transition-delay: 1.5ms;
```

```
div:hover  
{ width: 300px;  
height:  
300px;  
-webkit-transform:  
rotate(360deg);  
/* Chrome, Safari, Operarotate(360deg);  
background-color: orange;
```

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

Animation Properties

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.

```
div {  
    width: 100px;  
    height: 100px;  
    background:  
    red;  
    -webkit-animation: myfirst 6s; /* Chrome, Safari, Opera  
*/ animation: myfirst 5s;  
}
```

```
/* Chrome, Safari, Opera */  
@-webkit-keyframes myfirst  
{  
    from {background:  
    red;} to {background:  
    green;}  
}  
/* Standard syntax  
*/ @keyframes  
myfirst {  
    from {background:  
    red;} to {background:  
    green;}  
}
```

Ex. 2 with percentage values

- Moving Rectangle

```
div {  
    width: 100px;  
    height: 100px;  
    background: red;  
    position:  
    relative;  
    -webkit-animation: myfirst  
    5s;  
/* Chrome, Safari, Opera  
 */ animation: myfirst 5s;  
}
```

```
@keyframes myfirst {  
    0% {background:red; left:0px;  
    top:0px;} 25% {background:yellow;  
    left:300px;  
    top:0px;}  
    50% {background:blue; left:300px;  
    top:200px;}  
    75% {background:green; left:0px; top:200px;}  
    100% {background:red; left:0px;  
    top:0px;}  
}
```

@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.

Syntax:

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.

Ex.

```
div  
{ width:200px;  
height:200px;  
animation:demo 5s ease-in  
infinite, trans 5s ease-in-out  
infinite; border-radius:40px;  
}
```

@keyframes demo

```
{ 0%  
background:red;}  
50% {background:yellow;  
width:100px; height:100px;}  
100% {background:green; width:300px;  
height:300px;}  
}
```

@keyframes trans

```
{ 0%{transform:translate(0px)scale(1.4)  
rotate(80deg);}  
50%{transform:translate(250px)scale(1.2)  
rotate(40deg);}
```

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.... );
```

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.

Ex.

```
#grad1 {  
    height: 100px;  
  
    background: -webkit-linear-gradient(red, green); /* For Safari 5.1 to 6.0 */  
    background: -o-linear-gradient(red, green); /* For Opera 11.1 to 12.0 */  
    background: -moz-linear-gradient(red, green); /* For Firefox 3.6 to 15 */  
    background: linear-gradient(red, green); /* Standard syntax (must be last) */  
}
```

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.

```
#grad1 {  
    height: 100px;  
    background: -webkit-linear-gradient(left, red, green); /* For Safari 5.1 to 6.0 */  
    background: -o-linear-gradient(right, red, green); /* For Opera 11.1 to 12.0 */  
    background: -moz-linear-gradient(right, red, green); /* For Firefox 3.6 to 15 */  
    background: linear-gradient(to right, red , green); /* Standard syntax (must be last) */  
}
```

CSS Linear Gradient: (Diagonal)

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

Ex.

```
#grad1 {  
    height: 100px;  
    background: -webkit-linear-gradient(left top, red , green); /* For Safari 5.1 to 6.0 */  
    background: -o-linear-gradient(bottom right, red, green); /* For Opera 11.1 to 12.0 */  
    background: -moz-linear-gradient(bottom right, red, green); /* For Firefox 3.6 to 15 */  
    background: linear-gradient(to bottom right, red , green); /* Standard syntax (must be last) */  
}
```

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);
```

CSS Radial Gradient: (Evenly Spaced Color Stops)

Evenly spaced color stops is a by default radial gradient. Its by default shape is eclipse, size is farthest- carner, and position is center.

Ex.

```
#grad1 {  
    height: 150px;  
    width: 200px;  
    background: -webkit-radial-gradient(blue, green, red); /* Safari 5.1 to 6.0 */  
    background: -o-radial-gradient(blue, green, red); /* For Opera 11.6 to 12.0 */  
    background: -moz-radial-gradient(blue, green, red); /* For Firefox 3.6 to 15 */  
    background: radial-gradient(blue, green, red); /* Standard syntax (must be last) */  
}
```

Radial Gradient: (Differently Spaced Color Stops)

Ex.

```
#grad1 {  
    height: 150px;  
    width: 200px;  
    background: -webkit-radial-gradient(blue 5%, green 15%, red 60%); /* Safari 5.1 to 6.0 */  
    background: -o-radial-gradient(blue 5%, green 15%, red 60%); /* For Opera 11.6 to 12.0 */  
    background: -moz-radial-gradient(blue 5%, green 15%, red 60%); /* For Firefox 3.6 to 15 */  
    background: radial-gradient(blue 5%, green 15%, red 60%); /* Standard syntax (must be last)  
*/  
}
```

Radial Gradient: (From Corners)

Ex.

```
#grad1 {  
    background-image: radial-gradient(farthest-side at left bottom, red, yellow, cyan );  
}  
  
#grad2{  
    background-image: radial-gradient(farthest-corner at right bottom ,blue, yellow,green);  
}  
  
#grad3{  
    background-image: radial-gradient(closest-side , red, yellow, lightgreen);  
}  
  
#grad4{  
    background-image: radial-gradient(closest-corner , blue, yellow, green);  
}
```

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`

Ex.

```
div{  
    position:fixed;  
    z-index:20;  
}  
  
img{  
    position:relative;  
    z-index:22;  
}  
  
h1{  
    position:relative;  
    z-index:28;  
}
```

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.

Ex. 1: Mask

```
#mask{  
-webkit-mask-box-image: url(box.png);  
}
```

Ex. 2: Gradient

```
#mask{  
-webkit-mask-image: -webkit-gradient(linear, right top, right  
bottom, from(rgba(0,0,0,0)), to(rgba(0,0,0,0.9))); |  
-webkit-mask-image: radial-gradient(circle at 50% 50%, blue 40%,  
rgba(0,0,0,0.3) 70%);  
border: 9px ridge red;  
}
```

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.



different screen resolutions

Smartphone: 320px

Tablet: 768px

Netbook: 1024px

Desktop: 1600px

Large Desktop: 1920px

Ex.

```
<meta name="viewport" content="width=device-width, initial-  
scale=1.0"/>  
<style>  
body {  
    background-color:yellow;  
}  
  
@media only screen and (max-width: 500px)  
{ body {  
    background-color:green;  
}  
}  
</style>
```

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()

skew()

matrix()

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);
```

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);
```

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);
```

skewX()

The CSS skewX() method is used to skew an element along the X axis according to the given angle.

Ex:

```
transform: skewX(30deg);
```

skewY()

The CSS skewY() method is used to skew an element along the Y axis according to the given angle.

Ex:

```
transform: skewY(30deg);
```

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);
```

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);
```

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 */
```

The 3D rotateY() method (Y-axis rotation)

The CSS 3D rotateY() method is used to rotate an element around its Y-axis according to the given degree.

Ex.:

```
transform: rotateY(150deg); /* Standard syntax */
```

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 */
```

Module - 8 [JavaScript]

JavaScript

- ✓ What is JavaScript, Creating First JavaScript Program, Way to apply JavaScript , Event in JavaScript ,
- ✓ How to select Tag Classes and Id
- ✓ JS Introduction
- ✓ JS Getting Started
- ✓ JS Syntax
- ✓ JS Variables
- ✓ JS Generating Output
- ✓ JS Data Types
- ✓ JS Operators
- ✓ JS Events
- ✓ JS If, Else JS Switch Case

- ✓ JAVASCRIPT & DOM
- ✓ JS DOM Manipulation
- ✓ JS DOM Navigation
- ✓ Practical Example: 1) Create program for input color and output that code 2) Create program for pattern using loop
- ✓ Functions, Alert ,Confirm , Prompt , Addition of Two Number , Hide and Show Password
- ✓ JS Loops, JS Functions
- ✓ Practical Example: 1) Slider • If Else Statement, JavaScript Form Validation • Practical Examples: 1) Get input data and perform different operations
- ✓ 2) Make dynamic CSS by click

Javascript

- JavaScript is an object-based scripting language which is lightweight and cross-platform.
- JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.



<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/head.html>

What is JavaScript

- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
- It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
- It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser.

What is JavaScript

- With JavaScript, users can build modern web applications to interact directly without reloading the page every time.
- The traditional website uses js to provide several forms of interactivity and simplicity.
- Although, JavaScript has no connectivity with Java programming language. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

- All popular web browsers support JavaScript as they provide built-in execution environments.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).

Features of JavaScript

- JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
- It is a light-weighted and interpreted language.
- It is a case-sensitive language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.
- It provides good control to the users over the web browsers.

Application of JavaScript

JavaScript is used to create interactive websites.

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

Example of Javascript

```
<script type="text/javascript">  
document.write("Welcome to  
LearnVern");  
</script>
```

- > The script tag specifies that we are using JavaScript.
- > The text/javascript is the content type that provides information to the browser about the data.

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/body.html>

How to run JavaScript

Ways to add JS code

1. Between the body tag of html

```
<body> <script>...</script> </body>
```

2. Between the head tag of html

```
<head> <script>...</script> </head>
```

3. In .js file (external javaScript)

```
<script src=""> </script>
```

1. code between the body tag

Ex.:

```
<html>
<body>
<script
type="text/javascript">
alert("Hello Javatpoint");
</script>
</body>
</html>
```

2. code between the head tag

Ex.:

```
<script type="text/javascript">  
function  
msg(){ alert("Hello  
Javatpoint");  
}  
</script>  
  
<form>  
<input type="button" value="click" onclick="msg()" />
```

</form>

3. In .js file (External JavaScript)

- It provides code re usability because single JavaScript file can be used in several html pages.
- An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.
- In the current file, add the .js file through the <script> tag

Ex.:

```
<head>
<script type="text/javascript" src="message.js"></script>
</head>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflicts.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
1. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
1. The web browser needs to make an additional http request to get the js code.

Disadvantages of External JavaScript

4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
4. We need to check each file that depends on the commonly created external javascript file.
4. If it is a few lines of code, then better to implement the internal javascript code.

Comments in JavaScript

What is Comment

- The JavaScript comments are meaningful way to deliver message.
- It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
- The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of Comments

Advantages of JavaScript comments

- There are mainly two advantages of JavaScript comments.
- To make code easy to understand It can be used to elaborate the code so that end user can easily understand the code.
- To avoid the unnecessary code It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

JavaScript Single line Comment

- It is represented by double forward slashes (//). It can be used before and after the statement.

Ex.

```
<script>  
// It is single line comment  
document.write("hello  
javascript");  
</script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

Ex.

```
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
```

Variables in JavaScript

Variables in JavaScript

- A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.
- There are some rules while declaring a JavaScript variable (also known as identifiers).
 1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
 2. After first letter we can use digits (0 to 9), for example value1.
 3. JavaScript variables are case sensitive, for example x and X are different variables.

Ex.

- Correct way to define javascript variable:

```
var x = 10;
```

```
var _value="sonoo";
```

- Incorrect way to define Javascript variable:

```
var 123=30;
```

```
var *aa=320;
```

Ex.

```
<script>  
    var x = 10;  
    var y = 20;  
    var z=x+y;  
    document.write(z);  
</script>
```

Using let and const

- Before 2015, using the var keyword was the only way to declare a JavaScript variable.
- The 2015 version of JavaScript (ES6) allows the use of the const keyword to define a variable that cannot be reassigned, and the let keyword to define a variable with restricted scope.

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.1Variables%26DataTypes.html>

JavaScript Local Variable

- A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.

Ex.1

```
<script>  
function  
abc(){  
    var x=10;//local variable  
}  
</script>
```

Ex.2

```
<script>  
if(10<13){  
    var y=20;//JavaScript local  
    variable  
}  
</script>
```

JavaScript Global Variable

- A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable.

```
<script>
    var data=200;//global variable
    function a(){ document.writeln(data);
    }
    function
    b(){ document.writeln(data);
    }
    a();//calling JavaScript function b());
</script>
```

Global Variables in JavaScript

Global Variable

- A JavaScript global variable is declared outside the function or declared with window object. It can be accessed from any function.

Ex.

```
<script>  
var value=50;//global variable function  
  
a(){  
    alert(value);  
}  
  
function  
b(){ alert(value);  
}  
  
</script>
```

Declaring Global Variable in a Function

- To declare JavaScript global variables inside function, you need to use window object.

Ex.

```
function m(){  
    window.value=100;//declaring global variable by window object  
}  
  
function n(){  
    alert(window.value);//accessing global variable from other function  
}
```

Internals of global variable in JavaScript

- When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also.

Ex.

```
var  
value=50;  
function a(){  
alert(window.value);//accessing global variable  
}
```

Datatypes in JavaScript

Data Types

- JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.
 1. Primitive data type
 2. Non-primitive (reference) data type
- JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc.

```
var a=40; // Variable with numeric value
```

```
var b="Rahul"; // Variable with String  
value
```

JavaScript primitive data types

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

The Concept of Data Types

- var x = 15 + "Hello";
- var x = “15” + "Hello";
- var x = “Hello” + 15;
- var x = 50 + 50 + "Hello";
- var x = “Hello” + 50 + 50;

The Concept of Data Types

- **Assigning value to variable:**

```
var x;      // Now x is undefined  
x = 50;    // Now x is a Number  
x = "Joel"; // Now x is a String
```

- **String:**

```
var answer1 = "His "; // Single quote inside double quotes  
var answer2 = "name is 'Joel"'; // Single quotes inside double  
var answer3 = 'name is "Joel"'; quotes  
                           // Double quotes inside single  
                           quotes
```

The Concept of Data Types

- **Boolean:**

```
var x = 5;  
var y = 5;  
var z = 6;  
(x == y) // Returns true  
(x == z) // Returns false
```

- **Null:**

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

The Concept of Data Types

Ex.

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

```
<script>
document.getElementById("demo").innerHTML
= typeof undefined + "<br>" +
typeof null + "<br><br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>
```

JavaScript non-primitive data types

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values

Array

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.1Variables%26DataTypes.html>

Objects

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.1Variables&DataTypes.html>

Operator in JavaScript

Operators

- JavaScript operators are symbols that are used to perform operations on operands.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.2Operators.html>

Arithmetic Operator

- Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	$1 + 1$
-	Subtraction	$1 - 1$
*	Multiplication	$1 * 1$
/	Division	$1 / 1$
%	Modulus	$1 \% 1$
++	Increment	<code>++i</code>
--	Decrement	<code>--i</code>

Comparison Operator

- The JavaScript comparison operator compares the two operands.

Operator	Description	Example
<code>==</code>	Is equal to	<code>a == b</code>
<code>===</code>	Is equal and of same type	<code>a === b</code>
<code>!=</code>	Not equal to	<code>a != b</code>
<code>!==</code>	Not equal and of not of same type	<code>a !== b</code>
<code>></code>	Greater than	<code>a > b</code>
<code>>=</code>	Greater than or equal to	<code>a <= b</code>
<code><</code>	Less than	<code>a < b</code>
<code><=</code>	Less than or equal to	<code>a <= b</code>

Bitwise Operator

- The bitwise operators perform bitwise operations on operands.

Operator	Description	Example
&	Bitwise AND	a & b
	Bitwise OR	a b
^	Bitwise XOR	a ^ b
~	Bitwise NOT	~a
<<	Bitwise Left Shift	a << b
>>	Bitwise Right Shift	a >> b
>>>	Bitwise Right Shift with zero	a >>> b

Logical Operator

Operator	Description	Example
<code>&&</code>	Logical AND	<code>a && b</code>
<code> </code>	Logical OR	<code>a b</code>
<code>!</code>	Logical Not	<code>!a</code>

Assignment Operator

Operator	Description	Example
=	Assign	$a = b$
+=	Add and Assign	$a += b$
-=	Subtract and Assign	$a -= b$
*=	Multiply and Assign	$a *= b$
/=	Divide and Assign	$a /= b$
%=	Modulus and Assign	$a %= b$

Special Operator

Operator	Description	Example
? :	Conditional Operator	a ==b ? x=true : x=false;
,	Comma operator allows multiple expressions to be evaluated as single statement	let a = (1,2,3); a=3
delete	Delete property deletes an entry from the object	delete x
in	in property checks if the given property exist in the object	prop in object
instanctof	checks if the object is an instance of given type	object instanctof object_type
new	creates a new object	var a = new a();
typeof	checks the type of object	typeof 5 // "number"
void	it discards the expressions return value	void(x)

If Statement in JavaScript

Conditional Statements

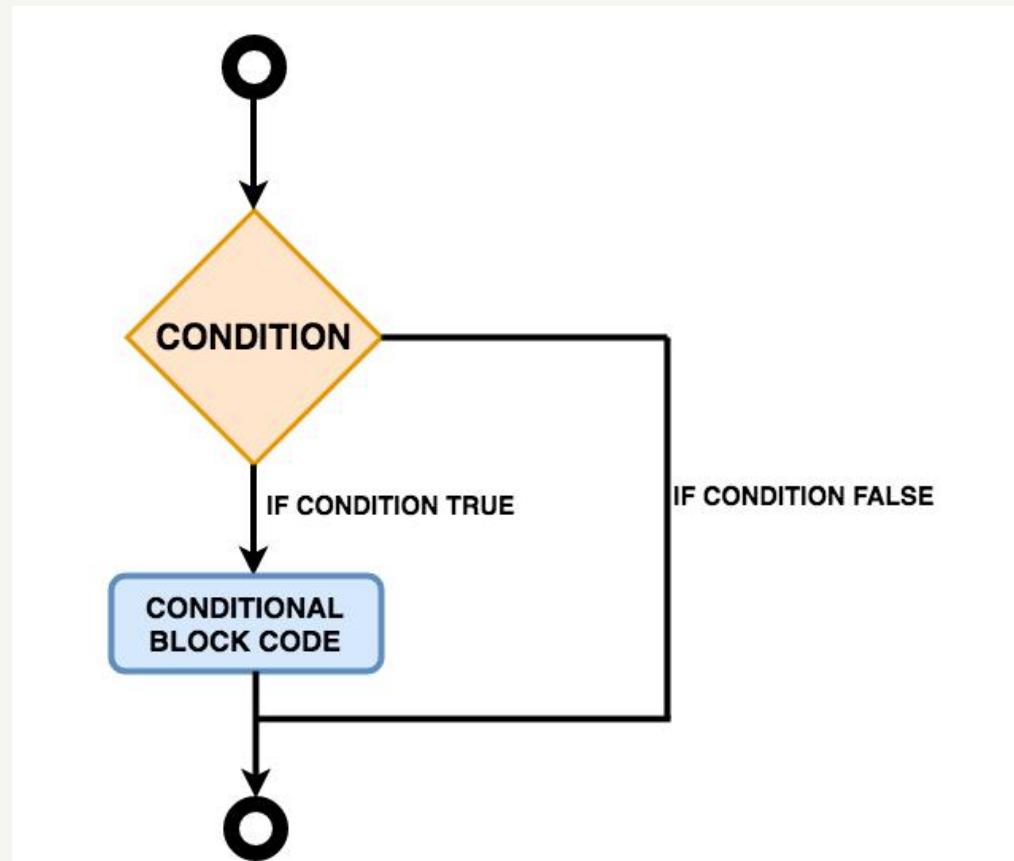
- The JavaScript Conditional statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.
 1. If Statement
 2. If else statement
 3. if else if statement

if statement

- It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

Syntax:

```
if(expression){  
    //content to be  
    evaluated  
}
```



Example of 'if'

Ex.

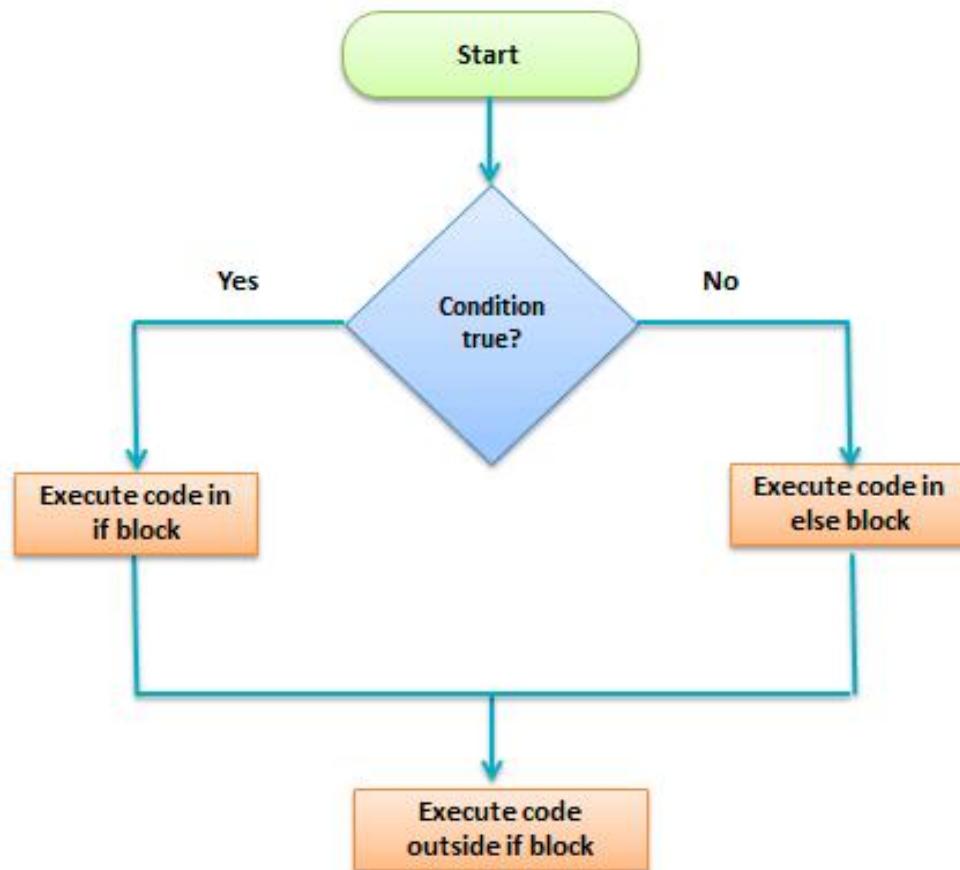
```
<script>  
var a=true;  
if(a){  
document.write("LearnVern is a wonderful place to learn  
Javascript");  
}  
</script>
```

if...else statement

- It evaluates the content whether condition is true or false. The syntax of JavaScript if-else statement is given below.

Syntax:

```
if(expression){  
//evaluated if condition is true  
}  
else{  
//evaluated if condition is  
false  
}
```



Example of 'if...else'

[LINK => https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.5ConditionalStatements.html](https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.5ConditionalStatements.html)

if...else...if statement

- It evaluates the content only if expression is true from several expressions.

Syntax:

```
if(expression1){  
    //content to be evaluated if expression1 is true  
}  
else if(expression2){  
    //content to be evaluated if expression2 is true  
}  
else{  
    //content to be evaluated if no expression is true  
}
```

Example of 'if...else...if'

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.5ConditionalStatements.html>

Switch Statement in JavaScript

Switch Statement

- The JavaScript switch statement is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page.
- It is convenient than if..else..if because it can be used with numbers, characters etc.

Switch Statement

Syntax:

```
switch(expression){ case
```

```
value1:
```

```
    //code to be executed;
```

```
    break;
```

```
case value2:
```

```
    //code to be executed;
```

```
    break;
```

```
.....
```

```
default:
```

```
    //code to be executed if above values are not matched;
```

```
}
```

The break and default Keyword

- When JavaScript reaches a break keyword, it breaks out of the switch block.
- This will stop the execution inside the switch block.
- It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.
- The default keyword specifies the code to run if there is no case match:

[LINK => https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.6SwitchCase.html](https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.6SwitchCase.html)

Strict Comparison

- Switch cases use strict comparison (`==`).
- The values must be of the same type to match.
- A strict comparison can only be true if the operands are of the same type.
- In the example explained in next slide there will be no match for `x`:

Ex. Strict Comparison

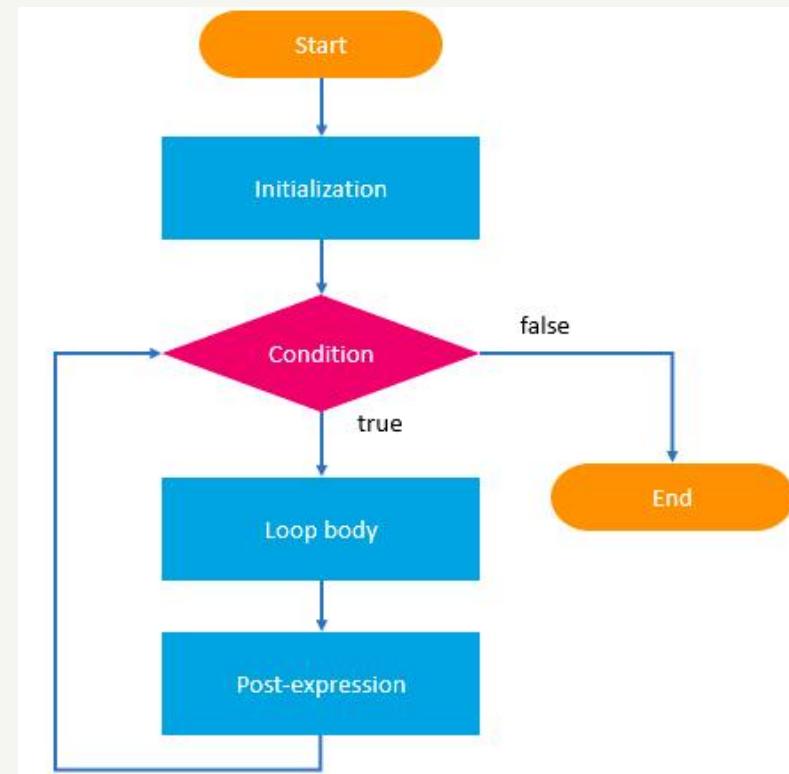
```
var x = "0";
switch (x)
{ case 0:
    text =
    "Off";
    break;
case 1:
    text = "On";
    break;
default:
    text = "No value found"; }
```

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/conditional-statements>

Loops in JavaScript

Loops

- The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
- There are four types of loops in JavaScript.
 1. for loop
 2. while loop
 3. do-while loop
 4. for-in loop
 5. for-of loop



for Loop

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

Syntax:

```
for (initialization; condition; increment)
{
    code to be executed
}
```

for Loop Example

```
<script>
for (i=1; i<=5; i++)
{
document.write(i +
"<br/>")
}
</script>
```

while Loop

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

Syntax:

```
while (condition)
{
    code to be executed
}
```

while Loop Example

```
<script>
var i=11;
while (i<=15)
{
document.write(i+
"<br/>"); i++;
}
</script>
```

do while Loop

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

Syntax:

```
do{  
    //code to be executed  
}while (condition);
```

do...while Loop Example

```
<script>
var i=21;
do{
document.write(i +
"<br/>"); i++;
}while (i<=25);
</script>
```

for...in Loop

- for in loops through the properties of an object

Syntax:

```
for (var in object) {  
    code block to be executed  
}
```

for...in Loop Example

```
var person = {fname:"John", lname:"Doe",  
age:25};
```

```
var text = "";  
var x;  
for (x in person) {  
    text += person[x] + " ";  
}
```

for...of Loop

- for of loops through the values of an iterable object

Syntax:

```
for (variable of iterable)
  { code block to be
    executed
  }
```

for...of Loop Example

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/looping-statements>

Function in JavaScript

What is Function

- JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.
- There are mainly two **advantages** of JavaScript functions.
 1. Code reusability: We can call a function several times so it save coding.
 2. Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

Syntax and Example

```
function functionName([arg1, arg2, ...argN]){
    //code to be executed
}
```

Note: JavaScript Functions can have 0 or more arguments.

JavaScript Function Object

- In JavaScript, the purpose of Function constructor is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax:

```
new Function ([arg1[, arg2[, ...argn]]],) funcBody)
```

Parameter:

arg1, arg2, , argn - It represents the argument used by function.

funcBody - It represents the function definition.

JavaScript Function Methods

Method	Description
call()	It is used to call a function contains this value and an argument list.
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
toString()	It returns the result in a form of a string.

call()

- The JavaScript Function call() method is used to call a function contains this value and an argument provided individually.

Syntax:

```
function.call(thisArg, arg1,arg2,. . .,argn)
```

Parameter:

thisArg - It is optional. The this value is given for the call to function.

arg1,arg2,...,argn - It is optional. It represents the arguments for the function.

apply()

- The JavaScript Function apply() method is used to call a function contains this value and an argument contains elements of an array. Unlike call() method, it contains the single array of arguments.

Syntax:

```
function.apply(thisArg, [array])
```

Parameter:

thisArg - It is optional. The this value is given for the call to a function.

array - It is optional. It is an array-like object.

bind()

- The JavaScript Function bind() method is used to create a new function. When a function is called, it has its own this keyword set to the provided value, with a given sequence of arguments.

Syntax:

```
function.bind(thisArg [, arg1[, arg2[, ...]]])
```

Parameter:

thisArg - The this value passed to the target function.

arg1,arg2,...,argn - It represents the arguments for the function.

toString()

- The JavaScript Function `toString()` method returns a string. Here, string represents the source code of the function.

Syntax:

```
function.toString()
```

toString() Example 2

<https://github.com/Brijesh1990/tops-website-development/blob/master/module4-javascript-basic-advanced/JavaScript/3.9Function.html>

Objects in JavaScript

What is object in Javascript

- A JavaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.
- JavaScript is an object-based language. Everything is an object in JavaScript.
- JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

What is object in Javascript

- There are 3 ways to create objects.
 1. By object literal
 2. By creating instance of Object directly (using new keyword)
 3. By using an object constructor (using new keyword)

1. Object by object literal

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

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

Object by object literal Example

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+
"+emp.salary);
</script>
```

2. By creating instance of Object

Syntax:

```
var objectname=new Object();
```

Here, new keyword is used to create object.

By creating instance of Object Example

```
<script>  
var emp=new Object();  
emp.id=101;  
emp.name="Ravi Malik";  
emp.salary=50000;  
document.write(emp.id+" "+emp.name+"  
"+emp.salary);  
</script>
```

3. By using an Object constructor

- you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.
- The **this** keyword refers to the current object.

Object constructor Example

[https://github.com/Brijesh1990/tops-website-
development/tree/master/module4-javascript-basic-advanced/JavaScript](https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript)

How to define method in JavaScript object

- We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript>

How to define method in javascript object

```
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+
"+e.salary); e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+
"+e.salary);
</script>
```

JavaScript Object

Methods	Description
Object.assign()	This method is used to copy enumerable and own properties from a source object to a target object
Object.create()	This method is used to create a new object with the specified prototype object and properties.
Object.freeze()	This method prevents existing properties from being removed.
Object.is()	This method determines whether two values are the same value.
Object.isExtensible()	This method determines if an object is extensible
Object.isFrozen()	This method determines if an object was frozen.
Object.isSealed()	This method determines if an object is sealed.
Object.seal()	This method prevents new properties from being added and marks all existing properties as non-configurable.
Object.values()	This method returns an array of values.

Browser Object Model

Browser Object Model

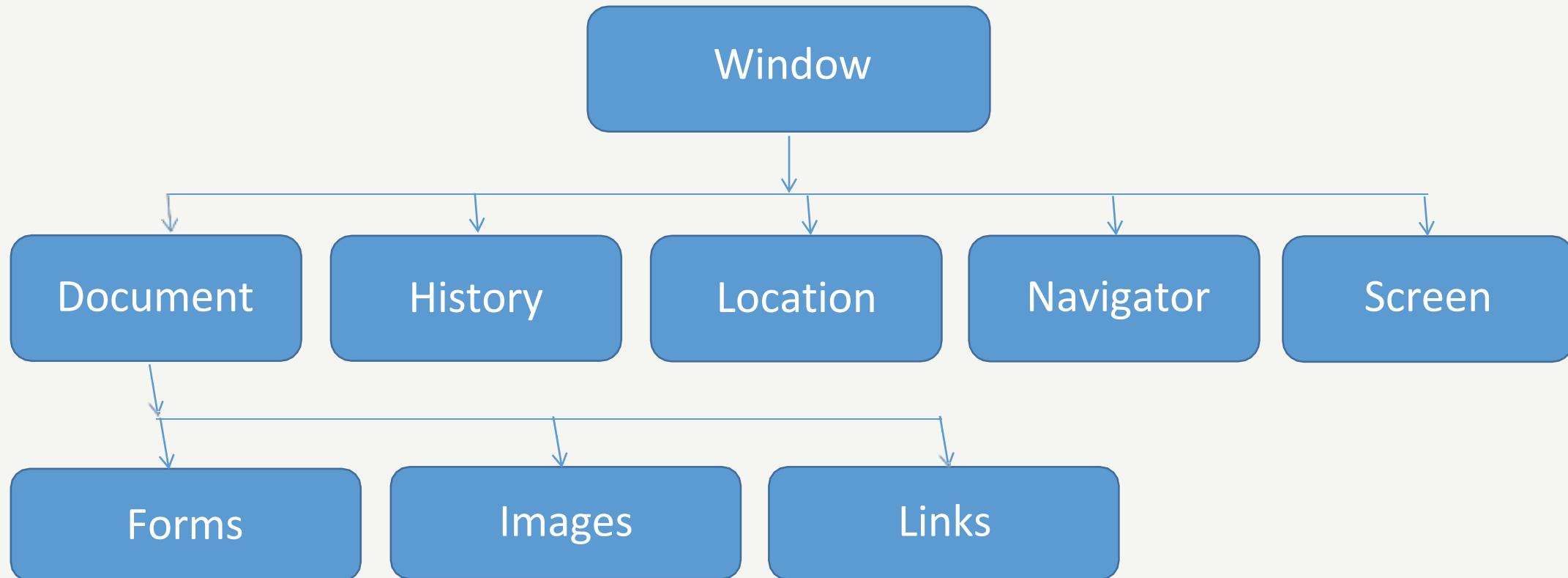
- The Browser Object Model (BOM) is used to interact with the browser.
- The default object of browser is window means you can call all the functions of window by specifying window or directly.

For example:

```
window.alert("hello  
LearnVern");
```

Above is same as : alert("hello LearnVern");

Browser Object Model



Window Object

Window Object

- The window object represents a window in browser. An object of window is created automatically by the browser.
- Window is the object of browser, it is not the object of javascript. The javascript objects are string, array, date etc.
- There are several Window Methods Available in through Window Object

Methods of Window Object

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions etc.

Ex.

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/javascript-boxes>

History Object

History Object in Javascript

- The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

Syntax:

window.history
or
history

Property

- There is only one Property in History Object
history.length;
- The length property returns the number of URLs in the history list of the current browser window.
- The property returns at least 1, because the list includes the currently loaded page.

Example:

```
var x = history.length;
```

History back() Method

- The back() method loads the previous URL in the history list. This is the same as clicking the "Back button" in your browser.

Example:

```
<button onclick="goBack()">Go Back</button>
```

```
<script>
function goBack()
{ window.history.back()
;
}
```

History forward() Method

- The forward() method loads the next URL in the history list. This is the same as clicking the "Forward button" in your browser, or history.go(1).

Example:

```
<button onclick="goForward()">Go Forward</button>
```

```
<script>
function goForward()
{ window.history.forward();
}
</script>
```

History go() Method

- The go() method loads a specific URL from the history list.

Example:

```
<button onclick="goBack()">Go Back 2 Pages</button>
```

```
<script>
function goBack() {
    window.history.go(-2);
}
</script>
```

Navigator Object

Navigator Object

- The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

Syntax:

window.navigator
or
navigator

Navigator Object

Property	Description
appName	returns the name
appVersion	returns the version
appCodeName	returns the code name
cookieEnabled	returns true if cookie is enabled otherwise false
userAgent	returns the user agent
language	returns the language. It is supported in Netscape and Firefox only.
userLanguage	returns the user language. It is supported in IE only.
plugins	returns the plugins. It is supported in Netscape and Firefox only.
systemLanguage	returns the system language. It is supported in IE only.
mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.

Navigator Object Properties

Property	Description
platform	returns the platform e.g. Win32.
online	returns true if browser is online otherwise false.

Screen Object

Screen Object

- The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

Syntax:

window.screen
or
screen

Screen Object Property

Method	Description
width	returns the width of the screen
height	returns the height of the screen
availWidth	returns the available width
availHeight	returns the available height
colorDepth	returns the color depth
pixelDepth	returns the pixel depth.

Document Object Model

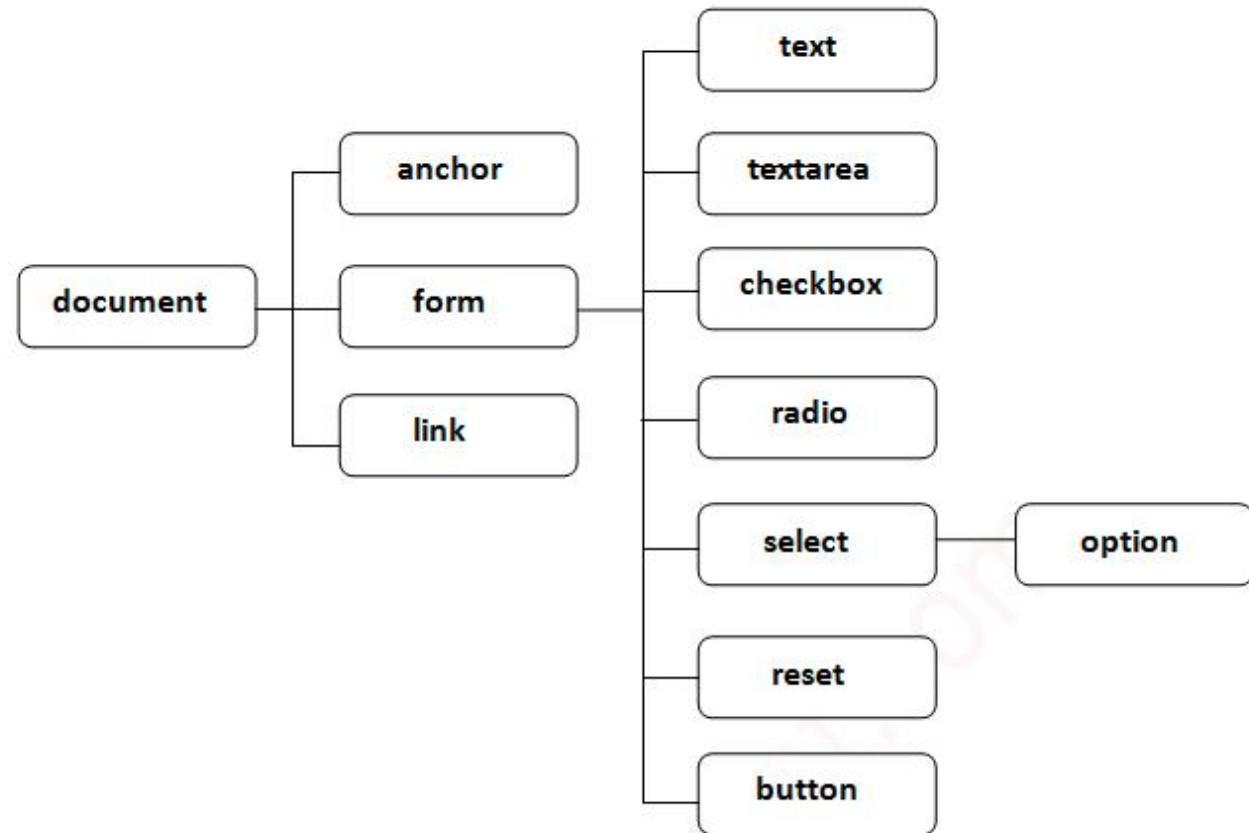
Document Object Model

- The document object represents the whole html document.
- When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document.
- It has properties and methods. By the help of document object, we can add dynamic content to our web page.

Syntax:

window.document
or
document

Properties of document object



Methods of Document Object

Method	Description
write("string")	writes the given string on the document.
writeln("string")	writes the given string on the document with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Document Object Methods

document.getElementById()

- The document.getElementById() method returns the element of specified id.
- In the previous page, we have used document.form1.name.value to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Ex.

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/javascript-output-method>

Form Validation Using JavaScript

Form Validation

- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
- JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.
- Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Form Validation Example => <https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/form-validation>

Email Validation using JavaScript

Email Validation

- We can validate the email by the help of JavaScript.
- There are many criteria that need to be follow to validate the email id such as:
 1. email id must contain the @ and . character
 2. There must be at least one character before and after the @.
 3. There must be at least two characters after . (dot).

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/form-validation>

What is Event

JavaScript Events

- The change in the state of an object is known as an Event.
- In html, there are various events which represents that some activity is performed by the user or by the browser.
- When javascript code is included in HTML, js react over these events and allow the execution.
- This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

Mouse Events

Event Performed	Event Handler	Description
Click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard Events

Event Performed	Event Handler	Description
keydown	onkeydown	When the user press and then release the key
keyup	onkeyup	When the user press and then release the key

Form Events

Event Performed	Event Handler	Description
Focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
Blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document Events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

addEventListener()

addEventListener()

- The addEventListener() method is used to attach an event handler to a particular element. It does not override the existing event handlers.
- Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred.

addEventListener()

- Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.
- The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

addEventListener()

Syntax: element.addEventListener(event, function, useCapture);

- **event:** It is a required parameter. It can be defined as a string that specifies the event's name.
- **function:** It is also a required parameter. It is a JavaScript function which responds to the event occur.
- **useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are true and false. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is false.

Event Bubbling or Event Capturing

- **Bubbling:** in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.
- **Capturing:** in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

```
addEventListener(event, function, useCapture);
```

Example

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript>

onclick event in JavaScript

onclick event in Javascript

- The onclick event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked.
- This event can be used for validating a form, warning messages and many more.
- Using JavaScript, this event can be dynamically added to any element.
- It supports all HTML elements except <html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>,
, <meta>, and <param>. It means we cannot apply the onclick event on the given tags.

onclick event in JavaScript

- **In HTML:**

```
<element onclick = "fun()">
```

- **In JavaScript:**

```
object.onclick = function() { myScript };
```

- **In JavaScript by using the addEventListener()
method:**

```
object.addEventListener("click", myScript);
```

Using onclick attribute in HTML

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript>

DoubleClick event in JavaScript

dblclick event

- The dblclick event generates an event on double click the element.
- The event fires when an element is clicked twice in a very short span of time.
- We can also use the JavaScript's addEventListener() method to fire the double click event.

dblclick event

- **In HTML:**

```
<element ondblclick = "fun()">
```

- **In JavaScript:**

```
object.ondblclick = function() { myScript };
```

- **In JavaScript by using the addEventListener()
method:**

```
object.addEventListener("dblclick", myScript);
```

Using ondblclick attribute in HTML

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript>

onload event in JavaScript

onload in javascript

- In JavaScript, this event can apply to launch a particular function when the page is fully displayed. It can also be used to verify the type and version of the visitor's browser. We can check what cookies a page uses by using the onload attribute.
- In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

onload in javascript

- In HTML, the `onload` attribute is generally used with the `<body>` element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with `<body>` tag, as it can be used with other HTML elements.
- The difference between the `document.onload` and `window.onload` is: `document.onload` triggers before the loading of images and other external content. It is fired before the `window.onload`. While the `window.onload` triggers when the entire page loads, including CSS files, script files, images, etc.

onload in javascript

Syntax:

```
window.onload =  
fun()
```

LINK

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/javascript-events>

onresize event in JavaScript

onresize event in javascript

- The onresize event in JavaScript generally occurs when the window has been resized. To get the size of the window, we can use the JavaScript's `window.outerWidth` and `window.outerHeight` events.
- We can also use the JavaScript's properties such as `innerWidth`, `innerHeight`, `clientWidth`, `ClientHeight`, `offsetWidth`, `offsetHeight` to get the size of an element.

onresize event in javascript

- In HTML, we can use the onresize attribute and assign a JavaScript function to it.
- We can also use the JavaScript's addEventListener() method and pass a resize event to it for greater flexibility.

Syntax

- **In HTML:**

```
<element onresize = "fun()">
```

- **In JavaScript:**

```
object.onresize = function() { myScript };
```

- **In JavaScript by using the addEventListener()
method:**

```
object.addEventListener("resize", myScript);
```

LINK

<https://github.com/Brijesh1990/tops-website-development/tree/master/module4-javascript-basic-advanced/JavaScript/javascript-events>

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
 - Bootstrap Utilities
- 1) Bootstrap Typography
- 2) Bootstrap Tables
- 3) Bootstrap Lists
- 4) Bootstrap List Groups
- 5) Bootstrap Forms
- 6) Bootstrap Custom Forms
- 7) Bootstrap Input Groups
- 8) Bootstrap Buttons
- 9) 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/>

[https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/online CDN](https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/online_CDN)

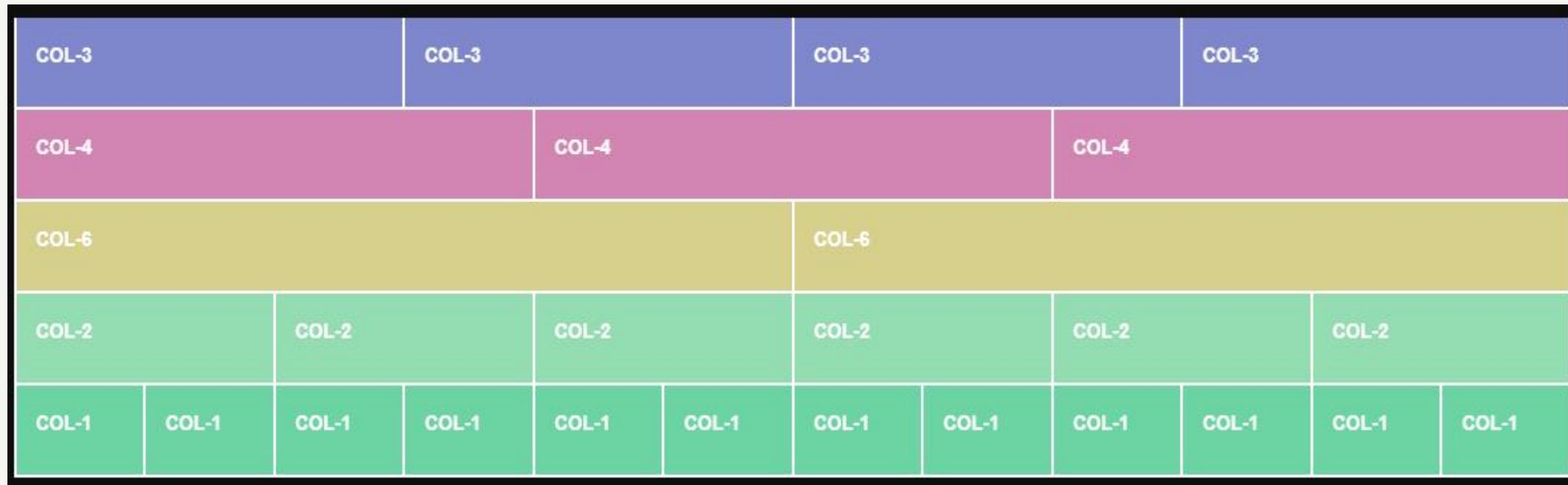
[https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/offline CDN](https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/offline_CDN)

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)

Example

[https://github.com/Brijesh1990/tops-website-development/tree/master/module6-
bootstrap-basic-advanced/bootstrap-grid](https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/bootstrap-grid)

Bootstrap Fixed Layout

Bootstrap Fluid Layout

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.

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

Bootstrap Tables

Bootstrap Tables

- We can create different types of Bootstrap tables by using different classes to style them.

Example

[**https://github.com/Brijesh1990/tops-website-development/blob/master/module6-bootstrap-basic-advanced/5.6.0BootstrapTable.html**](https://github.com/Brijesh1990/tops-website-development/blob/master/module6-bootstrap-basic-advanced/5.6.0BootstrapTable.html)

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

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

Bootstrap 4 Form

<https://github.com/Brijesh1990/tops-website-development/tree/master/module6-bootstrap-basic-advanced/bootstrap-modal>

Bootstrap Input Groups

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.

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.

Checkboxes and Radios

- Place any checkbox or radio option within an input group's addon instead of text.

Multiple inputs

- While multiple <input>s are supported visually, validation styles are only available for input groups with a single <input>.

Multiple addons

- Multiple add-ons are supported and can be mixed with checkbox and radio input versions.

Custom forms

- Input groups include support for custom selects and custom file inputs. Browser default versions of these are not supported.

Bootstrap Buttons

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

Bootstrap Images

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

Bootstrap Navbar

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 –
 1. Start with a basic unordered list with the base class of **.nav**
 2. Add class **.nav-tabs**.

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.

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 to do with the button, and the 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>

Navbar Brands

- Adding images to the .navbar-brand will likely always require custom styles or utilities to properly size.

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.

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).

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.

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.

Bootstrap Pagination

Pagination

- Pagination, an unordered list is handled by Bootstrap like a lot of other interface elements.

Pagination with Icons

- We can use icons in place of Next and Previous text in the pagination. Be sure to provide proper screen reader support with aria attributes and the .sr-only utility

Disabled and Active State

- Pagination links are customizable for different circumstances. Use `.disabled` for links that appear un-clickable and `.active` to indicate the current page.

Sizing

- Add .pagination-lg or .pagination-sm for additional sizes.

Alignment

- [Change the alignment of pagination components with flexbox utilities.](#)

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.

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

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/components/>

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.

Bootstrap Dropdown Divider

- The **class .divider** is used to separate links inside the dropdown menu

Bootstrap Dropdown Header

- The **class .dropdown-header** is used to add headers inside the dropdown menu.

Bootstrap Dropdown Disable an item

- Use the **class .disabled** to disable an item in the dropdown menu.

Bootstrap Dropup

- If you want to open the menu upwards instead of downwards, you need to change the element with class="dropdown" instead of class="dropdown":

Split Button Dropdowns

- It is used to show dropdowns as split buttons. Here we use all contextual classes.

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 `` 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

Button

- Badges can be used as part of links or buttons to provide a counter.

Bootstrap Layout: Jumbotron

Jumbotron

- A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.

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.

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).

Link Color

- Use the .alert-link utility class to quickly provide matching colored links within any alert.

Additional content

- Alerts can also contain additional HTML elements like headings, paragraphs and dividers.

Dismissing

- Using the alert JavaScript plugin, it's possible to dismiss any alert inline.

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.

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.

Labels

- Add labels to your progress bars by placing text within the .progress-bar.

Height

- We only set a height value on the .progress, so if you change that value the inner .progress-bar will automatically resize accordingly.

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.

Striped

- Add `.progress-bar-striped` to any `.progress-bar` to apply a stripe via CSS gradient over the progress bar's background color.

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.

Bootstrap Layout: List Group

List Group

- List groups are a flexible and powerful component for displaying a series of content.

Active Items

- Add `.active` to a `.list-group-item` to indicate the current active selection.

Disabled Item

- Add `.disabled` to a `.list-group-item` to make it *appear* disabled.

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`.

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).

Numbered

- Add the .list-group-numbered modifier class (and optionally use an `` element) to opt into numbered list group items.

Horizontal

- Add .list-group-horizontal to change the layout of list group items from vertical to horizontal across all breakpoints.

Contextual Class

- Use contextual classes to style list items with a stateful background and color.

Badges

- Add badges to any list group item to show unread counts, activity, and more with the help of some utilities.

Bootstrap Layout: Panels

Panels

- A panel in bootstrap is a bordered box with some padding around its content.

Panel Heading

- The .panel-heading class adds a heading to the panel.

Panel Footer

- The .panel-footer class adds a footer to the panel.

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

Panel with contextual classes

- To color the panel, use contextual classes (.panel-default, .panel-primary, .panel-success, .panel-info, .panel-warning, or .panel-danger)

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

Static Backdrop

- When backdrop is set to static, the modal will not close when clicking outside it. Click the button below to try it.

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`.

Vertically Centered

- Add .modal-dialog-centered to .modal-dialog to vertically center the modal.

Tooltips

- [Tooltips](#) can be placed within modals as needed. When modals are closed, any tooltip within are also automatically dismissed.

Using the Grid

- Utilize the Bootstrap grid system within a modal by nesting `.container-fluid` within the `.modal-body`. Then, use the normal grid system classes as you would anywhere else.

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.

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.

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.

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.

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.

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.

Bootstrap – Login Page

[https://github.com/Brijesh1990/tops-website-
development/tree/master/module6- bootstrap-basic-advanced/bootstrap-
modal](https://github.com/Brijesh1990/tops-website-development/tree/master/module6- bootstrap-basic-advanced/bootstrap-modal)

Bootstrap Project

Please find all GitHub links for projects

[https://github.com/Brijesh1990/tops-website-](https://github.com/Brijesh1990/tops-website-development/tree/master/psdtohtml)

<https://github.com/Brijesh1990/tops-website-development/tree/master/Live-projectexamples>

<https://github.com/Brijesh1990/tops-website-development/tree/master/psd>

Module-8

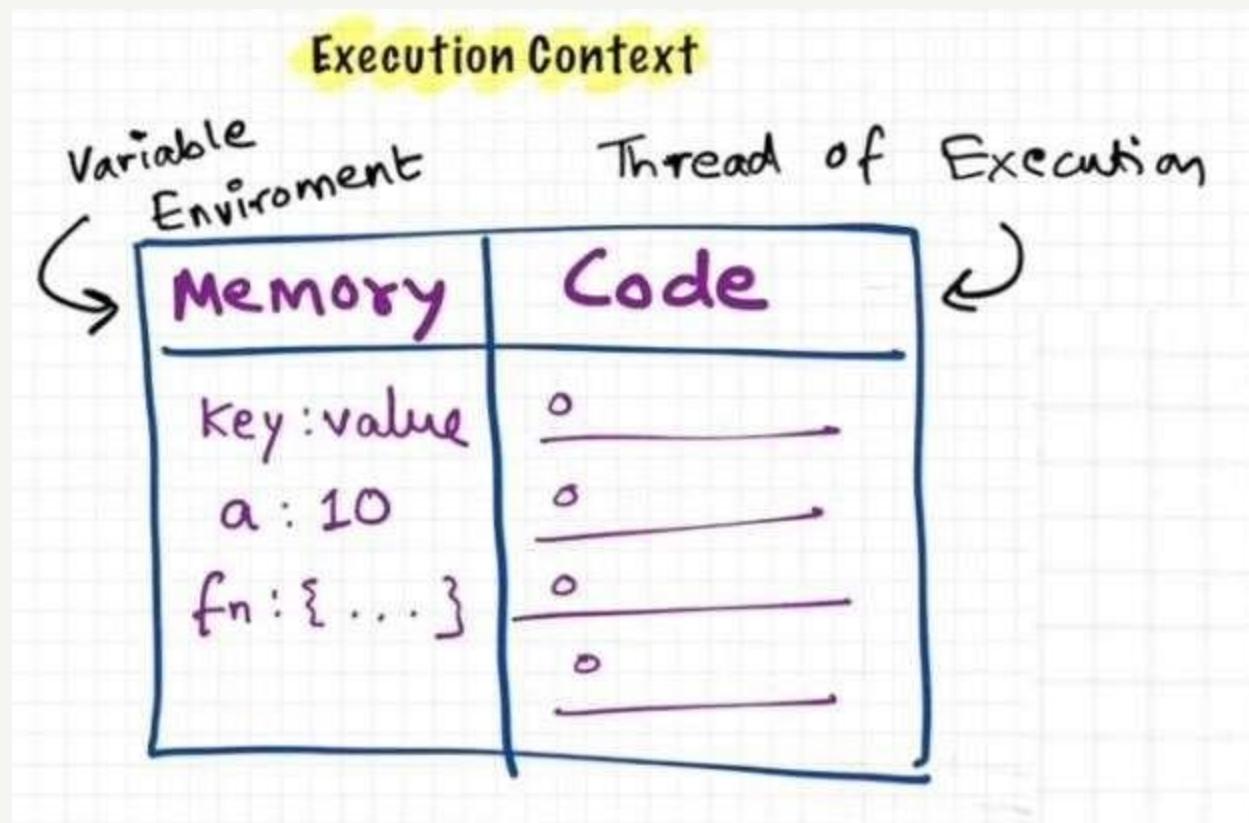
JavaScript Essentials And Advanced

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.
- 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.

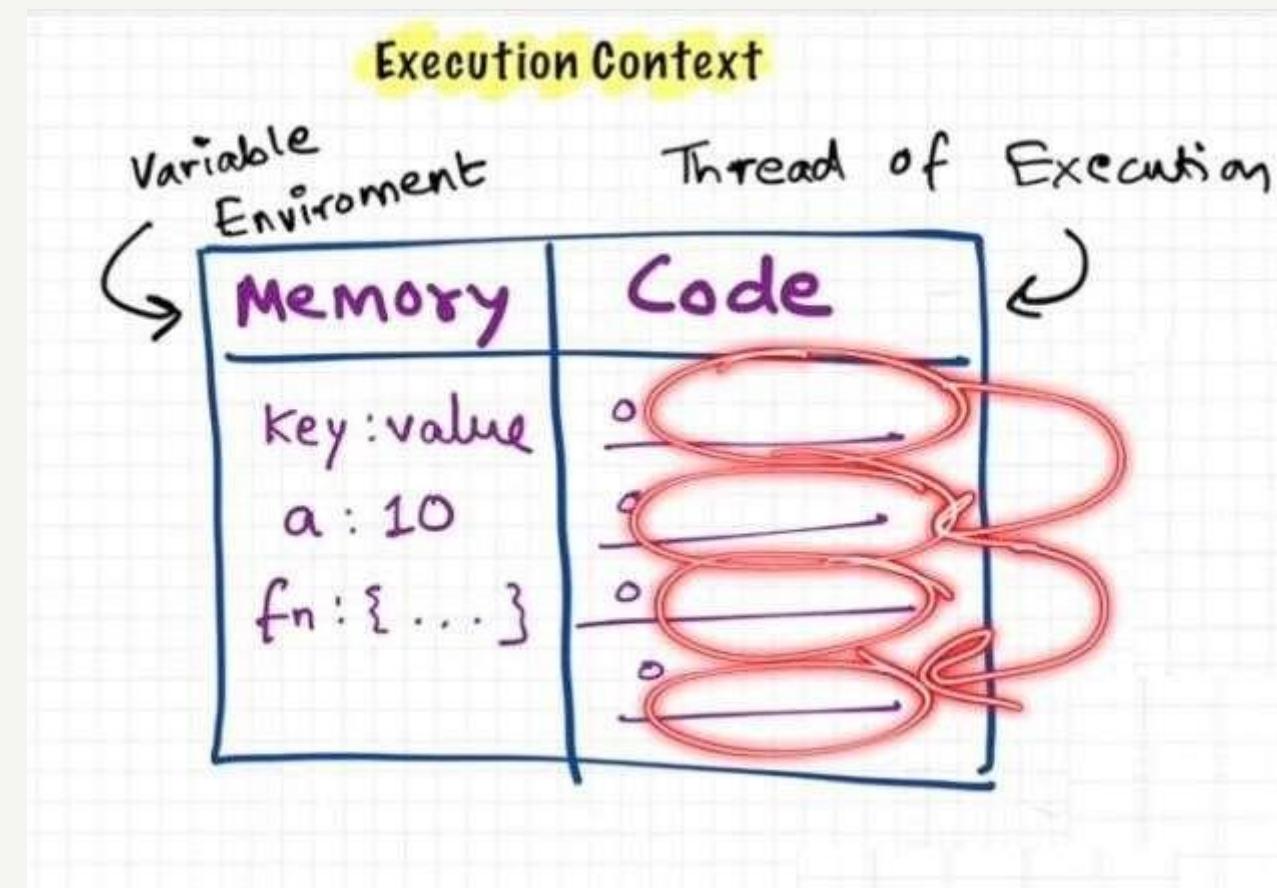
An Introduction to JavaScript

Everything in JavaScript Happens inside an execution context

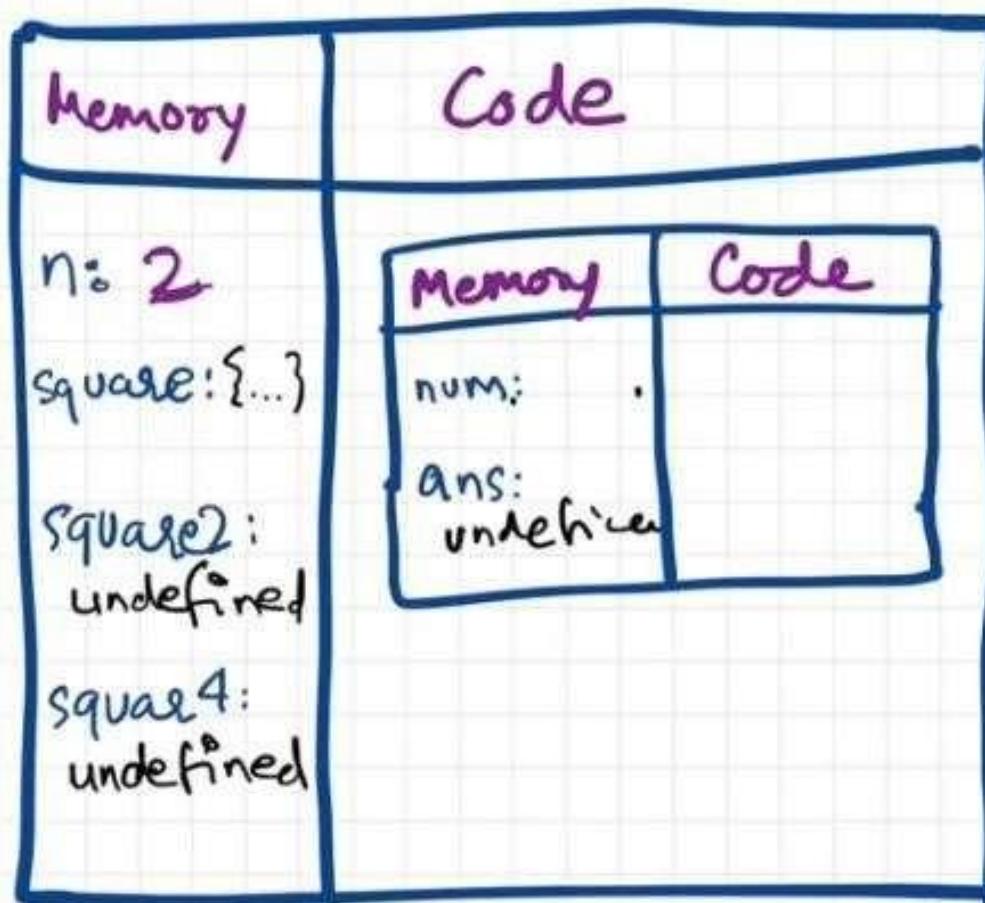


An Introduction to JavaScript

- JavaScript is a Synchronous 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 Manufacturer's 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.

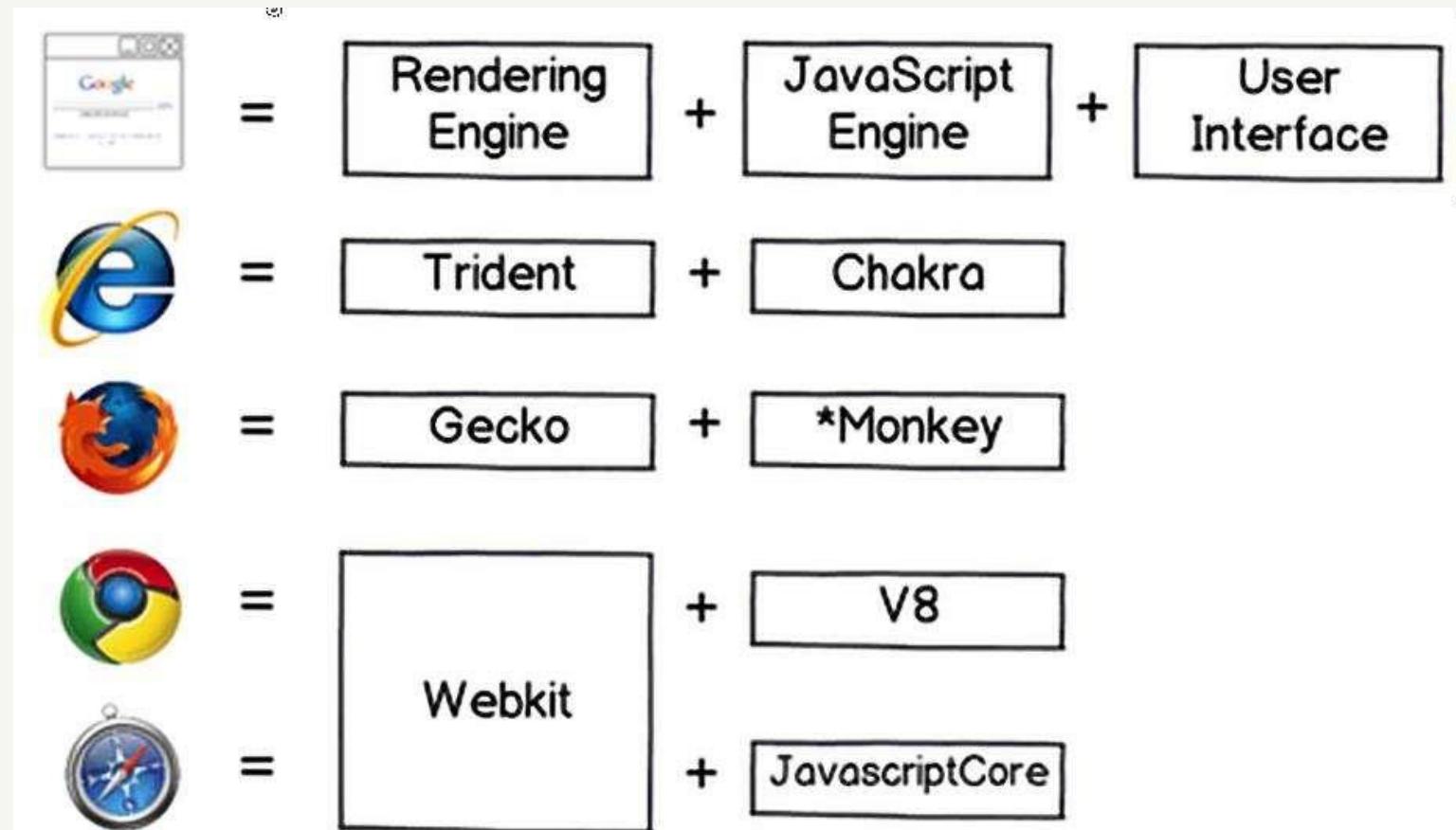
THE EVOLUTION OF JAVASCRIPT

- 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.
- **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.
- **Browsers Improved Processing Speeds** – Thanks to improvements in web browsers throughout the years, JavaScript is being processed even faster than ever. This makes it the most practical choice for web developers, because speed is everything.
- **Frameworks like jquery Reduce Learning Times** – Frameworks like jquery make the learning curve much easier for web developers to master JavaScript. Plus, these frameworks make it easier to create highly professional and interactive web applications efficiently.

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.
- 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:
- 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.

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.

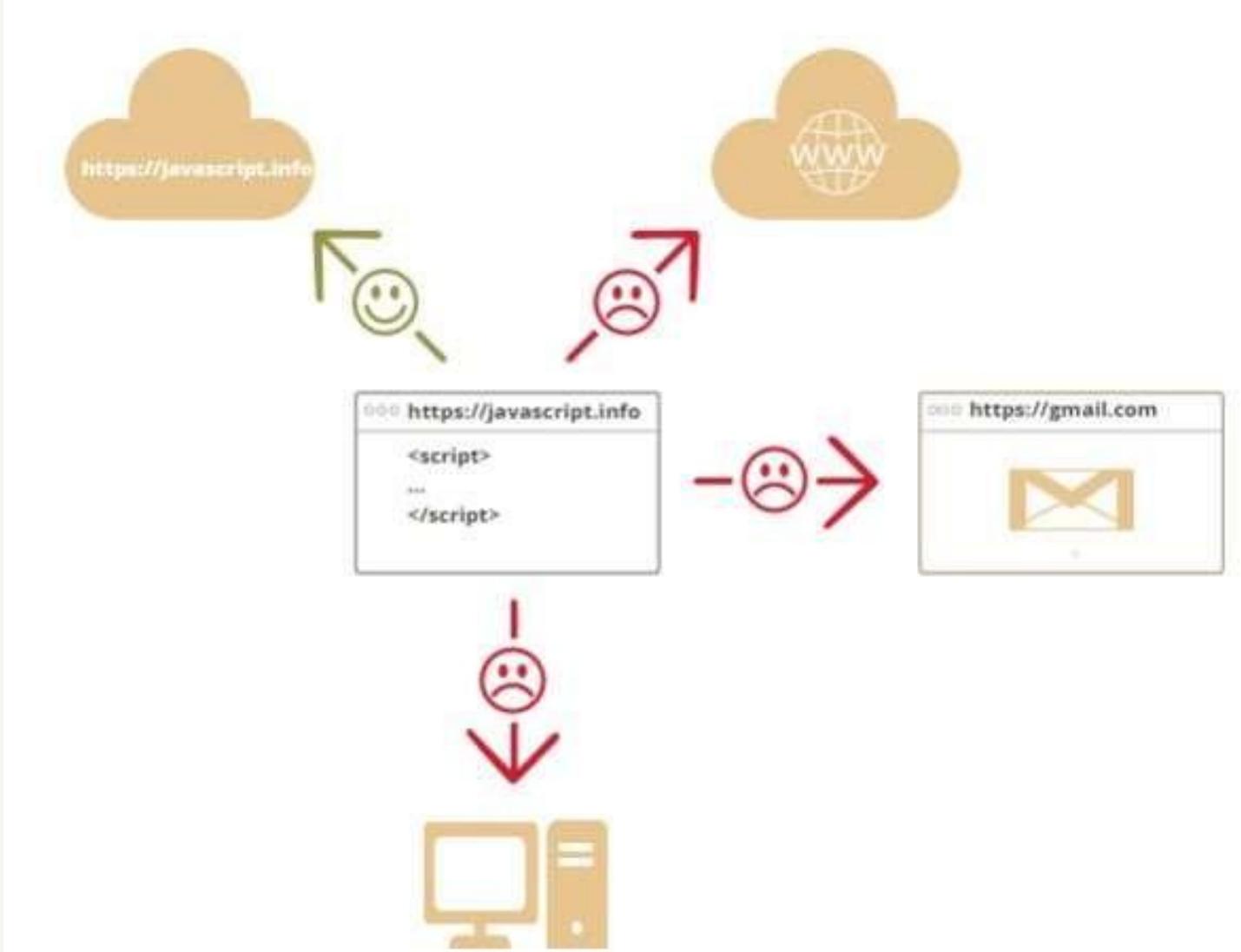
What CAN'T in-browser JavaScript do?

- Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the “Same Origin Policy”. To work around that, both pages must agree for data exchange and contain a special JavaScript code that handles it. We’ll cover that in the tutorial.

This limitation is, again, for the user’s safety. A page from <http://anysite.com> which a user has opened must not be able to access another browser tab with the URL <http://gmail.com> and steal information from there.

- JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that’s a safety limitation.



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.
- 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”.
- 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.
 - [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.

PROS

- Speed
- Simplicity
- Popularity
- Interoperability
- Server Load
- Rich Interfaces
- Extended Functionality
- Versatility
- Less Overhead

Pros and Cons of



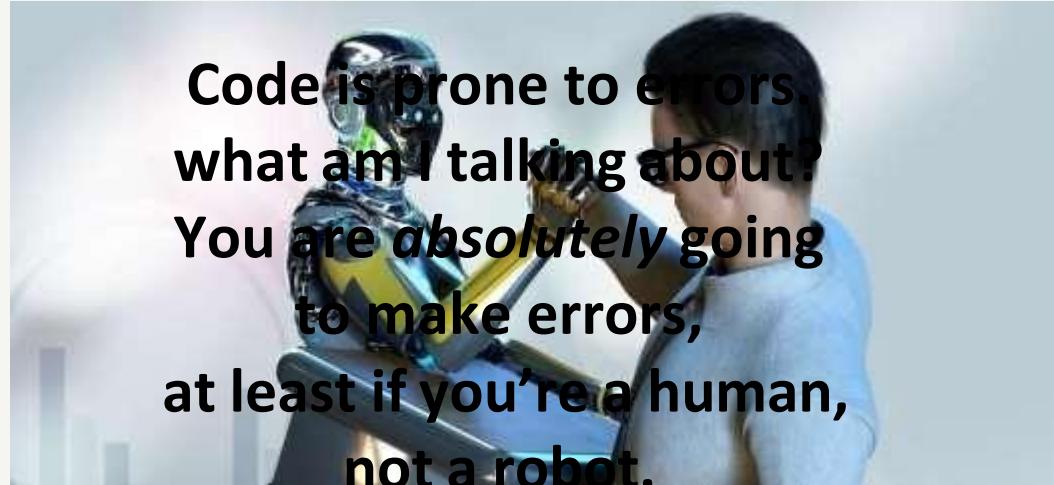
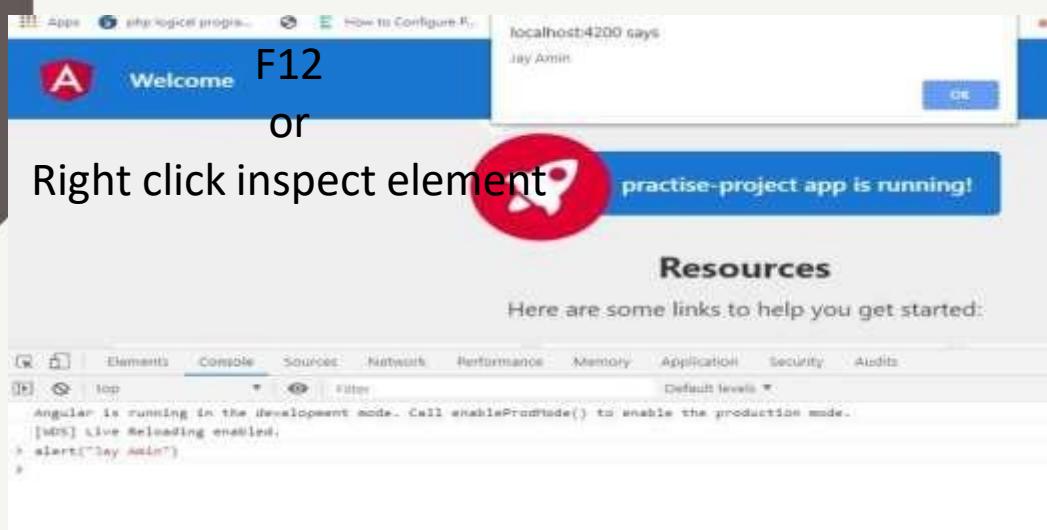
CONS

- Client-side Security
- Browser Support
- Lack of Debugging Facility
- Single Inheritance
- Sluggish Bitwise Function
- Rendering Stopped

JavaScript Uses in Web Designing



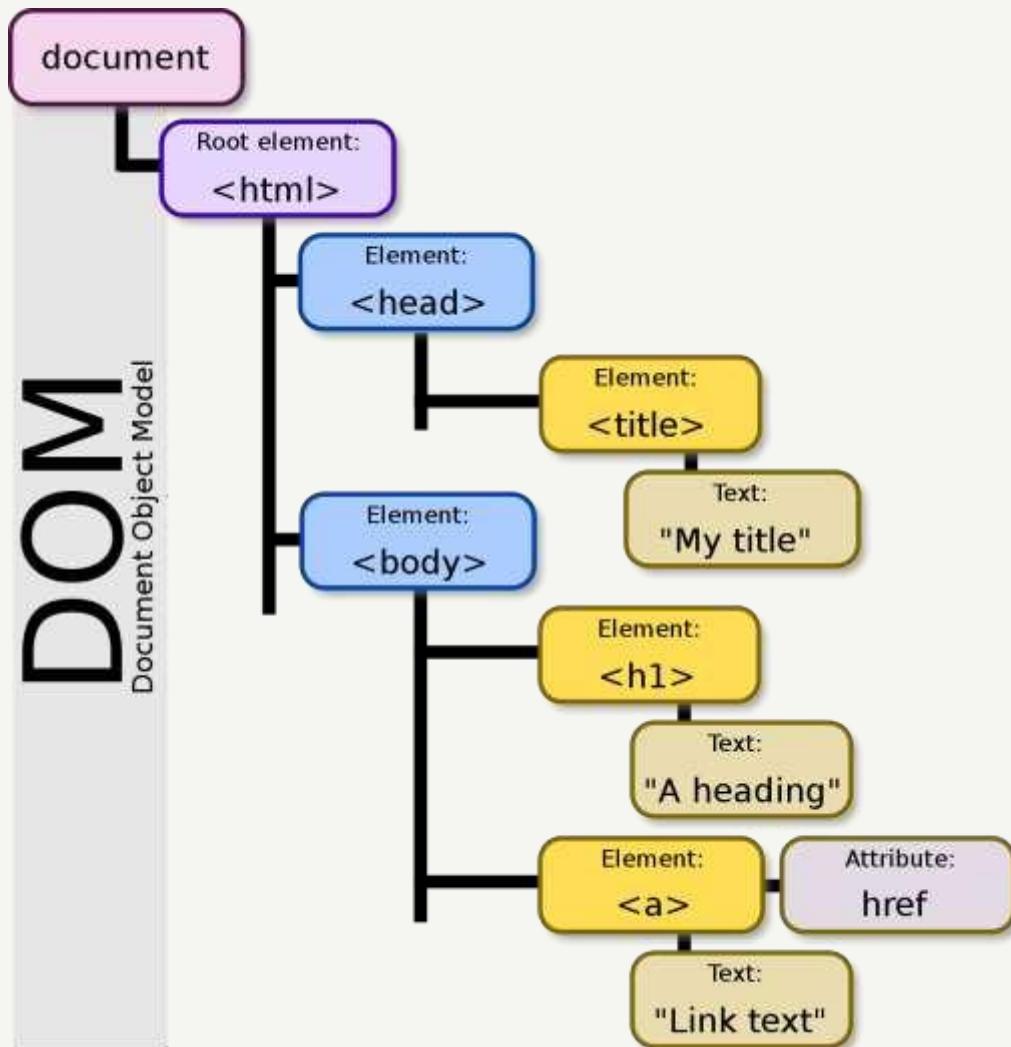
Developer console



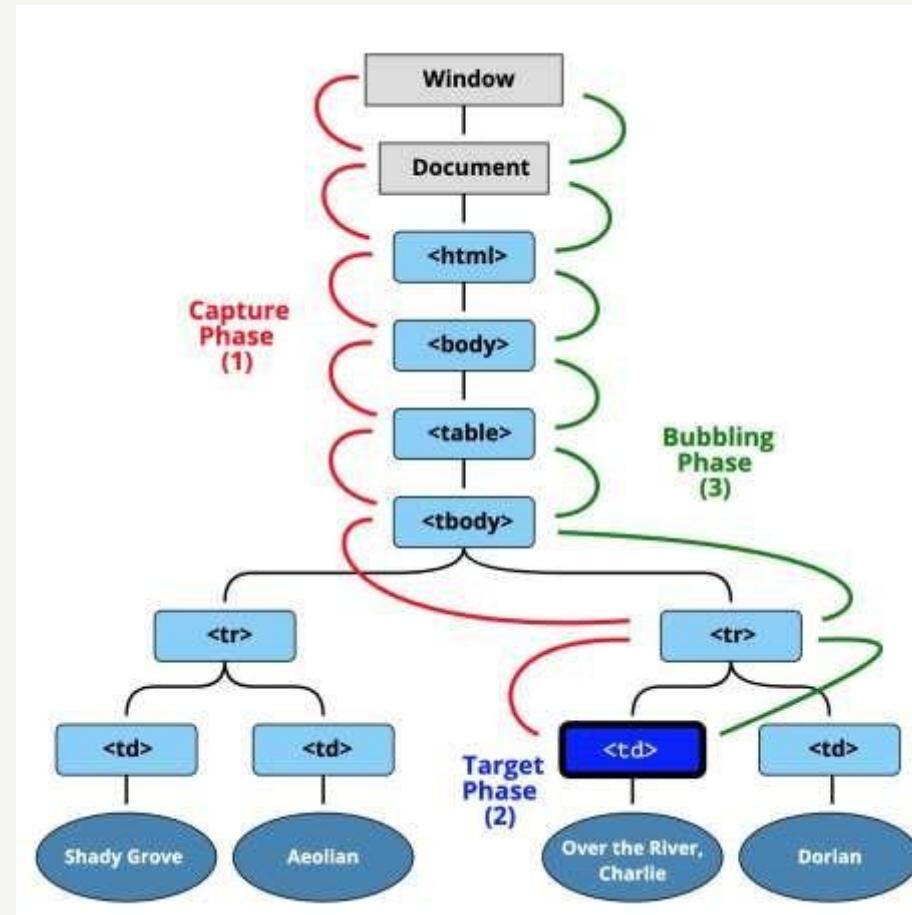
But in the browser, users don't see errors by default. So, if something goes wrong in the script, we won't see what's broken and can't fix it.

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.

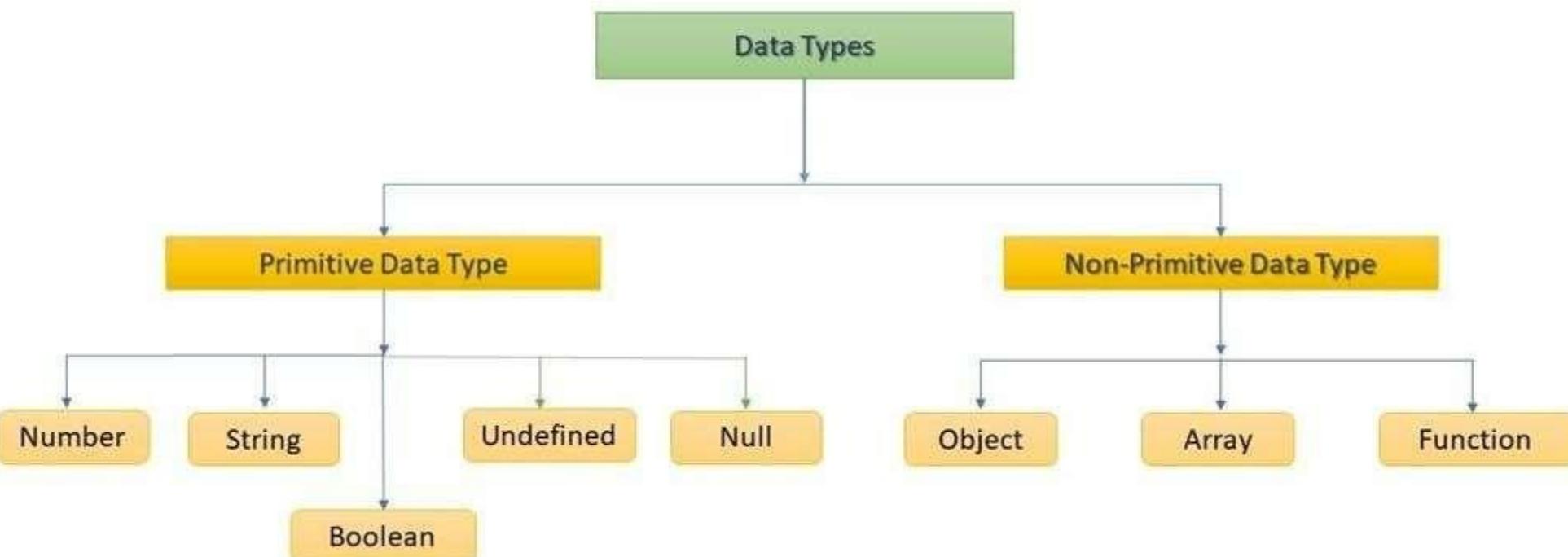
Developer tools are potent; they have many features. To start, we'll learn how to open them, look at errors, and run JavaScript commands.



Browser Events



Data Types



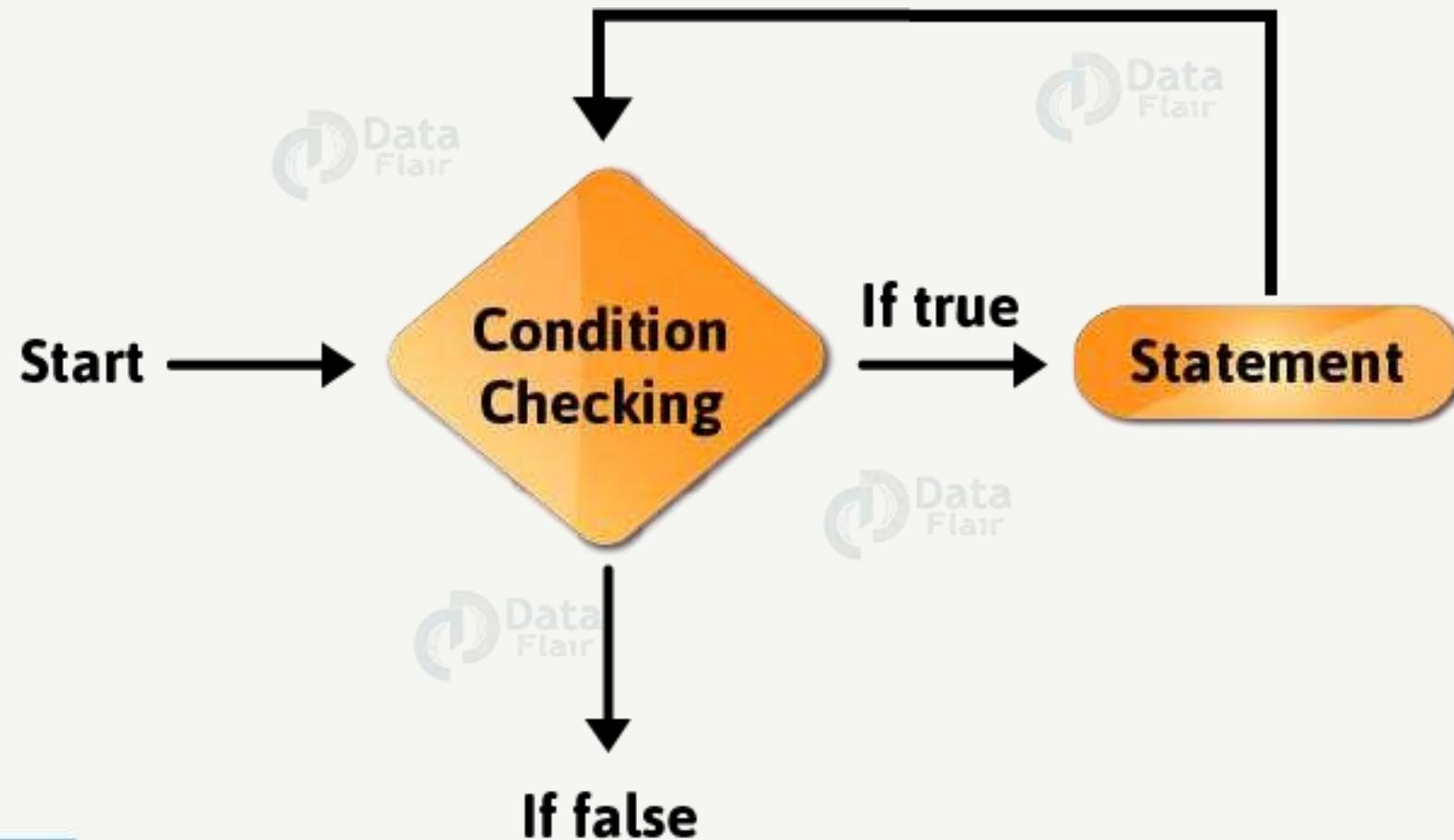
What is a JavaScript Loop?

- **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

- We repeat the sequence of instructions until the test condition is true. JavaScript provides the following loop statements to achieve this:
 - while statement
 - for statement
 - do..while statement
 - for...in statement
 - for...of statement
 - labeled statement
 - break statement
 - continue statement

While Statement



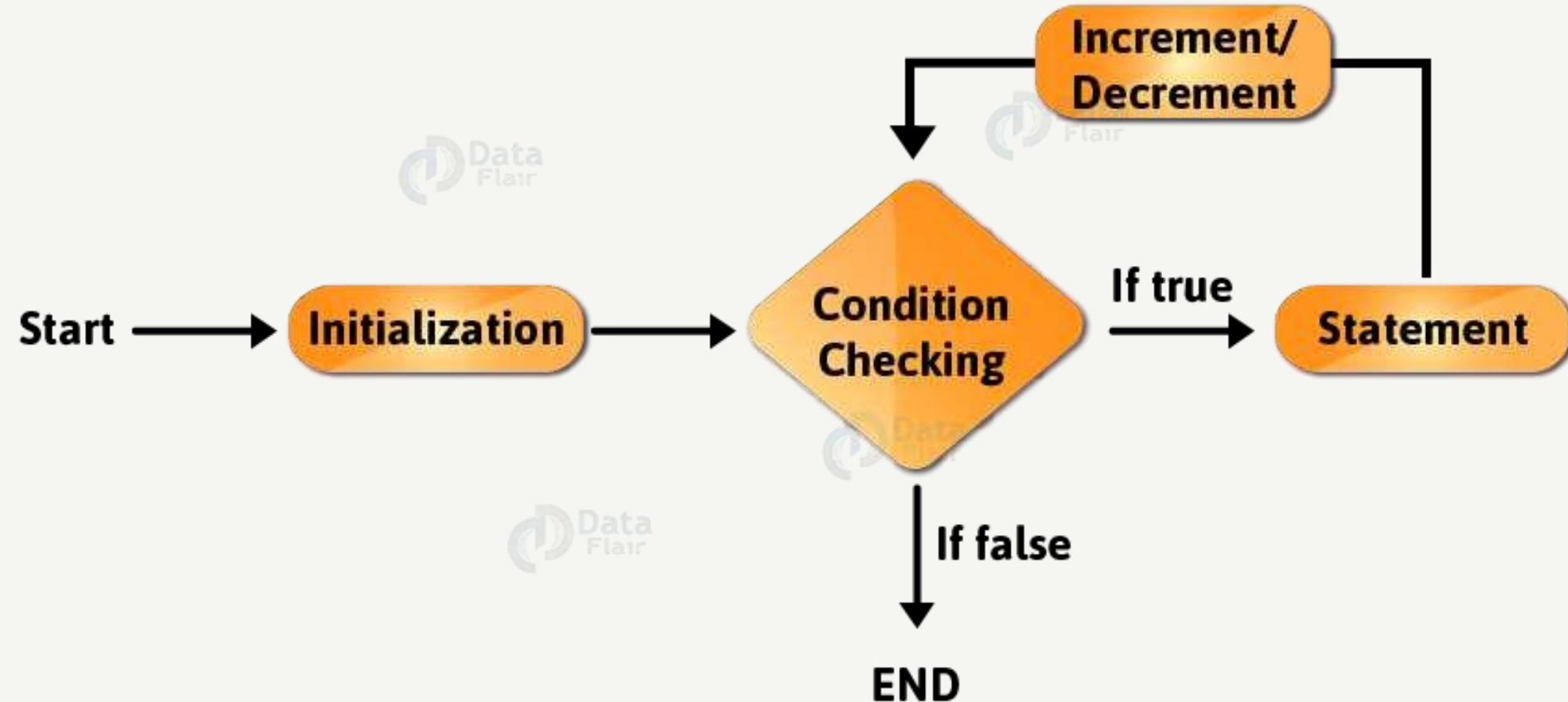
while

- A **while** statement in JavaScript executes until our boolean condition evaluates to **true**. This loop is an entry controlled loop.
- If the test condition returns **false**, then control passes to the statement just after the loop.
- If the test condition returns **true**, the loop body is executed and the condition is tested again.
- The syntax for a **while** statement is as follows:
- `while (boolean condition)`
- `statement`
- For multiple statements, group them with a block statement `{...}`.

While

```
<html>
  <body>
    <script>
var iterator1 = 0; //iteration variable initialized with 0
while (iterator1 < 5) //testing the condition
{
    document.write(iterator1 + 1 + ". " + "DataFlair Tutorial for JavaScript<br>");
    iterator1++; //incrementing the value of the variable
}
document.write("<br>Loop end");
</script>
</body>
</html>
```

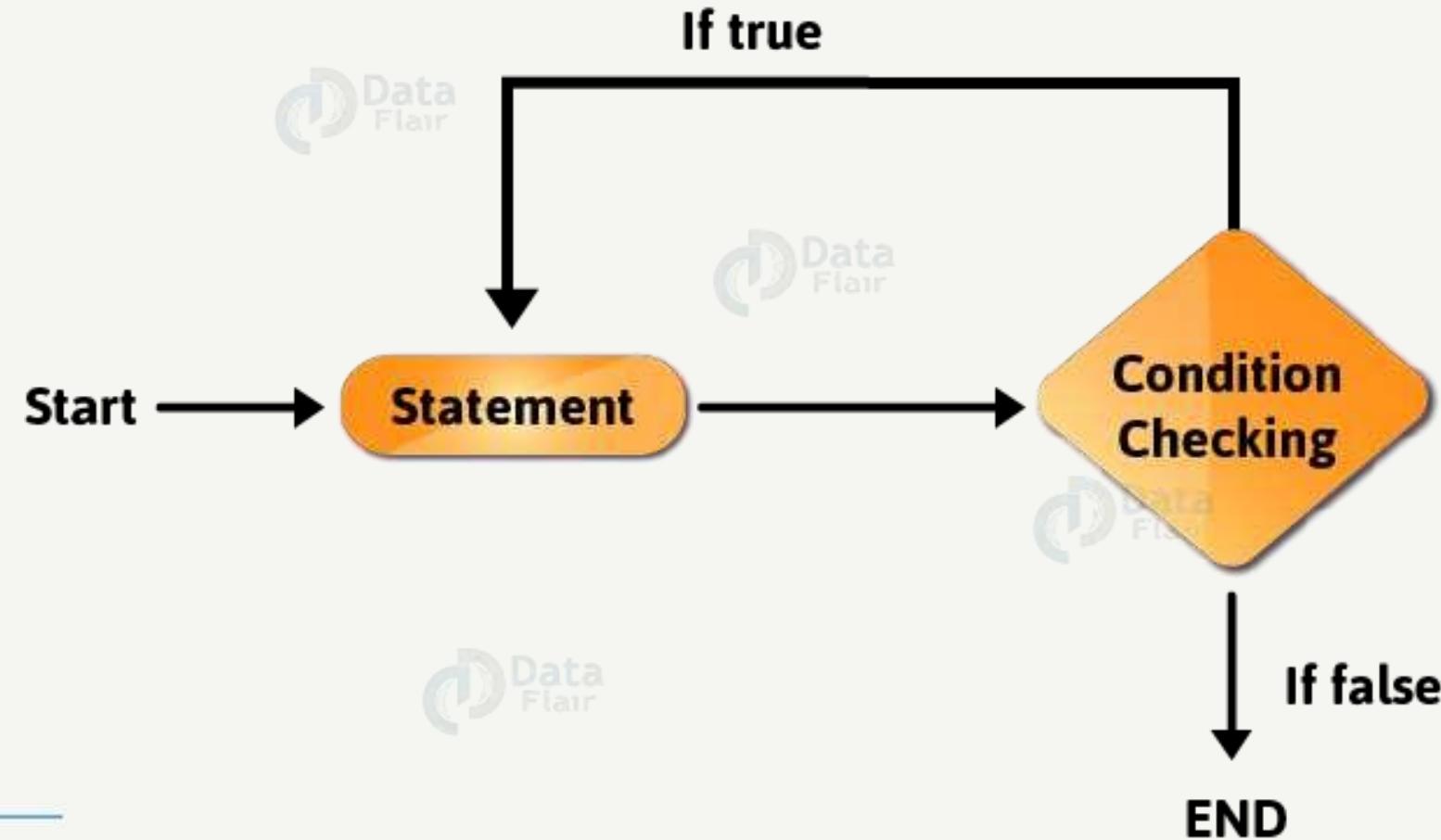
for Statement



For

- JavaScript **for loops** are similar to Java and C language **for loops**. It's an entry controlled loop. It consists of three parts, separated with a semicolon:
- **Initialization:** It initializes the loop with an iteration variable and executes once at the beginning of the loop.
- **Condition:** It specifies a certain condition in the loop that determines whether the loop body will execute or not. If the condition returns true, the code inside the **for** loop will execute. If it returns false, the control moves onto the statement after the loop.
- **Increment/ Decrement:** This section increments or decrements the value of our iteration variable by 1.

do...while Statement

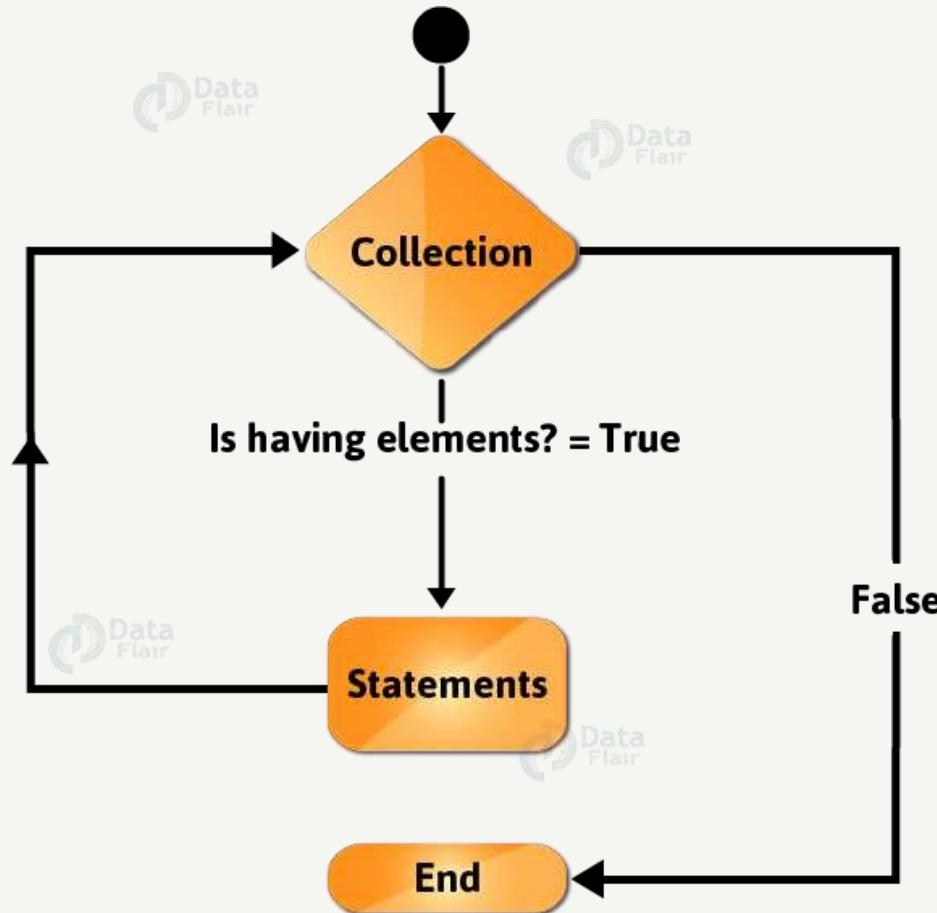


do...while Statement

- This is an exit controlled loop; the loop body executes at least once, even if the condition is not true. This is because the condition is tested when the loop body ends. If the condition returns **true**, the code inside the loop executes again. If it returns **false**, the JavaScript interpreter exits the loop.

```
do  
{  
//code executed at least once  
}while (condition);
```

for...in Statement



for...in Statement

- This is a modified **for loop** that doesn't require the programmer to know the number of iterations the loop needs to perform. It creates a loop iterating over all the enumerable properties of an object. The **keys** store the key of the current property and we access the property's value with it. This means that the loop executes a specific set of statements for each distinct property in a **JavaScript Object**. The working of a **for...in** statement in **JavaScript** is similar to **for-each** loop in Java.

The syntax of a **for...in** statement in JavaScript looks like this:

```
for (keys in objProperties) {  
    statements  
}
```

- The reason why the developers created the **for...in loop** is to work with user-defined properties in an object. It is better to use a traditional **for loop** over Array elements with numeric indexes. But, for this example, we are using a numeric array so you can understand how the loop works.

for...of Statement

- In the last JavaScript statement, we had to access the array's values with the help of square brackets []. We couldn't access them directly and it makes your program more error-prone. So the JavaScript developers got us a new way to access these values directly. This makes our code a lot more efficient than before because the values store the property's value, not the key.

A **for...of statement** looks as follows:

```
for (values of objProperties)
{
    statements
}
```

Infinite Loop

- One of the most common mistakes programmers make while using loops is creating an infinite loop. This happens when we accidentally add a condition that always returns true. It is very important that you avoid them in your code, but I still want you to practice these loops. Try thinking of some situations where the condition is always true. Let me give you an example:

```
while(1){  
//statements  
}
```

Topics

- Hello, world!
- Code structure
- The modern mode, "use strict"
- Variables
- Data types
- Type Conversions
- Operators
- Comparisons
- Interaction: alert, prompt, confirm
- Conditional operators: if, '?'
- Logical operators
- Loops: while and for
- The "switch" statement
- Functions
- Function expressions
- Arrow functions, the basics
- JavaScript specials

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.

JavaScript Primitives

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

- string
- number
- boolean
- null
- undefined

Array

- The JavaScript **Array** class is a global object that is used in the construction of arrays; which are high-level, list-like objects.
- Arrays are list-like objects whose prototype has methods to perform traversal and mutation operations. Neither the length of a JavaScript array nor the types of its elements are fixed. Since an array's length can change at any time, and data can be stored at non-contiguous locations in the array, JavaScript arrays are not guaranteed to be dense; this depends on how the programmer chooses to use them. In general, these are convenient characteristics; but if these features are not desirable for your particular use, you might consider using typed arrays.

Common operations

Create an Array

```
let fruits = [ 'Apple', 'Banana' ] console.log(fruits.length) // 2
```

Access an Array item using the index position

- `let first = fruits[0] // Apple`
- `let last = fruits[fruits.length - 1] // Banana`

Array

Loop over an Array

```
fruits.forEach(function(item, index, array)
  { console.log(item, index)
}) // Apple 0 // Banana 1
```

Add an item to the end of an Array

```
let newLength = fruits.push('Orange')
// ["Apple", "Banana", "Orange"]
```

Array

Remove an item from the end of an Array

```
let last = fruits.pop() // remove Orange (from the end) // ["Apple",  
"Banana"]
```

Remove an item from the beginning of an Array

```
let first = fruits.shift() // remove Apple from the front // ["Banana"]
```

Array

Add an item to the beginning of an Array

```
let newLength = fruits.unshift('Strawberry')
// add to the front // ["Strawberry", "Banana"]
```

Find the index of an item in the Array

```
fruits.push('Mango')
// ["Strawberry", "Banana", "Mango"]
let pos = fruits.indexOf('Banana')
// 1
```

Array

Remove an item by index position

```
let removedItem = fruits.splice(pos, 1)  
// this is how to remove an item  
// ["Strawberry", "Mango"]
```

Copy an Array

```
let shallowCopy = fruits.slice()  
// this is how to make a copy // ["Strawberry", "Mango"]
```

Array

Remove items from an index position

```
let vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot']
console.log(vegetables)
// ["Cabbage", "Turnip", "Radish", "Carrot"]
let pos = 1 let n = 2 let removedItems = vegetables.splice(pos, n)
// this is how to remove items, n defines the number of items to be
removed,
// starting at the index position specified by pos and progressing toward
the end of array.
console.log(vegetables) // ["Cabbage", "Carrot"] (the original array is
changed)
console.log(removedItems) // ["Turnip", "Radish"]
```

Array

Accessing array elements

```
let arr = ['this is the first element', 'this is the second element',
'this is the last element']

console.log(arr[0]) // logs 'this is the first element'
console.log(arr[1]) // logs 'this is the second element'
console.log(arr[arr.length - 1]) // logs 'this is the
last element'
```

Array

Relationship between length and numerical properties

- Several of the built-in array methods (e.g., `join()`, `slice()`, `indexOf()`, etc.) take into account the value of an array's `length` property when they're called.
- ```
const fruits = [] fruits.push('banana', 'apple', 'peach')
console.log(fruits.length) // 3
```
- ```
fruits[5] = 'mango' console.log(fruits[5]) // 'mango'
console.log(Object.keys(fruits)) // ['0', '1', '2', '5']
console.log(fruits.length) // 6
```

Array

Increasing the length.

```
fruits.length = 10
console.log(fruits)
// ['banana', 'apple', 'peach', empty x 2, 'mango', empty x 4]
console.log(Object.keys(fruits)) // ['0', '1', '2', '5']
console.log(fruits.length) // 10 console.log(fruits[8])
// undefined
```

Decreasing the length property does, however, delete elements.

```
fruits.length = 2
console.log(Object.keys(fruits)) // ['0', '1']
console.log(fruits.length) // 2
```

Array() constructor

The `Array()` constructor is used to create [Array](#) objects

```
// literal constructor  
[element0, element1, ..., elementN]  
  
// construct from elements  
new Array(element0, element1, ..., elementN)  
  
// construct from array length  
new Array(arrayLength)
```

Array literal notation

```
let fruits = ['Apple', 'Banana'];
console.log(fruits.length); // 2
console.log(fruits[0]); // "Apple"
```

Array constructor with a single parameter

```
let fruits = new Array(2); console.log(fruits.length); //
2 console.log(fruits[0]); // undefined
```

Array constructor with multiple parameters

```
let fruits = new Array('Apple',
'Banana'); console.log(fruits.length); //
2 console.log(fruits[0]); // "Apple"
```

Array.from()

The `Array.from()` static method creates a new, shallow-copied Array instance from an array-like or iterable object.

```
console.log(Array.from('foo'));
// expected output: Array ["f", "o", "o"]
console.log(Array.from([1, 2, 3], x => x + x));
// expected output: Array [2, 4, 6]
```

Parameters

arrayLike

An array-like or iterable object to convert to an array.

mapFn **Optional**

Map function to call on every element of the array.

thisArg **Optional**

Value to use as `this` when executing `mapFn`.

Return value

A new [Array](#) instance.

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword `new`.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using `Object.create()`.

Using an Object Literal

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs (like age:50) inside curly braces {}.
- The following example creates a new JavaScript object with four properties:
- `const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};`

Using the JavaScript Keyword new

- `const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";`

JavaScript Properties

- Accessing JavaScript Properties

The syntax for accessing the property of an object is:

- *objectName.property* // person.age

or

- *objectName["property"]* // person["age"]

or

- *objectName[expression]* // x = "age"; person[x]

- The expression must evaluate to a property name.

JavaScript Object Constructors

- ```
function Person(first, last, age, eye)
{ this.firstName = first;
 this.lastName = last;
 this.age = age;
 this.eyeColor = eye;
}
```

# Object Types (Blueprints) (Classes)

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, `function Person()` is an object constructor function.

Objects of the same type are created by calling the constructor function with the `new` keyword:

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

# Object Prototype

- JavaScript is a dynamic language. You can attach new properties to an object at any time as shown below.

```
function Student() { this.name = 'John'; this.gender = 'Male'; }
var studObj1 = new Student();
studObj1.age = 15; alert(studObj1.age); // 15
var studObj2 = new Student();
alert(studObj2.age); // undefined
```

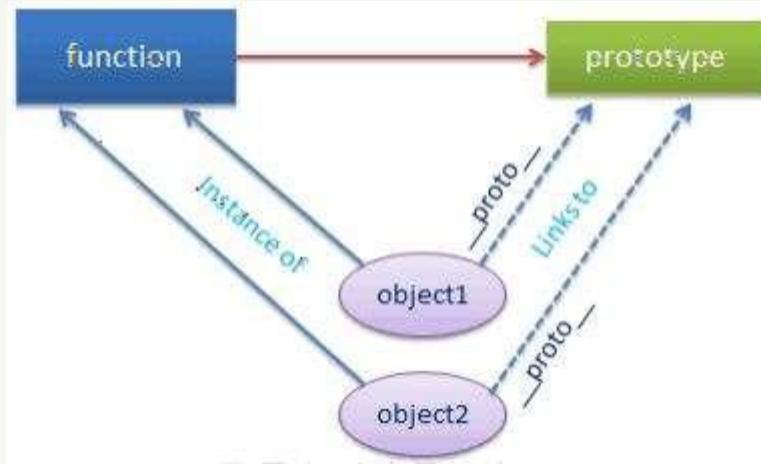
As you can see in the above example, age property is attached to studObj1 instance. However, studObj2 instance will not have age property because it is defined only on studObj1 instance.

So what to do if we want to add new properties at later stage to a function which will be shared across all the instances?

The answer is **Prototype**.

# Object Prototype

- The prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.



# Object Prototype

```
function Student()
{
 this.name =
 'John'; this.gender
 = 'M';
}
var studObj = new Student();
console.log(Student.prototype); // object
console.log(studObj.prototype); // undefined
console.log(studObj.__proto__); // object
console.log(typeof Student.prototype); // object
console.log(typeof studObj.__proto__); // object
console.log(Student.prototype === studObj.__proto__); // true
```

# Object's Prototype

- As mentioned before, object's prototype property is invisible.  
Use `Object.getPrototypeOf(obj)` method instead of `__proto__` to access prototype object.

```
function Student() {
 this.name = 'John';
 this.gender = 'M';
}
var studObj = new Student();
Student.prototype.sayHi= function(){ alert("Hi"); };
var studObj1 = new Student();
var proto = Object.getPrototypeOf(studObj1); // returns Student's prototype object
alert(proto.constructor); // returns Student function
```

# prototype object properties and methods.

| Property         | Description                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------|
| constructor      | Returns a function that created instance.                                                                |
| <u>__proto__</u> | This is invisible property of an object. It returns prototype object of a function to which it links to. |

# prototype object properties and methods.

| Method                 | Description                                                                                                                                                       |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hasOwnProperty()       | Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain. |
| isPrototypeOf()        | Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.                                     |
| propertyIsEnumerable() | Returns a boolean that indicates whether the specified property is enumerable or not.                                                                             |
| toLocaleString()       | Returns string in local format.                                                                                                                                   |
| toString()             | Returns string.                                                                                                                                                   |
| valueOf                | Returns the primitive value of the specified object.                                                                                                              |

Chrome and Firefox denotes object's prototype as `_proto_` which is public link whereas internally it reference as `[[Prototype]]`. Internet Explorer does not include `_proto_`. Only IE 11 includes it.

The `getPrototypeOf()` method is standardize since ECMAScript 5 and is available since IE 9.

# Changing Prototype

```
function Student()
{ this.name = 'John';
 this.gender = 'M';
}
Student.prototype.age = 15;
var studObj1 = new Student();
alert('studObj1.age = ' + studObj1.age); // 15
var studObj2 = new Student();
alert('studObj2.age = ' + studObj2.age); // 15
Student.prototype = { age : 20 };
var studObj3 = new Student();
alert('studObj3.age = ' + studObj3.age); // 20
alert('studObj1.age = ' + studObj1.age); // 15
alert('studObj2.age = ' + studObj2.age); // 15
```

# Inheritance in JavaScript

- Inheritance is an important concept in object oriented programming. In the classical inheritance, methods from base class get copied into derived class.
- In JavaScript, inheritance is supported by using prototype object. Some people call it "Prototypal Inheritance" and some people call it "Behaviour Delegation".
- Let's see how we can achieve inheritance like functionality in JavaScript using prototype object.
- Let's start with the Person class which includes FirstName & LastName property as shown below.

# Inheritance in JavaScript

```
function Person(firstName, lastName)
 { this.FirstName = firstName || "unknown";
 this.LastName = lastName || "unknown";
 };
Person.prototype.getFullName = function () {
 return this.FirstName + " " + this.LastName;
}
```

# Inheritance in JavaScript Multiple

```
function Person(firstName, lastName)
{ this.FirstName = firstName || "unknown";
this.LastName = lastName || "unknown"; }
Person.prototype.getFullName = function ()
{ return this.FirstName + " " + this.LastName; }
function Student(firstName, lastName, schoolName, grade) { Person.call(this,
firstName, lastName);
this.SchoolName = schoolName || "unknown"; this.Grade = grade || 0; }
//Student.prototype = Person.prototype;
Student.prototype = new Person();
Student.prototype.constructor = Student;
var std = new Student("James","Bond", "XYZ", 10);
alert(std.getFullName()); // James Bond
alert(std instanceof Student); // true alert(std instanceof Person); // true
```

# JavaScript Closure

- Closure is one of important concept in JavaScript. It is widely discussed and still confused concept. Let's understand what the closure is.
- First of all, let's see the definition of the Closure given by Douglas Crockford: [crockford.com/javascript/private.html](http://crockford.com/javascript/private.html)
- *Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.*
- You have learned that we can create [nested functions](#) in JavaScript. Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function). Consider the following example.

```
function OuterFunction() {
 var outerVariable = 1;
 function InnerFunction() {
 alert(outerVariable);
 }
 return InnerFunction;
}
var innerFunc = OuterFunction();
innerFunc(); // 100
```

# Use Of *Closure*

```
function Counter() {
 var counter = 0;
 function IncreaseCounter() {
 return counter += 1;
 };
 return IncreaseCounter;
}

var counter = Counter();
alert(counter()); // 1
alert(counter()); // 2
alert(counter()); // 3
alert(counter()); // 4
```

# Closure

```
function Counter() {
 var counter = 0; setTimeout(function () {
 var innerCounter = 0;
 counter += 1;
 alert("counter = " + counter);
 setTimeout(function ()
 { counter += 1;
 innerCounter += 1;
 alert("counter = " + counter + ", innerCounter = " + innerCounter) },
 500);
 }, 1000);
};
Counter();
```

# When to use Closure?

- Closure is useful in hiding implementation detail in JavaScript. In other words, it can be useful to create private variables or functions.

```
var counter = (function() { var privateCounter = 0; function changeBy(val) {
 privateCounter += val;
}
return {
 increment: function() {
 changeBy(1);
 },
 decrement: function()
 { changeBy(-1);
 }, value: function() {
 return privateCounter;
 } }; })();
alert(counter.value()); // 0
counter.increment();
counter.increment();
alert(counter.value()); // 2
counter.decrement();
alert(counter.value()); // 1
```

# Iterators and Generators

What is Iterables and Iterators?

ES6 introduces a new mechanism for traversing data: *iteration*. Two concepts are central to iteration:

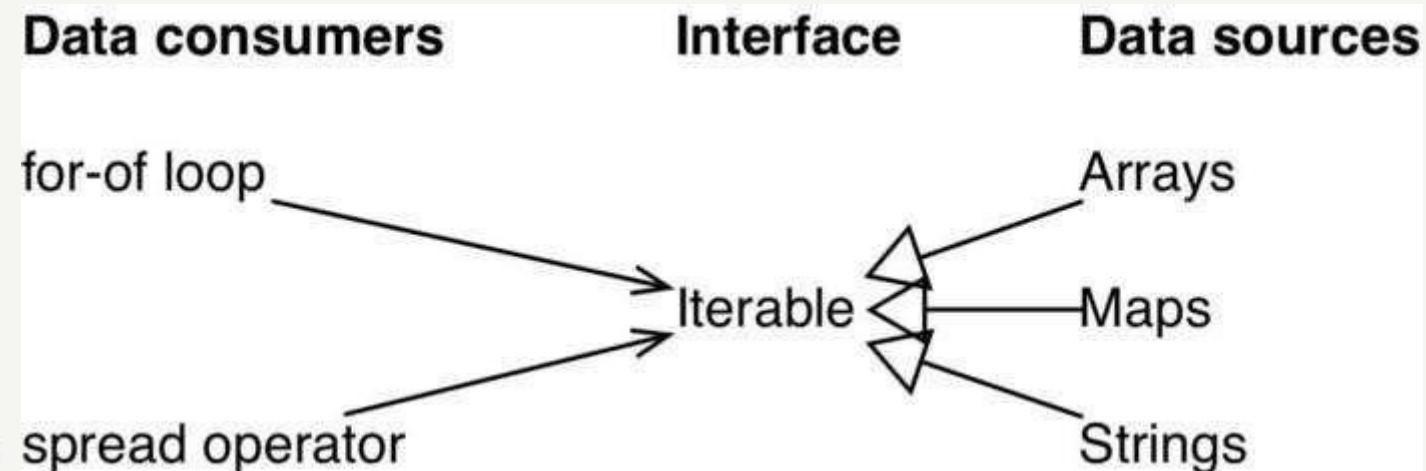
- An *iterable* is a data structure that wants to make its elements accessible to the public. It does so by implementing a method whose key is `Symbol.iterator`. That method is a factory for *iterators*.
- An *iterator* is a pointer for traversing the elements of a data structure (think cursors in databases)

# Iterable values in Javascript

- The following values are iterable:
  - Arrays
  - Strings
  - Maps
  - Sets
  - DOM data structures (work in progress)

# The idea of iterability is as follows.

- **Data consumers:** JavaScript has language constructs that consume data. For example, `for-of` loops over values and the spread operator (...) inserts values into Arrays or function calls.
- **Data sources:** The data consumers could get their values from a variety of sources. For example, you may want to iterate over the elements of an Array, the key-value entries in a Map or the characters of a string.



# The following ES6 language constructs make use of the Iterable:

Destructuring via an Array pattern

for-of loop

Array.from()

Spread operator (...)

Constructors of Maps and Sets

Promise.all(), Promise.race()

yield\*

# What are Generators?

- You can think of generators as processes (pieces of code) that you can pause and resume while executing that particular code :

```
function* func() {
 // (A)
 console.log('First');
 yield;
 console.log('Second');
}
```

The `yield` keyword pauses generator function execution and the value of the expression following the `yield` keyword is returned to the generator's caller. It can be thought of as a generator-based version of the `return` keyword.

# Generator function declarations:

- `function* genFunc() { ... }`  
`const genObj = genFunc();`

# Generator function expressions:

- `const genFunc = function* () { ... };`  
`const genObj = genFunc();`

# Generator method definitions in object literals:

- const obj = {  
  \* generatorMethod() {  
    ...  
  }  
};  
const genObj = obj.generatorMethod();

# Generator method definitions in class definitions

- ```
class MyClass {  
    * generatorMethod() {  
        ...  
    }  
}  
const myInst = new MyClass();  
const genObj = myInst.generatorMethod();
```

Usecase of Generators:

Simpler asynchronous code

we can execute simple asynchronous block of code.

Receiving asynchronous data using generators

ES8 will have async functions(async/await) which are internally based on generators. With them, the code looks like this:

```
async function fetchJson(url) {  
    try {  
        let request = await fetch(url);  
        let text = await request.text();  
        return JSON.parse(text);  
    }  
    catch (error) {  
        console.log(`ERROR: ${error.stack}`);  
    }  
}
```

Implementing Iterables

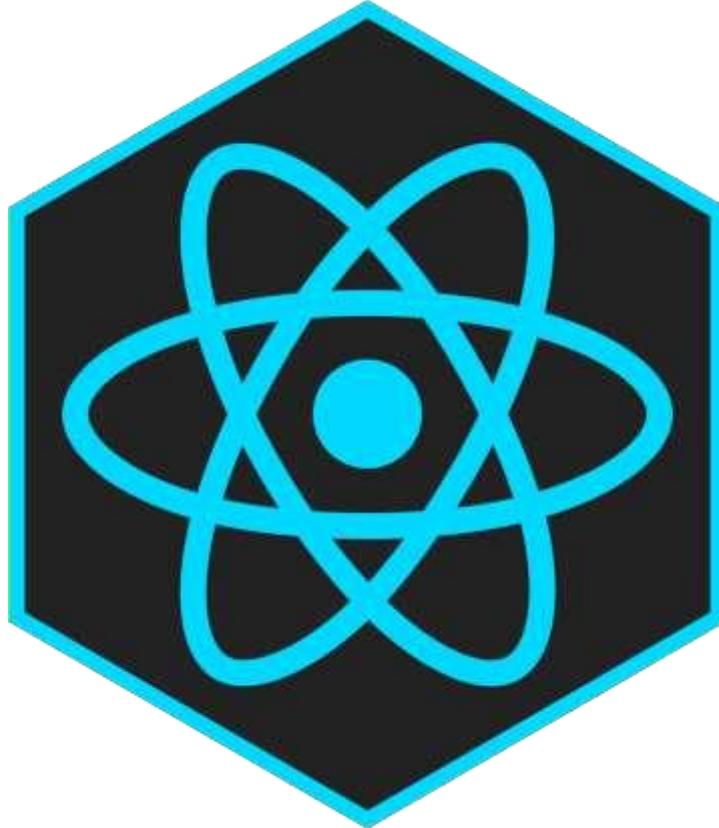
- ```
function* objectEntries(obj) {
 const propKeys = Reflect.ownKeys(obj);

 for (const propKey of propKeys) {
 // `yield` returns a value and then pauses
 // the generator. Later, execution continues
 // where it was previously paused.
 yield [propKey, obj[propKey]];
 }
}
```
- The above function returns an iterable over the properties of an object, one [key, value] pair per property, The objects returned by generators are iterable,using that iterable we can get next yield of [key,value] pair use of *for-of* loop,

# The following code use that Object.entries() method

- ```
const user= { name: 'raja', age: 25};  
for (const [key,value] of objectEntries(user)) {  
  console.log(` ${key}: ${value}`);  
}  
// Output:  
// name: raja  
// age: 25
```

[https://javascript.info/firs
t-steps](https://javascript.info/first-steps)



ReactJS

Module - 9

Fundamental of React js & Components

What is [HTML](#)

What is [CSS](#)

What is [DOM](#)

What is [ES6](#)

What is [Node.js](#)

What is [npm](#)

Introduction

- JavaScript is a cross-platform, object-oriented scripting language invented in web browsers to make web pages more dynamic and give feedback to your user. Adding JavaScript to your HTML code allows you to change completely the document appearance, from changing text, to changing colors, or changing the options available in a drop-down list, or switching one image with another when you roll your mouse over it and much more.
- JavaScript is mainly used as a client side scripting language. This means that all the action occurs on the client's side of things. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.

What can JavaScript do?

Display information based on the time of the day

JavaScript can check the computer's clock and pull the appropriate data based on the clock information.

Detect the visitor's browser

JavaScript can be used to detect the visitor's browser, and load another page specifically designed for that browser.

Control Browsers

JavaScript can open pages in customized windows, where you specify if the browser's buttons, menu line, status line or whatever should be present.

Validate forms data

JavaScript can be used to validate form data at the client-side saving both the precious server resources and time.

What can JavaScript do?

Create Cookies

JavaScript can store information on the visitor's computer and retrieve it automatically next time the user visits your page.

Add interactivity to your website

JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on a button.

Change page contents dynamically

JavaScript can randomly display content without the involvement of server programs .It can read and change the content of an HTML elements or move them around

How to implement JavaScript to an HTML page

You can link to outside files (with the file extension .js), or write blocks of code right into your HTML documents with the `<script>` tag. So, the `<script type="text/javascript">` and `</script>` tells where the JavaScript starts and ends.

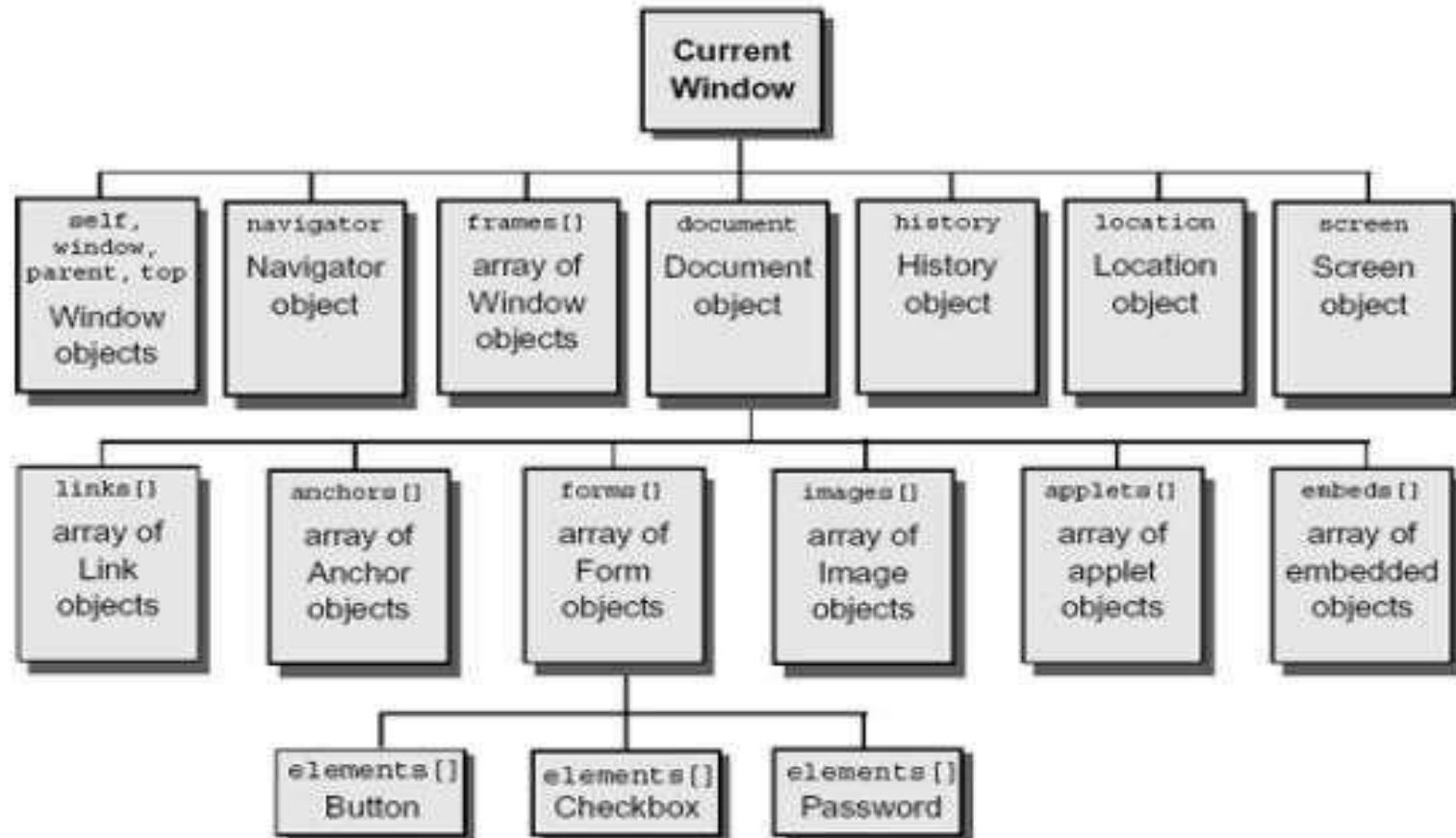
```
<html>
  <head>
    <title>My First Script</title>
  </head>
  <body>
    <script type="text/javascript">
      <!--
      //-->
    </script>
  </body>
</html>
```

Functions

Defining

```
<script type="text/javascript">
    function function_name (argument_1, ... ,
        argument_n){ statement_1;
        statement_2;
        statement_m;
        return return_value;
    }
</script>
```

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



Events

- By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger JavaScript. For example, we can use the `onClick` event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Events

- By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.
- Every element on a web page has certain events which can trigger JavaScript. For example, we can use the `onClick` event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Events

- Examples of events:
- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Electing an input field in an HTML form
- Submitting an HTML form
- A keystroke

Events

- Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!
- For a complete reference of the events recognized by JavaScript, go to our complete Event reference.
 - onLoad and onUnload
 - The onLoad and onUnload events are triggered when the user enters or leaves the page.
 - The onLoad and onUnload events are triggered when the user enters or leaves the page.

Events

- .The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information
- Both the onLoad and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".
- onFocus, onBlur and onChange

Events

- The onFocus, onBlur and onChange events are often used in combination with validation of form fields.
- Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:
- <input type="text" size="30" id="email" onchange="checkEmail()">
- onSubmit

Events

- onMouseOver and onMouseOut
- onMouseOver and onMouseOut are often used to create "animated" buttons.
- Below is an example of an onMouseOver event. An alert box appears when an event is detected: onMouseOver

```
<a href="http://www.w3schools.com" onmouseover="alert('onMouseOver event');return false"></a>
```

Example:-

```
<A HREF="jmouse.htm" onMouseover="window.status='Hi there!'; return true" onmouseout="window.status=' '; return true">Place your mouse here!</A>
```

Finding HTML Elements

Finding HTML Element By ID

`document.getElementById(id_string)`

Finding HTML Element By Tag Name

`document.getElementsByTagName(tag_name)`

Finding TML Element By Name

`document.getElementsByName(val)`

Finding HTML Element By Class

`document.getElementsByClassName(class_values)`

Finding HTML Elements

```
var myElement = document.getElementById("intro");
var x = document.getElementsByTagName("p");
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
var x = document.getElementsByClassName("intro");
var x = document.querySelectorAll("p.intro");
```

Module – 10

React Lists , Hooks , Localstorage , Api Project

What is ReactJS

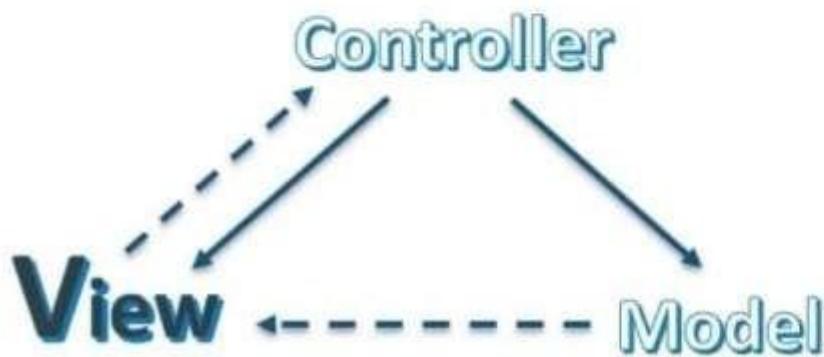
ReactJS is an open source JavaScript library used to develop User Interfaces.



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)



Why React?

React.js is a JavaScript library. It was developed by engineers at Facebook. Here are just a few of the reasons why people choose to program with React:

- React is fast. Apps made in React can handle complex updates and still feel quick and responsive.
- React is modular. Instead of writing large, dense files of code, you can write many smaller, reusable files. React's modularity can be a beautiful solution to JavaScript's [maintainability problems](#).
- React is scalable. Large programs that display a lot of changing data are where React performs best.

Why React?

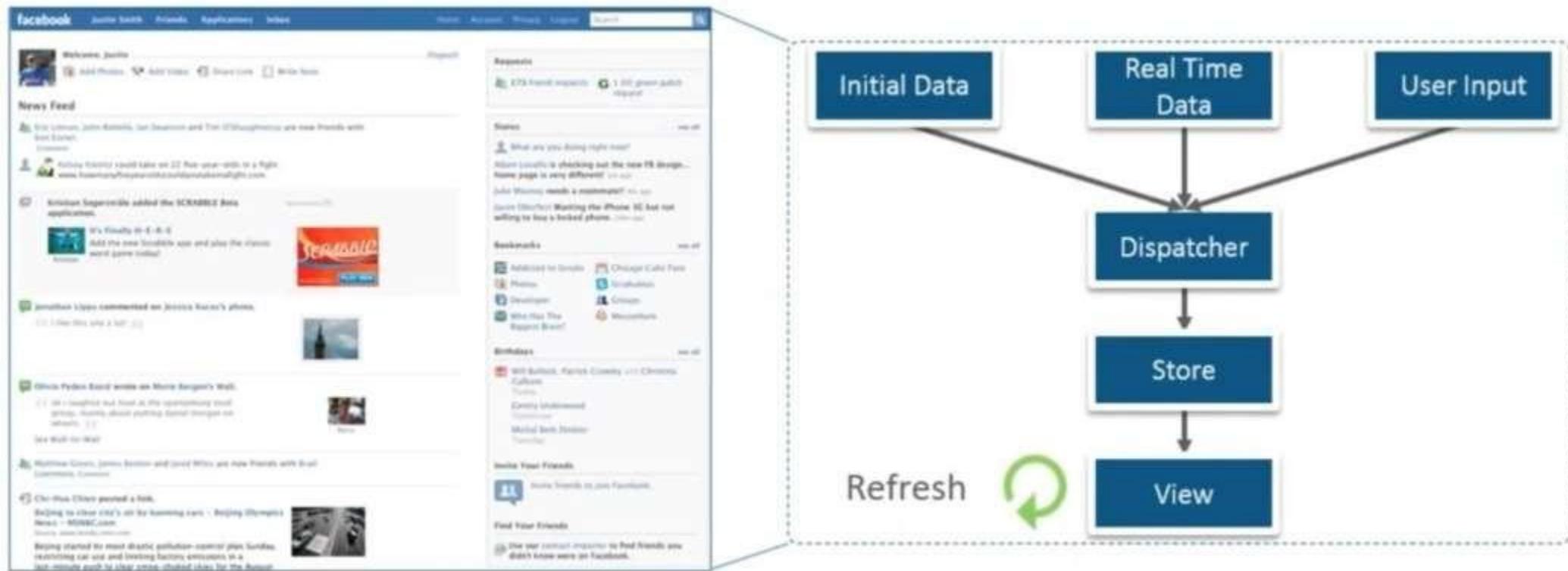
- React is flexible. You can use React for interesting projects that have nothing to do with making a web app. People are still figuring out React's potential. [There's room to explore.](#)
- React is popular. While this reason has admittedly little to do with React's quality, the truth is that understanding React will make you more employable.

If you are new to React, then this course is for you! No prior React knowledge is expected. We will start at the very beginning and move slowly.

If you are new to JavaScript, then consider taking [our JavaScript course](#) and then returning to React.

The three Codecademy React courses are not a high-level overview. They are a deep dive. Take your time! By the end, you will be ready to program in React with a real understanding of what you're doing.

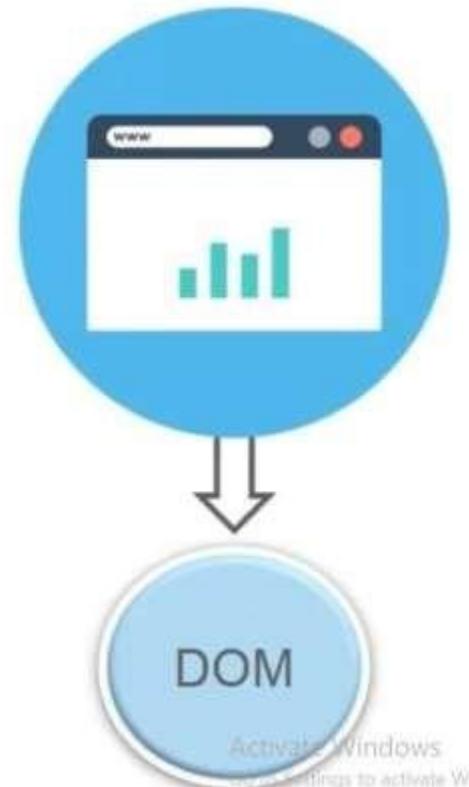
Why ReactJS



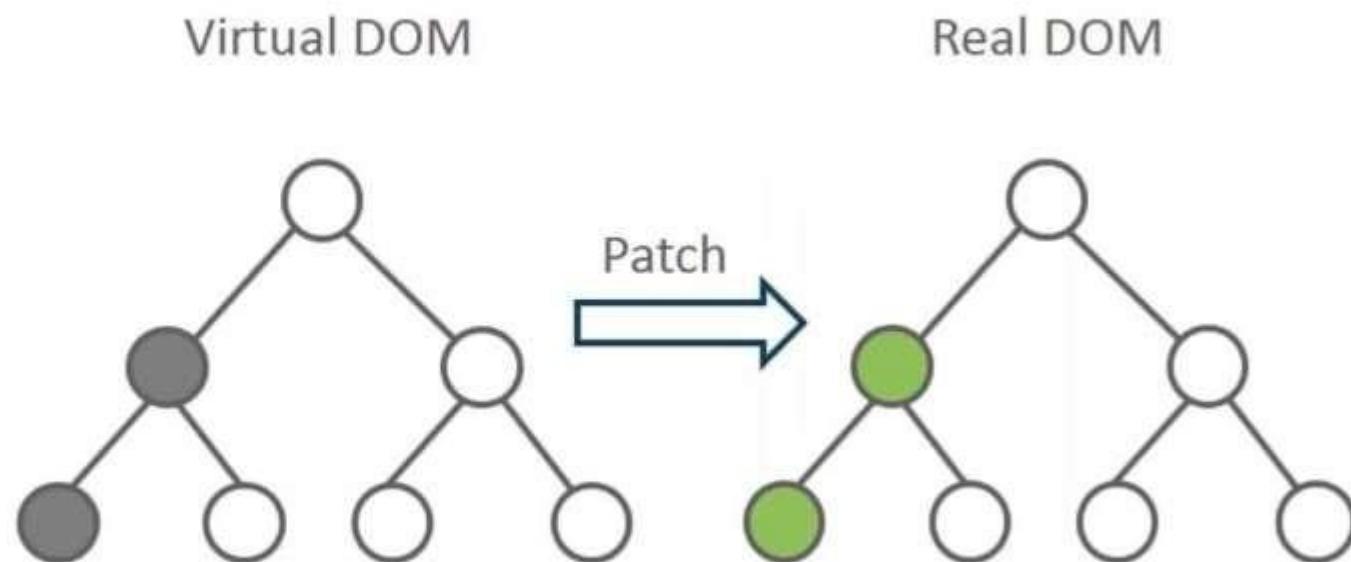
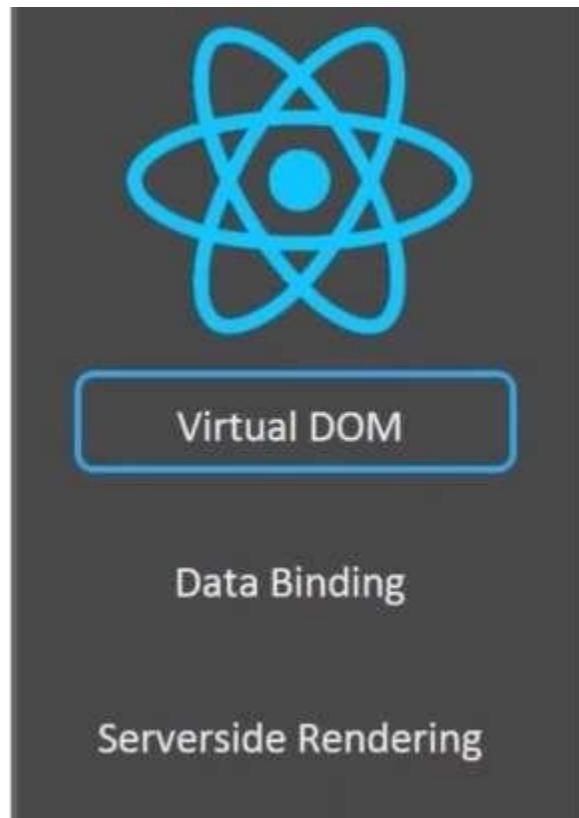
Why ReactJS

Drawbacks Of Traditional Data Flow

- Uses DOM (Document Object Model)
- More memory consumption
- Slow performance

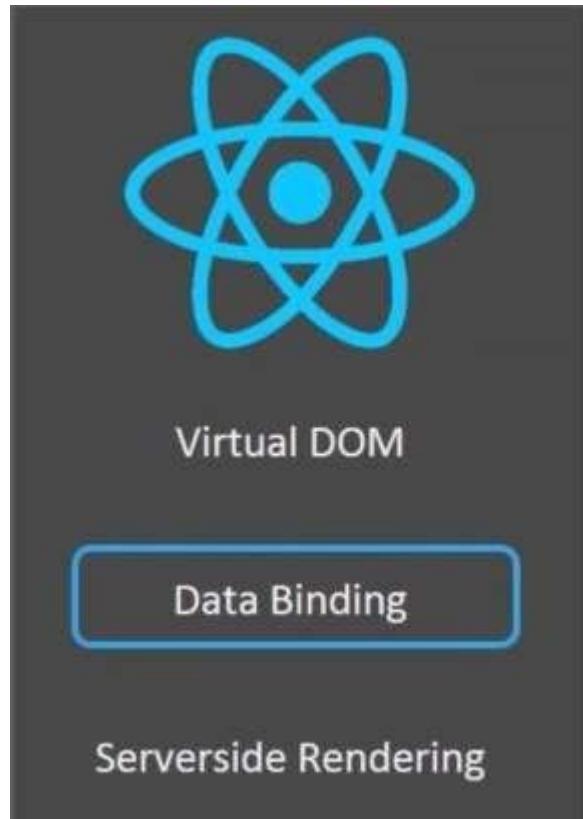


3D Aspects Of React Virtual Dom

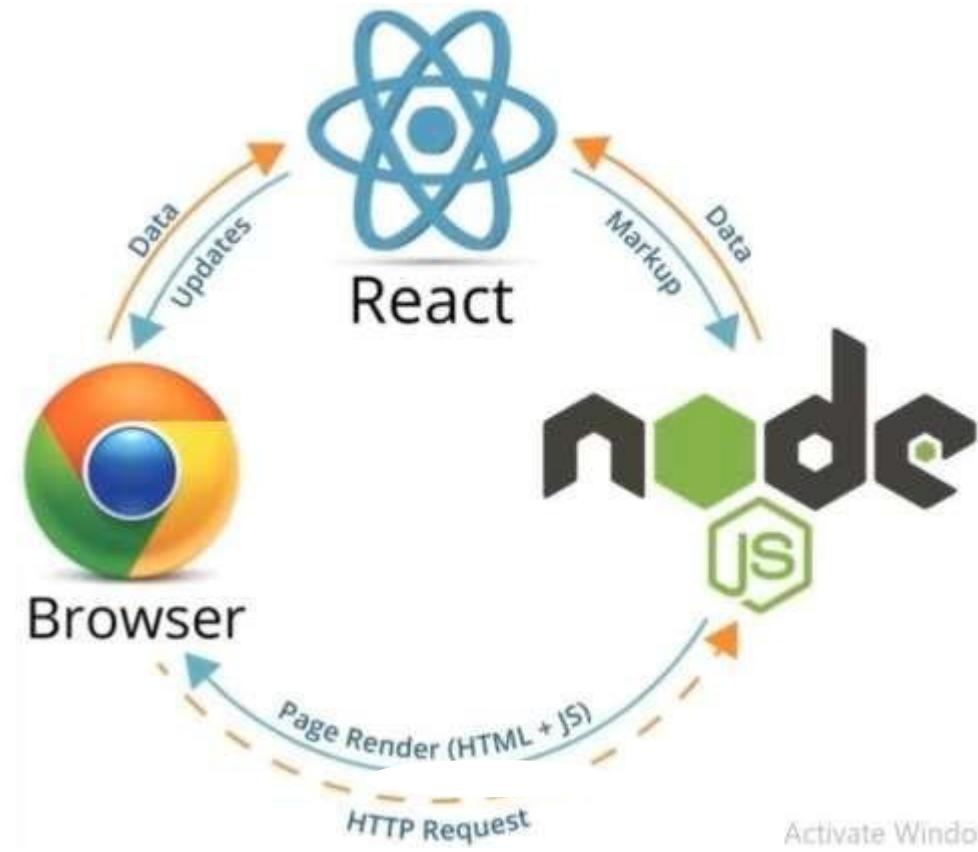
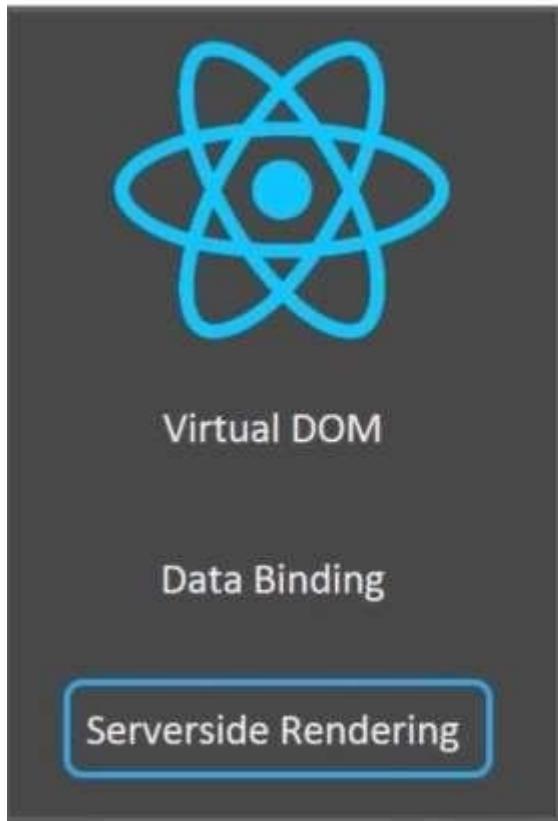


3D Aspects Of React

Unidirectional Data Binding



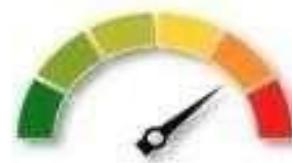
3D Aspects Of React Server-side Rendeting



Activate Windows

Advantages of ReactJS

Application's performance is increased



Used on Client as well as Server Side

Readability is improved



Easily used with other frameworks

Installation

ReactJS Installation

Add these <script> tags inside <head> tag of your HTML code.

```
html head
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <!-- React JS Scripts -->
</head>
<body>
</body>
</html>
```

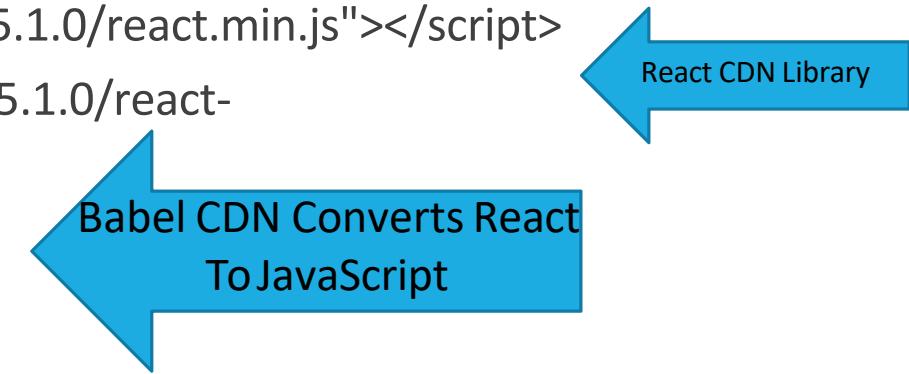
```
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.13.3/react.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.13.3/JSXTransformer.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
```

Activate Windows
Go to Settings to activate Windows.

edureka!

Installation

```
<!DOCTYPE html>
<html lang="en">
<title>Test React</title>
<script crossorigin src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react.min.js"></script>
<script crossorigin src="https://cdnjs.cloudflare.com/ajax/libs/react/15.1.0/react-dom.min.js"></script>
<script crossorigin src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.24.0/babel.js"></script>
<script crossorigin src="script.jsx" type="text/babel"></script>
<body>
<div id="root"></div>
</body>
</html>
```



Script.jsx

```
var MyComponent = React.createClass({  
  render : function ()  
    { return(<div>Hello world</div>)  
    }  
});  
  
ReactDOM.render(  
  <MyComponent/>,document.getElementById('root')  
);
```

Clock

Example :

<https://github.com/TopsCode/React/blob/master/Module2/2.1SimpleClock.html>

Dynamic Message Passing Through Props

Example :

https://github.com/TopsCode/React/blob/master/Module2/2.2function_Message_props.html



Date().toLocaleTimeString()

Example :

https://github.com/TopsCode/React/blob/master/Module2/2.3function_Message_state_time.html

React Component Props

Example :

<https://github.com/TopsCode/React/blob/master/Module2/2.4Props.html>



React Constructor

If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.

The constructor for a React component is called before it is mounted. When implementing the constructor for a `React.Component` subclass, you should call `super(props)` before any other statement. Otherwise, `this.props`

will be undefined in the constructor, which can lead to bugs.

React Constructor

Typically, in React constructors are only used for two purposes:

- Initializing local state by assigning an object to `this.state`.
- Binding event handler methods to an instance.

You should not call `setState()` in the `constructor()`. Instead, if your component needs to use local state, assign the initial state to `this.state` directly in the constructor:

```
constructor(props) {  
  super(props);  
  // Don't call this.setState() here!  
  this.state = { counter: 0 };  
  this.handleClick = this.handleClick.bind(this);  
}
```

React Constructor

Example : [With Default state value]

<https://github.com/TopsCode/React/blob/master/Module2/2.5constructor.html>

Example : [With SetState value]

<https://github.com/TopsCode/React/blob/master/Module2/2.6constructor1.html>

React CSS CDN

Example : [Css effect with CDN]

<https://github.com/TopsCode/React/blob/master/Module2/2.7CSSEffect.html>

Example : [JS function with CDN]

<https://github.com/TopsCode/React/tree/master/Module2/2.7JSXBuild>

Module –11

React -Advance React- Styling , Routing

React - Components, State, Props

Command Line Client

npm is installed with Node.js

This means that you have to install Node.js to get npm installed on your computer.

Download Node.js from the official Node.js web site: <https://nodejs.org>

C:\>npm install <package>

<https://nodejs.org/en/>

npm

- npm is the world's largest Software Library (Registry)
- npm is also a software Package Manager and Installer



Installation through node

Check Node version

- npm –version
- npm install -g create-react-app
- create-react-app –version
- create-react-app app-name
- cd app-name
- npm start

What is Babel?

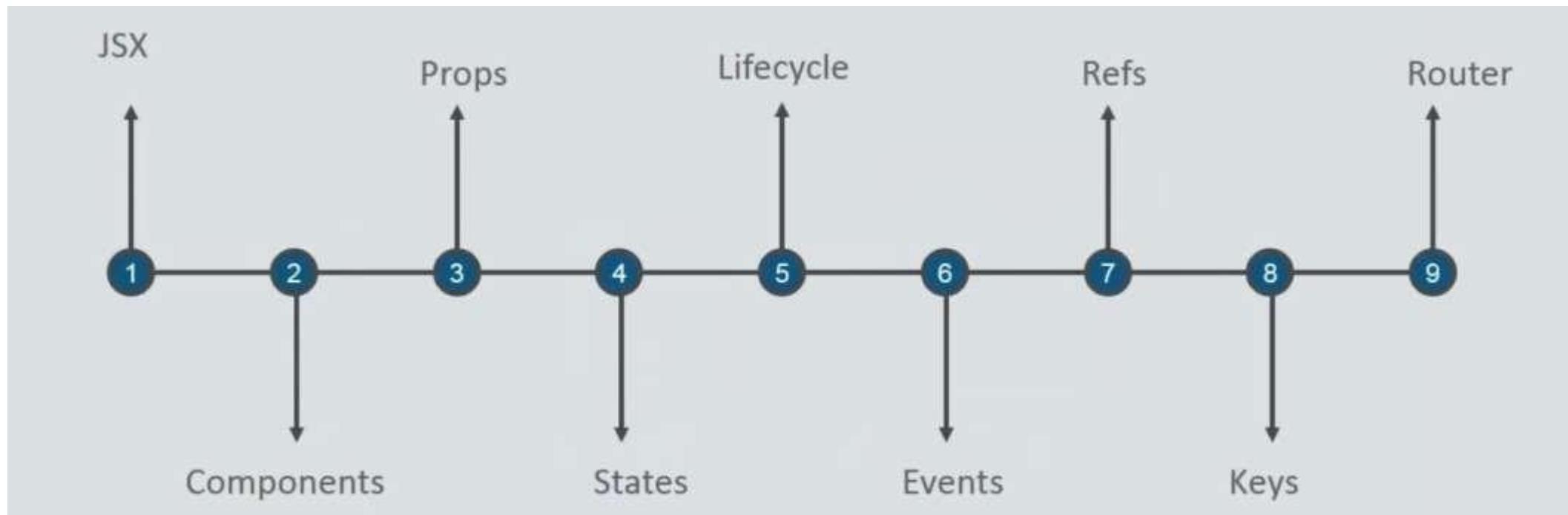
Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript.

With Babel, you can use the newest features of JavaScript (ES6 - ECMAScript 2015).

Babel is available for different conversions. React uses Babel to convert JSX into JavaScript.

- Please note that `<script type="text/babel">` is needed for using Babel.

ReactJS Fundamentals

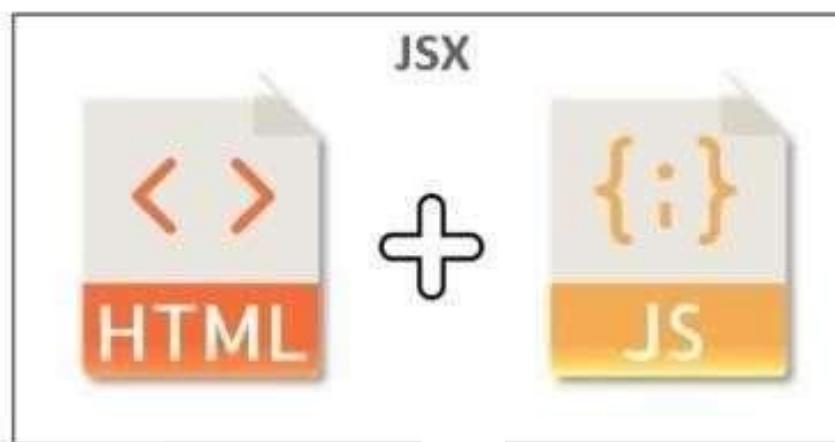


JSX

```
const element = <h2> Edureka! </h2> ;
```

```
function intro(){
    return <h1>Hello World!!</h1>;
}
```

- ✓ JSX stands for JavaScript XML
- ✓ Makes HTML easy to understand
- ✓ It is Robust
- ✓ Boosts up the JS performance



JSX Uses

Regular JSX

JSX Nested Elements

Specifying Attributes

Embedding JavaScript

```
var MyComponent = React.createClass({  
  render : function () {  
    return (  
      <div>  
        Hello World!!!  
      </div>  
    );  
  } );
```

JSX uses

Regular JSX

JSX Nested Elements

Specifying Attributes

Embedding JavaScript

```
var MyComponent = React.createClass( {  
  render : function () {  
    return (  
      <div>  
        <h1>Header</h1>  
        <h2>Content</h2>  
        <p>This is the content!!!</p>  
      </div>  
    );  
  }  
});
```

<h1>,<h2>,<p> nested
inside <div>

JSX uses

The diagram illustrates the use of JSX. On the left, there is a vertical stack of four dark grey rectangular buttons with white text: "Regular JSX", "JSX Nested Elements", "Specifying Attributes", and "Embedding JavaScript". To the right of this stack is a large, light-grey rounded rectangle containing a code snippet and a hand cursor icon. The code snippet is as follows:

```
var styles={ backgroundcolor: 'cyan' };
var MyComponent=React.createClass({
  render : function () {
    return (
      <div style={styles}>
        <h1>Header</h1>
      </div>
    );
  }
});
```

A hand cursor icon is positioned over the closing brace of the render function. A blue ribbon-like graphic starts from the bottom right and loops around the code area, ending with a small box containing the text "Adding Attributes".

JSX Uses

Regular JSX

JSX Nested Elements

Specifying Attributes

Embedding JavaScript

JavaScript Expression

```
var MyComponent = React.createClass({  
  render: function () {  
    return (  
      <div>  
        <h2> {2+4} </h2>  
      </div>  
    );  
  }  
});
```

Styling and CSS

How do I add CSS classes to components?

```
render() {  
    return <span className="menu navigation-menu">Menu</span>  
}
```

It is common for CSS classes to depend on the component props or state:

```
render() {  
    let className = 'menu';  
    if (this.props.isActive)  
    { className += ' menu-active';}  
    return <span className={className}>Menu</span>  
}
```

Styling and CSS

Example : Link

<https://github.com/TopsCode/React/blob/master/Module3/3.1indexSimpleCSSWithConst.js>

External CSS:

<https://github.com/TopsCode/React/blob/master/Module3/3.2.1indexExternalCSS.js>

Styling and CSS with Styled Component

Example : Link

<https://github.com/TopsCode/React/blob/master/Module3/3.2.2indexStyledComponent.js>

styled-components

- styled-components is the result of wondering how we could enhance CSS for styling React component systems. By focusing on a single use case we managed to optimize the experience for developers as well as the output for end users.
- Automatic critical CSS: styled-components keeps track of which components are rendered on a page and injects their styles and nothing else, fully automatically. Combined with code splitting, this means your users load the least amount of code necessary.
- No class name bugs: styled-components generates unique class names for your styles. You never have to worry about duplication, overlap or misspellings.

styled-components

Easier deletion of CSS: it can be hard to know whether a class name is used somewhere in your codebase. styled-components makes it obvious, as every bit of styling is tied to a specific component. If the component is unused (which tooling can detect) and gets deleted, all its styles get deleted with it.

Simple dynamic styling: adapting the styling of a component based on its props or a global theme is simple and intuitive without having to manually manage dozens of classes.

Painless maintenance: you never have to hunt across different files to find the styling affecting your component, so maintenance is a piece of cake no matter how big your codebase is.

Automatic vendor prefixing: write your CSS to the current standard and let styled-components handle the rest.

- You get all of these benefits while still writing the CSS you know and love, just bound to individual components.

CSS

```
const divStyle = {  
  margin: '40px',  
  border: '5px solid pink'  
};  
const pStyle =  
{ fontSize:  
'15px', textAlign:  
'center'  
};
```

```
const Box = () => (  
  <div style={divStyle}>  
    <p style={pStyle}>Get started  
    with inline style</p>  
  </div>  
);  
ReactDOM.render( <Box />,  
document.getElementById("root"));
```

Installation

Installing styled-components only takes a single command and you're ready to roll:

```
npm install --save styled-components
```

NOTE It's highly recommended (but not required) to also use the Babel plugin. It offers many benefits like more legible class names, server-side rendering compatibility, smaller bundles, and more.

OR (CDN)

```
<script src="https://unpkg.com/styled-  
components/dist/styled-components.min.js"></script>
```

Reactstrap

Install reactstrap and peer dependencies via NPM

Importing

```
npm install --save reactstrap react react-dom
```

```
import {Container, Row, Col} from 'reactstrap';
```

Example:

<https://github.com/TopsCode/React/blob/master/Module3/3.2.3indexReactStrap1.js>

Bootstrap

React-Bootstrap is a complete re-implementation of the Bootstrap components using React. It has no dependency on either bootstrap.js or jQuery. If you have React setup and React-Bootstrap installed you have everything you need.

```
npm install react-bootstrap bootstrap
```

```
import Button from 'react-bootstrap/Button';
// or less ideally
import { Button } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
```

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.2.4indexBootstrapDivStructure.js>

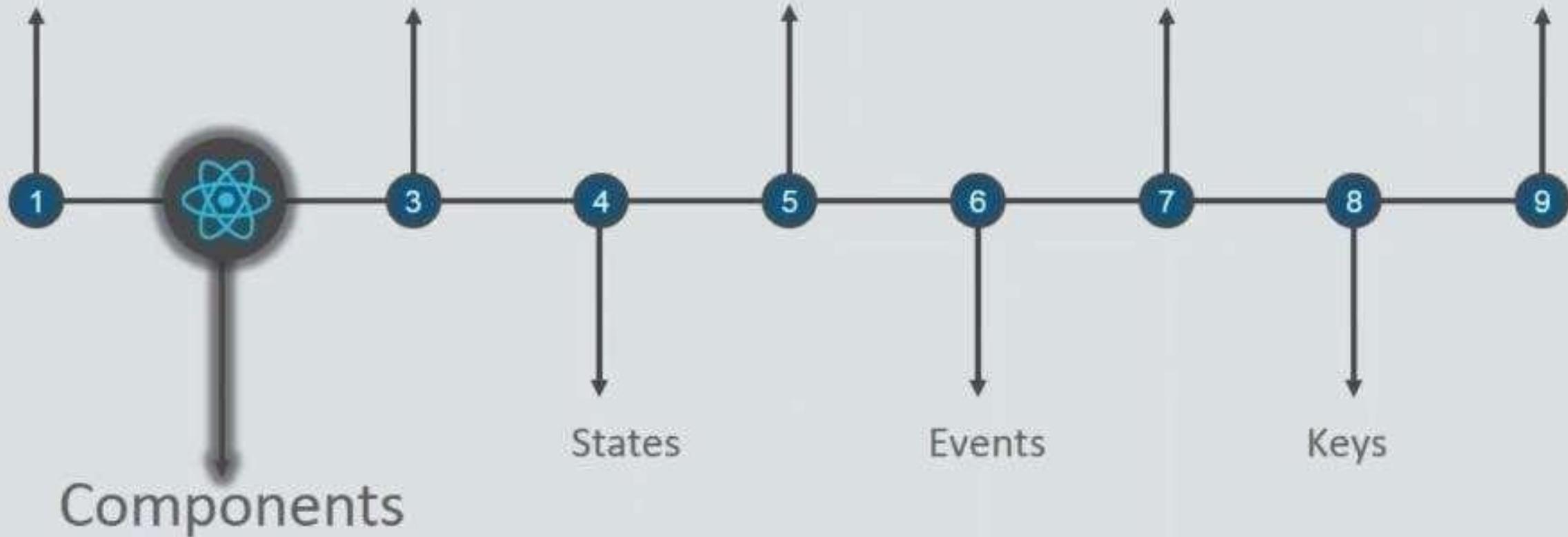
JSX

Props

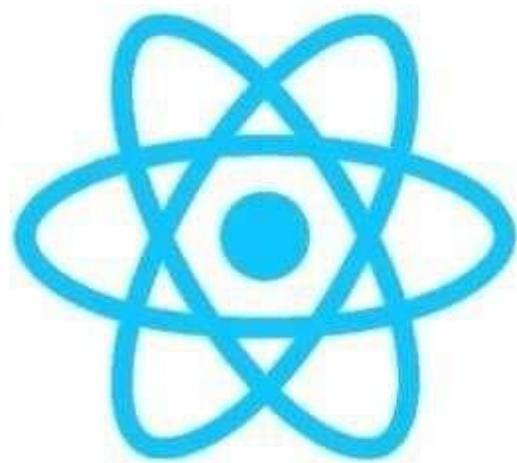
Lifecycle

Refs

Router



Components

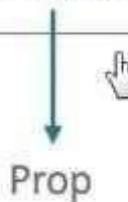


- 1 Everything in ReactJS is component
- 2 Each Component Return a DOM Object
- 3 It splits the UI into independent reusable pieces
- 4 Each independent piece is processed separately
- 5 It can refer other components in output
- 6 It can be further split into smaller components

Components

- ✓ A valid React component, accepts a single 'props' object argument to produce a React element.
- ✓ These are called "functional" as they literally are JavaScript functions.

```
<button onClick={this.handleClick}>Click Here</button>
```



Components

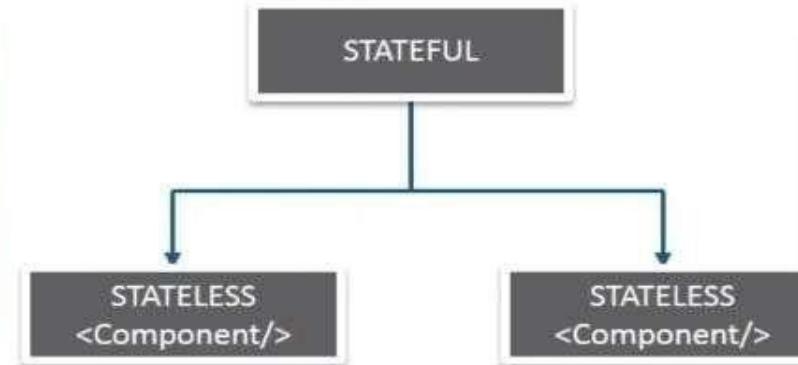
Simplest way of defining a component is through JavaScript



```
function PropDemo(props) {  
  return <h1> Hello...{props.name}</h1>  
}
```

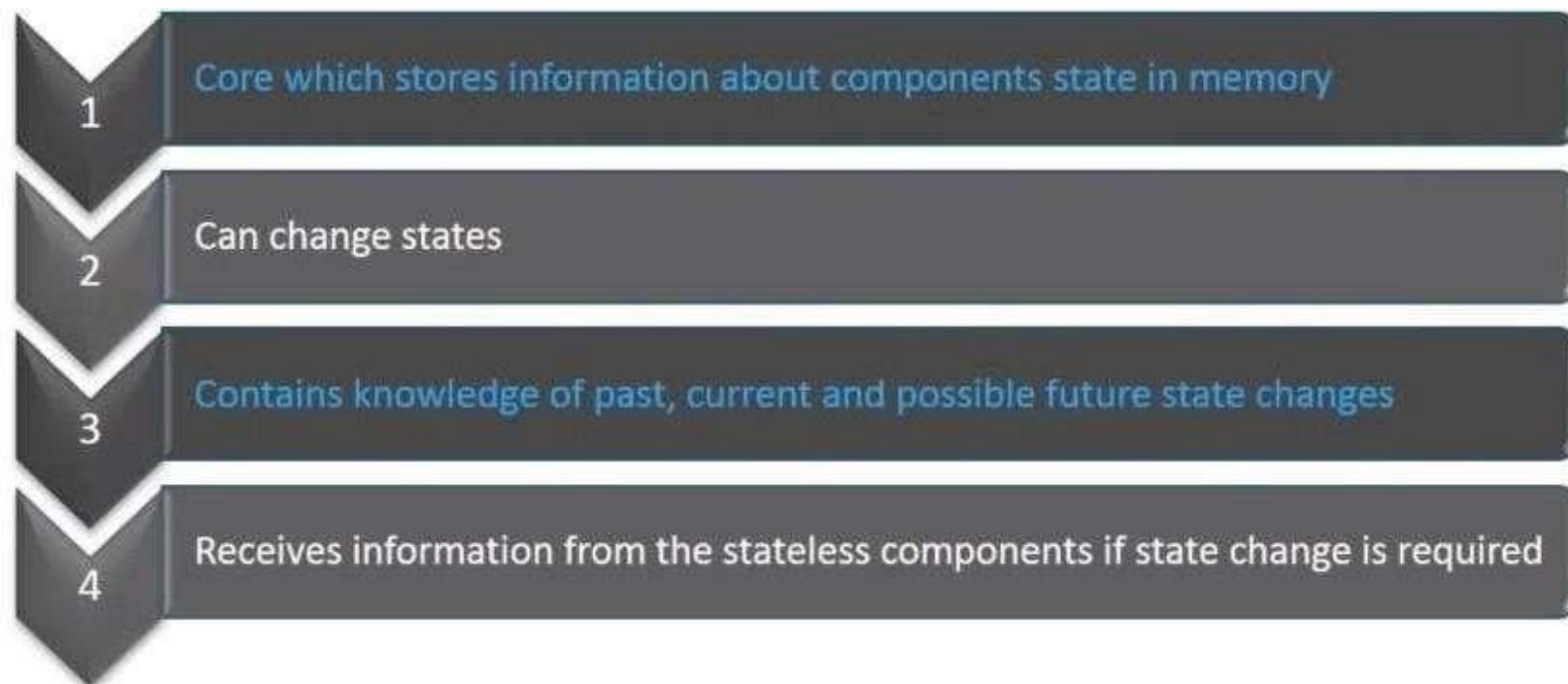
Components in ReactJS can be in two forms:

1. **Stateful**: Remembers everything it does
2. **Stateless**: Doesn't remembers anything it does.

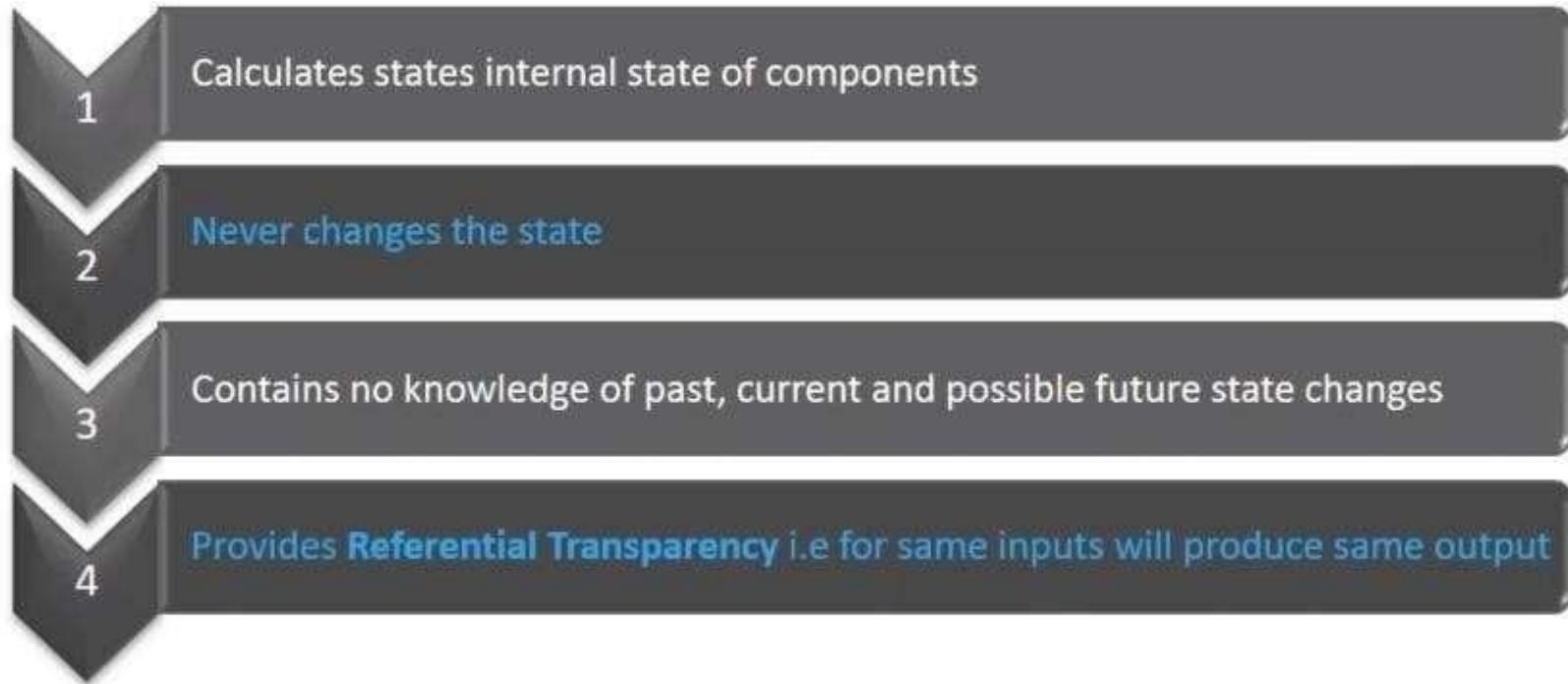


Example : Link

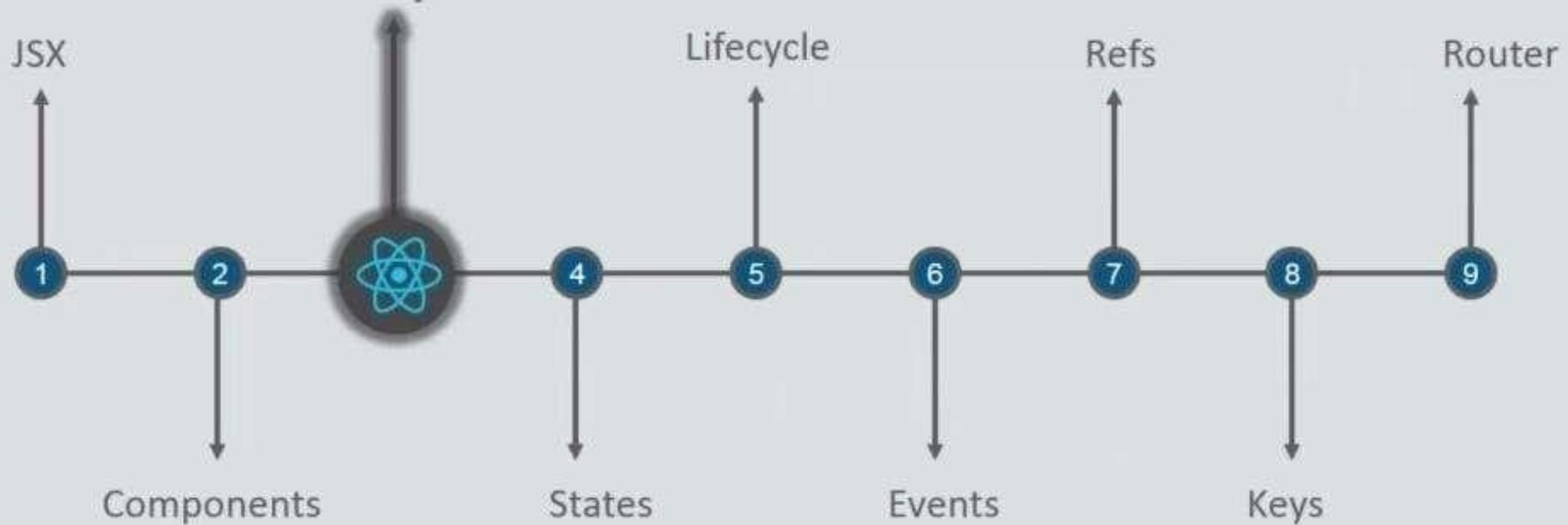
State-ful Components

- 
- 1 Core which stores information about components state in memory
 - 2 Can change states
 - 3 Contains knowledge of past, current and possible future state changes
 - 4 Receives information from the stateless components if state change is required

Stateless Components

- 
- 1 Calculates states internal state of components
 - 2 Never changes the state
 - 3 Contains no knowledge of past, current and possible future state changes
 - 4 Provides **Referential Transparency** i.e for same inputs will produce same output

Props

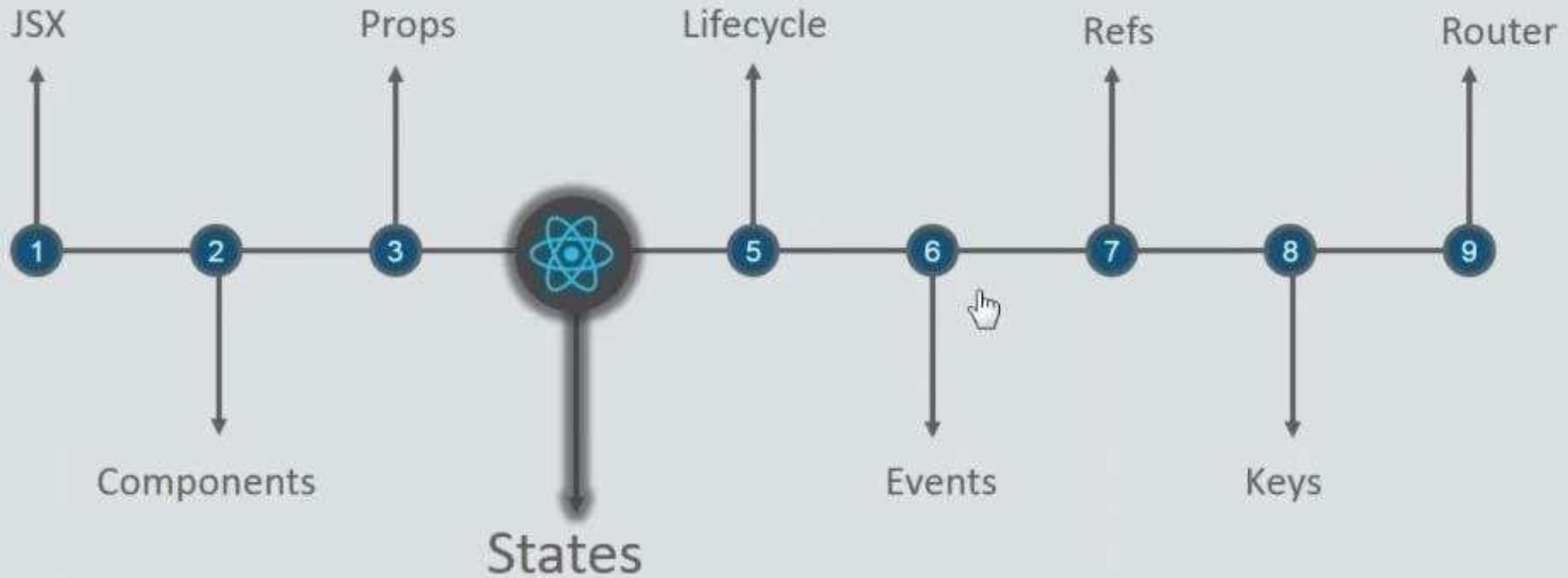


Props

- ✓ Props are read-only components
- ✓ Whether components are declared as function or class, it must never change its *props*
- ✓ Such components are called 'pure functions'

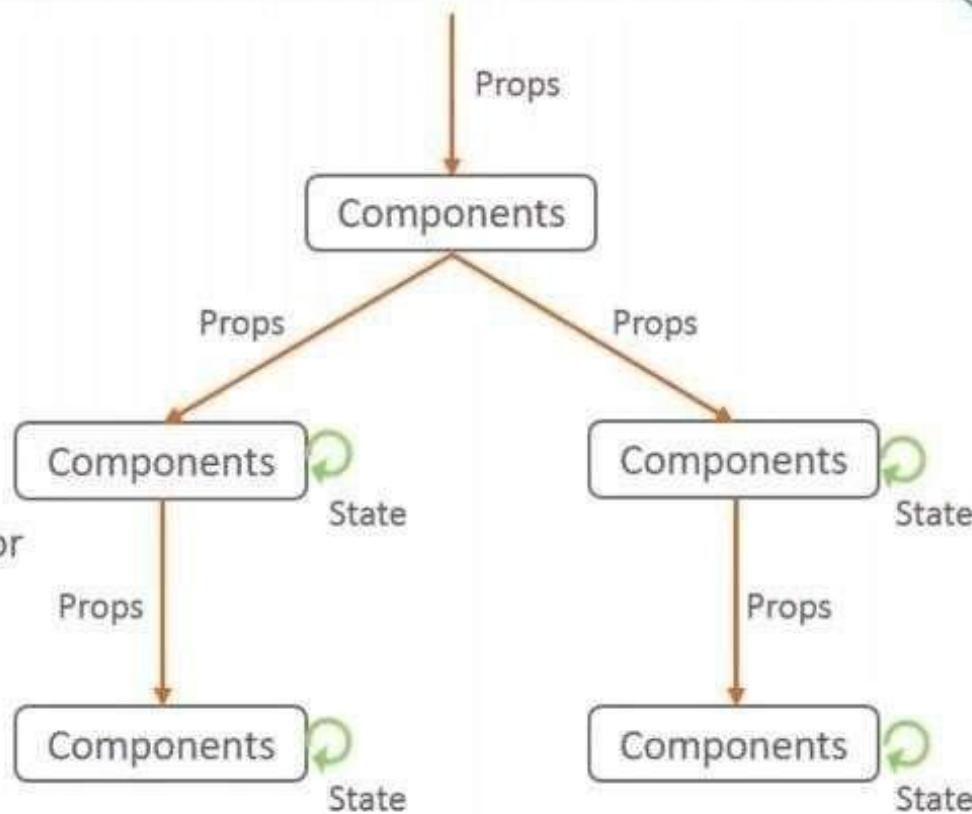
```
function sum(x,y) {  
    return x+y ;  
}
```

RULE: *All React components must act like pure functions with respect to their props.*



States

- ✓ Heart of react components
- ✓ Must be kept as simple as possible
- ✓ Determines components rendering and behavior
- ✓ Creates dynamic and interactive components



States

```
↳ CalUSX.html    ◊ script.jsx  ✘

 1  var style = {color:'grey'};
 2  var styleForHeading = {color:'skyblue'};
 3
 4  var Header = React.createClass({
 5    render : function () {
 6      return(
 7        <div>
 8          <h1>{ this.props.propsForPageTitle }</h1>
 9          <h2>This is the header Component</h2>
10        </div>
11    );
12  }
13);
14);
15 var Footer = React.createClass({
16   getInitialState:function(){
17     return{
18       user:"TOPS User",
19       rank : '1'
20     }
21 },
22 render : function () {
23   setTimeout(()=>{this.setState({user:'Jay Amin',rank:'1st'})},3000);
24   return(
25     <div>
26       <p>user is : {this.state.user}</p>
27       <p>Rank is : {this.state.rank}</p>
28       <h2>This is the Footer Component</h2>
29     </div>
30   );
31 });
32 });
33 );
```

```
34  var MyComponent = React.createClass({
35    render : function () {
36      return(
37        <div>
38          <Header propsForPageTitle = 'AboutUs' />
39
40          <h1 style={style}>Hello world</h1>
41          <h2 style={styleForHeading}>TOPS Technologies</h2>
42          <h2 style={styleForHeading}>Sum of 2 + 2 is : {2+2}</h2>
43          <Footer />
44        </div>
45      )
46    }
47  });
48
49  ReactDOM.render(
50    <MyComponent />,document.getElementById('root')
51  );
```

States

Example :

[https://github.com/TopsCode/React/blob/master/Module3/
3.4.1indexToggle.js](https://github.com/TopsCode/React/blob/master/Module3/3.4.1indexToggle.js)

Example 2:

[https://github.com/TopsCode/React/blob/master/Module3
/3.4.2indexOfSimpleComponent.js](https://github.com/TopsCode/React/blob/master/Module3/3.4.2indexOfSimpleComponent.js)

States

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.4.1indexToggle.js>

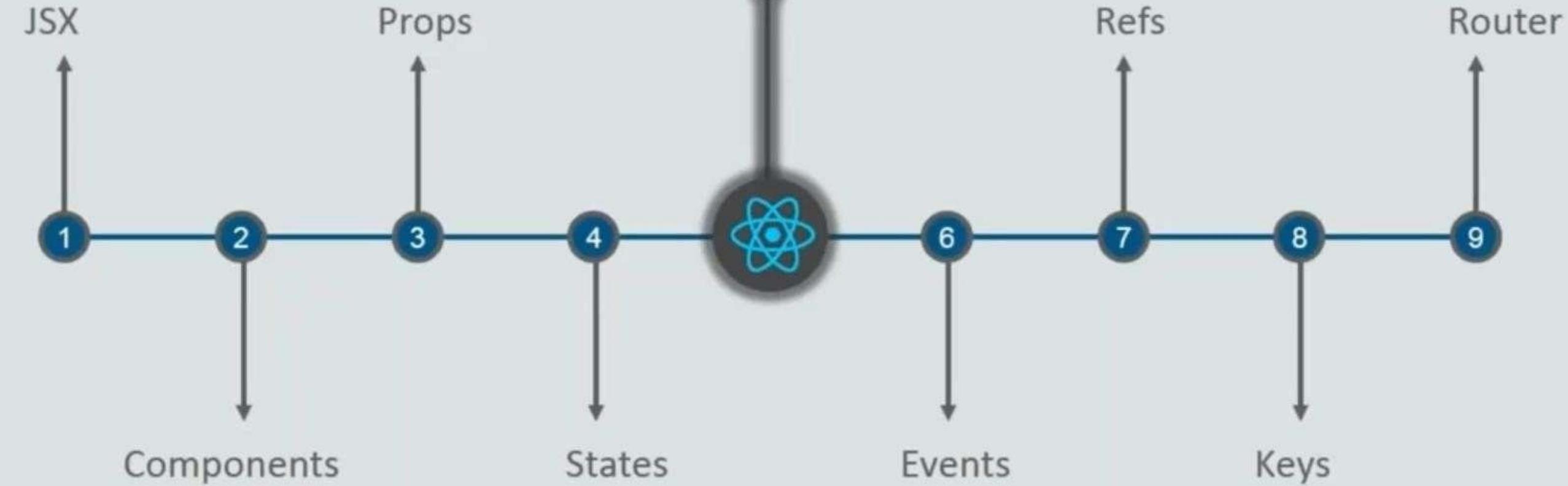
Example 2:

<https://github.com/TopsCode/React/blob/master/Module3/3.4.2indexOfSimpleComponent.js>

Example 3:

<https://github.com/TopsCode/React/blob/master/Module3/3.4.4indexInheritanceWithProps.js>

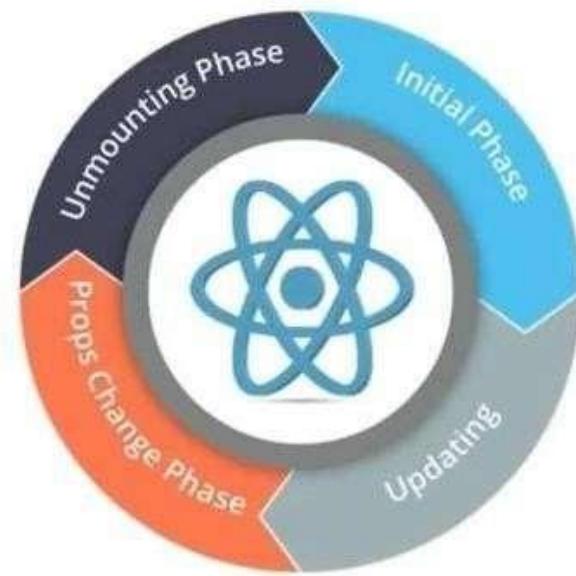
Lifecycle



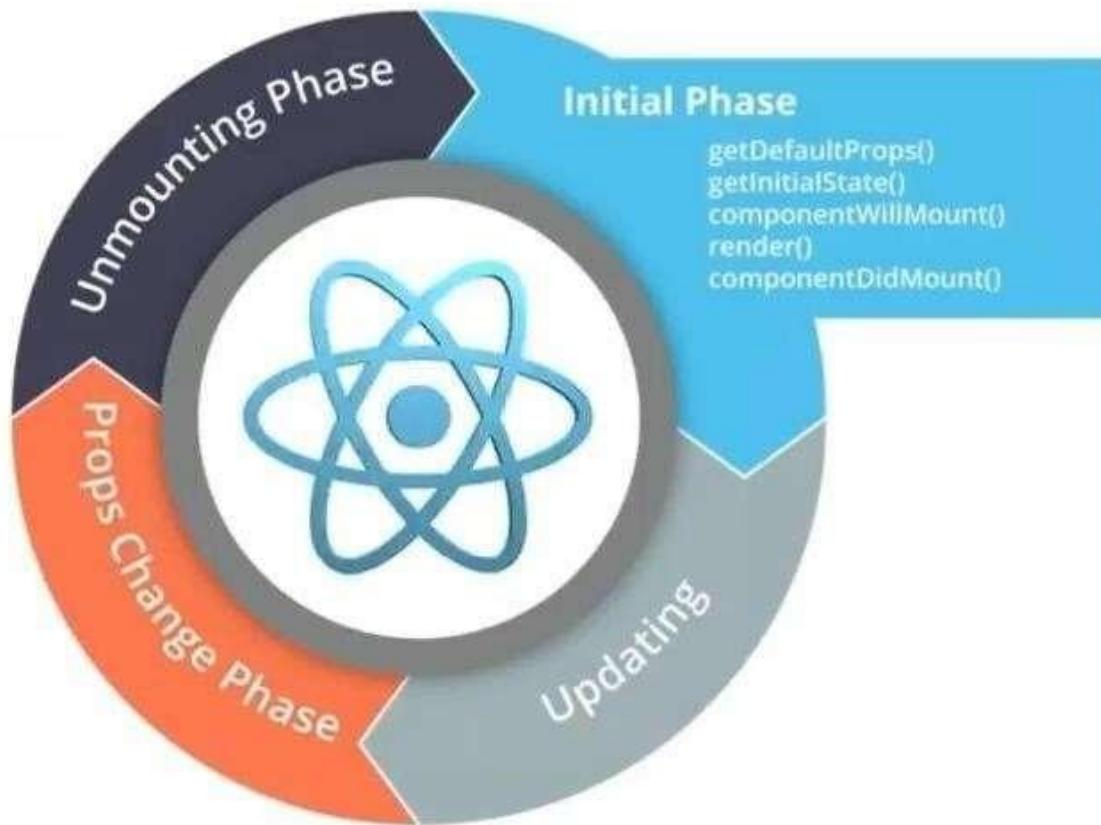
LifeCycle

React provides various methods which notifies when certain stage of the lifecycle occurs called Lifecycle methods

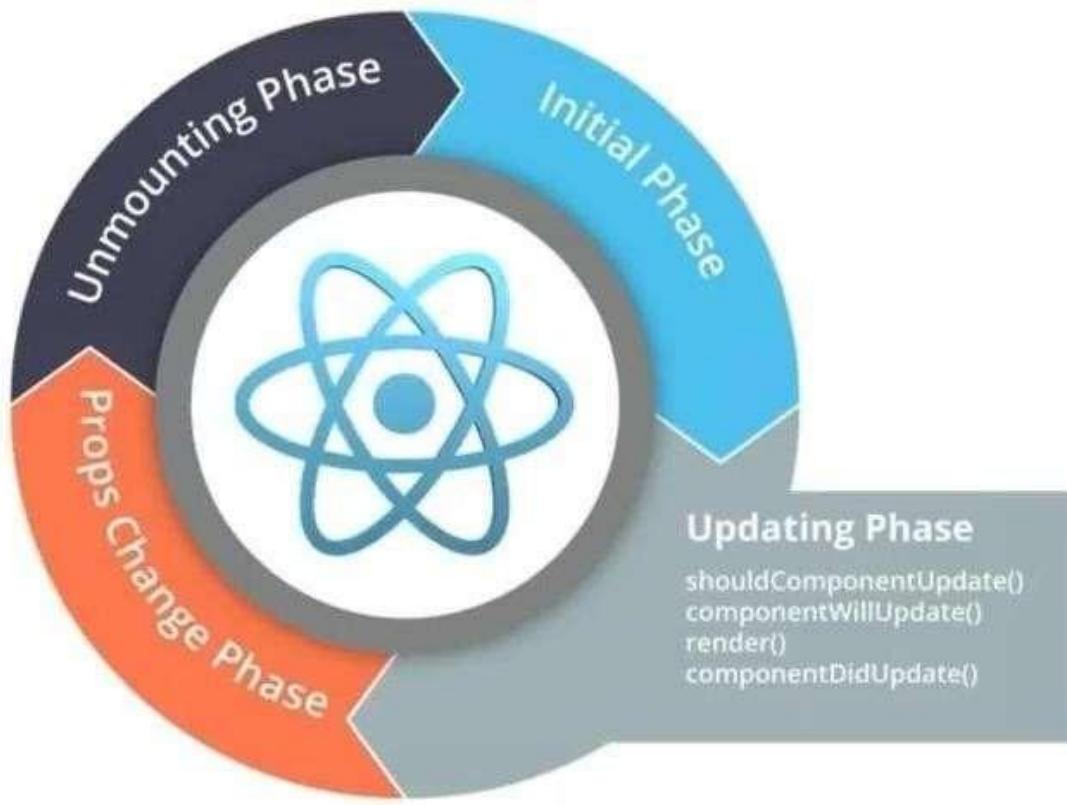
- ✓ These are special event handlers that are called at various points in components life
- ✓ Code can be added to them to perform various tasks
- ✓ They are invoked in a predictable order
- ✓ Entire lifecycle is divided into 4 phases.



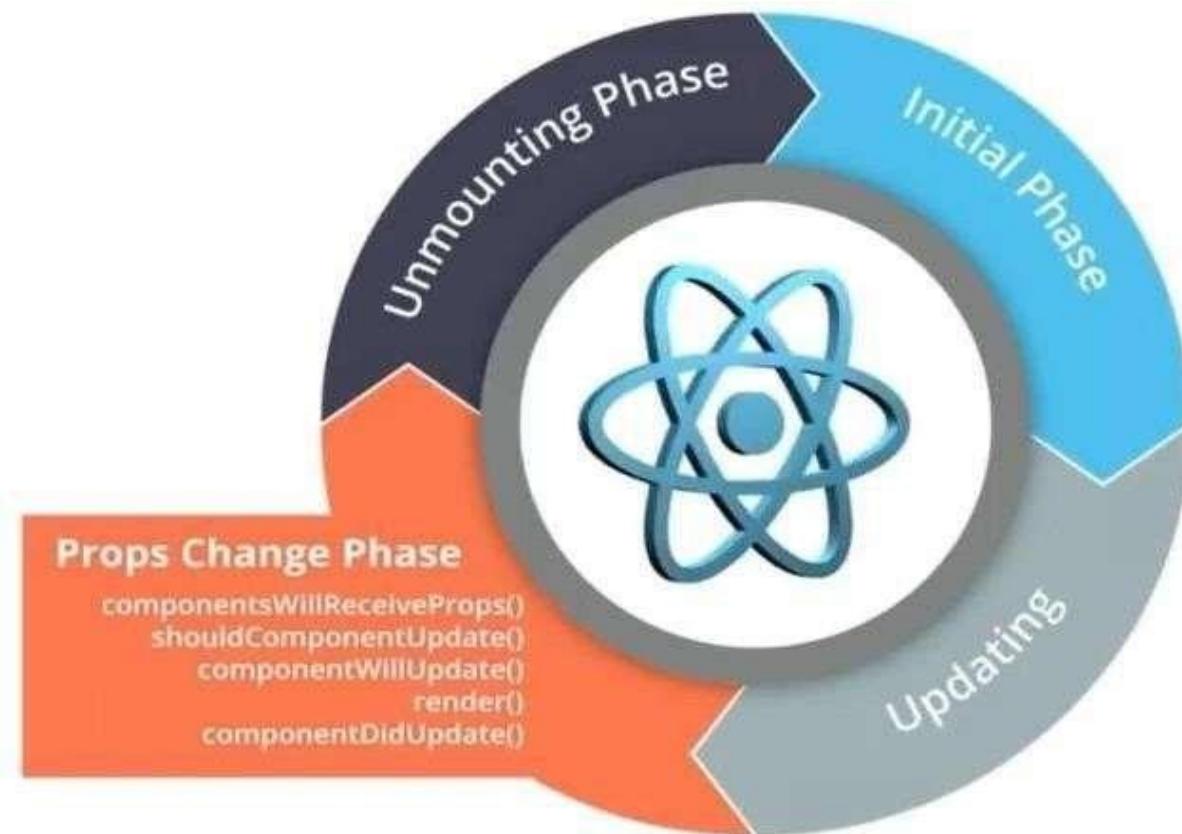
Initial Phase



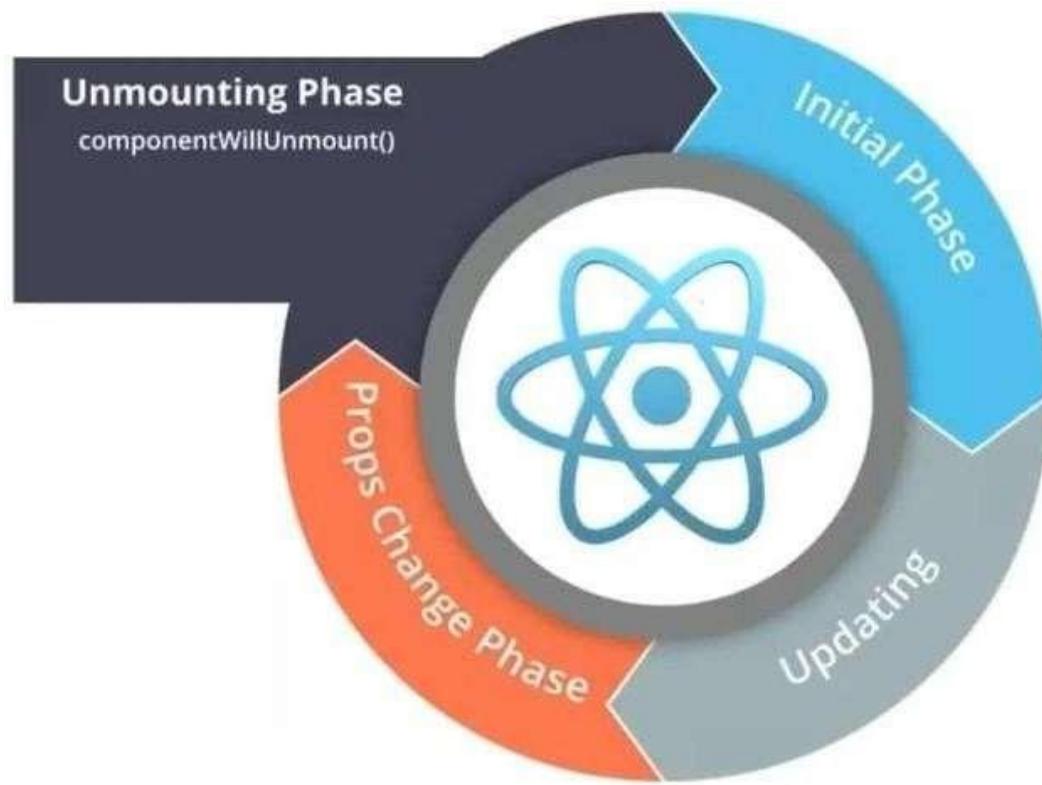
UpdatingPhase



Props Change Phase



Unmourning Phase



Component Life Cycle

Lifecycle Methods

What : We have seen so far that React web apps are actually a collection of independent components which run according to the interactions made with them. Every React Component has a lifecycle of its own, lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence. The definition is pretty straightforward but what do we mean by different stages? A React Component can go through four stages of its life as follows.

Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.

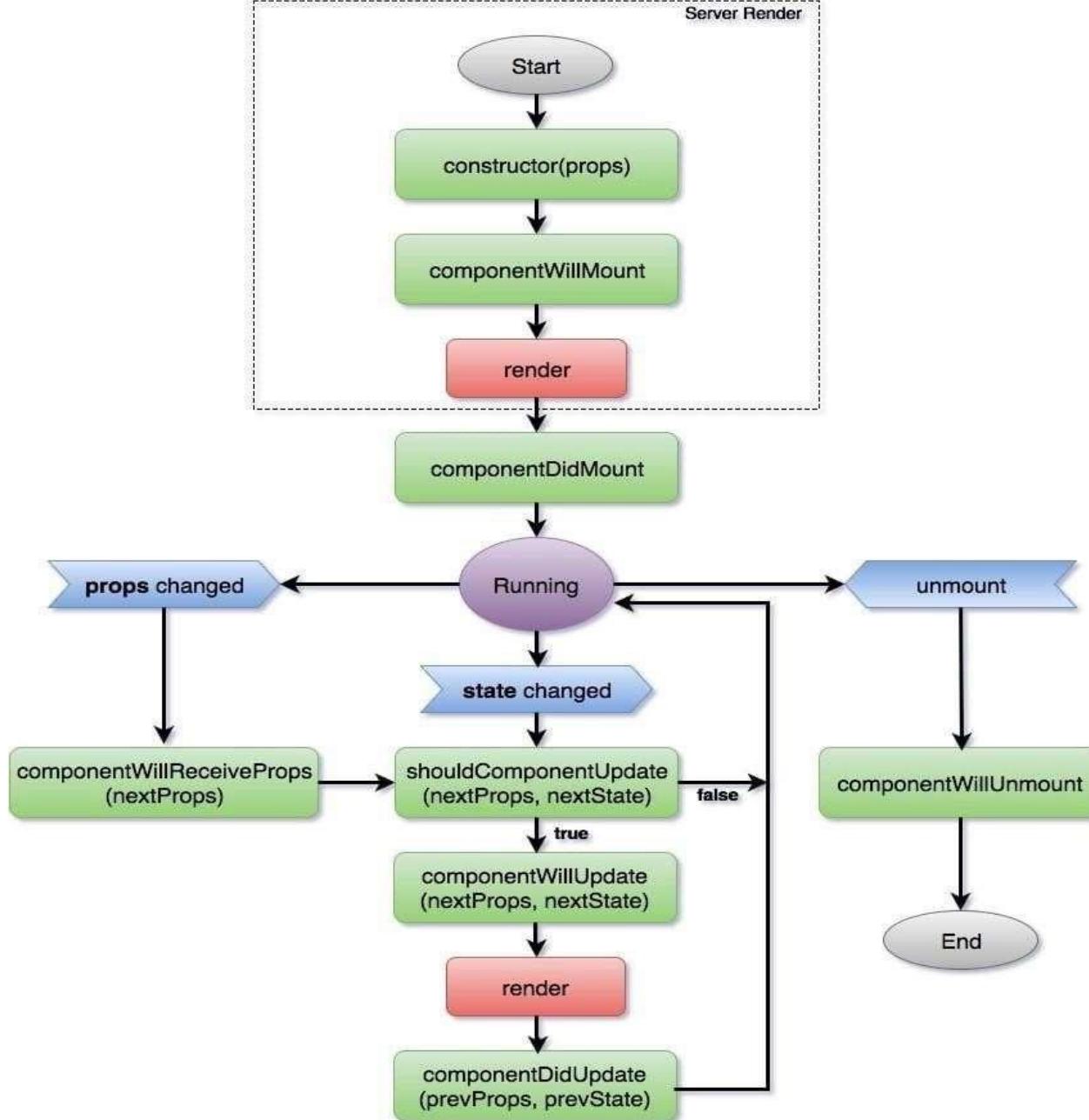
Updating: Updating is the stage when the state of a component is updated and the application is repainted.

Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

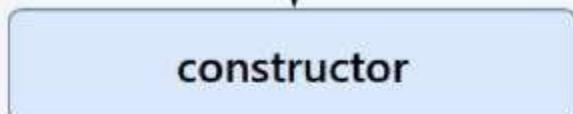
Component Life Cycle

React provides the developers a set of predefined functions that if present is invoked around specific events in the lifetime of the component. Developers are supposed to override the functions with desired logic to execute accordingly. We have illustrated the gist in the following diagram.



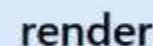


Mounting

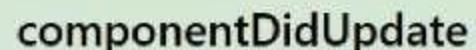
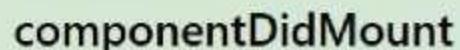


Updating

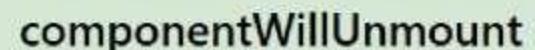
New props `setState()` `forceUpdate()`



React updates DOM and refs



Unmounting



Component Life Cycle

1. Initialization: In this phase the developer has to define the props and initial state of the component this is generally done in the constructor of the component. The following code snippet describes the initialization process.

```
class Clock extends React.Component {  
  constructor(props) {  
    // Calling the constructor of  
    // Parent Class React.Component  
    super(props);  
    // Setting the initial state  
    this.state = { date : new Date() };  
  }  
}
```

Component Life Cycle

2. Mounting: Mounting is the phase of the component lifecycle when the initialization of the component is completed and the component is mounted on the DOM and rendered for the first time in the webpage. Now React follows a default procedure in the Naming Conventions of this predefined functions where the functions containing “Will” represents before some specific phase and “Did” represents after the completion of that phase. Mounting phase consists of two such predefined functions as described below.

componentWillMount() Function: As the name clearly suggests, this function is invoked right before the component is mounted on the DOM i.e. this function gets invoked once before the render() function is executed for the first time.

componentDidMount() Function: Similarly as the previous one this function is invoked right after the component is mounted on the DOM i.e. this function gets invoked once after the render() function is executed for the first time.

Component Life Cycle

3. Updation: React is a JS library that helps create Active web pages easily. Now active web pages are specific pages that behave according to its user. For example, let's take the GeeksforGeeks {IDE} webpage, the webpage acts differently with each user. User A might write some code in C in the Light Theme while another User may write a Python code in the Dark Theme all at the same time. This dynamic behavior that partially depends upon the user itself makes the webpage an Active webpage. Now how can this be related to Updation? Updation is the phase where the states and props of a component are updated followed by some user events such as clicking, pressing a key on keyboard etc. The following are the descriptions of functions that are invoked at different points of Updation phase.

Component Life Cycle

componentWillReceiveProps() Function: This is a Props exclusive Function and is independent of States. This function is invoked before a mounted component gets its props reassigned. The function is passed the new set of Props which may or may not be identical to the original Props. Thus checking is a mandatory step in this regards. The following code snippet shows a sample use-case.

```
componentWillReceiveProps(newProps)
{
  if (this.props !== newProps) {
    console.log(" New Props have been assigned ");
    // Use this.setState() to rerender the page.
  }
}
```

Component Life Cycle

setState() Function: This is not particularly a Lifecycle function and can be invoked explicitly at any instant. This function is used to update the State of a component. You may refer to this article for detailed information.

shouldComponentUpdate() Function: By default, every state or props update re-render the page but this may not always be the desired outcome, sometimes it is desired that on updating the page will not be repainted. The shouldComponentUpdate() Function fulfills the requirement by letting React know whether the component's output will be affected by the update or not. shouldComponentUpdate() is invoked before rendering an already mounted component when new props or state are being received. If returned false then the subsequent steps of rendering will not be carried out. This function can't be used in case of forceUpdate(). The Function takes the new Props and new State as the arguments and returns whether to re-render or not.

Component Life Cycle

componentWillUpdate() Function: As the name clearly suggests, this function is invoked before the component is rerendered i.e. this function gets invoked once before the render() function is executed after the updation of State or Props.

componentDidUpdate() Function: Similarly this function is invoked after the component is rerendered i.e. this function gets invoked once after the render() function is executed after the updation of State or Props.

Component Life Cycle

Unmounting: This is the final phase of the lifecycle of the component that is the phase of unmounting the component from the DOM. The following function is the sole member of this phase.

- **componentWillUnmount() Function:** This function is invoked before the component is finally unmounted from the DOM i.e. this function gets invoked once before the component is removed from the page and this denotes the end of the lifecycle.

We have so far discussed every predefined function there was in the lifecycle of the component and we have also specified the order of execution of the function. Let us now see one final example to finish off the article while revising what's discussed above.

Example : [Link](#)

Component Life Cycle

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.3.1indexComponentLife.js>

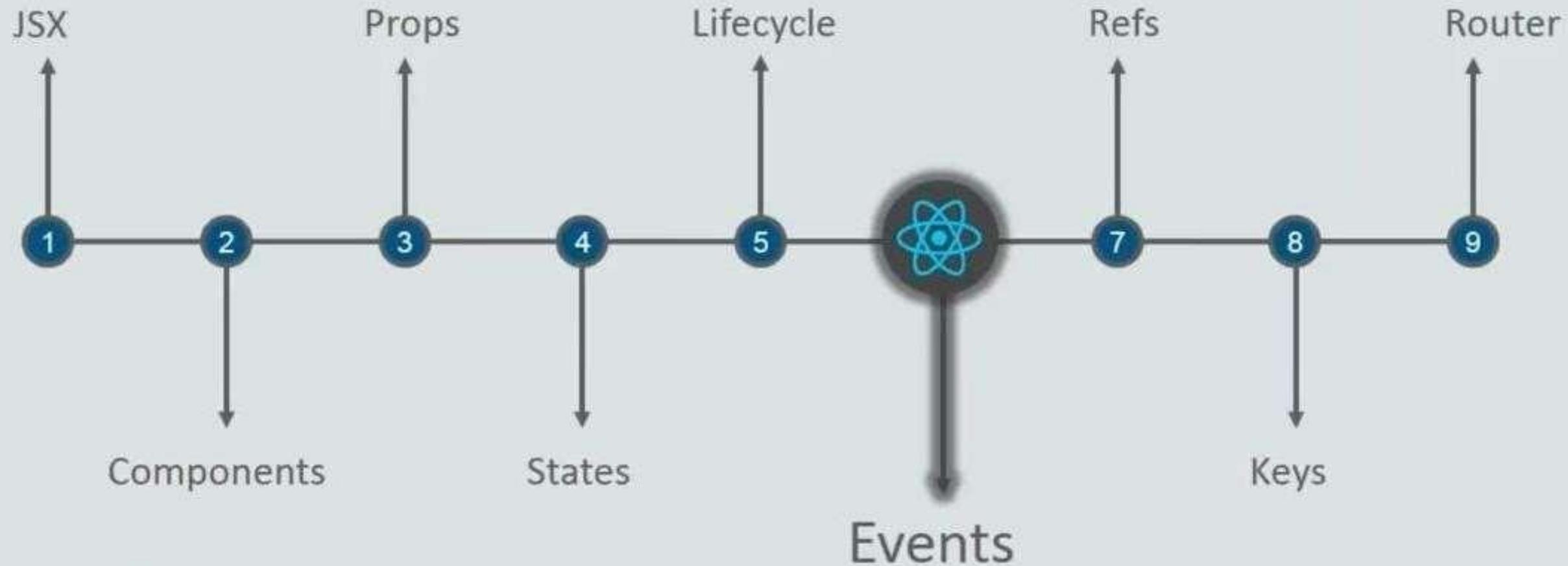
Example 2:

<https://github.com/TopsCode/React/blob/master/Module3/3.3.2indexComponentLifeCycle.js>

Component

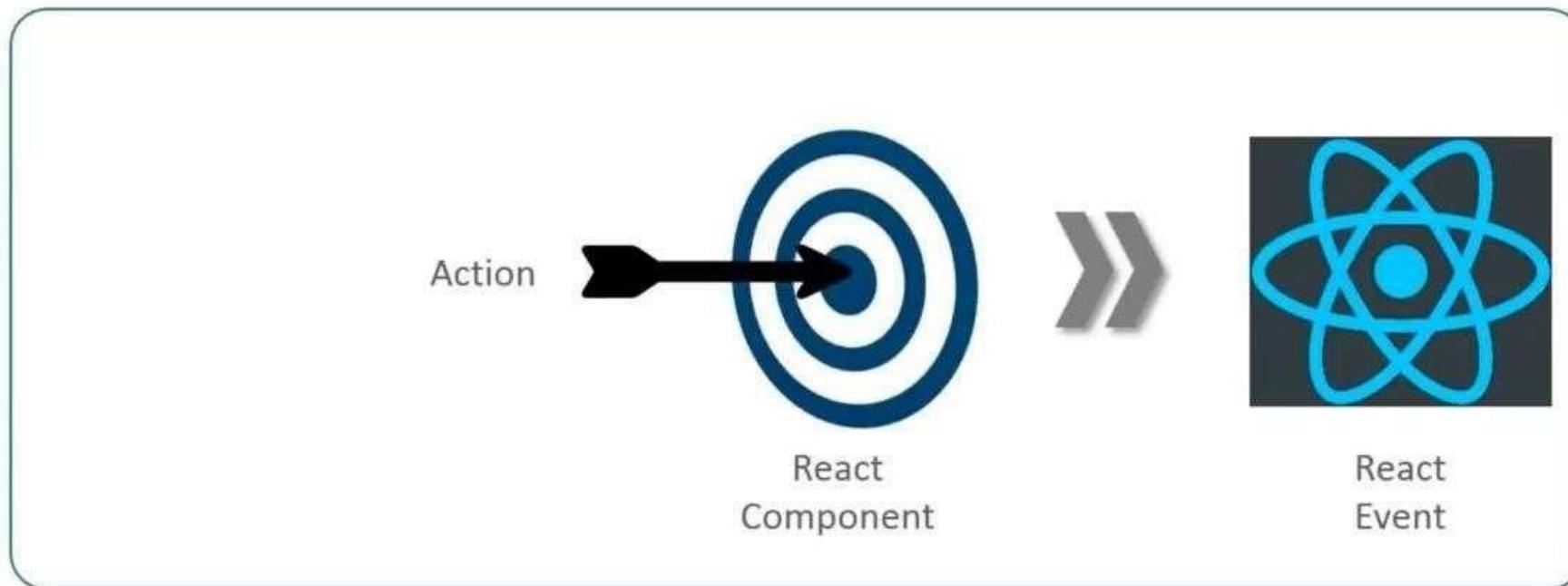
Example :

[https://github.com/TopsCode/React/blob/master/Module3/
3.3.5indexButtonClickConsole.js](https://github.com/TopsCode/React/blob/master/Module3/3.3.5indexButtonClickConsole.js)



Events

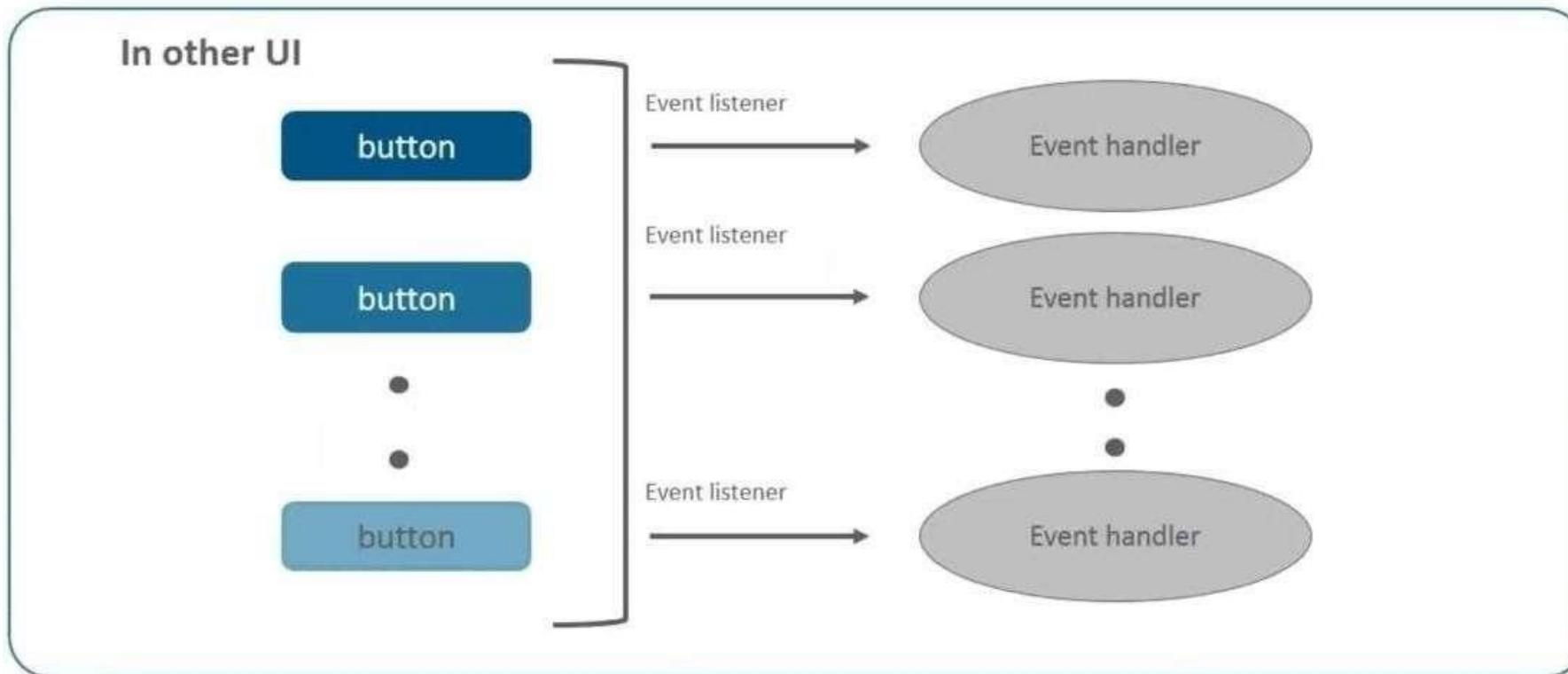
Events are the triggered reactions to specific actions like mouse hover, mouse click, key press etc.



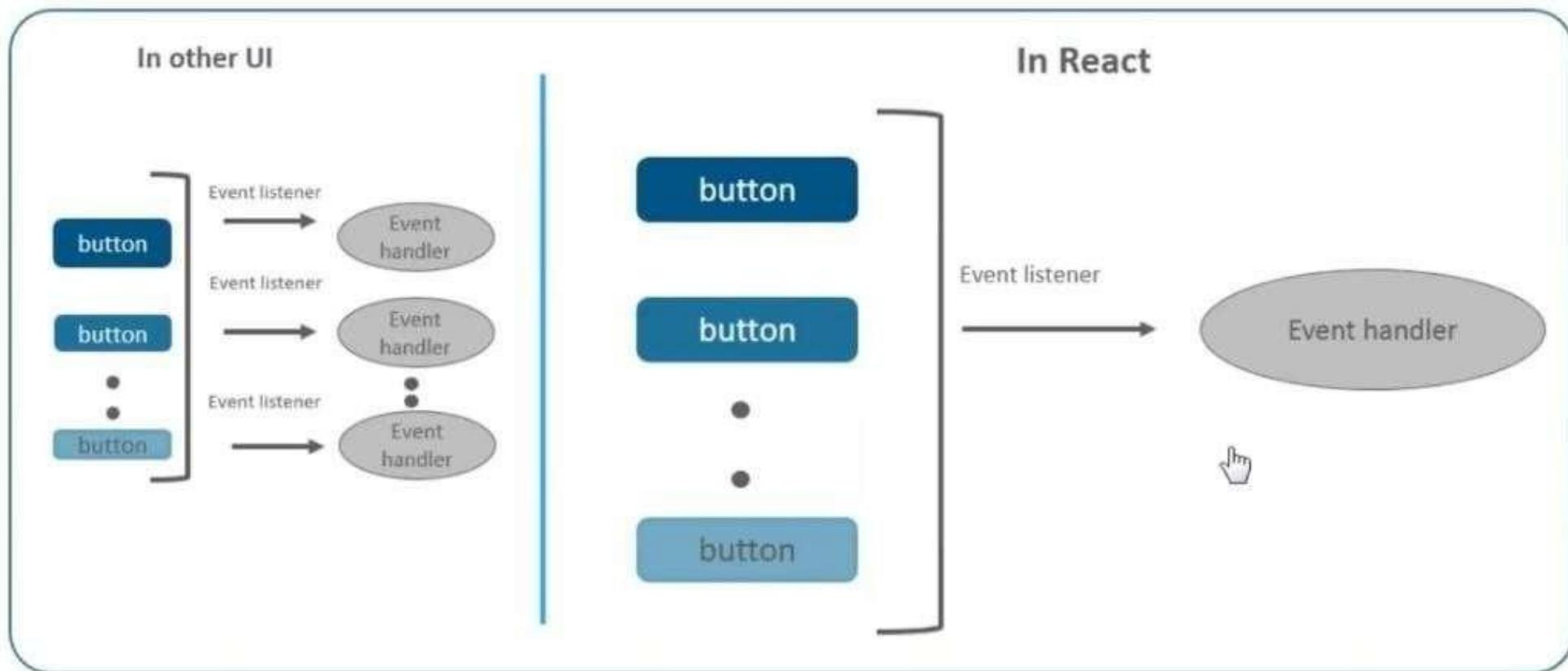
Events

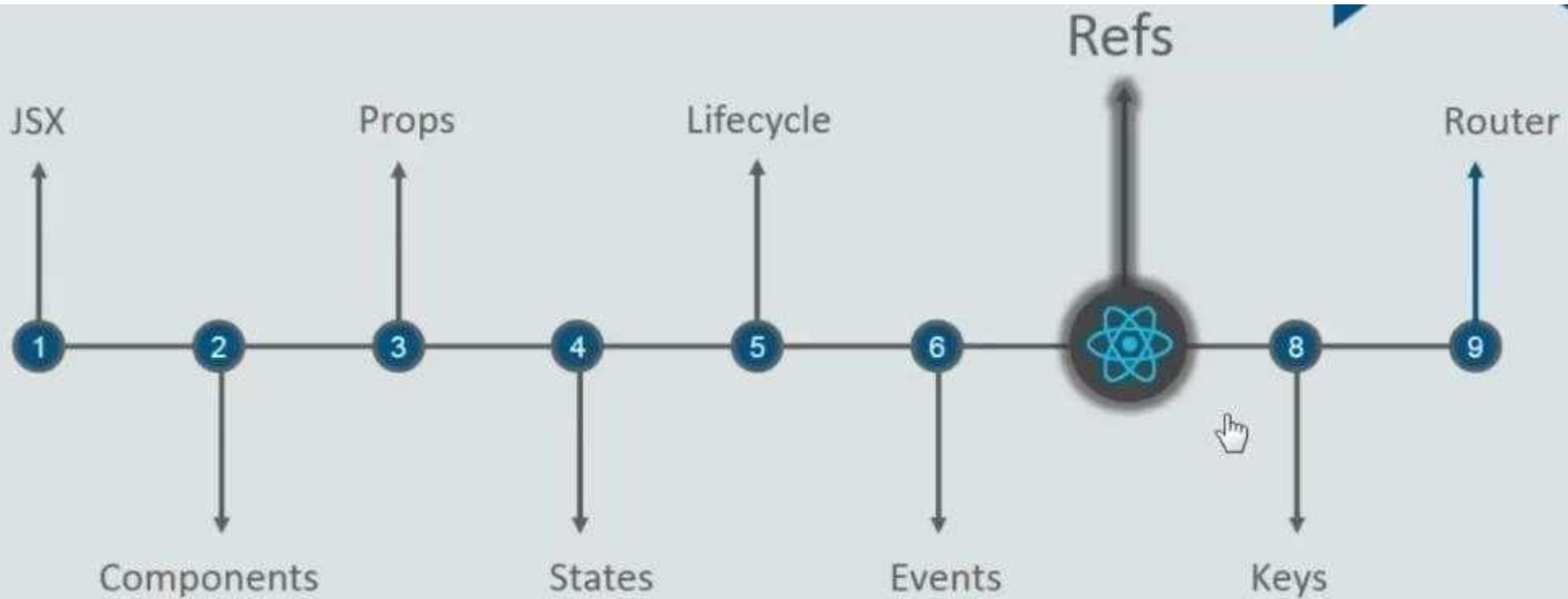
- ✓ Events pass event arguments to the event handler
- ✓ Whenever an event is specified in JSX, a synthetic event is generated
- ✓ This synthetic event wraps up the browser's native event and are passed as argument to the event handler

Events



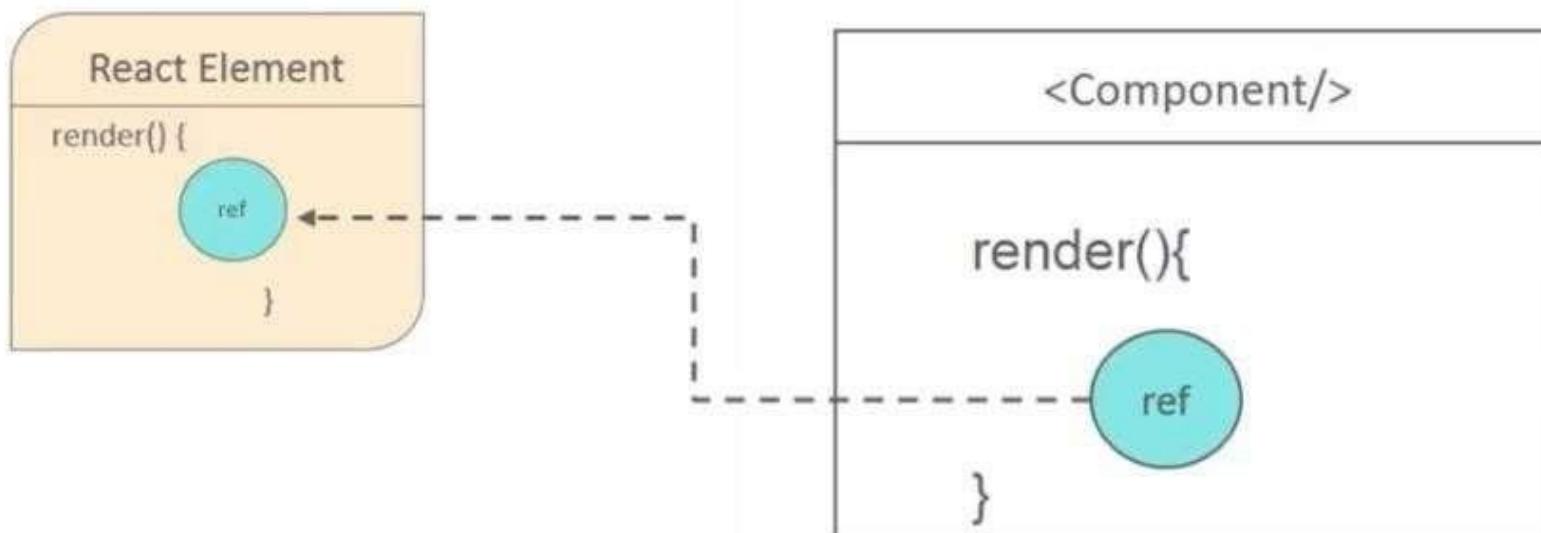
Events





Refs

- ✓ Refs stands for references
- ✓ Refs are used to return references to a particular element or component returned by render().



Create ref Example

```
var RefComponent=  
  React.createClass({ callRefFunction:function(){  
    var name = this.inputDemo.value;  
    document.getElementById('dispalyNameSpan').innerHTML  
    L = name;  
  },  
  render:  
  function(){ return(  
    <div>  
      <h2>Name : <input type="text" ref = { inputRef =>  
this.inputDemo = inputRef } /></h2><br/>  
      <button onClick={this.callRefFunction} >Click</button>  
      <h2>Hello, <span id="dispalyNameSpan"></span></h2>  
    </div>  
  ) } );  
};  
  
ReactDOM.render(  
  <RefComponent />,document.getElementById('root')  
);
```

Output



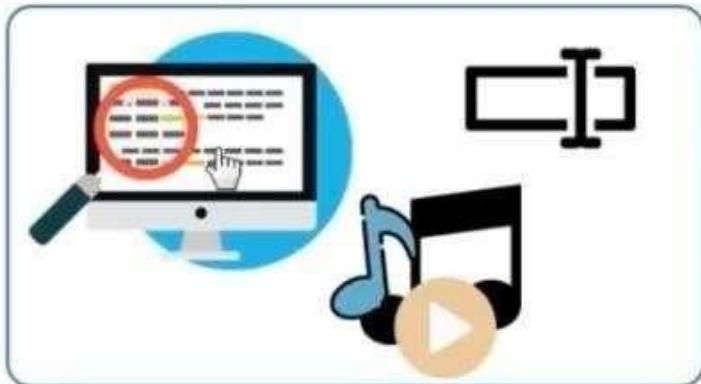
Name :

Click

Hello, Jay Amin

Refs Use Cases

- Managing focus, text selection or media playback

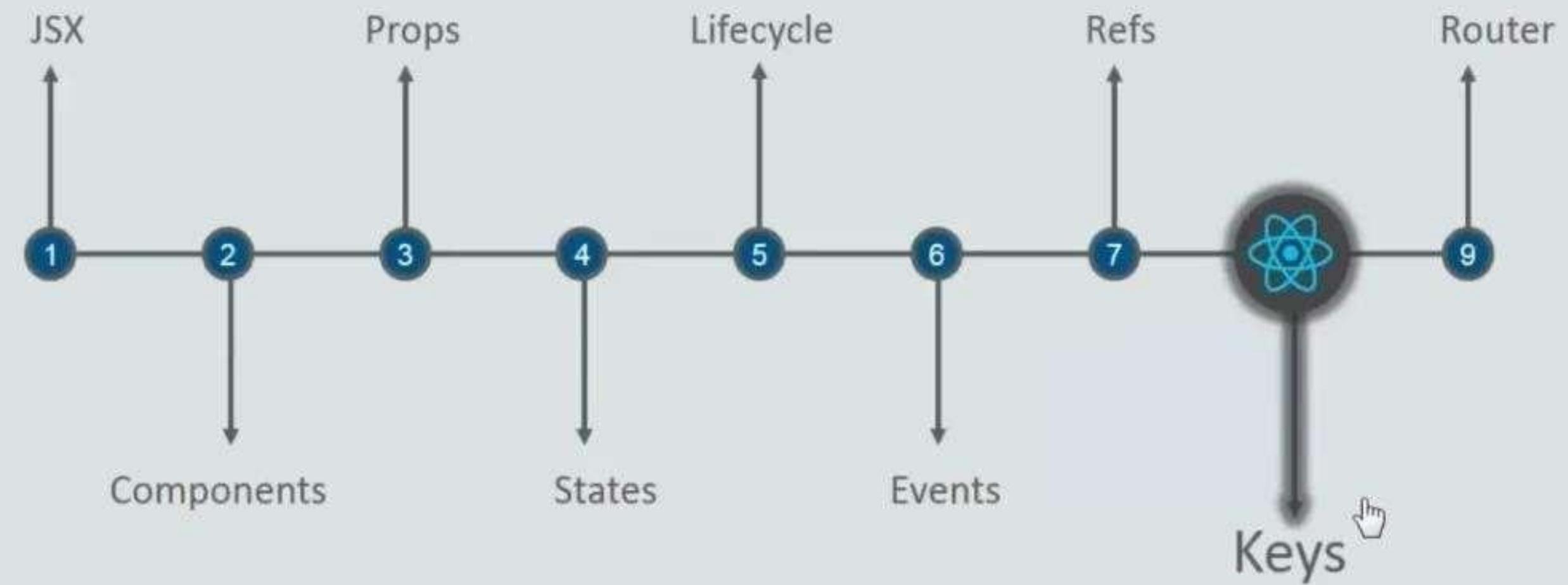


- Triggering imperative animations



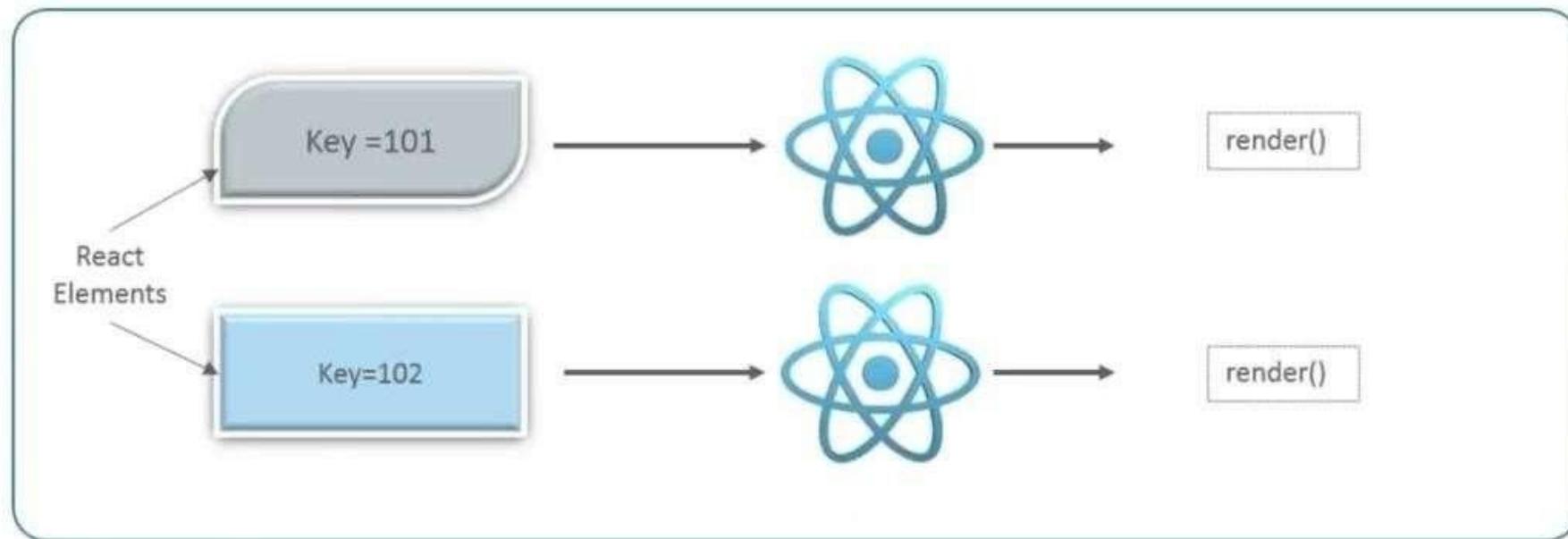
- Integrating with third-party DOM libraries





Keys

Keys are the elements which helps React to identify components uniquely



Lists and Keys

Basic List Component

Usually you would render lists inside a component.

We can refactor the previous example into a component that accepts an array of numbers and outputs a list of elements.

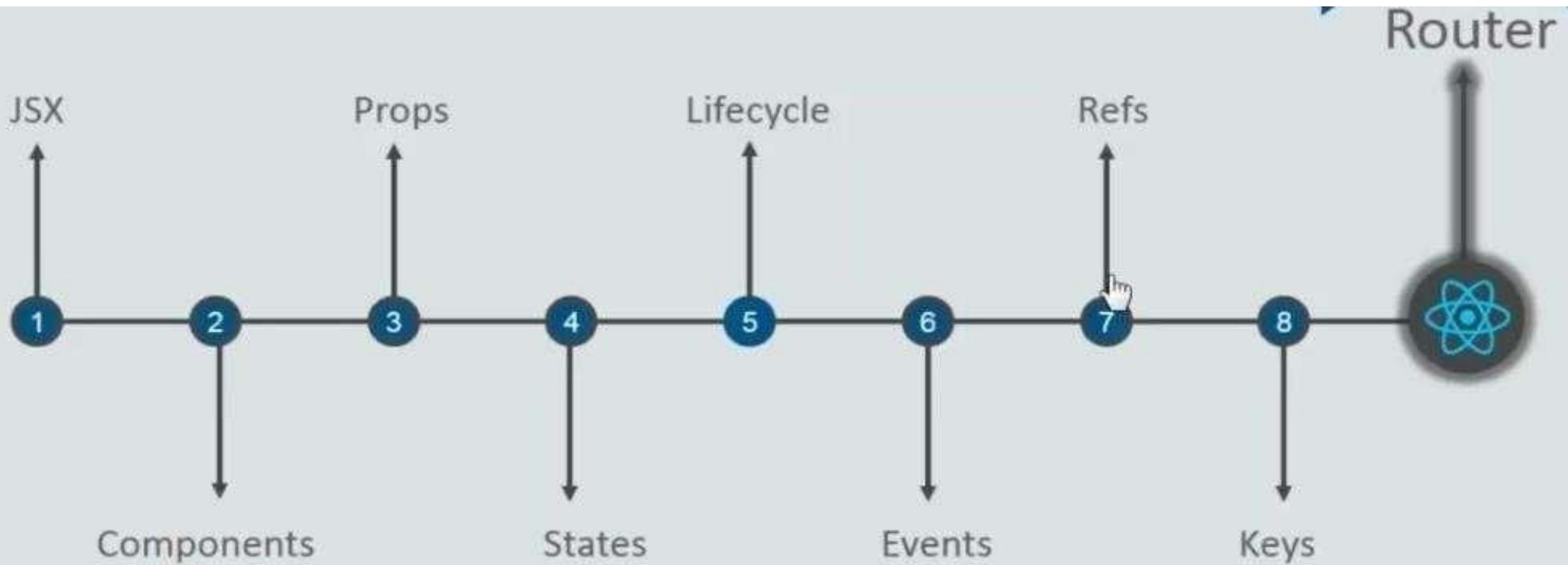
Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.4.5indexArray.js>

If Condition Example:

<https://github.com/TopsCode/React/blob/master/Module3/3.4.6indexIfCondition.js>





Function and Class Components

```
import ReactDOM from 'react-dom';
import React, { Component } from "react";
function Welcome(props) {
  return<h1>Hello, {props.name}</h1>;
}
ReactDOM.render( <Welcome/>, document.getElementById("root"));
ReactDOM.render( <Welcome name='test'/>, document.getElementById("root"));
```



Handling Events

Handling events with React elements is very similar to handling events on DOM elements. There are some syntactic differences:

React events are named using camelCase, rather than lowercase.

With JSX you pass a function as the event handler, rather than a string.

For example, the HTML:

```
<button onclick="activateLasers()">  
    Activate Lasers  
</button>
```

For example, the React:

```
<button onClick={activateLasers}>  
    Activate Lasers  
</button>
```

```
<a href="#" onclick="console.log('The link was clicked.');" return false">  
Click me </a>
```

Handling Events

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.4.8index6HideShowContentButtonClick.js>

Conditional Rendering

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the Conditional Operator to create elements representing the current state, and let React update the UI to match them.

Consider these two components:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props)
{ return <h1>Please sign
  up.</h1>;
}
```

Conditional Rendering

We'll create a Greeting component that displays either of these components depending on whether a user is logged in:

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.4.7index5InelfwithLogical%26%26Operator.js>

Controlled Components

In HTML, form elements such as `<input>`, `<textarea>`, and `<select>` typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the `state` property of components, and only updated with `setState()`.

We can combine the two by making the React state be the “single source of truth”. Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a “controlled component”.

For example, if we want to make the previous example log the name when it is submitted, we can write the form as a controlled component:

Composition vs Inheritance

React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.

In this section, we will consider a few problems where developers new to React often reach for inheritance, and show how we can solve them with composition.

Containment

Some components don't know their children ahead of time. This is especially common for components like Sidebar or Dialog that represent generic "boxes".

We recommend that such components use the special `children` prop to pass children elements directly into their output:

Composition vs. Inheritance

React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.

In this section, we will consider a few problems where developers new to React often reach for inheritance, and show how we can solve them with composition.

Containment

Some components don't know their children ahead of time. This is especially common for components like Sidebar or Dialog that represent generic "boxes".

We recommend that such components use the special `children` prop to pass children elements directly into their output:

Composition vs. Inheritance

Specialization

Sometimes we think about components as being “special cases” of other components. For example, we might say that a WelcomeDialog is a special case of Dialog.

In React, this is also achieved by composition, where a more “specific” component renders a more “generic” one and configures it with props:

Composition vs. Inheritance Specialization

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.6.11indexCompositionvsInheritance.js>

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.6.10indexConditionalRendering.js>

So What About Inheritance?

At Facebook, we use React in thousands of components, and we haven't found any use cases where we would recommend creating component inheritance hierarchies.

Props and composition give you all the flexibility you need to customize a component's look and behavior in an explicit and safe way. Remember that components may accept arbitrary props, including primitive values, React elements, or functions.

If you want to reuse non-UI functionality between components, we suggest extracting it into a separate JavaScript module. The components may import it and use that function, object, or a class, without extending it.

Module-12

JSON-server and Firebase Real Time Database

What is json server in React js

The JSON Server helps frontend developers simulate all types of HTTP requests (GET, POST, PATCH, DELETE) to the mock server. This aids in developing interactive applications that need to fetch data, create new data, update existing data, or delete data from the server.

Intsall json server

npm install json-server

Create a db.json

```
{  
  "posts": [  
    { "id": "1", "title": "a title", "views": 100 },  
    { "id": "2", "title": "another title", "views": 200 }  
  ],  
  "comments": [  
    { "id": "1", "text": "a comment about post 1", "postId": "1" },  
    { "id": "2", "text": "another comment about post 1", "postId": "1" }  
  ]  
}
```

```
"profile": {  
  "name": "typicode"  
}  
}
```

Pass it to JSON Server CLI

```
$ npx json-server db.json
```

```
{  
  "id": "1",  
  "title": "a title",  
  "views": 100  
}
```

Step 1: Break The UI Into A Component Hierarchy

The first thing you'll want to do is to draw boxes around every component (and subcomponent) in the mock and give them all names. If you're working with a designer, they may have already done this, so go talk to them! Their Photoshop layer names may end up being the names of your React components!

But how do you know what should be its own component? Just use the same techniques for deciding if you should create a new function or object. One such technique is the single responsibility principle, that is, a component should ideally only do one thing. If it ends up growing, it should be decomposed into smaller subcomponents.

Since you're often displaying a JSON data model to a user, you'll find that if your model was built correctly, your UI (and therefore your component structure) will map nicely. That's because UI and data models tend to adhere to the same information architecture, which means the work of separating your UI into components is often trivial. Just break it up into components that represent exactly one piece of your data model.

JSON Data in React Component

Step 2: Build A Static Version in React

Now that you have your component hierarchy, it's time to implement your app. The easiest way is to build a version that takes your data model and renders the UI but has no interactivity. It's best to decouple these processes because building a static version requires a lot of typing and no thinking, and adding interactivity requires a lot of thinking and not a lot of typing. We'll see why.

To build a static version of your app that renders your data model, you'll want to build components that reuse other components and pass data using props. Props are a way of passing data from parent to child. If you're familiar with the concept of state, don't use state at all to build this static version. State is reserved only for interactivity, that is, data that changes over time. Since this is a static version of the app, you don't need it.

You can build top-down or bottom-up. That is, you can either start with building the components higher up in the hierarchy (i.e. starting with `FilterableProductTable`) or with the ones lower in it (`ProductRow`). In simpler examples, it's usually easier to go top-down, and on larger projects, it's easier to go bottom-up and write tests as you build.

JSON Data in React Component

At the end of this step, you'll have a library of reusable components that render your data model. The components will only have `render()` methodssince this is a static version of your app. The component at the top of the hierarchy (`FilterableProductTable`) will take your data model as a prop. If you make a change to yourunderlying datamodel and call `ReactDOM.render()` again, the UI will be updated. It's easy to see how your UI is updated and where to make changes since there's nothing complicated going on. React's one-way data flow (also calledone-way binding) keeps everything modularand fast.

Simply refer to the React docs if you need help executing this step.

JSON Data in React Component

Step 3: Identify The Minimal (but complete) Representation Of UI State

To make your UI interactive, you need to be able to trigger changes to your underlying data model. React makes this easy with state.

To build your app correctly, you first need to think of the minimal set of mutable state that your app needs. The key here is DRY: Don't Repeat Yourself. Figure out the absolute minimal representation of the state your application needs and compute everything else you need on-demand. For example, if you're building a TODO list, just keep an array of the TODO items around; don't keep a separate state variable for the count. Instead, when you want to render the TODO count, simply take the length of the TODO items array.

JSON Data in React Component

Think of all of the pieces of data in our example application. We have:

- The original list of products
- The search text the user has entered
- The value of the checkbox
- The filtered list of products

Let's go through each one and figure out which one is state. Simply ask three questions about each piece of data:

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

JSON Data in React Component

The original list of products is passed in as props, so that's not state. The search text and the checkbox seem to be state since they change over time and can't be computed from anything. And finally, the filtered list of products isn't state because it can be computed by combining the original list of products with the search text and value of the checkbox.

So finally, our state is:

- The search text the user has entered
- The value of the checkbox

JSON Data in React Component

OK, so we've identified what the minimal set of app state is. Next, we need to identify which component mutates, or *owns*, this state.

Remember: React is all about one-way data flow down the component hierarchy. It may not be immediately clear which component should own what state. This is often the most challenging part for newcomers to understand, so follow these steps to figure it out:

For each piece of state in your application:

JSON Data in React Component

For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

JSON Data in React Component

Let's run through this strategy for our application:

- ProductTable needs to filter the product list based on state and SearchBar needs to display the search text and checked state.
- The common owner component is FilterableProductTable.
- It conceptually makes sense for the filter text and checked value to live in FilterableProductTable.

Cool, so we've decided that our state lives in FilterableProductTable. First, add an instance property `this.state = {filterText: "", inStockOnly: false}` to FilterableProductTable's constructor to reflect the initial state of your application. Then, pass filterText and inStockOnly to ProductTable and SearchBar as a prop. Finally, use these props to filter the rows in ProductTable and set the values of the form fields in SearchBar. You can start seeing how your application will behave: set filterText to "ball" and refresh your app. You'll see that the data table is updated correctly.

JSON Data in React Component

Example :

<https://github.com/TopsCode/React/blob/master/Module3/3.6.12indexJSONDatainReactComponent.js>

Authentication with firebase

Implementing User Login and SignUp with ReactJS and Firebase:

1. Setting up a ReactJS project and installing the necessary dependencies.
2. Creating a Firebase project and configuring Firebase Authentication.
3. Building the login functionality, including form validation and error handling.
4. Implementing the user signup feature, allowing new users to register.
5. Integrating Bootstrap for visually appealing and responsive designs.

Install all dependencies for **Login | Register**

npm install bootstrap

npm install firebase

npm install react-bootstrap

npm install react-google-button

npm install react-router-dom

npm install web-vitals

web-vitals: web-vitals is a package that helps measure and track essential web performance metrics, such as page load time, interactivity, and content rendering. Monitoring web vitals allows you to optimize your application's performance and ensure a smooth user experience.

Creating a Firebase project and configuring Firebase Authentication. To create a Firebase project and configure Firebase Authentication for your ReactJS application, follow these steps:

1. Create a Firebase Project:

- Go to the Firebase console at <https://console.firebaseio.google.com/> and sign in with your Google account.
- Click on the “Add project” button to create a new Firebase project.
- Provide a name for your project and select your preferred region.
- Click on the “Create project” button to create your Firebase project.

2. Enable Firebase Authentication:

- Once your Firebase project is created, you’ll be redirected to the project dashboard.
- In the left sidebar, under the “Develop” section, click on the “Authentication”

option.

- On the Authentication page, select the “Get Started” button under the “Sign-in method” tab.
- Choose the authentication providers — Email/Password and Google, and enable them.

3. Obtain Firebase Configuration Details:

- On the Firebase project dashboard, click on the “Project settings” gear icon located next to “Project Overview” in the top-left corner.
- In the “General” tab, scroll down to the section titled “Your apps” and click on the “</>” icon to add a new web app to your project.
- Provide a nickname for your app and make sure to leave the “Also set up Firebase Hosting for this app” checkbox unchecked.
- Click on the “Register app” button to proceed.

SDK snippet.

Connect ReactJS Application to Firebase:

Src=>firebase.js

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
import {getAuth} from "firebase/auth";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
```

```
apiKey: "AIzaSyBR3rJ0jHYuhJNo_9c5ve1qMKQxoMEjagc",
authDomain: "login-register-app-6ab89.firebaseio.com",
projectId: "login-register-app-6ab89",
storageBucket: "login-register-app-6ab89.appspot.com",
messagingSenderId: "850064191007",
appId: "1:850064191007:web:4df7371d7c6d02a3446f2a",
measurementId: "G-53G920RJ9F"
};
```

```
// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export default app;
```

Building Context APIs:

In ReactJS, the Context API provides a way to pass data down through the component tree without the need to pass props explicitly at each level. The Context API consists of two main components: the Context provider and the Context consumer.

A Context provider is a React component that provides the data that needs to be shared with other components in the tree. It acts as a source of truth for the data and makes it accessible to any component that is a descendant of it. The provider is responsible for creating a Context and specifying the data it wants to share.

Creating Firebase functions:

Now we will create LogIn, SignUp, LogOut, and googleSignIn functions. Here we will use the Auth instance we created in the `src=>context=>UserAuthContext.js` file. `src=>Context=>UserAuthContext.js`

```
import { createContext, useContext, useEffect, useState } from "react";
import {
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  onAuthStateChanged,
  signOut,
  GoogleAuthProvider,
  signInWithPopup,
} from "firebase/auth";
import { auth } from "../firebase";

const userAuthContext = createContext();

export function UserAuthContextProvider({ children }) {
  const [user, setUser] = useState({});
```

```
function logIn(email, password) {
  return signInWithEmailAndPassword(auth, email, password);
}
function signUp(email, password) {
  return createUserWithEmailAndPassword(auth, email, password);
}
function logOut() {
  return signOut(auth);
}
function googleSignIn() {
  const googleAuthProvider = new GoogleAuthProvider();
  return signInWithPopup(auth, googleAuthProvider);
}
useEffect(() => {
```

```
const unsubscribe = onAuthStateChanged(auth, (currentuser) => {
  console.log("Auth", currentuser);
  setUser(currentuser);
});

return () => {
  unsubscribe();
};

}, []);

return (
  <userAuthContext.Provider
    value={{ user, logIn, signUp, logOut, googleSignIn }}
  >
  {children}
  </userAuthContext.Provider>
```

```
 );
}

export function useUserAuth() {
  return useContext(userAuthContext);
}
```

Wrap the App with the Provider: src=>(App.js)

```
import { Container, Row, Col } from "react-bootstrap";
import { Routes, Route } from "react-router-dom";
import "./App.css";
import Home from "./components/Home";
import Login from "./components/Login";
import Signup from "./components/Signup";
```

```
import ProtectedRoute from "./components/ProtectedRoute";
import { UserAuthContextProvider } from "./context/UserAuthContext";

function App() {
  return (
    <Container style={{ width: "400px" }}>
      <Row>
        <Col>
          <UserAuthContextProvider>
            <Routes>
              <Route
                path="/home"
                element={
                  <ProtectedRoute>
                    <Home />
                }
              </Route>
            </Routes>
          </UserAuthContextProvider>
        </Col>
      </Row>
    </Container>
  );
}

export default App;
```

```
        </ProtectedRoute>
    }
  />
  <Route path="/" element={<Login />} />
  <Route path="/signup" element={<Signup />} />
</Routes>
</UserAuthContextProvider>
</Col>
</Row>
</Container>
);
}

export default App;
```

Creating components for register | Login and Home and Protected Route:
src=>components=>Signup.jsx

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { Form, Alert } from "react-bootstrap";
import { Button } from "react-bootstrap";
import { useUserAuth } from "../context/UserAuthContext";

const Signup = () => {
  const [email, setEmail] = useState("");
  const [error, setError] = useState("");
  const [password, setPassword] = useState("");
```

```
const { signUp } = useUserAuth();
let navigate = useNavigate();

const handleSubmit = async (e) => {
  e.preventDefault();
  setError("");
  try {
    await signUp(email, password);
    navigate("/");
  } catch (err) {
    setError(err.message);
  }
};

return (
  <>
```

```
<div className="p-4 box">
  <h2 className="mb-3">Firebase/ React Auth Signup</h2>

  {error && <Alert variant="danger">{error}</Alert>}

  <Form onSubmit={handleSubmit}>

    <Form.Group className="mb-3" controlId="formBasicEmail">
      <Form.Control
        type="email"
        placeholder="Email address"
        onChange={(e) => setEmail(e.target.value)}>
      />
    </Form.Group>

    <Form.Group className="mb-3" controlId="formBasicPassword">
```

```
<Form.Control  
  type="password"  
  placeholder="Password"  
  onChange={(e) => setPassword(e.target.value)}  
/>  
</Form.Group>  
  
<div className="d-grid gap-2">  
  <Button variant="primary" type="Submit">  
    Sign up  
  </Button>  
</div>  
</Form>  
</div>  
<div className="p-4 box mt-3 text-center">  
  Already have an account? <Link to="/">Log In</Link>
```

```
        </div>
      </>
    );
};

};
```

```
export default Signup;
```

```
src=>components=>Login.jsx
```

```
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { Form, Alert } from "react-bootstrap";
import { Button } from "react-bootstrap";
import GoogleButton from "react-google-button";
import { useUserAuth } from "../context/UserAuthContext";
```

```
const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const { logIn, googleSignIn } = useUserAuth();
  const navigate = useNavigate();
```

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError("");
  try {
    await logIn(email, password);
    navigate("/home");
  } catch (err) {
    setError(err.message);
```

```
};

const handleGoogleSignIn = async (e) => {
  e.preventDefault();
  try {
    await googleSignIn();
    navigate("/home");
  } catch (error) {
    console.log(error.message);
  }
};

return (
  <>
  <div className="p-4 box">
```

```
<h2 className="mb-3">Firebase/ React Auth Login</h2>

{error && <Alert variant="danger">{error}</Alert>}

<Form onSubmit={handleSubmit}>

  <Form.Group className="mb-3" controlId="formBasicEmail">
    <Form.Control
      type="email"
      placeholder="Email address"
      onChange={(e) => setEmail(e.target.value)}>
    />
  </Form.Group>

  <Form.Group className="mb-3" controlId="formBasicPassword">
    <Form.Control
```

```
    type="password"
    placeholder="Password"
    onChange={(e) => setPassword(e.target.value)}
/>
</Form.Group>

<div className="d-grid gap-2">
  <Button variant="primary" type="Submit">
    Log In
  </Button>
</div>
</Form>
<hr />
<div>
  <GoogleButton
    className="g-btn"
```

```
    type="dark"
    onClick={handleGoogleSignIn}
  />
</div>
</div>
<div className="p-4 box mt-3 text-center">
  Don't have an account? <Link to="/signup">Sign up</Link>
</div>
</>
);
};

export default Login;
src=>components=>Home.jsx
```

```
import React from "react";
import { Button } from "react-bootstrap";
import { useNavigate } from "react-router-dom";
import { useUserAuth } from "../context/UserAuthContext";

const Home = () => {
  const { logOut, user } = useUserAuth();
  const navigate = useNavigate();
  const handleLogout = async () => {
    try {
      await logOut();
      navigate("/");
    } catch (error) {
      console.log(error.message);
    }
  };
}
```

```
return (
  <>
  <div className="p-4 box mt-0 text-center">
    Welcome <br />
    {user && user.email}
  </div>
  <div className="d-grid gap-2">

    <Button variant="primary" onClick={handleLogout}>
      Log out
    </Button>

  </div>
</>
);
};
```

```
export default Home;
```

```
src=>components=>ProtectedRoute.jsx
```

```
import React from "react";
import { Navigate } from "react-router-dom";
import { useUserAuth } from "../context/UserAuthContext";
const ProtectedRoute = ({ children }) => {
  const { user } = useUserAuth();

  console.log("Check user in Private: ", user);
  if (!user) {
    return <Navigate to="/" />;
  }
}
```

```
}

return children;
};


```

```
export default ProtectedRoute;
```

ProtectedRoute

In ReactJS, a protected route refers to a route in your application that can only be accessed by authenticated users. It is a way to restrict access to certain parts of your application, ensuring that only users who have successfully logged in can view certain pages or components.

Applications.css

Src=>App.css

body {

```
background-color: #fafafa;  
display: flex;  
justify-content: center;  
align-content: center;  
width: 100%;  
align-items: center;  
height: 100vh;  
}
```

```
.box {  
border: 1px solid #dfdfdf;  
background-color: #fff;  
}
```

```
.g-btn {  
width: 100% !important;
```

```
height: 40px !important;  
line-height: 40px !important;  
font-size: 15px !important;  
margin-left: 20% !important;  
}  
.g-btn > div,  
.g-btn > div > svg {  
width: 40px !important;  
height: 38px !important;  
}
```

Accessibility

Why Accessibility?

Web accessibility (also referred to as [a11y](#)) is the design and creation of websites that can be used by everyone. Accessibility support is necessary to allow assistive technology to interpret web pages.

React fully supports building accessible websites, often by using standard HTML techniques.

Accessibility

Standards and Guidelines

WCAG

The Web Content Accessibility Guidelines provides guidelines for creating accessible web sites.

The following WCAG checklists provide an overview:

- WCAG checklist from Wuhcag
- WCAG checklist from WebAIM
- Checklist from The A11Y Project

Accessibility

WAI-ARIA

The Web Accessibility Initiative - Accessible Rich Internet Applications document contains techniques for building fully accessible JavaScript widgets.

Note that all aria-* HTML attributes are fully supported in JSX. Whereas most DOM properties and attributes in React are camelCased, these attributes should be hyphen-cased (also known as kebab-case, lisp-case, etc) as they are in plain HTML:

```
<input type="text"  
      aria-label={labelText}  
      aria-required="true"  
      onChange={onchangeHandler}  
      value={inputValue}  
      name="name"  
    />
```

Accessibility

Semantic HTML

Semantic HTML is the foundation of accessibility in a web application. Using the various HTML elements to reinforce the meaning of information in our websites will often give us accessibility for free.

- MDN HTML elements reference

Sometimes we break HTML semantics when we add `<div>` elements to our JSX to make our React code work, especially when working with lists (``, `` and `<dl>`) and the HTML `<table>`. In these cases we should rather use React Fragments to group together multiple elements.

Accessibility

```
class App extends React.Component
{ render() {
    return (
        <React.Fragment>
            <p>I would</p>
            <p>really like</p>
            <p>to render</p>
            <p>an array</p>
        </React.Fragment>
    );
}
}
```



Accessibility

Check this out – we write this just like we would the <div>-wrapper method, but it'll behave functionally equivalent to the array-render method, just with some nice JSX syntax. This will render those paragraph elements as an array, without any kind of wrapper <div>.



Accessibility

There's also an alternate, more concise syntax for using React Fragments:

```
class App extends React.Component {  
  render()  
  { return (<>  
    <p>I would</p>  
    <p>really like</p>  
    <p>to render</p>  
    <p>an array</p>  
  </>);  
}  
}
```



Accessibility

Depending on your tooling, linters, build pipeline, etc, this might not work for you – the release notes say that wider support is on the way, but I've noticed create-react-app doesn't support it yet.



Adding Images, Fonts, and Files

With Webpack, using static assets like images and fonts works similarly to CSS.

You can import a file right in a JavaScript module. This tells Webpack to include that file in the bundle. Unlike CSS imports, importing a file gives you a string value. This value is the final path you can reference in your code, e.g. as the src attribute of an image or the href of a link to a PDF.

To reduce the number of requests to the server, importing images that are less than 10,000 bytes returns a data URI instead of a path. This applies to the following file extensions: bmp, gif, jpg, jpeg, and png. SVG files are excluded due to #1153.

Example : [Link](#)

Extra Examples

- 1) <https://github.com/TopsCode/React/blob/master/Module3/3.6.1indexControlledFormComponents.js>
- 2) <https://github.com/TopsCode/React/blob/master/Module3/3.6.2indexInputTypeFillValuesFromDropDown.js>
- 3) <https://github.com/TopsCode/React/blob/master/Module3/3.6.3indexTempratureInput.js>
- 4) <https://github.com/TopsCode/React/blob/master/Module3/3.6.4indexTempratureConverter.js>
- 5) <https://github.com/TopsCode/React/blob/master/Module3/3.6.5indexCompositionCssFile.css>
- 6) <https://github.com/TopsCode/React/blob/master/Module3/3.6.6indexCompositionVsInheritanceWithCss.js>
- 7) <https://github.com/TopsCode/React/blob/master/Module3/3.6.7indexCompositionCssFile.css>
- 8) <https://github.com/TopsCode/React/blob/master/Module3/3.6.8indexCompositionWithBorder.js>
- 9) <https://github.com/TopsCode/React/blob/master/Module3/3.6.9indexFormWithBorderCss.css>

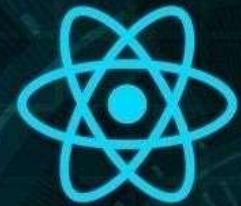
Ajax

<https://github.com/TopsCode/React/blob/master/Module4/4.1.1indexAjax.js>

<https://github.com/TopsCode/React/blob/master/Module4/4.1.2indexAjaxErrorHandler.js>

Error Boundaries

<https://github.com/TopsCode/React/blob/master/Module4/4.1.3indexErrorBoundaries.js>



React

Lazy-Loading Components & Code-Splitting

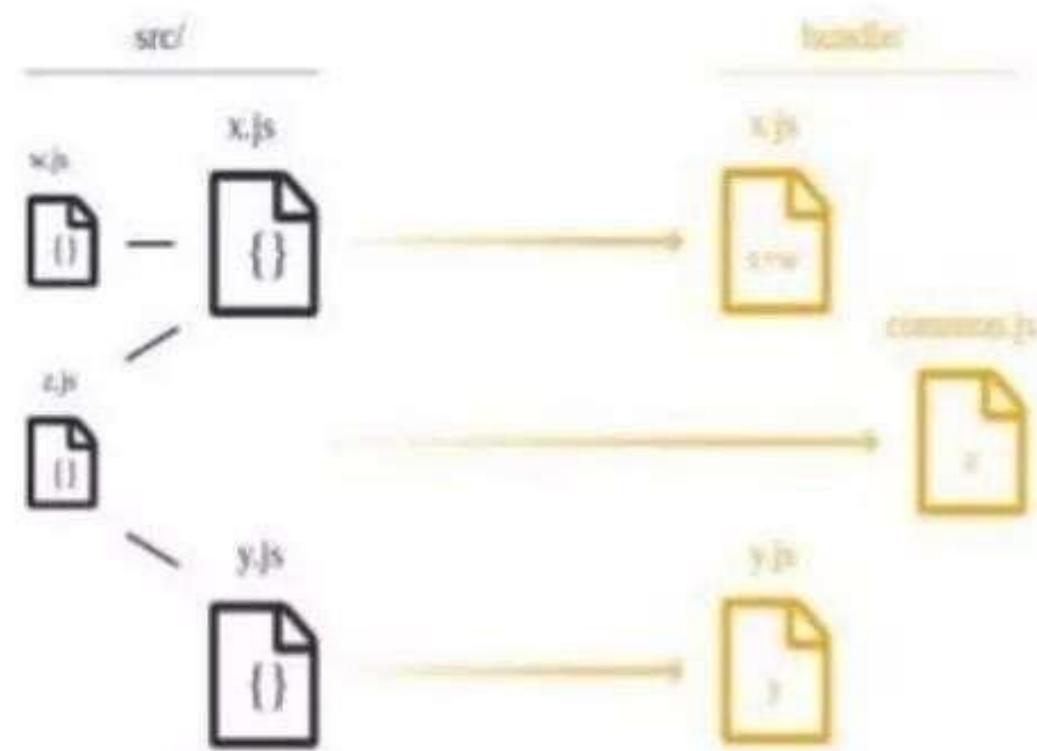
Code-Splitting

Instead of downloading the entire app before users can use it, code splitting allows you to split your code into small chunks which you can then load on demand.

This project setup supports code splitting via dynamic import(). Its proposal is in stage 3. The import() function-like form takes the module name as an argument and returns a Promise which always resolves to the namespace object of the module.

Code-Splitting

- It allows us to split our code into various chunks which you can then load on demand.
- It helps us to reduce the initial response size required to render the page ,



Code-Splitting

moduleA.js

```
const moduleA = 'Hello';
export { moduleA};
```

App.js

```
import React, { Component } from 'react';
class App extends Component
{ handleClick = () => {
    import('./moduleA')
    .then(({ moduleA }) => {
        // Use moduleA
    })
    .catch(err => {
        // Handle failure
    });
    render() { return (
        <div>
            <button onClick={this.handleClick}>Load</button>
        </div>
    ); }
}
export default App;
```

Import from
other file

Code-Splitting

Instead of downloading the entire app before users can use it, code splitting allows you to split your code into small chunks which you can then load on demand.

This project setup supports code splitting via dynamic import(). Its proposal is in stage 3. The import() function-like form takes the module name as an argument and returns a Promise which always resolves to the namespace object of the module.

React.lazy

The `React.lazy` function lets you render a dynamic import as a regular component.

Note:

React.lazy and Suspense is not yet available for server-side rendering. If you want to do code-splitting in a server rendered app, we recommend Loadable Components. It has a nice guide for bundle splitting with server-side rendering.

React.lazy

Before

```
import OtherComponent from
'./OtherComponent';
function MyComponent() {
  return (
    <div>
      <OtherComponent />
    </div>
  );
}
```

After

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
function MyComponent() {
  return (
    <div> <OtherComponent /> </div>
  );
}
```

React.lazy

This will automatically load the bundle containing the OtherComponent when this component gets rendered.

React.lazy takes a function that must call a dynamic import(). This must return a Promise which resolves to a module with a default export containing a React component.



React Suspense

If the module containing the OtherComponent is not yet loaded by the time MyComponent renders, we must show some fallback content while we're waiting for it to load - such as a loading indicator. This is done using the Suspense component.

Note:

Install bootstrap for use Spinner class

React Suspense

- ***Pause any state update until the data is ready***
- ***Add async data to any component without “plumbing”***
- ***On a fast network, render after the whole tree is ready***
- ***On a slow network, precisely control the loading states***
- ***There's both a high-level and a low-level API***

Example Link :

<https://github.com/TopsCode/React/tree/master/Module4/4.3srcRouterWithlazyComponent>

Router



- ✓ React Router is a powerful routing library built on top of React Framework.
- ✓ It helps in adding new screens and flows to the application very quickly.
- ✓ It keeps the URL in sync with data that's being displayed on the web page.

Router Advantages

- 1 Easily understands the application views.
- 2 It can restore any state and view with simple URL.
- 3 It handles nested views and the resolutions.
- 4 State can be restored by the user by moving backward and forward.
- 5 It maintains a standardized structure and behavior.
- 6 While navigating it can do implicit CSS transitions.

Other Example

- 1) <https://github.com/TopsCode/React/tree/master/Module4/4.2RouteSrc/src>
- 2) <https://github.com/TopsCode/React/tree/master/Module4/4.3srcRouterWithlazyComponent>
- 3) <https://github.com/TopsCode/React/tree/master/Module4/4.4validationSrc/src>
- 4) <https://github.com/TopsCode/React/tree/master/Module4/4.5srcBPMCalculator>

Module –13

React-Redux



Quick Start

React Redux is the official React binding for Redux. It lets your React components read data from a Redux store, and dispatch actions to the store to update data.

Installation

React Redux 7.x requires React 16.8.3 or later.

To use React Redux with your React app:

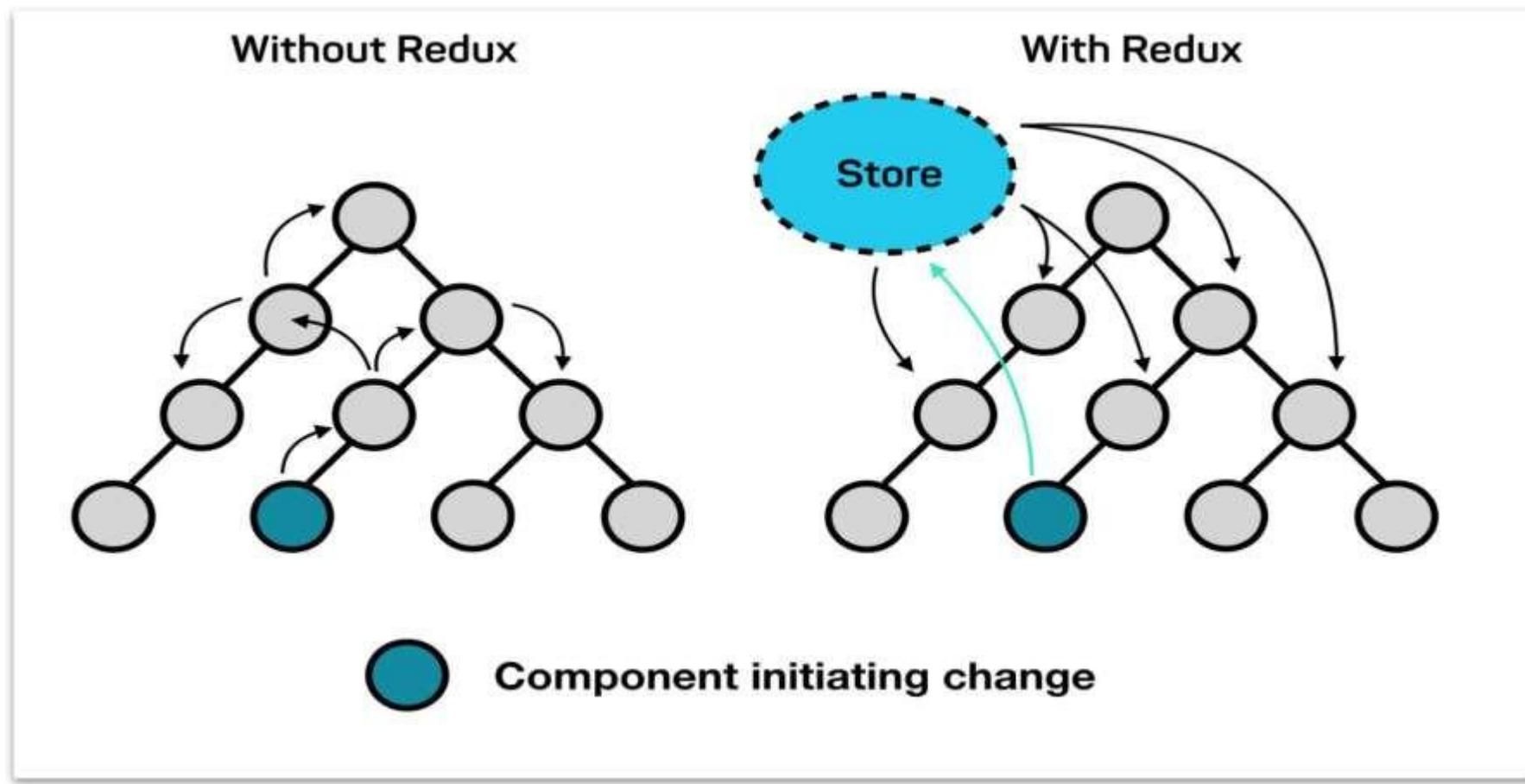
```
npm install react-redux r
```

Provider

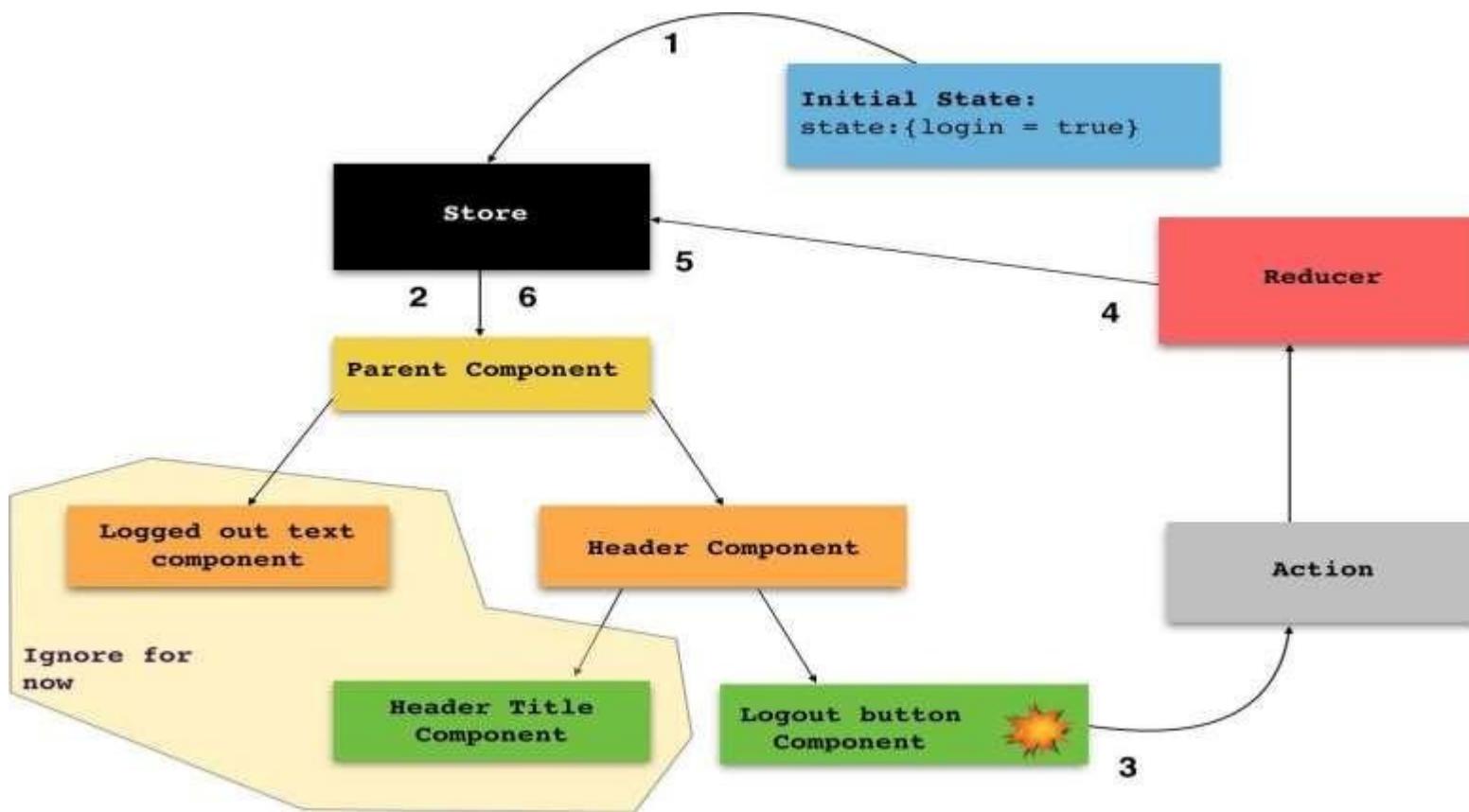
React Redux provides `<Provider />`, which makes the Redux store available to the rest of your app:

```
import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'
import store from './store'
import App from './App'
const rootElement = document.getElementById('root')
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  rootElement )
```

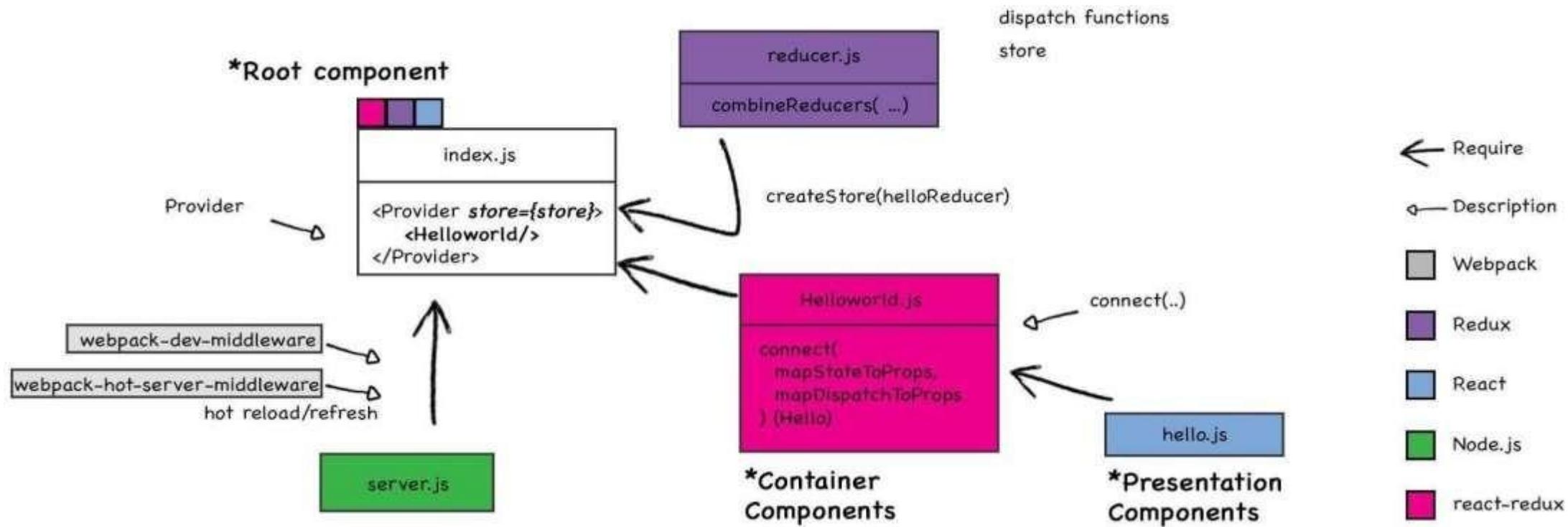
React-Redux



React-Redux



React-Redux



React-Redux

Example :

<https://github.com/TopsCode/React/tree/master/Module5/5.1ReactReduxsrc>

Module –14

Fetch Data using GraphQL

What is GraphQL

GraphQL is an open source server-side technology which was developed by Facebook to optimize RESTful API calls. It is an execution engine and a data query language. This tutorial will introduce you to the fundamental concepts of GraphQL including –

- Implement GraphQL API using Apollo server
 - Test GraphQL API using GraphiQL
- Build ReactJS (with Apollo Client library) and
- jQuery client applications to consume the API

Why GraphQL

RESTful APIs follow clear and well-structured resource-oriented approach. However, when the data gets more complex, the routes get longer. Sometimes it is not possible to fetch data with a single request. This is where GraphQL comes handy. GraphQL structures data in the form of a graph with its powerful query syntax for traversing, retrieving, and modifying data.

Consider the GraphQL query given below –

```
{  
  students {  
    id  
    firstName  
  }  
}
```

}

GraphQL - Environment Setup

Step 1 – Verify Node and Npm Versions

After installing NodeJs, verify the version of node and npm using following commands on the terminal –

```
C:\Users\Admin>node -v  
v8.11.3
```

```
C:\Users\Admin>npm -v 5.6.0
```

Step 2 – Create a Project Folder and Open in VSCode

The root folder of project can be named as test-app.

Open the folder using visual studio code editor by using the instructions below –

```
C:\Users\Admin>mkdir test-app
```

```
C:\Users\Admin>cd test-app
```

```
C:\Users\Admin\test-app>code.
```

Step 3 – Create package.json and Install the Dependencies

Create a package.json file which will contain all the dependencies of the GraphQL server application.

```
{  
  "name": "hello-world-server",  
  "private": true,  
  "scripts": {  
    "start": "nodemon --ignore data/ server.js"  
  },  
  
  "dependencies": {  
    "apollo-server-express": "^1.4.0",  
    "body-parser": "^1.18.3",
```

```
"cors": "^2.8.4",
"express": "^4.16.3",
"graphql": "^0.13.2",
"graphql-tools": "^3.1.1"
},
"devDependencies": {
  "nodemon": "1.17.1"
}
```

Install the dependencies by using the command as given below –

```
C:\Users\Admin\test-app>npm install
```

Step 4 – Create Flat File Database in Data Folder

In this step, we use flat files to store and retrieve data. Create a folder data and add two files **students.json** and **colleges.json**.

Following is the **colleges.json** file –

```
[  
{  
  "id": "col-101",  
  "name": "Brijesh kumar pandey",  
  "location": "Uttar Pradesh",  
  "rating":5.0  
},  
{
```

```
"id": "col-102",
"name": "CUSAT",
"location": "Kerala",
"rating": 4.5
}]
```

Following is the **students.json** file –

```
[
{
  "id": "S1001",
  "firstName": "Mohtashim",
  "lastName": "Mohammad",
  "email": "mohtashim.mohammad@tutorialpoint.org",
  "password": "pass123",
  "collegeId": "col-102"
},
```

```
{  
  "id": "S1002",  
  "email": "kannan.sudhakaran@tutorialpoint.org",  
  "firstName": "Kannan",  
  "lastName": "Sudhakaran",  
  "password": "pass123",  
  "collegeId": "col-101"  
},  
  
{  
  "id": "S1003",  
  "email": "kiran.panigrahi@tutorialpoint.org",  

```

```
"collegeId": "col-101"  
}]
```

Step 5 – Create a Data Access Layer

We need to create a datastore that loads the data folder contents. In this case, we need collection variables, *students* and *colleges*. Whenever the application needs data, it makes use of these collection variables.

Create file db.js with in the project folder as follows –

```
const { DataStore } = require('notarealdb');  
const store = new DataStore('./data');  
module.exports = {  
  students:store.collection('students'),
```

```
colleges:store.collection('colleges')};
```

Step 6 – Create Schema File, schema.graphql

Create a schema file in the current project folder and add the following contents –

```
type Query {  
  test: String  
  
}
```

Step 7 – Create Resolver File, resolvers.js

Create a resolver file in the current project folder and add the following contents –

```
const Query = {
```

```
test: () => 'Test Success, GraphQL server is up &
running !!'}module.exports = {Query}
```

Step 8 – Create Server.js and Configure GraphQL

Create a server file and configure GraphQL as follows –

```
const bodyParser = require('body-parser');const cors = r
equire('cors');const express = require('express');const db =
require('./db');

const port = process.env.PORT || 9000;const app = express();
const fs = require('fs')const typeDefs =
fs.readFileSync('./schema.graphql', {encoding:'utf-8'})const resolvers =
require('./resolvers')
```

```
const {makeExecutableSchema} = require('graphql-tools')
const schema = makeExecutableSchema({typeDefs, resolvers})

app.use(cors(), bodyParser.json());
const {graphiqlExpress, graphqlExpress} = require('apollo-server-express')
app.use('/graphql', graphqlExpress({schema}))
app.use('/graphiql', graphiqlExpress({endpointURL: '/graphql'}))

app.listen(
  port, () => console.info(`Server started on port ${port}`)
);
```

Step 9 – Run the Application and Test with GraphiQL

Verify the folder structure of project test-app as follows –

```
test-app /  
-->package.json  
-->db.js  
-->data  
  students.json  
  colleges.json  
-->resolvers.js  
-->schema.graphql  
-->server.js
```

Run the command npm start as given below –

```
C:\Users\Admin\test-app>npm start
```

The server is running in 9000 port, so we can test the application using GraphiQL tool. Open the browser and enter the URL

<http://localhost:9000/graphiql>. Type the following query in the editor –

```
{  
Test  
}
```

The response from the server is given below –

```
{  
"data": {  
"test": "Test Success, GraphQL server is running !!"  
}  
}
```

The screenshot shows a GraphiQL interface running at `localhost:9000/graphiql`. The query entered is:

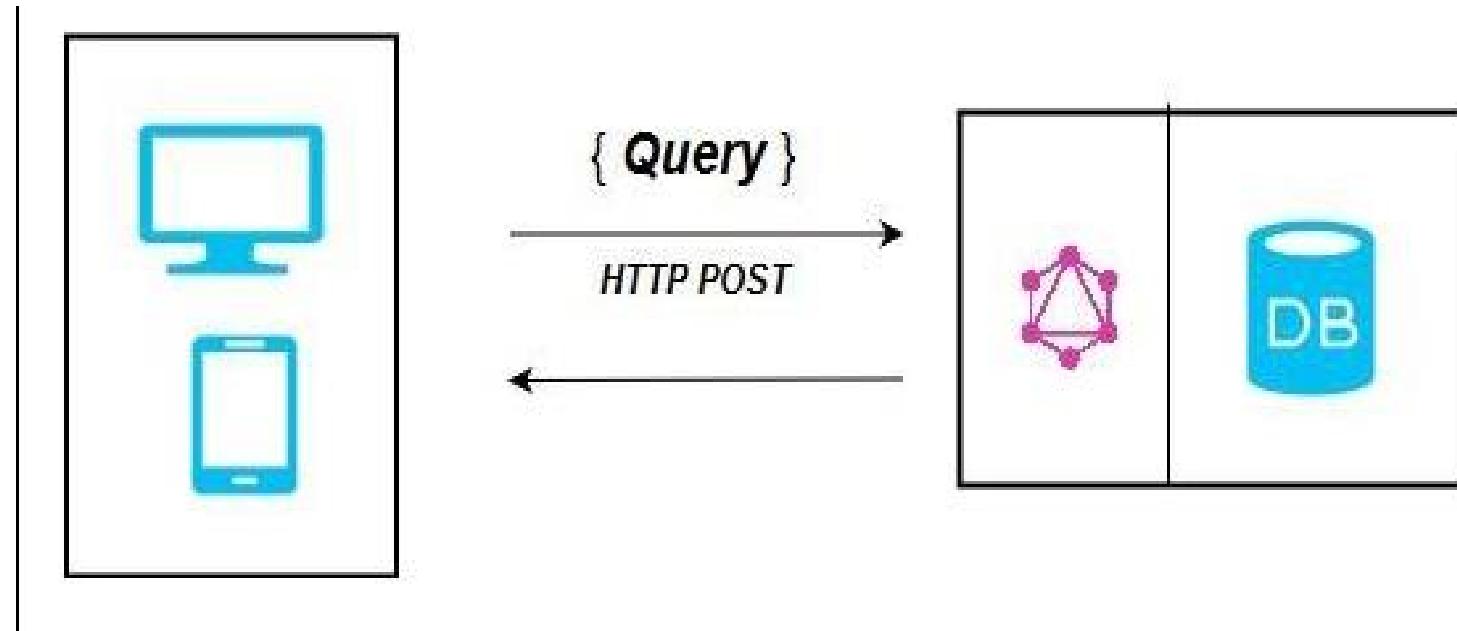
```
1 {  
2   test  
3 }
```

The response received is:

```
* {  
  "data": {  
    "test": "Test Success, GraphQL server is up & running!!"  
  }  
}
```

GraphQL Server with Connected Database

This architecture has a GraphQL Server with an integrated database and can often be used with new projects. On the receipt of a Query, the server reads the request payload and fetches data from the database. This is called resolving the query. The response



GraphQL - Apollo Client

We have used Apollo Server to build graphql specification on server side. It is quick and easy to build production ready GraphQL server. Now let us understand the client side.

Apollo Client is the best way to use GraphQL to build client applications. The client is designed to help developer quickly build a UI that fetches data with GraphQL and can be used with any JavaScript front-end.

Step 1 – Download and Install Required Dependencies for the Project

Create a folder apollo-server-app. Change your directory to **apollo-server-app** from the terminal. Then, follow steps 3 to 5 explained in the Environment Setup chapter.

Step 2 – Create a Schema

Add **schema.graphql** file in the project folder **apollo-server-app** and add the following code –

Step 3 – Add Resolvers

Create a file **resolvers.js** in the project folder and add the following code –

Step 4 – Run the Application

Create a **server.js** file. Refer step 8 in the Environment Setup Chapter. Execute the command *npm start* in the terminal. The server will be up and running on 9000 port. Here, we will use GraphiQL as a client to test the application.

Open browser and type the URL **http://localhost:9000/graphiql**. Type the following query in the editor.

Module -15

Next JS

What is Next.js ?

The Next.js is React Based framework with server side rendering capability. It is very fast and SEO friendly. Using Next.js, you can create robust react based application quite easily and test them.

The Next.js is React Based framework with server side rendering capability. It is very fast and SEO friendly.

Using Next.js, you can create robust react based application quite easily and test them. Following are the key features of Next.js.

Hot Code Reload – Next.js server detects modified files and reloads them automatically.

Automatic Routing – No need to configure any url for routing. files are to be placed in pages folder. All urls will be mapped to file system. Customization can be done.

Component specific styles – styled-jsx provides support for global as well as component specific styles.

Server side rendering – react components are prerendered on server hence loads faster on client.

Node Ecosystem – Next.js being react based gels well with Node ecosystem.

Automatic code split – Next.js renders pages with libraries they need. Next.js instead of creating a single large javascript file, creates multiples resources. When a page is loaded, only required javascript page is loaded with it.

Prefetch – Next.js provides Link component which is used to link multiple components supports a prefetch property to prefetch page resources in background.

Dynamic Components – Next.js allows to import JavaScript modules and React Components dynamically.

Export Static Site – Next.js allows to export full static site from your web application.

Built-in Typescript Support – Next.js is written in Typescripts and provides excellent Typescript support.

.

Next Js Environment setup

As Next.js is a react based framework, we are using Node environment. Now make sure you have **Node.js** and **npm** installed on your system. You can use the following command to install Next.js –

```
npm install next react react-dom
```

You can observe the following output once Next.js is successfully installed –

```
+ react@16.13.1
+ react-dom@16.13.1
+ next@9.4.4
```

```
added 831 packages from 323 contributors and audited 834
packages in 172.989s
```

Now, let's create a node package.json –

```
npm init
```

Select default values while creating a package.json –

This utility will walk you through creating a package.json file. It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (nextjs)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to \Node\nextjs\package.json:

{

 "name": "nextjs",

 "version": "1.0.0",

```
"description": "",  
"main": "index.js",  
"dependencies": {  
    "next": "^9.4.4",  
    "react": "^16.13.1",  
    "react-dom": "^16.13.1"  
},  
"devDependencies": {},  
"scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
},  
"author": "",  
"license": "ISC"  
}
```

Is this OK? (yes)

Now update the scripts section of package.json to include Next.js commands.

```
{  
  "name": "nextjs",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "dependencies": {  
    "next": "^9.4.4",  
    "react": "^16.13.1",  
    "react-dom": "^16.13.1"  
  },  
  "devDependencies": {}  
}
```

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "dev": "next",  
  "build": "next build",  
  "start": "next start"  
},  
"author": "",  
"license": "ISC"}
```

Create pages directory.

Create a pages folder within nextjs folder and create an index.js file with following contents.

```
function HomePage() {  
  return <div>Welcome to Next.js!</div>  
}  
export default HomePage
```

Start Next.js Server

Run the following command to start the server –.

```
npm run dev  
> nextjs@1.0.0 dev \Node\nextjs  
> next
```

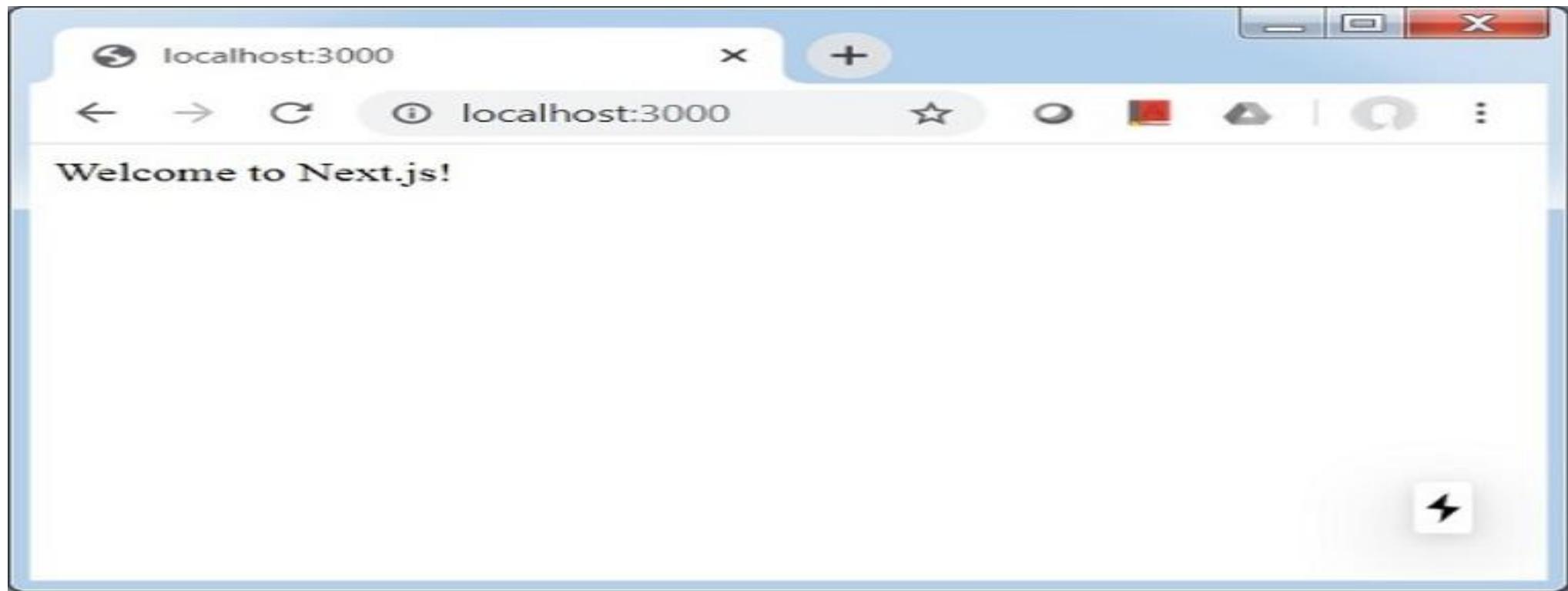
```
ready - started server on http://localhost:3000  
event - compiled successfully
```

```
event - build page: /
wait  - compiling...
event - compiled successfully
event - build page: /next/dist/pages/_error
wait  - compiling...
event - compiled successfully
```

Explore our latest online courses and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Verify Output

Open localhost:3000 in a browser and you will see the following output.



B

Next.js - Pages

In Next.js, we can create pages and navigate between them using file system routing feature. We'll use **Link** component to have a client side navigation between pages.

In Next.js, a page is a React Component and are exported from `pages` directory. Each page is associated with a route based on its file name. For example

pages/index.js is linked with '/' route.

pages/posts/first.js is linked with '/posts/first' route and so on.

Let's update the nextjs project created in [Environment Setup](#) chapter.

Create post directory and first.js within it with following contents.

```
export default function FirstPost() {  
return <h1>My First Post</h1>}
```

Add Link Support to go back to Home page. Update first.js as follows -

```
import Link from 'next/link'  
export default function FirstPost() {  
return (
```

```
<>
<h1>My First Post</h1>
<h2>
  <Link href="/">
    <a>Home</a>
  </Link>
</h2>
</>
)}
```

Add Link Support to home page to navigate to first page. Update index.js as follows –

```
import Link from 'next/link'
function HomePage() {
  return (
<>
```

```
<div>Welcome to Next.js!</div>
<Link href="/posts/first"><a>First Post</a></Link>
</>
)
}

export default HomePage
```

Start Next.js Server

Run the following command to start the server –.

```
npm run dev
> nextjs@1.0.0 dev \Node\nextjs
> next
```

```
ready - started server on http://localhost:3000
event - compiled successfully
event - build page: /
wait - compiling...
```

```
event - compiled successfully
event - build page: /next/dist/pages/_error
wait  - compiling...
event - compiled successfully
```

Next.js - Routing

Next.js uses file system based router. Whenever we add any page to **pages** directory, it is automatically available via url. Following are the rules of this router.

Index Routes – An index.js file present in a folder maps to root of directory. For example –

pages/index.js maps to '/'.

pages/posts/index.js maps to '/posts'.

Nested Routes – Any nested folder structure in pages directory because router url automatically. For example –

pages/settings/dashboard/about.js maps to '/settings/dashboard/about'.

pages/posts/first.js maps to '/posts/first'.

Dynamic Routes – We can use named parameter as well to match url. Use brackets for the same. For example –

pages/posts/[id].js maps to '/posts/:id' where we can use URL like '/posts/1'.

pages/[user]/settings.js maps to '/posts/:user/settings' where we can use URL like '/abc/settings'.

pages/posts/...all.js maps to '/posts/*' where we can use any URL like '/posts/2020/jun/'.

Page Linking

Next.JS allows to link pages on client side using Link react component.

It has following properties –

href – name of the page in pages directory. For example **/posts/first** which refers to first.js present in pages/posts directory.

Let's create an example to demonstrate the same.

In this example, we'll update index.js and first.js page to make a server hit to get data.

Let's update the nextjs project used in [Global CSS Support](#) chapter.

Update index.js file in pages directory as following.

```
import Link from 'next/link'import Head from 'next/head'  
function HomePage(props) {  
return (  
  <>  
  <Head>  
    <title>Welcome to Next.js!</title>  
  </Head>  
  <div>Welcome to Next.js!</div>  
  <Link href="/posts/first">> <a>First Post</a></Link>  
  <br/>  
  <div>Next stars: {props.stars}</div>  
    
</>
```

```
)}

export async function getServerSideProps(context) {
  const res = await fetch('https://api.github.com/repos/vercel/next.js')
  const json = await res.json()
  return {
    props: { stars: json.stargazers_count }
  }
}

export default HomePage
```

Start Next.js Server

Run the following command to start the server –.

```
npm run dev
> nextjs@1.0.0 dev \Node\nextjs
> next
```

ready - started server on http://localhost:3000
event - compiled successfully
event - build page: /
wait - compiling...
event - compiled successfully
event - build page: /next/dist/pages/_error
wait - compiling...
event - compiled successfully



My First Post

x

+



localhost:3000/posts/...



My First Post

[Home](#)

Next stars: 50599

Next.js - Dynamic Routing

In Next.js, we can create routes dynamically. In this example, we'll create pages on the fly and their routing.

Step 1. Define [id].js file – [id].js represents the dynamic page where id will be relative path. Define this file in pages/post directory.

Step 2. Define lib/posts.js – posts.js represents the ids and contents. lib directory is to be created in root directory.

[id].js

Update [id].js file with `getStaticPaths()` method which sets the paths and `getStaticProps()` method to get the contents based on id.

```
import Link from 'next/link'
import Head from 'next/head'
import Container from '../../components/container'
import { getAllPostIds, postData } from '../../lib/posts'
export default function Post({ postData }) {
  return (
    <Container>
      {postData.id}
      <br />
      {postData.title}
      <br />
      {postData.date}
    </Container>
  )
}

export async function getStaticPaths() {
  const paths = getAllPostIds()
  return {
    paths,
```

```
fallback: false
}
export async function getStaticProps({ params }) {
const postData = getpostData(params.id)
return {
props: {
postData
}
}
}
```

posts.js

posts.js contains getAllPostIds() to get the ids and getpostData() to get corresponding contents.

```
export function getpostData(id) {
const postOne = {
```

```
title: 'One',
id: 1,
date: '7/12/2020'
}
```

```
const postTwo = {
title: 'Two',
id: 2,
date: '7/12/2020'
}
```

```
if(id == 'one'){
    return postOne;
}else if(id == 'two'){
    return postTwo;
} }
```

```
export function getAllPostIds() {
```

```
return [{}  
  params: {  
    id: 'one'  
  }  
,  
{  
  params: {  
    id: 'two'  
  }  
}];}
```

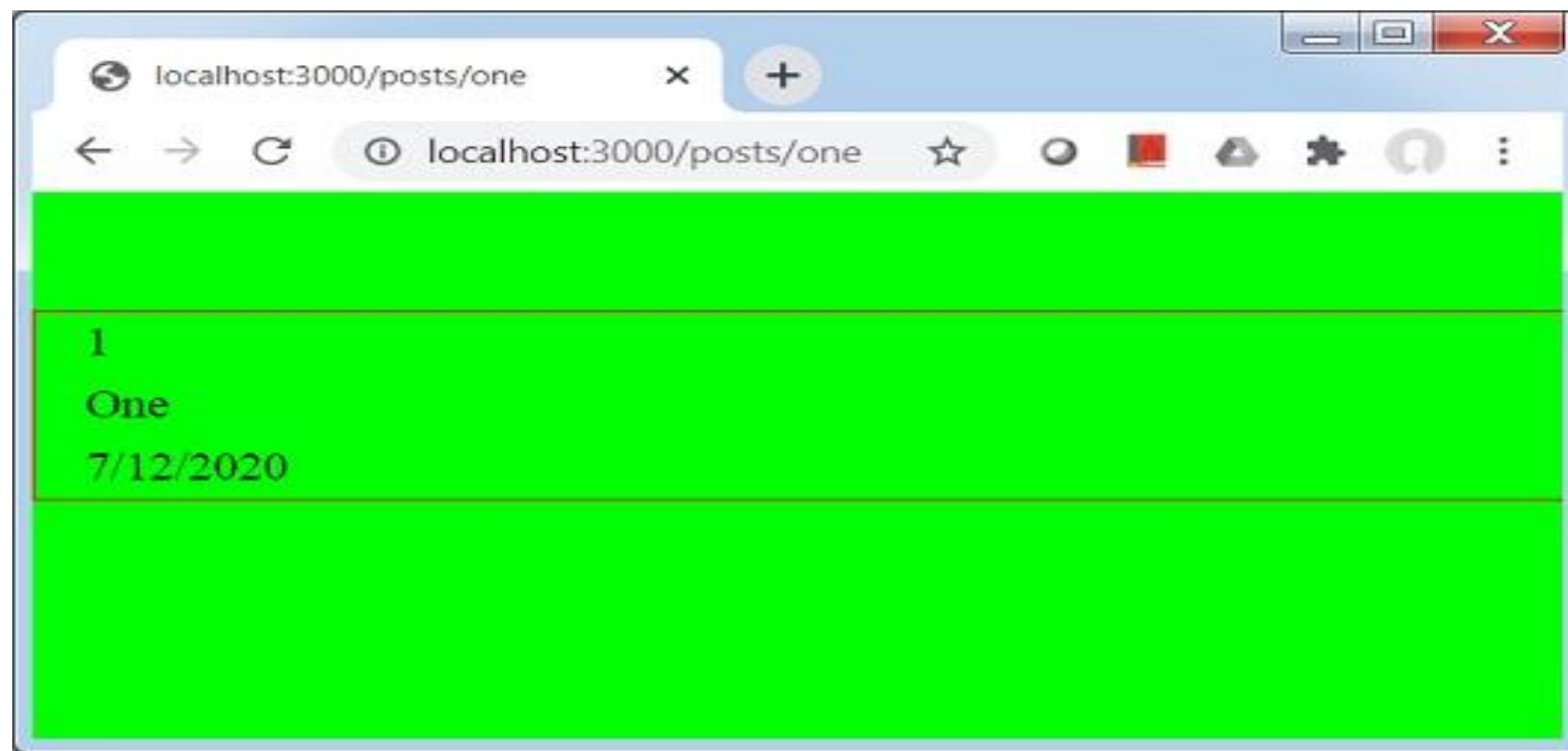
Explore our latest online courses and learn new skills at your own pace.
Enroll and become a certified expert to boost your career.

Start Next.js Server

Run the following command to start the server –.

```
npm run dev  
> nextjs@1.0.0 dev \Node\nextjs  
> next
```

```
ready - started server on http://localhost:3000  
event - compiled successfully  
event - build page: /  
wait - compiling...  
event - compiled successfully  
event - build page: /next/dist/pages/_error  
wait - compiling...  
event - compiled successfully
```



Module 16

Introduction –Vue.js, D3.js, Chart.js

What is Vue.js ?

Vue.js lets you **extend** HTML with HTML attributes called **directives**

Vue.js directives offers **functionality** to HTML applications

Vue.js provides **built-in** directives and **user defined** directives

Vue.js Directives

Vue.js uses double braces **{} {}** as place-holders for data.

Vue.js directives are HTML attributes with the prefix **v-**

Vue Example

In the example below, a new Vue object is created with **new Vue()**.

The property **el:** binds the new Vue object to the HTML element with **id="app"**.

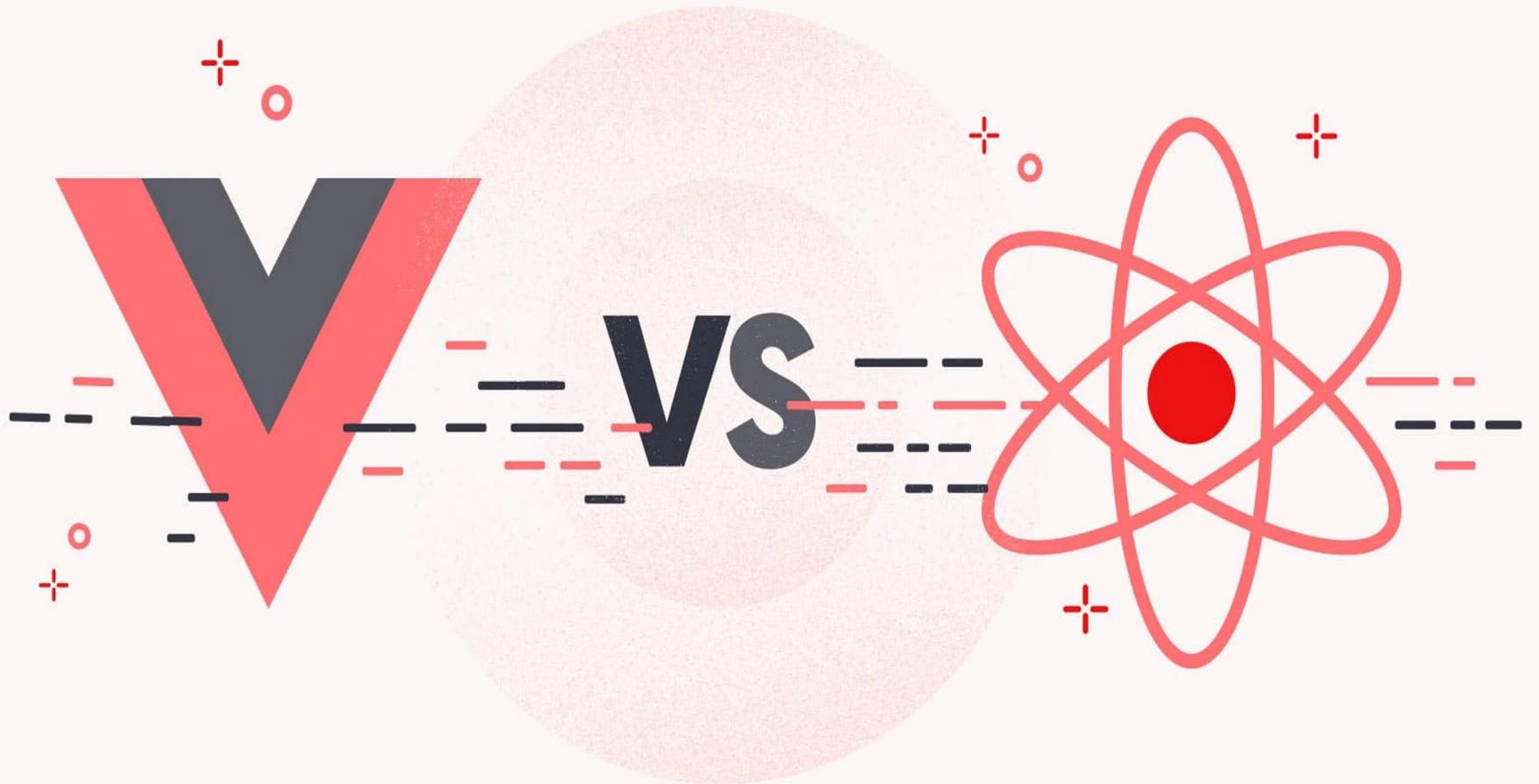
Example :

```
<div id="app">
<h1>{{ message }}</h1>
</div>
```

```
<script>
var myObject = new Vue({
el: '#app',
```

```
data: {message: 'Hello Vue!'}  
}  
</script>
```

React js vs Vue js



Virtual DOM

Virtual component-based UI development DOM

Official component library for building mobile apps

Open source

Creator

Initial release

Written in

License

Syntax

Live Websites (BuiltWith, 2022)

Top features



VUE



Evan You

2014

JavaScript

MIT

HTML (default), JSX

2 million

- Elegant programming style and patterns
- Easy learning curve
- Thorough documentation



REACT



Facebook / Jordan Walk

2013

JavaScript

MIT

JSX

11 million

- Elegant programming style and patterns
- Rich package ecosystem
- Widespread usage



What is D3.js

D3.js is a JavaScript library for manipulating HTML based on data.

How to Use D3.js?

To use D3.js in your web page, **add a link** to the library:

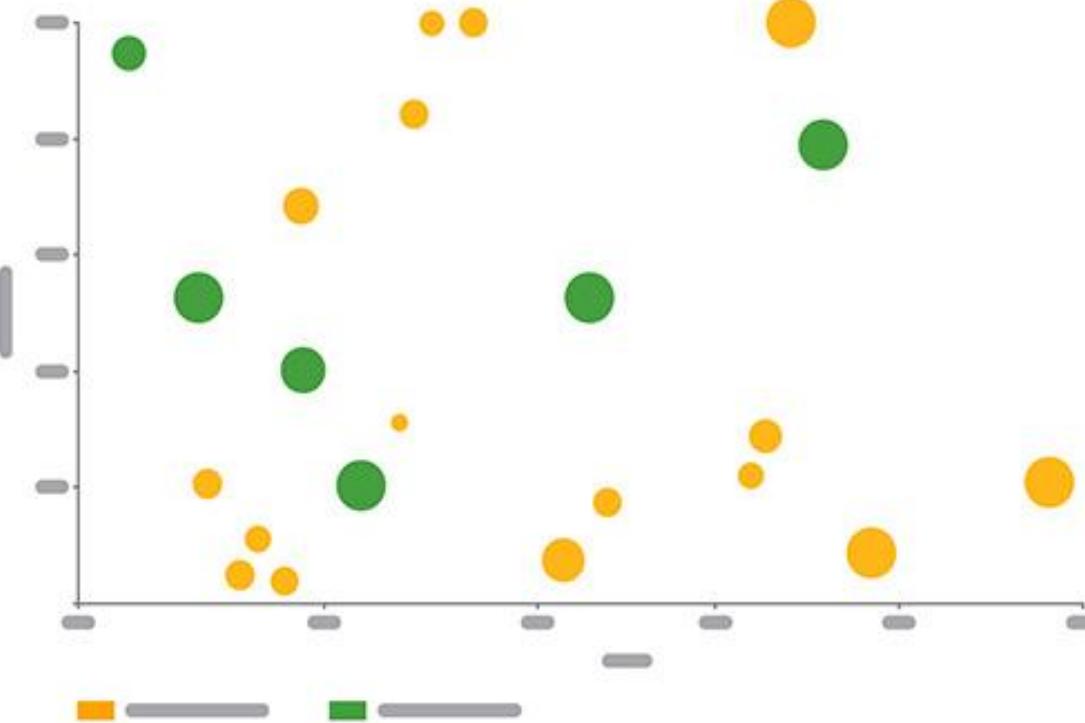
```
<script src="//d3js.org/d3.v3.min.js"></script>
```

D3.js is easy to use.

This script selects the body element and appends a paragraph with the text "Hello World!":

```
d3.select("body").append("p").text("Hello World!");
```

Scatter Plot



What is Scatter Diagram and How to Create it.

```
// Set Dimensions
const xSize = 500;
const ySize = 500;
const margin = 40;
const xMax = xSize - margin*2;
const yMax = ySize - margin*2;

// Create Random Points
const numPoints = 100;
const data = [];
for (let i = 0; i < numPoints; i++) {
    data.push([Math.random() * xMax, Math.random() * yMax]);
}
```

```
// Append SVG Object to the Page
const svg = d3.select("#myPlot")
    .append("svg")
    .append("g")
    .attr("transform", "translate(" + margin + "," + margin +
")");
```



```
// X Axis
const x = d3.scaleLinear()
    .domain([0, 500])
    .range([0, xMax]);
```



```
svg.append("g")
    .attr("transform", "translate(0," + yMax + ")")
```

```
.call(d3.axisBottom(x)) ;  
  
// Y Axis  
const y = d3.scaleLinear()  
    .domain([0, 500])  
    .range([-yMax, 0]);  
  
svg.append("g")  
    .call(d3.axisLeft(y));  
  
// Dots  
svg.append('g')  
    .selectAll("dot")  
    .data(data).enter()
```

```
.append("circle")
  .attr("cx", function (d) { return d[0] } )
  .attr("cy", function (d) { return d[1] } )
  .attr("r", 3)
  .style("fill", "Red");
```

Advantages of D3.js

- . D3.js is a Javascript library. So, it can be used with any JS framework of your choice like Angular.js, React.js or Ember.js.
- . D3 focuses on data, so it is the most appropriate and specialized tool for data visualizations.
- . D3 is open-source. So you can work with the source code and add your own features.

- It works with web standards so you don't need any other technology or plugin other than a browser to make use of D3.
- D3 works with web standards like HTML, CSS and SVG, there is no new learning or debugging tool required to work on D3.
-
- D3 does not provide any specific feature, so it gives you complete control over your visualization to customize it the way you want. This gives it an edge over other popular tools like Tableau or QlikView.
-
- Since D3 is lightweight, and works directly with web standards, it is extremely fast and works well with large datasets.

What is Chart.js ?

Chart.js is a free JavaScript library for making HTML-based charts. It is one of the simplest visualization libraries for JavaScript, and comes with the following built-in chart types:

- Scatter Plot
- Line Chart
- Bar Chart
- Pie Chart
- Donut Chart
- Bubble Chart

- Area Chart
- Radar Chart
- Mixed Chart

How to Use Chart.js?

Chart.js is easy to use.

First, add a link to the providing CDN (Content Delivery Network):

```
<script  
src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js">  
</script>
```

Then, add a <canvas> to where you want to draw the chart:

```
<canvas id="myChart" style="width:100%;max-width:700px"></canvas>
```

The canvas element must have a unique id.

That's all!

Typical Scatter Chart Syntax:

```
const myChart = new Chart("myChart", {  
  type: "scatter",  
  data: {},  
  options: {}  
});
```

Advantages of chart.js

- . Provides responsive, animated charts with a clean look & feel
- . Lightweight core library: 199kb minified and 67kb Gzipped according to Bundlephobia
- . Free open source, with a permissive MIT license
- . Simple, easy to use and get started
- . Good documentation & samples for an open source library
- . Popular, with a large community and plenty of contributors
- . Extensible, via a plugin architecture allowing for additional chart types, features or interactions to be built.