Training Report

HTML and CSS, JavaScript
Or
ReactJS

Name: Vijay Saini
Roll No: 19EBTCS036

Submitted to the Department of computer Science & Engineering.

Bhartiya Institute of Engineering & Technology, Sikar

Certificate

HTML

- Introduction of HTML
- Html tags
- Html attributes
- Elements in html
- Html heading
- Html Style
- Html javaScript
- Html Responsive

CSS

- Introduction of CSS
- CSS Selector
- CSS sizing
- CSS margin
- CSS padding
- CSS positioning
- CSS Animation
- CSS 2-D Transfromation
- CSS 3-D Transfromation

JavaScript

- Introduction of javaScript
- JavaScript & EcmaScript
- JavaScript objects
- JavaScript function
- JavaScript Clint Side web api
- Synchronous javaScript
- JavaScript Callback
- JavaScript Promises
- JavaScript Async await
- javaScript Inport and Export

ReactJs

- Introduction of Reactjs
- Features of Reactjs
- Hooks in reactJs
- Reactjs State
- Reactjs component life cycle
- Html Style
- Html javaScript
- Html Responsive

o Introduction to HTML

HTML is the most powerful language for creating web pages and web applications to be displayed in a web browser, and it stands for Hyper-Text Markup Language. HTML describes the structure of the web pages how it looks and decides that how to display the content of web pages in the browser

o HyperText Markup Language Basics

At its core, HTML is a series of short codes typed into a text-file. These are the tags that power HTML's capabilities. The text is saved as an HTML file and viewed through a web browser. The browser reads the file and translates the text into a visible form, as directed by the codes the author used to write what becomes the visible rendering. Writing HTML requires tags to be used correctly to create the author's vision.

o Html tags

HTML tags help web browsers convert
HTML documents into web pages.

<h1> - <h6>Heading
<p> Paragraph
<i> or Italic / Emphasis
<b> or Bold / Strong
<a> Anchor
 & Unordered List & List Item
<blockquote> Blockquote
<hr> Horizontal Rule
<img> Image
 Division

○ Html attributes

It provide additional information about the element.

alt	Specifies an alternative text for an image
disabled	Specifies that an input element should be disabled
href	Specifies the URL (web address) for a link
id	Specifies a unique id for an element
src	Specifies the URL (web address) for an image
style	Specifies an inline CSS style for an element
title	Specifies extra information about an element (displayed as a tool tip)

Types of Attribute

1--> Global Attributes

2--> Multi-tag Attributes

3--> Single-tag Attributes

○ Html Elements

Html tags are defined by starting and ending tags.

All the information are contained by these tags

There are following elements are mostly used in Html:

1.Main Root---->

The <html> HTML element represents the root (top-level element) of an HTML document, so it is also referred to as the root element. All other elements must be descendants of this element.

2.Sectioning Root ----->

The <body> HTML element represents the content of an HTML document. There can be only one <body> element in a document.

3.Content Sectioning--->

A-The <address> HTML element indicates that the enclosed HTML provides contact information for a person or people, or for an organization.

B- The <aside> HTML element represents a portion of a document whose content is only indirectly related to the document's main content. Asides are frequently presented as sidebars or call-out boxes.

C- The <h1> to <h6> HTML elements represent six levels of section headings. <h1> is the highest section level and <h6> is the lowest.

D- The <main> HTML element represents the dominant content of the body of a document.

E- The <nav> HTML element represents a section of a page whose purpose is to provide navigation links, either within the current document or to other documents.

F- The <section> HTML element represents a generic standalone section of a document, which doesn't have a more specific semantic element to represent it

○ Test content--->

A- The <a> HTML element (or anchor element), with its href attribute, creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address

B- The
 HTML element produces a line break in text.

C- The HTML element is a generic inline container for phrasing content, which does not inherently represent anything. It can be used to group elements for styling purposes

○ Image and multimedia-→

A- The <img> HTML element embeds an image into the document.

B- The **<video>** HTML element embeds a media player which supports video playback into the document.

- Html heading

Section one, HTML headings. HTML defines six levels of headings, and these heading elements are H1, H2, H3, H4, H5, and H6. The H1 element is the highest or most important level, and the H6 element is the least important. These different heading levels help to communicate the organization and hierarchy of the content on a page. For example, headings with an equal or higher ranking indicate the start of a new section, and headings with a lower rank indicate the start of a new subsection that is part of a higher-ranked section.

There are following types of headings

```
<h1>Heading level 1</h1>
<h2>Heading level 2</h2>
<h3>Heading level 3</h3>
<h4>Heading level 4</h4>
<h5>Heading level 5</h5>
<h6>Heading level 6</h6>
```

○ Html Style

Html style are used to add style to an element like font size , width , height , margin , padding etc.

There are following way of html style

A- Inline CSS
B- Internal CSS
C- External CSS

A-Inline CSS example :

```
<p style = "color:#009900; font-size:50px;
        font-style:italic; text-align:center;">
</p>
```

B-internal CSS example :

```
<style>
    .main {
        text-align:center;
      }
    .main_container {
       color:#009900;
       font-size:50px;
        font-weight:bold;
     }
   .ineer_container {
        font-style:bold;
         font-size:20px;
    }
</style>
```

C-External file :

In this we can create a file and import to the html file .

○ Html javaScript

In html javascript are used with Script tags in html page.
It will be in clint side or server side.
It is use to manuplate the DOM (document object model)

There are following Syntext of javascript

JavaScript can change content:

```
document.getElementById("demo").innerHTML = "Hello JavaScript!";
```

JavaScript can change styles:

```
document.getElementById("demo").style.fontSize = "25px";
document.getElementById("demo").style.color = "red";
document.getElementById("demo").style.backgroundColor = "yellow";
```

JavaScript can change attributes:

```
document.getElementById("image").src = "picture.gif";
```

○ Html Responsive

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones):

Setting the view-port

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

eta name="viewport" content="width=device-width, initial-scale=1.0">

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Responsive image

If the CSS width property is set to 100%, the image will be responsive.

Responsive text – size

The text size can be set with a "vw" unit, which means the "viewport width".

That way the text size will follow the size of the browser window :

○ Introduction of CSS

Cascading **S**tyle **S**heets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

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

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

Advantages of Css

CSS saves time − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.

Pages load faster − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.

Easy maintenance − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.

○ Css Selector

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts.

1. Selector − A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.

2. Property − A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border etc.

3. Value − Values are assigned to properties. For example, color property can have value either red or #F1F1F1 etc.

Syntax as follows –

selector { property: value }

There are following type of selector in Css

A-The type selector(target to the html tags)

B-Universal selector (*)

C- Descendant Selector (select two html tags)

D-class selector(.)

E-Id selector(#)

- CSS Sizing

1.Css height and width →

The height and width properties are used to set the height and width of an element.

The height and width properties do not include padding, borders, or margins. It sets the

height/width of the area inside the padding, border, and margin of the element.

CSS height width values -->

A- auto - This is default. The browser calculates the height and width.

B- length - Defines the height/width in Px, cm, etc.

C- % - Defines height/width in percent of the containing block

D- initial - Sets the height/width to its default value.

E- inherit - The height/width will be inherited from its parent value.

2.Max-height and width→

A- The max-width property is used to set the maximum width of an element.

B- The max-width can be specified in length values, like px, cm, or in % of the containing block etc.

○ CSS Margin

Margins are used to create space around elements, outside of any defined borders.

The CSS margin properties are used to create space around elements, outside of any defined borders.

There are following individual sides →

1. Margin – top
2. Margin - bottom
3. Margin – left
4. Margin – Right

Example of margin properties →

Property	Description
margin	A shorthand property for setting all the margin properties in one declaration
Margin-bottom	Sets the bottom margin of an element
Margin-left	Sets the left margin of an element
Margin-right	Sets the right margin of an element
Margin-top	Sets the top margin of an element

○ CSS Padding

The CSS padding properties are used to generate space around an element's content, inside of any defined borders.

There are following individual sides →

1.padding – top

2.padding - bottom

3.padding – left

4.padding – Right

Example of Padding properties →

Property	Description
padding	A shorthand property for setting all the padding properties in one declaration
Padding-bottom	Sets the bottom padding of an element
Padding-left	Sets the left padding of an element
Padding-right	Sets the right padding of an element
Padding-top	Sets the top padding of an element

○ CSS Positioning

The Position property specifies the type of positioning method used for an element.

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

There are five different position values:

1. Static
2. Relative
3. fixed
4. Absolute
5. Sticky

1.Position Static

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element is not positioned in any special way; it is always positioned according to the normal flow of the page:

2.Position Relative

An element with position relative is positioned relative to its normal position.

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

3.Position Fixed

An element with position fixed is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

4.Position absolute

An element with position absolute is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

5.Postion sticky

An element with position absolute is positioned based on the user's scroll position.

A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place .

○ CSS Animation

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.

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.

○ CSS 2-D Transformation

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.

The CSS 2D transforms are used to re-change the structure of the element as translate, rotate, scale and skew etc.

Following is a list of 2D transforms methods→

Translate (x ,y): It is used to transform the element along X-axis and Y-axis.

Translate X(n): It is used to transform the element along X-axis.

Translate Y(n): It is used to transform the element along Y-axis.

rotate(): It is used to rotate the element on the basis of an angle.

Scale (x, y): It is used to change the width and height of an element.

Scale X(n): It is used to change the width of an element.

Scale Y(n): It is used to change the height of an element.

Skew X(): It specifies the skew transforms along with X-axis.

Skew Y(): It specifies the skew transforms along with Y-axis.

matrix(): It specifies matrix transforms.

○ CSS 3-D Transformation

The CSS 3D transforms facilitates you to move an element to X-axis, Y-axis and Z-axis.

Following is a list of 3D transforms methods→

○ JavaScript Introduction

JavaScript is a cross-platform, object-oriented scripting language used to make webpages

matrix3D(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n)	It specifies a 3D transformation, using a 4x4 matrix of 16 values.
translate3D(x,y,z)	It specifies a 3D translation.
translateX(x)	It specifies 3D translation, using only the value for the X-axis.
translateY(y)	It specifies 3D translation, using only the value for the Y-axis.
translateZ(z)	It specifies 3D translation, using only the value for the Z-axis.
scale3D(x,y,z)	It specifies 3D scale transformation
scaleX(x)	It specifies 3D scale transformation by giving a value for the X-axis.
scaley(y)	It specifies 3D scale transformation by giving a value for the Y-axis.
scaleZ(z)	It specifies 3D scale transformation by giving a value for the Z-axis.
rotate3D(X,Y,Z,angle)	It specifies 3D rotation along with X-axis, Y-axis and Z-axis.
rotateX(angle)	It specifies 3D rotation along with X-axis.
rotateY(angle)	It specifies 3D rotation along with Y-axis.
rotateZ(angle)	It specifies 3D rotation along with Z-axis.
perspective(n)	It specifies a perspective view for a 3D transformed element.

interactive (e.g., having complex animations, clickable buttons, popup menus, etc.). There are

also more advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website than downloading files (such as Realtime collaboration between multiple computers). Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.

JavaScript contains a standard library of objects, such as `Array`, `Date`, and `Math`, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

- *Client-side JavaScript* extends the core language by supplying objects to control a browser and its *Document Object Model* (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

- *Server-side JavaScript* extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

This means that in the browser, JavaScript can change the way the webpage (DOM) looks. And, likewise, Node.js JavaScript on the server can respond to custom requests sent by code executed in the browser.

○ ECMA-Script

JavaScript is standardized ECMA International— the European association for standardizing information and communication systems (ECMA was formerly an acronym for the European

Computer Manufacturers Association) to deliver a standardized, international programming language based on JavaScript. This standardized version of JavaScript, called ECMAScript, behaves the same way in all applications that support the standard. Companies can use the open standard language to develop their implementation of JavaScript. The ECMAScript standard is documented in the ECMA-262 specification.

How the name ECMAScript 6

I am taking you back to the 90's , when Java was first introduced at the year 1995 by Sun Microsystems. It was one of the secure language and was very famous but Java wasn't made compatible for web-browsers when it was initially developed and also wasn't a easy language for beginners. **Brendan Each** of Netscape developed a language which is more easier for beginners in May 1995 and named it as **LiveScript**.

LiveScript has been integrated with Netscape Navigator Web-browser and released in September 1995. So Sun Microsystems decided to join hands with Netscape Navigator and released "JavaScript". They integrated the JavaScript with Netscape Navigator and released the version 2.0 of Netscape in the year March 1996. JavaScript became the most used client side scripting language.

What is ES6 →

ECMAScript 2015 is the sixth edition of ECMAScript language specification standard which is used in the implementation of JavaScript. In order to run the ES6 code in modern browser , we use BABEL. BABEL is a transpiler for JavaScript which makes the ES6 code to run in any browser.

- JavaScript Objects

Objects in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

In JavaScript, an object is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color , a design, weight, a material it is made of, etc. The same way, JavaScript objects can have properties, which define their characteristics.

Object and properties →

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

Ex- objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let's create an object named `myCar` and give it properties named `make`, `model`, and `year` as follows:

```
const myCar = new Object();
myCar.make = 'Ninja';
myCar.model = 'BMW';
myCar.year = 2009;
```

The above example could also be written using and object inilization, which is a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces (`{}`):

```
const myCar = {
 make: 'Ford',
model: 'Mustang',
year: 1969
};
```

○ JavaScript-function

Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there

is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

Function declaration

A **function definition** (also called a **function declaration**, or **function statement**) consists of the `function` keyword, followed by:

The name of the function.

A list of parameters to the function, enclosed in parentheses and separated by commas.

For example, the following code defines a simple function named Fun:

```
function Fun(number) {
return number * number;
}
```

`The function square takes one parameter, called number. The function consists of one statement that says to return the parameter of the function (that is, number) multiplied by itself. The statement return specifies the value returned by the function:`

```
return number * number;
```

Function Calling

Defining a function does not *execute* it. Defining it names the function and specifies what to do when the function is called.

Calling the function actually performs the specified actions with the indicated parameters. For example, if you define the function `square`, you could call it as follows:

○ JavaScript Web Api's

When writing client-side JavaScript for web sites or applications, you will quickly encounter **Application Programming Interfaces** (**APIs**). APIs are programming features for manipulating different aspects of the browser and operating system the site is running on, or manipulating data from other web sites or services. In this module, we will explore what APIs are, and how to use some of the most common APIs you'll come across often in your development work.

Introduction to web Api →

First up, we'll start by looking at APIs from a high level — what are they, how do they work, how do you use them in your code, and how are they structured? We'll also take a look at what the different main classes of APIs are, and what kind of uses they have.

Manipulating Documents →

When writing web pages and apps, one of the most common things you'll want to do is manipulate web documents in some way. This is usually done by using the Document Object Model (DOM), a set of APIs for controlling HTML and styling information that makes heavy use of the Document object. In this article, we'll look at how to use the DOM in detail, along with some other interesting APIs that can alter your environment in interesting ways.

Fetching Data From Server →

Another very common task in modern websites and applications is retrieving individual data items from the server to update sections of a webpage without having to load an entirely new page. This seemingly small detail has had a huge impact on the performance and behavior of sites. In this article, we'll explain the concept, and look at technologies that make it possible, such as XMLHttpRequest and the FetchApi.

Clint side Storage →

Modern web browsers feature a number of different technologies that allow you to store data related to web sites and retrieve it when necessary allowing you to persist data long term, save sites offline, and more. This article explains the very basics of how these work.

- Asynchronous Vs synchronous

By default, JavaScript is a synchronous, single threaded programming language. This means that instructions can only run one after another, and not in parallel. Consider the little code snippet below:

```
let a = 1;
let b = 2;
let sum = a + b;
console.log(sum);
```

The above code is pretty simple – it sums two numbers and then logs the sum to the browser console. The interpreter executes these instructions one after another in that order until it is done.

But this method comes along with disadvantages. Say we wanted to fetch some large amount of data from a database and then display it on our interface. When the interpreter reaches the instruction that fetches this data, the rest of the code is blocked from executing until the data has been fetched and returned.

Now you might say that the data to be fetched isn't that large and it won't take any noticeable time. Imagine that you have to fetch data at multiple different points. This delay compounded doesn't sound like something users would want to come across.

Luckily for us, the problems with synchronous JavaScript were addressed by introducing asynchronous JavaScript.

Think of asynchronous code as code that can start now, and finish its execution later. When JavaScript is running asynchronously, the instructions are not necessarily executed one after the other as we saw before.

In order to properly implement this asynchronous behavior , there are a few different solutions developers has used over the years. Each solution improves upon the previous one, which makes the code more optimized and easier to understand in case it gets complex.

To further understand the asynchronous nature of JavaScript, we will go through callback functions, promises, and async and await.

- Callback In JavaScript

A callback is a function that is passed inside another function, and then called in that function to perform a task.

Example:

```
console.log('fired first');
console.log('fired second');
setTimeout(()=>{
   console.log('fired third');
},2000);
console.log('fired last');
```

The snippet above is a small program that logs stuff to the console. But there is something new here. The interpreter will execute the first instruction, then the second, but it will skip over the third and execute the last.

The `setTimeout` is a JavaScript function that takes two parameters. The first parameter is another function, and the second is the time after which that function should be executed in milliseconds. Now you see the definition of callbacks coming into play.

The function inside `setTimeout` in this case is required to run after two seconds (2000 milliseconds). Imagine it being carried off to be executed in some separate part of the browser, while the other instructions continue executing. After two seconds, the results of the function are then returned.

This method was very efficient, but only to a certain point. Sometimes, developers have to make multiple calls to different sources in their code. In order to make these calls, callbacks are being nested until they become very hard to read or maintain. This is referred to as **Callback Hell**

- JavaScript Promises

We hear people make promises all the time. That cousin of yours who promised to send you free money, a kid promising to not touch the cookie jar again without permission...but promises in JavaScript are slightly different.

A promise, in our context, is something which will take some time to do. There are two possible outcomes of a promise:

- We either run and resolve the promise, or
- Some error occurs along the line and the promise is rejected

Promises came along to solve the problems of callback functions. A promise takes in two functions as parameters. That is, resolve and reject. Remember that resolve is success, and reject is for when an error occurs.

How to promise work's:

```
const getData = (dataEndpoint) => {
  return new Promise ((resolve, reject) => {
        if(request is successful){
                resolve();
              }
         else if(there is an error){
                reject();
              }

      });
};
```

The code above is a promise, enclosed by a request to some endpoint. The promise takes in `resolve` and `reject` like I mentioned before.
After making a call to the endpoint for example, if the request is successful, we would resolve the promise and go on to do whatever we want with the response. But if there is an error, the promise will get rejected.

Promises are a neat way to fix problems brought about by callback hell, in a method known as **promise chaining**. You can use this method to sequentially get data from multiple endpoints, but with less code and easier methods.

But there is an even better way! You might be familiar with the following method, as it's a preferred way of handling data and API calls in JavaScript.

○ JavaScript Async & Await

The thing is, chaining promises together just like callbacks can get pretty bulky and confusing. That's why Async and Await was brought about.

To define an async function, you do this:

```
const asyncFunc = async() => {

}
```

Note that calling an async function will always return a Promise. Take a look at this:

Running the above in the browser console, we see that the asyncFunc returns a promise.

Let's really break down some code now. Consider the little snippet below:

```
const asyncFunc = async () => {
        const response = await fetch(resource);
        const data = await response.json();
}
```

The `async` keyword is what we use to define async functions as I mentioned above. But how about `await` ? Well, it stalls JavaScript from assigning `fetch` to the response variable until the promise has been resolved. Once the promise has been resolved, the results from the fetch method can now be assigned to the response variable.

The same thing happens on line 3. The `.json` method returns a promise, and we can use `await` still to delay the assigning until the promise is resolved.

○ JavaScript Import – Export

Export :

The **`export`** declaration is used to export values from a JavaScript module. Exported values can then be imported into other programs with the improt declaration or dynamic import. The value of an imported binding is subject to change in the module that exports it — when a module updates the value of a binding that it exports, the update will be visible in its imported value.

In order to use the `export` declaration in a source file, the file must be interpreted by the runtime as a module. In HTML, this is done by adding `type="module"` to the <script> tag, or by being imported by another module. Modules are automatically interpreted in Strict mode.

Syntext→

```
export let name1, name2/*, … */; // also var
export const name1 = 1, name2 = 2/*, … */; // also
var, let
export function functionName() { /* … */ }
export class ClassName { /* … */ }
export function* generatorFunctionName() { /* … */
}
export const { name1, name2: bar } = o;
export const [ name1, name2 ] = array;
```

Import:

The static **`import`** declaration is used to import read-only live bindings which are exported by another module. The imported bindings are called *live bindings* because they are updated by the module that exported the binding, but cannot be modified by the importing module.

In order to use the `import` declaration in a source file, the file must be interpreted by the runtime as a module In HTML, this is done by adding `type="module"` to the <script> tag. Modules are automatically interpreted in Strict mode.

There is also a function-like dynamic inport(). which does not require scripts of `type="module"`.

Syntext→

```
import defaultExport from "module-name";
import * as name from "module-name";
import { export1 } from "module-name";
import { export1 as alias1 } from "module-name";
```

○ Introduction of ReactJs

React is a popular JavaScript library used for web development. React.js or ReactJS or React are different ways to represent ReactJS. Today's many large-scale companies (Netflix, Instagram, to name a few) also use React JS. There are many advantages of using this framework over other frameworks, and It's ranking under the top 10 programming languages for the last few years under various language ranking indices.

What is ReactJs

React.js is a front-end JavaScript framework developed by Facebook. To build composable user interfaces predictably and efficiently using declarative code, we use React. It's an open-source and component-based framework responsible for creating the application's view layer.

- ReactJs follows the Model View Controller (MVC) architecture, and the view layer is accountable for handling mobile and web apps.
- React is famous for building single-page applications and mobile apps.

History of ReactJs

- Jordan Walke created React, who worked as a software engineer in Facebook has first released an early React prototype called "FaxJS."
- In 2011, React was first deployed on Facebook's News Feed, and later in 2012 on Instagram.
- Current version of reatjs is 18.2.0 .

Why do people chose to programming in Reactjs

- **Fast** - Feel quick and responsive through the Apps made in React can handle complex updates.
- **Scalable** - React performs best in the case of large programs that display a lot of data changes.
- **Popular** - ReactJS gives better performance than other JavaScript languages due to t's implementation of a virtual DOM.
- **Easy to learn** - Since it requires minimal understanding of HTML and JavaScript, the learning curve is low.
- **Reusable UI components** - React improves development and debugging processes.

- Features of Reatjs

1.Jsx:-javascript Syntax Extension

JSX is a preferable choice for many web developers. It isn't necessary to use JSX in React development, but there is a massive difference between writing react.js documents in JSX and JavaScript. JSX is a syntax extension to JavaScript. By using that, we can write HTML structures in the same file that contains JavaScript code.

2.Unidirectionl data flow and flux:-

React.js is designed so that it will only support data that is flowing downstream, in one direction. If the data has to flow in another direction, you will need additional features.

React contains a set of immutable values passed to the component renderer as properties in HTML tags. The components cannot modify any properties directly but support a call back function to do modifications.

3.Virtual document object model:-

React contains a lightweight representation of real DOM in the memory called Virtual DOM. Manipulating real DOM is much slower compared to VDOM as nothing gets drawn on the screen. When any object's state changes, VDOM modifies only that object in real DOM instead of updating whole objects.

That makes things move fast, particularly compared with other front-end technologies that have to update each object even if only a single object changes in the web application.

4.Extension

React supports various extensions for application architecture. It supports server-side rendering, Flux, and Redux extensively in web app development. React Native is a popular framework developed from React for creating cross-compatible mobile apps.

5.Debuging

Testing React apps is easy due to large community support. Even Facebook provides a small browser extension that makes React debugging easier and faster.

Next, let's understand some essential concepts of ReactJS.

○ Hooks in ReactJs

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes.

Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.

When we use hooks:-

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

Rule of Hooks:-

Hooks are similar to JavaScript functions, but you need to follow these two rules when using them. Hooks rule ensures that all the stateful logic in a component is visible in its source code. These rules are:

Only call hooks at top of the level:-

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

Only call hooks reactjs function:-

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

1.Hook Effice:-

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods which are available in class components. In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

- Updating the DOM,
- Fetching and consuming data from a server API,
- Setting up a subscription, etc.

2.Effect without cleneUp:-

It is used in useEffect which does not block the browser from updating the screen. It makes the app more responsive. The most common example of effects which don't require a cleanup are manual DOM mutations, Network requests, Logging, etc.

3.Effect with cleanUp:-

Some effects require cleanup after DOM updation. For example, if we want to set up a subscription to some external data source, it is important to clean up memory so that we don't introduce a memory leak. React performs the cleanup of memory when the component unmounts. However, as we know that, effects run for every render method and not just once. Therefore, React also cleans up effects from the previous render before running the effects next time.

4.Custome Hooks:-

A custom Hook is a JavaScript function. The name of custom Hook starts with "use" which can call other Hooks. A custom Hook is just like a regular function, and the word "use" in the beginning tells that this function follows the rules of Hooks. Building custom Hooks allows you to extract component logic into reusable functions.

○ Reactjs States

The state is an updatable structure that is used to contain data or information about the component. The state in a component can change over time. The change in state over time can happen as a response to user action or system event. A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.

A state must be kept as simple as possible. It can be set by using the **setState()** method and calling setState() method triggers UI updates. A state represents the component's local state or information. It can only be accessed or modified inside the component or by the component directly. To set an initial state before any interaction occurs, we need to use the **getInitialState()** method.

For example, if we have five components that need data or information from the state, then we need to create one container component that will keep the state for all of them.

Defining State:-

To define a state, you have to first declare a default set of values for defining the component's initial state. To do this, add a class constructor which assigns an initial state using this.state. The '**this.state**' property can be rendered inside **render()** method.

Changing the State:-

We can change the component state by using the setState() method and passing a new state object as the argument. Now, create a new method toggleDisplayBio() in the above example and bind this keyword to the toggleDisplayBio() method otherwise we can't access this inside toggleDisplayBio() method.

○ Component life cycle

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and called at various points during a component's life. The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase

Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.

1. Initial Phase

It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

- **getDefaultProps()**
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.

- **getInitialState()**
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

2. Mounting Phase

In this phase, the instance of a component is created and inserted into the DOM. It consists of the following methods.

- **componentWillMount()**
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
- **componentDidMount()**
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
- **render()**
  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

3. Updating Phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

- **componentWillRecieveProps()**
  It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.
- **shouldComponentUpdate()**
  It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.
- **componentWillUpdate()**
  It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method. It will not be called, if **shouldComponentUpdate()** returns false.
- **render()**
  It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number. If

shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.

- **componentDidUpdate()**
  It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

5. Unmounting Phase

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.

- **componentWillUnmount()**
  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.