**TRAINING**

**03/01/25**

**Local repo :** Divided into 2

* Un-tracked files : 1. Create

2. Update

3. Delete

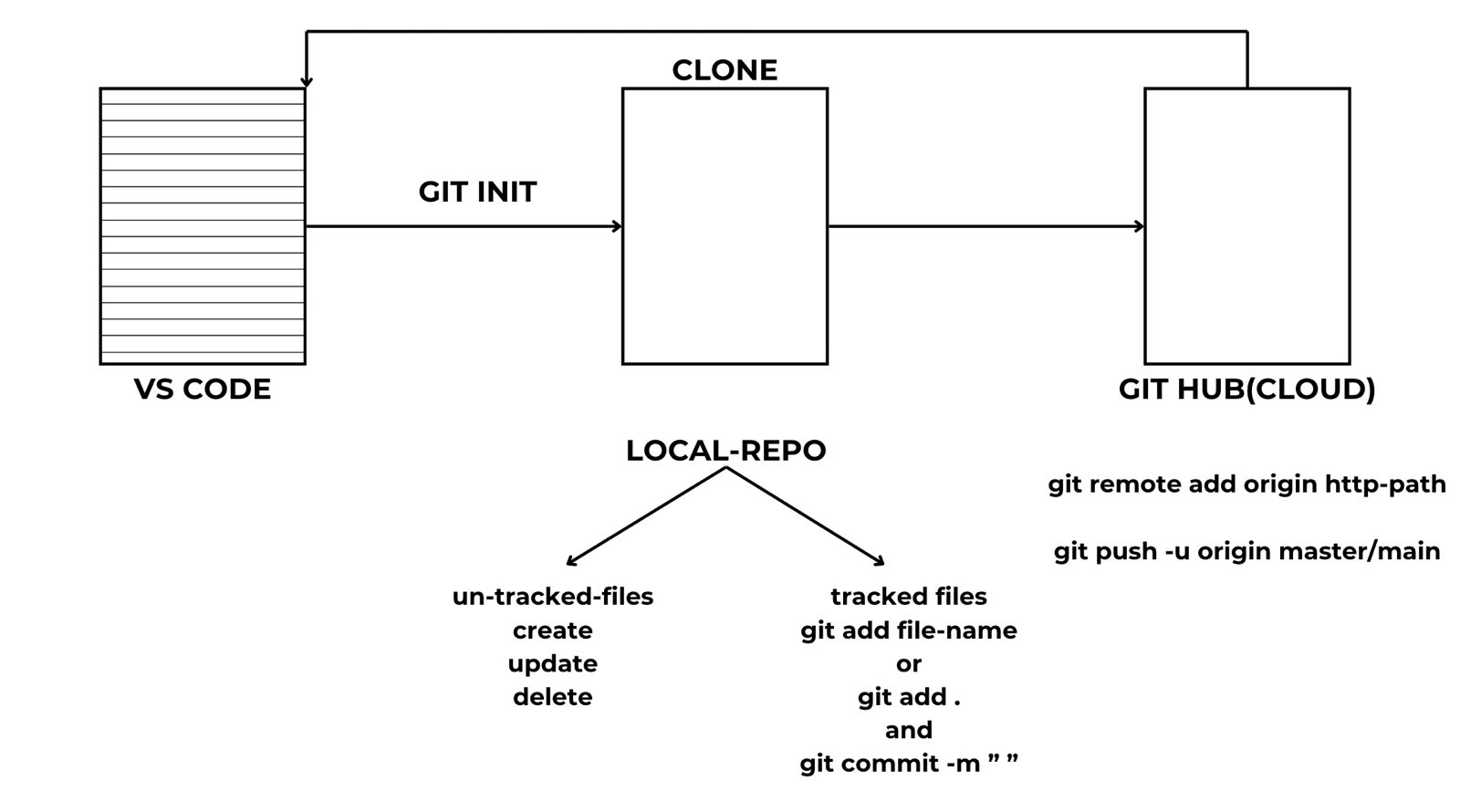
* Tracked files : 1. git add file-name - this command is used to add individual files.

(or) git add . - this command is used to add multiple files.

2. git commit -m ""

**Github (cloud) :**

* **Git remote add origin http-path**
* **Git push –u origin master/main**

****

**04/01/25**

* React native is used to develop Mobile application.
* Electron JS is used to develop Desktop application.
* React JS is used to develop Web application.

**What is JavaScript ?**

* It is used to convert static page into dynamic page.
* It is used to develop dynamic websites.
* It is a dynamically typed language.

**Variables in JS :**

* Set of blocks used to store data/values (data types). (any kind of data)
* Dynamic in nature (because there is no need of declaring any data types).
* They are case sensitive.
* Can start with letters, under score (\_), dollars($).
* Cannot start with numbers.
* Reserved words (keywords) are not allowed.
* In variables, we have 3 types of scopes :

1. **Global scope:** we can declare variables outside the function, and can access inside the function.
2. **Local scope/script scope :** if we declare a variable inside a function.
3. **Block scope** : let is possible, var is not possible.

* Variables can be declared using 3 keywords :

i. var (1995-2015) (used only in old browsers)

ii. let (2015)

iii. const

* There are 2 types of variables :

1. Static type
2. Dynamic type

**06/01/25**

* **Static websites :** Remains same to everyone.

Ex : fb login.

* **Dynamic websites :** Changes to everyone (data).

Ex : Youtube, Cricbuzz, Google maps #time-time#, Instagram.

**Reassigning variables :**

* Reassigning variables is possible by ‘var’ and ‘let’.

Ex 1: var a=1;

      a=5;

      document.write(a+a);

      console.log(a+a);

Output : 10

Ex 2 : var fav\_std = "ramya";

        fav\_std = "kavya";

        fav\_std = "sam";

        document.write(fav\_std);

        console.log(fav\_std);

Output : sam

Ex 3 : let fav\_std = "ramya";

        fav\_std = "kavya";

        fav\_std = "sam";

        document.write(fav\_std);

        console.log(fav\_std);

Output : sam

* Reassigning variables is not possible by ‘const’.
* The latest value assigned to the variable will be the output.

**Redeclaring variables :**

* Redeclaring with ‘var’ is accepted.
* Redeclaring with ‘let’ and ‘const’ is not accepted.

**07/01/25**

**Data types in JS :**

To check the data type - **typeof**

2 types:

1. Primitive Data types:

* Pre-defined.
* We can store single values.
* Ex:

1. Number (integers, floats)
2. Boolean (true or false)
3. String (Stream of characters enclosed in quotes : single, double or backtick quotes)

* Single and double quotes works the same.
* Backtick quotes were introduced in ES6 version.
* It provides extra functionality in which we can insert the variables.
* Ex (1):

let a = 1

console.log("hi +a+a")

document.write('hi + a+a' , "<br>")

document.write(`hi+ ${a+a}`)

console.log(`hi +${a+a}`)

**Output:** hi+a+a  
 hi+ 2

* Ex(2):

var a = "sai"

var b = "kumar"

var age = 30

let c = `I am ${a+” ”+b}, my age is ${age} years old`

console.log(c);

document.write(c);

**Output :** I am saikumar, my age is 30 years old

1. Undefined (declared variable with no value).
2. Null (type of null in JS is object) – empty value or null value

* Ex:

let a = null

document.write(a , "<br")

document.write(typeof(a))

**Output:** null

1. Big int.
2. Non Primitive data types:

* Ex:

1. Class
2. Objects
3. Arrays
4. Functions
5. Maps
6. Sets

**08/01/25**

**JavaScript Operators:**

1. **Arithmetic Operators :**

* Multiplication - \*
* Division - / : gives quotient as the result
* Modulus Operator - % : gives remainder as the result
* Exponential (raised to the power) - \*\*
* ++ : increments by +1
* -- : decrements by -1
* Ex:

let a=4

let b=5

a++

b—

document.write(a, “<br>”)

document.write(a, “<br>”)

document.write(a\*b , "<br>")

document.write(a\*\*b, "<br>")

document.write(a/b, "<br>")

document.write(a%b, "<br>")

Output :

5  
4  
20  
625  
1.25  
1

1. **Assignment Operators:**

* Ex:

let a=5

let b=2

a+=40 //a=a+40 a=5+40=45

b-=5 //b=b-5

document.write(a , "<br>")

document.write(b , "<br>")

Output:

45  
-3

1. **Logical Operators:**

* Ex:

let a=5

let b=2

document.write(a>b && a>2 , "<br>")

document.write(a>b && a>12 , "<br>")

document.write(a>b || a>12 , "<br>")

document.write(!(a>12) , "<br>")

Output:

true  
false  
true  
true

1. **Relational Operators:**

* They are also known as Comparison Operators.
* “==” operator doesn’t check for the data type of both the values.
* “===” operator checks the data type of both the values and gives the output accordingly.
* Ex:

let a=5

let b="5"

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

document.write(a!=b, "<br>")

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

**document.write(a!==b, "<br>")**

Output:

true  
false  
false  
true

1. **Conditional Operators:**

* Ex:

let t = 40

let r = (t>40)?'Ac-off':'Ac-on'

document.write(r, "<br>")

Output:

Ac-on

**Window Methods:**

1. Prompt – by default it is considered as string.
2. Alert

Ex:

//alert("Zoro")

//document.write(alert)

//a=prompt("Enter Your Name")

//alert(a)

let aa = prompt("Enter your name")

alert(`My name is : ${aa}`)

**09/01/25**

**Conditional Statements:**

* To make the set of instructions (block of code)(conditional block) execute only when the given condition is True.
* Used to decide whether the code has to be executed or skip based on the given condition.
* Line-Line(sequence)
* **Block of code:** set of instructions.
* It will execute only when that specific condition is True.
* **Condition:** An expression that evaluates a result(True or False).
* **Ex:** console.log(5>6)
* **Methods:**

1. **If:**

* Only one possible condition.
* If the condition is true, it will execute otherwise it skips.
* The outer block doesn’t look at the condition, it will always execute.
* Ex:

let a=5

let b=10

if(a<b){

document.write("Hi Luffy","<br>")

}

document.write("Welcome to INDIA") //outer block

Output:

Hi Luffy  
Welcome to INDIA

1. **If-else:**

* Ex:

let a=5

let b=10

if(a>b){

document.write("Hi luffy","<br>")

}else{

document.write("Zoro","<br>")

}

document.write("Welcome to INDIA")

1. **Else if:**

* Ex(1):

age = prompt("Enter your age")

name = prompt("Enter your name")

if(age<18){

alert(name+ ", You are too young to marry")

}else if(age>35){

alert(name+ ", You are too old to marry")

}else{

alert(name+ " Perfect age to marry")

}

alert("Welcome to India")

* Ex(2):

let a = prompt("Enter the A score")

let b = prompt("Enter the B score")

if(a>300 || b>300){

alert("You can team up")

}

else if(a+b<500){

alert("You can team up")

}

else{

alert("You can't team up")

}

1. **Switch:**
2. **Ternary operator**

**20/01/25**

**LOOPS:**

* For loop
* For in
* For of
* While
* Do while

1. **For loop:**

Syntax:

For (initialization; condition; update-expression){

---block of code

}

**Ex:**

<script>

        //ex1

        for(let i = 0; i<7; i=i+2){

            document.write(i+2)

        }

        //ex2

        for (let i=0; i<7; i=i+2){

            document.write(i+2)

        }

        //i=0 0<7 True, 0+2 = 2

        //i=2 2<7 True, 2+2 = 4

        //i=4 4<7 True, 4+2 = 6

        //i=6 6<7 True, 6+2 = 8

        //i=8 8<7 False, loop ends

    </script> //OUTPUT : 2468

//ex3

        //for(let i = 10; i<=15; i++){

            //document.write(i, "<br>")

        //}

        //ex4

        //for(let i=1; i<=5; i++){

            //document.write("dakshuu", "<br>")

        //}

        //ex5

        for(let i=1; i<=10; i++){

            document.write(i\*\*2)

        }

        //ex6

        document.write("<br>")

        for(let i=1; i<=10; i++){

            document.write("9 x "+i+"= "+ i\*9, "<br>")

        }

//ex7 adding the numbers in array

        const numbers = [1,2,3,4,5]

        let sum = 0

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

            sum = sum + numbers[i]

        }

        document.write(sum)

        //0+1=1 #0 = 1

        //1+2=3 #1 = 2

        //3+3=6 #2 = 3

        //6+1=4 #3 = 4

        //10+5=15 #4 = 5

**21/01/25**

1. **For in loop:**

Ex:

//ex1

        const a = "dakshita"

        for (let i in a){

            console.log(a[i])

            document.write(a[i])

        }

        //ex2

        const s = "itachi"

        for(let i in s){

            console.log(s[i])

        }

1. **For of loop:**

Ex:

const arr = ["itachi","obito","shisui"]

        for (let e of arr){

            console.log(e)

        }

1. **While loop:**

* We use while loop when we don’t know the number of iterations.
* Syntax:

While(condition){

//block of code

}

* Ex:

//while loop

let i=2, n=8

while(i<=n){

console.log(i+1)

i=i+2

}

i++;

1. **Do while loop:**

* Syntax:

Do{

//block of code

}

While(condition){}

* Even when the condition is false, the while loop will execute atleast once.
* Ex:

//do while loop

let c=0

do{

console.log("count is:" + c)

c++

while(c>5)

}

//do while loop

        //let c=0

        //do{

            //console.log("count is:" + c)

            //c++

            //while(c>5)

        //}

        for(let i=1; i<=8; i=i+2){

            if(i==4){

                break;

            }

            console.log(i)

        }

        for(let i=1; i<=8; i=i+2){

            if(i>=4){

                break;

            }

            console.log(i)

        }

//ex4

for(let i=1; i<=8; i=i+1){

            if(i>=4){

                continue;

            }

            console.log(i)

        }

**22/01/25**

Ex:

//ex1

for(let i=1; i<=8; i=i+1){

if(i==5){

break;

       }

       console.log(i)

}

//ex2

for(let i=1; i<=8; i=i+2){

            if(i>5){

                continue;

            }

            console.log(i)

        }

        //Output: 1 3 5

**Nested Loops:**

* A loop inside another loop is called as Nested Loops.
* The inner loop will execute one time for each iteration of outer loop.
* An inner loop within the repeating block of outer loop.
* Ex:

//outer loop

for(let i=1; i<=3; i++){

document.write(i, "<br>")

//inner loop

for (let j=1; j<=4; j++){

document.write(j, " ")

}

document.write("<br>")

}

Ex 2: Printing patterns

//right angle pattern

        for(let i=1;i<=5;i++){

            for(let j=1;j<=i;j++){

                document.write("\*")

            }

            document.write("<br>")

        }

//triangle pattern

        let n=5;

        for(let i=1; i<=n; i++){

            for(let j=1; j<=n-i; j++){

                document.write("&nbsp;&nbsp;")

            }

            for(let j=1;j<=2\*i-1;j++){

                document.write("\*")

            }

            document.write("<br>")

        }

**23/01/25**

Topics to be covered:

* Functions
* DOM
* Events
* Mini project
* Major project
* HTML, CSS, JS, React JS
* JAVA, Spring Boot, My SQL
* Web applications
* Front-end
* Database, JAVA

**Functions:**

* A function is a reusable block of code.
* It can be called anywhere in the program.
* **Reusable code:** using an existing code without writing it ever we need.
* You can use the same code with different arguments to get the results.
* Syntax for writing the function in JS:

Keyword = function

()

Parameters = value

(Parameters are the values which are declared inside the parenthesis while defining a function)

* **Arguments** are the values which are passed through the parameters.

Ex:

        //ex1 without parameters

        function greet(){

            console.log("Hello Dakshita")

        }

//output: Hello Dakshita

        //ex2

        greet()

        console.log("Hello Dakshuu")

        greet()

//output: Hello Dakshita

Hello Dakshuu

Hello Dakshita

//ex3 with parameters

        function g(name){ //name is a parameter here

            console.log("Hello " + name)

        }

        let name = prompt("Enter your name")

        g(name)

//EX4 adding 2 numbers using functions

        function sum(number1, number2){ //name is a parameter here

            console.log(number1+number2)

        }

        sum(10,20)

**Function Return:**

* The return statement can be used to return the value to function call.
* Ex:

Function a(aa,bb){

return aa+bb;

}

//ex5

        function sumtwo(a,b){

            return a+b

        }

        let n1 = parseFloat(prompt("Enter number1"))

        let n2 = parseFloat(prompt("Enter number2"))

        let result = sumtwo(n1,n2)

        console.log("The sum is: " + result)

**Anonymous Function:**

* A function without name, after we create a function without a name and we assign it to a variable.
* Ex:

//ex6

sum = function(n1, n2){

let r = n1+n2

return r

}

console.log(sum(5,10))

**24/01/25**

**Arrow Functions:**

* ES6 version
* More readable and more structured.
* Aka Anonymous functions (or lambda functions).
* Here we don’t write the function name but they are assigned to a variable.
* Ex:

//EX1 arrow functions without parameters

let C=()=>{

console.log("Hello world")

}

C()

//EX2 arrow functions with parameters

        let a = (x,y)=>{

            console.log(x+y)

        }

        a(20,30)

//EX3 arrow functions with one parameter

        let greet=x=>{

            console.log(x)

        }

        greet('HELLO')

//EX4 arrow functions with no argument

        let greetings=()=>{

console.log("Hello")

        }

        greetings()

//EX5 arrow functions as an expression

        let age=18;

        let welcome=(age<18) ?

        ()=>console.log("Hi itachi"):

        ()=>console.log("Hi Obito");

        welcome()

**Synchronous and Asynchronous Functions:**

**Synchronous Functions:**

* They get executed step by step.
* Ex:

//synhronous functions

function task1(){

console.log("Task1: Start")

}

function task2(){

console.log("Task2: Start")

}

function task3(){

console.log("Task3: Start")

}

task1()

task2()

task3()

**Asynchronous Functions:**

* Allows to execute multiple tasks at a time.
* It won’t wait for the remaining tasks to get completed.
* Ex:

//asynchronous function

console.log("Hello Dakshita")

setTimeout(()=>{

console.log("Hi i am waiting")

},6000);

console.log("I am in lab")

**JavaScript Call back Functions:**

* A call back function is a function that is passes arguments to another function.
* **Execution:** The function receiving the callback will execute the callback at some point during execution (often at the end or after an asynchronous operation is done).
* Ex:

function greet(name,callback){

console.log(`Hello,${name}!`)

callback();

//this calls the callback function after greeting

}

function sayGoodbye(){

console.log("Goodbye")

}

//call greet and pass sayGoodbye as the callback

greet("Sam",sayGoodbye)

//ex2 25jan

        function mainfunction(callback){

            console.log("Executing main function");

            callback()

            //here we call the callback function

        }

        function callbackFunction(){

            console.log("Callback function executed");

        }

        mainfunction(callbackFunction);

**Exp:**

* mainfunction takes a parameter callback
* callbackFunction is passed as an argument to mainfunction
* it will execute the code inside the mainfunction
* it will print(“Executing main function”)
* then it reaches to callback(); where it calls the function which is passed to the mainfunction(callbackFunction)

//ex3

        //passing a Named function as a callback

        function greeting(name, callback){

            console.log("Hello " +  name);

            callback();

            //executes the callback function greeting

        }

        function goodbye(){

            console.log("Goodbye")

        }

        greeting("Dakshita", goodbye)

//ex4

        //using an anonymous function (function without a name) as a callback

        greets("Obito", function(){

            console.log("Goodbye!");

        })

        //anonymous function as function

//ex5

        //asynchronous callback with setTimeout

        function fetchData(callback){

            console.log("Fetching data...")

            setTimeout(function(){

                console.log("Data Fetched");

                callback();

            },2000);

        }

        function processData(){

            console.log("Processing the fetched data..");

        }

        fetchData(processData);

//ex6

        //using a callback with array methods(map)

        const numbers = [1,2,3,4,5]

        const d = numbers.map(function(num) {

            return num \* 2

        })

        console.log(d)

        //1. map() is a built in array method that callback function

        //to process each item in the array

        //2. the callback here doubles each number (num\*2) and will result a

        //new array with doubled values

//ex7

        function registerUser(username, email, callback){

            console.log("registering user"+username);

            setTimeout(function(){

                console.log("user"+username+"registered successfully");

                callback(email);

            },2000);

        }

        function sendWelcomeEmail(email){

            console.log("sending welcome email to"+email);

        };

        registerUser("Itachi","itachi123@gmail.com, sendWelcomeEmail");

**Promises:**

* They are mainly used to check eventual completions (whether the event is completed or not).
* It has 3 states:

1. **Pending:** Still not completed.
2. **Fulfilled:** Task completed.
3. **Rejected:** The operation is failed.

* Ex:

let myPromise = new Promise((resolve,reject)=>

{

let success = true

if(success){

resolve("Task completed")

}else{

reject("Task failed")

}

})

**27/01/25**

**PROMISE:**

* A promise is a special javascript object that represents the eventual completion (or failure) of an asynchronous operation.
* It allows to handle asynchronous tasks more efficiently than callbacks.
* There are 3 states in promise:

1. **Pending:** The initial state, the promise is still waiting for the operation to finish.
2. **Fulfilled:** The operation completed successfully.
3. **Rejected:** The operation failed.

* Explanation:
* To create a promise – new Promise()

Two parameters:

**resolve():** To mark the promise as successfully.

**reject():** Mark it as failed.

* Handling the result:

**.then():**

-This method is called when promise is fulfilled.

-It will receive the result from resolve().

**.catch():**

-This method is called if the promise is rejected.

-It receives the error message from reject().

-Example: let myPromise= new Promise((resolve,reject)=>{

let success=true;

if(success){

resolve("The operation is successfull.");

}else{

reject("The operation is failed.");

}

});

myPromise.then(result=>{

console.log(result);

}).catch(error=>{

console.log(error);

});

**Changing Promises:**

**-**Example: let promise=new Promise((resolve,reject)=>{

resolve(5)

//intial resolve value is 5.

})

promise.then(result=>{

console.log(result)

return result\*2

//returns a new value

}).then(result=>{

console.log(result)

return result+3

//returns a new value

}).then(result=>{

console.log(result);

}).catch(error=>{

console.log("Error",error)

})

**Promise with setTimeOut:**

-Asynchronous task using setTimeOut, and the promise will resolve after a certain amount of time.

-Example: let fetchData=new Promise((resolve,reject)=>{

let dataFetched=true

setTimeout(()=>{

if(dataFetched=true){

resolve("Data fetched successfully.")

}else{

reject("Data fetched failed.")

}

},2000)

})

fetchData.then(result=>{

console.log(result)

}).catch(error=>{

console.log("Error",error)

})

**27/01/25**

**DOM (Document Object Model):**

* It is mainly used in web development used to interact and manipulate (change) the html documents.
* When the web page is loaded, the browser creates the DOM of the web page.
* When the html file is loaded into the browser, the JavaScript cannot understand the html document directly.
* **DOM:** It is basically the representation of the same html document but in a tree-like structure composed of object.
* **Document is the entry point of DOM.**

Key Points:

* **Structure:** The DOM represents the structure of html document as a tree. Each node is tree represents the elements, an attribute or text.
* **Manipulation:** Using JavaScript, you can interact with elements (buttons, paragraphs) on the webpage, changing the content, styles, can ass or remove elements.
* **Event Handling:** You can handle the interactions like click or pressing the button.
* **W3C (World Wide Web Consortium):** According to W3C we need to build the webpage.

Methods of Document Object:

Used to interact with and manipulate the document.

**1.Getting Element:**

* Getting an Element by Id:
* Getting an Element by Class:
* Getting an Element by Tag name:
* Getting an Element by Query Selector

**29-01-2025**

* Getting an Element by Id:

**-**It uses the **getElementById()** method.

* Getting an Element by Class name:

**-**It uses the **getElementsByCLassName()** method.

-returns all the elements having the same class name.

-It returns an HTML collection object.

-A HTML Collection is an array of html elements/list of html elements.

* Getting an Element by Tag name:

-Is uses the **getElementsByTagName()** method.

* Getting an Element by Query Selector:

-The document query selector method selects a HTML elements by tag name, by id, by class name

-**QuerySelector():** It can be

-**QuerySelectorAll():** It selects all the html elements by its class name and tag name.

**Manipulating Elements:**

**To create HTML elements:**

**30-01-2025**

**To create multiple elements:**

-**Example 1:**<script>

const titles=['sai','sam','rahul']

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

let title=document.createElement("h1")

title.className="a"

title.textContent=titles[i]

console.log(title)

}

</script>

-**Example 2:**<script>

let title

for(let i=0;i<3;i++){

title=document.createElement("h2")

title.className="tt"

title.textContent=i

console.log(title)

}

</script>

**Removing a child element from a parent node:**

-**Example:**<h1>Removing Child Node</h1>

<ul>

<li>Sai</li>

<li>Sam</li>

<li>Sanvi</li>

</ul>

<script>

const ul=document.querySelector('ul')

const lists=document.querySelectorAll('li')

for(const list of lists){

console.log(`Removing ${list.textContent}`)

ul.removeChild(list)

}

</script>

**To append a new element:**

**-append():** To append new element to the parent node.

**-Example:**<div id="a">

<h1>sia</h1>

<h1>ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.createElement("h2")

n.textContent="Sai is the team leader"

//get the parent element

let p=document.getElementById("b")

console.log(p.innerText)

//To append new element to the parent node

p.appendChild(n)

//log the inner text of the parent

console.log(p)

console.log(p.innerText)

</script>

**31-01-2025**

**InsertBefore():**

**-**It inserts a new node before an existing node or child node of a specific parent node.

-This requires a reference to both and the existing node you want to insert before.

-**Example 1:**<div id="a">

<h1>sia</h1>

<h1 id="r">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.createElement("h2")

n.textContent="Rahul"

let p=document.getElementById('a')

let l=document.getElementById('r')

p.insertBefore(n,l)

console.log(p)

</script>

-**Example 2:**<div id="a">

<h1 id="c">sia</h1>

<h1 id="r">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let nc=document.createElement("h1")

nc.innerText="Chandana"

let pc=document.getElementById('a')

let lc=document.getElementById('c')

pc.insertBefore(nc,lc)

console.log(pc)

let n=document.createElement("h2")

n.innerText="Rahul"

let p=document.getElementById('a')

let l=document.getElementById('r')

p.insertBefore(n,l)

console.log(p)

</script>

**To remove an element:**

**-Example:** let b=documnet.getElementById("b")

b.remove()

**To remove a child element:**

-**Example:** **:**<div id="a">

<h1 id="c">sia</h1>

<h1 id="re">ria</h1>

</div>

<div id="b">

<h1>ram</h1>

<h1>sam</h1>

</div>

<script>

let n=document.getElementById("a")

let l=document.getElementById("re")

console.log(l)

n.removeChild(l)

</script>

**To replace an child element:**

-**Example:** let n=document.createElement("h1")

n.innerText="Version it"

let p=document.getElementById("a")

p.replaceChilderen(n)

console.log(p)

**03-02-2025**

* **React.js:**

-JavaScript library.

-Single page application.

-Advantages:

* All the updates, components, update specify, update, refresh.

-Example: Amazon

**04-02-2025**

**To create an element by react.js:**

-React.createElement(“hi”)

-Syntax:

* React.createElement(type, prop)

-Type: tag name(div,h1,p)

-props: classname, id, onClick.

**To display the element:**

**-**ReactDOM.createRoot()

-ReactDOM.render()

-Syntax:

-Reactelemnet: what to render

-Container: where to render

* **Intro to JSX:**

-JavaScript XML (JSX) is a syntax extension for JS in react.js.

-It allows us to write html code in React.

-Make us easier to write html in react.

* + **JSX:** JSX code gets complied into JS.
  + **Babel:** A tool converts html code into JS.
    - While using babel all the html tags must be closed.

**With JSX:**

-Example: <div id="app"></div>

<script type="text/babel">

const element=<h1 classname="a">Hello JSX!</h1>

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(element)

</script>

**Without JSX:**

-Example:<div id="app"></div>

<script type="text/babel">

const element=React.createElement('h1',{classname:"a"},"Hello JSX!")

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(element)

</script>

**Using a call function:**

-You can call the function call directly within the JSX expression.

-Example: <div id="a"></div>

<script type="text/babel">

function getGreeting(name){

return `Hello,${name}!Welcome to React`

}

function Greeting(){

const name="Sam"

return <h1>{getGreeting(name)}</h1>

}

const root=ReactDOM.createRoot(document.getElementById('a'));

root.render(</Greeting>)

</script>

**05-02-2025**

**Expressions:**

Ex: <div id="app"></div>

<script type="text/babel">

const n=10;

const m=<h2>React is {n\*2} times better than jsx</h2>

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(m)

</script>

**Conditional Rendering:**

Ex:<div id="app"></div>

<script type="text/babel">

const App=()=>{

const isLoggedIn=false;

return(

<div>

{isLoggedIn?(<h1>Welcome back user!</h1>):(<h1>Please log in.</h1>)}

{isLoggedIn &&<h2>You have new notification</h2>}

</div>

);

};

const root=ReactDOM.createRoot(document.getElementById('app'))

root.render(<App/>)

</script>

**07-02-2025**

**Conditional Rendering:**

**Example (1):**

**Greeting.js**

import React from 'react';

function Greeting({isLoggedIn}){

    if(isLoggedIn){

        return <h1>Welcome Back!!</h1>

    }else{

        return <h1>Please Login</h1>

    }

}

export default Greeting;

**App.js**

import React from 'react'

import Greeting from './Greeting';

function App(){

  const isLoggedIn = true;

  return(

    <div className = "App">

      <Greeting isLoggedIn = {isLoggedIn}></Greeting>

    </div>

  )

}

export default App;

**Lists and Keys:**

* In React, a **list** is a collection of items you want to show on screen.
* **Keys** in React helps us to keep the track of items.
* By using these keys, React knows which items you have changed.
* Example:

**App.js**

import React from 'react';

const FruitList=()=>{

    const fruits = ['Apple','Cherry','Orange'];

    return(

        <div>

            <h1>Fruits List</h1>

            <u1>

                {fruits.map((fruits,index)=>(

                    <li key = {index}>{fruits}</li>

                ))}

            </u1>

        </div>

    )

}

export default FruitList;

**Maps():**

* Elements in an array.
* Function will be applied to all the elements.
* Example:

Const n = [1,2,3,4,5]

Const d = n.map(num =>num\*2)

Console.log(d)

**07-02-2025**

**A HTML, CSS & JS CODE FOR BULB ON & OFF**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

    <link rel="stylesheet" href="bulb.css">

</head>

<body id="body">

    <img src="https://d1tgh8fmlzexmh.cloudfront.net/ccbp-dynamic-webapps/bulb-go-off-img.png" alt="" class="light-bulb" id="bulbImage">

    <button class="button" id="toggleButton">Turn on</button>

    <script>

        let isOn=false;

        function toggleLight(){

            const body= document.getElementById('body');

            const bulbImage= document.getElementById('bulbImage');

            const toggleButton= document.getElementById('toggleButton');

            isOn=!isOn;

            if(isOn){

                body.style.backgroundColor='lightyellow';

                bulbImage.src='https://d1tgh8fmlzexmh.cloudfront.net/ccbp-dynamic-webapps/bulb-go-on-img.png';

                toggleButton.innerText='Turn off';

            }else{

                body.style.backgroundColor='black';

                bulbImage.src='https://d1tgh8fmlzexmh.cloudfront.net/ccbp-dynamic-webapps/bulb-go-off-img.png';

                toggleButton.innerText='Turn on';

            }

        }

        document.getElementById('toggleButton').addEventListener('click',toggleLight);

    </script>

</body>

</html>

**CSS:**

body{

    display:flex;

    flex-direction:column;

    align-items:center;

    height : 100vh;

}

.light-bulb{

    width: 150px;

}

.button{

    margin-top: 20px;

    padding: 10 px 20 px;

    font-size: 18px;

    cursor: pointer;

}

**State:**

**State in React:**

* In react, state is like a container that holds the data or information for a component.
* This data can change overtime based on user actions or events.
* State is an essential part in react because it allows components to be dynamic, interactive and capable of responding to user input or change overtime.

**Why state is important?**

* It allows the components to remember everything.
* **Example:** If you click a button to change a colour, the state will store & hold the colour and show on the screen.

**How to use States in Functional Components?**

* Syntax:

const[statevariable, setstatefunction] = useState(initialvalue)

**statevariable:** holds the current state(ex: name, colour)

**setstate function:** a function which is used to update the state

**initial value:** the initial value of the state variable when the component first renders.

**10-02-2025**

**React Hooks:**

* Introduced in react 16.8 version.
* To manage state and life cycle features in functional components.

**Implementation of States in React:**

1. **useStateHook:**

**useState** allows you to add state to functional components.

**Syntax:**

const [state,setState] = useState(initial value)

**state:** This is current state

**setState:** This is the function to update the state.

**Initial value:** The value you want to set as the initial value.

**Example(1): App.js**

import React, {useState} from 'react';

//counter components

const Counter=()=>{

    const[count, setCount]=useState(0);

    //current state = count

    //function to update state = setCount

    //useState(): it is used to initialize the state = 0

    return(

        <div>

            <h1>{count}</h1>

            <button onClick={() => setCount(count+1)}>Increment</button>

            <button onClick={() => setCount(count-1)}>Decrement</button>

        </div>

    );

};

export default Counter;

**Ex(2):**

import React, {useState} from 'react';

const ThemeToggler =()=>{

    const[theme,setTheme]=useState('light');

    const toggleTheme=()=>{

        setTheme(prevTheme=>(prevTheme=='light'?'dark':'light'))

    }

    return(

        <div style={{backgroundColor:theme=='light'?'white':'black', color:theme=='light'?'black':'white', textAlign:'center'}}>

            <h1> The Current Theme is {theme}</h1>

            <button onClick={toggleTheme}> toggleTheme</button>

        </div>

    )

}

Export default ThemeToggler;

**11-02-2025**

**ASSIGNMENT 1 (TASK 2)**

ProfileCard.js:

import React from "react";

function ProfileCard(props){

    const{name, greeting, Imgsrc} = props;

    return(

        <div className="profile-card">

            <img src={Imgsrc} alt='profile'></img>

            <h1>{greeting}, {name}</h1>

        </div>

    )

}

export default ProfileCard;

App.js:

import React, { useState } from 'react';

import ProfileCard from './ProfileCard';

function App() {

    return (

        <div>

            <ProfileCard

                Imgsrc = 'https://media.istockphoto.com/id/517188688/photo/mountain-landscape.jpg?s=1024x1024&w=0&k=20&c=z8\_rWaI8x4zApNEEG9DnWlGXyDIXe-OmsAyQ5fGPVV8='

                name='Dakshita'

                greeting='Hello'>

            </ProfileCard>

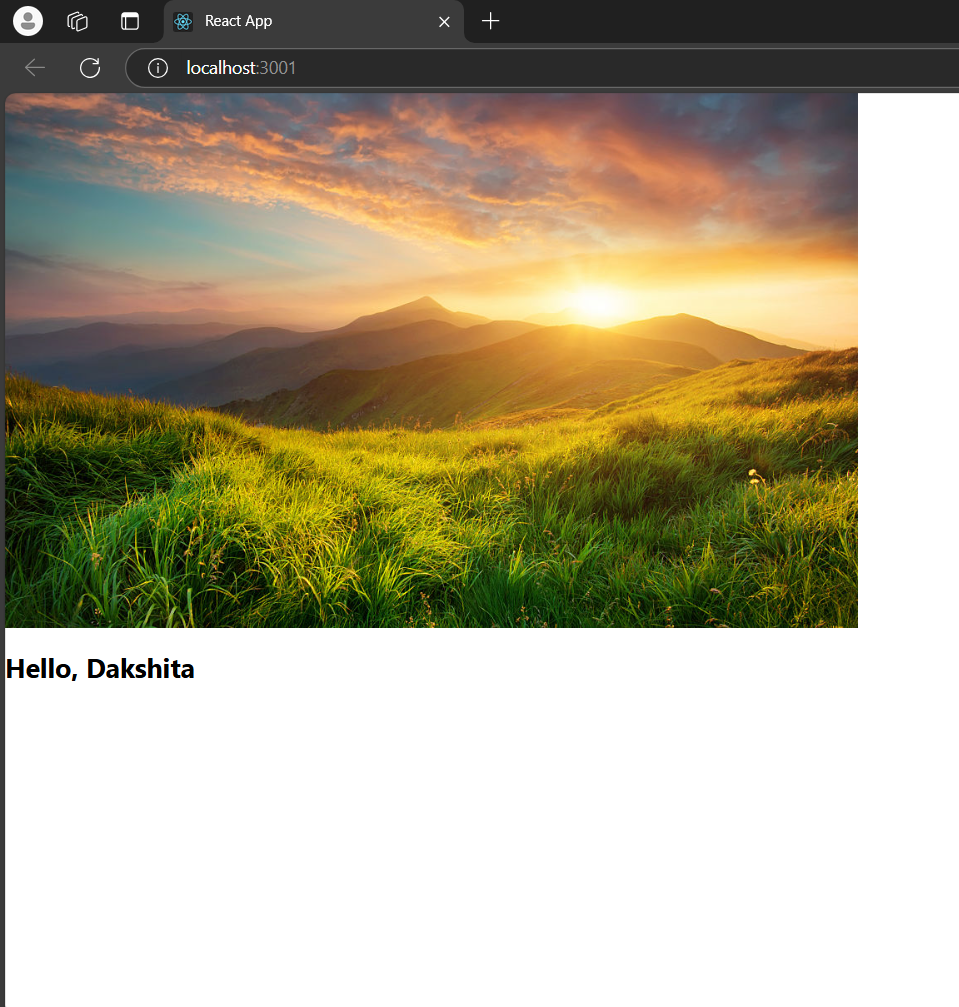
        </div>

    )

}

export default App;

Output:



**12-02-2025**

**React Memo:**

* It is a HOC (High Order Component).
* It is not a React Hook.
* It will stop the unnecessary rendering of the functional component.
* It will improve the performance of the functional components.
* **Example:**

**App.js:**

import {useState} from 'react';

import ChildA from "./ChildA";

import ChildB from "./ChildB";

const Parent =()=>{

    const[count, setCount] = useState(0);

    const increment =()=>{

        setCount(c=>c+1);

    };

    return(

        <div>

            <ChildA/>

            <ChildB count={count} increment={increment}/>

        </div>

    );

};

export default Parent;

**ChildA.js: (without memo)**

const ChildA=()=>{

    console.log("Child A rendered")

    return <h2> This is Child A</h2>

};

export default ChildA;

**ChildB.js: (without memo)**

const ChildB=({count, increment})=>{

    console.log("Child B is rendered");

    return(

        <div>

            <h1>Count:{count}</h1>

            <button onClick={increment}>Increment</button>

        </div>

    );

};

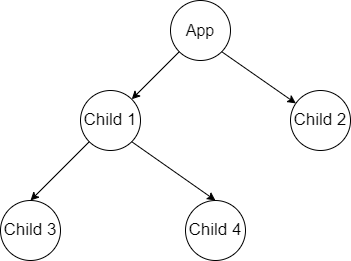
export default ChildB;

* Whenever we click on the button (Increment), the ChildA and ChildB gets rendered because their parent is rendered.

**13-02-2025**

**useContext():**

* **Props (properties) in React:** are used to pass data from parent component to child component.
* **Prop drilling** is the process of passing data from parent component to deeply (nested) child component through intermediate components, by passing data as props at each level.



* **Example:**

**App.js**

function App(){

    const data = "Hello. I am Dakshita";

    return(

        <div>

            <User1 data = {data}/>

        </div>

    );

};

//user1 component

function User1({data}){

    return <User2 data = {data}/>

};

//user2 component

function User2({data}){

    return <User3 data = {data}/>

};

//user3 component

function User3({data}){

    return <User4 data = {data}/>

};

//user4 component

function User4({data}){

    return <div>{data}</div>

}

export default App;

**Alternate code for the above example (using useContext()):**

import React, {createContext, useContext} from 'react';

//create the context to hold the data

const DataContext = createContext()

function App(){

    //The data we want to share with user4

    const data = "Hello. I am Dakshita";

    return(

        <DataContext.Provider value={data}>

            <User1/>

        </DataContext.Provider>

    );

};

//user1 component

function User1({data}){

    return<User2/>

};

//user2 component

function User2({data}){

    return<User3/>

};

//user3 component

function User3({data}){

    return<User4/>

};

//user4 component

function User4(){

    const data = useContext(DataContext)

    return <div>{data}</div>

}

export default App;

**ASSIGNMENT ON REACT CONTEXT:**

import React, { createContext, useContext } from 'react';

const DataContext = createContext();

function App() {

    const data = "Dakshita";

    const data1 = "2211cs010138@mallareddyuniversity.ac.in";

    return (

        <DataContext.Provider value={{ data, data1 }}>

            <Profile1 />

        </DataContext.Provider>

    );

}

// Profile1 component

function Profile1() {

    return <Profile2 />;

}

// Profile2 component

function Profile2() {

    return <Profile3 />;

}

// Profile3 component

function Profile3() {

    return <Profile4 />;

}

// Profile4

function Profile4() {

    const { data, data1 } = useContext(DataContext);

    return (

        <div>

            <h1>{data}</h1>

            <p>{data1}</p>

        </div>

    );

}

export default App;

**14-02-2025**

**React Reducer():**

* A Reducer is a function that helps us to update (like score, a list) based on actions we take.
* **currentState():** The current value of a state.
* Action:
* **Example:**
* Action: add or remove an item
* State: initial value (3 items)

Ex: Tracking apples

import React, {useReducer} from 'react';

//reducer function to track the number of apples

function appleReducer(state=3, action){

    switch(action.type){

        case 'ADD\_APPLE':

            return state+1;

        case 'REMOVE\_APPLE':

            return state-1;

        default:

            return state;

    }

}

function App(){

    const[state, dispatch] = useReducer(appleReducer, 3);

    return(

        <div>

            <h1>APPLE COUNTER</h1>

            <p>Number of Apples: {state}</p>

            <button onClick={()=>dispatch({type:'ADD\_APPLE'})}>ADD APPLE</button>

            <button onClick={()=>dispatch({type:'REMOVE\_APPLE'})}>REMOVE APPLE</button>

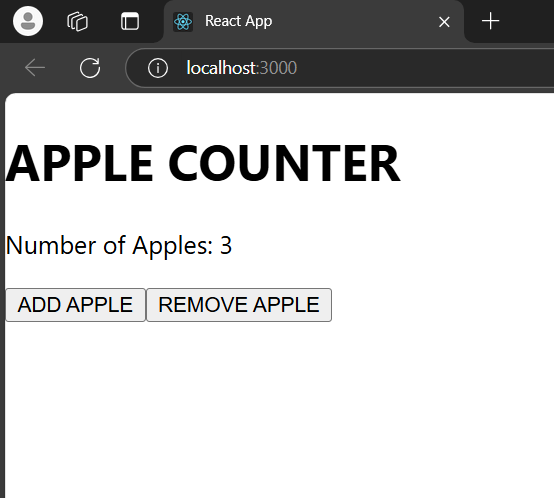
        </div>

    )

}

export default App;

Output:



With CSS:

<div style={{padding:'30px', display:'flex', flexDirection:'column', alignItems:'center', justifyContent:'center'}}>

            <h1>APPLE COUNTER</h1>

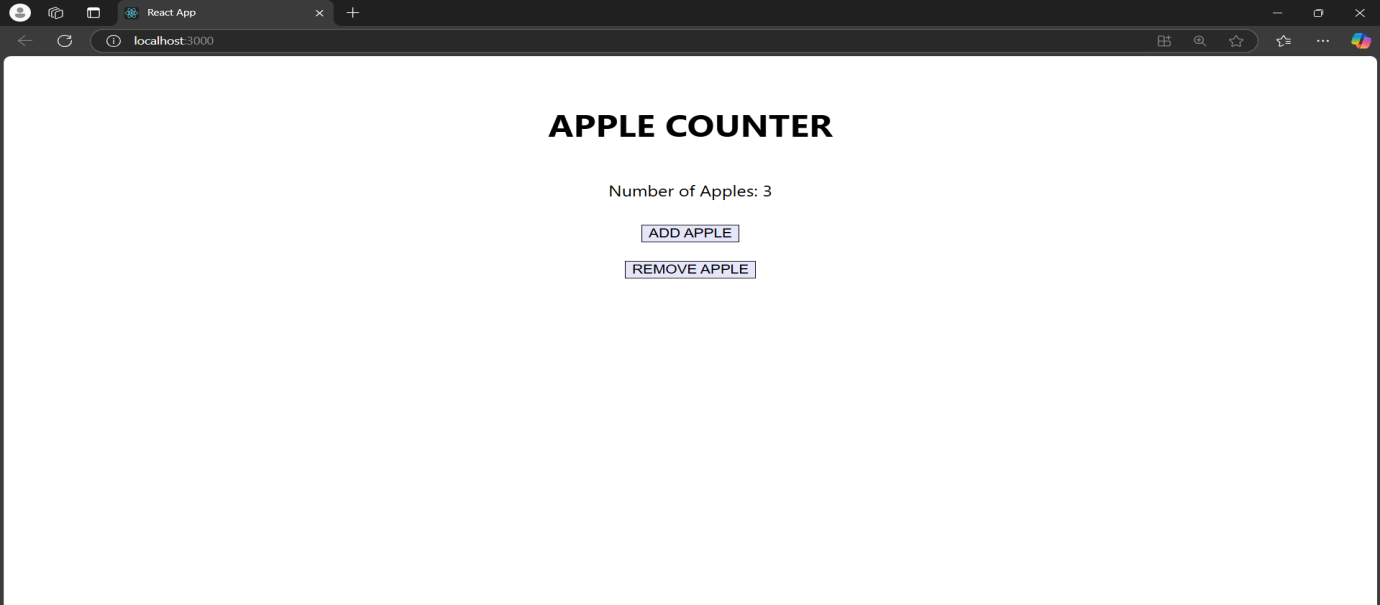
            <p>Number of Apples: {state}</p>

            <button style={{backgroundColor:'lavender', border:'solid 1px', margin:'10px'}} onClick={()=>dispatch({type:'ADD\_APPLE'})}>ADD APPLE</button>

            <button style={{backgroundColor:'lavender', border:'solid 1px', margin:'10px'}} onClick={()=>dispatch({type:'REMOVE\_APPLE'})}>REMOVE APPLE</button>

        </div>

Output:



**Callback in React:**

* In react, a call back is a function that is passed as an argument to another function.
* Call backs are often used to communicate with different components.
* Example: A parent component passing as Call back to child component.
* Parent component displays a message; and child component has a button.
* When the button is clicked, the child will use call back and tells the parent component to update.

App.js:

import React, {useState} from 'react';

import Child from './Child';

const Parent=()=>{

    const[message, setMessage] = useState('Hello World');

    const changeMessage=()=>{

        setMessage('You clicked the button');

    };

    return(

        <div>

            <h1>{message}</h1>

            {/\* passing the change message function as a prop to the child\*/}

            <Child changeMessage={changeMessage}></Child>

        </div>

    )

}

export default Parent;

Child.js:

import React from 'react';

const Child=({changeMessage})=>{

    return(

        <button onClick={changeMessage}>Click Me!!</button>

    )

}

export default Child;