git-hub(cloud)

local repo----untracked files , tracked files

un-t-->create update delete

tracked ---->git add file-name or git add . and git commit -m

repository-> Folder

git remote add origin http-path---> Establishing the path in the local repo

git push -u origin master/main--->here main is the default

origin is nothing but git-hub

--------------------------------------------------------------------------------------------------------

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

DEVELOP THE DYNAMIC WEBSITES

Static is fixed and remains same for everyone

Ex: fb login.

Dynamic changes from person to person based on the conditions of the user

Ex: youtube, maps, Instagram etc…

DATA TYPES

* PRIMITIVE DATA TYPE :

Num, string, Boolean, undefined, bigint

* NON PRIMITIVE DATA TYPE :

Class, object, array, functions, map, sets

In JS variables are dynamic in nature, which means no need of declaring the data type while assigning a value to variable.

In java variables are static.

In JS we have SCOPE concept

**Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to JavaScript**

They are of 3 types:

Global scope – declare outside and access it in inside the block

Local scope/ Script scope - Variables declared within a JavaScript function

Block scope - Variables declared inside a { } block cannot be accessed from outside the block

IN JS DATA TYPES THEY ARE DIVIDED INTO 3 :

VAR

LET

CONST

First preference for “const” then “let” then “var”

In const reinitialization and redeclaration is not possible

In let reinitialization is possible

In var both are possible

**06-01-2025**

**VARIABLES:**

* Static typed and dynamic types
* They are used to store the data(any kind of data)
* Case – sensitive
* Allow letters ,\_ $(dollar sign) [variables can be started by using these]
* Variables cant start with numbers
* Reserved words(Key words) are also not allowed

How to declare variables?

Var keyword (1995-2015)

Let keyword 2015

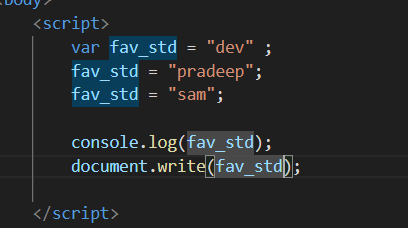
Var used only in old browsers

1. Reassigning variables is possible by var

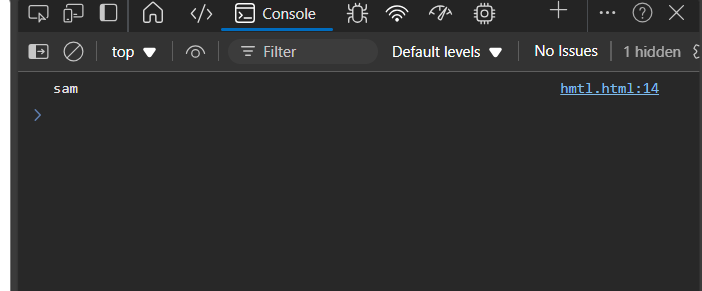
It will be taking latest assigned values

1. Reassigning variables is possible by let
2. Reassigning variables is not possible by const

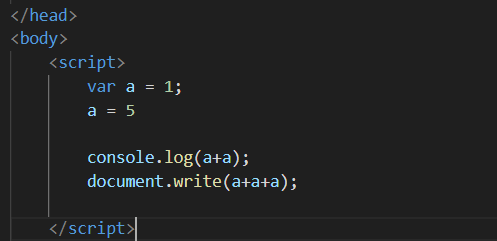
VAR :



OUTPUT :



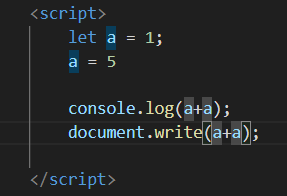
Ex 2:



OUTPUT :



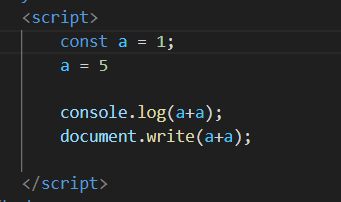
Reassigning with LET keyword :



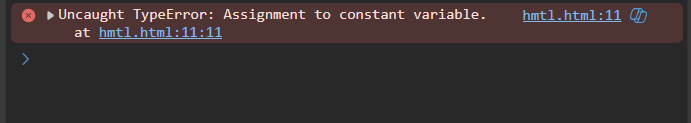
Output:



Reassigning with CONST :

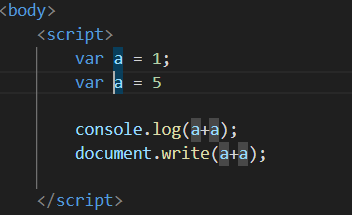


Output :



REDECLARING :

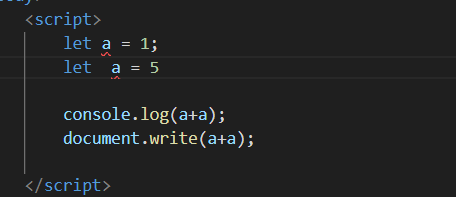
VAR :



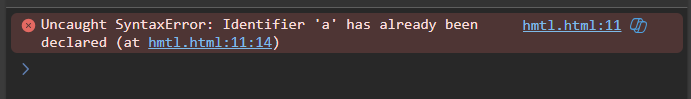
OUTPUT :



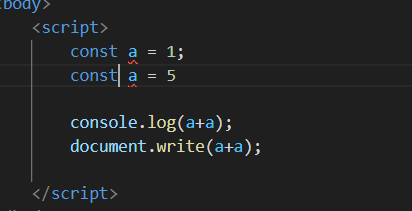
LET :



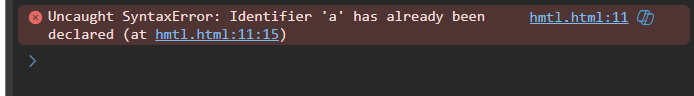
Output :



CONST :



Output :



Here ,

Redeclaration with var is accepted

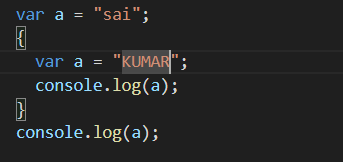
Redeclaration with let is not accepted

Redeclaration with const is not accepted

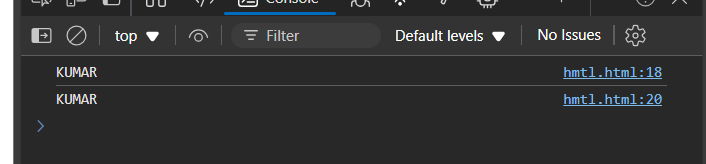
**BLOCK SCOPE :**

Variables declared with

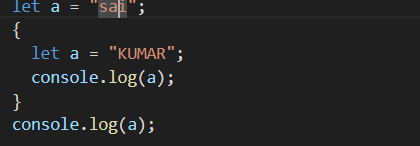
VAR :



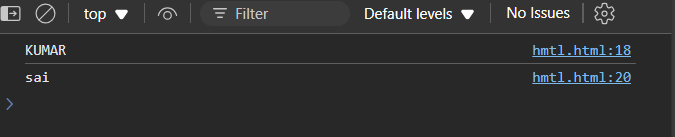
Output:



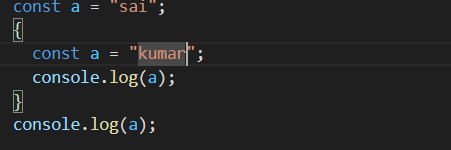
LET :



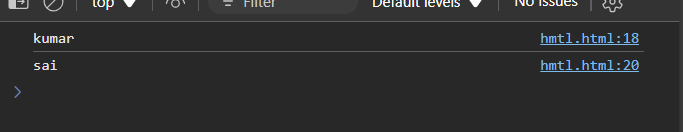
Output :



CONST :l



OUTPUT :



**07-01-2025**

**Two types of data types in JS :**

* Primitive
* Non-primitive : objects, arrays

**TO CHECK THE DATA TYPE – TYPE OF:**

1. Primitive data type : predefined : we can store single values
2. Numbers : integers, floats
3. String : Stream of characters, enclosed in quotes ( single , double, backtick quotes)
4. Boolean : true or false
5. Null : empty or no value
6. Undefined : declared variable with no value

**Strings :** Stream of characters enclosed in quotes

it will provides a functionality

**PYTHON, JAVA, C, C++**

**DATA STRUCTURES**

**#leet code**

**#hackerrand**

**#codechef**

**#Written test in mncs**

**#technical interview**

**#Conditional statements – only once #1 – 10C**

**#Loops :**

**#functions :**

**#Oops**

**Conditional statements :**

To make the set of instructions execute only when the conditional id True Block of Code (conditional block) set of instructions

When this block of code will execute when the specific condition is true or false based on the condition.

Indentation : { }

The code used or written inside the indentation can be considered as the block code OR conditional block

Eg : {

------ block code

}

**08-01-2025**

**Java script operators :**

+,-\*,

Multiplication - \*

Division - / (Q)

Modulus operator - % (R)

Exponential (raised to the power) - \*\*

ASSIGNMENT OPERATORS :

Var name – “swetha”

+=

-=

/=

%=

**LOGICAL OPERATORS :**

AND - &&

T AND T=T

OR = ||

T OR F = F

NOT = !

**RELATIONAL OPERATORS (COMPARISION) VALUES :**

< , > , >=, <=, ==, ===, !=, !===

=== - checks the datatype and also the value

== - only checks the value doesn’t checks the datatype

**CONDITIONAL OPERATORS : TERNARY OPERATOR (?)**

First evaluates for true or false

Syntax :

Condition : ? e1 : e2

**WINDOWS :**

ALERT –

PROMPT –

**09-01-2025**

**CONDTIONAL STATEMENTS :**

To make 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 the specific conditions are true

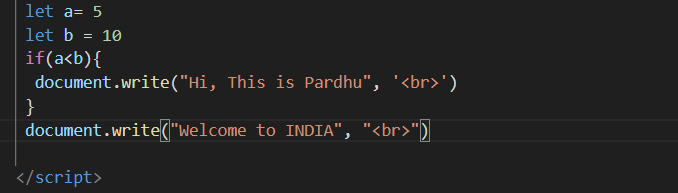
**Condition :** An expression that evaluates an result (TRUE OR FALSE)

**Ex :** cosole.log(5>6)

**METHODS :**

If : only one possible condtion

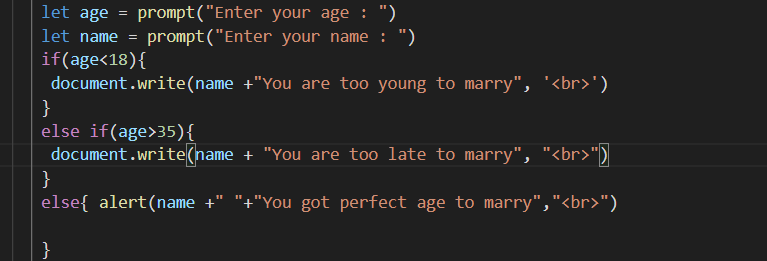
If the condition is true it will execute else it will not execute



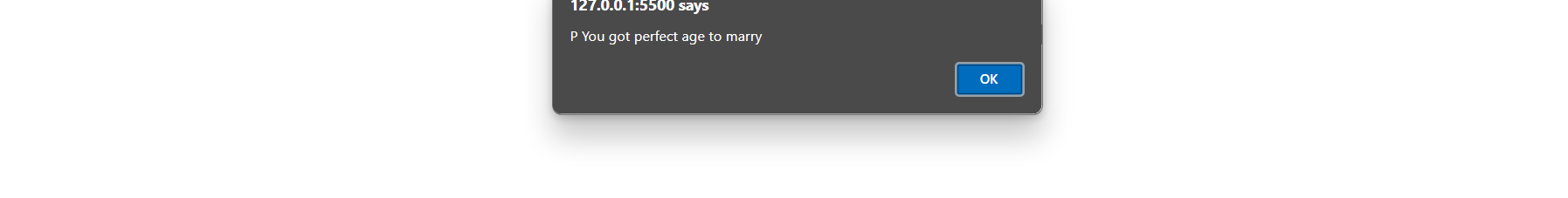
OUTPUT :



If-else :



OUTPUT :

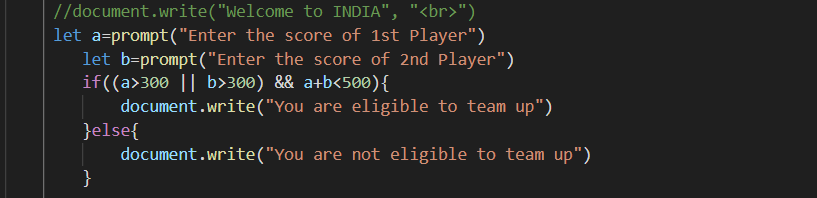


ASSIGNMENT :

Write a program that reads the scored A and B of two players and checks if one of the scores is greater than 300 and sum of the scores is less than 500

INPUT : the first linne of input contains an integer representing the score A. and second line of input contains an integer representing the score B

OUTPUT : The output should be a single line containing a string. Can team up should be printed if one of the score is greater than 300 and the sum of the scores is less than 500, otherwise cannot team up should be printed



**10-01-2025**

**PROBLEM SOLVING WITH JAVASCRIPT**

1. Write a program to check whether the number is divisible by 2 and 3 and both (take prompt)
2. Write a program to find largest number out of three numbers expected from the user
3. Accept the following from the user and calculate the percentage of classes attended:
4. Total number of working days
5. Total number of days for absent

**After calculating the percentage, show that, if the percentage is less than 75 then the student will not eligible to write exam**

**21-01-2025**

let i=1, n=5;

      while(i<=n){

        console.log(i)

        document.write(i)

        i=i+1

****

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

            if(i==4){

            break;

        }

        console.log(i)

        document.write(i)

      }

****

**22-01-2025**

**Nested loops:**

A loop inside the another loop

The inner loop will execute one time for each iteration of outer loop

An inner loop with the repeating block of outer loop

**23-21-2025**

**FUNCTIONS:**

A Function is reusable block of code

It can we call any where in the program

**REUSABLE CODE :** Using an existing code with out writing it ever we need

**You can use the same code with different arguments to get the results**

**JS:**

**KEY WORD : FUNCTION**

**PARAMETERS : values(variables) which are declared inside the () while defining a function**

**ARGUMENTS : values which are passed through the parameters**

**ANONYMOUS FUNCTION :**

**A function with out name , after we create a function with out name and we assign it to a variable**

**Example :**

**Sum = function(n1, n2){**

**Let r = n1+n2**

**Return r**

**}**

**Console.log(sum(8,9))**

**24-01-2025**

**Arrow functions :**

* **Es6 version**
* **More readable and more structure**
* **Anonymous functions (lambda functions)**
* **Without function name but they are assigned to a variable**

**EXAMPLE :**

**Let C=()=>{**

**Console.log(“Hi Pardhu”)**

**}**

**C()**

**ARROW FUNCTIONS WITH PARAMETERS :**

**Ex:**

            let x=(x,y) =>{

                console.log(x+y)

            }

            x(20,30)

y(40,50)

**ARROW FUNCTION WITH ONE PARAMETER**

**Ex:**

        let greet = x=> console.log(x)

        greet('hello')

**ARROW FUNCTION WITH OUT ARGUMENT :**

**EX:**

    let greet=()=>console.log(x)

    greet('hello')

**EX:** arrow functions as an expression

**SYNCHRONOUS AND ASYNCHRONOUS**

* Step by step
* one task
* **example :**

function task1(){

console.log(“Task 1 : start”)

}

function task2(){

console.log(“Task 2 : start”)

}

function task3(){

console.log(“Task 3 : start”)

}

Task1()

**ASYNCHRONOUS FUNCTION:**

console.log("Hi Pardhu")

    setTimeout(()=>{

        console.log("Hi Im waiting")

    },5000);

    console.log("Im in LAB")

**JAVASCRIPT CALLBLACK FUNCTION :**

A call back function is a function that is passed to argument to another function.

Ex :

function greet(name,callback){

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

        callback()

    }

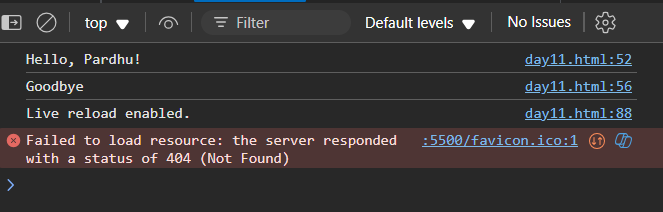
    function saygoodbye(){

        console.log("Goodbye")

    }

    greet("Pardhu",saygoodbye)

OUTPUT:



**PROMISE :**

**Eventual completion**

**Three states :**

1. **pending : still not completed – continuing**
2. **fulfilled == task completed**
3. **rejected == the operation is failed**

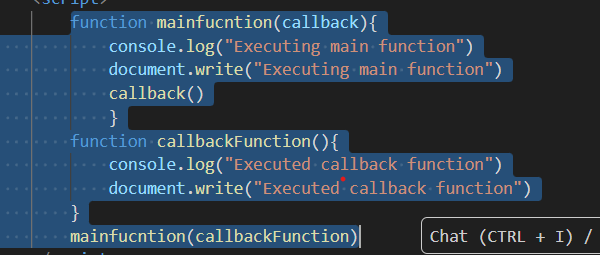
**25-01-2025**

**Callback functions :**

**You can pass the callback function as ana argument 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)**

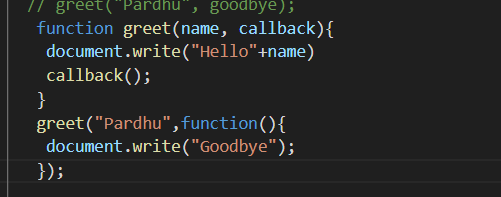
**EXAMPLE :**

****

**OUTPUT :**

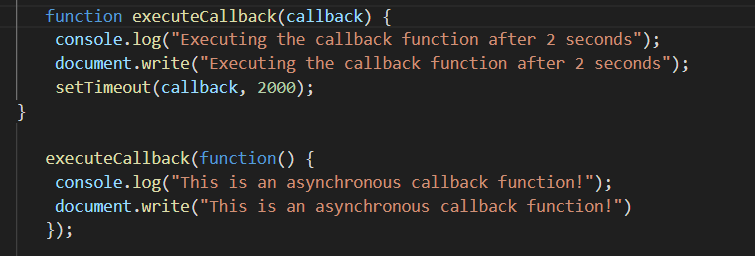
****

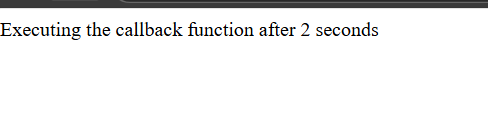
ANONYMOUS FUNCTION :



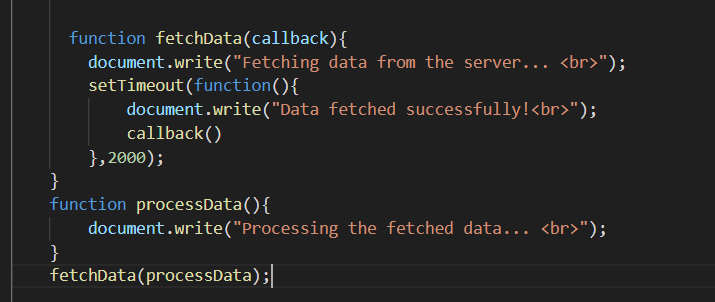
OUTPUT :



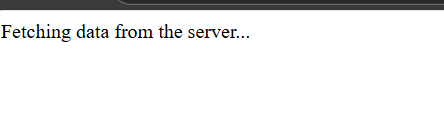
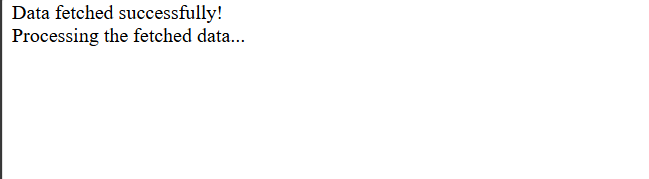








OUTPUT :

EX 5 :

Using a callback function with array methods (map)

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

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

Return num\*2

});

Console.log(d)



function registeruser(username, email,callback){

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

    document.write("registering user : "+username)

    setTimeout(function(){

        console.log("user"+username+"registered successfully","<br>");

        document.write("user"+username+"registered successfully","<br>");

        callback(email)

    },2000);

   }

   function sendwelcomeemail(email){

    console.log("sending welcome email to "+email,"<br>")

    document.write("sending welcome email to "+email,"<br>")

   };

   registeruser("Pardhu", "pardhu@gmail.com",sendwelcomeemail);

OUTPUT :





**27-01-2025**

Promise :

A promise a special javascript object that represents the eventual completion(or failure) asynchronous operation.

It allows to handle asynchronous tasks more efficient than call backs

**THREE STATES :**

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

**EXP :**

1. **To create a promise – newpromise()**

**TWO PARAMETERS –**

**Resolve –** to mark the promise as successful

**Reject :** mark it as failed

1. **HANDLING THE RESULT :**

.then() method is called when promise is fulfilled. It will receives the result from resolve()

**.catch() :** method is called if the promise is rejected, it receives the error message from reject()

Ex :

let mypromise = new Promise((resolve, reject) => {

            let success = false;

            if(success){

                resolve("The operation is successfull");

            }else{

                reject("The operation is failed");

            }

        })

        mypromise

        .then(result =>{

            console.log(result);

            document.write(result)

        })

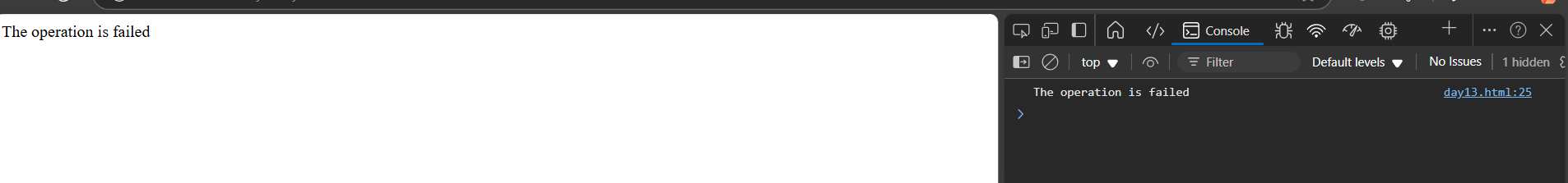
        .catch(error =>{

            console.log(error);

            document.write(error);

        })

OUTPUT :



**CHAINING PROMISES :**

 let promise = new Promise((resolve,reject) => {

            resolve(5);

        })

        promise

        .then(result => {

            console.log(result);

            document.write(result)

            return result\*2

        })

        .then(result => {

            console.log(result);

            document.write(result)

            return result+3

        })

        .then(result => {

            console.log(result);

            document.write(result)

            return result

        })

        .catch(error => {

            console.log("error", error);

        })

**OUTPUT :**

****

**EX : PROMISE WITH setTimeout**

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

let promise = new Promise((resolve,reject) => {

        resolve(5);

    })

    promise

    .then(result => {

        console.log(result);

        document.write(result)

        setTimeout((promise)=> {

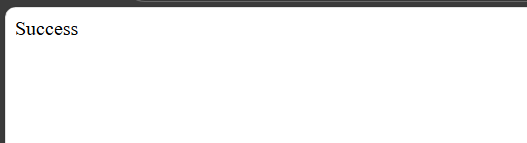
            console.log("Success")

            document.write("Success")

        },3000)})

OUTPUT :





**29-01-2024**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <h1 id = 'a'>I am sai</h1>

    <h1 id = 'b'>I am lohith</h1>

    <h1 id = 'c'>I am pardhu</h1>

    <script>

        let x = document.getElementById('a')

        console.log(x)

    </script>

</body>

</html>

**OUTPUT**

****

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Document</title>

</head>

<body>

    <h1 id = 'a' class="a">I am sai</h1>

    <h1 id = 'b' class="b">I am lohith</h1>

    <h1 id = 'c' class="c">I am pardhu</h1>

    <h1 class = "a">I am srinath</h1>

    <script>

        let x = document.getElementsByClassName('a')

        console.log(x)

    </script>

</body>

</html>

**OUTPUT :**



QueryselectorAll:sssssssssssssssss

It selects all the html elements by its class name and tagname

MANIPULATING ELEMENTS :

* To create HTML elements :

createElement() – create a new element

method – document.createElement()

* textContent :

it is used to insert the text

**30-01-2025**

EX :

let title

        for(let i=0; i<3;i++){

            title=document.createElement('h2')

            title.className = "tt"

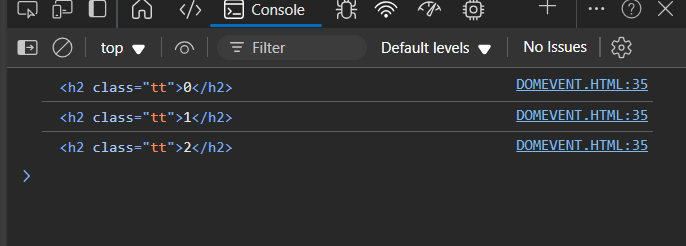
            title.textContent = i

            console.log(title)

           document.write(title)

        }

OUTPUT:



Ex :

 const titles = ["Pardhu", "Lohith", "Saicharan"]

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

            let title = document.createElement("h1");

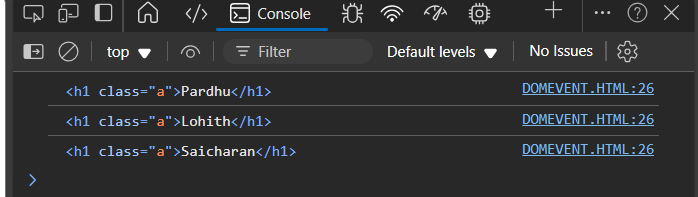
            title.className = "a";

            title.textContent = titles[i]

            console.log(title)

        }

Output :



**REMOVING A CHILD ELEMENT FROM PARENT NODE**

 <h1> Removing child node</h1>

    <h1> I AM PARDHU</h1>

    <ul>

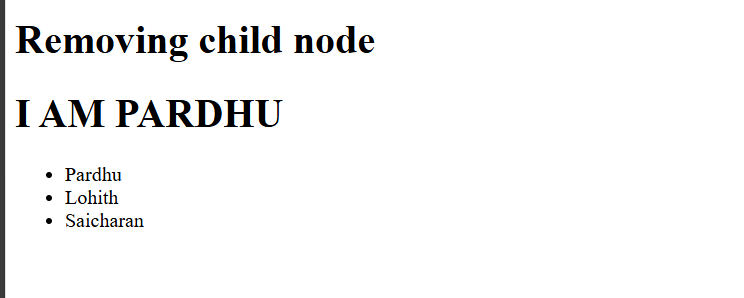
        <li> Pardhu</li>

        <li> Lohith</li>

        <li>Saicharan</li>

    </ul>

OUTPUT :



    <h1> Removing child node</h1>

    <h1> I AM PARDHU</h1>

    <ul>

        <li> Pardhu</li>

        <li> Lohith</li>

        <li>Saicharan</li>

    </ul>

    <script>

        const ul = document.querySelector('ul')

        const lists = document.querySelectorAll('li')

        console.log(lists)

        for(const list of lists){

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

            ul.removeChild(list)

            console.log(lists)

        }

    </script>

OUTPUT : 

<h1> Removing child node</h1>

    <h1> I AM PARDHU</h1>

    <ul>

        <li> Pardhu</li>

        <li> Lohith</li>

        <li>Saicharan</li>

    </ul>

    <script>

        const ul = document.querySelector('ul')

        const lists = document.querySelectorAll('li')

        for(const list of lists){

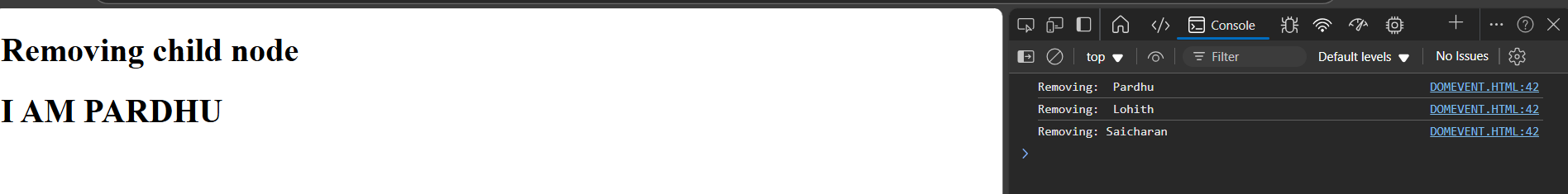
            console.log(`Removing: ${list.textContent}`)

            ul.removeChild(list)

        }

    </script>

Output :



**TO APPEND A NEW ELEMENT :**

Append() – to append a new element in parent node

PROGRAM :

<div id="a">

        <h1>Pardhu</h1>

        <h1>Lohith</h1>

    </div>

    <div id="b">

        <h1>Saicharan</h1>

        <h1>Srinath</h1>

    </div>

    <script>

        let n = document.createElement("h2")

        n.textContent = "Lohith is a mental patient";

// get the parent element

        let p = document.getElementById("b")

// append the new element to the parent node

        p.appendChild(n)

// log the inner text of the parent

        console.log(p.innerText)

    </script>

OUTPUT :



 <div id="a">

        <h1>Pardhu</h1>

        <h1>Lohith</h1>

    </div>

    <div id="b">

        <h1>Saicharan</h1>

        <h1>Srinath</h1>

    </div>

    <script>

        let n = document.createElement("h2")

        n.textContent = "Lohith is a mental patient";

// get the parent element

        let p = document.getElementById("b")

        console.log(p.innerText)

// append the new element to the parent node

        p.appendChild(n)

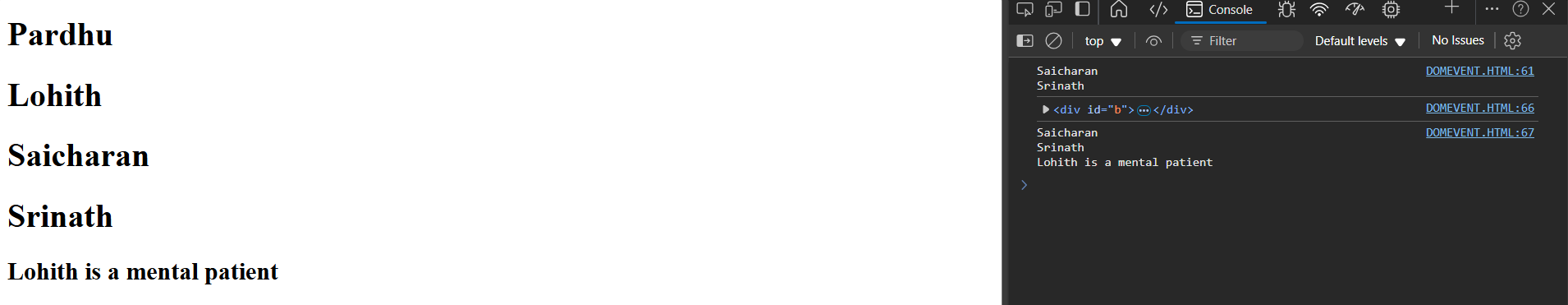
// log the inner text of the parent

        console.log(p)

        console.log(p.innerText)

    </script>

OUTPUT :



**31-01-2025**

insertBefore() : insert a new node before an existing node or child node of a specific parent code

this requires a reference to both the new node and the existing node where you want to insert before.

<div>

    <div id="a">

        <h1>Pardhu</h1>

        <h1>Lohith</h1>

    </div>

    <div id="b">

        <h1>Saicharan</h1>

        <h1>Srinath</h1>

</div>

    <script>

        let n = document.createElement("h2")

        n.textContent = 'Askay'

        let p = document.getElementById('a')

        let l = document.getElementById('r')

        p.insertBefore(n,l)

        console.log(p)

    </script>

OUTPUT :



To remove an element :

Let b = document.getElementById(“b”)

**01-02-2025**

1. variables : let, var, const
2. operators
3. conditional statements
4. loops
5. functions
6. promises
7. DOM