Script.js

```
                    //* #. 💫💫Value and Variable 💫💫

// 1.Naming Variable:Rules and Best Practices

// var my_firstName='Adarsh';   ✅

//var 123Name='Adarsh'; ❌

//var _myLastName='Rai' ✅

//var $cityName='Bhiwandi' ✅

//var my@Email='adarshrai@gmail.com' ❌






                    //* #. 💫💫 Data Type 💫💫

//?⚔️ Primitive Data Type

//1.String    var name='Adarsh'
//2.Number    var num=45
//3.Boolean   var isRaining =true
//4.BigInt     const bigInt=12345678912345+789456248925599623 55n
//5.Undefined   var name;
//6.Null         var name=null
//7.Symbol       const mysymbol=symbol('description')


//?⚔️Object Data Type

//1.An Object
//2.An Array
//3. Date


                      // #. ❌❌ InterView Question ❌❌


//!1.What is the perpose of typeof operator
//ans👉 use to find the datatype of a variable

//Example👇

// var num=56;
// console.log(typeof num);

// var myname='adarsh'
// console.log(typeof myname);



//!2.what is the result of typeof null
// console.log(typeof null);  //result is object⟹ its a javascript bug



//!3.convert a string to number
//⟹ we just need to add "+" sign before the string
//Example👇

// var myString ='10'
// console.log(typeof +myString);
// console.log(typeof Number(myString));   using Number Function



//!4.convert a number to string
//⟹ we just need to add a empty string after the number
//Example👇
// var str=5;
// console.log(typeof (str + ''));
// console.log(typeof String(str));  //using string function




//!5.what are the truthy and falsy value

//💫Truthy Value

//👉true
//👉any non-empty string('hello')
//👉any non-zero number (25)
//  👉array and object



//💫Falsy Value
//👉False
//👉any empty string('')
//👉any zero number (0)
//  👉Undefined
//👉Null
//👉NaN(Not a Number)




                        //⚔️⚔️Bonus⚔️⚔️
//❤️❤️ParseInt and ParseFloat

```

```
//👆👆👆 Both are the javaScript function used to convert string to number


//!1.❤️ParseInt   ==> used to convert string to into Integer Value eliminate decimal value

//Example👇
// var myString ='55'
// var myNumber=parseInt(myString)
// console.log(typeof myNumber);

// var myString=55.5
// console.log(parseInt(myString));   return only 55 not 55.5



//!2.❤️ParseFloat ==> used to cnvert string to any number including float value
//Example👇
// var myString ='55.5'
// var myNumber=parseFloat(myString)
// console.log(myNumber);
// console.log(typeof myNumber);



//!  🤔💭What will be the output of 💭🤔🤔
// console.log(parseInt('077'));  //==> 77 not print 0
// console.log(parseInt('-123'));  //==> -123
// console.log(parseInt('@123'));  //==> NaN
// console.log(parseInt('xyz'));   //==> NaN(Nan Stand or Not a number return a when a methamatical operation doesn't yield a valid number)


//To check a value a number or not use isNaN() functiom
//Example👇
// console.log(isNaN('xyz'));  //==> true
// console.log(isNaN('56'));  //==> false  because it is a number ignore quots








                    //*🖊️🖊️Expression & Opeartors🖊️🖊️

//?🤔🤔Type of Operator

//1.👉Assignment Operator  (=)
//2.👉Arthmetic Operator   (+,-,*,/,%)
//3.👉Comparison Operator   (==,===,<,>,<=,>=)
//4.👉Logical Operator       (AND(&&), OR(||),NOT(!))
//5.👉String Opeartor         (+)
//6.👉Unary Operator
```

```
//7.👉Ternary Operator        (var result=a⩾18?'can vote':'can't vote')
//8.👉Typeof Operator         (typeof) ⇒check type of data type







                    //* 👉👉Control Statement & Loops👉👉

//1.👉IF-Else Statement
//2.👉Switch Statement
//3.👉While Loop
//4.👉Do-While Loop
//5.👉For-In /For-of Loop





//?1.👉IF-Else Statement

//Syntax 👇

//if(condition){ code .... }
// else{  code ...    }



//Example👇
// var temp=40
// if(temp>30){
//      console.log('lets go beach..🏖🏖');

// }
// else{
//      console.log('Watch Tv at Home 📺👀');

// }


//we can also use an else if clause to check additional condition

// var temp=25;
// if(temp>30){
//      console.log('lets go beach..🏖🏖');
// }
// else if(temp>20 && temp<30){
//      console.log('Watch Tv at Home 📺👀');

// }
// else{
//      console.log('thandi hai so jao');
// }




//!InterView Question 1 ❓ ❓ ❓

//if the person is yonger than 18 , not a citizen , or not
// registered to vote display a message saying they are not eligible to vote

//if the person is 18 or older but not a citizen,display a message
// saying they are not eligible due to citizenship status

//if the person is 18 or older a citizen  but not registerd to vote display
//a message they are not eligible due to registration status

//Ans 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇 👇

// var age=17;
// var isCitizenOfIndia=false;
// var isRegistredForVote=false;

// if(age>=18 && isCitizenOfIndia && isRegistredForVote){
//     console.log('Congratulation you can vote');

// }
// else if(age>=18 && !isCitizenOfIndia && !isRegistredForVote){
//     console.log('you are not eligible');

// }
// else if(age>=18 && !isCitizenOfIndia && isRegistredForVote){
//     console.log('you are not eligible due to citizenship status');

// }
// else if(age>=18 && isCitizenOfIndia && !isRegistredForVote){
//     console.log('you are not eligible due to registration status');

// }
// else{
// console.log('you are not eligible due to age,citizenship status and registration status');

// }




//Method 2 👇 👇 👇 :

// var age=12;
// var isCitizenOfIndia=false;
// var isRegistredForVote=true;

```

```javascript
// if(age>=18){
//     if(isCitizenOfIndia){
//         if(isRegistredForVote){
//             console.log('Congratulation you can vote');
//         }
//         else{
//             console.log('you are not eligible due to registration status');
//         }
//     }
//     else{
//         console.log('you are not eligible due to citizenship status');
//     }
// }
// else{
//     console.log('You are not eligible to vote');

// }



//!InterView Question 2 ❓❓❓

//❓❓write a program to check if a number is even or odd

// var num=13;
// if(num%2==0){
//     console.log('num is even');

// }
// else{
//     console.log('num is odd');

// }



//!❓❓write a program to check if a number is Prime
// var num=13;
// var isPrime=true

// for(var i=2;i<num;i++){
//     if(num%i==0){
//         isPrime=false;
//         break;
//     }
// }
// if(isPrime){
//     console.log('num is prime');

// }
// else{
//     console.log('num is not prime');

// }
```

```

//! ❓❓ write a program to check if a number is positive , negative or zero

// var num =-1
// if(num===0){
//     console.log('num is zero');
// }
// else if(num>0){
//     console.log('num is positive');
// }
// else{
//     console.log('num is negative');
// }



//?2.Switch Statement💭💭
//Syntax👇👇

// switch(expression){
//     case value1:
//         code
//         break;

//     case value1:
//         code
//         break;

//     default:
// }


//Example👇👇👇👇👇

// var day='Monday';

// switch(day){
//     case 'Monday':
//         console.log('Today is Monday');
//         break


//     case 'Friday':
//         console.log('lets party🥳🎉🎊');
//         break

//     case 'Sunday':
//         console.log('lets go for movie');
//         break

//     default:
//         console.log('no condition match');

// }
```

```




//?3.👀 👀 While Loop👀 👀

//syntax 👇 👇 👇 👇
// while (condition) {

// }

//Example==> Print 1 to 10 number using while loop ⁉️ ⁉️ ⁉️

// var i=1;
// while (i<=10) {
//     console.log(i);
//     i++;

// }

//!Create a Table of 5 using while loop

// var num=5;
// var i=1
// while (i<=10) {
//     console.log(num+"X"+i+'='+num*i);
//     i++;
// }







//?4.👀 👀 do-While Loop👀 👀

//syntax 👇 👇 👇 👇
//  do {

    //code

// }while (condition)

//Example==> Print 1 to 10 number using do-while loop ⁉️ ⁉️ ⁉️

// var i=1;
//  do{
//     console.log(i);
//     i++;

// }while (i<=10)

//!Create a table of 19 using do-while loop
```

```

// var num=19
// var i=1
// do{
//      console.log(num+"X"+i+'='+num*i);
//      i++
// }while(i<=10)


//5.👀👀 For Loop👀👀

//syntax 👇👇👇👇
// for(initialization;condtion;iteration){
//      code
// }

//Example==> Print 1 to 10 number using For Loop ⁉️⁉️⁉️

// for(var i=1;i<=10;i++){
//      console.log(i);

// }


//!Create a table of 29 using for loop
// var num=29
// for(let i=0;i<=10;i++){
//      console.log(num+"X"+i+'='+num*i);
// }


//!Create a infinit for loop
// for( ;; ){
//      console.log('Hello Adarsh');

// }



//!Practic : 👇👇👇👇
//!Calculate the sum of number from 1 to 10 using for loop

// var  sumOfNumber=0;
// for(i=1;i<=10;i++){
//    sumOfNumber=sumOfNumber+i;

// }
// console.log(sumOfNumber);



//!Program to check if a year is a leap year

```

```
//*condition for leap year ==> a year is divisible by 4 and 400 and not divisible
//* by 100

// var year=2002

// if(year % 4===0){
//   if(year % 100 === 0 ){

//     if(year % 400===0){
//       console.log(year,'is a leap year');

//     }
//     else{
//       console.log(year,'is not a leap year');

//     }
//   }
//   else{
//     console.log(year,'is a leap year');

//   }
// }
// else{
//   console.log(year,'is not a leap year');

// }



//!Draw a pattern

// *
// * *
// * * *
// * * * *
// * * * * *


// for(var i=1;i<=5;i++){
//   var pattern="";
//   for(var j=1;j<=i;j++){
//     pattern=pattern+" *"
//   }
//   console.log(pattern);

// }
```

```
                    //* #. 💫💫Function In JavaScript 💫💫



//?1.Fucntion Decleration-:👇👇👇
//?Declare the function using 'FUNCTION' keyword followed by the function name
//? parameter(if any) and the function bodey

//Example-:👇👇

// function greet(){
//     console.log('hello i am a function');

// }


//?2.Fucntion Invocation (calling a function)-:👇

//? A function you can invoke or call it by using it's name followed by
//? parentheses if a function has parameter provide value(argument) for
//? those parameter inside the parantheses


//Example-:👇👇

// greet()

//?Practice Time 👀👀👀

//!write a function to find the sum of two number

// function sumOfTwoNumber(){
//     var a=5
//     var b=10
//     console.log(a+b);

// }
// sumOfTwoNumber()



//?3.Function Parameter 👇👇
//? It act as a placeholder for a value that will be provided when
//? the function is called

//Example-:👇👇

// function greet(parameter1,parameter2,parameter13,...){
      // code
// }
// greet()


//?4.Function Argument 👇👇
```

```
//? A function argument is a value that you provide when you call
//? a function. argument are passes into a function to fill parameter
//? defined in the function Decleration


//Example-:👇👇
// greet(Argument1,Argument2,Argument3, ... )


//?Practice Time 👀👀👀

//!write a function to find the sum of two number with parameter

// function sumOfTwoNumber(a,b){
//     console.log(a+b);

// }
// sumOfTwoNumber(78,89)


// var result=(function (a,b){
//     console.log(a+b)

// })(5,10);







                        //* #. 💫💫ECMASCRIPT In JavaScript 💫💫

// ?.ECMASCRIPT 2015/ES6

//todo 1.LET AND CONST
//todo 2.Template Strings
//todo 3.Default Arguments
//todo 4.Arrow Function
//todo 5.Destructuring
//todo 6.Object Properties
//todo 7.Rest Operators
//todo 8.Sperad Operators







//? 1.LET AND CONST

//*LET⟹It is used to declared varible with block-scoped.LET are Mutable
```

```
//*CONST⟹It is used to declared varible with block-scoped.CONST are immutable

//Example-:👇👇

// let myName='Adarsh';
// myName='vivek'
// console.log(myName);



// const Name='Adarsh';
// Name='vivek'
// console.log(Name);    //Assignment to constant variable. hence const are immutable






//*how LET and Const are block-Scope Variable

//Example 2-:👇👇

// if(true){
//     let myName='Adarsh'
//     const mySurname='Rai'
// }

// console.log(myName);
// console.log(mySurname);     //we cant access the LET and CONST variable outside the block scope





//? 2.Template Strings
//In ES6,template string also known as template litrals provide
// a convenient and flexible way to craete string in javascript.

//template string are enclsed in backticks (` `) rather than single or
// double quots

//syntax:-👇👇

//${``}


//Example:-👇👇

// let my_firstName='Adarsh';
// let my_lastName='Rai'

// let My_FullName=`${my_firstName} ${my_lastName}`
// console.log(My_FullName);



//Example 2:-👇👇

// const age=22;
// const message=`I am ${age} Years Old`
// console.log(message);




//? 3.Default Arguments
//default function parameter allow named parameter to be
// initialized with default value if no value or undefined or undefined is passed



//Example :-👇👇

//!Write a function to find sum of two number ?
//!what if during function call user only passed one arugumet



// function sum(a,b=20){   //here b set to the default parameter
//     return a+b
// }

// console.log(sum(10));






//? 4.Arrow Function

//Arrow function also known as fat arrow function were introduced
//as a concise way to write anonymous function.

//* Normal Function Expression

// const sum=function(a,b){
//     return a+b
// }
// console.log(sum(5,20));



//* convert  Normal Function Expression to fat arrow function

```

```
// const sum=(a,b)=>{
//     return a+b
// }
// console.log(sum(89,89));



// const sum=(a,b)=>`the sum of two number is :==>${a+b}`




//!Interview  Questions 💭💭💭💭

//* 1. Write a javascript function calculator that take two number and operator
//*as parameter and return the result of two operator.the function support
//* addition,substraction,multiplication and division

// const calculator=(a,b,operator)=>{
//     switch(operator){
//         case '+':
//             console.log(a+b);
//             break

//         case '-':
//             console.log(a-b);
//             break

//         case '*':
//             console.log(a*b);
//             break

//         case '/ ':
//             console.log(a/b);
//             break

//         default:
//             console.log('please enter valid operator');


//     }
// }

// calculator(400,4,'+')
// calculator(400,4,'-')
// calculator(400,4,'*')
// calculator(400,4,'/')





//*2.Write a Function to reserve a given string without using built-in
//* reverse methods
```

```

// const reverseString=(string)=>{
//     debugger
//     let result="";
//     for(let i=string.length-1;i>=0;i--){
//          result=result+string[i]

//     }

//    return result

// }
// console.log(reverseString('Adarsh Rai'));



//*2.Write a Function to determine if a given string is a palinfrome
//* (read the same backward as forward)


// const findPalindrome=(string)=>{
//     debugger
//     let result="";
//     for(let i=string.length-1;i>=0;i--){
//          result=result+string[i]

//     }

//    if(result===string){
//         return 'is a palindrome'
//    }
//    else{
//     return 'not a palindrome'
//    }

// }
// console.log(findPalindrome('oyo'));





                        //* #. 💫💫 JavaScript Arrays 💫💫


//*Array 👇👇👇
//? ==>JavaScript  Array is an object that reprsent a collection of
//? similar type of elements

//? Each value will be called as an elements

//? In array each elements is represented by an inedx which start with zero

```

```
//Syntax 👇 👇
//                  0       1        2
// const person=['Ram','Adarsh','Hari']

//? We can access each element by using indexes

//person[0] ⟹Ram
//person[1] ⟹Adarsh



// const person=['Ram','Adarsh','Hari']
//                 👇               👇
//          Lower index        Upper Index

//* First Elements or head ⟹Refer to the element at index 0
//*Last Element or tail ⟹ refer to the element at the last which can be
obtained
                              //* using array.length-1

//?Es6 2022 also introdice new .at() method in array which help to
//? index from last elements too easily



// person[-1] ⟹error
//person.at(-1) ⟹Hari
//person.at(-2) ⟹ Adarsh





//* What we will cover 🥳 🥳 🥳

//todo 1. Creating Array / Accessing Element/Modifying Element
//todo 2. Array Traversal / Iterating over Array
//todo 3. How to insert , Add ,Replace and Delete Element in an Array(CURD)
//todo 4. Filter in an Array
//todo 5. Searching In Array
//todo 6. How to sort and complex an Array
//todo 7. Very Vary imp array Method


//? 1. Creating Array / Accessing Element/Modifying Element
//Array In javaScript can be created using the array constructor or with
//array literals (square bracket [])

//* Using Array Constructor

// let fruits=new Array('apple','orange','banana')
// console.log(fruits);


```

```
//*Uisng Array Literals

// let fruits=['apple','orange','banana']
// console.log(fruits);


//* How to create empty array

// let arr=[]
// console.log(arr);
// console.log(typeof arr);   //type of array is object




//*Accessing Elements

//Array elements are accessed uisng zero-based indices

// let fruits=['apple','orange','banana']
//            0        1        2

// console.log(fruits[0]);





//*Modifying Elements

//you can modify array elements by assigning new values to specific indices

// let fruits =['apple','orange','banana']
// fruits[1]='mango'
// console.log(fruits);




//? 2. Array Traversal / Iterating over Array

//*for of loop also known as iterable

//the for of loop is used to iterate over the value of an iterable
// object such as array,string or other iterable object

// let fruits=['apple','orange','banana','mango','grapes']

// for(let items of fruits){
//     console.log(items);

// }

```

```

//* Using for loop

// let fruits=['apple','orange','banana','mango','grapes']
// for(let i=0;i<fruits.length;i++){
//     console.log(fruits[i]);
// }



//*for in loop

//the for in loop is used to iterate over the value properties
//(including indices) of an object


// let fruits=['apple','orange','banana','mango','grapes']
// for (let items in fruits){
//     console.log(items);
// }




//*forEach Method 👀 👀

//the arr.forEach() method call provided function once for
// each element of the array. the provided function may perform
 //any kind of operation on the element of the given array


//?Syntax - forEach 👇 👇

// array.forEach((currElement,index,array)=>{
//      your logic
// },thisValue)



// let fruits=['apple','orange','banana','mango','grapes']
// fruits.forEach((currelem,index)=>{
//     console.log(`${currelem} ${index}`);

// })



// let fruits=['apple','orange','banana','mango','grapes']
// const myforEach=fruits.forEach((currelem,index)=>{
//     return `${currelem} ${index}`

// })
// console.log(myforEach);   //undefined  ==> we can't return in forEach method

```

```
//* MAP Method 👀 👀

//map create a new array from calling a function for every array
//element. map() does not change the original array


//? Syntax-: Map 👇 👇 👇

// array.map((currElement,index,array)=>{
//       logic
// })


//?Example -: 👀 👀

// let fruits=['apple','orange','banana','mango','grapes']

// fruits.map((curElem,index)=>{
//     console.log(`${curElem} ${index}`);

// })



// let fruits=['apple','orange','banana','mango','grapes']

// const myMaparr=fruits.map((curElem,index)=>{
//     return `${curElem} ${index}`
// })

// console.log(myMaparr);  //it create a new array




//!Practice Time-:🥳 🥳

//*Write a program to multiply each element with 2

// const array=[1,2,3,4,5]
// const multiplyElem=array.map((elem)=>{
//     return elem*2
// })

// console.log(multiplyElem);


//!Notes 💭 💭

//!use forEach to perform action on each element
//! use map to create a mew array with transformed elements
```

```
//? 3. How to insert , Add ,Replace and Delete Element in an Array(CURD)


//*1. Push()-: Method that adds one or more element to the end of an Array

//Syntax-:👇👇👇
// push(element_name)

//Example:-👇👇👇

// let fruits=['apple','orange','banana','mango','grapes']
// fruits.push('pineapple')
// console.log(fruits.push('pineapple'));
// console.log(fruits);



//*2. Pop()-:Method that remove the last element from an array

//Syntax-:👇👇👇
// pop(element_name) or pop()⟹if we don't put anything it remain remove alement

//Example:-👇👇👇

// let fruits=['apple','orange','banana','mango','grapes']
// fruits.pop()
//  console.log(fruits.pop()); //mango⟹return which element is pop
// console.log(fruits);


//*3.unshift()-: Method that add one or more element to the beginning of an array

//Syntax-:👇👇👇
// unshift(element_name)

//Example:-👇👇👇

// let fruits=['apple','orange','banana','mango','grapes']
// fruits.unshift('pineapple')
// console.log(fruits.unshift('guava'));
// console.log(fruits);


//*4.shift()-: method that remove the first element from an array

//Syntax-:👇👇👇
// shift(element_name) or shift()

//Example:-👇👇👇

```

```
// let fruits=['apple','orange','banana','mango','grapes']
// fruits.shift()
// console.log(fruits.shift());
// console.log(fruits);




//*What if want add or remove anywhere in an element

//?Splice()🫤🫤

//the splice() method of array instances change the content of an array
// by removing or replacing existing element and/or adding new element in place

//Syntex:-👇👇

// splice(StaticRange,deletCout,item1,item2,item3 ... itemN)

//Example-:👇👇

//let person=['Adarsh','Sita','Gita','Vivek','Ram']
//console.log(person.splice());  // it return the empty array
//console.log(person.splice(1));   //it start from index 1 and  return ⟹[
'Sita', 'Gita', 'Vivek', 'Ram' ]
//console.log(person.splice(0));  // it start from index 0  and return [
'Adarsh', 'Sita', 'Gita', 'Vivek', 'Ram' ]

//console.log(person.splice(1,1));    //it return deleted element⟹ [ 'Sita' ]


//?How to delete🫤🫤
//person.splice(1,    1)
//            👇     👇
//          start  DeleteCount
//console.log(person);  // start from 1 and delete 1 element ⇒ [ 'Sita ]  and
return [ 'Adarsh', 'Gita', 'Vivek', 'Ram' ]



//?How to Add🫤🫤

//person.splice(2,1,'Hari','Narayan','Golu')  //start from 2 deleted [ 'Gita' ]
and add 'Hari','Narayan','Golu'
//console.log(person);    //result is ['Adarsh','Sita','Hari',
Narayan','Golu','Vivek','Ram']


//!What if you want to add the  element at the end

// let person=['Adarsh','Sita','Gita','Vivek','Ram']
// person.splice(person.length,0,'Vicky')
// console.log(person);

```

```
//? 5. Searching In Array

//for Searching we have indexOf,lastIndexOf and includes

//const numbers=[1,2,3,4,5,6,7,8,9]

//*1.indexOf Method :→ the index of method return the first index at
//* which a given element can be found in the array or -1 if it is not found


//syntax-:👇👇

//1.indexOf(searchElement)
//2.indexOf(searchElement,fromIndex)

//Example-:👇👇

// console.log(numbers.indexOf(4));   //index of 4 is 3
// console.log(numbers.indexOf(4,     5));   //return -1  because 4 is not present after 5th index
//                              👇      👇
 //                 SearchElament  from_Index


 //*lastIndexOf():→ the lastIndexOf() method array return the last index at which a
 //* given element can be found in the array , or -1 if it is not present.
 //* the array is searched backward, starting at fromIndex

//  const numbers=[1,2,3,4,5,6,6,7,8,9,1]
//                          ←———————————  it sarch from backward
//  console.log(numbers.lastIndexOf(1));   // it reatun 10 because it return last index
//  console.log(numbers.lastIndexOf(1,5));    //it sarch from 5 to 0 index and 1 is find at 0 index as it search backward
 //



 //*Includes():→ The includes() method check whether an array includes a certain elements
 //*returning true or false

 //syantx:-👇👇

 //includes(searchElement)
 //includes(searchElem,fromIndex)


 //Example-:👇👇

```

```
//  const numbers =[1,2,3,4,5,6,7,8,9]
//  console.log(numbers.includes(5));  //return true because 5 is present at index 4

//  console.log(numbers.includes(4,5));  //return false because 4 is not present after 5th index






//todo Challange time 💭💭🤔🤔

//!1. Add dec at the end of an array

// const months=['jan','feb','march','april','may','june','july','aug','sep','oct','nov']
// months.splice(months.length,0,'dec');
// console.log(months);  //addes dec 😎


//!2.what is the return value of splice method

//an empty array ([])

//!3.update march to March

// const months=['jan','feb','march','april','may','june','july','aug','sep','oct','nov']

// months.splice(2,1,'March')
// console.log(months);   //updated 😎😎


//!4. delete june from an array
// const months=['jan','feb','march','april','may','june','july','aug','sep','oct','nov']
// months.splice(5,1)
// console.log(months);  //june is deleted 😎😎





//? 4. Filter in an Array

//*1.Find() Method-:

// the find method is udes to find the first element in an array
// that satisfies a provided testing function. it return the first matching
// element or undefined if no element is found


```

```


//Syntax-:👇👇

// Array.find((currElem,index,array)=>{
//     logic
// })


//Example-:👇👇

// const numbers=[1,2,3,4,5,4,5,6,7,8,4,1,2,3]

// const result=numbers.find((curElem)=>{
//     return curElem>5                      //it start searching and if element is found
                                            // then stop searching and give result
// })

// console.log(result);




//*findIndex() Method:-

//it return the index of the first element that satisfies the provided testing function
//if no element satisfy the testing function -1 is returned


//Syntax-:👇👇

// Array.findIndex((currElem,index,array)=>{
//     logic
// })


//Example-:👇👇

//  const numbers=[1,2,3,4,5,4,5,6,7,8,4,1,2,3]

//  const result=numbers.findIndex((curElem)=>{
//     return curElem>5
//  })

//  console.log(result);    //7 it return the index of the result




//*Filter Method():-> The filter method craete a new array with all elements that pass
//*test implemented by the provided function
```

```


//Syntax:-👇👇

//1.filter(callbackFn)
//2.filter(callbackFn,thisArg)


//Example-:👇👇

// const numbers=[1,2,3,4,5,6,7,8,9]

// const result=numbers.filter((elem)=>{
//     return elem>5
// })

// console.log(result);



//?UseCase:-->In a E-commerce website when we went to reomve or delete any product
//? from addToCard page

//Example-:👇👇

//!lets say user want to delete value 6

// const numbers=[1,2,3,4,5,6,7,8,9,6]

// const deleteElement=numbers.filter((curElem)=>{
//     return curElem!==6
// })

// console.log(deleteElement);



//! practice Time

//!1.Filtering Products by Price

// const products=[
//     {name:'Leptop',price:1200},
//     {name:'Phone',price:800},
//     {name:'Tablet',price:500},
//     {name:'SmartWatch',price:300},
//     {name:'AirPod',price:150}
// ]

//!Q1.filter product with a price less than or equal to 500

// const FilteredItem=products.filter((elem)=>{
//     return elem.price<=500
// })

// console.log(FilteredItem);



//!Q2. Filter Unique Value

// const numbers=[1,2,3,4,5,6,5,7,8,9]
// const unique=[]
// const UniqueValue=numbers.filter((elem)⇒{

//   if(!unique.includes(elem)){
//         unique.push(elem)
//     }
// })

// console.log(unique);







//? 6. How to sort and comppare an Array

//*sorting an Array:→The sort method sort the element of an array
//* in place and return the sorted array.by default it sort element as string


// const fruits=['Banana','Apple','Orange','Mango']
// fruits.sort()
// console.log(fruits);

// const numbers=[1,2,13,8,4,5,1,2,4,6]
// numbers.sort()
// console.log(numbers);





//*Comparing the array

//syntax :-👇👇
// const sortedNumber=Number.sort((a,b)⇒{
//     return a-b
// })

// const numbers=[1,2,4,5,11,45,12,23]
// numbers.sort((a,b)⇒{
//     return a-b
// })
// console.log(numbers);
```

```



//? how to get descending order

// const numbers=[1,12,3,4,9,7,8,6]
// numbers.sort((a,b)⇒{
//     return b-a
// })

// console.log(numbers);






//? 7. Very Vary imp array Method

//*Map():🙄🙄

//⭐ Create a new array from a calling a function for every array element
//⭐ does not execute the function for empty element
//⭐ does not change the original array

//!1.Using the map method write a function that takes an array of numbers and
return
//! a array where each number is squared but only if its an even number

// const numbers=[1,2,3,4,5,6,7,8,9]

// const evenNumberSquare=numbers.map((elem)⇒{
//    if(elem % 2 ══0){
//     return elem*elem
//    }
// }).filter((elem)⇒{
//     return elem ≠ undefined
// })

// console.log(evenNumberSquare);




//!2. using the map method write a function that takes an array of string and
return a new
//! array where each string is captialized

// const words =['apple','banana','cherry','date']

// const captilizedWords=words.map((elem)⇒{
//     return elem.toUpperCase()
// })
// console.log(captilizedWords);
```

```



//!3. using the map metgod write a function that takes an array
//! of name and return a new array each name s prefixed with 'Mr'

// const Names=['Adarsh','Ram','Hari','Shyam']
// const AddMr=Names.map((elem)⇒{
//     return `Mr ${elem}`
// })
// console.log(AddMr);







//*Reduce Method():👇👇

//The reduce method in javascript is used to accumulate or reduce
// an array to a single value. it iterate over the element of array
// and applies a callback function to each element updating an accumulator value
//with the result. the reduce method takes a callback  as it's first argument and
// an optional initialVlaue for the accumulator as the second argument

//Synatx 👇👇

// Array.reduce((accumulator,currentValue,index,array)⇒{
//     logic
// },initialValue)



//*BreakDown

//1.Callback :- a function that is called once for each element in tha array
//2.accumulator :- the accumulated result of the privious iterations
//3.currentValue :- the currentValue element being processed in the array
//4. index :- the index of the curElem
//5. array :- the original array
//6. initialValue :- An initialValue for the accumulator if not provided the first element
                       // of the array is used as the initial accumulator




//!Q.1 Write a javaScript function that calculate the total price of item
//!in a shopping cart the function should take an array of item price an input
//! and return the total price


// const productsPrice=[100,200,5,7,89,45,12,13]

// const totalPrice=productsPrice.reduce(function(accu,currElem,index,array){
//     return accu+currElem
// })

// console.log(totalPrice);





//*Algorithm of reduce method

//const productsPrice=[100,  200,5,7,89,45,12,13]
//                     👇     👇
 //                   acc    currElem


 //!1. First Iteration 👇👇

//accu+currElem ⇒ 100+200=300

//const productsPrice=[300,   5,7,89,45,12,13]
//                     👇     👇
//                    acc    currElem



 //!1. Second Iteration 👇👇

//accu+currElem ⇒ 300+5=305

//const productsPrice=[305,   7,89,45,12,13]
//                     👇     👇
 //                   acc    currElem



 //!1. Third Iteration 👇👇

//accu+currElem ⇒ 305+7

//const productsPrice=[312,89,45,12,13]
//                     👇    👇
//                    acc    currElem






 //!1. Forth Iteration 👇👇

//accu+currElem ⇒ 312+89
```

```

//const productsPrice=[401,45,12,13]
//                     👇    👇
//                   acc   currElem




 //!1. fifth Iteration 👇👇

//accu+currElem ⇒ 401+45

//const productsPrice=[446,12,13]
//                     👇    👇
//                   acc   currElem




 //!1. sixth Iteration 👇👇

//accu+currElem ⇒ 446+12

//const productsPrice=[458,13]
//                     👇    👇
//                   acc   currElem




 //!1. seventh Iteration 👇👇

//accu+currElem ⇒ 458+13=471

//const productsPrice=471








                            //* #.  💫💫 JavaScript Strings  💫💫



//*What we will cover

//todo 1. String and it's properties
//todo 2. Escape Character
//todo 3. String Search Method
//todo 4. Extracting String Parts
//todo 5. Extracting String Characters
//todo 6. Replacing String Content
//todo 7. Other Useful Methods

```

```javascript





//? 1. String and it's properties

//*String in javaScript

//string in javascript are a fundamental data type that
// represent a sequence of characters


//!Notes:-

//!string created with single or double quotes work the same
//! there is no different between the single and double quotes



//*String Properties

//1.length:→property that return the length of the string(number of characters)

// const str='hello world'
// console.log(str.length);   //including spaces





//? 2. Escape Character

//escape characters:→In javaScript backslash (\) is used as an
//escape characters.it allow you to include special characters in a
//string


//    code                 Result              Description
//     \'                    '                   single quotes
//     \"                    "                    double quots
//     \\                    \                    backslash



//Example:👇👇

// let text="My name is "Adarsh Rai" "
// console.log(text);       //we can't print double quots  it give syntax error



//solution 1:→ put double quots inside single quots or vise varsha  👇👇

```

```
// const text ='My name is "Adarsh Rai "'
// console.log(text);


//solution 2:→use backslash 👇👇

// const text ="My name is \"Adarsh Rai\""
// console.log(text);


//*if we want a new line we use \n

// const text ="my name is \"Adarsh Rai\" \n i am a full stack developer"
// console.log(text);






//? 3. String Search Method


//*1.IndexOf():→ The indexOf() metgod return index of the
//* first occurrence of a string in a string or it return -1 if
//* the string is not found


//Syntex:-👇👇
//1.indexOf(SearchString)
//2.indexOf(SearchString,Position)


//Exmaple:-👇👇

// let text='Adarsh Rai'
// console.log(text.indexOf('rai'));   //it is case-sensitive



// let text='Adarsh Rai'
// console.log(text.indexOf('Rai'));   // return 7 ⟹ look only the 1st letter


// let text='Adarsh Rai'
// console.log(text.indexOf('a',7));  //return 8 ⟹ after 7 index 'a' index is 8 because
                                    //it search from 7th index





//*2.lastIndexOf():-the lastIndex() method return the index of the last
//* occurrance of a specified text in a String
```

```
//Syntax-:👇👇
//1.lastIndexOf(searchElem)
//2.lastIndexOf(searchElem,position)


//Example:-👇👇

// let text ='Hello JavaScript,welcome to our world best JavaScript Course'
// console.log(text.lastIndexOf('JavaScript')); // return 43 ⟹ it search backward and 2nd
                                             // 'JavaScript' find in last and then it return last 'JavaScript' index


// let text ='Hello JavaScript,welcome to our world best JavaScript Course'
// console.log(text.lastIndexOf('our',5)); //return -1 it start searching backward and it search from 5-0
                                         // and 'our' is not find from 5-0 so it return -1



//*3.Search():→The Search() method search a string for a string(or regular expression)
//* and return the position of the match

//*return the index number where the first match is found return -1
//* if no match found

//*it is case-sensitive


//Example:-👇👇

// let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
//console.log(text.search('JavaScript')); // return 6

//*In search we use Regular Expression to avoid case-sensitive

// console.log(text.search(/javascript/i));  // return 6 ⟶ i is a regular expression to avoid case-sensitive



//!Important Tips 👇👇

//!1.The search() method can't take second start position argument

//search(searchElem,fromIndex) ❌❌

//!the indexof() method cannot take powerful search value (regular Expression)
```

```
//!they accept same argument(parameter) and return the same value




//*4.match():⟶return an array of the match value or null
//* if no match is found also a case-sensitive


//  let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
// console.log(text.match('javascript'));  //return null if string is not found


// console.log(text.match('JavaScript'));

//it return 👇👇 full details in array formate

//! [
//!     'JavaScript',
//!     index: 6,
//!     input: 'Hello JavaScript,welcome to our world best JavaScript Course',
//!     groups: undefined
//!   ]




//!Tips:here the JS convert the normal text into regular expression
text.match(/javascript/)
//!without the g falg

// console.log(text.match(/JavaScript/g));  //it return ⟹[ 'JavaScript',
'JavaScript' ]

// console.log(text.match(/javascript/ig));  // it avoid case-sensitive if we use
regular expression




//*5.matchAll():⟶ return an iterator of all the match's providing detailed
//*information about each match return an empty iterator if no match is found


// let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
// console.log(text.matchAll('javaScript'));  // if no match found it return ⟹
Object [RegExp String Iterator] {}

// console.log(text.matchAll('JavaScript'));  //also return same ⟹ Object
[RegExp String Iterator] {}

//!Tips:here the JS convert the normal text into regular expression
text.match(/javascript/)
//!also add the  the g falg at the end
```

```

// let matchAll=text.matchAll('JavaScript')
// console.log( ...matchAll);   //after this it give the result




//*6.includes():→return true if the string contain the specified value
//* and false otherwise case-sensitive

// let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
// console.log(text.includes('Java'));  //return true beacuse it match in 'JavaScript'

// console.log(text.includes('Javac'));  //return false because it not match


//*7.StartWith():→the startWith() method return true if string begins with
//* a specified value otherwise it return false



// let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
// console.log(text.startsWith('JavaScript'));  //return false because string start with "Hello'

// console.log(text.startsWith('hello'));  // rteurn false it is a case-sensitive

// console.log(text.startsWith('Hello'));  //return true because string start with 'Hello'




//*endWith():→ the endWith() method  return true if a string end with a
//*specified value otherwise it return false

// let text  ='Hello JavaScript,welcome to our world best JavaScript Course';
// console.log(text.endsWith('Hello'));  //return false beacuse string end with Course not Hello

// console.log(text.endsWith('course'));  //return false because it is case-sensitive

// console.log(text.endsWith('Course'));  //return true because string end with Course





//? 4. Extracting String Parts

//*1.Substr():→ it is deprecated
```

```

//*2.Slice():→ extract a part of a string and return the extracted part in
//* a new string.but not including last index

//syntax:-👇👇

//slice(start,end)


//example:👇👇

// let text='Adarsh'
// console.log(text.slice(1,4));  //retrun dar start from 1st index and end 3rd index not include 4th index

// console.log(text.slice(1));  //return darsh start with 1st and include all till end




//*Substring():→ extract a portion of the string based on starting and ending indices.
//*it also not include last index


//syntax:-👇👇
//1.substring(index)
//2.substring(startIndex,endIndex)


//!Notes:-
//substring () is similar to slice() the difference is that start and end
// values less than 0 are treated as 0 in substring ()

// let text ='Adrash Rai'
// console.log(text.substring(-1));   //return Adrash Rai it treat 0 and print all data from 0 to end index


// console.log(text.substring(1,5));  //return dras start from 1 and end at 4th index


//!similarities:-both slice and  substring exclude the endindex




//!Interview Question :-👇👇👀👀

//!1.What is the output for the following code?

// let text='Hello JavaScript,welcome to our world best JavaScript Course';
```

```javascript
// console.log(text.slice(1));  // start from 1st index and go to end

// console.log(text.substring(1));  //same output







//? 5. Extracting String Characters

//there are 3 method for extracting string characters

//1.The chatAt(position)
//2.the charCodeAt(position)
//3.the at(position)




//*1.The chatAt(position)

//the charAt() method return the character at a specified index(position)
//in a string if not it give empty string

// let text='Adarsh Rai'
// console.log(text.charAt(5));  //return h because it present at 5th index


// console.log(text.charAt(20));  //no char found at 20th index so it give empty string







//*1.The chatCodeAt(position)

//the charCodeAt() method return the code of the character at a specified index
//in a string. the method return a UTF-16 code (an integer between 0 and 65535)


// let text ='Adarsh Rai'
// console.log(text.charCodeAt(5));  //return 104 h is present at 5th index and it's charCode is 104




//*3.the at(position)

//the at() method return the character at a specified index in a string.
```

```
//the at() method return the same as charAt()

// let text='Adarsh Rai'
// console.log(text.at(5));  //return h because h is present at 5th index

//!it also take negative value
// console.log(text.at(-2));   //at -2 index a is present


//!Note
//1.the at() method is a new addition to javascript
//2.it allow the use of nagative index while charAt() do not
//3.it introduce in ES2022



//? 6. Replacing String Content

//*1.replace():→the method is used to replace a specified value with another value

// const text='Adarsh Rai'
// console.log(text.replace('Adarsh','Vivek'));  //Adarsh replaced with Vivek


//!case-insensitive Replacement : to perform a case-insensitive replacement you can use the
//! i falg in the regular expression

// let text='Adarsh Rai'
// console.log(text.replace(/adarsh/i,'Vivek'));  //it  avoid case-sensitive and replace adarsh with Vivek



//*2.replaceAll():- the method is used to replace all match value with another value

 let text='Hello Adarsh Rai,Good Morning Adarsh'
// console.log(text.replaceAll(text.replaceAll('Adarsh','Vivek')));



//!To avoid case-sensitive  use regular expression

// console.log(text.replaceAll(/adarsh/gi,'Vivek'));  //g falg for change all match value



//? 7. Other Useful Methods

//*1.toUpperCase and toLowerCase:→ convert the string to uppercase or lowercase

```

```
// const str='javaScript'
// console.log(str.toLowerCase());  //return ⟹javascript convert string into lowerCase
// console.log(str.toUpperCase());  //convert return ⟹JAVASCRIPT string into uppercase



//*2.trim:→ remove whitespace from both end of the string

// const str='    hello    '
// console.log(str.trim());   //remove white space from both end




//*3.split():→ split the string an array or substring based on a specified delimiter

// const str='apple,orange,banana'
// console.log(str.split(",")); //return [ 'apple', 'orange', 'banana' ] split based on ","

// console.log(str.split(""));  //split based on empty string


//*to reverse the array

// console.log(str.split(",").reverse());  //reverse the array [ 'banana', 'orange', 'apple' ]


//*to get back into the original string

// console.log(str.split(",").join());  //use join to get vack into the original string  apple,orange,banana







//!InterView Question 👇👇👇👀👀

//!1.write a javascript function that print the letter 'a' through 'z' in the console
//!you should use a loop to iterate through the letter and print each one on a new line


// const charCodeAt=()⇒{
//     for(i=97;i⩽122;i++){
//         console.log(String.fromCharCode(i));
//         fromCharCode() used to convert back to the string in this
//          we want to pass integer value
//     }
// }
// charCodeAt()




//!2.write a function to count the number of vowels in a string

// const countVowels=(str)=>{
//     const vowels='aeiou'
//     let count =0
//     for(let char of str){
//         if(vowels.includes(char)){
//             count++
//         }
//     }

//     return count

// }
// console.log(countVowels('hello a e i o u Morning'));





//!3.write a function to check if all the vowels presnet in a string or not

// const checkVowelsPresent=(str)=>{
//     const vowels='aeiou'
//     for(let char of vowels){
//         if(!str.includes(char)){
//             return false
//         }
//     }
//     return true
// }

// console.log(checkVowelsPresent('hello  e i o'));





//! write a javascript function ispangram that takes a string as input and return true
//! if the string is a panfram(contain all letter of the alphabet) and false otherwise
//!the function should be case insensitive and ignore spacse


//*pangram:-> a string contain A TO Z alphabet
```

```

// const ispangram=(str)=>{
// let alphabet=''
// for(let i=97;i<=122;i++){
//     alphabet=alphabet+String.fromCharCode(i)
// }
//     for(let char of alphabet){
//         if(!str.includes(char)){
//             return false
//         }
//     }
//     return true
// }

// console.log(ispangram('the quick brown fox jumps over the lazy dog'));





                            //* #. 💫💫 JavaScript Math Object 💫💫


//*Math:->the math object contain static properties and methods
//* for mathematical constant and function

//*Math work with the number type it does't work with BigInt


//?1.Constants

//*i Math.PI:->represent the mathematical constant PI

// console.log(Math.PI);  //return 3.141592653589793

//?Basic Operation

//*ii Math.abs:-> the math.abs() return the absolute value of a number
//* how far the number is from 0

// console.log(Math.abs(5));  //return 5 always return positive value

// console.log(Math.abs(-5));  //convert negative number to positive value


//*iii Math.round(x):->round a number to the nearest Integer

// console.log(Math.round(4.9));
// console.log(Math.round(4.5));


//*Vi Math.floor(x):->return the value of a x rounded down to its
//*nearest integer

```

```
// console.log(Math.floor(3.7));  //convert to the down nearest value



//*Math.ceil(x):→return the value of x rounded up to its nearest value

// console.log(Math.ceil(3.7));  //convert up its nearest value


//*Math.trunc(x):→ return integer party of x

// console.log(Math.trunc(3.7));  //return 3 because it is a integer part






//!Difference between floor and trunc
// console.log(Math.floor(-3.7));  //it return rounded down value
// console.log(Math.trunc(-3.7));  //it return integer part






//*Math.min(x,y):→ return min value

// console.log(Math.min(5,10));  //return 5



//*Math.max(x,y):→ return max value

// console.log(Math.min(5,10)); //return 10



//?Exponitional and logarithmic function

//*i.Math.pow(x,y):-return the value of x to the power of y

// console.log(Math.pow(4,2));  //4 square is 16



//*ii.Math.squrt(x):→ return the square root of x

// console.log(Math.sqrt(4));  //square root of 4 is 2


//*Math.log(x):→ return the natural logarithm of x

// console.log(Math.log(1)); //return 0
// console.log(Math.log(2));  //return 0.6931471805599453
```

```


//*iii. Math.log2(x):→ return the base 2 logrithm of x

// console.log(Math.log2(5));





//*Math.random():→Math.random() return a random number
//* between 0(inclusive) and 1(exclusive)

// console.log(Math.random()); //return 0.6680246362533855  its a random number


// console.log(Math.round(Math.random()*10));  //now return the number between 1 to 10


// console.log((Math.random()*100).toFixed(3)); // only return 3 value after decimal








                         //* #. 💫💫 Window In JavaScript 💫💫

//?.Window Object :=> the window object represent the global window in a browser
//? Both the (BOM) and (DOM) are part of the window object


//? BOM(Browser object model) :=>the BOM represent the browser as an object and
//? provide method and properties form interacting with browser itself
//? (not directly related to the content of a web page)


//Example of BOM features include window.navigator for browser info
// window.location for URL manipulation and window.alert for displaying alert



//? DOM(Document object Model):=> the dom represent the structured document
//?  as a tree of object where each object corresponds to a part of the
//? document(such as element , attributes and text)

//Dow allow javascript to intreact with and manipulate the HTML element


//! Summery ⇒ so while the DOM is focused on the content of the page the
//! BOM is focused on the browser environment. the winode object serve as a
//! global object





//* Window Object
//<button onclick="window.open('https://hacklikeberlin.netlify.app/','_blank')">
//navigate
//</button>

//👆👆 here if we click navigate button we go the the given website




//*Window History  Object

//1.<button onclick="history.back()">Go Back</button> ⟹ go back to the backward privious website

//2. <button onclick="history.forward()">Go Forward</button>  ⟹ go forward to the new open website

//3. <button onclick="history.go(-1)">Move Backwoard or forward</button>  ⟹ move any forward or backward





//*Location Object

//1.<button onclick="location.reload()">Reload The Page</button>  ⟹ here we use to reload the page(refresh page)

//2. <button onclick="location.replace('https://hacklikeberlin.netlify.app/')">Nagivate Page using replace</button> ⟹here we go to the given URL but
// problem is that we can't go back or forward (← →  this is disable)

//3.<button onclick="location.assign('https://hacklikeberlin.netlify.app/')">Navigate page using Assign</buttono>  ⟹ here we go the given URL but benifit is that we can go privious website (← → this is enable)
```

```
//* DOM Navigation
//go to 👉👉👉👉 \JavaScript_Full_course\window object\bom\DOM\DomNavigation.html

//*Dom  Searching

//? getElementById(id): Find an element by its ID.

//? getElementsByClassName(className): Find elements with a specific class name.

//? getElementsByTagName(tagName): Find elements with a specific tag name.

//? querySelector(selector): Find the first element that matches the specified CSS selector.

//? querySelectorAll(selector): Find all elements that match the specified CSS selector.


//go to 👉👉👉 \JavaScript_Full_course\window object\bom\DOM\DomSearching.html



//* ==========================================
//* DOM - CRUD (Create, Read, Update, Delete):
//* ==========================================

//? createElement(tagName): Create a new HTML element.

//? appendChild(node): Append a node as the last child of a parent node.

//? removeChild(node): Remove a child node from its parent.

//? addEventListener(event, function): Create an event listener to handle events.

//? removeEventListener(event, function): Remove an event listener.

//? setAttribute(name, value): Set the value of an attribute on an element.

//? getAttribute(name): Get the value of a specific attribute on an element.

//? innerHTML: Read or update the HTML content of an element.

//? textContent: Read or update the text content of an element.



//go to 👉👉👉 \JavaScript_Full_course\window object\bom\DOM\DOMCURD.html
```

```javascript



                    //* #. 💫💫 Events In JavaScript 💫💫

//todo 1. 3 ways of writing Events in JS
//todo 2. What is Event object
//todo 3.MouseEvent in JavaScript
//todo 4.KeyBoardEvent in JavaScript
//todo 5.InputEvent in JavaScript


// go to 👉👉 \JavaScript_Full_course\Events


                    //* #. 💫💫 Local Storage In JavaScript 💫💫

//? it allow you to save key/value parirs in the browser
//? stores data with no expiration data
//? the data is not deleted when the browser is closed and are available
//? for future sessions



//? what we will cover

//todo how to add/store data in localStorage
//todo how to get/retrive data in localStorage
//todo how to remove data in localStorage


//? how to add/store data in localStorage
// localStorage.setItem('name','adarsh')   //name is a key and adarsh is a value


//? how to get/retrive data in localStorage
// localStorage.getItem('name')  return adarsh  if key does't exist it give null value

//? how to remove data in localStorage
// localStorage.removeItem('name')


//! to check data go to browser and inspect  then go to application > localstorage >Chrome://new-tab



//todo  Note:=>local storage can only store string, so when you want to store
//todo a complex data structure like an array or an object you need to convert it to a string
//todo using JSON.stringify()

```

```
//*1.JSON.Stringify()
//? Convert A JavaScript object into a JSON string
//? useful when you want to send data to a server or store it in a text file
//? as JSON is a common data interchange format



//Example:👇👇

// const data={name:'Adarsh',age:21,city:'thane'}

// const JsonString=JSON.stringify(data)
// console.log(JsonString);  //now convert data into string


//*1.JSON.Parse()
//? convert a JSON string into a JavaScript object

// const ParseData=JSON.parse(JsonString)
// console.log(ParseData);  //convert back to the object

//!check the project in JavaScript_Full_course\LocalStorage and Json Project


                    //* #. 💫💫 Date and Time Object 💫💫

//?The Date() constructor create Date object when called as function,it
//? return a string representing the current time


//synatx 👇👇
//1.new Date()
//2.new Date(date string)



//? 9 ways to create a new Date objects

//1.new Date()
//2.new Date(date string)
//3.new Date(year,month)
//4.new Date(year,month,day)
//5.new Date(year,month,day,hours)
//6.new Date(year,month,day,hours,minutes)
//7.new Date(year,month,day,hours,minutes,seconds)
//8.new Date(year,month,day,hours,minutes,seconds,millisecond)
//9.new Date(millisecond)
```

```


//*1.new Date()⇒create a date object representing the current date and time

// const currDate=new Date()
// console.log(currDate);  //return 2025-04-29T06:36:38.239Z

//👆👆👆👆 this is a ISO 8081 formate which is standard for representing date and time
//in this format the date and time are represented together seperated by the letter
// 'T' and 'Z' at the end indicate that the time is in UTC (cordinated universal time)

//!if we run in browser it return
// Tue Apr 29 2025 12:10:28 GMT+0530 (India Standard Time)


//*2.new Date(date string)
// const dateString=new Date('2025-04-25T11:25:43.136Z')
// console.log(dateString);  return 2025-04-25T11:25:43.136Z



//*3.new Date(year,month)
// const date1=new Date(2025,4)
// console.log(date1);

//*4.new Date(year,month,day)
// const date2=new Date(2024,1,19)
// console.log(date2);


//*5.new Date(year,month,day,hours)
// const date3=new Date(2024,1,19,10)
// console.log(date3);


//*6.new Date(year,month,day,hours,minutes)
// const date4=new Date(2024,1,19,10,44)
// console.log(date4);

//*7.new Date(year,month,day,hours,minutes,seconds)
// const date5=new Date(2024,1,19,10,44,30)
// console.log(date5);

//*8.new Date(year,month,day,hours,minutes,seconds,millisecond)
// const date6=new Date(2024,1,19,10,44,30,500)
// console.log(date6);


//*9.new Date(millisecond)
// const dateFromMillisecond=new Date(1745583148462)
// console.log(dateFromMillisecond);
```

```


//! Note 📚📚
//! javascript count month 0 to 11  [jan=0,feb=1..] and [sun=0 mon=1...]
//! javascript store data as millisecond: javascript store data as number of
//!millisecond since jan 01 1970


//? date string formate : if the datestring is in a recognizable formate
//? the date object will be created accordingly

// const date1=new Date('2024-01-05')
// const date2=new Date('january 5 ,2024')

// console.log(date1);
// console.log(date2);   //both give correct data





//*JavaScript Get Date Methods 👇👇

//?1.GetFullYear 👇👇
// const currDate=new Date()
// const year=currDate.getFullYear()
// console.log(year);  //return 2025


//?2.GetMonth 👇👇
// const month=currDate.getMonth()
// console.log(month);
// console.log(currDate);  //return 3 {april index is 3}



//?2.GetDate 👇👇
// const date=currDate.getDate()
// console.log(date);  //return 26{current date}



//?2.GetDay👇👇
// const day=currDate.getDay()
// console.log(day);  /return 6 {saturday index number is 6 and sun index number
is 0}



//*JavaScript Get Time Methods 👇👇


//?1.GetHours👇👇
// const hour=currDate.getHours()
// console.log(hour);  //return 16 {4'o clock}



//?2.GetMinutes👇👇
// const minute=currDate.getMinutes()
// console.log(minute);  //return 10 {10 minutes }



//?3.GetSecond👇👇
// const second=currDate.getSeconds()
// console.log(second);   //return 30 {30 second}



//?4.GetMillisecond👇👇
// const millisecond=currDate.getMilliseconds()
// console.log(millisecond);  return 576 {576 millisecond as 1s=1000ms}


//?5.GetTime👇👇
// const time =currDate.getTime()
// console.log(time);  //return 1745910454425 ms from jan 1 1970



//*JavaScript Set Date Methods 👇👇

// const date=new Date()


//?1.setFullYear 👇👇
// date.setFullYear(2026)
// console.log(date);  return 2026-04-23T11:45:02.642z  {2026 is set}


//?2.setMonth 👇👇
// date.setMonth(5)
// console.log(date); //return 2026-06-23T11:45:02.642z


//?3.setDate 👇👇
// date.setDate(15)
// console.log(date); //return 2026-06-15T1:45:02.642z





//*JavaScript Set Time Methods 👇👇

//?1.setHours 👇👇
// date.setHours(10)
// console.log(date);  //return 10:26:24 {run in browser}


//?2.setMinutes 👇👇
// date.setMinutes(30)
// console.log(date);  //return 10:30:24


//?3.setSecond 👇👇
// date.setSeconds(45)
// console.log(date);  //return 10:30:45


//?4.setMillisecond 👇👇
// date.setMilliseconds(450)
// console.log(date);   //return 10:30:45 {we can't see millisecond}



//?5.setTime
// date.setTime(1832090690883)
// console.log(date);  //return Fri Jan 21 2028 23:34:50 GMT+0530 (India Standard Time)





//* A few useful methods of the date object in javascript

//?1.toLocalString():→ return a string representing the date and time portion of a date
                        //? object using the current locals conventions

// console.log(date.toLocaleString()); //return '1/21/2028, 11:34:50 PM'



//?2.toLocalDateString():→return a string representing the date and time portion of a date
                        //? object using the current locals conventions
// console.log(date.toLocaleDateString());  //return 1/21/2028



//?3.toLocalTimeString():→ →return a string representing the time portion of a date
                        //? object using the current locals conventions
// console.log(date.toLocaleTimeString()); //return 11:34:50 PM




//?parse():→ parse a string representation of a date and time return the number
```

```
            //? of millisecond since jan 1,1970 00:00:00 UTC
// console.log(Date.parse('2025-04-28T05:16:30.238Z'));  //return 1703054200000 here Date is a object





//*Bonus 👇👇

//?Date.now()
// let newDate=new Date()
// console.log(Date.now());  //return 1745912201688 this is current date and time
// console.log(newDate.getTime());  //also this return same


//!it return the current timestamp (number of millisecond) representing the current moment
//! use new Date().getTime() if you have an existing Date object from elsewhere and
//! want it's timestamp


//*interview Question 👇👇📚📚💭🤔🔧

//!2.write a function to add a specified number of days to a given date

// const addDate=(date,extraDate)=>{
//      return new Date(date.setDate(date.getDate()+extraDate)).toLocaleString()


// }

// const date=new Date('2025-04-10')
// console.log(addDate(date,7));


//!3.Write a function to calculate the difference in days between two given dates

// const getDaysDifference=(date1,date2)=>{
//     let oneDayMs=24*60*60*1000
//     let diff=date2-date1
//     console.log(diff);

//     return diff/oneDayMs
// }

// const date1=new Date('2024-02-19')
// const date2=new Date('2024-03-01')
// console.log(getDaysDifference(date1,date2));




```

```
                    //* #. 💫💫 List of Timing Based Events 💫💫

//todo1. setTimeout()
//todo1. setInterval()
//todo1. clearTimeout()
//todo1. setInterval()



//*.setTimeout() 👇👇
//? used to execute a function or code block after a specified delay in ms

//syntax 👇👇

// setTimeout(()=>{
//     console.log('it sehedule the callback function after a delay of 2000 ms');

// },2000)  //after 2 sec function is run and then stop


//*.setInterval() 👇👇
//? used to repeatedly execute a function or code block after a specified interval in ms

//syntax 👇👇

// setInterval(()=>{
//     console.log('it sehedule the callback function after a delay of 2000 ms');

// },2000)  //after 2 sec function is run and run repeately until we stop


//!setInterval=> execute a function after and after a particular time
//!setTimeout=> execute a function  after a particular time and then stop




//*clearTimeout() 👇👇
//? if you want to cancle a scheduled timeout before it occur,you can
//? use the clearTimeout function

//? it cancles a timeout priviously established by calling function


//syntax 👇👇
// clearTimeout(timeOutId)


//Example 👇👇

// const clearTimeOut=setTimeout(()=>{
//     console.log('clear the time out');

// },2000)

// clearTimeout(clearTime)   //it stop the execution of the function




//*clearInterval() 👇👇
//? if you want to cancle a scheduled interval before it occur,you can
//? use the clearInterval function

//syntax 👇👇
// clearInterval(IntervalId)


//Example 👇👇

// const clearIntervals=setInterval(()=>{
//     console.log('clear interval before it occur');

// },2000)

// clearInterval(clearIntervals) //it stop the execution of the function


//!write a javaScript program that defines a function called repeatedFunction.
//!this function should log the message "this function repeat every 1000 ms(1sec)"
//!then,set up interval using setInterval() to call repeatedFunction every 1000
//!ms. additionally after 5 second have elapsed,use setTimeout() to clear the
//! interval previously set up.make sure log the message "interval cleared after "
//!after 5 sec" when the interval is cleared

// const repeatedFunction=()=>{
//     console.log('this function repeat every 1000 ms(1sec)');

// }

// const intervalId=setInterval(repeatedFunction,1000)

// setTimeout(()=>{
//     clearInterval(intervalId)
// },5000)   //here we stop the function after 5 second


//! the projects in JavaScript_Full_course\TimeBasedEvents



                //* #. 💫💫 Object In JavaScript💫💫


```

```
//*What is the object 🤔🤔
//? Objects are a fundamental part of JavaScript, providing a way to group
//? related data and functions together. In JavaScript, an object is a collection
//? of key-value pairs, where each key is a string (or a symbol) and each value can
//? be any data type, including other objects.

//? Objects can have properties and
//? methods, making them versatile for various use cases.


//syntax  👇👇

// const obj={}


//*Creating Object

//? There are several ways to create objects in JavaScript.
//? The most common one is using object literals: 😬👇👇


// let person = {
//     name: "Adarsh",
//     age: 30,
//     "is'Student": false,
//     greet: function () {
//       console.log("Welcome to World Best JavaScript Course");
//     },
//   };    //here person is a object name is a key and Adarsh is a value






//*Accessing Properties

//? You can access object properties using dot notation
//? or square bracket notation:

// let person={
//     name:'Adarsh',
//     age:22,
//     "is'Student":false,
//     greet:function (){
//         console.log('hello');

//     }
// }

// console.log(person.age);  //return 22
// console.log(person.name);  //return Adarsh
//console.log(person.is'student);  // give error so we use square bracket
```

```
// console.log(person["is'Student"]);  //always put in string or back
tick[tamplate litrals]






//*Adding and Modifying Property

//? You can add new property or modify existing once

// person.job='web dev'     //here we add a new  property in person object
// person.age=21;           //here we modify the age
// person['age']=21       //both are same
// console.log(person);  👇👇👇👇

//!Result 👇👇

// let person={
//     name:'Adarsh',
//     age:21,
//     "is'Student":false,
//     greet:function (){
//         console.log('hello');

//     },
 //      job:'web Dev',
// }





//*Methods In Object

//? Methods in objects are functions associated with the object.
//? They can be invoked using the same notation as properties:

// console.log(person.greet);  //it return [Function:greet]

//To avoide this use 👇👇
// console.log(person.greet());  //return hello





//*We Can Add Dynamic Key in an Object


// let idType="studentId";
// let student={
//     idType:'A123456',
//     s_name:'Adarsh',
//     s_age:22,
//     "is'Student":false,
//     greet:function (){
//         console.log(`hey my ${idType} is ${student[idType]} and my name is ${student.s_name}`);

//     }
// }

//? if we want to make idType dynamic we use []

// let idType="studentId";
// let student={
//     [idType]:'A123456',
//     s_name:'Adarsh',
//     s_age:22,
//     "is'Student":false,
//     greet:function (){
//         console.log(`hey my ${idType} is ${student[idType]} and my name is ${student.s_name}`);

//     }
// }

// student.greet()   // return Hey my studentId is A123456 and my name is Adarsh





//*Data Modeling

//? Data modeling is the process of creating a visual representation of
//? either a whole information system or parts of it to communicate
//? connections between data points and structures. The goal is to illustrate
//? the types of data used and stored within the system, the relationships among
//? these data types, the ways the data can be grouped and organized and its
//? formats and attributes.


//? Objects are excellent for modeling real-world entities.
//? For instance, you might represent a car, a user, or a product as
//? an object with properties like color, brand, username, etc.

// let car ={
//     brand:'Toyota',
//     model:'camry',
//     year:'2022',
//     start:function(){
//         console.log('start');

//     }
// }   //all info of the car
```

```
//!Interview Question

//! Explain the difference between passing objects by reference and by value in
JavaScript. Provide an example to demonstrate each scenario.

//? sol: In JavaScript, primitive data types like numbers and strings are
//? passed by value, while objects are passed by reference.

//? Passing by value:=> When passing by value, a copy of the primitive value
//? is created and passed to the function or assigned to a variable. Any changes
//? made to the copy do not affect the original value.

// let a=10;
// const modifiedValue=(x)⇒(x=20)
// console.log(modifiedValue(a));
// console.log(a);  //here original data is not change in pass by value



//? Passing by reference: When passing by reference, a reference to the
//? memory location of the object is passed to the function or assigned to a
//? variable. Any changes made to the object through this reference will affect
//? the original object.



// let obj={id:5,name:'Adarsh'}

// let obj1=obj
// console.log(obj1);   //it copy the obj into the obj1 and return
{id:5,name:'Adarsh'}

// obj.name='Vivek'
// console.log(obj1);  //{ id: 5, name: 'Vivek' }
// console.log(obj);    //{ id: 5, name: 'Vivek' }  here value is change in both
obj



//? To avoid this behavior and create a true copy of the object,
//? you can use methods like Object.assign() or the spread operator (...):


//*1.Object.assign() :→is used to copy properties from one or more source
//* objects to a target object.

// let obj={id:5,name:'Adarsh'}

// let newObj=Object.assign({},obj)  //here obj copy in {}

// console.log(newObj);  //return {id:5,name:'Adarsh'}

// newObj.name='Vivek'
// console.log(newObj);  //it change  { id: 5, name: 'Vivek' }

// console.log(obj);  //here original data not change {id:5,name:'Adarsh'}




//*Comparison by Reference:👇👇

//? Two objects are equal only if they refer to the same object.
//? Independent objects (even if they look alike) are not equal:

// let obj1={name:'adarsh'}
// let obj2={name:'adarsh'}
// let obj3=obj1

// const isEqual=obj1==obj2 ? true:false

// console.log(isEqual);  //return false

// const isEqual=obj3==obj1 ? true:false
// console.log(isEqual);  //true because obj3 refer to the obj1 and both memory is same hence it is true





//* =====================================
//* "this" Object
//* ======================================


//? In JavaScript, the this keyword refers to an object.

// Which object depends on how this is being invoked (used or called).

// The this keyword refers to different objects depending on how it is used:

// In an object method, this refers to the object.
// Alone, this refers to the global object.
// In a function, this refers to the global object.
// In a function, in strict mode, this is undefined.
// In an event, this refers to the element that received the event.
// Methods like call(), apply(), and bind() can refer this to any object.

// Note
// this is not a variable. It is a keyword. You cannot change the value of this.
// ("use strict");



// const obj={
//     name:'Adarsh',
//     greet:function(){
//         console.log(this)
//     }
// }

// obj.greet()  return {name:'Aarsh',greet:[Function:greet]}  because this refer to obj


//? In this example, the greet method is defined using the
//? "Method Shorthand" syntax.
//? It's a more concise way to define methods in object literals.


//*for method shorthand syntax

// const obj={
//     name:'Adarsh Rai',
//     greet(){                    //this is method shorthand syntax we no write to fn keyword
//         console.log(this);

//     }
// }

// obj.greet()  //return { name: 'Adarsh Rai', greet: [Function: greet] }



//*For fat arrow function

// const obj={
//     name:'Adarsh Rai',
//     greet:()=>{
//         console.log(this);

//     },
// }

// obj.greet()   //return {}

//?👆👆👆 if i run in vs code arrow function do not have their own this
//? they look outside to find  what is this and in vs code this is just {}
//? so it return {}=>empty object

//?but if we run in console it return window object





//* ═══════════════════════════════════════
//* Object Useful Method
//* ══════════════════════════════════════


// const product={
//     id:1,
//     name:'Leptop',
//     category:'Computer',
//     brand:'ExampleBrand',
//     price:999.99,
//     stock:50,
//     description:'powerful laptop with a quad-core i5 processor 8gb ram,256gb
ssd',
//     image:'will be available',

// }


//*Object.keys():→Returns an array containing the names of all enumerable
//* own properties of an object.

// let keys=Object.keys(product)
// console.log(keys);

//return 👇👇
// [
//     'id',
//     'name',
//     'category',
//     'brand',
//     'price',
//     'stock',
//     'description',
//     'image'
//   ]





//*Object.value():return an array containing the values of all enumerable own
//* properties of an object

// let values=Object.values(product)
// console.log(values);

//rteurn 👇👇
// [
//     1,
//     'Leptop',
//     'Computer',
//     'ExampleBrand',
//     999.99,
//     50,
//     'powerful laptop with a quad-core i5 processor 8gb ram,256gb ssd',
//     'will be available'
//   ]

//*Object.entries():return an array containing array of key-value pairs for
//*each enumerable own properties of an object

// let entries=Object.entries(product)
// console.log(entries);  //convert object into array every key-value pair convert into array

//return 👇👇
// [
//     [ 'id', 1 ],
//     [ 'name', 'Leptop' ],
//     [ 'category', 'Computer' ],
//     [ 'brand', 'ExampleBrand' ],
//     [ 'price', 999.99 ],
//     [ 'stock', 50 ],
//     [
//       'description',
//       'powerful laptop with a quad-core i5 processor 8gb ram,256gb ssd'
//     ],
//     [ 'image', 'will be available' ]
//   ]


//*Object.hasOwnProperty(): return a boolean indicating whether the object has the
//*specified property as an own property.

// console.log(product.hasOwnProperty('name'));  //return true
// console.log(product.hasOwnProperty('isStudent')); //return false




//*Object.assign(): copies the value of all enumerable own property
//* from one or more source object to a target object.

// const target={a:1,b:2,b:5}
// const source={c:3,b:4}
// const newObj={}

// Object.assign(newObj,target,source)  //here target value and source value is copy to newObjcet


// console.log(newObj);  //{ a: 1, b: 4, c: 3 }  here source b:4 is overwrite to target b:2 and than we get only sourse b:4




```

```
//*object.freez():freezs an object,preventing new properties from being added to it
//*and existing properties from being modified or deleted


// Object.freeze(product)
// product.id='5656'
// console.log(product);  //we can't edit after applying object.freez




//!Interview Question


//!Given an object representing a student write a function to add a new
//!subject with it's corresponding grade to the student record also check if
//!the grade property is paresent or not


// let student={
//     name:'Bob',
//     age:20,
//     gardes:{
//         math:90,
//         science:85,
//         history:88
//     }

// }

// const addSubjectGrade=(obj,sub_name,grade)⇒{
//   if(!obj.gardes){
//     obj.gardes={}
//   }
//   return (obj.gardes[sub_name]=grade)
// }

// console.log(addSubjectGrade(student,"computer",92));
// console.log(student);






//!write a function that transform an array of an object into an
//!object where the key are the object ids
//! convert into this 👇👇
//! { '1': { id: 1, name: 'Adarsh' }, '2': { id: 2, name: 'Vivek' }, '3': { id: 3, name: 'Amit' } }


// let inputArray=[
//     {id:1,name:'Adarsh'},
//     {id:2,name:'Vivek'},
//     {id:3,name:'Amit'},
// ]


// const arrToObj=(arry)=>{
//     let obj={}
//     for(let key of arry){

//         obj[key.id]=key

//     }

//     return obj
// }

// console.log(arrToObj(inputArray));




                    //* #.  💫💫  How JavaScript work💫💫


//! 1: Parsing Phase
//* 1. Lexical analysis
//? In this phase JavaScript break the program into the tokens


//* 2. Syntax analysis
//? Takes the stream of tokens from lexical analysis and checks them
//? for correct syntax. If the syntax is correct, syntax analysis generates
//? a tree-like structure called a parse tree or abstract syntax tree (AST).
//? The AST represents the hierarchical structure of the program.

//* 3. Compilation (JIT - Just-In-Time Compilation):
//?here JavaScript convert the code into machine code/Bytes code

//* 3. Execution:
//? Once the code is compiled, the JavaScript engine executes
//? it. During execution, the engine creates execution contexts,
//? manages the scope chain, handles variable assignments, and calls functions.

//? The execution context consists of two phases:

//*i Creation Phase → Variable and function are hoisted
//*ii execution phase → where the code is actual run


//?JavaScript engine use a call stack and heap memory

//! Call Stack
//? In order to manage the execution contexts,
```

```javascript
//? the JavaScript engine uses a call stack.
//? The call stack is a data structure that keeps track of
//? the currently executing functions in a program. It operates on the
//? Last In, First Out (LIFO) principle, meaning that the last function
//? added to the stack is the first one to be executed and completed.

//! Heap Memory:
//? The heap memory is where dynamically allocated memory resides.
//? This is where objects, closures, and other dynamically allocated data are
//? stored. While the call stack manages the flow of execution and function
//?  contexts, the heap memory holds data that is referenced by these execution
//? contexts.


//*Synchronous Vs Asynchronous

//? Synchronous code executes line by line, blocking further
//?execution until each line is completed, while asynchronous
//?code allows other code to continue executing while it waits for
//?an asynchronous operation to complete.

//*Synchronous code

// const fun2=()=>{
//     console.log('fun2 start and end');

// }

// const fun1=()=>{
//     console.log('fun1 start ');
//     fun2()
//     console.log('fun1 end ');

// }

// fun1()   //here it run line by line




//*Asynchronous code

// const fun2=()=>{
//    setTimeout(()=>{
//     console.log('fun2 start and end');

//    },2000)

// }

// const fun1=()=>{
//     console.log('fun1 start ');
//     fun2()
//     console.log('fun1 end ');
// }

// fun1()  //here fun2 wait 2 sec and fu1 start run after executing fun2 also execute





//*Hoisting In JavaScript
//? Hoisting is a JavaScript mechanism where variables and function
//? declarations are moved to the top of their scope before code execution.
//? This means that no matter where functions and variables are declared,
//? they are moved to the top of their scope regardless of whether their
//? scope is global or local.

//!But it work only in case of var not let and const


// console.log(myVar);
// greet();

// var myVar = 10;
// function greet() {
//   console.log("Welcome, If you are reading this, Don't forget you are awesome");
// };



//* ───────────────────────────────
//*  Closure:
//* ───────────────────────────────

//? A closure is created when an inner function has access to the
//? variables of its outer function, even after the outer function has
//? finished executing.



// function outerFunction(){
//     var outerVariable='i am from outer'

//     function innerFunction(){
//         console.log(outerVariable);

//     }

//     return innerFunction;
// }

// var closureFunction=outerFunction()
// closureFunction()    // inner function access because the variable store in heap momery






                        //* #. 💫💫ECMASCRIPT 2015-2023 💫💫



//*ECMASCRIPT-2015

//* Destructuring Arrays:
//? Destructuring is a JavaScript expression that makes it possible to unpack
//? values from arrays, or properties from objects, into distinct variables.
//? That is, we can extract data from arrays and objects and assign them to
//? variables.

//? 1: Extracting specific elements:
// const numbers = [10, 20, 30];
// const first = numbers[0]; // Traditional way
// const [first, second, third] = numbers;
// console.log(second);  retun 20

//? 2: Ignoring elements:
// const [, , third] = numbers;
// console.log(third);


//!Write a program to swap two variable  without using third  variable

// let a=10;
// let b=20;

// [a,b]=[b,a]
// console.log(a,b);



//*Destructuring Object

// const user ={name:'Adarsh',age:30}

// const {myName,Myage}=user
// console.log(myName,Myage);   //return undefined because we should write only key name is object Destructuring

// const {name,age}=user
// console.log(name,age);   //now it give correct result
```

```

//*Rename Properties

// const user={
//     name:'adarsh',
//     age:30
// }

// const {name:myName,age}=user;   //here we rename name with Myname
// console.log(myName,age);



//*Spread Operator

//The syntax 👇👇

// ( ... )


//*1. Copying  an Array

// let fruits=['Apple','Orange','Mango'];

// let newFruits=[ ... fruits]
// console.log(newFruits);  //return ['Apple','Orange','Mango'];


//*1. Concatonating The Array

// const num1=[1,2,3]
// const num2=[4,5,6]

// const allNum=[ ... num1, ... num2]
// console.log(allNum);   //return [ 1, 2, 3, 4, 5, 6 ]



//*Adding Element to existing array

// let num=[1,2,3,4,5]
// num.push( ... [6,7,8,9,10])
// console.log(num);   //return [1,2,3,4,5,6,7,8,9,10]



//! One more useCases
//? In JavaScript, when you spread a string using the spread syntax ( ... ),
//? it converts the string into an array of its individual characters.


//*Traditional Way

// const country='INDIA'
// console.log(country.split("")); //return [ 'I', 'N', 'D', 'I', 'A' ]

//*New Way

// console.log([...country]);  //return [ 'I', 'N', 'D', 'I', 'A' ]


//* ==========================================
//*  Rest parameters  - Modern JavaScript
//* =========================================
//? The rest parameter syntax allows a function to accept an indefinite
//? number of arguments as an array, providing a more flexible way to work
//? with functions that can accept varying numbers of arguments.


// const sum=(...numbers)=>{
//     console.log(numbers);  //it is array [ 1, 2, 3, 4 ]

//     return numbers.reduce((acc,curr)=>{   //it is array so we use reducer function for operation
//         return acc+curr
//     })
// }

// console.log(sum(1,2,3,4));


//TODO NOTE: A function definition can only have one rest parameter, and the rest parameter must be the last parameter in the function definition.
// function wrong1(...one, ...wrong) {}    //there should be only one rest parameter
// function wrong2(...wrong, arg2, arg3) {}   //always put in last not first because it take all argument





//*ECMASCRIPT-2016

//*Exponentiation Operator

//? ES7 introduces a new mathematical operator called exponentiation operator.
//? This operator is similar to using Math.pow() method. Exponentiation operator
//? is represented by a double asterisk **. The operator can be used only with
//? numeric values.


//Synatx 👇👇

// base_value ** exponent_value

//Example 👇👇

```

```
// let base=2
// let expo=3

// console.log(base ** expo);   //return 8 as 2 to the power 3 is 8



//* ═══════════════════════════════════════════
//*    ECMAScript Features (2017) / ES8
//* ══════════════════════════════════════════

//? List of new useful features added in ES8  👇
//todo 1. String padding
//todo 2. Object.values()
//todo 3. Object.entries()
//todo 4. Trailing commas in function parameter lists and calls
//todo 5. Async functions

//* ═════════════════════
//*  String padding
//* ═════════════════════

//? String padding in JavaScript is a way to add extra characters (like spaces)
//? to a string to make it a specific length.

//? Using padStart() to pad from the beginning:

// const Myname='Adarsh'
// const padName=Myname.padStart(10)
// console.log(padName);  //it add space from beggning
// console.log(padName.length);

// console.log(Myname.padEnd(10,'$'));   //here we can also add any thing in palce of space  it add 4 star and length become 10




//* ═════════════════════
//*  Trailling Comma
//* ═════════════════════

// function greet(name,age,boolean,){   //here we can add commo it not give error
//     console.log(`hello ${name} you are ${age} year old`);

// }

// greet('Adarsh',30,true,)  //here also
```

```
//* ==========================================
//*    ECMAScript Features (2018) / ES9
//* =========================================

//? List of new useful features added in ES8  👇
//todo 1. Rest/Spread Properties
//todo 2. Promise.prototype.finally()



//* Rest/Spread Properties

// const number=[1,2,3,4]
// const [fisrt ,second,...other]=number   //here we use rest operator in array
// console.log(other);



//*Rest Opeartor in Object

// `const student={
//     name:'Adarsh',
//     age:10,
//     isStudent:true,
// }

// const {age,...other}=student
// console.log(other);`  //it return { name: 'Adarsh', isStudent: true } new obj



// const obj1={a:10,b:20,c:50}
// const obj2={c:30,d:40}
// const newObj={...obj1,...obj2}

// console.log(newObj);  //it return { a: 10, b: 20, c: 30, d: 40 }  here it overWrite c value in obj not array





//* ==========================================
//*    ECMAScript Features (2019) / ES10
//* =========================================

//? List of new useful features added in ES8  👇
//todo 1. Array.prototype.{flat,flatMap}
//todo 2. Object.fromEntries()
//todo 3. String.prototype.{trimStart,trimEnd}
//todo 4. Symbol.prototype.description
//todo 5. Optional catch binding


```

```

//*Array.flat() and Array.flatMap()

//? flat() is a new array instance method that can create a one-dimensional
//? array from a multidimensional array. (nested arrays into a single, flat array.)


// const array=[1,2,[3,4],5]
// console.log(array.flat());   //here 2-dimension array convert to single dimeansion



// const array2=[1,[2,[3,4]],5]
// console.log(array2.flat());   //it return [ 1, 2, [ 3, 4 ], 5 ]  it properly not convert in single dimension so we increae the level



// const array3=[1,[2,[3,4]],5]
// console.log(array3.flat(2));   //here we increase the level  and return [ 1, 2, 3, 4, 5 ]


// const array4=[1,[2,[3,[4]]],5]
// console.log(array4.flat(3));    //here we increase the more level



//? flatMap() is a new Array instance method that combines flat() with map().
//? It’s useful when calling a function that returns an array in the map()
//? callback, but you want your resulted array to be flat:


//*If we use Map

// const arr=['my name','is adarsh','rai']

// const newArr=arr.map((elem)⇒{
//     return elem.split(' ')
// })

// console.log(newArr);   //it return  multi dimension array [ [ 'my', 'name' ], [ 'is', 'adarsh' ], [ 'rai' ] ]



//*To avoid this we use flatMap()

// const arr=['my name','is adarsh','rai']

// const newArr=arr.flatMap((currElem)⇒{
//     console.log(currElem);

//     return currElem.split(" ")
// })

// console.log(newArr);   //now it return single dimension array[ 'my', 'name', 'is', 'adarsh', 'rai' ]




//* ============================
//*  Object.fromEntries()
//* =============================
//? Objects have an entries() method, since ES2017.
//? It returns an array containing all the object own properties,
//? as an array of [key, value] pairs:

// const person={name:'adarsh',age:30}
// const entries=Object.entries(person)
// console.log(entries);    //convert object to array in key pair form

// const newPerson=Object.fromEntries(entries)
// console.log(newPerson);   //convert back to the object from array



// console.log(person===newPerson);   //here perosn and new person are not same obj because
                                        //there memory location is difference we compare object based on reference not values






 //* ========================================
//* String.prototype.{trimStart,trimEnd}
//* =========================================

//? trimStart(): Return a new string with removed white space from the start
//? of the original string
// console.log("Testing".trimStart());
// console.log("           Testing".trimStart());
// console.log("     Testing     ".trimStart());
// console.log("Testing    ".trimStart());

//? trimEnd(): Return a new string with removed white space from the end of
//? the original string
// console.log("Testing".trimEnd());
// console.log("           Testing".trimEnd());
// console.log("     Testing     ".trimEnd());
// console.log("Testing    ".trimEnd());

```

```


//* ================================
//* Symbol.prototype.description
//* ===============================

//? In JavaScript, a Symbol is a primitive data type introduced
//? in ECMAScript 2015 (ES6). It represents a unique identifier that is
//? immutable and guaranteed to be unique. Symbols are often used as
//? property keys in objects to avoid naming conflicts.

//? The Symbol.prototype.description property is a new feature introduced in
//? ECMAScript 2019 (ES10). It allows you to retrieve the description of a
//? symbol. When you create a symbol, you can optionally provide a description as
//? its parameter. The description property lets you access this description.

// const mySymbol = Symbol("This is my symbol");
// console.log(typeof mySymbol);
// console.log(mySymbol.description);


//* ================================
//*  Optional catch binding
//* ===============================
//? In ECMAScript 2019 (ES10), a new feature called "optional catch binding"
//? was introduced for the try...catch statement. This feature allows you to omit
//? the parameter in the catch block, making it optional.

//? We previously had to do: 👇
//try {
    // ...
 // } catch (e) {
    //handle error
//  }

  //? Now we can omit that optional parameter 👇
//   try {
//     10 + 5;
//   } catch {   //here parameter is omit
//     console.log("there is an error");
//   }





//* ==========================================
//*    ECMAScript Features (2020) / ES11
//* =========================================

//? List of new useful features added in ES8  👇
// BigInt
// Nullish Coalescing Operator ??
// Optional Chaining Operator ?.
// promise.allSettled



//* ===================
//*  BigInt
//* ================
//? BigInt: BigInt in JavaScript is a data type used to represent and perform
//? operations on large integers that exceed the limits of regular numbers.


//? Creating BigInts:
//? - Using the `n` suffix:

//Using the BigInt() constructor:
// const anotherLargeNumber = BigInt("123456789012345678901234567890");
// console.log(anotherLargeNumber);  //it return 123456789012345678901234567890n n show that num is big int


// let number=Number.MAX_SAFE_INTEGER
// console.log(number);  //9007199254740991 ==> this is max int value


// number=BigInt(number)
// let num=number+10n
// console.log(num);   //9007199254741001n now it become big int after adding 10n



//* =======================================
//*  Nullish Coalescing Operator ??
//* ===================================
//? In JavaScript, the nullish coalescing operator (??) is a logical operator that
//? provides a concise way to handle nullish (null or undefined) values.
//? It returns its right-hand operand when its left-hand operand is null or
//? undefined, otherwise, it returns the left-hand operand.

// let userFav=0;
// let userNumber =userFav ?? 10
// console.log(userNumber);     //because it is not null and undefined thats why it give  left side value


// let userFav=null
// let userNumber=userFav ?? 10
// console.log(userNumber);   //it is null thats why it give right side value



//* =======================================
//*  Optional Chaining Operator (?.)
//* ===================================
//? It provides a concise way to access properties of an object without worrying
//? about the existence of intermediate properties. It's particularly useful when
//?  working with nested objects or accessing properties of objects that may be
//? null or undefined.

// const person = {
//   name: "John",
//   address: {
//     city: 'newYork',
//     zipCode: 12345,
//   },
// };


// const city=person.address?.city
// console.log(city);   //return  newyork



// const person = {
//     name: "John",
//     address: {
//       city: "New York",
//       zipCode: 12345,
//       coordinates: {
//         latitude: 40.7128,
//         longitude: -74.006,
//       },
//     },
//   };


//   const latitude = person.address?.coordinates?.latitude ?? "not present";
//   console.log(latitude);  //retrun 40.7128


//* ══════════════════════════════════════════
//*    ECMAScript Features (2021) / ES12
//* ═════════════════════════════════════════

//? List of new useful features added in ES8  👇
// String.prototype.replaceAll()
// Logical Assignment Operators (||=, &&=, ??=)
// Numeric Separators
// Promise.any()

//* ══════════════════════════════════════════
//*  String.prototype.replaceAll()
//* ═════════════════════════════════════════
//? replaceAll in JavaScript is a function that replaces all occurrences of a
//? specified value with another value in a given string.

//? Replacing all occurrences of a word:
// const originalString = "Hello, world! Hello again.";
// const newString = originalString.replaceAll("Hello", "Hi");
// console.log(newString);

//? Replacing multiple spaces with a single space:
// const text = "This   has   extra       spaces.";
// const normalizedText = text.replaceAll(/\s+/g, " ");
// console.log(normalizedText);

//* =====================================================
//*  Logical Assignment Operators (||=, &&=, ??=)
//* ====================================================

//? Logical OR-Assignment (||=): This operator assigns the value of its
//? right-hand operand to its left-hand operand if the left-hand operand
//? evaluates to a falsy value (false, null, undefined, 0, '', NaN). Otherwise,
//? it leaves the left-hand operand unchanged.

// let x = false;
// x = x || true; // equivalent to: x = x || true;
// console.log(x); // Output: true

// let y = 10;
// y ||= 20; // equivalent to: y = y || 20;
// console.log(y); // Output: 10 (unchanged)

//? Logical AND-Assignment (&&=): This operator assigns the value of its
//? right-hand operand to its left-hand operand if the left-hand operand
//? evaluates to a truthy value. Otherwise, it leaves the left-hand operand
//? unchanged.
// let x = true;
// x &&= false; // equivalent to: x = x && false;
// console.log(x); // Output: false

// let y = 0;
// y &&= 20; // equivalent to: y = y && 20;
// console.log(y); // Output: 20

//* ======================
//*  Numeric Separators
//* ====================
//? This feature allows underscores (_) to be used as separators within numeric
//? literals to improve readability.
// const bigNumber = 1_000_000;
// console.log(bigNumber);
// Output: 1000000;




//* ==========================================
//*    ECMAScript Features (2022) / ES13
//* =========================================

//? List of new useful features added in ES8  👇
```

```
// .at() function for indexing
// Object.hasOwn(obj, propKey)




//* ══════════════
//*  .at()
//* ════════════
//? Before ES2022, square bracket notation was used to fetch a particular
//? element in an array. This method is straightforward unless you need to
//? perform a backward iteration, i.e., negative indexing. In the case of negative
//? indexing, the common practice was to use arr[arr.length — N],
//? where array length is referred to and indexed from the end.

//? The .at() method introduced in ES2022 has simplified this process. In a case of positive indexing, .at() will work the same as the square brackets. But for negative indexing, the .at() method allows starting the iteration from the end.

// const array = [1, 2, 4, 5, 6, 7];

// console.log(array.at(-1));  //return 7





//* ════════════════════════════════════
//*  Object.hasOwn(obj, propKey)
//* ════════════════════════════════════
//? Object.hasOwn() is a static method that you can use to check if a
//? property exists in an object or not. It returns true if the specified object
//? contains the indicated property as its own, and if the property is inherited
//? or doesn’t exist, it returns false. This method takes the object as the first
//? argument and the property you want to check as the second.

//? Object.hasOwn is the intended alternative for the Object.prototype.hasOwnProperty
//? method. Although Object.prototype.hasOwnProperty has been in JavaScript
//? specification for quite a time, it has some drawbacks.

// const book = {
//   name: "World Best JS Course",
//   author: "Thapa Technical",
// };


// console.log(book.hasOwnProperty('name'));  //return true


//! Issue with hasOwnProperty it does't work for object created
//! using Object.create(null)

```

```javascript

// const student =Object.create(null)
// student.name='Adarsh'
// console.log(student.hasOwnProperty('name'));  //it give erroe

// console.log(Object.hasOwn(student,'name'));  //it retun true result


//* ==========================================
//*    ECMAScript Features (2022) / ES13
//* =========================================

//? List of new useful features added in ES8  👇
// Array.findLast()
// Array.findLastIndex()
// Array.prototype.toReversed()
// Array.prototype.toSorted(compareFn)
// Array.prototype.toSpliced(start, deleteCount, ...items)
// Array.prototype.with(index, value)

//* ===============================================
//*  Array.findLast() & Array.findLastIndex()
//* ==============================================
//? This function will allow us to find element from the last to first
//? of the array based on a condition.
// const array = [1, 2, 3, 4, 5, 6, 4];
// console.log(array.findLast((elem) => elem > 4));
// console.log(array.findLastIndex((elem) => elem));

//* ===============================================
//*  New Array.prototype functions
//* ==============================================

const myNames = ["Adarsh", "Rai", "Vivek", "HackLikeBerlin"];

//* Array.prototype.toReversed();
// const reversedNum = myNames.toReversed();
// console.log("original", myNames);
// console.log("reversed", reversedNum);
//todo if it's not working run in browser

//* Array.prototype.toSorted(compareFn);
// const sortedArr = myNames.toSorted();
// console.log("original", myNames);
// console.log("sorted", sortedArr);

//*Array.prototype.toSpliced(start, deleteCount, ...items);
// const splicedArr = myNames.toSpliced(1, 1, "thapaTechnical");
// console.log("original", myNames);
// console.log("spliced", splicedArr);

//* Array.prototype.with(index, value);
//?The with() method in JavaScript is used to change the value
//? of an element at a specific index in an array. It takes two arguments:
```

```
//? the index of the element to be changed and the new value.
//? It returns a new array with the changed element,
//? leaving the original array unchanged.

// const replaceWith = myNames.with(2, "thapaTechnical");
// console.log("original", myNames);
// console.log("replaced", replaceWith);



                      //* #. 💫💫Advanced JavaScript 💫💫

//todo 1.Event Propagation (Bubbling & Capturing)
//todo 2.Higher order function
//todo 3.Callback function
//todo 4. Closure & function curring
//todo 5.Callback hell
//todo 6.fetch Api
//todo 7.Promises
//todo 8. Asyn-await
//todo 9. Error Handling



//* 1.Event Propagation (Bubbling & Capturing)

//? it is the process of how event propogates or travel through the DOM hierarchy
//? in JS there are tow phase of event propagation capturing phase and bubbling phase



//*Capturing Phase
//? the event start from the root of the DOm and goes down to the target elem

//*Target Phase
//?the event reaches the target element

//*Bubbling Phase
//? the event start from the target elem and bubbles up to the root of the DOM
//? this is default

//!JavaScript_Full_course\Event_Propagation\Event_Propogation.html  ⟹ check example here




//* ═══════════════════════════════════
//*  Event Delegation:
//* ═══════════════════════════════════
//? Event delegation is a concept in JavaScript where instead of attaching event
//?listeners to individual elements, you attach a single event listener to a
//?common ancestor of those elements. This is particularly useful when you have a
```

```
//?large number of similar elements and want to reduce the number of event listeners,
//?improve performance, and simplify code.

//* Example: List with Delegation ⟶
//? Consider a scenario where you have an unordered list (<ul>) with multiple
//? list items (<li>), and you want to handle click events on each list item.
//? Instead of adding a separate event listener to each list item,
//?  you can use event delegation. ⟶</li>

//!JavaScript_Full_course\Event_Propagation\Event_delegation.html  ⟹ here is a example of event delegation



//* ──────────────────────────────────────────────────
//*  First-Class Function - it's just a concept
//* ──────────────────────────────────────────────────

//? A "first-class function" means that functions can be treated as
//? values, assigned to variables, and passed around as arguments.

// function sayHello(){
//     return 'hello'
// }

// var greetFunction=sayHello();
// console.log(greetFunction);    //this is first order functiom






//* ──────────────────────────────
//*  Higher-Order Functions:
//* ──────────────────────────────
//? Definition: A higher-order function is a function that takes one or
//? more functions as arguments or returns a function as a result.

//* ──────────────────────────────
//*  Callback Functions:
//* ──────────────────────────────
//? Definition: A callback function is a function passed as an argument
//? to another function and is executed after the completion of a task.

//* Here is the example ✅
// Callback function
// function processUserInput(name, greetUser) {
//   console.log("Received input: " + name);
//   greetUser(name);
// }

//*  Function to be used as a callback
```

```javascript
// function greetUser(name) {
//   console.log(`Hello! ${name}`);
// }

// processUserInput("Vinod", greetUser);

//* processUserInput ⟹
//? is a higher-order function because it takes another
//? function (callback) as an argument.
//* greetUser ⟹
//? is a callback function because it's passed as an argument to
//? processUserInput and gets executed after the completion of the main task



//* ═══════════════════════════════════
//* Interview Question:
//* ═══════════════════════════════════
//! Write a program to perform mathematical operations using callback
//! functions and two variables in JavaScript.


// const mathOperation = (a, b, operation) ⇒ {
//     return operation(a, b);
//   };

//   const add = (a, b) ⇒ {
//     return a + b;
//   };

//   const sub = (a, b) ⇒ {
//     return b - a;
//   };

//   console.log(mathOperation(5, 15, add));
//   console.log(mathOperation(5, 15, sub));





//* ════════════════════════════════
//*  Callback hell
//* ════════════════════════════════

//? Callback hell, also known as the Pyramid of Doom, refers to a
//? situation in asynchronous JavaScript programming where multiple nested
//? callbacks are used to handle asynchronous operations.
//? This often results in code that is difficult to read, understand,
//? and maintain due to its deeply nested structure.

// const studentData=()⇒{
//     setTimeout(()⇒{
//         console.log('hello i am adarsh ');
//         setTimeout(()⇒{
//             console.log('hello my last name is rai  ');
//             setTimeout(()⇒{
//                 console.log('born in mumbai  ');
//                 setTimeout(()⇒{
//                     console.log('i am from up  ');
//                     setTimeout(()⇒{
//                         console.log('i am mern stack developer  ');
//                         setTimeout(()⇒{
//                             console.log('i like cricket ');
//                     },1000)
//                 },1000)
//             },1000)
//         },1000)
//     },1000)
// },1000)
// }

// studentData()




//* ══════════════════════════════════════════
//*  Promise in JavaScript
//* ══════════════════════════════════════════

//?👉 In simpler terms, a promise is like a placeholder for the result
//? of an asynchronous operation. Or A container for the future result or value.

//* It can be in one of three states:

//? Pending: Initial state, neither fulfilled nor rejected.
//* Fulfilled(Resolved): The operation completed successfully.
//! Rejected: The operation failed or encountered an error.


//? Promises have built-in methods like then and catch to handle the results
//? of asynchronous operations when they complete or encounter errors, making it
//? easier to write asynchronous code that is more readable and maintainable
//? compared to traditional callback-based approaches.


//* ══════════════════════════════════════════
//* Using the Promise Constructor (Class):
//* ══════════════════════════════════════════

//? You can create a promise using the Promise constructor.
//? This involves creating a new instance of the Promise class,
//? which takes a function as an argument. This function, often referred
//? to as the "executor function," takes two parameters: resolve and reject.
//? You call resolve when the asynchronous operation is successful and reject
//? when it encounters an error.

```

```
// const Promise=new Promise(function(resolve,reject){
//   code
// })


//* ==========================================
//* 2: Using a Function (Promise Wrapper):
//* ==========================================

//? You can also create a promise by defining a function
//? that returns a promise. This function usually encapsulates
//? some asynchronous operation. Inside this function, you manually
//? create a promise and resolve or reject it based on the result of
//? the asynchronous operation.

// syntax
// function myPromiseFunction() {
//   return new Promise((resolve, reject) ⇒ {
//        code
//   });
// }




//!Exmaple 1. 👇👇👇

// const pr=new Promise((resolve,reject)⇒{
//     setTimeout(()⇒{
//        resolve('hii miss you')

//     },2000)
// })

// .then((res)⇒{
//     console.log(res);

// }).catch((error)⇒{
//     console.log(error);

// }).finally(()⇒{
//     console.log('dont worry we all miss you and keep smilling');

// })



//!Exmaple 2. 👇👇👇

// const studentName='Adarsh'

// const enrollStudent=(studentName)⇒{
//     return new Promise((res,rej)⇒{
//         setTimeout(()⇒{
```

```
//             const isSuccessfull=Math.random()>0.4;
//             if(isSuccessfull){
//                 res(`Enrollment successful for ${studentName}`)
//             }
//             else{
//                 rej(`Enrollment failed for ${studentName}. Please try again.`)
//             }
//         },2000)
//     })
// }


// enrollStudent(studentName)

// .then((res)=>{
//     console.log(res);

// }).catch((e)=>{
//     console.log(e);

// }).finally(()=>{
//     console.log("Enrollment process completed.");

// })





//* ==========================================
//* Promise Methods
//* ==========================================

//* Promise.all()
//? is used when you want to wait for all promises to complete
//? successfully. Reject state will throw an error.

//? Promise.allSettled()
//? is used when you want to wait for all promises to complete,
//? regardless of success or failure, and get information about their outcomes.

//* Promise.race()
//? is used when you are interested in the result of the first
//? promise that completes, regardless of success or failure.


//!All Example 👇👇👇

// const promise1=new Promise((resolve,reject)=>{
//     setTimeout(()=>resolve('First'),2000)
// })

// const promise2=new Promise((resolve,reject)=>{
//     setTimeout(()=>resolve('Second'),5000)
// })

// const promise3=new Promise((resolve,reject)=>{
//     setTimeout(()=>resolve('Third'),3000)
// })

// const promise4=new Promise((resolve,reject)=>{
//     setTimeout(()=>reject('Fourth'),100)
// })



// Promise.all([promise1,promise2,promise3,promise4])
// .then((res)=>{
//     console.log(res);

// }).catch((e)=>{
//     console.log(e);

// })



// Promise.allSettled([promise1,promise2,promise3,promise4])
// .then((res)=>{
//     console.log(res);

// }).catch((e)=>{
//     console.error(e);


// })



// Promise.race([promise1,promise2,promise3,promise4])
// .then((res)=>{
//     console.log(res);

// }).catch((e)=>{
//     console.log(e);

// })


//!JavaScript_Full_course\Promise\Promise.html  ==>check the project in this file




//* ==========================================
//*  Async and Await
//* ==========================================

//? async and await are JavaScript keywords introduced in ECMAScript 2017 (ES8)
//? that make asynchronous code look and behave more like synchronous code,
//? making it easier to write, read, and reason about asynchronous operations. --
>

//? async Function Declaration: The async keyword is used to
//? declare an asynchronous function. An asynchronous function returns a
//? Promise implicitly, even if the return value is not explicitly wrapped
//? in a Promise. Inside an asynchronous function, you can use the await
//? keyword to pause the execution of code until a Promise is resolved or
//? rejected. -->

//? await Operator: The await keyword is used to pause the execution
//? of an async function until a Promise is settled (resolved or rejected).
//? It can only be used inside an async function. When await is used with a
//? Promise, it waits for the Promise to resolve and returns the resolved value.
//? If the Promise is rejected, it throws an error that can be caught using a
//? try...catch block. -->



//!JavaScript_Full_course\Promise\Asyn-await.html  ==> check here project




//* ==========================================
//*  Error-handling in JS(try-catch)
//* ==========================================

//? In JavaScript, the try...catch statement is used for error handling.
//? It allows you to catch and handle exceptions (errors) that occur within a
//? block of code. Here's how it works:

//? try Block: The try block contains the code that you want to execute.
//? It is the block of code where you anticipate that an error might occur. -->

//? catch Block: The catch block follows the try block and is used to
//? catch any exceptions (errors) that occur within the try block.
//? If an exception occurs, JavaScript jumps to the catch block to handle
//? the error. The catch block takes an error object as a parameter,
//? which can be used to access information about the error, such as
//? its message or stack trace. -->

//? finally Block (Optional): The finally block, if provided,
//? is executed regardless of whether an error occurs or not.
//? It is typically used for cleanup tasks that should always be
//? performed, such as closing resources or releasing locks. -->



//!JavaScript_Full_course\Promise\error-handling.html  ==> check here for example


```

```

//*
═══════════════════════════════════════════════════════════════════════════
//*     😉 😉 😉 😉  🎊 🎊 🎊 🎊 🎉 🥳 🥳    ....END..      🎊 🎊 🎊 🎊 🎉 🥳 🥳 😉 😉 😉 😉
//*
═══════════════════════════════════════════════════════════════════════════
```