

```
// const chalk = require("chalk");
```

```
/***** Section 1☞ we need to do it in console *****/
```

```
// alert("Welcome, to Complete JavaScript course");
```

```
// console.log('Welcome, to complete JavaScript Course');
```

```
/***** Section 2☞ Code Editor for writing JS *****/
```

```
/***** Section 3☞ values and variables in JavaScript *****/
```

```
// var myName = 'vinod bahadur thapa';
```

```
// var myAge = 26;
```

```
// console.log(myage);
```

```
// Naming Practice
```

```
// var _myName = "vinod";
```

```
// var 1myName = "thapa";
```

```
// var _1my__Name = "bahadur";
```

```
// var $myName = "thapa technical";
```

```
// var myNaem% = "thapa technical";
```

```
// console.log(myNaem%);
```

```
// // *****
```

```
// // 📄 // 📄 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 📄
```

```
// 📄 // 📄 https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
```

```
// // *****
```

```
/*** Section 4 📄 Data Types in JavaScript ***/
```

```
// var myName = "vinod thapa";
```

```
// console.log(myName);
```

```
// var myAge = 26;
```

```
// console.log(myAge);
```

```
// var iAmThapas = false;
```

```
// console.log(iAmThapas);
```

```
// // typeof operator
```

```
// console.log(typeof(iAmThapas));
```

```
// DataTypes Practice
```

```
// console.log( 10 + "20");  
// 9 - "5"  
// console.log( 9 - "5"); //bug  
// "Java" + "Script"  
// console.log( "Java " + "Script");  
// " " + " "  
// console.log( " " + 0);  
// " " + 0  
// "vinod" - "thapa"  
// true + true  
// true + false  
// false + true  
// false - true
```

```
// console.log("vinod" - "thapa");
```

```
// 🧑🏻‍💻 Interview Question 1 🧑🏻‍💻
```

```
// Difference between null vs undefined?
```

```
// var iAmUseless = null;  
// console.log(iAmUseless);  
// console.log(typeof(iAmUseless));  
// //2nd javascript bug
```

```
// var iAmStandBy;
```

```
// console.log(iAmStandBy);  
// console.log(typeof(iAmStandBy));
```

```
// 🧑🏻🤖 Interview Question 2 🧑🏻🤖
```

```
// What is NaN?
```

```
// NaN is a property of the global object.
```

```
// In other words, it is a variable in global scope.
```

```
// The initial value of NaN is Not-A-Number
```

```
// var myPhoneNumber = 9876543210;
```

```
// var myName = "thapa technical";
```

```
// console.log(isNaN(myPhoneNumber));
```

```
// console.log(isNaN(myName));
```

```
// if(isNaN(myName)){
```

```
//   console.log("plz enter valid phone no");
```

```
// }
```

```
// NaN Practice 📝
```

```
// NaN === NaN;
```

```
// Number.NaN === NaN;
```

```
// isNaN(NaN);
```

```
// isNaN(Number.NaN);
```

```
// Number.isNaN(NaN);
```

```
// console.log(Number.isNaN(NaN));
```

```
// 🧑🏻🧑🏻🧑🏻 Interview Question 1 🧑🏻🧑🏻🧑🏻
```

```
// var vs let vs const
```

```
/* Section 5 📄 Arithmetic operators in JavaScript */
```

```
// console.log(5+20);
```

```
// 1️⃣ Assignment operators
```

```
// An assignment operator assigns a value to its left operand
// based on the value of its right operand.
// The simple assignment operator is equal (=)
```

```
// var x = 5;
```

```
// var y = 5;
```

```
// console.log("is both the x and y are equal or not" + x == y );
```

```
// I will tell you when we will see es6
```

```
// console.log(`Is both the x and y are equal : ${x == y}`);
```

```
// 2. Arithmetic operators
```

```
// An arithmetic operator takes numerical values
// (either literals or variables) as their operands and
// returns a single numerical value.
```

```
// console.log(3+3);
```

```
// console.log(10-5);
```

```
// console.log(20/5);
```

```
// console.log(5*6);
```

```
// console.log("Remainder Operator " + 27%4);
```

```
// 2 Increment and Decrement operator
```

```
// Operator: x++ or ++x or x-- or --x
```

```
// If used postfix, with operator after operand (for example, x++),
```

```
// the increment operator increments and returns the value before incrementing.
```

```
// var num = 15;
```

```
// var newNum = num-- + 5;
```

```
// console.log(num);
```

```
// console.log(newNum);
```

```
// Postfix increment operator means the expression is evaluated
```

```
// first using the original value of the variable and then the
```

```
// variable is incremented(increased).
```

```
// If used prefix, with operator before operand (for example, ++x),
```

```
// the increment operator increments and returns the value after incrementing.
```

```
// var num = 15;
```

```
// var newNum = --num + 5;
```

```
// console.log(num);
```

```
// console.log(newNum);
```

```
// Prefix increment operator means the variable is incremented first and then  
// the expression is evaluated using the new value of the variable.
```

```
// 3. Comparison operators
```

```
// A comparison operator compares its operands and  
// returns a logical value based on whether the comparison is true.
```

```
// var a = 30;
```

```
// var b = 10;
```

```
// Equal (==)
```

```
// console.log(a == b);
```

```
// Not equal (!=)
```

```
// console.log(a != b);
```

```
// // Greater than (>)
```

```
// console.log(a > b);
```

```
// // Greater than or equal (>=)
```



```
// console.log(a >= b);
```

```
// // Less than (<)
```

```
// console.log(a < b);
```

```
// // Less than or equal (<=)
```

```
// console.log(a <= b);
```

```
// 4 Logical operators
```

```
// Logical operators are typically used with Boolean (logical) values;
```

```
// when they are, they return a Boolean value.
```

```
// var a = 30;
```

```
// var b = -20;
```

```
// Logical AND (&&)
```

```
// The logical AND (&&) operator (logical conjunction) for a set of
```

```
// operands is true if and only if all of its operands are true.
```

```
// console.log(a > b && b > -50 && b < 0);
```

```
// Logical OR (||)
```

**// The logical OR (||) operator (logical disjunction) for a set of
// operands is true if and only if one or more of its operands is true.**

// console.log((a < b) || (b > 0) || (b > 0));

// Logical NOT (!)

// The logical NOT (!) operator (logical complement, negation)

// takes truth to falsity and vice versa.

// console.log(!((a>0) || (b<0)));

// console.log(!true);

// // *****

// //  //  SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 

//  //  <https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA>

// // *****

//  String Concatenation(operators)

// The concatenation operator (+) concatenates two string values together,

// returning another string that is the union of the two operand strings.

// console.log("Hello World");

// console.log("hello " + "world");

```
// var myName = "vinod";

// console.log(myName + " thapa");
// console.log(myName + " bahadur");
// console.log(myName + " bahadur Thapa");
```

// 😊 4 Challenge Time

```
// What will be the output of 3**3?
// What will be the output, when we add a number and a string?
// Write a program to swap two numbers?
// Write a program to swap two numbers without using third variable?
```

```
// sol 1: ✓
// console.log(9**2); // 9*9
// console.log(10 ** -1); 1/10
```

```
// sol 2: ✓
// console.log(5 + "thapa");
```

```
// sol 3: ✓
```

```
// var a = 5;
// var b = 10;
```

```
// output b=5; a=10
```

```
// var c = b; //c = 10
```

```
// b = a; // b = 5;
```

```
// a = c;
```

```
// console.log("the value of a is " + a);
```

```
// console.log("the value of b is " + b);
```

```
// sol 4: ✓
```

```
// var a = 5;
```

```
// var b = 10;
```

```
// // output b=5; a=10
```

```
// a = a + b; // a = 15
```

```
// b = a - b; // b = 5;
```

```
// a = a - b; // a = 10;
```

```
// console.log("the value of a is " + a);
```

```
// console.log("the value of b is " + b);
```

```
// 🧑🏻‍💻 Interview Question 4 🧑🏻‍💻
```

```
// What is the Difference between == vs === ?
```

```
// sol
// var num1 = 5;
// var num2 = '5';

// console.log(typeof(num1));
// console.log(typeof(num2));

// console.log(num1 == num2 );
```

```
// var num1 = 5;
// var num2 = '5';
```

```
// console.log(typeof(num1));
// console.log(typeof(num2));
// console.log(num2);
```

```
// console.log(num1 === num2 );
```

```
// *****
```

```
/**** Section 6  Control Statement -
```

```
*
```

```
*  If...Else */
```

```
// The if statement executes a statement if a specified condition is truthy.
```

```
// If the condition is falsy, another statement can be executed.
```

```
// if raining = raincoat
```

```
// else no raincoat
```

```
// var tomr = 'sunny';
```

```
// if(tomr == 'rain'){
```

```
//   console.log('take a raincoat');
```

```
// }else{
```

```
//   console.log('No need to take a raincoat');
```

```
// }
```

```
// 🧑Challenge Time
```

```
// write a program that works out whether if a given year is a leap year or not?
```

```
// A normal year has 365 days, leap years have 366, with an extra day in February.
```

```
// var year = 2020;
```

```
// debugger;
```

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

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

```
//     if(year % 400 === 0){
```

```
//       console.log("The year " + year + " is a leap year");
```

```
//     }else{
```

```
//       console.log("The year " + year + " is not a leap year");
```

```
//     }
```

```
//   }else{
```

```
//     console.log("The year " + year + " is a leap year");
```

```
//   }
```

```
// }else{  
// console.log("The year " + year + " is not a leap year");  
// }
```

// What is truthy and falsy values in Javascript?


// we have total 5 falsy values in javascript
// 0, "", undefined, null, NaN, false** is false anyway

```
// if(score = 5){  
// console.log("OMG, we loss the game 😞");  
// }else{  
// console.log("Yay, We won the game 😊");  
// }
```

// Conditional (ternary) operator

```
// The conditional (ternary) operator is the only JavaScript operator  
// that takes three operands  
// var age = 17;  
// if(age >= 18){  
// console.log("you are eligible to vote");  
// }else{  
// console.log("you are not eligible to vote");  
// }
```

```
// var age = 18;  
// console.log((age >= 18) ? "you can vote" : "you can't vote");
```

```
//  switch Statement  
// Evaluates an expression, matching the expression's value to a  
// case clause, and executes statements associated with that case.
```

```
// 1st without break statment  
// Find the Area of circle, triangle and rectangle?
```

```
// var area = "square" ;  
// var PI = 3.142, l=5, b=4, r=3;
```

```
// if(area == "circle"){  
//   console.log("the area of the circle is : " + PI*r**2);  
// }else if(area == "triangle"){  
//   console.log("the area of the triangle is : " + (l*b)/2);  
// }else if(area == "rectangle"){  
//   console.log("the area of the rectangle is : " + (l*b));  
// }else{  
//   console.log("please enter valid data");  
// }
```

```
// var area = "dsfsad" ;  
// var PI = 3.142, l=5, b=4, r=3;
```

```
// switch(area){
```



```
// case 'circle':  
//   console.log("the area of the circle is : " + PI*r**2);  
//   break;  
  
// case 'triangle':  
//   console.log("the area of the triangle is : " + (l*b)/2);  
//   break;  
  
// case 'rectangle':  
//   console.log("the area of the rectangle is : " + (l*b));  
//   break;  
  
// default:  
//   console.log("please enter valid data");  
// }
```

```
// ⚡break  
// Terminates the current loop, switch, or label  
// statement and transfers  
// program control to the statement following the terminated statement.
```

```
// ⚡continue  
// Terminates execution of the statements in the current iteration of the  
// current or labeled loop, and continues execution of the loop with the  
// next iteration.
```

// 4 ☐ While Loop Statement

// The while statement creates a loop that executes a specified statement

// as long as the test condition evaluates to true.

// var num=20;

// // block scope

// while(num <= 10){

// console.log(num); //infinte loop

// num++;

// }

// 5 ☐ Do-While Loop Statement

// var num = 20;

// do{

// debugger;

// console.log(num); //infinte loop

// num++;

// }while(num <= 10);

// 6 ☐ For Loop

// for(var num = 0; num <= 10; num++){

// debugger;

// console.log(num);

// }

// 😊6: challenge Time 🚩

// JavaScript program to print table for given number (8)?

// output : 8 * 1 = 8

// 8 * 2 = 16(8*2)

// => 8 * 10 = 80

// for(var num = 1; num<= 10; num++){

// var tableOf = 12;

// console.log(tableOf + " * " + num + " = " + tableOf * num);

// }

// *****

/** Section 5  Functions in JavaScript ****/**

// A JavaScript function is a block of code designed to perform a particular task.

//  Function Definition

// Before we use a function, we need to define it.

// A function definition (also called a function declaration, or function statement)

// consists of the function keyword, followed by:

// The name of the function.

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

// The JavaScript statements that define the function, enclosed in curly brackets, {...}.

```
// var a = 10;  
// var b = 20;  
// var sum = a+b;  
// console.log(sum);
```

```
// function sum(){  
//   var a = 10, b = 40;  
//   var total = a+b;  
//   console.log(total);  
// }  
// //
```

// 2  Calling functions

// Defining a function does not execute it.

// A JavaScript function is executed when "something" invokes it (calls it).

```
// function sum(){  
//   var a = 10, b = 40;  
//   var total = a+b;  
//   console.log(total);  
// }
```

```
// sum();
```

// 3  Function Parameter vs Function Arguments

// Function parameters are the names listed in the function's definition.

```
// Function arguments are the real values passed to the function.
```

```
// function sum(a,b){  
//   var total = a+b;  
//   console.log(total);  
// }
```

```
// sum();  
// sum(20,30);  
// sum(50,50);  
// sum(5,6)
```

```
// // *****
```

```
// // 📌 // 📌 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 📌
```

```
// 📌 // 📌 https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
```

```
// // *****
```

```
// 🧑🏻‍💻🧑🏻‍💻 Interview Question 🧑🏻‍💻🧑🏻‍💻
```

```
// Why Functions?
```

```
// You can reuse code: Define the code once, and use it many times.
```

```
// You can use the same code many times with different arguments,
```

```
// to produce different results.
```

// OR

**// A function is a group of reusable code which can be called anywhere
// in your program. This eliminates the need of writing the same code
// again and again.**

// DRY => do not repeat yourself

// 4☐ Function expressions

**// "Function expressions simply means
// create a function and put it into the variable "**

**// function sum(a,b){
// var total = a+b;
// console.log(total);
// }**

// var funExp = sum(5,15);

// 5☐ Return Keyword

**// When JavaScript reaches a return statement,
// the function will stop executing.**

```
// Functions often compute a return value.  
// The return value is "returned" back to the "caller"
```

```
// function sum(a,b){  
//   return total = a+b;  
// }
```

```
// var funExp = sum(5,25);
```

```
// console.log('the sum of two no is ' + funExp );
```

```
// 6  Anonymous Function
```

```
// A function expression is similar to and has the same syntax  
// as a function declaration One can define "named"  
// function expressions (where the name of the expression might  
// be used in the call stack for example)  
// or "anonymous" function expressions.
```

```
// var funExp = function(a,b){  
//   return total = a+b;  
// }
```

```
// var sum = funExp(15,15);  
// var sum1 = funExp(20,15);
```

```
// console.log(sum > sum1 );
```

```
// *****
```

```
// 🧐 Now It's Time for Modern JavaScript 😊😊
```

```
// 🧑🧑 Features of ECMAScript 2015 also known as ES6 🧑🧑
```

```
// 1️⃣ LET VS CONST vs VAR
```

```
// var myName = "thapa technical";
```

```
// console.log(myName);
```

```
// myName = "vinod thapa";
```

```
// console.log(myName);
```

```
// let myName = "thapa technical";
```

```
// console.log(myName);
```

```
// myName = "vinod thapa";
```

```
// console.log(myName);
```

```
// const myName = "thapa technical";
```

```
// console.log(myName);
```

```
// myName = "vinod thapa";
```

```
// console.log(myName);
```



```

// function biodata() {
//   const myFirstName = "Vinod";
//   console.log(myFirstName);

//   if(true){
//     const myLastName = "thapa";
//   }

//   // console.log('innerOuter ' + myLastName);
// }

// console.log(myFirstName);

// biodata();

// var => Function scope
// let and const => Block Scope

```

```

// 📄 Template literals (Template strings)

```

```

// JavaScript program to print table for given number (8)?

```

```

// output : 8 * 1 = 8
// 8 * 2 = 16(8*2)

```

```
// => 8 * 10 = 80
```

```
// for(let num = 1; num<= 10; num++){  
//   let tableOf = 12;  
//   // console.log(tableOf + " * " + num + " = " + tableOf * num);  
//   console.log(` ${tableOf} * ${num} = ${tableOf * num}` );  
// }
```

```
// 3  Default Parameters
```

```
// Default function parameters allow named parameters to be  
// initialized with default values if no value or undefined is passed.
```

```
// function mult(a,b=5){  
//   return a*b;  
// }
```

```
// console.log(mult(3));
```

```
// 4  Destructuring in ES6
```

```
// The destructuring assignment syntax is a JavaScript expression  
// that makes it possible to unpack values from arrays,
```

// or properties from objects, into distinct variables.

// → Array Destructuring 

```
// const myBioData = ['vinod', 'thapa', 26];
```

```
// let myFName = myBioData[0];
```

```
// let myLName = myBioData[1];
```

```
// let myAge = myBioData[2];
```

```
// let [myFName,myAge, myLName] = myBioData;
```

```
// console.log(myAge);
```

// we can add values too

```
// let [myFName,myLName,myAge, myDegree="MCS"] = myBioData;
```

```
// console.log(myDegree);
```

// → Object destructuring 

```
// const myBioData = {
```

```
//   myFname : 'vinod',
```

```
//   myLname : 'thapa',
```

```
//   myAge : 26
```

```
// }
```

```
// let age = myBioData.age;
```

```
// let myFname = myBioData.myFname;
```

```
// let {myFname,myLname,myAge, myDegree="MCS"} = myBioData;
```

```
// console.log(myLname);
```

```
// 5  Object Properties
```

```
// → we can now use Dynamic Properties
```

```
// let myName = "vinod";  
// const myBio = {  
//   [myName] : "hello how are you?",  
//   [20 + 6] : "is my age"  
// }
```

```
// console.log(myBio);
```

```
// → no need to write key and value, if both are same
```

```
// let myName = "vinod thapa";  
// let myAge = 26;
```

```
// const myBio = {myName,myAge}
```

```
// console.log(myBio);
```

// 6  Fat Arrow Function

//  Normal Way of writing Function

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

```
// function sum() {  
//   let a = 5; b = 6;  
//   let sum = a+b;  
//   return `the sum of the two number is ${sum}`;  
// }
```

//  How to convert in into Fat Arrow Function

```
// const sum = () => `the sum of the two number is ${a=5)+(b=6)}`;
```

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

// 7  Spread Operator

```
// const colors = ['red', 'green', 'blue', 'white', 'pink'];
```

```
// const myColors = ['red', 'green', 'blue', 'white', 'pink', 'yellow', 'black'];
```

```
// // // 2nd time add one more color on top and tell we need to write it again
```

```
// // // on myColor array too
```

```
// const MyFavColors = [ ...colors, 'yellow', 'black'];
```

```
// console.log(MyFavColors);
```

```
// ES7 features
```

```
// 1: array include
```

```
// const colors = ['red', 'green', 'blue', 'white', 'pink'];
```

```
// const isPresent = colors.includes('purple');
```

```
// console.log(isPresent);
```

```
// 2: **
```

```
// console.log(2**3);
```

```
// ES8 Features
```

```
// String padding
```

```
// Object.values()
```

```
// Object.entries()
```

```
// const message = "my name is vinod";
```

```
// console.log(message);
```

```
// console.log(message.padStart(5));
```

```
// console.log(message.padEnd(10));
```

```
// const person = { name: 'Fred', age: 87 };
```

```
// // // console.log( Object.values(person) );
```

```
// const arrObj = Object.entries(person);
```

```
// console.log(Object.fromEntries(arrObj));
```

```
// ES2018
```

```
// const person = { name: 'Fred', age: 87, degree : "mcs" };
```

```
// const sPerson = { ...person };
```

```
// console.log(person);
```

```
// console.log(sPerson);
```

```
// ES2019
```

```
// Array.prototype.{flat,flatMap}
```

```
// Object.fromEntries()
```

```
// ES2020
```

```
// #1: BigInt
```

```
// let oldNum = Number.MAX_SAFE_INTEGER;
```

```
// // console.log(oldNum);
```

```
// // console.log( 9007199254740991n + 12n );
```

```
// const newNum = 9007199254740991n + 12n;
```

```
// console.log(newNum);
```

```
// console.log(typeof newNum);
```

```
// const foo = null ?? 'default string';
```

```
// console.log(foo);
```

```
// ES2014
```

```
// "use strict";
```

```
// x = 3.14;
```

```
// console.log(x);
```

```
// *****
```

```
/*** Section 7  Arrays in JavaScript ***/
```

```
// When we use var, we can stored only one value at a time.
```

```
// var friend1 = 'ramesh';
```

```
// var friend2 = 'arjun';
```

```
// var friend3 = 'vishal';
```

```
// var myFriends = ['ramesh',22,male,'arjun',20,male,'vishal',true, 52];
```

```
// When we feel like storing multiple values in one variable then
```


// instead of var, we will use an Array.

// In JavaScript, we have an Array class, and

// arrays are the prototype of this class.

// example

// var myFriends = ['ramesh',22,male,'arjun',20,male,'vishal',true, 52];

// 1. Array Subsection 1 → Traversal in array

// navigate through an array

// if we want to get the single data at a time and also

// if we want to change the data

// var myFriends = ['vinod','ramesh','arjun','vishal'];

// console.log(myFriends[myFriends.length - 1]);

// if we want to check the length of elements of an array

// console.log(myFriends.length);

// we use for loop to navigate



```
// var myFriends = ['vinod','ramesh','arjun','vishal'];
// for(var i=0; i<myFriends.length; i++){
//   console.log(myFriends[i]);
// }
```

// After ES6 we have for..in and for..of loop too

```
// var myFriends = ['vinod','ramesh','arjun','vishal'];

// for(let elements in myFriends){
//   console.log(elements);
// }
```

```
// for(let elements of myFriends){
//   console.log(elements);
// }
```

```
// Array.prototype.forEach()  
// Calls a function for each element in the array.
```

```
// var myFriends = ['vinod','ramesh','arjun','vishal'];

// myFriends.forEach(function(element, index, array) {
//   console.log(element + " index : " +
//     index + " " + array);
// });

// myFriends.forEach((element, index, array) => {
//   console.log(element + " index : " +
//     index + " " + array);
// });
```

// 📁 Array Subsection 2 🖱️ Searching and Filter in an Array

// Array.prototype.indexOf() 🧐♂️📁

// Returns the first (least) index of an element within the array equal
// to an element, or -1 if none is found. It search the element from the
// 0th index number

// var myFriendNames = ["vinod","bahadur","thapa","thapatechnical","thapa"];

// console.log(myFriendNames.indexOf("Thapa", 3));

// Array.prototype.lastIndexOf() 🧐♂️📁

// Returns the last (greatest) index of an element within the array equal
// to an element, or -1 if none is found. It search the element last to first

// var myFriendNames = ["vinod","bahadur","thapa","thapatechnical","thapa"];

// console.log(myFriendNames.lastIndexOf("Thapa",3));

```
// Array.prototype.includes() 🕵️♂️❏  
// Determines whether the array contains a value,  
// returning true or false as appropriate.  
  
// var myFriendNames = ["vinod","bahadur","thapa","thapatechnical"];  
  
// console.log(myFriendNames.includes("thapa"));
```

```
// Array.prototype.find() 🕵️♂️❏  
  
// arr.find(callback(element[, index[, array]]), thisArg)  
  
// Returns the found element in the array, if some element in the  
// array satisfies the testing function, or undefined if not found.  
// Only problem is that it return only one element  
  
// const prices = [200,300,350,400,450,500,600];  
  
// price < 400  
// const findElem = prices.find((currVal) => currVal < 400 );  
// console.log(findElem);
```

```
// console.log( prices.find((currVal) => currVal > 1400 ) );
```

```
// // *****
```

```
// // 📄 // 📄 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 📄
```

```
// 📄 // 📄 https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
```

```
// // *****
```

```
// Array.prototype.findIndex() 🧑🏻📄
```

```
// Returns the found index in the array, if an element in the  
// array satisfies the testing function, or -1 if not found.
```

```
// console.log( prices.findIndex((currVal) => currVal > 1400 ) );
```

```
// Array.prototype.filter() 🧑🏻📄
```

```
// Returns a new array containing all elements of the calling  
// array for which the provided filtering function returns true.
```

```
// const prices = [200,300,350,400,450,500,600];

// // price < 400
// const newPriceTag = prices.filter((elem, index) => {
//   return elem > 1400;
// })
// console.log(newPriceTag);
```

```
// 📖 Array Subsection 3 📖 How to sort an Array
```

```
// Array.prototype.sort() 🧑🏻📖
```

```
// The sort() method sorts the elements of an array in place and
//returns the sorted array. The default sort order is ascending, built
//upon converting the elements into strings,
// then comparing their sequences of UTF-16 code units values.
```

```
// const months = ['March', 'Jan', 'Feb', 'April', 'Dec', 'Nov'];
```

```
// console.log(months.sort());
```

```
// const array1 = [1, 30, 4, 21, 100000, 99];
```

```
// console.log(array1.sort());
```

```
// However, if numbers are sorted as strings,  
// "25" is bigger than "100", because "2" is bigger than "1".
```

```
// Because of this, the sort() method will produce an incorrect  
// result when sorting numbers.
```

```
// 🤖7: challenge Time 🚩
```

```
// 1: How to Sort the numbers in the array in ascending (up) and descending (down) order?
```

```
// compareFunction Optional.
```

```
// A function that defines an alternative sort order. The function should return a negative, zero, or  
positive value, depending on the arguments, like:
```

```
// function(a, b){return a-b}
```

```
// for ascending order
```

```
// array1.sort(function(a,b){
```

```
//   console.log(a,b);
```

```
//   if(a>b){
```

```
//     return 1;
```

```
//     // b comes first and then a
```

```
//   }
```

```
//   if(a<b){
```

```
//    // a comes first and then b
//    return -1;
// }
//  if(a==b){
//    // No changes
//    return 0;
//  }
// };
```

```
// for descending order
// array1.sort(function(a,b){
//  console.log(a,b);
//  if(a>b){
//    return -1;
//    // b comes first and then a
//  }
//  if(a<b){
//    // a comes first and then b
//    return 1;
//  }
//  if(a==b){
//    // No changes
//    return 0;
//  }
// });
```

```
// console.log(array1);
```



```
// 2: sort the array in descending order
```

```
// var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
// let aFruits = fruits.sort();
```

```
// //Array.prototype.reverse() 🧑🏻♂️📄
```

```
// // The reverse() method reverses an array in place.
```

```
// // The first array element becomes the last, and
```

```
// // the last array element becomes the first.
```

```
// 4📄 Array Subsection 4 🧑🏻♂️ Perform CRUD
```

```
// Array.prototype.push() 🧑🏻♂️📄
```

```
// The push() method adds one or more elements to the
```

```
// end of an array and returns the new length of the array.
```

```
// const animals = ['pigs', 'goats', 'sheep'];
```

```
// // const count = animals.push('chicken');
```

```
// // console.log(count);
```

```
// animals.push('chicken', 'cats', 'cow');
```

```
// console.log(animals);
```

```
// Array.prototype.unshift() 🤖👤📄
```

```
// The unshift() method adds one or more elements to the
```

```
// beginning of an array and returns the new length of the array.
```

```
// const animals = ['pigs', 'goats', 'sheep'];
```

```
// const count = animals.unshift('chicken');
```

```
// console.log(count);
```

```
// console.log(animals);
```

```
// animals.unshift('chicken', 'cats', 'cow');
```

```
// console.log(animals);
```

```
// 2nd example
```

```
// const myNumbers = [1,2,3,5];
```

```
// myNumbers.unshift(4,6);
```

```
// console.log(myNumbers);
```

```
// Array.prototype.pop() 🤖👤📄
```

```
// The pop() method removes the last element from an array and returns
```

```
// that element. This method changes the length of the array.
```

```
// const plants = ['broccoli', 'cauliflower', 'kale', 'tomato', 'cabbage'];
```

```
// console.log(plants);  
// console.log(plants.pop());  
// console.log(plants);
```

```
// Array.prototype.shift() 🤖  
// The shift() method removes the first element from an array and returns  
// that removed element. This method changes the length of the array.
```

```
// const plants = ['broccoli', 'cauliflower', 'kale', 'tomato', 'cabbage'];  
// console.log(plants);  
// console.log(plants.shift());  
// console.log(plants);
```

```
// 🤖8: challenge Time 🚩
```

```
// Array.prototype.splice() 🤖  
// Adds and/or removes elements from an array.
```

```
// 1: Add Dec at the end of an array?  
// 2: What is the return value of splice method?  
// 3: update march to March (update)?  
// 4: Delete June from an array?
```

```
// sol1:  
// const newMonth = months.splice(months.length,0,"Dec");
```

```
// console.log(months);

// sol2:
// console.log(newMonth);

// sol3:
// const months = ['Jan', 'march', 'April', 'June', 'July'];

// const indexOfMonth = months.indexOf('June');

// if(indexOfMonth !== -1){
//   const updateMonth = months.splice(indexOfMonth,1,'june');
//   console.log(months);
// }else{
//   console.log('No such data found');
// }

// sol3:
// const months = ['Jan', 'march', 'April', 'June', 'July'];

// const indexOfMonth = months.indexOf('April');

// if(indexOfMonth !== -1){
//   const updateMonth = months.splice(indexOfMonth,2);
//   console.log(months);
//   console.log(updateMonth);
// }else{
//   console.log('No such data found');
// }
```

// 📄 Array Subsection 4 📖 Map and Reduce Method

// Array.prototype.map() 🧑🏻🔗

```
// let newArray = arr.map(callback(currentValue[, index[, array]]) {  
//   // return element for newArray, after executing something  
// }, thisArg]);
```

// Returns a new array containing the results of calling a
// function on every element in this array.

```
// const array1 = [1, 4, 9, 16, 25];  
// num > 9  
// let newArr = array1.map((curElem, index, arr) => {  
//   return curElem > 9;  
// })  
// console.log(array1);  
// console.log(newArr);
```

```
// let newArr = array1.map((curElem, index, arr) => {  
//   return `Index no = ${index} and the value is ${curElem} belong to ${arr}`  
// }).reduce().  
// console.log(newArr);
```

```
// let newArrfor = array1.forEach((curElem, index, arr) => {  
//   return `Index no = ${index} and the value is ${curElem} belong to ${arr}`
```

```
// }
```

```
// console.log(newArrfor);
```

```
// It return new array without mutating the original array
```

```
// // *****
```

```
// // 📌 // 📌 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 📌
```

```
// 📌 // 📌 https://www.youtube.com/channel/UCwfaAHy4zQub2APNOGXUCCA
```

```
// // *****
```

```
// 😊9: challenge Time 🚩
```

```
// 1: Find the square root of each element in an array?
```

```
// 2: Multiply each element by 2 and return only those
```

```
// elements which are greater than 10?
```

```
// sol1:
```

```
// let arr = [25, 36, 49, 64, 81];
```

```
// let arrSqr = arr.map((curElem) => Math.sqrt(curElem) )
```

```
// console.log(arrSqr);
```

```
// sol 2:
```

```
// let arr = [2, 3, 4, 6, 8];
```

```
// let arr2 = arr.map((curElem) => curElem * 2).filter((curElem) => curElem > 10)
//   ).reduce((accumulator, curElem) => {
//     return accumulator += curElem;
//   });
// console.log(arr2);

// we can use the chaining too
```

// 📖 Reduce Method

```
// flatten an array means to convert the 3d or 2d array into a
// single dimensional array
```

```
// The reduce() method executes a reducer function (that you provide)
// on each element of the array, resulting in single output value.
```

```
// The reducer function takes four arguments:
```

```
// Accumulator
```

```
// Current Value
```

```
// Current Index
```

```
// Source Array
```

```
// 4 subj = 1sub= 7
```

```
// 3dubj = [5,6,2]
```

```
// let arr = [5,6,2];
```

```
// let sum = arr.reduce((accumulator, curElem) => {  
//     debugger;  
//     return accumulator += curElem;  
// },7)  
// console.log(sum);
```

```
// How to fatten an array
```

```
// converting 2d and 3d array into one dimensional array
```

```
// const arr = [  
//     ['zone_1', 'zone_2'],  
//     ['zone_3', 'zone_4'],  
//     ['zone_5', 'zone_6'],  
//     ['zone_7', ['zone_7', ['zone_7', 'zone_8']]]  
// ];
```

```
// // let flatArr = arr.reduce((accum, currVal) => {  
// //     return accum.concat(currVal);  
// // })
```

```
// console.log(arr.flat(Infinity));
```



```
// console.log(flatArr);
```

```
// const arr = [ ['zone_1', 'zone_2'], ['zone_3', ['zone_1', 'zone_2'], ['zone_1', 'zone_2']] ];
```

```
// console.log(arr.flat(3));
```

```
// console.log(arr);
```

```
/* Section 7  Strings in JavaScript */
```

```
// A JavaScript string is zero or more characters written inside quotes.
```

```
// JavaScript strings are used for storing and manipulating text.
```

```
// You can use single or double quotes
```

```
// Strings can be created as primitives,
```

```
// from string literals, or as objects, using the String() constructor
```

```
// let myName = "vinod thapa";
```

```
// let myChannelName = 'vinod thapa';
```

```
// // let ytName = new String("Thapa Technical");
```

```
// let ytName = 'thapa technical';
```

```
// console.log(myName);
```

```
// console.log((ytName));
```

// ☞ How to find the length of a string

// String.prototype.length 🧐☞

// Reflects the length of the string.

// let myName = "vinod thapa";

// console.log(myName.length);

// ☞ Escape Character

// let anySentence = "We are the so-called \"Vikings\" from the north.";

// console.log(anySentence);

// // if you dont want to mess, simply use the alternate quotes

// let anySentence = " We are the so-called 'Vikings' from the north. ";

// console.log(anySentence);

// ☞ Finding a String in a String

```
// String.prototype.indexOf(searchValue [, fromIndex]) 🧐♂️
```

```
// The indexOf() method returns the index of (the position of) the first  
// occurrence of a specified text in a string
```

```
// const myBioData = 'I am the thapa Technical';  
// console.log(myBioData.indexOf("t", 6));
```

```
// // JavaScript counts positions from zero.  
// // 0 is the first position in a string, 1 is the second, 2 is the third ...
```

```
// // String.prototype.lastIndexOf(searchValue [, fromIndex]) 🧐♂️
```

```
// // Returns the index within the calling String object of the  
// // last occurrence of searchValue, or -1 if not found.
```

```
// const myBioData = 'I am the thapa Technical';  
// console.log(myBioData.lastIndexOf("t", 6));
```

// ☞ Searching for a String in a String

// String.prototype.search(regex) 🔍

**// The search() method searches a string for a specified
// value and returns the position of the match**

**// const myBioData = 'I am the thapa Technical';
// let sData = myBioData.search("technical");
// console.log(sData);**

// The search() method cannot take a second start position argument.

// ☞ Extracting String Parts

// There are 3 methods for extracting a part of a string:

**// slice(start, end)
// substring(start, end)**

// substr(start, length)

// The slice() Method 🤖👤

**// slice() extracts a part of a string and returns the extracted part
// in a new string.**

**// The method takes 2 parameters: the start position,
// and the end position (end not included).**

// var str = "Apple, Bananaa, Kiwi, mango";

**// // let res = str.slice(0,4);
// let res = str.slice(7);
// console.log(res);**

**// The slice() method selects the elements starting at the
// given start argument, and ends at, but does not include,
// the given end argument.**

**// Note: The original array will not be changed.
// Remember: JavaScript counts positions from zero. First position is 0.**

// 😊11: challenge Time 🚩

// Display only 280 characters of a string like the

// one used in Twitter?

// let myTweets = "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Why do we use it? ";

// let myActualTweet = myTweets.slice(0,280);

// console.log(myActualTweet);

// console.log(myActualTweet.length);

// The substring() Method 🤖♂️

// substring() is similar to slice().

// The difference is that substring() cannot accept

// negative indexes.

// var str = "Apple, Bananaa, Kiwi";

// let res = str.substring(8,-2);

// console.log(res);

// // If we give negative value then the characters are

// counted from the 0th pos

// The substr() Method 🧐☑

// substr() is similar to slice().

// The difference is that the second parameter specifies the

// length of the extracted part.

// var str = "Apple, Bananaa, Kiwi";

// // let res = str.substr(7,-2);

// let res = str.substr(-4);

// console.log(res);

// 📄 Replacing String Content()

// String.prototype.replace(searchFor, replaceWith) 🧐☑

// The replace() method replaces a specified value

// with another value in a string.

// let myBioData = `I am vinod bahadur thapa vinod`;

// let repalceData = myBioData.replace('Vinod','VINOD');

// console.log(repalceData);

// console.log(myBioData);

```
// Points to remember
// 1: The replace() method does not change the string
// it is called on. It returns a new string.
// 2: By default, the replace() method replaces only
// the first match
// 3: By default, the replace() method is case sensitive.
// Writing VINOD (with upper-case) will not work
```

```
//👉 Extracting String Characters
```

```
// There are 3 methods for extracting string characters:
```

```
// charAt(position)
// charCodeAt(position)
// Property access [ ]
```

```
// The charAt() Method 🤖♂️
// The charAt() method returns the character at a
// specified index (position) in a string

// let str = "HELLO WORLD";
```



```
// console.log(str.charAt(9));
```

```
// The charCodeAt() Method 🧐♂️
```

```
// The charCodeAt() method returns the unicode of the  
// character at a specified index in a string:
```

```
// The method returns a UTF-16 code  
// (an integer between 0 and 65535).
```

```
// The Unicode Standard provides a unique number for every  
// character, no matter the platform, device, application,  
// or language. UTF-8 is a popular Unicode encoding which  
// has 8-bit code units.
```

```
// var str = "HELLO WORLD";
```

```
// console.log( str.charCodeAt(0) );
```

// 😊12: challenge Time 🚩

// Return the Unicode of the last character in a string

// let str = "HELLO WORLD";

// let lastChar = str.length - 1;

// console.log(str.charCodeAt(lastChar));

// Property Access

// ECMAScript 5 (2009) allows property access [] on strings

// var str = "HELLO WORLD";

// console.log(str[1]);

//👉 Other useful methods

```
// let myName = "vinod tHapa";  
// console.log(myName.toUpperCase());  
// console.log(myName.toLowerCase());
```

```
// The concat() Method 🤖  
// concat() joins two or more strings
```

```
// let fName = "vinod"  
// let lName = "thapa"  
  
// console.log(fName + lName );  
// console.log(`${fName} ${lName}`);  
// console.log(fName.concat(lName));  
// console.log(fName.concat(" ",lName));
```

```
// String.trim() 🤖  
// The trim() method removes whitespace from both  
// sides of a string
```

```
// var str = "    Hello    World!    ";  
// console.log(str.trim());
```

```
// Converting a String to an Array  
// A string can be converted to an array with the  
// split() method
```

```
// var txt = "a, b,c d,e"; // String  
// console.log(txt.split(", ")); // Split on commas  
// console.log( txt.split(" ")); // Split on spaces  
// console.log(txt.split("|")); // Split on pipe
```

```
/**** Section 8📄 Date and Time in JavaScript *****/
```

```
// JavaScript Date objects represent a single moment in time in a  
// platform-independent format. Date objects contain a Number  
// that represents milliseconds since 1 January 1970 UTC.
```

```
// 📄 Creating Date Objects
```

// There are 4 ways to create a new date object:

// new Date()

// new Date(year, month, day, hours, minutes, seconds, milliseconds)

// // it takes 7 arguments

// new Date(milliseconds)

// // we cannot avoid month section

// new Date(date string)

// new Date()  

// Date objects are created with the new Date() constructor.

// let currDate = new Date();

// console.log(currDate);

// console.log(new Date());

// console.log(new Date().toLocaleString()); // 9/11/2019, 1:25:01 PM

// console.log(new Date().toString()); // Wed Sep 11 2019 13:25:01 GMT+0700 (GMT+07:00)

// Date.now()  

// Returns the numeric value corresponding to the current time—the number

// of milliseconds elapsed since January 1, 1970 00:00:00 UTC

// console.log(Date.now());

// new Date(year, month, ...)  

// 7 numbers specify year, month, day, hour, minute, second,

// and millisecond (in that order)

// Note: JavaScript counts months from 0 to 11.

// January is 0. December is 11.

// var d = new Date(2021,0);

// console.log(d.toLocaleString());

// new Date(dateString) 🤖👤

// new Date(dateString) creates a new date object from a date string

// var d = new Date("October 13, 2021 11:13:00");

// console.log(d.toLocaleString());

// new Date(milliseconds) 🤖👤

// new Date(milliseconds) creates a new date object as zero time plus milliseconds:

// var d = new Date(0);

// var d = new Date(1609574531435);

// var d = new Date(86400000*2);

// console.log(d.toLocaleString());

//📅 Dates Method

// const curDate = new Date();

// // how to get the individual date

// console.log(curDate.toLocaleString());

// console.log(curDate.getFullYear());

```
// console.log(curDate.getMonth()); // 0-11 jan to dec
// console.log(curDate.getDate());
// console.log(curDate.getDay());

// // how to set the individual date

// console.log(curDate.setFullYear(2022));
// // The setFullYear() method can optionally set month and day
// console.log(curDate.setFullYear(2022, 10, 5));
// let setmonth = curDate.setMonth(10); // 0-11 jan to dec
// console.log(setmonth);
// console.log(curDate.setDate(5));
// console.log(curDate.toLocaleString());
```

//📅 Time Methods

```
// const curTime = new Date();

// how to get the individual Time

// console.log(curTime.getTime());
// // // The getTime() method returns the number of milliseconds
// // since January 1, 1970
// console.log(curTime.getHours());
// // // The getHours() method returns the hours of a date as a
// // number (0-23)
// console.log(curTime.getMinutes());
```

```
// console.log(curTime.getSeconds());  
// console.log(curTime.getMilliseconds());
```

```
// // how to set the individual Time
```

```
// let curTime = new Date();
```

```
// // console.log(curTime.setTime());  
// console.log(curTime.setHours(5));  
// console.log(curTime.setMinutes(5));  
// console.log(curTime.setSeconds(5));  
// console.log(curTime.setMilliseconds(5));
```

```
// // Practice Time
```

```
// new Date().toLocaleTimeString(); // 11:18:48 AM
```

```
// //---
```


```
// new Date().toLocaleDateString(); // 11/16/2015
```

```
// //---
```

```
// new Date().toLocaleString(); // 11/16/2015, 11:18:48 PM
```



```
// Challenge Time NOT yet decided
// (function(){
//   setInterval(()=> {
//     console.log(new Date().toLocaleTimeString());
//   }, 1000)
// })();
```

**** Section 9  Math Object in JavaScript ****/

// The JavaScript Math object allows you to perform mathematical tasks on numbers.

```
// console.log(Math.PI);  
```

```
// console.log(Math.PI);
```

```
// Math.round()  
```

// returns the value of x rounded to its nearest integer

```
// let num = 10.501;
```

```
// console.log(Math.round(num));
```

```
// Math.pow()  
```

// Math.pow(x, y) returns the value of x to the power of y

```
// console.log(Math.pow(2,3));
```

```
// console.log(2**3);
```

```
// Math.sqrt() 🐞☐
```

```
// Math.sqrt(x) returns the square root of x
```

```
// console.log(Math.sqrt(25));
```

```
// console.log(Math.sqrt(81));
```

```
// console.log(Math.sqrt(66));
```

```
// Math.abs() 🐞☐
```

```
// Math.abs(x) returns the absolute (positive) value of x
```

```
// console.log(Math.abs(-55));
```

```
// console.log(Math.abs(-55.5));
```

```
// console.log(Math.abs(-955));
```

```
// console.log(Math.abs(4-6));
```

```
// Math.ceil() 🐞☐
```

```
// Math.ceil(x) returns the value of x rounded up to its nearest integer
```

```
// console.log(Math.ceil(4.51));
```

```
// console.log(Math.round(4.51));
```

```
// console.log(Math.ceil(99.01));
```

```
// console.log(Math.round(99.1));
```

// Math.floor() 🧐☑

// Math.floor(x) returns the value of x rounded down to its nearest integer

// console.log(Math.floor(4.7));

// console.log(Math.floor(99.1));

// Math.min() 🧐☑

// Math.min() can be used to find the lowest value in a list of arguments

// console.log(Math.min(0, 150, 30, 20, -8, -200));

// Math.max() 🧐☑

// Math.max() can be used to find the highest value in a list of arguments

// console.log(Math.max(0, 150, 30, 20, -8, -200));

// Math.random() 🧐☑

// Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

// console.log(Math.floor(Math.random()*10));

// console.log(Math.floor(Math.random()*10)); // 0 to 9

```
// Math.round() 🧐
```

```
// The Math.round() function returns the value of a number  
// rounded to the nearest integer.
```

```
// console.log(Math.round(4.6));  
// console.log(Math.round(99.1));
```

```
// Math.trunc() 🧐
```

```
// The trunc() method returns the integer part of a number
```

```
// console.log(Math.trunc(4.6));  
// console.log(Math.trunc(-99.1));
```

```
// Practice Time
```

```
// if the argument is a positive number, Math.trunc() is equivalent to  
// Math.floor(),  
// otherwise Math.trunc() is equivalent to Math.ceil().
```

// Section 10☞ Document Object model in JavaScript

**// 1☐ Window is the main container or we can say the
// global Object and any operations related to entire
// browser window can be a part of window object.**

// For ex ☞ the history or to find the url etc.

// 1☐ whereas the DOM is the child of Window Object

// // *****

// // ☞ // ☐ SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL ☐

// ☞ // ☐ <https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA>

// // *****

// 2☐ All the members like objects, methods or properties.

// If they are the part of window object then we do not refer

// the window object. Since window is the global object

// so you do not have to write down window.

// - it will be figured out by the runtime.

```
// For example
// 📺 window.screen or just screen is a small information
// object about physical screen dimensions.
// 📺 window.location giving the current URL
// 📺 window.document or just document is the main object
// of the potentially visible (or better yet: rendered)
// document object model/DOM.
```

```
// 2📄 Where in the DOM we need to refer the document,
// if we want to use the document object, methods or properties
// For example
// 📺 document.getElementById()
```

```
// 3📄 Window has methods, properties and object.
// ex setTimeout() or setInterval() are the methods
// where as Document is the object of the Window and
// It also has a screen object with properties
// describing the physical display.
```

```
// Now, I know you have a doubt like we have seen the methods
// and object of the global object that is window. But What about
// the properties of the Window Object ?
```

```
// so example of window object properties are
// innerHeight,
// innerWidth and there are many more
```

// let's see some practical in DOM HTML file

// ***** DOM vs BOM *****

// ☞ The DOM is the Document Object Model, which deals with the document,
// the HTML elements themselves, e.g. document and all traversal you
// would do in it, events, etc.

// For Ex: 🧑🏻👤
// change the background color to red
// document.body.style.background = "red";

// ☞ The BOM is the Browser Object Model, which deals with browser components
// aside from the document, like history, location, navigator and screen
// (as well as some others that vary by browser). OR
// In simple meaning all the Window operations which comes under BOM are performed
// usign BOM

// Let's see more practical on History object

// Functions alert/confirm/prompt are also a part of BOM:
// they are directly not related to the document,
// but represent pure browser methods of communicating with the user.

```
// alert(location.href); // shows current URL

// if (confirm("Want to Visit ThapaTechnical?")) {

//   location.href = "https://www.youtube.com/thapatechnical"; // redirect the browser to
//   another URL

// }
```

// Section 3: Navigate through the DOM

```
// 1: document.documentElement
    // returns the Element that is the root element of the document.

// 2: document.head

// 3: document.body

// 4: document.body.childNodes (include tab,enter and whiteSpace)
    // list of the direct children only

// 5: document.children (without text nodes, only regular Elements)

// 6: document.childNodes.length
```

```
// Practice Time

// How to check whether an element has child nodes or not?

// we will use hasChildNodes()
```

```
// Practice Time

// How to find the child in DOM tree

// firstChild vs firstElementChild

// lastChild vs lastElementChild

// const data = document.body.firstElementChild;

// undefined
```



```
// data
// data.firstChild
// data.firstChild.firstChild
// data.firstChild.firstChild.style.color = "red"
// vs
// document.querySelector(".child-two").style.color = "yellow";
```

```
// ☞ How to find the Parent Nodes
```

```
// document.body.parentNode
```

```
// document.body.parentElement
```

```
// ☞ How to find or access the siblings
```

```
// document.body.nextSibling
```

```
// document.body.nextElementSibling
```

```
// document.body.previousSibling
```

```
// document.body.previousElementSibling
```

```
//SECTION 4☐: How to search the Elements and the References
```

```
// We will see the new file
```

```
// *****
```

```
/*** Section 11☞ EVENTS in JavaScript ***/
```

// HTML events are "things" that happen to HTML elements.
// When JavaScript is used in HTML pages, JavaScript can "react" on these events.

// 🧑🏻 HTML Events

// An HTML event can be something the browser does, or something a user does.

// Here are some examples of HTML events:

// An HTML web page has finished loading

// An HTML input field was changed

// An HTML button was clicked

// Often, when events happen, you may want to do something.

// JavaScript lets you execute code when events are detected.

// HTML allows event handler attributes, with JavaScript code,

// to be added to HTML elements.

// section 1 📄 4 ways of writing Events in JavaScript

// 1: using inline events alert();

// 2: By Calling a function (We already seen and most common way of writing)

// 3: using Inline events (HTML onclick="" property and element.onclick)

// 4: using Event Listeners (addEventListener and IE's attachEvent)

// check the Events HTML File

// section 2☐: What is Event Object?
// Event object is the parent object of the event object.
// for Example
// MouseEvent, focusEvent, KeyboardEvent etc

// section ☐3☐ MouseEvent in JavaScript
// The MouseEvent Object
// Events that occur when the mouse interacts with the HTML
// document belongs to the MouseEvent Object.

// section ☐4☐ KeyboardEvent in JavaScript
// Events that occur when user presses a key on the keyboard,
// belongs to the KeyboardEvent Object.
// https://www.w3schools.com/jsref/obj_keyboardevent.asp

// Section 5☐ InputEvents in JavaScript
// The onchange event occurs when the value of an element has been changed.

// For radiobuttons and checkboxes, the onchange event occurs when the
// checked state has been changed.

```
// *****
```

```
// ☞ JavaScript Timing Events
```

```
// *****
```

```
// The window object allows execution of code at specified time intervals.
```

```
// These time intervals are called timing events.
```

```
// The two key methods to use with JavaScript are:
```

```
// setTimeout(function, milliseconds)
```

```
// Executes a function, after waiting a specified number of milliseconds.
```

```
// setInterval(function, milliseconds)
```

```
// Same as setTimeout(), but repeats the execution of the function continuously.
```

```
// 1☐ setTimeout()
```

```
// 2☐ clearTimeout()
```

```
// 3☐ setInterval()
```

```
// 4☐ clearInterval()
```

```
// *****
```

```
// 📖 object oriented Javascript
```

```
// *****
```

```
// 1️⃣ What is Object Literal?
```

```
// Object literal is simply a key:value pair data structure.
```

```
// Storing variables and functions together in one container,
```

```
// we can refer this as an Objects.
```

```
// object = school bag
```

```
// How to create an Object?
```

```
// 1st way
```

```
    // let bioData = {
```

```
    //   myName : "thapatechnical",
```

```
    //   myAge : 26,
```

```
    //   getData : function(){
```

```
    //       console.log(`My name is ${bioData.myName} and my age is ${bioData.myAge}`);
```

```
    //   }
```

```
    // }
```


```
    // bioData.getData();
```

```
// 2nd way no need to write functions as well after es6
```

```
// let bioData = {  
  // myName : "thapatechnical",  
  // myAge : 26,  
  // getData () {  
    // console.log(`My name is ${bioData.myName} and my age is ${bioData.myAge}`);  
  // }  
// }  
  
// bioData.getData();
```

// ☞ What if we want object as a value inside an Object

```
// let bioData = {  
  // myName : {  
    // realName : "vinod",  
    // channelName : "thapa technical"  
  // },  
  // myAge : 26,  
  // getData () {  
    // console.log(`My name is ${bioData.myName} and my age is ${bioData.myAge}`);  
  // }  
// }  
  
// console.log(bioData.myName.channelName );
```

// 2  What is this Object?

// The definition of "this" object is that it contain the current context.

// The this object can have different values depending on where it is placed.

// // For Example 1

// console.log(this.alert('Awesome'));

// // it refers to the current context and that is window global object

// // ex 2

// function myName() {

// console.log(this);

// }

// myName();

// // ex 3

// var myNames = 'vinod';

// function myName() {

// console.log(this.myNames);

// }

// myName();

// // ex 4

```
// const obj = {  
  //   myAge : 26,  
  //   myName() {  
    //     console.log(this.myAge);  
  //   }  
// }  
// }  
// obj.myName();
```

// // ex 5

// // this object will not work with arrow function bcz arrow function is bound to class.

```
// const obj = {  
  //   myAge : 26,  
  //   myName : () => {  
    //     console.log(this);  
  //   }  
// }  
// }  
// obj.myName();
```

// // ex 6

```
// let bioData = {  
  //   myName : {  
    //     realName : "vinod thapa",  
    //     channelName : 'thapa technical'  
  //   },  
  //   // things to remember is that the myName is the key and the object is act like a value  
  //   myAge : 26,  
  //   getData () {
```



```
// console.log(`My name is ${this.myName.channelName} and my age is ${this.myAge} `);  
// }  
// }
```

```
// bioData.getData();
```

```
// // call method is used to call the method of another object
```

```
// // or with call(), an object can use a method belonging to another object
```

```
// // But as per other it is simply the way to use the this keyword or another object
```

```
// // *****
```

```
// // 📄 How JavaScript Works? Advanced and Asynchronous JavaScript
```

```
// // *****
```

```
// // Advanced JavaScript Section
```

```
// // 📄: Event Propagation (Event Bubbling and Event Capturing)
```

```
// // check html file
```

// // 2: Higher Order Function

// // function which takes another function as an arguments is called HOF

// // wo function jo dusre function ko as an argument accept krta hai use HOF

// // 3: Callback Function

// // function which get passed as an argument to another function is called CBF

// // A callback function is a function that is passed as an argument to

// // another function, to be “called back” at a later time.

// // Jis bhi function ko hum kisi or function ke under as an arguments passed

// // krte hai then usko hum CallBack fun bolte hai

// // // we need to create a calculator

// const add = (a,b) => {

// return a+b;

// }

// // console.log(add(5,2));

// const subs = (a,b) => {

// return Math.abs(a-b);

// }

// const mult = (a,b) => {

// return a*b;

// }

// const calculator = (num1,num2, operator) => {

// return operator(num1,num2);

// }

```
// calculator(5,2,subs)
```

```
// console.log(calculator(5,2,subs));
```

```
// // // I have to do the hardcoded for each operation which is bad
```

```
// // // we will use the callback and the HOF to make it simple to use
```

```
// // // Now instead of calling each function individually we can call it
```

```
// // // by simply using one function that is calculator
```

```
// console.log(calculator(5,6,add));
```

```
// console.log(calculator(5,6,subs));
```

```
// console.log(calculator(5,6,mult));
```

```
// // In the above example, calculator is the higher-order function,
```

```
// // which accepts three arguments, the third one being the callback.
```

```
// // Here the calculator is called the Higher Order Function because it takes
```

```
// // another function as an argument
```

```
// // and add, sub and mult are called the callback function bcz they are passed
```

```
// // as an argument to another function
```

```
// // Interview Question
```

```
// // Difference Between Higher Order Function and Callback Function ?
```

```
// // Part Asynchronous JavaScript
```

```
// // 6: Synchronous JavaScript Prog
```

```
// 1work = 10min
```

```
// 2work = 5s
```

```
// const fun2 = () => {
```

```
//   console.log(`Function 2 is called`);
```

```
// }
```

```
// const fun1 = () => {
```

```
//   console.log(`Function 1 is called`);
```

```
//   fun2();
```

```
//   console.log(`Function 1 is called Again 🐼`);
```

```
// }
```

```
// fun1();
```

```
// Asynchronous JavaScript Prog
```

```
// const fun2 = () => {
```

```
//   setTimeout(()=> {
```

```
//     console.log(`Function 2 is called`);
```

```
//   }, 2000);
```

```
// }
```

```
// const fun1 = () => {
```

```
//   console.log(`Function 1 is called`);
```

```
//   fun2();
```

```
// console.log(`Function 1 is called Again 🙌`);  
// }
```

```
// fun1();
```

```
// //🤖 What is Event Loop in JavaScript?
```

```
// // ppt explain
```

```
// // 📖 Hoisting in JavaScript
```

```
// // we have a creation phase and execution phase.
```

```
// // Hoisting in Javascript is a mechanism where variables and functions
```

```
// // declarations are moved to the top of their scope before the code execute.
```

```
// For Example 🖱
```

```
// console.log(myName);
```

```
// let myName;
```

```
// myName = "thapa";
```

```
// // How it will be in output during creation phase
```

```
// 1: var myName = undefined;
```

```
// 2: console.log(myName);
```

```
// 3: myName = "thapa";
```

```
// //🔗 In ES2015 (a.k.a. ES6), hoisting is avoided by using the let keyword
```

```
// // instead of var. (The other difference is that variables declared
```

// // with let are local to the surrounding block, not the entire function.)

// // 📄 What is Scope Chain and Lexical Scoping in JavaScript?

// // The scope chain is used to resolve the value of variable names

// // in JS.

// // scope chain in js is lexically defined, which means that we can

// // see what the scope chain will be by looking at the code.

// // At the top, we have the Global Scope, which is the window Object

// // in the browser.

// // Lexical Scoping means Now, the inner function can get access to

// // their parent functions variables But the vice-versa is not true.

// // For Example 🖱️

// let a = "Hello guys. "; // global scope

// const first= () => {

// let b = " How are you?"

// const second = () => {

// let c = " Hii, I am fine thank you 🖱️";

// console.log(a+b+c);

// }

// second();

```
// console.log(a+b+c); //I can't use C
// }
```

```
// first();
```

```
// // 📖 What is Closures in JavaScript ?
```

```
// // A closure is the combination of a function bundled together (enclosed) with references
// // to its surrounding state (the lexical environment).
```

```
// // In other words, a closure gives you
// // access to an outer function's scope from an inner function.
```

```
// // In JavaScript, closures are created every time a function is created, at function creation time.
```

```
// // For Example 🖱
```

```
// const outerFun = (a) => {
//   let b = 10;
//   const innerFun = () => {
//     let sum = a+b;
//     console.log(`the sum of the two no is ${sum}`);
//   }
//   innerFun();
// }
// outerFun(5);
```

```
// // it same like lexical scoping
```

// // One more Example 🖱

```
// const outerFun = (a) => {  
  // let b = 10;  
  // const innerFun = () => {  
    // let sum = a+b;  
    // console.log(`the sum of the two no is ${sum}`);  
  // }  
  // return innerFun;  
// }  
// let checkClousure = outerFun(5);  
// console.dir(checkClousure);
```

// "use strict"

```
// let x = "vinod";  
// console.log(x);
```

// // 🚧 Back To Advanced JavaScript

// Currying

```
// const sum = (num1) => (num2) => (num3) => console.log(num1+num2+num3);
```

```
// sum(5)(3)(8);
```



```
// // *****
```

```
// // 📄 // 📄 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 📄
```

```
// 📄 // 📄 https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
```

```
// // *****
```

```
// // 8📄: CallBack Hell
```

```
// setTimeout(()=>{  
//   console.log(`1📄 works is done`);  
//   setTimeout(()=>{  
//     console.log(`2📄 works is done`);  
//     setTimeout(()=>{  
//       console.log(`3📄 works is done`);  
//       setTimeout(()=>{  
//         console.log(`4📄 works is done`);  
//         setTimeout(()=>{  
//           console.log(`5📄 works is done`);  
//           setTimeout(()=>{  
//             console.log(`6📄 works is done`);  
//             }, 1000)  
//           }, 1000)  
//         }, 1000)  
//       }, 1000)  
//     }, 1000)  
//   }, 1000)  
// }, 1000)
```

```
// // *****
```

```
// // ☞ // Bonus JSON ☞
```

```
// // *****
```

```
// // ☞ JSON.stringify turns a JavaScript object into JSON text and  
// stores that JSON text in a string, eg:
```

```
// var my_object = { key_1: "some text", key_2: true, key_3: 5 };
```

```
// var object_as_string = JSON.stringify(my_object);  
// // "{"key_1":"some text","key_2":true,"key_3":5}"
```

```
// console.log(object_as_string);
```

```
// typeof(object_as_string);  
// "string"
```

```
// // ☞ JSON.parse turns a string of JSON text into a JavaScript object, eg:
```

```
// var object_as_string_as_object = JSON.parse(object_as_string);  
// // {key_1: "some text", key_2: true, key_3: 5}
```

```
// typeof(object_as_string_as_object);  
// // "object"
```

// // 📄 AJAX Call using XMLHttpRequest

// // how to handled with the events and callback

// // XMLHttpRequest (XHR) objects are used to interact with servers.

// // You can retrieve data from a URL without having to do a full

// // page refresh. This enables a Web page to update just part

// // of a page without disrupting what the user is doing.

// // XMLHttpRequest is used heavily in AJAX programming.

// const request = new XMLHttpRequest();

// // we need to call the api or request the api using GET method ki, me jo

// // url pass kar kr rha hu uska data chaiye

// request.open('GET', "https://covid-api.mmediagroup.fr/v1");

// request.send(); // we need to send the request and its async so we need to

// // add the event to load the data adn get it

// // to get the response

// request.addEventListener("load", () => {

// console.log(this.responseText);

// });