index.js

```
// 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);

```

```
// 1️⃣Assignment 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}`);

// 2️⃣Arithmetic 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);

// 🙄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.

// 3️⃣Comparison 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;
```

```
166
167 // Equal (==)
168 // console.log(a == b);
169
170 // Not equal (≠)
171 // console.log(a ≠ b);
172
173 // // Greater than (>)
174 // console.log(a > b);
175
176 // // Greater than or equal (>=)
177 // console.log(a >= b);
178
179 // // Less than (<)
180 // console.log(a < b);
181
182 // // Less than or equal (<=)
183 // console.log(a <= b);
184
185 // 4️⃣ Logical operators
186 // Logical operators are typically used with Boolean (logical) values;
187 // when they are, they return a Boolean value.
188
189 // var a = 30;
190 // var b = -20;
191
192 // Logical AND (&&)
193 // The logical AND (&&) operator (logical conjunction) for a set of
194 // operands is true if and only if all of its operands are true.
195
196 // console.log(a > b && b > -50 && b < 0);
197
198 // Logical OR (||)
199 // The logical OR (||) operator (logical disjunction) for a set of
200 // operands is true if and only if one or more of its operands is true.
201
202 // console.log((a < b) || (b > 0) || (b > 0));
203
204 // Logical NOT (!)
205 // The logical NOT (!) operator (logical complement, negation)
206 // takes truth to falsity and vice versa.
207
208 // console.log(!((a>0) || (b<0)));
209 // console.log(!true);
210
211 // // ************************************************************************
212
213 // // 👉 // 😍 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 😍
214 //   👉 // 😍  https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
215
216 // // ************************************************************************
217
218 // 5️⃣ String Concatenation(operators)
219 // The concatenation operator (+) concatenates two string values together,
220 // returning another string that is the union of the two operand strings.
221
```

```
// 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 -
 *
 * 1️⃣If ... 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 😀");
// }

// 2️⃣Conditional (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");

// 3️⃣ 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.

// 4️⃣ 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++;
// }

// 5️⃣ Do-While Loop Statement

// var num = 20;
// do{
//   debugger;
//   console.log(num); //infinte loop
//   num++;
// }while(num <= 10);

// 6️⃣ 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);
// }
```

```
446
447 // ************************************************************
448
449 /**** Section 5 👉 Functions in JavaScript ****/
450
451 // A JavaScript function is a block of code designed to perform a particular task.
452
453 //   1️⃣Function Definition
454
455 // Before we use a function, we need to define it.
456
457 // A function definition (also called a function declaration, or function statement)
458 // consists of the function keyword, followed by:
459
460 // The name of the function.
461 // A list of parameters to the function, enclosed in parentheses and separated by
    commas.
462 // The JavaScript statements that define the function, enclosed in curly brackets,
    { ... }.
463
464 // var a = 10;
465 // var b = 20;
466 // var sum = a+b;
467 // console.log(sum);
468
469 // function sum(){
470 //   var a = 10, b = 40;
471 //   var total = a+b;
472 //   console.log(total);
473 // }
474 // //
475
476 // 2️⃣Calling functions
477 // Defining a function does not execute it.
478 // A JavaScript function is executed when "something" invokes it (calls it).
479
480 // function sum(){
481 //   var a = 10, b = 40;
482 //   var total = a+b;
483 //   console.log(total);
484 // }
485
486 // sum();
487
488 // 3️⃣ Function Parameter vs Function Arguments
489 // Function parameters are the names listed in the function's definition.
490 // Function arguments are the real values passed to the function.
491
492 // function sum(a,b){
493 //   var total = a+b;
494 //   console.log(total);
495 // }
496
497 // sum();
498 // sum(20,30);
499 // sum(50,50);
500 // sum(5,6)
```

```
501
502 // // ************************************************************************
503
504 // // 👉 // 😍 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 😍
505 //   👉 // 😍  https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
506
507 // // ************************************************************************
508
509 // 🙋‍♂️🙋‍♂️ Interview Question 🙋‍♂️🙋‍♂️
510
511 // Why Functions?
512
513 // You can reuse code: Define the code once, and use it many times.
514 // You can use the same code many times with different arguments,
515 // to produce different results.
516
517 // OR
518
519 // A function is a group of reusable code which can be called anywhere
520 // in your program. This eliminates the need of writing the same code
521 // again and again.
522
523 // DRY => do not repeat yourself
524
525 // 4️⃣ Function expressions
526 // "Function expressions simply means
527 // create a function and put it into the variable "
528
529 // function sum(a,b){
530 //   var total = a+b;
531 //   console.log(total);
532 // }
533
534 // var funExp = sum(5,15);
535
536 // 5️⃣ Return Keyword
537 // When JavaScript reaches a return statement,
538 // the function will stop executing.
539
540 // Functions often compute a return value.
541 // The return value is "returned" back to the "caller"
542
543 // function sum(a,b){
544 //   return total = a+b;
545 // }
546
547 // var funExp = sum(5,25);
548
549 // console.log('the sum of two no is ' + funExp );
550
551 // 6️⃣ Anonymous Function
552
553 // A function expression is similar to and has the same syntax
554 // as a function declaration One can define "named"
555 // function expressions (where the name of the expression might
556 // 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

// 2️⃣ Template literals (Template strings)
```

```
613
614 // JavaScript program to print table for given number (8)?
615
616 // output :  8 * 1 = 8
617 //   8 * 2 = 16(8*2)
618 //  => 8 * 10 = 80
619
620 // for(let num = 1; num<= 10; num++){
621 //     let tableOf = 12;
622 //   // console.log(tableOf + " * " + num + " = " + tableOf * num);
623 //   console.log( ` ${tableOf} * ${num} = ${tableOf * num}` );
624 // }
625
626 // 3️⃣  Default Parameters
627 // Default function parameters allow named parameters to be
628 // initialized with default values if no value or undefined is passed.
629
630 // function mult(a,b=5){
631 //   return a*b;
632 // }
633
634 // console.log(mult(3));
635
636 // 4️⃣ Destructuring in ES6
637 // The destructuring assignment syntax is a JavaScript expression
638 // that makes it possible to unpack values from arrays,
639 // or properties from objects, into distinct variables.
640
641 //  ➡ Array Destructuring  🏁
642
643 // const myBioData = ['vinod', 'thapa', 26];
644
645 // let myFName = myBioData[0];
646 // let myLName = myBioData[1];
647 // let myAge = myBioData[2];
648
649 // let [myFName,myAge, myLName] = myBioData;
650 // console.log(myAge);
651
652 // we can add values too
653
654 // let [myFName,myLName,myAge, myDegree="MCS"] = myBioData;
655 // console.log(myDegree);
656
657 // ➡ Object destructuring 🏁
658 // const myBioData = {
659 //   myFname : 'vinod',
660 //   myLname : 'thapa',
661 //   myAge : 26
662 // }
663
664 // let age = myBioData.age;
665 // let myFname = myBioData.myFname;
666
667 // let {myFname,myLname,myAge, myDegree="MCS"} = myBioData;
668 // console.log(myLname);
```

```

// 5️⃣ 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);

// 6️⃣ Fat Arror 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());

// 7️⃣Spread 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];

// 1️⃣ 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.
```

```
837
838 // var myFriends = ['vinod','ramesh','arjun','vishal'];
839
840 // myFriends.forEach(function(element, index, array) {
841 //       console.log(element + " index : " +
842 //         index + " " + array);
843 // });
844
845 // myFriends.forEach((element, index, array) => {
846 //   console.log(element + " index : " +
847 //     index + " " + array);
848 // });
849
850 // 2️⃣ Array Subsection 2 👉 Searching and Filter in an Array
851
852 // Array.prototype.indexOf() 🙋
853
854 // Returns the first (least) index of an element within the array equal
855 // to an element, or -1 if none is found. It search the element from the
856 // 0th index number
857
858 // var myFriendNames = ["vinod","bahadur","thapa","thapatechnical","thapa"];
859
860 // console.log(myFriendNames.indexOf("Thapa", 3));
861
862 // Array.prototype.lastIndexOf() 🙋
863 // Returns the last (greatest) index of an element within the array equal
864 // to an element, or -1 if none is found. It search the element last to first
865
866 // var myFriendNames = ["vinod","bahadur","thapa","thapatechnical","thapa"];
867
868 // console.log(myFriendNames.lastIndexOf("Thapa",3));
869
870 // Array.prototype.includes() 🙋
871 // Determines whether the array contains a value,
872 // returning true or false as appropriate.
873
874 // var myFriendNames = ["vinod","bahadur","thapa","thapatechnical"];
875
876 // console.log(myFriendNames.includes("thapa"));
877
878 // Array.prototype.find() 🙋
879
880 // arr.find(callback(element[, index[, array]])[, thisArg])
881
882 // Returns the found element in the array, if some element in the
883 // array satisfies the testing function, or undefined if not found.
884 // Only problem is that it return only one element
885
886 // const prices = [200,300,350,400,450,500,600];
887
888 // price < 400
889 // const findElem = prices.find((currVal) =>  currVal < 400 );
890 // console.log(findElem);
891
892 // console.log(  prices.find((currVal) => currVal > 1400 )  );
```

```
893
894 // // ************************************************************************
895
896 // // 👉 // 😍 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 😍
897 //   👉 // 😍  https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA
898
899 // // ************************************************************************
900
901 // Array.prototype.findIndex() 🙋‍♂️
902
903 // Returns the found index in the array, if an element in the
904 // array satisfies the testing function, or -1 if not found.
905
906 // console.log(  prices.findIndex((currVal) => currVal > 1400 )  );
907
908 // Array.prototype.filter() 🙋‍♂️
909
910 // Returns a new array containing all elements of the calling
911 // array for which the provided filtering function returns true.
912
913 // const prices = [200,300,350,400,450,500,600];
914
915 // // price < 400
916 // const newPriceTag = prices.filter((elem, index) => {
917 //   return elem > 1400;
918 // })
919 // console.log(newPriceTag);
920
921 // 3️⃣ Array Subsection 3 👉 How to sort an Array
922
923 // Array.prototype.sort() 🙋‍♂️
924
925 // The sort() method sorts the elements of an array in place and
926 //returns the sorted array. The default sort order is ascending, built
927 //upon converting the elements into strings,
928 // then comparing their sequences of UTF-16 code units values.
929
930 // const months = ['March', 'Jan', 'Feb', 'April' , 'Dec', 'Nov'];
931
932 // console.log(months.sort());
933
934 // const array1 = [1, 30, 4, 21, 100000, 99];
935 // console.log(array1.sort());
936
937 // However, if numbers are sorted as strings,
938 // "25" is bigger than "100", because "2" is bigger than "1".
939
940 // Because of this, the sort() method will produce an incorrect
941 // result when sorting numbers.
942
943 // 😃7: challenge Time  🏁
944
945 // 1: How to Sort the numbers in the array in ascending (up) and descending (down)
    order?
946
947 // 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 asecnding 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 desecnding 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.

// 4️⃣ 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.
```

```
1003
1004 // const animals = ['pigs', 'goats', 'sheep'];
1005
1006 // // const count = animals.push('chicken');
1007 // // console.log(count);
1008
1009 // animals.push('chicken', 'cats','cow');
1010 // console.log(animals);
1011
1012 // Array.prototype.unshift() 🙋
1013 // The unshift() method adds one or more elements to the
1014 // beginning of an array and returns the new length of the array.
1015
1016 // const animals = ['pigs', 'goats', 'sheep'];
1017
1018 // const count = animals.unshift('chicken');
1019 // console.log(count);
1020 // console.log(animals);
1021
1022 // animals.unshift('chicken', 'cats','cow');
1023 // console.log(animals);
1024
1025 // 2nd example
1026
1027 // const myNumbers = [1,2,3,5];
1028
1029 // myNumbers.unshift(4,6);
1030 // console.log(myNumbers);
1031
1032 // Array.prototype.pop() 🙋
1033 // The pop() method removes the last element from an array and returns
1034 // that element. This method changes the length of the array.
1035
1036 // const plants = ['broccoli', 'cauliflower',  'kale', 'tomato', 'cabbage'];
1037
1038 // console.log(plants);
1039 // console.log(plants.pop());
1040 // console.log(plants);
1041
1042 // Array.prototype.shift() 🙋
1043 // The shift() method removes the first element from an array and returns
1044 // that removed element. This method changes the length of the array.
1045
1046 // const plants = ['broccoli', 'cauliflower',  'kale', 'tomato', 'cabbage'];
1047 // console.log(plants);
1048 // console.log(plants.shift());
1049 // console.log(plants);
1050
1051 // 😃8: challenge Time 🏁
1052
1053 // Array.prototype.splice() 🙋
1054 // Adds and/or removes elements from an array.
1055
1056 // 1: Add Dec at the end of an array?
1057 // 2: What is the return value of splice method?
1058 // 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');
// }

// 5️⃣ 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((curElm, index, arr) => {
//     return `Index no = ${index} and the value is ${curElm} belong to ${arr} `
// }).reduce().
```

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

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

// It return new array without mutating the orignal 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((curElm) => curElm * 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]
```

```
1170
1171 // let arr = [5,6,2];
1172
1173 // let sum = arr.reduce((accumulator, curElem) => {
1174 //         debugger;
1175 //       return accumulator += curElem;
1176 // },7)
1177 // console.log(sum);
1178
1179 // How to fatten an array
1180 // converting 2d and 3d array into one dimensional array
1181
1182 // const arr = [
1183 //         ['zone_1', 'zone_2'],
1184 //         ['zone_3', 'zone_4'],
1185 //         ['zone_5', 'zone_6'],
1186 //         ['zone_7', ['zone_7', ['zone_7', 'zone_8']]]
1187 //     ];
1188
1189 // // let flatArr = arr.reduce((accum, currVal)  => {
1190 // //           return accum.concat(currVal);
1191 // // })
1192
1193 // console.log(arr.flat(Infinity));
1194
1195 // console.log(flatArr);
1196
1197 // const arr = [ ['zone_1', 'zone_2'], ['zone_3', ['zone_1', 'zone_2', ['zone_1', '
    zone_2']]] ];
1198 // console.log(arr.flat(3));
1199 // console.log(arr);
1200
1201 /**** Section 7👉 Strings in JavaScript  ****/
1202
1203 // A JavaScript string is zero or more characters written inside quotes.
1204
1205 // JavaScript strings are used for storing and manipulating text.
1206 // You can use single or double quotes
1207
1208 // Strings can be created as primitives,
1209 // from string literals, or as objects, using the String() constructor
1210
1211 // let myName = "vinod thapa";
1212 // let myChannelName = 'vinod thapa';
1213
1214 // // let ytName = new String("Thapa Technical");
1215 // let ytName = 'thapa technical';
1216
1217 // console.log(myName);
1218 // console.log((ytName));
1219
1220 // 👉 How to find the length of a string
1221 // String.prototype.length 🙋‍♂️
1222 // Reflects the length of the string.
1223
1224 // 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(regexp) 🙋

// 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.
```

```
1281
1282 // The method takes 2 parameters: the start position,
1283 // and the end position (end not included).
1284
1285 // var str = "Apple, Bananaa, Kiwi, mango";
1286
1287 // // let res = str.slice(0,4);
1288 // let res = str.slice(7);
1289 // console.log(res);
1290
1291 // The slice() method selects the elements starting at the
1292 // given start argument, and ends at, but does not include,
1293 // the given end argument.
1294
1295 // Note: The original array will not be changed.
1296 // Remember: JavaScript counts positions from zero. First position is 0.
1297
1298 // 😜11: challenge Time 🏁
1299
1300 // Display only 280 characters of a string like the
1301 // one used in Twitter?
1302
1303 // 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? ";
1304
1305 // let myActualTweet = myTweets.slice(0,280);
1306 // console.log(myActualTweet);
1307 // console.log(myActualTweet.length);
1308
1309 // The substring() Method 🙋
1310 // substring() is similar to slice().
1311
1312 // The difference is that substring() cannot accept
1313 // negative indexes.
1314
1315 // var str = "Apple, Bananaa, Kiwi";
1316 // let res = str.substring(8,-2);
1317 // console.log(res);
1318
1319 // // If we give negative value then the characters are
1320 //  counted from the 0th pos
1321
1322 // The substr() Method 🙋
1323 // substr() is similar to slice().
1324
1325 // The difference is that the second parameter specifies the
1326 // length of the extracted part.
1327
1328 // var str = "Apple, Bananaa, Kiwi";
1329 // // let res = str.substr(7,-2);
1330 // let res = str.substr(-4);
1331 // console.log(res);
1332
```

```
// 👉 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 88-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
```

```
1389
1390 // let str = "HELLO WORLD";
1391 // let lastChar = str.length - 1;
1392 // console.log(str.charCodeAt(lastChar));
1393
1394 // Property Access
1395 // ECMAScript 5 (2009) allows property access [ ] on strings
1396
1397 // var str = "HELLO WORLD";
1398 // console.log(str[1]);
1399
1400 //👉  Other useful methods
1401
1402 // let myName = "vinod tHapa";
1403 // console.log(myName.toUpperCase());
1404 // console.log(myName.toLowerCase());
1405
1406 // The concat() Method 🙋‍♂️
1407 // concat() joins two or more strings
1408
1409 // let fName = "vinod"
1410 // let lName = "thapa"
1411
1412 // console.log(fName + lName );
1413 // console.log(`${fName} ${lName}`);
1414 // console.log(fName.concat(lName));
1415 // console.log(fName.concat(" " ,lName));
1416
1417 // String.trim() 🙋‍♂️
1418 // The trim() method removes whitespace from both
1419 // sides of a string
1420
1421 // var str = "              Hello         World!            ";
1422 // console.log(str.trim());
1423
1424 // Converting a String to an Array
1425 // A string can be converted to an array with the
1426 // split() method
1427
1428 // var txt = "a, b,c d,e";   // String
1429 // console.log(txt.split(","));           // Split on commas
1430 // console.log( txt.split(" "));          // Split on spaces
1431 // console.log(txt.split("|"));         // Split on pipe
1432
1433 /**** Section 8👉 Date and Time in JavaScript ****/
1434
1435 // JavaScript Date objects represent a single moment in time in a
1436 // platform-independent format. Date objects contain a Number
1437 // that represents milliseconds since 1 January 1970 UTC.
1438
1439 // 👉 Creating Date Objects
1440 // There are 4 ways to create a new date object:
1441
1442 // new Date()
1443 // new Date(year, month, day, hours, minutes, seconds, milliseconds)
1444 // // 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 indivisual 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 indivisual 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 indivisual 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 indivisual 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);
```

```
1556
1557 // Math.round() 🙋
1558 // returns the value of x rounded to its nearest integer
1559
1560 // let num = 10.501;
1561 // console.log(Math.round(num));
1562
1563 // Math.pow() 🙋
1564 // Math.pow(x, y) returns the value of x to the power of y
1565
1566 // console.log(Math.pow(2,3));
1567 // console.log(2**3);
1568
1569 // Math.sqrt() 🙋
1570 // Math.sqrt(x) returns the square root of x
1571
1572 // console.log(Math.sqrt(25));
1573 // console.log(Math.sqrt(81));
1574 // console.log(Math.sqrt(66));
1575
1576 // Math.abs() 🙋
1577 // Math.abs(x) returns the absolute (positive) value of x
1578
1579 // console.log(Math.abs(-55));
1580 // console.log(Math.abs(-55.5));
1581 // console.log(Math.abs(-955));
1582 // console.log(Math.abs(4-6));
1583
1584 // Math.ceil() 🙋
1585 // Math.ceil(x) returns the value of x rounded up to its nearest integer
1586
1587 // console.log(Math.ceil(4.51));
1588 // console.log(Math.round(4.51));
1589 // console.log(Math.ceil(99.01));
1590 // console.log(Math.round(99.1));
1591
1592 // Math.floor()  🙋
1593 // Math.floor(x) returns the value of x rounded down to its nearest integer
1594
1595 // console.log(Math.floor(4.7));
1596 // console.log(Math.floor(99.1));
1597
1598 // Math.min() 🙋
1599 // Math.min() can be used to find the lowest value in a list of arguments
1600
1601 // console.log(Math.min(0, 150, 30, 20, -8, -200));
1602
1603 // Math.max() 🙋
1604 // Math.max() can be used to find the highest value in a list of arguments
1605
1606 // console.log(Math.max(0, 150, 30, 20, -8, -200));
1607
1608 // Math.random() 🙋
1609 // Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)
1610
1611 // 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

// 1️⃣ 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.

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

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

// // 👉 // 😍 SUBSCRIBE TO THAPA TECHNICAL YOUTUBE CHANNEL 😍
//  👉 // 😍  https://www.youtube.com/channel/UCwfaAHy4zQUb2APNOGXUCCA

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

// 2️⃣ 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.

// 2️⃣ 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()
```

```
1668
1669 // 3️⃣ Window has methods, properties and object.
1670 // ex setTimeout() or setInterval() are the methods
1671 // where as Document is the object of the Window and
1672 // It also has a screen object with properties
1673 // describing the physical display.
1674
1675 // Now, I know you have a doubt like we have seen the methods
1676 // and object of the global object that is window. But What about
1677 // the properties of the Window Object 😮
1678
1679 // so example of window object properties are
1680 // innerHeight,
1681 // innerWidth and there are many more
1682
1683 // let's see some practical in DOM HTML file
1684
1685 // ************** DOM vs BOM ********************
1686
1687 // 👉 The DOM is the Document Object Model, which deals with the document,
1688 // the HTML elements themselves, e.g. document and all traversal you
1689 // would do in it, events, etc.
1690
1691 // For Ex: 🤦
1692 // change the background color to red
1693 // document.body.style.background = "red";
1694
1695 // 👉 The BOM is the Browser Object Model, which deals with browser components
1696 // aside from the document, like history, location, navigator and screen
1697 // (as well as some others that vary by browser). OR
1698 // In simple meaning all the Window operations which comes under BOM are performed
1699 // usign BOM
1700
1701 // Let's see more practical on History object
1702
1703 // Functions alert/confirm/prompt are also a part of BOM:
1704 // they are directly not related to the document,
1705 // but represent pure browser methods of communicating with the user.
1706
1707 // alert(location.href); // shows current URL
1708 // if (confirm("Want to Visit ThapaTechnical?")) {
1709 //   location.href = "https://www.youtube.com/thapatechnical"; // redirect the
    browser to another URL
1710 // }
1711
1712 // Section 3️⃣: Navigate through the DOM
1713
1714 // 1: document.documentElement
1715 // returns the Element that is the root element of the document.
1716 // 2: document.head
1717 // 3: document.body
1718 // 4: document.body.childNodes (include tab,enter and whiteSpace)
1719 // list of the direct children only
1720 // 5: document.children (without text nodes, only regular Elements)
1721 // 6: document.childNodes.length
1722
```

```
// 👉 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.firstElementChild
// data.firstElementChild.firstElementChild
// data.firstElementChild.firstElementChild.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 4️⃣:  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 1️⃣ 4 ways of writing Events in JavaScript

// 1: using inline events alert();
// 2: By Calling a funcion (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()

// 2clearTimeout()

// 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 );

// 2️⃣ 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

// // 1️⃣: Event Propagation (Event Bubbling and Event Capturing)

// // check html file

// // 2️⃣: 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

// // 3️⃣: 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
```

```
2000
2001 // // // Now instead of calling each function indivisually we can call it
2002 // // // by simply using one function that is calculator
2003
2004 // console.log(calculator(5,6,add));
2005 // console.log(calculator(5,6,subs));
2006 // console.log(calculator(5,6,mult));
2007
2008 // // In the above example, calculator is the higher-order function,
2009 // // which accepts three arguments, the third one being the callback.
2010 // // Here the calculator is called the Higher Order Function because it takes
2011 // // another function as an argument
2012
2013 // // and add, sub and mult are called the callback function bcz they are passed
2014 // // as an argument to another fucntion
2015
2016 // // InterView Question
2017 // // Difference Between Higher Order Function and Callback Function ?
2018
2019 // // 🏁🏁Asynchronous JavaScript
2020
2021 // // 6️⃣:   Synchronous JavaScript Prog
2022
2023 // 1work = 10min
2024 // 2work = 5s
2025
2026 // const fun2 = () => {
2027 //   console.log(`Function 2️⃣ is called`);
2028 // }
2029
2030 // const fun1 = () => {
2031 //   console.log(`Function 1 is called`);
2032 //   fun2();
2033 //   console.log(`Function 1 is called Again ✌️`);
2034 // }
2035
2036 // fun1();
2037
2038 // Asynchronous JavaScript Prog
2039
2040 // const fun2 = () => {
2041 //   setTimeout(()=> {
2042 //       console.log(`Function 2️⃣  is called`);
2043 //   }, 2000);
2044 // }
2045
2046 // const fun1 = () => {
2047 //   console.log(`Function 1 is called`);
2048 //   fun2();
2049 //   console.log(`Function 1 is called Again ✌️`);
2050 // }
2051
2052 // fun1();
2053
2054 // //👻 What is Event Loop in JavaScript?
2055 // // ppt explain
```

```
2056
2057 // // 5️⃣ Hoisting in JavaScript
2058
2059 // // we have a creation phase and execution phase.
2060
2061 // // Hoisting in Javascript is a mechanism where variables and functions
2062 // // declarations are moved to the top of their scope before the code execute.
2063
2064 // For Example 👇
2065 // console.log(myName);
2066 // let myName;
2067 // myName = "thapa";
2068
2069 // // How it will be in output during creation phase
2070
2071 // 1: var myName = undefined;
2072 // 2: console.log(myName);
2073 // 3: myName = "thapa";
2074
2075 // //😳 In ES2015 (a.k.a. ES6), hoisting is avoided by using the let keyword
2076 // // instead of var. (The other difference is that variables declared
2077 // // with let are local to the surrounding block, not the entire function.)
2078
2079 // // 6️⃣ What is Scope Chain and Lexical Scoping in JavaScript?
2080
2081 // // The scope chain is used to resolve the value of variable names
2082 // // in JS.
2083
2084 // // scope chain in js is lexically defined, which means that we can
2085 // // see what the scope chain will be by looking at the code.
2086
2087 // // At the top, we have the Global Scope, which is the window Object
2088 // // in the browser.
2089
2090 // // Lexical Scoping means Now, the inner function can get access to
2091 // // their parent functions variables But the vice-versa is not true.
2092
2093 // // For Example 👇
2094
2095 // let a = "Hello guys. "; // global scope
2096
2097 // const first= () => {
2098 //   let b = " How are you?"
2099
2100 //     const second = () => {
2101 //       let c = " Hii, I am fine thank you🙏";
2102 //       console.log(a+b+c);
2103 //     }
2104 //     second();
2105 //     console.log(a+b+c); //I can't use C
2106 // }
2107
2108 // first();
2109
2110 // // 7️⃣ What is Closures in JavaScript 🤔
2111
```

```
// // 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

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

```

```
// // 8️⃣: CallBack Hell

// setTimeout(()=>{
//     console.log(`1️⃣ works is done`);
//     setTimeout(()=>{
//         console.log(`2️⃣ works is done`);
//         setTimeout(()=>{
//             console.log(`3️⃣ works is done`);
//             setTimeout(()=>{
//                 console.log(`4️⃣ works is done`);
//                 setTimeout(()=>{
//                     console.log(`5️⃣ works is done`);
//                     setTimeout(()=>{
//                         console.log(`6️⃣ works is done`);
//                     }, 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"

// // 7️⃣ 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);
// });

```