JavaScript Notes.

What is JavaScript ?

i. Javascript is a high-level , object based ,programming language. Javascript is a interpreted, user friendly ,client-side scripting language .

History of javascript ?

i. javascript developed by Brenden Eich-1995 .
ii. First name of javascript --> MOCHA .
iii. next --> LIVESCRIPT .
iv. next--> JAVASCRIPT .

Features of JavaScript?

- Light Weight
- Open Source
- Cross Platform
- Dynamically typed language
- Interpreted language
- Client-side scripting language
- Synchronous language
- Weakly typed language

Javascript characteristics ?

- Clientside Scripting language - no compiler and without help of server logic we can update the data .
- High-level language - user-friendly .
- Interpreted language - line by line execution .
- Single threaded - one call stack and one heap area .
- Loosely typed - no strong syntax .
- Dynamic language - can change the datatype during the runtime .
- Object-based language - In JavaScript everything was object .

Advantages of Javascript ?

1. Client-Side Execution :

JavaScript runs in the browser, reducing the load on the server and improving performance .

2. Speed :

   It executes quickly within the user's browser without the need for server interaction .

3. Interactivity & Dynamic Content :

   Allows for interactive web pages (e.g., animations, dynamic updates, form validations) .

4. Rich Ecosystem & Libraries :

   Large ecosystem with frameworks like React, Vue, and Angular, as well as libraries like jQuery and Lodash .

5. Cross-Browser Support :

   Works on almost all modern browsers without requiring additional software .

6. Versatile (Front-end & Back-end) :

   Can be used for both client-side and server-side development (Node.js) .

7. Asynchronous Processing :

   Supports asynchronous programming using async/await and Promises, making it efficient for handling APIs and real-time applications .

Disadvantages of Javascript ?

1. Security Issues :

   Since JavaScript runs on the client side, it is vulnerable to attacks like cross-site scripting (XSS) and code injection . (Vulnerable means weak or unprotected, making something easy to attack or harm . In JavaScript security, if a website is vulnerable, it means hackers can easily break in, steal data, or damage the site because there are security weaknesses .)

2. Browser Inconsistencies :

   Different browsers may interpret JavaScript code differently, leading to compatibility issues .

3. Limited Multithreading :

   JavaScript is single-threaded meaning heavy tasks can block the main thread and affect performance

4. Loose Typing (Type Errors) :

   JavaScript is dynamically typed, which can lead to unexpected errors if not handled properly. (e.g., "5" + 5 = "55")

5. Dependency on Third-Party Libraries :

   Many projects rely heavily on external libraries, which can introduce security risks and maintenance issues .

Uses of JavaScript ?

We use javascript for Web application , Web development , Mobile application , Games development , Server application , Animate elements , Web server , Client-Side validation .

Difference betweeen Java and Javascript ?

Java	JavaScript
1. java is a strongly typed language and variables must be declared first to use in the program. In Java, the type of a variable is checked at compile-time .	1. JavaScript is a loosely typed language and has a more relaxed syntax and rules .
2. Java is an object-oriented programming language primarily used for developing complex logic applications .	2. JavaScript is a scripting language used for creating interactive and dynamic web pages .
3. Java applications can run in any virtual machine(JVM) or browser .	3. JavaScript code used to run only in the browser, but now it can run on the server via Node.js .
4. Objects of Java are class-based even we can't make any program in java without creating a class .	4. JavaScript Objects are prototype-based .
5. Java program has the file extension ".Java" and translates source code into bytecodes which are executed by JVM(Java Virtual Machine) .	5. JavaScript file has the file extension ".js" and it is interpreted but not compiled, every browser has the Javascript interpreter to execute JS code .
6. Java supports multithreading, which allows multiple threads of execution to run concurrently within a single program .	6. JavaScript does not support multithreading, although it can simulate it through the use of web workers .

7. Java is mainly used for backend .	7. Javascript is used for the frontend and backend both .
8. Java is statically typed .	8. JavaScript is dynamically typed .
9. Java uses more memory .	9. Javascript uses less memory .

How many ways add javascript file ?

Attaching the javascript file to html file : We can attach the js file with html file in 2 ways.

1. Internal Way : By writing the code of javascript inside script tag

2. External Way : To write javascript externally to html file, we need to create an external javascript file with extension as .js (ex: index.js) . After that link that javascript file to the html by using script tag .

Token :

Token is a smallest unit cell in a programming language and it has some specific meaning . The combination of keywords , identifiers , literals and operators make a valid token .

Keywords :

Reserved words that have special meanings in the language. Examples include var, if, else, for, while, etc . Every keyword must be in the lower case and it is not used as Identifiers.

Identifiers :

These are names that are used to identify variables, functions, and other objects. For example, the identifier myVariable could be used to identify a variable that stores the value 5.

Rulers: -an identifiers can't start with number. -We can't use keywords as identifiers. - Except $, _ no other Special Characteristics are allowed.

Literals :

Data which is used in the js programming is called as literals. For example, the literal 5 is a numeric literal (Data types), and the literal "Hello, world!" is a string literal

Operators :

These are symbols that are used to perform operations on operands. For example, the operator + is used to add two numbers together.

JavaScript Variable and rules ?

A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable .

There are some rules while declaring a JavaScript variable (also known as identifiers) .

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign .
2. After first letter we can use digits (0 to 9), for example value1 .
3. JavaScript variables are case sensitive, for example x and X are different variables .

Javascript Data Types :

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript .

1. Primitive data type :

Primitive data type can contain only single value .

1. String : To store string value we use "" , '' , `` .
2. Number : 1 , 0.1 , -1 .
3. Boolean : True , False .
4. Undefined : Define but not given the value (Declair but not initialize) .
5. Null : Empty Container .
6. BigInt : Means More than 16 digit to show it we use n after the number .

2. Non-primitive (reference) data type :

Non-Primitive data type can contain multiple value .

1. Object : represents instance through which we can access members .
2. Array : represents group of similar values .
3. Functions : represents reusable codes .

Operaters :

Operators are special symbols in JavaScript that are used to perform operations on values. There are many different types of operators, including arithmetic operators, comparison operators, logical operators, and assignment operators .

1. Arithmetic operators :

Arithmetic operators are use to do some arithmetic operation between two values .

```
EX -: + , - , * , / , % , **
```

2. Comparison operators :

Comparison operators are use to compair two values .

```
EX -: = , == , === , != , !== , > , >= , < , <=
```

3. Logical operators :

Logical operators are use to do some Logical operations .

```
EX -: && , || , !
```

4. Assignment operators :

Assignment operators are use to assign a value to a variable .

```
EX -: += , -+ , *= , /= , %= , **=
```

Conditional statement :

A conditional statement is a programming instruction that executes different actions based on whether a specified condition is true or false. It helps in decision-making in code .

1. JavaScript If statement :

It evaluates the content only if expression is true .

```
Syntax -: if(expression){

//content to be evaluated

}
```

```
EX -: var a=20;

if(a>10){

document.write("value of a is greater than 10");
```

```
}
```

2. JavaScript If...else Statement :

It evaluates the content whether condition is true of false .

```
Syntax -: if(expression){

//content to be evaluated if condition is true

}

else{

//content to be evaluated if condition is false

}
```

```
EX -: var a=20;

if(a>10){

document.write("value of a is greater than 10");

}
```

3. JavaScript If...else if statement :

it evaluates the content only if expression is true from several expressions .

```
Syntax -: if(expression1){

//content to be evaluated if expression1 is true

}

else if(expression2){

//content to be evaluated if expression2 is true

}

else if(expression3){

//content to be evaluated if expression3 is true

}
```

```
else{

//content to be evaluated if no expression is true

}
```

```
Ex -: var a=20;

if(a==10){

document.write("a is equal to 10");

}

else if(a==15){

document.write("a is equal to 15");

}

else if(a==20){

document.write("a is equal to 20");

}

else{

document.write("a is not equal to 10, 15 or 20");

}
```

4. JavaScript Switch statement :

Switch Statement Execute the block of code depend on the different cases . if we not give the break keyword then ,even the condition is satisfied it will execute the remaining blocks .

Break Keyword: It Will Break the Execution At that particular point ..

```
Syntax -: switch(expression) {

case n:

// code block

break;
```

```
case n:
//code block
break;
default:
default: //code block
}
```

```
EX -: const food = "nuts";
switch (food) {
case "cake":
console.log("I like cake");
break;
case "pizza":
console.log("I like pizza");
break;
default:
console.log("I like all foods");
break;
case "ice cream":
console.log("I like ice cream");
break;
}
```

JavaScript Loops :

JavaScript Loops are powerful tools for performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true. OR

Whenever you want to execute a block of code again and again until the condition fails you need to use loops .

1. For Loop :

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping .

```
Syntax -: for (initialization; testing condition; increment/decrement) {

statement(s)

}
```

```
Ex -: for (x = 2; x <= 4; x++) {

console.log("Value of x: " + x);

}
```

Initialization condition : It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only .

Test Condition : It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements .

Statement execution : Once the condition is evaluated to be true, the statements in the loop body are executed .

Increment/ Decrement : It is used for updating the variable for the next iteration .

Loop termination : When the condition becomes false, the loop terminates marking the end of its life cycle

2. While Loop :

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement . (First it will check the condition then execute the code .)

```
Syntax -: while (boolean condition) {

loop statements...

}
```

```
Ex -: let val = 1;

while (val < 6) {

console.log(val);

val += 1;

}
```

While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the Entry control loop .

Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration .

When the condition becomes false, the loop terminates which marks the end of its life cycle.

3. do-While Loop :

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an Exit Control Loop. It executes loop content at least once event the condition is false . (first execute the code then check the condition .)

```
Syntax -: do {

Statements...

}

while (condition);
```

```
Ex -: let test = 1;

do {

console.log(test);

test++;

} while(test <= 5);
```

The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time .

After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts .

When the condition becomes false, the loop terminates which marks the end of its life cycle .

It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop .

4. For-in Loop :

There are also other types of loops. The for..in loop in JavaScript allows you to iterate over all property keys of an object .

To iterate an object we use for-in loop

Note: In each iteration of the loop, a key is assigned to the key variable. The loop continues for all object properties.

```
Syntax -: for (key in object) {

// body of for...in

}
```

```
Ex -: const student = {

name: 'Monica',

class: 7,

age: 12

}

// using for...in Loop

for ( let key in student ) {

// display the properties

console.log(`${key} => ${student[key]}`);
```

```
}
```

4. For-of Loop :

You should not use for...in to iterate over an array where the index order is important. One of the better ways to iterate over an array is using the for...of loop .

The for...of loop was introduced in the later versions of JavaScript ES6 .

The for..of loop in JavaScript allows you to iterate over iterable objects (arrays, sets, maps, strings etc) .

```
Syntax -: for (element of iterable) {

// body of for...of

}
```

iterable - an iterable object (array, set, strings, etc) .

element - items in the iterable

```
Ex -: const students = ['John', 'Sara', 'Jack'];

// using for...of

for ( let element of students ) {

// display the values

console.log(element);

}
```

4. for-of Loop V/S for-in Loop :

---

for-of Loop	for-in Loop
1. The for...of loop is used to iterate through the values of an iterable .	1. The for...in loop is used to iterate through the keys of an object .
2. The for...of loop cannot be used to iterate over an object .	2. You can use for...in to iterate over an iterable such arrays and strings but you should avoid using for...in for iterables .

Var _ let _ const :

Var :

In Var we can do Declaration , Initialization , Declaration and Initialization , Re-Declaration , Re-Initialization , Re-Declaration and Re-Initialization

Let :

In Let we can do Declaration , Initialization , Declaration and Initialization , Re-Initialization but We Can not do Re-Declaration , Re-Declaration and Re-Initialization

Const :

In Const we can do Declaration and Initialization but we can not do Declaration , Initialization , Re-Declaration , Re-Initialization , Re-Declaration and Re-Initialization

Scope :

Scope in JavaScript refers to the accessibility of variables, functions, and objects in different parts of the code .

There are 5 types of scope:

1) Global Scope
2) Script Scope
3) Block Scope
4) Local Scope
5) Function Scope

Hoisting :

What is Hoisting ?

Hoisting is a part of javascript in which it move the code in to top of level of the file .

What is Function Hoisting ?

Moving the function top of the file called Function Hoisting

What is Variable Hoisting ?

Moving the variable top of the file called Variable Hoisting

JavaScript function :

A JavaScript function is a block of code designed to perform a particular task . A JavaScript function is executed when "something" invokes it (calls it) .

JavaScript Function Syntax :

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses () .

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables) .

The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...) .

The code to be executed, by the function, is placed inside curly brackets: {}

```
Ex -: function name(parameter1, parameter2, parameter3) {

// code to be executed

}
```

Function Invocation :

The code inside the function will execute when "something" invokes (calls) the function :

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

Function Return :

When JavaScript reaches a return statement, the function will stop executing.If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement .

Why Functions use ?

- With functions you can reuse code .
- You can write code that can be used many times .
- You can use the same code with different arguments, to produce different results .

Types of Functions in JavaScript :

Function devide in 8 types .

1. Anonymous function :

A function without name is known as Anonymous function .

```
Syntax -: function(parameters) {

// function body

}
```

2.Named Function :

A function with name is called as named function

```
Syntax -: function functionName(parameters) {

// function body

}
```

3. Function with expression :

It is the way to execute the anonymous function Passing whole anonymous function as a value to a variable is known as function with expression .

The function which is passed as a value is known as first class function .

```
Ex -: let x=function(){

//block of code

}

x();
```

4. Nested function :

A function which is declared inside another function is known as nested function . Nested functions are unidirectional i.e., We can access parent function properties in child function but vice-versa is not possible . The ability of js engine to search a variable in the outer scope when it is not available in the local scope is known as lexical scoping or scope chaining .

```
Ex -: Function parent(){

let a=10;

function child(){

let b=20;

console.log(a+b);

}

child ();

}

parent ();
```

What is a Closure?

A **closure** is a function that **remembers** the variables from its **outer function**, even after the outer function has **finished executing**.

Closures let you:

- Remember values
- Create private variables
- Maintain state between function calls

```
function outer() {

  let name = "JavaScript"; // <- this is the outer variable

  function inner() {

    console.log("Hello, " + name); // <- inner function uses it

  }
```

return inner;}

What happened here?

- outer() was called.
- It created a variable name and defined inner().
- outer() returned inner() — but inner() **remembers** name even after outer() is done!

const greet = outer(); // outer() runs, returns inner function

greet(); // Output: Hello, JavaScript

JavaScript currying :

Calling a child function along with parent by using one more parenthesis is known as java script currying .

```
Ex -: Function parent () {

let a=10;

function child () {

let b=20;

console.log(a+b);

}

return child;

}

parent () ();
```

5. Immediate invoking function(IIF) :

A function which is called immediately as soon as the function declaration is known as IIF We can invoke anonymous function by using IIF .

```
Ex -: (function () {

console.log("Hello");

})();
```

6. Arrow function :

The main purpose of using arrow function is to reduce the syntax .

It was introduced in ES6 version of JS .

If function has only one statement, then block is optional .

It is mandatory to use block if function consists of multiple lines of code or if we use return keyword in function .

```
Ex -: Let x= (a, b) =>console.log(a+b);

Let y=(a,b)=>{return a + b };
```

7. Higher order function(HOF) :

A function which accepting another function as as argument the accepting function called as higher order function or a function return another function is called HOF .

```
Ex -: function hof (a,b,task){

console.log(a,b);

var res = task(a,b)

console.log(res);

}

hof(10,20,(x,y)=>x+y)
```

8. Callback function :

A function which passed another function as an argument the passed function called Callback function

```
Ex -: function demo(x){

console.log(x);

}

const test = () =>{

console.log("callback")

}

demo(test)
```

String And Array Methods :

String -: Collection of characters (or) bunch of characters we called it as string .

Array -: Collection of values .

String Methods :

1. String.length() : Returns the length (number of characters) in a string .

```
Ex -: let str = "Hello, World!";

console.log(str.length); // Output: 13
```

2. String.slice(start, end) : Extracts a part of a string and returns it as a new string. The end index is not included .

```
Ex -: let str = "JavaScript";

console.log(str.slice(0, 4)); // Output: "Java"

console.log(str.slice(-6)); // Output: "Script"
```

3. String.substring(start, end) : Similar to slice(), but it does not support negative indices .

```
Ex -: let str = "JavaScript";

console.log(str.substring(0, 4)); // Output: "Java"

console.log(str.substring(4, 10)); // Output: "Script"
```

4. String.substr(start, length) : Extracts a part of a string starting from a specific index and taking a specified number of characters .

```
Ex -: let str = "JavaScript";

console.log(str.substr(4, 6)); // Output: "Script"
```

5. String.replace(searchValue, replaceValue) : Replaces the first occurrence of a substring with another string .

```
Ex -: let str = "I love JavaScript";

console.log(str.replace("JavaScript", "Python")); // Output: "I love Python"
```

6. String.replaceAll(searchValue, replaceValue) : Replaces all occurrences of a substring with another string .

```
Ex -: let str = "apple apple apple";

console.log(str.replaceAll("apple", "orange")); // Output: "orange orange orange"
```

7. String.toUpperCase() : Converts a string to uppercase .

```
Ex -: let str = "hello world";

console.log(str.toUpperCase()); // Output: "HELLO WORLD"
```

8. String.toLowerCase() : Converts a string to lowercase .

```
Ex -: let str = "HELLO WORLD";

console.log(str.toLowerCase()); // Output: "hello world"
```

9. String.concat(str1, str2, ...) : Joins two or more strings and returns a new string .

```
Ex -: let str1 = "Hello";

let str2 = "World";

console.log(str1.concat(", ", str2, "!")); // Output: "Hello, World!"
```

10. String.trim() : Removes whitespace from both sides of a string .

```
Ex -: let str = " Hello World! ";

console.log(str.trim()); // Output: "Hello World!"
```

11. String.trimStart() : Removes whitespace from the beginning of a string .

```
Ex -: let str = " Hello!";
```

```
console.log(str.trimStart()); // Output: "Hello!"
```

12. String.trimEnd() : Removes whitespace from the end of a string .

```
Ex -: let str = "Hello! ";

console.log(str.trimEnd()); // Output: "Hello!"
```

13. String.padStart(targetLength, padString) : Pads the start of a string with a specified character until it reaches the given length .

```
Ex -: let str = "5";

console.log(str.padStart(3, "0")); // Output: "005"
```

14. String.padEnd(targetLength, padString) : Pads the end of a string with a specified character until it reaches the given length .

```
Ex -: let str = "5";

console.log(str.padEnd(3, "0")); // Output: "500"
```

15. String.charAt(index) : Returns the character at the specified index .

```
Ex -: let str = "JavaScript";

console.log(str.charAt(4)); // Output: "S"
```

16. String.charCodeAt(index) : Returns the Unicode (ASCII) value of the character at the specified index .

```
Ex -: let str = "ABC";

console.log(str.charCodeAt(0)); // Output: 65 (ASCII value of 'A')
```

17. String.split(separator) : Splits a string into an array of substrings based on a separator .

```
Ex -: let str = "apple,banana,grape";

console.log(str.split(",")); // Output: ["apple", "banana", "grape"]
```

Array Methods :

1. push() : Adds an element to the end of an array .

```
Ex -: let arr = [1, 2, 3];
```

```
arr.push(4);

console.log(arr); // Output: [1, 2, 3, 4]
```

2. pop() : Removes the last element from an array and returns it .

```
Ex -: let arr = [1, 2, 3];

let removed = arr.pop();

console.log(arr); // Output: [1, 2]

console.log(removed); // Output: 3
```

3. unshift() : Adds an element to the beginning of an array .

```
Ex -: let arr = [1, 2, 3];

arr.unshift(0);

console.log(arr); // Output: [0, 1, 2, 3]
```

4. shift() : Removes the first element from an array and returns it .

```
Ex -: let arr = [1, 2, 3];

let removed = arr.shift();

console.log(arr); // Output: [2, 3]

console.log(removed); // Output: 1
```

5. indexOf() : Returns the index of the first occurrence of a specified element. Returns -1 if not found .

```
Ex -: let arr = ["apple", "banana", "cherry"];

console.log(arr.indexOf("banana")); // Output: 1

console.log(arr.indexOf("grape")); // Output: -1
```

6. includes() : Checks if an array contains a specific element. Returns true or false .

```
Ex -: let arr = ["apple", "banana", "cherry"];

console.log(arr.includes("banana")); // Output: true
```

```
console.log(arr.includes("grape")); // Output: false
```

7. at() : Returns the element at the specified index. Supports negative indexing .

```
Ex -: let arr = [10, 20, 30, 40];

console.log(arr.at(1)); // Output: 20

console.log(arr.at(-1)); // Output: 40 (last element)
```

8. slice(start, end) : Returns a shallow copy of a portion of an array. It does not modify the original array .

```
Ex -: let arr = [10, 20, 30, 40, 50];

console.log(arr.slice(1, 4)); // Output: [20, 30, 40]

console.log(arr); // Original array remains unchanged
```

9. splice(start, deleteCount, item1, item2, ...) : It will give slice of an array and it will not affect the original array .

```
Ex -: (Remove elements):

let arr = [10, 20, 30, 40];

arr.splice(1, 2); // Removes 2 elements starting from index 1

console.log(arr); // Output: [10, 40]

Ex -: (Add elements):

let arr = [10, 40];

arr.splice(1, 0, 20, 30); // Adds 20 and 30 at index 1

console.log(arr); // Output: [10, 20, 30, 40]
```

10. join(separator) : Converts all elements of an array into a string, separated by the specified separator .

```
Ex -: let arr = ["apple", "banana", "cherry"];

console.log(arr.join(", ")); // Output: "apple, banana, cherry"
```

11. concat() : Merges two or more arrays and returns a new array.

```
Ex -: let arr1 = [1, 2, 3];

let arr2 = [4, 5, 6];

let merged = arr1.concat(arr2);

console.log(merged); // Output: [1, 2, 3, 4, 5, 6]
```

12. toString() : It converts all the elements in an array into a single string and returns that string .

```
Ex -: let arr = [1, 2, 3, 4];

console.log(arr.toString()); // Output: "1,2,3,4"
```

13. split(separator) : Splits a string into an array based on a specified separator .

```
Ex -: let str = "apple,banana,cherry";

console.log(str.split(",")); // Output: ["apple", "banana", "cherry"]
```

14. reverse() : Reverses the order of elements in an array .

```
Ex -: let arr = [1, 2, 3, 4];

arr.reverse();

console.log(arr); // Output: [4, 3, 2, 1]
```

15. Array.from() : Converts an array-like object or a string into an array .

```
Ex -: let str = "Hello";

console.log(Array.from(str)); // Output: ["H", "e", "l", "l", "o"]
```

Spread operator and Rest parameter :

What is Spread operator ?

The spread operator helps us expand an iterable such as an array where multiple arguments are needed, it also helps to expand the object expressions .

Note: There can be more than one spread operator in javascript functions.

```
Syntax -: var var_name = [...iterable];

Ex1 -: var array1 = [10, 20, 30, 40, 50];
```

```
var array2 = [60, 70, 80, 90, 100];

var array3 = [...array1, ...array2];

console.log(array3); OUTPUT -: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Ex2 -: const obj = {

firstname: "Divit",

lastname: "Patidar",

};

const obj2 = { ...obj };

console.log(obj2);

OUTPUT -: {

firstname: "Divit",

lastname: "Patidar"

}
```

Note : There are three distinct places that accept the spread syntax :

- Function arguments list (myFunction(a, ...iterableObj, b))
- Array literals ([1, ...iterableObj, '4', 'five', 6])
- Object literals ({ ...obj, key: 'value' })

What is Rest parameter ?

The rest parameter is converse to the spread operator. while the spread operator expands elements of an iterable, the rest operator compresses them. It collects several elements. In functions when we require to pass arguments but were not sure how many we have to pass, the rest parameter makes it easier .

Note: There must be only one rest operator in javascript functions .

Note: The rest parameter has to be the last argument, as its job is to collect all the remaining arguments into an array .

```
Syntax -: function function_name(...arguments) {

statements;
```

```
}
```

Spread operator V/S Rest parameter :

Spread Operator	Rest Parameter
1. The spread operator is used to expand elements of an iterable (array, string, etc.)	1. The rest parameter is used to collect multiple elements and condense them into a single element .
2. The spread operator is used in function calls and array literals .	2. The rest parameter is used in function declarations to accept an indefinite number of arguments .
3. The spread operator is used to copy the elements of an iterable into another iterable .	3. The rest parameter is used to collect all the remaining arguments into an array .

JavaScript Promise :

To make javascript synchronous to asynchronous we use Promise. it has 3 stages :

Pending - The initial state, before the operation completes.

Fulfilled - The operation was successful, and a result is available.

Rejected - The operation failed, and an error is available.

A Promise takes a function with two arguments: resolve (success) and reject (failure).

To handle promise we use .then() and .catch()

Instead of .then() and .catch(), we can use async/await

```
Syntax -: let promise = new Promise(function(resolve, reject){

//do something

});
```

Promise then() Method:

It is invoked when a promise is either resolved or rejected. It may also be defined as a carrier that takes data from promise and further executes it successfully.

Promise catch() Method:

It is invoked when a promise is either rejected or some error has occurred in execution. It is used as an Error Handler whenever at any step there is a chance of getting an error.

Async and Await :

Async/await is a feature in JavaScript that allows you to work with asynchronous code in a more synchronous-like manner, making it easier to write and understand asynchronous code. Async Functions always return a promise. Await Keyword is used only in Async Functions to wait for promise.

Async Function :

The Async function simply allows us to write promises-based code as if it were synchronous and it checks that we are not breaking the execution thread. Async functions will always return a value. It makes sure that a promise is returned and if it is not returned then JavaScript automatically wraps it in a promise which is resolved with its value

```
Syntax -: async function myFunction() {

return "Hello";

}
```

Await Keyword :

Await is used to wait for the promise. It could be used within the async block only. It makes the code wait until the promise returns a result.

```
Syntax -: let value = await promise;
```

Time delays :

setTimeOut() :

Runs the code with time delay

cancel a setTimeout, you use clearTimeout.

```
Syntax -: window.setTimeOut( function , delay ) ;
```

function: The function you want to execute.

delay: The amount of time (in milliseconds) before the function is executed.

setInterval() :

The function similar to setTimeOut, but it schedules a function to be executed repeatedly at specified interval.

```
Syntax -: window.setInterval( function , delay ) ;
```

Dom :

What is Dom and its Methods ?

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page. (Dom is an API with the help of js we can able to access Dom) OR In simple words, it can be categorized as a programming interface for HTML as well as XML (eXtensible Markup Language) documents. It defines the logical structure of the documents in the form of a tree of objects where each HTML element is represented as a node and it also describes the way these documents can be manipulated. There are 5 Method to access

- HTML DOM getElementById()
- HTML DOM getElementsByClassName()
- HTML DOM getElementsByTagName()
- HTML DOM querySelector()
- HTML DOM querySelectorAll()

getElementById() :

This method is used when developers have defined certain HTML elements with IDs that uniquely identify the same elements in the whole document. It returns an Element object which matches the specified ID in the method. If the ID does not exist in the document, it simply returns null .

```
Syntax -: document.getElementById(id);
```

Parameter -: It have one single parameter that is ID .

- ID : The ID of the element to locate in the HTML document. It should be a case-sensitive string .

Return Value -: It returns the object corresponding to the passed ID in the method, or null if no match is found .

getElementsByClassName() :

This method is used when there are multiple HTML elements with the same class name. It returns a collection of all objects which match the specified class in the method .

```
Syntax -: document.getElementsByClassName(className);
```

Parameter -: It have one single parameter that is className .

- className : The class name of the element(s) to locate in the HTML document. It should be a case-sensitive string .

Return Value -: It returns a html collection and looks like an array but it is not an array if any element is not there it will return empty array .

getElementsByTagName() :

The getElementsByTagName() returns a HTMLCollection of objects which match a tag name in the HTML document .

```
Syntax -: document.getElementsByTagName(tagName);
```

Parameter -: It have one single parameter that is tagName .

- tagName : The tag name of the element(s) to locate in the HTML document. It should be a case-sensitive string .

Return Value -: It returns a html collection of objects corresponding to the passed tag name in the method if any element is not there it will return empty array .

querySelector() :

This method returns the first match of an element that is found within the HTML document with the specific selector. If there is no match, null is returned .

```
Syntax -: document.querySelector(selector);
```

Parameter -: It have one single parameter that is selector .

- selector : A string containing one or more selectors to match elements located in the HTML document .

Return Value -: It return only one node or first element from multiple element if no element is there it will return null .

querySelectorAll() :

This method returns a static NodeList of all elements that are found within the HTML document with the specific selector .

```
Syntax -: document.querySelectorAll(selector);
```

Parameter -: It have one single parameter that is selector .

- selector : A string containing one or more selectors to match elements located in the HTML document .

Return Value -: It returns a NodeList of the objects corresponding to the passed selector in the method if there is no node it will return empty nodelist .

DOM createElement() Method:

In an HTML document, the document.createElement() is a method used to create the HTML element .

```
Syntax -: let element = document.createElement("elementName")
```

In the above syntax, elementName is passed as a parameter in string form. elementName specifies the type of the created element. The nodeName of the created element is initialized to the elementName value. The document.createElement() returns the newly created element .

innerText() :

Returns the text content of an element and all its child elements, excluding HTML tags and CSS hidden text. innerText also ignores HTML tags and treats them as part of the text .

- Gets or sets only the visible text inside an element .
- Ignores HTML tags and returns plain text .
- Does not parse or execute any HTML .

```
Ex -: let div = document.createElement("div");

div.innerText = "Hello World!";

document.body.appendChild(div);

OutPut -: Hello World! (Displays as plain text, not bold)
```

innerHTML() :

Returns the text content of an element, including all spacing and inner HTML tags. innerHTML recognizes HTML tags and renders the content according to the tags. innerHTML allows you to see exactly what's in the HTML markup contained within a string, including elements like spacing, line breaks, and formatting .

- Gets or sets the HTML content inside an element .
- Parses and renders HTML tags .
- Can inject HTML code inside an element .

```
Ex -: let div = document.createElement("div");

div.innerHTML = "Hello World!";

document.body.appendChild(div);

OutPut -: Hello World! (Displays "Hello" in bold)
```

Key Differences :

Property	Handles HTML	Parses & Executes HTML	Returns Hidden Text
innerText	✗ No	✗ No	✗ No
innerHTML	✓ Yes	✓ Yes	✓ Yes

Methods :

appendchild() :

The appendChild() method adds a new child element to a parent element in the DOM. It places the child at the end of the parent's existing content . it can append one node at a time .

```
Syntax -: parentElement.appendChild(childElement);

Ex -: let parent = document.getElementById("container"); // Get parent element

let newDiv = document.createElement("div"); // Create a new <div>

newDiv.textContent = "Hello, World!"; // Add text content

parent.appendChild(newDiv); // Add <div> to the parent
```

append() :

The Element.append() method inserts a set of Node objects or string objects after the last child of the Element

```
Syntax -: parentElement.append(child1, child2, ...);
```

difference betweeen append() and appendChild() :

Feature	append()	appendChild()
Adds text?	✓ Yes	✗ No
Adds multiple elements?	✓ Yes	✗ No(only one)
Returns a value?	✗ No	✓ Yes (appended node)

---

removeChild() :

The removeChild() method of JavaScript will remove the element from the document . Removes a specific child element from a parent and Requires the parent element to call it and Returns the removed element .

```
Syntax -: parent.removeChild(child);
```

```
Ex -: let parent = document.getElementById("container");

let child = document.getElementById("childElement");

parent.removeChild(child); // Removes the child from the parent
```

remove() :

Remove method removes the element itself without needing the parent and it does not return anything and More modern and easier than removeChild() .

```
Syntax -: element.remove();
```

```
Ex -: let element = document.getElementById("childElement");

element.remove(); // Removes itself from the DOM
```

Differences betweeen remove and removeChild :

Feature	removeChild()	remove()

Requires Parent	✓ Yes	✗ No (removes itself)
Returns Element	✓ Yes	✗ No
Simplicity	✗ More complex	✓ Easier & modern

Attribute method :

setAttribute() :

The setAttribute() method is used to set or add an attribute to a particular element and provides a value to it. If the attribute already exists, it only set or changes the value of the attribute. So, we can also use the setAttribute() method to update the existing attribute's value. If the corresponding attribute does not exist, it will create a new attribute with the specified name and value. This method does not return any value. The attribute name automatically converts into lowercase when we use it on an HTML element . OR (To get the value of an attribute, we can use the getAttribute() method)

```
Syntax -: element.setAttribute(attributeName, attributeValue);
```

```
Ex -: let div = document.createElement("div");

div.setAttribute("id", "myDiv"); // Sets id="myDiv"

document.body.appendChild(div);
```

attributeName : It is the name of the attribute that we want to add to an element. It cannot be left empty; i.e., it is not optional .

attributeValue : It is the value of the attribute that we are adding to an element. It is also not an optional value .

removeAttribute() :

The removeAttribute() method removes the attribute with the specified name . ✓ Removes only one attribute at a time . Does not remove the element itself, only the attribute.

```
Syntax -: element.removeAttribute(attributeName);
```

Parameter Values = attributename : It is the required parameter that specifies the attribute's name to remove from the element. If the attribute doesn't exist, the method doesn't create any error .

```
Ex -: let div = document.getElementById("myDiv");
```

```
div.removeAttribute("id"); // Removes the id attribute
```

getAttribute() :

The getAttribute() method is used to get the value of an attribute of the particular element. If the attribute exists, it returns the string representing the value of the corresponding attribute. If the corresponding attribute does not exist, it will return an empty string or null .

```
Syntax -: element.getAttribute(attributeName);
```

attributename : It is the required parameter. It is the name of the attribute we want to get the value from .

Event Handling :

addEventListener() :

The addEventListener() method in JavaScript is used to attach an event handler to an element. it accept 2 arguments which event you want to trigger and then after trigger what functionality you need to do .

Event : An action performed by the user on the webpage is known as an event

```
Syntax -: element.addEventListener(event, listener, useCapture);
```

event : It is a string that specifies the name of the event . events are Click , Mouseover , Mouseout , Doubleclick , Submit

listener : It is a function that specifies the function to run when the event occurs .

useCapture : It is an optional parameter that specifies whether the event should be executed in the capturing or bubbling phase .

```
Ex -: let button = document.getElementById("myButton");

button.addEventListener("click", function() {

console.log("Button clicked!");

});
```

Event Bubbling :

what is Event bubbling ?

Event bubbling is the phase where the event bubbles up from the target element all the way to the global window object .

How event bubbling works ?

When an event occurs on a child element, it first triggers on that element and then bubbles up to its ancestors in the DOM hierarchy. This means the event starts at the targeted element and moves upward to the parent, grandparent, and so on, until it reaches the root (document).

Event Bubbling: Child → Parent → Grandparent

Bubbling phase :

The bubbling phase, which is the last phase, is the reverse of the capturing phase. In this phase, the event bubbles up the target element through its parent element, the ancestor, to the global window object. By default, all events you add with addEventListener are in the bubble phase .

Event Capturing :

Event capturing occurs when a nested element gets clicked. The click event of its parent elements must be triggered before the click of the nested element. This phase trickles down from the top of the DOM tree to the target element .

Event capturing can't happen until the third argument of addEventListener is set to the Boolean value of true .

Whenever the third argument of addEventListener is set to true, event handlers are automatically triggered in the capturing phase. With this, the event handler attached to an ancestor element will be executed first when an event occurs on a nested element within the DOM hierarchy .

Capturing phase :

The first phase is the capturing phase, which occurs when an element nested in various elements gets clicked. Right before the click reaches its final destination, the click event of each of its parent elements must be triggered. This phase trickles down from the top of the DOM tree to the target element .

Target phase :

The target phase is the second phase that begins immediately after the capturing phase ends. This phase is basically the end of the capturing and the beginning of the bubbling phase .