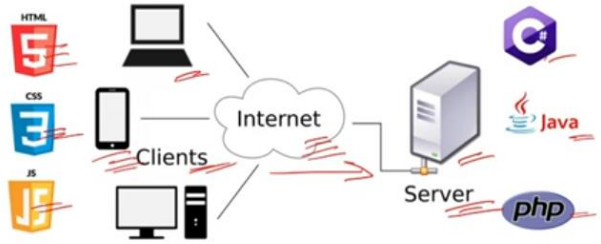
JS

Q. What is JavaScript? What is the role of JavaScript engine? V. IMP.

- JavaScript is a programming language that is used for converting static web pages to **interactive and dynamic** web pages.
- A JavaScript engine is a program present in web browsers that executes JavaScript code.

Q. What are Client side and Server side? V. IMP.

- A client is a device, application, or software component that **requests** and consumes services or resources from a server.
- A server is a device, computer, or software application that **provides** services, resources, or functions to clients.



Q. What are variables? What is the difference between var, let, and const ? V. IMP.

- var creates a **function-scoped** variable.
- let creates a **block-scoped** variable
- const can be assigned only once, and its value **cannot be changed** afterwards.

```
//using var
function example() {
  if (true) {
    var count = 10;
    console.log(count);
    //output: 10
  }
  console.log(count);
  //Output: 10
}
```

```
//using let
function example() {
  if (true) {
    let count = 10;
    console.log(count);
    //Output: 10
  }
  console.log(count);
  //Output: Uncaught
  //Reference Error:
  //count is not defined
}
```

```
// Using constant
const z = 10;
z = 20;

// This will result
//in an error
console.log(z);
```

Q. What are some important string operations in JS?

javaScript String Methods JS

substr()	indexOf()	trim()
substring()	includes()	charAt()
replace()	slice()	valueOf()
search()	concat()	split()
toLocaleLowerCase()	lastIndexOf()	toString()
toLocaleUpperCase()	charCodeAt()	match()

Q. What are some important string operations in JS?

```
//Add multiple string
let str1 = "Hello";
let str2 = "World";
let result = str1 + " " + str2;
console.log(result);
// Output: Hello World
```

```
// Using concat() method
let result2 = str1.concat(" ", str2);
console.log(result2);
// Output: Hello World
```

```
//Extract a portion of a string
let subString = result.substring(6, 11);
console.log(subString);
// Output: World
```

```
//Retrieve the length of a string
console.log(result.length);
// Output: 11
```

```
//Convert a string to uppercase or lowercase
console.log(result.toUpperCase());
// Output: HELLO WORLD
console.log(result.toLowerCase());
// Output: hello world
```

```
//Split a string into an array of substrings
//based on a delimiter
let arr = result.split(" ");
console.log(arr);
// Output: ["Hello", "World"]
```

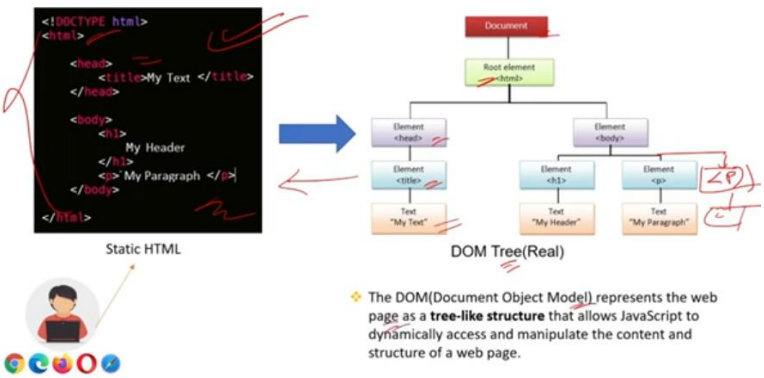
```
//Replace occurrences of a substring within a string
console.log(result.replace("World", "JavaScript"));
// Output: Hello JavaScript
```

```
//Remove leading and trailing whitespace
let str = "   Hello World   ";
let trimmedStr = str.trim();
console.log(trimmedStr);
// Output: Hello World
```

Q. What is DOM? What is the difference between HTML and DOM? V. IMP.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>My Text </title>
    </head>
    <body>
        <h1>
            My Header
        </h1>
        <p> My Paragraph </p>
    </body>
</html>
```

Static HTML



DOM Tree(Real)

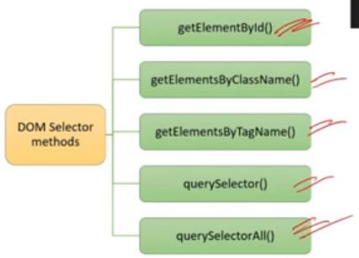
- The DOM(Document Object Model) represents the web page as a **tree-like structure** that allows JavaScript to dynamically access and manipulate the content and structure of a web page.

Q. What are selectors in JS? V. IMP.

- Selectors in JS help to **get specific elements from DOM** based on IDs, class names, tag names.

```html
<div id="myDiv">Test</div>
```

```js
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
```



Q. What is the difference between getElementById, getElementsByClassName and getElementsByTagName? V. IMP.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Methods</title>
  </head>
  <body>
    <div id="myDiv" class="myClass">1</div>
    <div class="myClass">2</div>
    <p class="myClass">3</p>

    <script src="index.js"></script>
  </body>
</html>
```

```js
//getElementById - select a single element
const elementById = document.getElementById("myDiv");
console.log(elementById.innerHTML);
// Output: 1
```

```js
//getElementsByClassName - select multiple elements that
                         //share the same class name
const elements = document.getElementsByClassName("myClass");

for (let i = 0; i < elements.length; i++) {
  console.log(elements[i].textContent);
}
// Output: 1 2 3
```

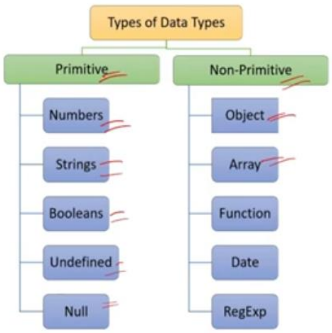
```js
//getElementsByTagName - select multiple elements based
                       //on their tag name
const elementsTag = document.getElementsByTagName("div");

for (let i = 0; i < elementsTag.length; i++) {
  console.log(elementsTag[i].textContent);
}
// Output: 1 2
```

Q. What are data types in JS?

- A data type determines the **type of variable.**

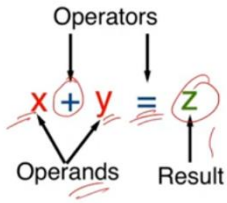
```js
//Number
let age = 25;
```

```js
//String
let message = 'Hello!';

//Boolean
let isTrue = true;

//Undefined
let x;
console.log(x);
// Output: undefined

//Null
let y = null;
console.log(y);
// Output: null
```

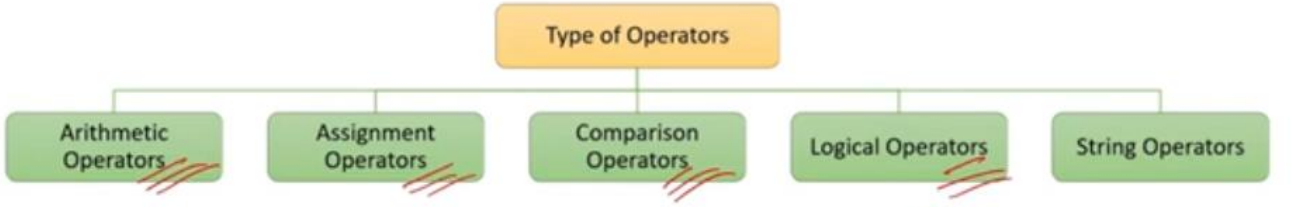


Q. What are operators? What are the types of operators in JS? V. IMP.

- Operators are **symbols or keywords** used to perform operations on operands.



Q. What are operators? What are the types of operators in JS? V. IMP.




```js
let x = 5;
let y = 2;

console.log(x + y);
console.log(x - y);
console.log(x * y);
console.log(x / y);
console.log(x % y);
//Remainder: 1
console.log(x ** y);
//Exponentiation: 25
```

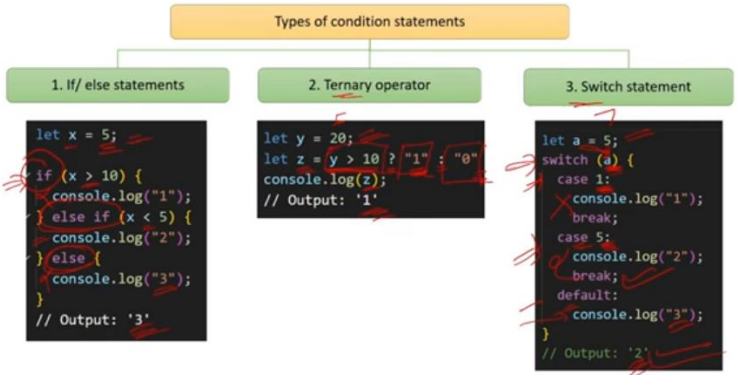
```js
let x = 10;

x += 5;
//x = x + 5
console.log(x);
// Output: 15

x *= 2;
//x = x * 2
console.log(x);
// Output: 30
```

```js
let x = 5;
let y = 3;

console.log(x > y);
console.log(x < y);
console.log(x >= y);
console.log(x <= y);
console.log(x === y);
// Equality: false
console.log(x !== y);
// Inequality: true
```

```js
let x = true;
let y = false;

console.log(x && y);
//Logical AND: false

console.log(x || y);
//Logical OR: true

console.log(!x);
//Logical NOT: false
```

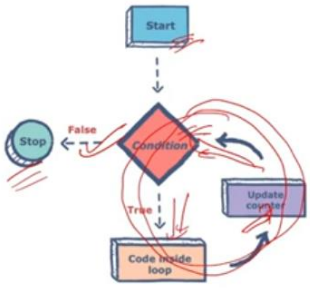
```js
let a = 'Hello ';
let b = 'World';

var c =(a + b);
// Concatenation
// "Hello World"
```

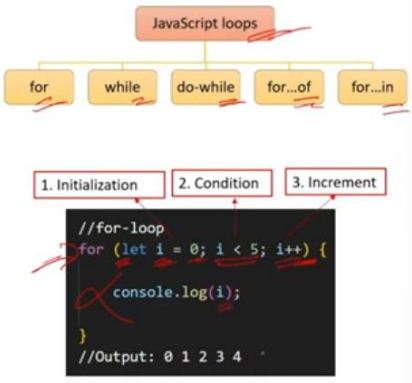
Q. What are the types of conditions statements in JS? V. IMP.



Q. What is a loop? What are the types of loops in JS? V. IMP.

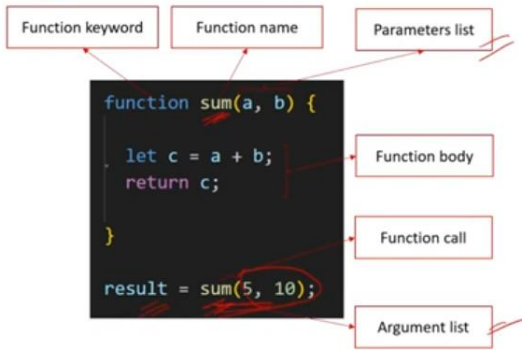
- A loop is a programming way to run a piece of **code repeatedly** until a certain condition is met.



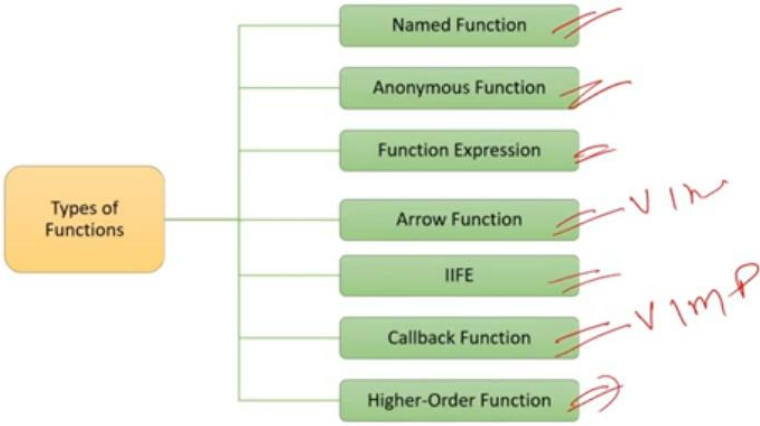


Q. What are Functions in JS? What are the types of function? V. IMP.

- A function is a **reusable block of code** that performs a specific task.

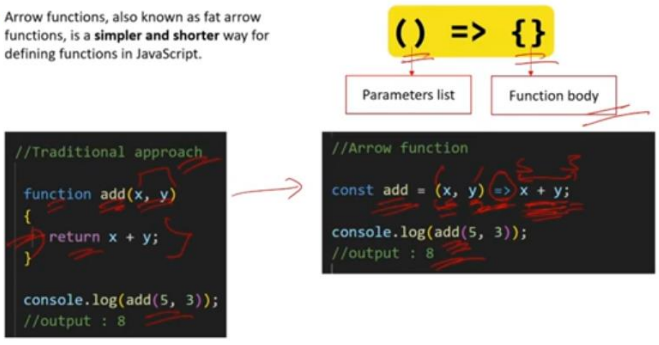


Q. What are Functions in JS? What are the types of function? V. IMP.



Q. What are Arrow Functions in JS? What is it use? V. IMP.

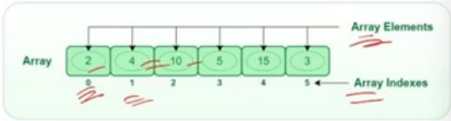
- Arrow functions, also known as fat arrow functions, is a **simpler and shorter** way for defining functions in JavaScript.

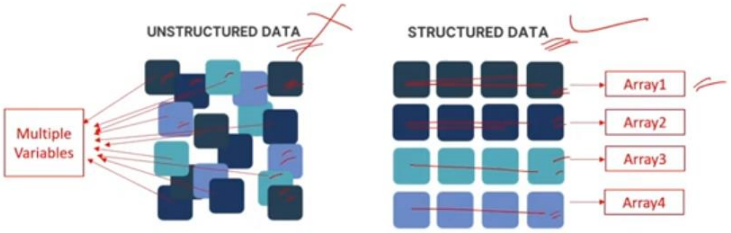


Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.

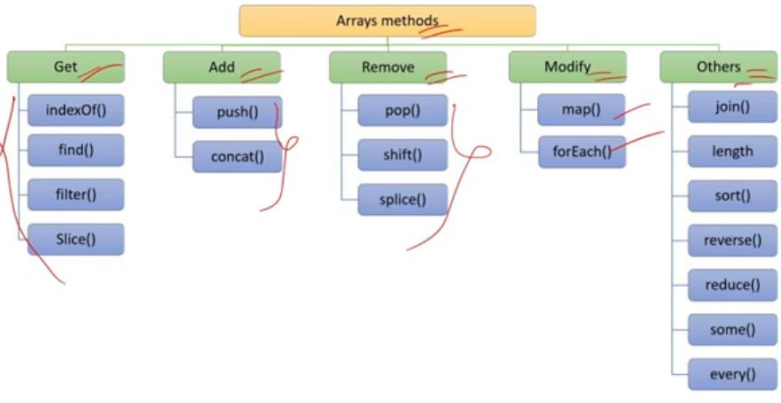
- An array is a data type that allows you to **store multiple values** in a single variable.

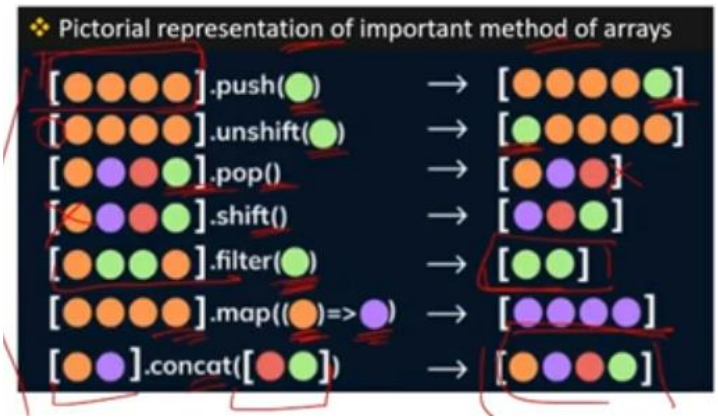
```
//Array
let fruits = ["apple", "banana", "orange"];
```





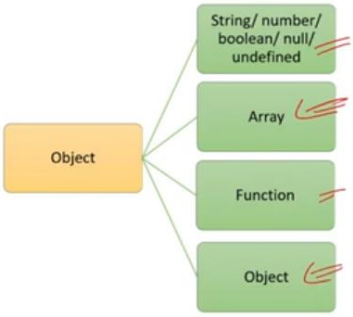
Q. What are Arrays in JS? How to get, add & remove elements from arrays? V. IMP.





Q. What are Objects in JS? V. IMP.

- An object is a data type that allows you to store **key-value** pairs.




```
//Object Example
let person = {
  name: "Happy",
  hobbies: ["Teaching", "Football", "Coding"],
  greet: function () {
    console.log("Name: " + this.name);
  },
};

console.log(person.name);
// Output: "Happy"

console.log(person.hobbies[1]);
// Output: "Football"

person.greet();
// Output: "Name: Happy"
```

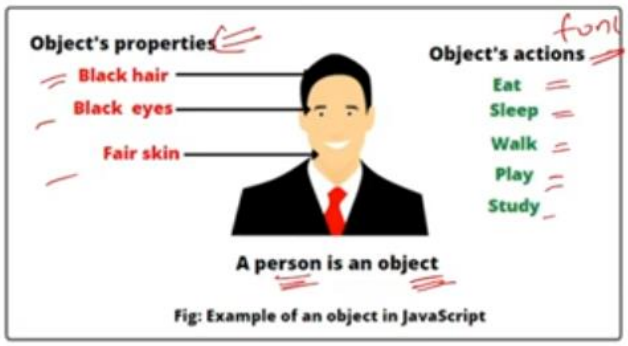


Fig: Example of an object in JavaScript

Q. What is Scope in JavaScript? V. IMP.

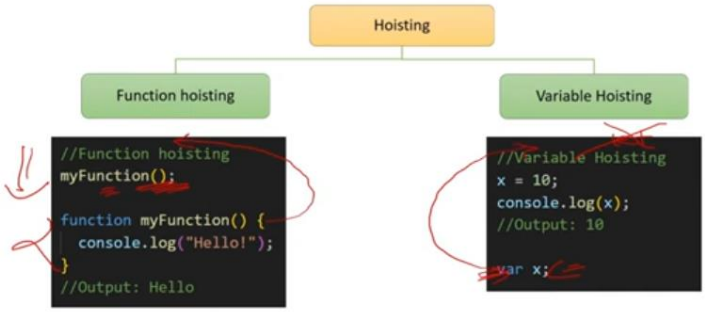
- Scope determines where variables are **defined** and where they can be **accessed**.

Global | Function | Block

```
//Global - accessbile anywhere
let globalVariable = "global";

greet();

function greet() {
  //Function - accessbile inside function only
  let functionVariable = "function";
  if (true) {
    //Block - accessbile inside block only
    let blockVariables = "block";
    console.log(blockVariables); //Output: block
    console.log(functionVariable); //Output: function
    console.log(globalVariable); //Output: global
  }
  console.log(functionVariable); //Output: function
  console.log(globalVariable); //Output: global
}
console.log(globalVariable); //Output: global
```

Q. What is Hoisting in JavaScript? V. IMP.

- Hoisting is a JavaScript behavior where functions and variable **declarations are moved to the top** of their respective scopes during the compilation phase.

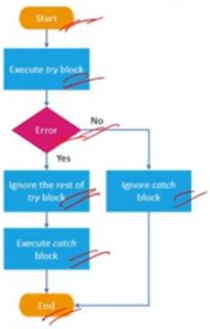



```
//Function hoisting
myFunction();

function myFunction() {
  console.log("Hello!");
}
//Output: Hello
```

```
//Variable Hoisting
x = 10;
console.log(x);
//Output: 10

var x;
```

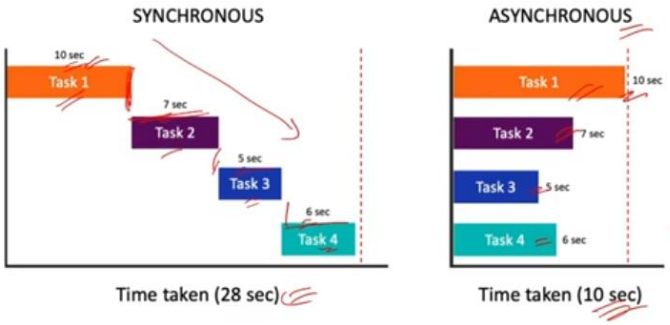
Q. What is Error Handling in JS? V. IMP.

- Error handling is the process of **managing errors**.



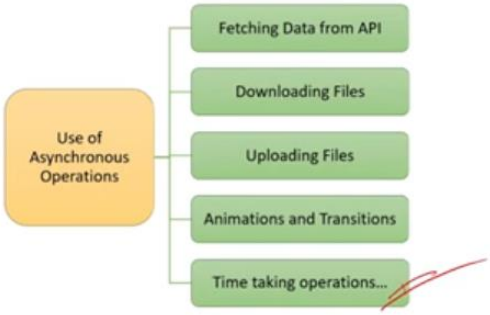

```
//try block contains the code that might throw an error
try {
  const result = someUndefinedVariable + 10;
  console.log(result);
}
//catch block is where the error is handled
catch (error) {
  console.log('An error occurred:', error.message);
}
//Output
//An error occurred: someUndefinedVariable is not defined
```

Q. What is asynchronous programming in JS? What is its use? V. IMP.



Q. What is asynchronous programming in JS? What is its use? V. IMP.

- Asynchronous programming allows multiple tasks or operations to be initiated and **executed concurrently**.
- Asynchronous operations **do not block** the execution of the code.

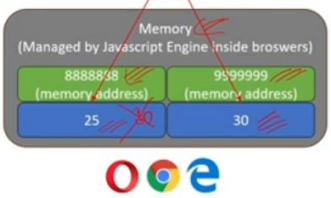



```
// Synchronous Programming
// Not efficient
console.log("Start");
Function1();
Function2();
console.log("End");

// Time taking function
function Function1() {
  // Loading Data from an API
  // Uploading Files
  // Animations
}
function Function2() {
  console.log(100 + 50);
}
```

Q. What is the difference between primitive and non-primitive data types? V. IMP.

- Primitive data types can hold only **single** value.
- Primitive data types are **immutable**, meaning their values, once assigned, cannot be changed.

```
//Number
let age = 25;
    age = 30;
```



- Non primitive data types can hold **multiple** value.
- They are mutable and their values can be changed.

```
//Non primitive data types

//Array
let oddNumbers = [1, 3, 5]

//Object
let person = {
  name: "John",
  age: 30,
  grades: ["A","B","C"],
  greet: function() {
  console.log(this.name);
  }
};
```

Q. What is the difference between primitive and non-primitive data types? V. IMP.

Primitive Data Types	Non-primitive Data Types
1. Number, string, Boolean, undefined, null are primitive data types.	Object, array, function, date, RegExp are non-primitive data types.
2. Primitive data types can hold only **single** value.	Non-primitive data types can hold **multiple** values and methods.
3. Primitive data types are **immutable** and their values cannot be changed.	Non-primitive data types are **mutable** and their values can be changed.
4. Primitive data types are **simple** data types.	Non-primitive data types are **complex** data types.

Q. What is the difference between null and undefined in JS?

```
let value1 = 0;

let value2 = '';
```

- (A stand on the wall with also a paper holder) Means there is a **valid variable** with also a value of **data type number**.

```
let value3 = null;
```

- (There is just a stand on the wall) Means there is a **valid variable** with a value of **no data type**.

```
let value4;
```

- (There is nothing on the wall) Means variable is **incomplete variable** and not assigned anything.

Q. What is the difference between null and undefined in JS?

```
let undefinedVariable; //no value assigned
console.log(undefinedVariable);
// Output: undefined
```

- undefined: When a variable is declared but has **not been assigned a value**, it is automatically initialized with undefined.
- Undefined can be used when you don't have the value right now, but you will get it after some logic or operation.

```
let nullVariable = null; //null assigned
console.log(nullVariable);
// Output: null
```

- null: null variables are intentionally assigned the **null value**.
- Null can be used, when you are sure you do not have any value for the particular variable.

Q. What is the use of typeof operator?

- typeof operator is used to determine the **type** of each variable.
- Real application use -> typeOf operator can be used to **validate the data** received from external sources(api).

```
let num = 42;
let str = "Hello, world!";
let bool = true;
let obj = { key: "value" };
let arr = [1, 2, 3];
let func = function() {};
```

```
//using typeof
console.log(typeof num); // Output: "number"
console.log(typeof str); // Output: "string"
console.log(typeof bool); // Output: "boolean"
console.log(typeof obj); // Output: "object"
console.log(typeof arr); // Output: "object"
console.log(typeof func); // Output: "function"
console.log(typeof undefinedVariable);
// Output: "undefined"
```

Q. What is type coercion in JS?

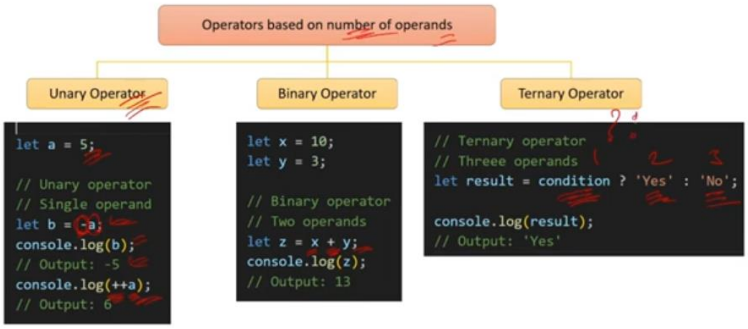
- Type coercion is the automatic conversion of values from one data type to another during certain operations or comparisons.
- Uses of type coercion:

1. Type coercion can be used during **String and Number concatenation.**
2. Type coercion can be used while using **Comparison operators**.

```
let string = "42";
let number = 42;
let boolean = true;
let nullValue = null;
```

```
//Type coercion - automatic conversion
console.log(string + number); // Output: "4242"

console.log(number + boolean); // Output: 43

console.log(number == string); // Output: true
console.log(boolean == 1); // Output: true
console.log(boolean + nullValue); // Output: 1
```

Q. What is the difference between unary, binary, and ternary operators?



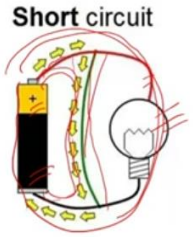

```
let a = 5;

// Unary operator
// Single operand
let b = -a;
console.log(b);
// Output: -5
console.log(++a);
// Output: 6
```

```
let x = 10;
let y = 3;

// Binary operator
// Two operands
let z = x + y;
console.log(z);
// Output: 13
```

```
// Ternary operator
// Threee operands
let result = condition ? 'Yes' : 'No';

console.log(result);
// Output: 'Yes'
```

Q. What is short-circuit evaluation in JS?

- Short-circuit evaluation stops the execution as soon as the **result can be determined without evaluating** the remaining sub-expressions.




```
// Short-circuit evaluation with logical AND
let result1 = false && someFunction();
console.log(result1);
// Output: false
```

```
// Short-circuit evaluation with logical OR
let result2 = true || someFunction();
console.log(result2);
// Output: true
```

Q. When to use which type of conditions statements in real applications? V. IMP.

- If...else : for complex, different & multiline execution.
- Benefit: Cover all scenarios.

```
const age = 25;
const height = 6

if (age < 25 && height < 5) {
  console.log("You are a minor.");
  console.log("You are a short.");
} else if (age >= 18 && height > 6) {
  console.log("You are an adult.");
  console.log("You are an tall.");
} else {
  console.log("You are average");
}
// Output: "You are average"
```

- Ternary operators : for simple conditions & single value evaluations.
- Benefit: Short one line syntax.

```
const isUser = true;

const user = isUser ? 10 : 20;

console.log(user);
// Output: "10"
```

- Switch case: For same left side values.
- Benefit: More structured code.

```
const dayOfWeek = "Tuesday";

switch (dayOfWeek) {
  case "Monday":
    console.log("Start ");
    break;
  case "Tuesday":
  case "Sunday":
    console.log("Weekend!");
    break;
  default:
    console.log("Invalid");
}
// Output: "Weekend!"
```

Q. What is operator precedence?

- As per operator precedence, operators with higher precedence are **evaluated first.**


```
let a = 6;
let b = 3;
let c = 2;

//BracketOf-Division-Multiplication-Add-Sub
let result = a + b * c + (a - b);

console.log(result);
// Output: 15
```

Q. What is the difference between == and ===? V. IMP.

```
//Loose Equality
console.log(1 == '1');
console.log(true == 1);
// Output: true
```

- Loose Equality (==) operator compares two values for equality after performing **type coercion**

```
//Strict Equality
console.log(1 === '1');
console.log(true === 1);
// Output: false
```

- Strict Equality (===) operator compares two values for equality **without** performing type coercion.

- Normally **=== is preferred** in use to get more accurate comparisons.

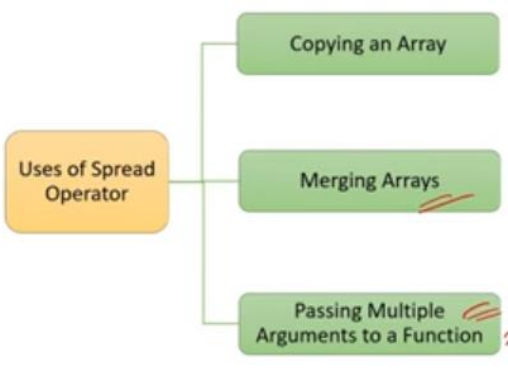
Q. What is the difference between Spread and Rest operator in JS?

- The rest operator is used in function parameters to collect all **remaining arguments** into an array.

```
// Rest Operator Example
display(1, 2, 3, 4, 5);

function display(first, second, ...restArguments) {
  console.log(first); // Output: 1
  console.log(second); // Output: 2

  console.log(remaining); // Output: [3, 4, 5]
}
```

Q. What is the difference between Spread and Rest operator in JS?

- The spread operator(...) is used to **expand or spread elements** from an iterable (such as an array, string, or object) into individual elements.




```
// Spread Operator Examples
const array = [1, 2, 3];
console.log(...array); // Output: 1, 2, 3
```

```
// Copying an array
const originalArray = [1, 2, 3];
const copiedArray = [...originalArray];
console.log(copiedArray); // Output: [1, 2, 3]
```

```
// Merging arrays
const array1 = [1, 2, 3];
const array2 = [4, 5];
const mergedArray = [...array1, ...array2];
console.log(mergedArray); // Output: [1, 2, 3, 4, 5]
```

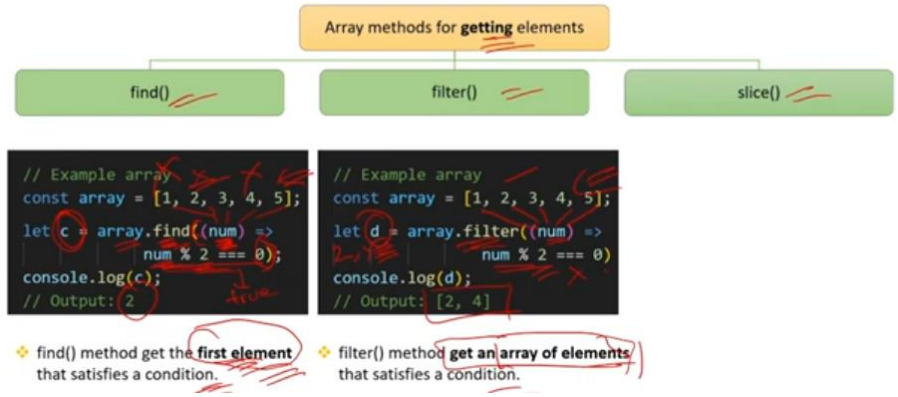
```
// Passing multiple arguments to a function
const numbers = [1, 2, 3, 4, 5];
sum(...numbers);
function sum(a, b, c, d, e) {
  console.log(a + b + c + d + e); //Output: 15
}
```

Q. What is the indexOf() method of an Array?

- IndexOf() method **gets the index** of a specified element in the array.

```
// Example array
const array = [1, 2, 3, 4, 5];

let a = array.indexOf(3);
console.log(a);
// Output: 2
```

Q. What is the difference between find() and filter() methods of an Array? V. IMP.




```
// Example array
const array = [1, 2, 3, 4, 5];
let c = array.find((num) =>
          num % 2 === 0);
console.log(c);
// Output: 2
```

- find() method get the **first element** that satisfies a condition.

```
// Example array
const array = [1, 2, 3, 4, 5];
let d = array.filter((num) =>
          num % 2 === 0)
console.log(d);
// Output: [2, 4]
```

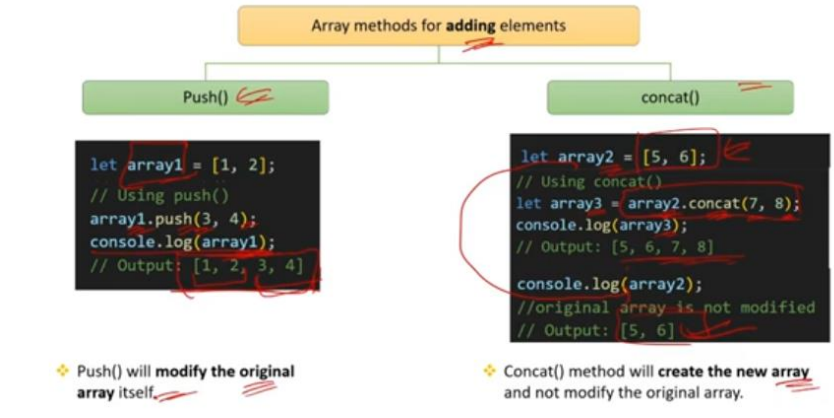
- filter() method **get an array of elements** that satisfies a condition.

slice()

```
const array = ["a", "b", "c", "d", "e"];

let e = array.slice(1, 4);
console.log(e);

// Output: ['b', 'c', 'd']
```

- Slice() method get a **subset of the array** from start index to end index(end not included).

Q. What is the difference between push() and concat() methods of an Array?



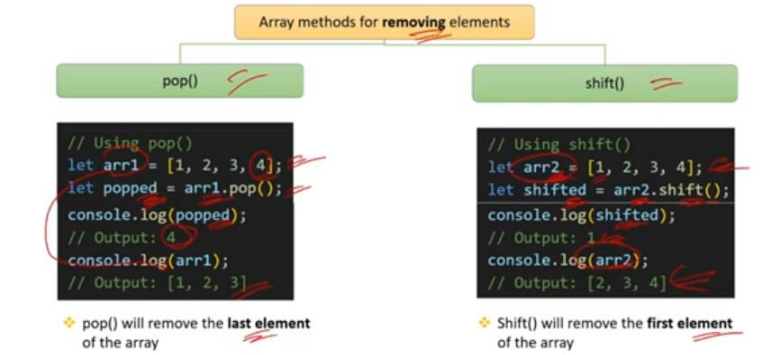

```
let array1 = [1, 2];
// Using push()
array1.push(3, 4);
console.log(array1);
// Output: [1, 2, 3, 4]
```

- Push() will **modify the original array** itself.

```
let array2 = [5, 6];
// Using concat()
let array3 = array2.concat(7, 8);
console.log(array3);
// Output: [5, 6, 7, 8]

console.log(array2);
//original array is not modified
// Output: [5, 6]
```

- Concat() method will **create the new array** and not modify the original array.

Q. What is the difference between pop() and shift() methods of an Array?




```
// Using pop()
let arr1 = [1, 2, 3, 4];
let popped = arr1.pop();
console.log(popped);
// Output: 4
console.log(arr1);
// Output: [1, 2, 3]
```

- pop() will remove the **last element** of the array

```
// Using shift()
let arr2 = [1, 2, 3, 4];
let shifted = arr2.shift();
console.log(shifted);
// Output: 1
console.log(arr2);
// Output: [2, 3, 4]
```

- Shift() will remove the **first element** of the array

Q. What is the splice() method of an Array? V. IMP.

- The splice() method is used to **add, remove, or replace** elements in an array.

```
array.splice(startIndex, deleteCount, ...itemsToAdd);
```

```
let letters = ['a', 'b', 'c'];

// Add 'x' and 'y' at index 1
letters.splice(1, 0, 'x', 'y');
console.log(letters);
// Output: ['a', 'x', 'y', 'b', 'c']

// Removes 1 element starting from index 1
letters.splice(1, 1);
console.log(letters);
// Output: ['a', 'y', 'b', 'c']

// Replaces the element at index 2 with 'q'
letters.splice(2, 1, 'q');
console.log(letters);
// Output: ['a', 'y', 'q', 'c']
```

Q. What is the difference between the slice() and splice() methods of an Array?

- The slice() method is used get a subset of the array from the start index to the end index(end not included).
- The splice() method is used to **add, remove, or replace** elements in an array.

Q. What is the difference map() and forEach() array methods of an Array?


```
// Using map()
let arr1 = [1, 2, 3];
let mapArray = arr1.map((e) => e * 2);
console.log(mapArray);
//map return a new array
// Output: [2, 4, 6]
```

- The map() method is used when you want to modify each element of an array and create a **new array** with the modified values.

```
// Using forEach()
let arr2 = [1, 2, 3];
arr2.forEach((e) => {
  console.log(e * 2);
});
//Does not return anything
// Output: 2 4 6
console.log(arr2);
// Output: [1, 2, 3]
```

- The forEach() method is used when you want to perform some operation on each element of an array **without creating a new array**.

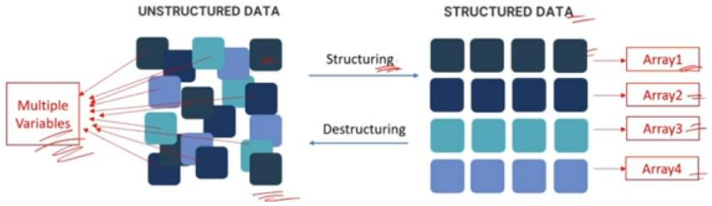
Q. How to sort and reverse an array?

- Array can be sorted or reversed by using **sort()** and **reverse()** methods of array.

```
let array = ['c', 'e', 'a', 't'];

//sort the array
array.sort();
console.log(array);
// Output: ['a', 'c', 'e', 't']

//reverse the array
array.reverse();
console.log(array);
// Output: ['t', 'e', 'c', 'a']
```

Q. What is Array Destructuring in JS? V. IMP.

- Array destructuring allows you to extract elements from an array and assign them to **individual variables** in a single statement.
- Array destructuring is introduced in **ECMAScript 6 (ES6)**.



Q. What is Array Destructuring in JS? V. IMP.

```
// Example array
const fruits = ['apple', 'banana', 'orange'];

// Array destructuring
const [firstFruit, secondFruit, thirdFruit] = fruits;

// Output
console.log(firstFruit); // Output: "apple"
console.log(secondFruit); // Output: "banana"
console.log(thirdFruit); // Output: "orange"
```

Q. What are array-like objects In JS? V. IMP.

- Array-like objects are objects that have indexed elements and a length property, **similar to arrays**, but they may not have all the methods of arrays like push(), pop() & others.

Types of array like object:

- arguments

```
sum(1, 2, 3);
//Arguments Object
function sum() {
  console.log(arguments); // Output: [1, 2, 3]
  console.log(arguments.length); // Output: 3
  console.log(arguments[0]); // Output: 1
}
```

- Strings

```
//String
const str = "Hello";
console.log(str); // Output: Hello
console.log(str.length); // Output: 5
console.log(str[0]); // Output: H
```

- HTML collections

```
// Accessing HTML collection
var boxes = document.getElementsByClassName('box');
// Accessing elements in HTML collection using index
console.log(boxes[0]);
// Accessing length property of HTML collection
console.log(boxes.length);
```

Q. How to convert an array-like object into an array?

Methods to convert an array-like object into an array:

- Array.from()

```
// Example array-like object
var arrayLike = {0: 'a', 1: 'b', 2: 'c', length: 3};

// Using Array.from()
var array1 = Array.from(arrayLike);
console.log(array1);
// Output: ['a', 'b', 'c']
```

- Spread Syntax (...)

```
// Using spread syntax (...)
var array2 = [...arrayLike];
console.log(array2);
// Output: ['a', 'b', 'c']
```

- Array.prototype.slice.call()

```
// Using Array.prototype.slice.call()
var array3 = Array.prototype.slice.call(arrayLike);
console.log(array3);
// Output: ['a', 'b', 'c']
```

Q. What is the difference between while and for loops?

- For loop allows to iterate a block of code a **specific number** of times.
- for loop is better for condition with initialization and with increment because all can be set in just **one line of code**.

```
// for loop
for (let i = 0; i < 5; i++) {

console.log(i);

}
// Output: 0 1 2 3 4
```

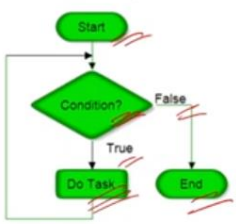
- While loop execute a block of code while a certain **condition** is true.
- While loop is better when there is **only condition**, no initialization, no increment.

```
// while loop
let j = 0;

while (j < 5) {

  console.log(j);
  j++;

}
// Output: 0 1 2 3 4
```

```
// condition only
while ("a" == "a") {
  console.log("Happy");
}
//Output: Happy(infinity)
```

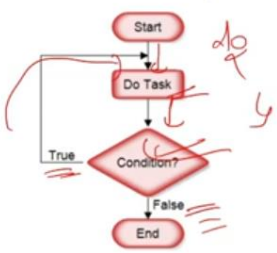
Q. What is the difference between while and do-while loops? V. IMP.

While Loop



- While loop execute a block of code while a certain **condition** is true.

Do While Loop



- The do-while loop is similar to the while loop, except that the block of code is **executed at least once**, even if the condition is false.

Q. What is the difference between while and do-while loops? V. IMP.

```
// while loop
let j = 0;

while (j < 5) {

  console.log(j);
  j++;

}
// Output: 0 1 2 3 4
```

```
// do-while loop
let k = 0;

do {

  console.log(k);
  k++;

} while (k > 1);
// Output: 0
```

Q. What is the difference between break and continue statement? V. IMP.

- The "break" statement is used to **terminate** the loop.

```
//break statment
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    break;
  }
  console.log(i);
}
//Output: 1 2
```

- The "continue" statement is used to **skip the current iteration** of the loop and move on to the next iteration.

```
//continue statment
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    continue;
  }
  console.log(i);
}
//Output: 1 2 4 5
```

Q. What is the difference between for and for...of loop in JS?

- for loop is slightly more complex having more lines of code whereas **for...of is much simpler** and better for iterating arrays.

```
let arr = [1, 2, 3];
```

```
// for loop has more code
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
//Output: 1 2 3
```

```
// for of is much simpler
for (let val of arr) {
  console.log(val);
}
//Output: 1 2 3
```

Q. What is the difference between for...of and for...in loop? V. IMP.

- for...of loop is used to loop through the **values** of an object like arrays, strings.
- It allows you to access each value **directly**, without having to use an index.

```
let arr = [1, 2, 3];
for (let val of arr) {
  console.log(val);
}
//Output: 1 2 3
```

- for...in loop is used to loop through the **properties** of an object.
- It allows you to iterate **over the keys of an object** and access the values associated by using keys as the index.

```
// for-in loop
const person = {
  name: 'Happy',
  role: 'Developer'
};

for (let key in person) {
  console.log(person[key]);
}
//Output: Happy Developer
```

Q. What is forEach method? Compare it with for...of and for...in loop? V. IMP.

- forEach() is a method available on arrays or object that allows you to **iterate over each element** of the array and perform some action on each element.

```
const array = [1, 2, 3];
```

```
//for...of loop
for (let item of array) {
  console.log(item);
}
//Output: 1 2 3
```

```
//forEach method
array.forEach(function(item) {
  console.log(item);
});
//Output: 1 2 3
```

```
const person = {
  name: 'Happy',
  role: 'Developer'
};
```

```
// for-in loop
for (let key in person) {
  console.log(person[key]);
}
//Output: Happy Developer
```

```
//forEach method
Object.values(person).forEach(value =>{
  console.log(value);
});
//Output: Happy Developer
```

Q. When to use for...of loop and when to use forEach method in applications? V. IMP.

- for...of loop is suitable when you need **more control over the loop**, such as using break statement or continue statement inside.
- forEach method **iterate over each element** of the array and perform some action on each element.

```
const array = [1, 2, 3];
```

```
// for-of loop
for (let item of array) {
  console.log(item);

  if (item === 2) {
    break;
  }

}
//Output: 1 2
```

```
//forEach method
array.forEach(function (item) {
  console.log(item);
  if (item === 2) {
    break;
  }
});
// Error: Illegal break statement
```

Q. What is the difference between named and anonymous functions? When to use what in applications?

- Named functions have a **name identifier**
- Anonymous functions **do not have a name identifier** and cannot be referenced directly by name.

```
// Named function

function sum(a, b) {
  return a + b;
}

console.log(sum(5, 3));
// Output: 8
```

```
// Anonymous function

console.log(function(a, b) {

  return a * b;

}(4, 5));

// Output: 20
```

- Use named functions for **big and complex** logics.
- Use when you want to **reuse** one function at multiple places.
- Use anonymous functions for **small logics**.
- Use when want to use a function in a **single place**.

Q. What is function expression in JS?

- A function expression is a way to define a function by **assigning it to a variable**.

```
//Anonymous Function Expression

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

console.log(add(5, 3));
//Output: 8
```

```
//Named Function Expression

const add = function sum(a, b) {
  return a + b;
};

console.log(add(5, 3));
//Output: 8
```

Q. What are Callback Functions? What is it use? V. IMP.

- A callback function is a function that is **passed as an argument** to another function.

Higher-order function | Callback function

```
function add(x, y) {
  return x + y;
}

let a = 3, b = 5;
let result = add(a, b)

console.log(result);
//Output: 8
```

```
function display(x, y, operation) {

  var result = operation(x, y);
  console.log(result);

}

display(10, 5, add);
display(10, 5, multiply);
display(10, 5, subtract);
display(10, 5, divide);
```

Q. What is Higher-order function In JS?

- A Higher order function:
1. Take one or more functions as **arguments**(callback function) OR
2. **Return** a function as a result

```
//Take one or more functions
//as arguments
function hof(func) {

  func();
}

hof(sayHello);

function sayHello() {
  console.log("Hello!");
}
// Output: "Hello!"
```

Callback function

High-order function

```
//Return a function as a result
function createAdder(number) {

  return function (value) {
    return value + number;
  };

}

const addFive = createAdder(5);

console.log(addFive(2));

// Output: 7
```

Q. What is the difference between arguments and parameters?

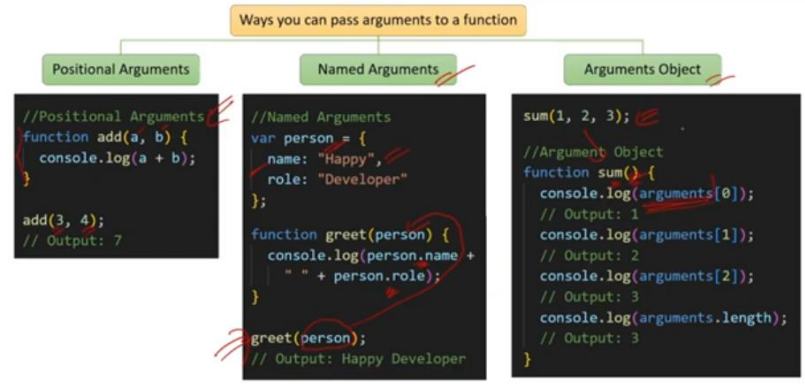
- Parameters are the **placeholders** defined in the function declaration.

```
//a and b are parameters
function add(a, b) {
  console.log(a + b);
}
```

- Arguments are the **actual values passed** to a function when it is invoked or called.

```
add(3, 4);
// 3 and 4 are aruguments
```

Q. In how many ways can you pass arguments to a function?




```
//Positional Arguments
function add(a, b) {
  console.log(a + b);
}

add(3, 4);
// Output: 7
```

```
//Named Arguments
var person = {
  name: "Happy",
  role: "Developer"
};

function greet(person) {
  console.log(person.name +
    " " + person.role);
}

greet(person);
// Output: Happy Developer
```

```
sum(1, 2, 3);

//Argument Object
function sum() {
  console.log(arguments[0]);
  // Output: 1
  console.log(arguments[1]);
  // Output: 2
  console.log(arguments[2]);
  // Output: 3
  console.log(arguments.length);
  // Output: 3
}
```

Q. How do you use default parameters in a function?

- In JavaScript, default parameters allow you to specify **default values** for function parameters.

```
//Function with default parameter value
function greet(name = "Happy") {

  console.log("Hello, " + name + "!");

}
```

```
greet();
// Output: Hello, Happy!
```

```
greet("Amit");
// Output: Hello, Amit!
```

Q. What is the use of event handling in JS? V. IMP.

- Event handling is the process of **responding to user actions** in a web page.
- The **addEventListener** method of Javascript allows to attach an **event name** and with the **function** you want to perform on that event.

```
<button id="myButton">Click me</button>

// Get a reference to the button element
const button = document.getElementById('myButton');

// Add an event listener for the 'click' event
button.addEventListener('click', function() {

  alert('Button clicked!');

});
```

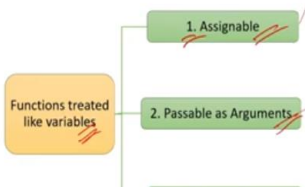
Event

Callback function

- Click Event: addEventListener('click', handler)
- Mouseover Event: addEventListener('mouseover', handler)
- Keydown Event: addEventListener('keydown', handler)
- Keyup Event: addEventListener('keyup', handler)
- Submit Event: addEventListener('submit', handler)
- Focus Event: addEventListener('focus', handler)
- Blur Event: addEventListener('blur', handler)
- Change Event: addEventListener('change', handler)
- Load Event: addEventListener('load', handler)
- Resize Event: addEventListener('resize', handler)

Q. What are First-Class functions in JS?

- A programming language is said to have First-class functions if functions in that language are treated like **other variables**.




```
// 1. Assigning function like a variable
const myFunction = function () {
  console.log("Interview, Happy!");
};
myFunction(); // Output: "Interview, Happy!"
```

```
function double(number) {
  return number * 2;
}
// 2. Passing function as an argument like a variable
function performOperation(double, value) {
  return double(value);
}
console.log(performOperation(double, 5)); // Output: 10
```

```
// 3. A function that returns another function
function createSimpleFunction() {
  return function () {
    console.log("I am from return function.");
  };
}
const simpleFunction = createSimpleFunction();
simpleFunction(); // Output: "I am from return function."
```

Q. What are Pure and Impure functions in JS?

1. A pure function is a function that always produces the **same output for the same input**.
2. Pure functions cannot modify the **state**.
3. Pure functions cannot have **side effects**.

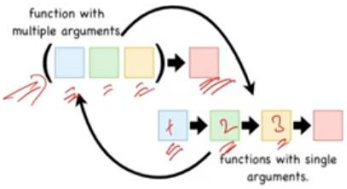
```
// Pure function
function add(a, b) {
  return a + b;
}

console.log(add(3, 5));
// Output: 8

console.log(add(3, 5));
// Same Output: 8
```

1. An impure function, can produce **different outputs for the same input**.
2. Impure functions can modify the state.
3. Impure functions can have side effects.

```
// Impure function
let total = 0;

function addToTotal(value) {
    total += value;
    return total;
}

console.log(addToTotal(5));
// Output: 5

console.log(addToTotal(5));
// Not same output: 10
```

Q. What is Function Currying in JS?

- Currying in JavaScript transforms a function with multiple arguments into a **nested series of functions**, each taking a single argument.
- Advantage : **Reusability, modularity, and specialization**. Big, complex functions with multiple arguments can be broken down into small, reusable functions with fewer arguments.




```
// Regular function that takes two arguments
// and returns their product
function multiply(a, b) {
  return a * b;
}

// Curried version of the multiply function
function curriedMultiply(a) {
  return function (b) {
    return a * b;
  };
}

// Create a specialized function for doubling a number
const double = curriedMultiply(2);
console.log(double(5));
// Output: 10 (2 * 5)

// Create a specialized function for tripling a number
const triple = curriedMultiply(3);
console.log(triple(5));
// Output: 15 (3 * 5)
```

Q. What are call, apply and bind methods in JS?

- call, apply, and bind are three methods in JavaScript that are used to work with functions and **control how they are invoked** and what context they operate in.
- These methods provide a way to manipulate the **this value** and pass arguments to functions.

```
// Defining a function that uses the "this" context and an argument
function sayHello(message) {
  console.log(`${message}, ${this.name}!`);
}
const person = { name: 'Happy' };
```

```
// 1. call - Using the "call" method to invoke the function
// with a specific context and argument
sayHello.call(person, 'Hello');
// Output: "Hello, Happy!"
```

```
// 2. apply - Using the "apply" method to invoke the function
// with a specific context and an array of arguments
sayHello.apply(person, ['Hi']);
// Output: "Hi, Happy!"
```

```
// 3. bind - Using the "bind" method to create a new function
// with a specific context (not invoking it immediately)
const greetPerson = sayHello.bind(person);
greetPerson('Greetings');
// Output: "Greetings, Happy!"
```

Q. What is a String?

- A string is a **data type** used to **store and manipulate data.**

```
// Single quotes ('')
var str1 = 'Hello';
```

Q. What are template literals and string interpolation in strings? V. IMP.

- A template literal, also known as a template string, is a feature introduced in ECMAScript 2015 (ES6) for **string interpolation** and **multiline strings** in JavaScript.

`${Template} Literal`

```
// Backticks (`)
//Template literals with string interpolation
var myname = "Happy";
var str3 = `Hello ${myname}!`;
console.log(str3);
// Output: Hello Happy!
```

```
// Backticks (`)
//Template literals for multiline strings
var multilineStr = `
This is a
multiline string.
`;
```

Q. What is the difference between single quotes (''), double quotes ("") & backticks (``)?

```
// Single quotes ('')
var str1 = 'Hello';
```

```
// Backticks (`)
//Template literals with string interpolation
var myname = "Happy";
var str3 = `Hello ${myname}!`;
console.log(str3);
// Output: Hello Happy!
```

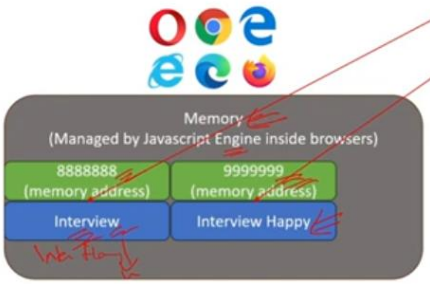
```
// Double quotes ("")
var str2 = "World";
```

```
// Backticks (`)
//Template literals for multiline strings
var multilineStr = `
This is a
multiline string.
`;
```

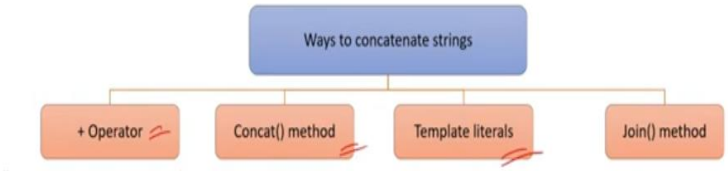
Q. What is string immutability? V. IMP.

- Strings in JavaScript are considered **immutable** because you **cannot modify** the contents of an existing string directly.

```
var str = 'Interview';

// Creates a new string
str = str + ' Happy';
```



Q. In how many ways you can concatenate strings?




```
let s1 = 'Hello';
let s2 = 'World';
```

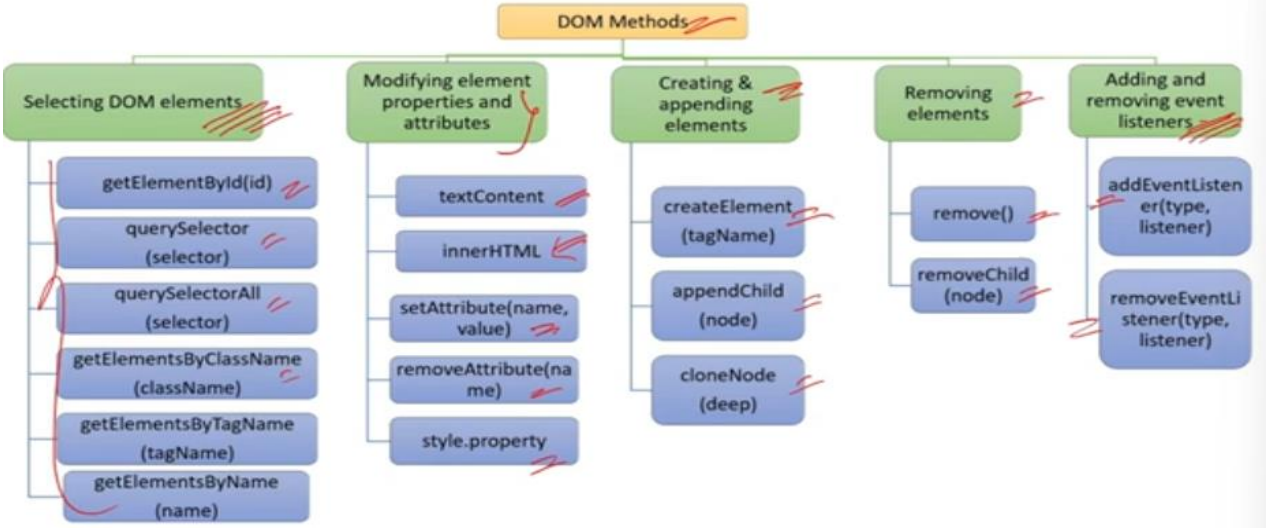
```
// + operator
let r1 = s1 + s2;
console.log(r1);
// Output: HelloWorld
```

```
// concat() method
let r2 = s1.concat(s2);
console.log(r2);
// Output: HelloWorld
```

```
// template literals
let r3 = `${s1} ${s2}`;
console.log(r3);
// Output: Hello World
```

```
// join() method
let strings = [s1, s2];
let r4 = strings.join(' ');
console.log(r4);
// Output: Hello World
```

Q. How do you select, modify, create and remove DOM elements?



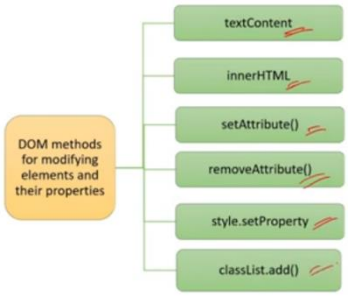
Q. What is the difference between querySelector() and querySelectorAll()?

```html
<html>
  <head>
    <title></title>
  </head>
  <body>
    <div class="myClass">Element 1</div>
    <div class="myClass">Element 2</div>
    <div class="myClass">Element 3</div>
    <script src="index.js"></script>
  </body>
</html>
```

```
// Using querySelector() - returns the first element
var element = document.querySelector('.myClass');
console.log(element.textContent);
// Output: Element 1
```

```
// Using querySelectorAll() - returns all the elements
var elements = document.querySelectorAll('.myClass');
elements.forEach(function(element) {
  console.log(element.textContent);
});
// Output: Element 1, Element 2, Element 3
```

Q. What are the methods to modify elements properties and attributes?




```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>

    <div id="myElement">Hello, World!</div>

    <script src="index.js"></script>
  </body>
</html>
```

Q. What is the difference between innerHTML and textContent? V. IMP.


```
<div id="myElement1">Hello</div>
<div id="myElement2">World</div>
```

```
// textContent property used to assign plain text to element
var element1 = document.getElementById("myElement1");
element1.textContent = "<strong>Happy</strong>";
```

```
// innerHTML property, the browser interprets the content
// as HTML and renders it accordingly
var element1 = document.getElementById("myElement2");
element1.innerHTML = "<strong>Happy</strong>";
```

Happy

Happy

Q. How to add and remove properties of HTML elements in the DOM using JS?


```
<div id="myElement">Hello, World!</div>
```

```
var element = document.getElementById("myElement");
// setAttribute method is used to add propetry
element.setAttribute("data-info", "new value");
```

```
<body>
  <div id="myElement" data-info="new value">Hello, World!</div>
```

```
// removeAttribute method is used to remove propetry
element.removeAttribute("data-info");
```

```
<body> == $0
  <div id="myElement">Hello</div>
```

Q. How to add and remove style from HTML elements in DOM using JS?

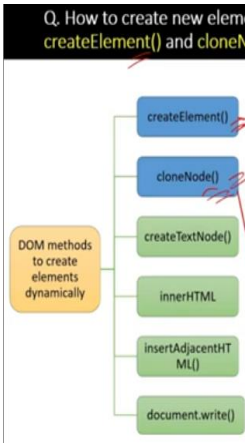

```
<div id="myElement">Hello, World!</div>
```

```
var element = document.getElementById("myElement");

// style.setProperty is used to modify element style
element.style.setProperty("color", "blue");
```

```
// add the new class to element
element.classList.add("highlight");
element.classList.remove("highlight");
element.classList.toggle("highlight");
```

```
<div id="myElement" class="highlight" style="color: blue;">Hello, World!</div>
```

Q. How to create new elements in DOM using JS? What is the difference between createElement() and cloneNode()?



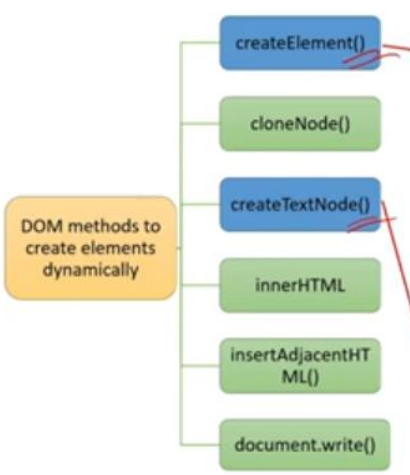

```
// createElement() - create new element
var newDiv = document.createElement('div');
newDiv.textContent = 'Newly created div';
document.body.appendChild(newDiv);
```

```
<div>Newly created div</div>
```

```
<div id="parentElement">
  <h1>Existing Element</h1>
</div>
```

```
// cloneNode() - copy the existing element with all attributes
var existingElement = document.getElementById('parentElement');
var clonedElement = existingElement.cloneNode(true);
clonedElement.textContent = 'Cloned element';
document.body.appendChild(clonedElement);
```

Q. What is the difference between createElement() and createTextNode()?




```
// createElement() - create new element
var newDiv = document.createElement('div');
newDiv.textContent = 'Newly created div';
document.body.appendChild(newDiv);
```

```
<div>Newly created div</div>
```

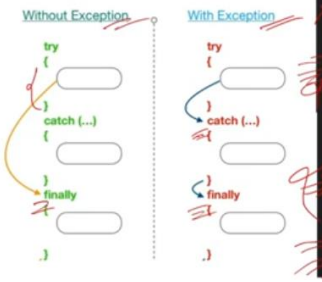
```
<div id="parentElement">
  <h1>Existing Element</h1>
</div>
```

```
var parentElement = document.getElementById("parentElement");
// Create a new text node
var newText = document.createTextNode(" This is some text.");
// Append the text node to the parent
parentElement.appendChild(newText);
```

```
<div id="parentElement">
  <h1>Existing Element</h1>
  " This is some text."
</div>
```

Q. What is the role of finally block in JS?

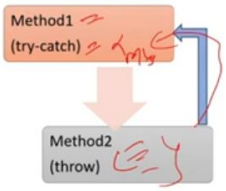
- Finally, block is used to **execute some code irrespective of error.**




```
//try block contains the code that might throw an error
try {
  const result = someUndefinedVariable + 10;
  console.log(result);
}
//catch block is where the error is handled
catch (error) {
  console.log('An error occurred:', error.message);
}
//finally block to execute code regardless of whether
//an error occured or not
finally{
  console.log("finally executed");
}

//Output
//An error occurred: someUndefinedVariable is not defined
//finally executed
```

Q. What is the purpose of the throw statement in JS?

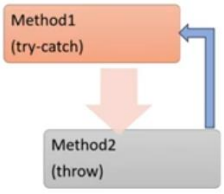
- The throw statement stops the execution of the current function and **passes the error to the catch block of calling function.**




```
function UserData() {
  try {
    validateUserAge(25);
    validateUserAge("invalid"); // This will throw
    validateUserAge(15); // This will not execute
  } catch (error) {
    console.error("Error:", error.message);
  }
}
function validateUserAge(age) {
  if (typeof age !== "number") {
    throw new Error("Age must be a number");
  }
  console.log("User age is valid");
}
```

Q. What is Error propagation in JS?

- Error propagation refers to the process of passing or propagating an error from **one part of the code to another** by using the **throw statement** with try catch.




```
function UserData() {
  try {
    validateUserAge(25);
    validateUserAge("invalid"); // This will throw
    validateUserAge(15); // This will not execute
  } catch (error) {
    console.error("Error:", error.message);
  }
}
function validateUserAge(age) {
  if (typeof age !== "number") {
    throw new Error("Age must be a number");
  }
  console.log("User age is valid");
}
```

Q. What are the best practices for error handling?

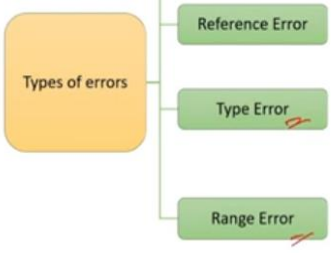
```
// 1. Use Try Catch and Handle Errors Appropriately
try {
  // Code that may throw an error
} catch (error) {
  // Error handling and recovery actions
}
```

```
// 3. Avoid Swallowing Errors
try {
  // Code that may throw an error
} catch (error) {
  // Do not leave the catch blank
}
```

```
// 2. Use Descriptive Error Messages
throw new Error("Cannot divide by zero");
```

```
// 4. Log Errors
try {
  // Code that may throw an error
} catch (error) {
  console.error("An error occurred:", error);
  // Log the error with a logging library
}
```

Q. What are the different types of errors In JS?



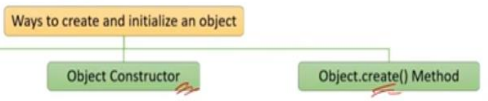

```
// Syntax Error
console.log("Hello, World!"
// Missing closing parenthesis );
```

```
// Reference Error
console.log(myVariable);
// myVariable is not defined
```

```
// Type Error
const number = 42;
console.log(number.toUpperCase());
// number.toUpperCase is not a function
```

```
// Range Error
const array = [1, 2, 3];
console.log(array[10]);
// Index 10 is out of bounds
```

Q. In how many ways we can create an object?




```
// Object literal
var person = {
  name: "Happy",
  age: 38,
  role: "Trainer"
};
console.log(person);
```

```
▼ {name: 'Happy', age: 38, role: 'Trainer'}
  age: 38
  name: "Happy"
  role: "Trainer"
  ▶ [[Prototype]]: Object
```

```
// Object Constructor
var person = new Object();
person.name = "Happy";
person.age = 38;
person.role = "Trainer";
console.log(person);
```

```
// Object.create() Method
var person = {
  name: "",
  age: 0,
  role: ""
};
var men = Object.create(person);
men.name = "Happy";
men.age = 38;
men.role = "Trainer";
```

Q. What is the difference between an array and an object?

Arrays	Objects
1. Arrays are collection of values.	Objects are collections of key-value pairs.
2. Arrays are denoted by square brackets [].	Objects are denoted by curly braces {}.
3. Elements in array are ordered.	Properties in objects are unordered.

```
// Array
var fruits = ["apple", "banana", "orange"];
```

```
// Object
var person = {
  name: "Amit",
  age: 25,
  city: "Delhi"
};
```

Q. How do you add or modify or delete properties of an object?

```
//Blank object
var person = {};
```

```
// Adding Properties
person.name = "Happy";
person.age = 35;
person.country = "India"
```

```
// Modifying Properties
person.age = 30;
```

```
// Deleting Properties
delete person.age;
```

Q. Explain the difference between dot notation and bracket notation?

- Both dot notation and bracket notation are used to **access properties or methods** of an object.
- Dot notation is more popular and used due to its **simplicity**.

```
const person = {
  name: 'Happy',
  age: 35,
};
```

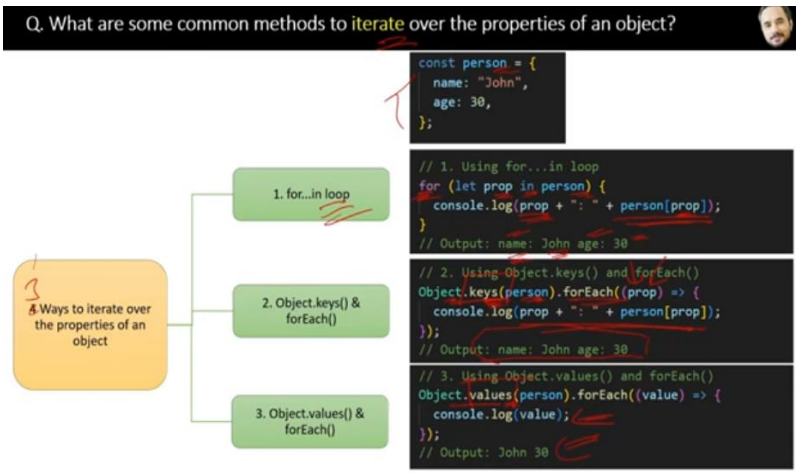
```
// Dot notation:
console.log(person.name);
// Output: 'Happy'
```

```
// Bracket notation:
console.log(person['name']);
// Output: 'Happy'
```

- Limitation of dot notation - In some scenarios bracket notation is the only option, such as when accessing properties when the **property name is stored in a variable**.

```
// Dynamically assign property name
// to a variable
var propertyName = 'age';

console.log(person[propertyName]);
// Output: '35'

console.log(person.propertyName);
// Output: undefined
```

Q. What are some common methods to iterate over the properties of an object?




```
const person = {
  name: "John",
  age: 30,
};
```

```
// 1. Using for...in loop
for (let prop in person) {
  console.log(prop + ": " + person[prop]);
}
// Output: name: John age: 30
```

```
// 2. Using Object.keys() and forEach()
Object.keys(person).forEach((prop) => {
  console.log(prop + ": " + person[prop]);
});
// Output: name: John age: 30
```

```
// 3. Using Object.values() and forEach()
Object.values(person).forEach((value) => {
  console.log(value);
});
// Output: John 30
```

Q. How do you check if a property exists in an object?

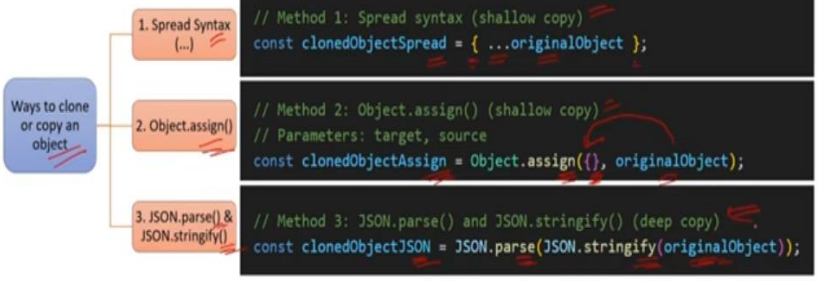
```
var person = {
  name: "Alice",
  age: 25
};
```

```
// 1. Using the in Operator
console.log("name" in person); // Output: true
console.log("city" in person); // Output: false
```

```
// 2. Using the hasOwnProperty() Method
console.log(person.hasOwnProperty("name")); // Output: true
console.log(person.hasOwnProperty("city")); // Output: false
```

```
// 3. Comparing with undefined
console.log(person.name !== undefined); // Output: true
console.log(person.city !== undefined); // Output: false
```

Q. How do you clone or copy an object?




```
// Original object
const originalObject = {
  name: 'Happy',
  age: 35,
  city: 'Delhi',
};
```

```
// Method 1: Spread syntax (shallow copy)
const clonedObjectSpread = { ...originalObject };
```

```
// Method 2: Object.assign() (shallow copy)
// Parameters: target, source
const clonedObjectAssign = Object.assign({}, originalObject);
```

```
// Method 3: JSON.parse() and JSON.stringify() (deep copy)
const clonedObjectJSON = JSON.parse(JSON.stringify(originalObject));
```

Q. What is the difference between deep copy and shallow copy in JS?

- Shallow copy in **nested objects case** will modify the parent object property value, if cloned object property value is changed. But deep copy will not modify the parent object property value.

```
// Original object
const person = {
  name: 'Happy',
  age: 30,
  address: {
    city: 'Delhi',
    country: 'India'
  }
};
```

```
// Shallow copy using Object.assign()
const shallowCopy = Object.assign({}, person);

shallowCopy.address.city = 'Mumbai';

console.log(person.address.city); // Output: "Mumbai"
console.log(shallowCopy.address.city); // Output: "Mumbai"
```

```
// Deep copy using JSON.parse() and JSON.stringify()
const deepCopy = JSON.parse(JSON.stringify(person));

deepCopy.address.city = 'Bangalore';

console.log(person.address.city); // Output: "Delhi"
console.log(deepCopy.address.city); // Output: "Bangalore"
```

Q. What is Set Object in JS?

- The Set object is a collection of **unique values**, meaning that duplicate values are not allowed.
- Set provides methods for **adding, deleting, and checking the existence** of values in the set.
- Set can be used to **remove duplicate values** from arrays.

```
// Set can be used to remove
// duplicate values from arrays
let myArr = [1, 4, 3, 4];
let mySet = new Set(myArr);

let uniqueArray = [...mySet];
console.log(uniqueArray);
// Output: [1, 4, 3]
```

```
// Creating a Set to store unique numbers
const uniqueNumbers = new Set();
uniqueNumbers.add(5);
uniqueNumbers.add(10);
uniqueNumbers.add(5); //Ignore duplicate values

console.log(uniqueNumbers);
// Output: {5, 10}

// Check size
console.log(uniqueNumbers.size);
// Output: 2

// Check element existence
console.log(uniqueNumbers.has(10));
// Output: true

// Delete element
uniqueNumbers.delete(10);
console.log(uniqueNumbers.size);
// Output: 1
```

Q. What is Map Object in JS?

- The Map object is a collection of **key-value** pairs where each key can be of **any type**, and each value can also be of any type.
- A Map maintains the **order** of key-value pairs as they were inserted.

```
// Creating a Map to store person details
const personDetails = new Map();
personDetails.set("name", "Alice");
personDetails.set("age", 30);

console.log(personDetails.get("name"));
// Output: "Alice"

console.log(personDetails.has("age"));
// Output: true

personDetails.delete("age");
console.log(personDetails.size);
// Output: 1
```

Q. What is the difference between Map and Object in JS?

Map	Javascript Object
1. Keys in a Map can be of any data type, including strings, numbers, objects, functions etc.	Keys in a regular JavaScript object are limited to strings and symbols.
2. A Map maintains the order of key-value pairs as they were inserted.	In a regular object, there is no guaranteed order of keys.
3. Useful when keys are of different types, insertion order is important.	Useful when keys are strings or symbols and there are simple set of properties.

Q. What are Events? How are events triggered? V. IMP.

- Events are **actions** that happen in the browser, such as a button click, mouse movement, or keyboard input.

```html
<button id="myButton">Click Me</button>
```

```js
// Get the reference of button in a variable
var button = document.getElementById("myButton");

// Attach an event handler to the button
button.addEventListener("click", handleClick);

// Event handler function
function handleClick() {
  alert("button clicked");
}
```

Event | Event handler/ Callback function

Q. What are the types of events in JS? V. IMP.

```html
<button id="myButton">Click Me</button>
```

```js
// Get the reference of button in a variable
var button = document.getElementById("myButton");

// Attach an event handler to the button
button.addEventListener("click", handleClick);

// Event handler function
function handleClick() {
  alert("button clicked");
}
```

Event

- Click Event: addEventListener('click', handler)
- Mouseover Event: addEventListener('mouseover', handler)
- Keydown Event: addEventListener('keydown', handler)
- Keyup Event: addEventListener('keyup', handler)
- Submit Event: addEventListener('submit', handler)
- Focus Event: addEventListener('focus', handler)
- Blur Event: addEventListener('blur', handler)
- Change Event: addEventListener('change', handler)
- Load Event: addEventListener('load', handler)
- Resize Event: addEventListener('resize', handler)

Q. What is Event Object in JS?

- Whenever any event is triggered, the browser automatically creates an event object and **passes it as an argument** to the event handler function.
- The event object contains various properties and methods that provide **information about the event**, such as the type of event, the element that triggered the event etc.

```html
<button id="myButton">Click Me</button>
```

```js
// Get the button element
var button = document.getElementById("myButton");

// Attach event listener to the button element
button.addEventListener("click", handleClick);

// Event handler function
function handleClick(event) {
  // Accessing properties of the event object
  console.log("Event type:", event.type);
  console.log("Target element:", event.target);
}
```

```
Elements Console Sources Network Performance
top Filter
Event type: click
Target element: <button id="myButton">Click Me</button>
```

Q. What is Event Delegation in JS? V. IMP.

- Event delegation in JavaScript is a technique where you attach a **single event handler to a parent element** to handle events on its **child elements**.

```html
<ul id="myList">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

- Item 1
- Item 2
- Item 3

```
Clicked: Item 1
Clicked: Item 2
Clicked: Item 3
```

```js
var parentList = document.getElementById("myList");

// Attach event handler to parent element
parentList.addEventListener("click", handleItemClick);

// Event handler function
function handleItemClick(event) {
  var target = event.target;
  console.log("Clicked: ", target.textContent);
}
```

Q. What is Event Bubbling In JS? V. IMP.

- Event bubbling is the process in JavaScript where an event triggered on a child element **propagates up the DOM tree**, triggering event handlers on its parent elements.

```
Click Me
Elements Console
top Filter
Bubbling: myButton
Bubbling: inner
Bubbling: outer
>
```

```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

```js
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");

// Attach event handlers with elements
outer.addEventListener("click", handleBubbling);
inner.addEventListener("click", handleBubbling);
button.addEventListener("click", handleBubbling);

function handleBubbling(event) {
  console.log("Bubbling: " + this.id);
}
```

Q. How can you stop event propagation or event bubbling in JS?

- Event bubbling can be stopped by calling **stopPropagation()** method on event.

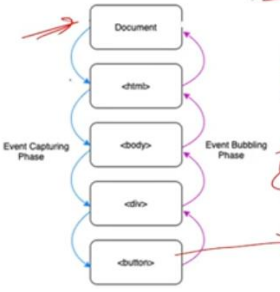
```
Click Me
Elements Console
top Filter
Bubbling: myButton
>
```

```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

```js
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");

// Attach event handlers with elements
outer.addEventListener("click", handleBubbling);
inner.addEventListener("click", handleBubbling);
button.addEventListener("click", handleBubbling);

function handleBubbling(event) {
  console.log("Bubbling: " + this.id);
  event.stopPropagation(); // Stop event propagation
}
```

Q. What is Event Capturing in JS?

- Event capturing is the process in JavaScript where an event is handled starting from the highest-level ancestor (the root of the DOM tree) and **moving down to the target element**.




```html
<div id="outer">
  <div id="inner">
    <button id="myButton">Click Me</button>
  </div>
</div>
```

```js
// Get the reference of elements
var outer = document.getElementById("outer");
var inner = document.getElementById("inner");
var button = document.getElementById("myButton");

// Attach event handlers with elements
outer.addEventListener('click', handleCapture, true);
inner.addEventListener('click', handleCapture, true);
button.addEventListener('click', handleCapture, true);

function handleCapture(event) {
  console.log("Capturing: " + this.id);
}
```

Q. What is the purpose of the event.preventDefault() method in JS?

- The event.preventDefault() method is used to **prevent the default behavior** of an event and the link click will be prevented.

```html
<a href="https://example.com" id="myLink">Click Me</a>
```

```js
var link = document.getElementById('myLink');

link.addEventListener('click', handler);

function handler(event)
{
    event.preventDefault(); //Prevent default action

    // Perform custom behavior
    console.log('Clicked, default action prevented');
}
```

Q. What is the use of "this" keyword in the context of event handling in JS?

- "this" keyword refers to the **element** that the event handler is attached to.

```html
<button id="myButton">Click Me</button>
```

```js
var button = document.getElementById("myButton");
button.addEventListener("click", handler);

function handler(event) {
  console.log("Clicked:", this.id);
  this.disabled = true;
}
```

Q. How to remove an event handler from an element in JS?

- **removeEventListener()** method is used to remove event handler from element.

```html
<button id="myButton">Click Me</button>
```

```js
var button = document.getElementById("myButton");

// Attach the event handler
button.addEventListener("click", handleClick);

function handleClick() {
  console.log("Button clicked!");
}
```

```js
// Remove the event handler
button.removeEventListener("click", handleClick);
```