**Javascript**

**Javascript is a programming language used both on a client side and server-side that allows you to make web pages interactive**

## **Application of JavaScript**

**JavaScript is used to create interactive websites. It is mainly used for:**

* **Client-side validation,**
* **Dynamic drop-down menus,**
* **Displaying date and time,**
* **Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),**
* **Displaying clocks etc.**

### **Features of JavaScript**

1. **Object based scripting language**
2. **It is a light-weighted and interpreted language.**
3. **It is a case-sensitive language.**
4. **JavaScript is supportable in several operating systems including, Windows, macOS, etc.**
5. **It provides good control to the users over the web browsers.**
6. **javascript is both server and client side language.**

#### **Differences between java & javascript**

1. **Java is an object oriented programming language,javascript is object based scripting language**
2. **Java can run on any virtual machines(JVM) or browser,javascript code used to run only in browsers.**
3. **Java is server side programming language,javascript is both server and client side language**
4. **Java is used for core business logic,javascript used for both UI and business logic**
5. **Java is a compiled language,javascript is a scripting language.**

## **Javascript comments**

**The JavaScript comments are a meaningful way to deliver messages. It is used to add information about the code**

#### **Advantages of JavaScript comments**

**There are mainly two advantages of JavaScript comments.**

1. **To make code easy to understand It can be used to elaborate the code so that end users can easily understand the code.**
2. **To avoid the unnecessary code It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after some time, there may be a need to disable the code. In such cases, it is better to use comments.**

**We can comment the code in javascript by two ways**

1. **Single line comment**

**a single line comment can be added by using two forward slashes //**

**Ex:**

**let x = 5; // Declare x, give it the value of 5**

**let y = x + 2; // Declare y, give it the value of x + 2**

**2) Multi line comment**

**Javascript multiline comments, also known as block comments, start with a forward slash followed by an asterisk (/\*) and end with an asterisk followed by a forward slash (\*/).**

**EX:**

**/\* It is a multi-line comment.**

**It will not be displayed \*/**

**document.write("example of javascript multiline comment");**

**Variables and values:**

**Variable - variables are containers to store the values. Variables means anything that can vary.**

**value - value is the smallest information that we have in JS.**

**let x = 20;**

**console.log(x); \*/// preferable**

**// document.write(x);**

**While declaring a identifier we should follow some set of rules**

**Rule while creating identifiers**

**1. It shouldn't be a reserved word.**

**2. It shouldn't start with numbers.**

**3. It shouldn't contains any special characters other than $ and \_**

**Multiple variable declaration**

**Declaration and initialization both are different**

**let name; //declaration**

**name = "Sanjay"; // Initialization**

**let one = 1;**

**let two = 2;**

**let three = 3;**

**let one = 1, two = 2, three = 3;**

**Loosely coupled (typed) variables**

**While runtime in js we can change the datatype of a variable. It means it does not require a data type to be declared. you can assign any literal values to a variable.**

**\*\*\*\*\*javascript is loosely coupled or strictly coupled language\*\*\*\*\* loosely coupled language**

**let num1 = 10;**

**console.log(num1);**

**While run time we can change the data type of a variable**

**num1 = "India";**

**console.log(num1);**

**We can't change the values while run time called strictly coupled**

**var, let and const**

**Reassignment:**

**We can change the values in middle of the code**

**var x = 10;**

**console.log(x);**

**x = 50; // reassignment is possible**

**console.log(x);**

**let a = "Hello";**

**console.log(a);**

**a = 100; // reassignment is possible**

**console.log(a);**

**const dob = 1995;**

**console.log(dob);**

**// dob = 2000; // reassignment is NOT possible**

**console.log(dob);**

**If we declare a variable as a const we can't able to reassign it .it will fixed for entire execution**

**when ever javascript find any error remaining code will not gonna execute remaining code it will block,we should resolve it first**

**{**

**var b = 5;**

**let c = 6; // block scoped**

**const d = 7 // block scoped**

**}**

**console.log(c, d);**

**Differences between let var const:**

* **When we declare a variable with var keyword inside the block we can access outside the block also**
* **When we declare a variable with let keyword inside the block we cannot able to access outside the block**
* **By using let and var keyword we can able to reassign the values**
* **If we declare variable as a const we cant re assign the values it is fixed for entire execution**

# **Data Types**

**In every programming language has own representation of data.In js the data types are classified as Primitive and non primitive**

1. **Primitive data type**

**Number --------->represents numeric values**

**String —---------> represents the sequence of characters eg:”hello” We can represent the string by using single or doubled quote**

**Boolean —-------> represents boolean value either false or true**

**Null —------->represents null i.e. no value at all ,Variable should not contain any value it is empty for entire execution**

**Undefined —---->represents undefined value,define a variable not assigned any value**

**Bigint—-----------> large integer value**

**symbol—---------> by using the symbols last two are in developed state only**

**2.Non-primitive (reference) data type**

**Arrays**

**Objects**

**Functions**

**Sets**

**Maps**

# **Operators**

**Operators operate on one or more operands and produce the result**

**1. Arithmetic Operators(+, -, \*, /, %, ++, --)**

**2. Assignment Operators (=, +=, -=, \*=)**

**3. Comparison Operators (==, ===, >, <, >=, <=, !=)**

**4.Logical Operators( &&, ||, !)**

**5. Conditional / ternary (?:)**

**Arithmetic operator**

**let a = 10;**

**let b = 20;**

**let sum = a + b;**

**console.log(sum);**

**console.log(10 + 20);**

**console.log(20 - 15);**

**console.log(5 \* 6);**

**console.log(10 / 2);**

**console.log(15 % 2);**

**//increment**

**//prefix increment**

**let i = 3;**

**console.log(++i); // i = i + 1**

**In prefix first it is going to increase the line and going to execute the line**

**//postfix increment**

**let j = 6;**

**j++;**

**console.log(j);**

**In postfix first it execute the line and after it will increment the line**

**//Decrement**

**let k = 5;**

**//prefix decrement**

**console.log(--k); // k = k -1**

**//postfix decrement**

**k--;**

**console.log(k);**

**Assignment operators**

**//+=**

**let a = 17;**

**const b = 15;**

**a += b; // a = a + b**

**It avoid one variable declaration**

**console.log(a);**

**// -=**

**let x = 100;**

**const y = 50;**

**x -= y; // x = x -y**

**console.log(x);**

**// \*=**

**let m = 10;**

**const n = 2;**

**m \*= n // m = m \* n**

**console.log(m);**

**Comparison operators**

**Compare two values and produce a boolean value**

**let str1 = "10";**

**let str2 = 10;**

**console.log(str1 == str2);**

**If we use == comparison operator never ever check the data type it only check the value**

**console.log(str1 === str2);**

**=== it is strict comparison first it check the datatype both are matches next it is going to check the values both are matched then it is going to execute**

**console.log(100 > 50);**

**console.log(100 < 50);**

**console.log(10 >= 9);**

**console.log(20 <= 19);**

**console.log(100 !== 100);**

**Logical operators (&&, ||, !)**

**console.log(true && true);**

**console.log(false || false);**

**console.log(!false);**

#### **Type conversions/Type casting**

#### **Type conversion:**

**The process of converting a value from one type to another**

* **Changing the expression from one data type to another**
* **If the conversion happens explicitly by typing a method or a function name before converting values called type conversion**
* **We can convert only numbers,string and boolean**

**Why type conversion is used:**

**Transfer the data from one data type to another**

**Example:**

**number**

**let year="2023";//string**

**console.log(typeof year);**

**//////converting string into number**

**////by using number**

**const res=Number(year);**

**console.log(year);**

**console.log(res);**

**console.log(typeof year);**

**console.log(typeof res);**

**//////////by using plus operator**

**const new year=+year;**

**console.log(typeof new year);**

**////by using parseint()**

**const char="123.75hello 43$";**

**console.log(parseInt(char));**

**//////parse int extract only the first occurance of the number**

**/// if we want extract float values we can use th parsefloat()**

**console.log(parseFloat(char));**

**///////converting number into string**

**let num=100;**

**console.log(typeof num .toString());**

**/////booleanvalues conversions**

**//truthy values----> Except falsy values remaining negative and positive values**

**// Falsy values==>0."",null,undefined,NaN**

**console.log(Boolean(0));**

**console.log(Boolean(""));**

**console.log(Boolean(-2));**

**console.log(Boolean(9));**

**console.log(Boolean(NaN));**

### **Coercion/Implicit Type Conversion**

* **Coercion is an automatic type conversion that occurs in JavaScript when you want to perform certain operations.**
* **Type coercion is the automatic or implicit conversion of values from one data type to another**

**Example:**

**Automatic type conversion**

**//Addition**

**console.log(1+"2");**

**console.log(1+2);**

**console.log("3"+4+4);**

**console.log(5+3+"13");**

**console.log("12"+undefined);**

**console.log("12"+null);**

**console.log("12"+NaN);**

**//Subtraction**

**console.log("12"-2);**

**console.log(12-2);**

**console.log("12"\*2);**

**console.log("12"/2);**

**console.log("hello"-3);**

**//////Boolean**

**console.log(true+1);**

**console.log(false-1);**

**console.log(true\*undefined);**

**console.log(false/null);**

**console.log(true-NaN);**

## **Statements**

* **JavaScript statements are the commands to tell the browser what action to perform. Statements are separated by semicolon (;). JavaScript statement constitutes the JavaScript code which is translated by the browser line by line**

**if Statement**

* **If statement allows a block of code execution when the condition is true.**
* **If the condition is true it is going to be executed ,if it is not true it won’t execute**

**syntax:**

**if(condition) {**

**//code to be executed**

**}**

**if (10 > 13) {**

**alert("10 is greater than 3")**

**}**

**if-else statement**

* **If the condition is true if block is going to execute,condition is false else block will execute**

**if (100 > 150) {**

**console.log('100 is greater than 50');**

**}**

**else {**

**console.log('not equal');**

**}**

**String or template literal(ES6)**

**const age = 30;**

**const city = 'Mumbai'**

**// console.log('The age of a person is ' + age);**

**console.log(`The age of a person is ${age} and city is ${city}.`);**

**const inpAge1 = +prompt('Please enter the age of person 1'); // 30**

**const inpAge2 = +prompt('Please enter the age of person 2'); //25**

**if (inpAge > inpAge2) {**

**console.log(`Person 1 age ${inpAge1} is greater than Person 2 age ${inpAge2}`);**

**} else {**

**console.log(`Person 2 age ${inpAge2} is greater than Person 1 age ${inpAge1}`);**

**}**

**else-if**

**let age = 5;**

**if (age >= 14) {**

**console.log(`Student is allowed to admit HighSchool, Since his/her age is ${age}.`);**

**} else if (age >= 11 && age <= 13) {**

**console.log(`Student is allowed to admit MiddleSchool, Since his/her age is ${age}.`);**

**} else if (age >= 7 && age <= 10) {**

**console.log(`Student is allowed to admit ElementarySchool, Since his/her age is ${age}.`);**

**} else {**

**console.log(`Student is allowed to Daycare, Since his/her age is ${age}.`);**

**}**

**Switch**

**Syntax:**

**switch(condition) {**

**case x:**

**code to be executed;**

**break;**

**case y:**

**code to be executed;**

**break;**

**.**

**.**

**default:**

**code to be executed;**

**}**

**Example1:**

**const day = "";**

**switch (day) {**

**case "monday":**

**console.log("It's Monday!, Time to work.");**

**break;**

**case "tuesday":**

**console.log("It's Tuesday, Second day of a week.");**

**break;**

**case "wednesday":**

**console.log("It's Wednesday, Already Hump day!");**

**break;**

**case "thursday":**

**console.log("It's Thursday, One more day to weekend!");**

**break;**

**case "friday":**

**console.log("Happy Friday!");**

**break;**

**case "saturday":**

**console.log("It's weekend!");**

**break;**

**case "sunday":**

**console.log("It's time to relax!");**

**break;**

**default:**

**console.log("Something went wrong!");**

**}**

**Example2:**

**const marks = 75;**

**switch (true) {**

**case marks >= 90:**

**console.log('A');**

**break;**

**case marks >= 80:**

**console.log('B');**

**break;**

**case marks >= 70:**

**console.log('C');**

**break;**

**case marks >= 60:**

**console.log('D');**

**break;**

**case marks >= 50:**

**console.log('E');**

**break;**

**default:**

**console.log('F');**

**}**

**Conditional / ternary operator(?:)**

**It's a special operator in JS , which assigns value to a variable based on some condition. It's an alternative to if-else.**

**syntax:**

**let <variable\_name> = (condition) ? Value 1 : Value 2;**

**Example:**

**let x = (5 > 3) ? 5 : 3;**

**console.log(x);**

**let statement1 = "";**

**let statement2 = "Hello World";**

**let val = statement1 ? statement1 : statement2;**

**console.log(val);**

**Loosely-coupled / Strictly coupled**

**Loosely-coupled**

**In JS while runtime we can change the datatype of a variable.**

**let num = 10;**

**console.log(num);**

**num = "Ten";**

**console.log(num);**

**Dynamic / Static**

**Dynamic**

**In JS we are not assigning any datatype for a variable, by seeing the value a variable will take the datatype.**

**let a = true;//Boolean**

## **LOOPS**

**Loops: execution of code block repeatedly or continuously.**

**There are two types of loops in JS**

**1. Entry controlled loop - Condition is checks before the execution of loop.(for-loop, while)**

**2. Exit controlled loop - Condition is checks after the execution of loop.(do-while)**

**iteration - repetition of a process.**

**Initialization - starting point of the loop.**

**For loop**

**Initialization : loop start by initialization starting point of the lop**

**condition: Specify a condition that evaluate to true for next iteration**

**increment/decrement : increase or decrease the counter**

**syntax:**

**for(initialization; condition; increment/decrement) {**

**// code to be repeated**

**}**

**1)print hello world program 5 times**

**for (let i = 1; i <= 5; i++) {**

**document.write('Hello World <br/>')**

**}**

**for (let i = 1; i <= 5; i++) {**

**document.write(`${i} - Hello World <br/>`)**

**}**

**For loop provides statements**

1. **Continue**
2. **Break**

**1)continue:**

**If you want skip one cycle you can use continue**

**The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.**

**for (let i = 1; i <= 5; i++) {**

**if (i == 3) {**

**continue;**

**}**

**// if (i == 3)**

**// continue;**

**console.log(i);**

**}**

**Note:**

* **In if block only one line of code its not required to write curly braces more than one line of code it is not mandatory**
* **JS understood no need to write curly braces ,these single line belongs to the if block**

**2) Break:**

**The break statement is used to jump out of a loop. It can be used to “jump out” of a switch() statement. It breaks the loop and continues executing the code after the loop.**

**for (let i = 1; i <= 5; i++) {**

**if (i == 3) {**

**break;**

**}**

**console.log(i);**

**}**

* **Whenever i reaches to 3 it stop the execution and it comes out of loop and continue the with the remaining code execution**

**Nested loops**

**for (let i = 1; i <= 3; i++) {**

**console.log(`---i = ${i}---`);**

**for (let j = 1; j <= 2; j++) {**

**console.log(`j = ${j}`);**

**}**

**}**

**while loop**

* **JS includes a while loop to execute code repeatedly till it satisfies specified condition, unlike for loop, while loop only requires condition expression.**
* **Executes block of code until it returns false.Unlike for loop while loop does not expect any initializer,increment/decrement**
* **While loop expect only condition**

**syntax:**

**while(condition){**

**//code to be executed**

**}**

**Ex:**

**let i = 1;**

**while (i <= 5) {**

**console.log(i);**

**i++;**

**}**

**1) When do we use while loop and when do we use for loop?**

* **If you know when does loop starts and when does loop ends go for for loop**
* **If you don't know starting and ending points you have only one condition on that time go for while loop**

**Do-while:**

**Do-while loop is similar to while loop the only difference is it evaluates condition expression after the execution of code block. so do-while loop will execute the code block at least once.**

**syntax:**

**do {**

**//code to be executed**

**} while(condition)**

**Example:**

**let i = 7;**

**do {**

**console.log(i);**

**i++;**

**} while (i <= 5)**

**Advantages of do while**

* **First it executes the block,even though condition is false it executes the block minimum one time**

## **Strings**

* **If we declare anything in single or double quotes called a string**
* **String contains a group of characters or sequence of characters**
* **It is a primitive data type**
* **String is textual context.It must be enclosed in single or double quotes**
* **Two ways we can construct a string**
* **String constructor**
* **String literal**

1. **String constructor:**

* **Const time=new string()**
* **It is a built in constructor it has to invoke constructor with a new keyword which will helps us to invoke a constructor**
* **Const time=new string (‘It’ s a)**
* **It is starts throwing an error why because we want define a string with single quotes which includes apostrophe,Javascript traits it is string**
* **Inside a single quote variable we can’t able define an apostrophe**
* **You can use single quote by using (\)**
* **Const time=new string(It\’s six o\’ clock)**

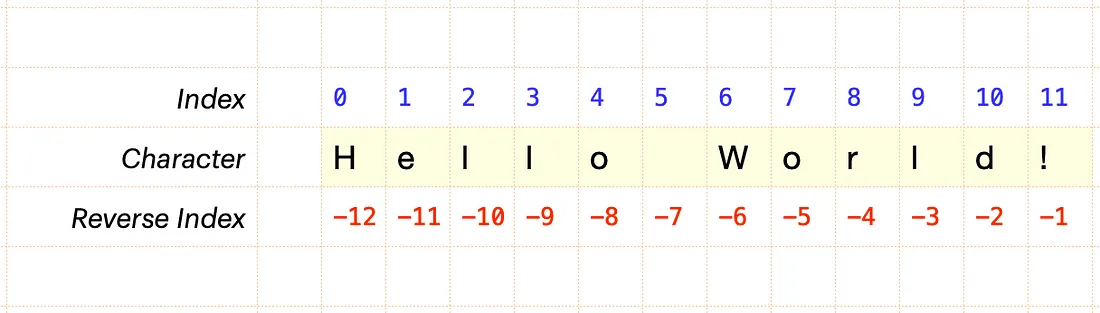
**Or**

* **Const time=new string(“it’ six o’clock**

1. **String literal(Best approach):**

* **Const course=”UI Development”;**
* **console.log(typeof course);//string**
* **String characters are index based . We can able to access each of the character based on index**
* **A string can also be treated as zero index based character array**
* **Index start with 0**
* **length=course.length**
* **Last index=length-1**
* **Strings are immutable:**

**Why because when we do some operations using string the original string not to be affected original string never ever alter**

****

**let str="ravisekhar ramadevi harshavardhan"**

**for(i=0;i<str.length;i++){**

**console.log(str[i])**

**}**

**String methods:**

**let str="ravisekhar ramadevi harshavardhan"**

**1) indexOf():**

**It returns the index of first occurrence of given characters**

**console.log(str.indexOf("ha"));**

**2) last IndexOf()**

**It returns the index of last occurrence of given characters**

**If you get negative index means your character doesn’t exist inside a string it is a validation**

**console.log(str.lastIndexOf("ha"));**

**3) charAt():It returns the character at specified index**

**console.log(str.charAt("0"));//R**

**4) charCodeAt(): It returns the ASCII code of specified index**

**console.log(str.charCodeAt(7));**

**Why we use character code if we want separate all the integers from string we can use char code at()**

**let num="123hello 45";**

**let res='';**

**for (let i=0;i<=num.length-1;i++){**

**if(num[i].charCodeAt(0)>=48 && num[i].charCodeAt(0)<=57){**

**res+=num[i];**

**}**

**}**

**console.log(res)////12345**

**5) substring(): Extract the portion of a string from index and to index**

**console.log(str.substring(0,10));**

**6) split(): Converts string into an array**

**console.log(str.split(""));**

**7) replace(): Replace a old word to a new word,it will replace the first occurrence only if we want to replace all we can use all**

**console.log(course.replace all(“AIT”,’course’);**

**console.log(str.replace("ra","ma"));**

**8) include():**

**console.log(str.includes("i"));**

## **ARRAYS**

**An array is a special type of variable, which can store multiple values using a single variable.**

* **Every value associated with the numeric index starts with 0.**
* **Arrays - contiguous memory allocation of same or different types.**
* **Each element is associated with a index start with 0**
* **Array index must be numeric.**
* **Array is a pack of data**

**There are two ways to create an array.**

**1. Array literal**

**2. Array constructor**

**1. Array Literal(Best approach)**

**syntax:**

**let <variable\_name> = [element 0, element 1, element 2......., element N];**

**const countries = ["India", "Italy", "USA", "UK"];**

* **Accessing array elements**
* **Directly we cannot add two arrays subtracting and comparing not possible**
* **First we have to unpack the array and then we can able to reuse it**

**let a = countries[0];**

**console.log(a);**

**let b = countries[3];**

**console.log(b);**

**console.log(countries.length); total number of elements in a array**

**console.log(typeof countries);/// Array is an object**

**console.log(countries);**

**2. Array constructor**

**const fruits = new Array("Apple", 'Mango', 'Kiwi', 'Orange');**

**console.log(fruits);**

**Accessing all the elements using a for loop**

**const nums = [1, 'two', 3, 4, 5, true];**

**for (let i = 0; i < nums.length; i++) {**

**console.log(nums[i]);**

**Array methods:**

**const countries = ["India", "Italy", "USA", "UK"];**

**1) push ():**

* **Adding one or more values into an array at the last position.**
* **Push is not going to alter the remaining element position.**
* **It is going to add a new element at the last position.**
* **We can add multiple values at a time**

**countries.push("Japan", "Australia");**

**console.log(countries);**

**2) pop() :**

* **Deletes an element from an array at last position**
* **Push and pop are not going to alter the original array element positions**
* **Adding and removing happens at last position**

**countries.pop();**

**console.log(countries);**

**3) shift() - Shift is going to Delete an element from an array at the beginning position .Existing element positions alter**

**countries.shift();**

**console.log(countries);**

**4) unshift() - Adding one or more values into an array at the beginning position**

**countries.unshift('India');**

**console.log(countries);**

**5) indexOf() - to get the index of an element**

**console.log(countries.indexOf('UK'));//3**

**6) includes()- Evaluate the given element that exists in the array.**

**console.log(countries.includes('India'));**

**7) concat() -**

* **Creates a new array by merging existing arrays**
* **It is going to merge two arrays after merging two arrays creates a brand new array**
* **Never ever effect the original array**

**const newArr = countries.concat(fruits);**

**console.log(newArr);**

**console.log(countries);**

**8) slice() - extracts a part of an array**

**console.log(countries.slice(1, 4));**

**console.log(countries);**

**9) splice()**

**countries.splice(2, 0, 'France');**

**console.log(countries);**

**10) join() - converts an array into string**

**console.log(countries.join(""));**

**11) reverse()------- It reverse the entire element**

**console.log(countries.reverse());**

## **OBJECTS**

* **An Object is a non-primitive data type. Objects hold multiple values in terms of properties (key and value pair) and methods.**
* **Object is a collection of properties associated with key and value.**
* **Object keys are always strings.**
* **One key only one value**
* **Values can be any data type**
* **If we want to multiple values in a single key we want to go for array (or) Nested object**
* **We can create an object by {} braces it is data structure not a block**
* **const car = ["Ferrari", 'Red', 2, 'Petrol'];**
* **Array is just store a values in order It doesn’t contain any information what it is**
* **Arrays and objects are co existing both will exist simultaneously; we will work with arrays of objects.**

**Example:**

**const cup = {**

**color: 'Black',**

**shape: 'Cylinder',**

**size: '250ml',**

**price: 199**

**}**

**const car = {**

**"brand": 'Ferrari',**

**color: 'Red',**

**seats: 2,**

**fuel: "Petrol",**

**}**

**console.log(car);**

**There are three way we can create an object in JS**

**1. Object literal**

**2. Object constructor**

**3. Constructor function**

**1. Object literal(best approach)**

**const car = {**

**"brand": 'Ferrari',**

**color: 'Red',**

**seats: 2,**

**fuel: "Petrol",**

**};**

**console.log(car);**

**2. Object constructor**

**const person = new Object();**

**adding properties into object of person**

**person.firstName = "Peter";**

**person.age = 25;**

**person.city = 'Bangalore';**

**console.log(person);**

**Accessing the properties of an Object—-- Accessing can be happen by two ways**

**1. Dot notation**

**const person = {**

**firstName: "Jessica",**

**lastName: 'Davis',**

**age: 25,**

**job: 'Engineer',**

**city: 'Mumbai'**

**};**

**let a = person.firstName;**

**let b = person.city**

**console.log(a, b);**

**Note:**

* **We cannot extract multiple values at one shot. Only one value we can retrieve**
* **We can’t able to duplicate the keys**
* **If we declare array with const keyword can we able to push any value**
* **Yes we can able to push,pop but we cannot able to re assign it**
* **Adding a value and adding an array is different**
* **Push ,pop, shift, unshift,slice effects the original array**

**2. Bracket notation**

**let c = person['age'];**

**console.log(c);**

**let d = person['city'];**

**console.log(d);**

**Dynamically accessing object properties—------ By using using bracket notation we can dynamically access the elements**

**let x = "Name";**

**let e = person['first' + x]; // person['firstName']**

**console.log(e);**

**Adding a new property and modifying the existence property**

**const car = {**

**brand: 'Ferrari',**

**color: 'Red',**

**seats: 2,**

**fuel: "Petrol",**

**}**

**const apple = {**

**brand: "Apple",**

**model: 'iPhone 14',**

**city: "Chennai",**

**price: 80**

**};**

**console.log(apple)**

**adding a new property**

**apple.country = "India"**

**modifying existence**

**apple.price = 100;**

**console.log(apple);**

**hasOwnProperty:**

**Like include it will check the it has own property not the prototype property**

**console.log(apple.hasOwnProperty('area'));**

**Static methods:**

* **Static methods means these methods directly available inside the object constructor not in a object**
* **These methods we should not use over an object**

**1) Object.keys():**

* **It collects all the keys and store in a array**
* **We can’t iterate over an object,arrays can be iterate arrays are iterable,Arrays are position based objects are not position based**
* **We cannot directly compare the objects programmatically we can compare.**

**const keys = Object.keys(apple);**

**console.log(keys);**

**const x = {**

**a: 1,**

**b: 2**

**}**

**const y = {**

**b: 2,**

**a: 1,**

**}**

**const keysX = Object.keys(x); //['a', 'b']**

**const keysY = Object.keys(y); // ['b', 'a']**

**let isEqual = false;**

**console.log(keysX, keysY);**

**if (keysX.length == keysY.length) {**

**for (let i = 0; i < keysX.length; i++) {**

**if (x[keysX[i]] == y[keysX[i]]) {**

**isEqual = true;**

**}**

**else {**

**isEqual = false;**

**break;**

**}**

**}**

**} else {**

**console.log('Objects are NOT equal');**

**}**

**console.log(isEqual);**

**2)Object.values()---- Extracts all values of an object and store in a an array**

**const values = Object.values(apple);**

**console.log(values);**

**3) Object.entries() —-----It is going to store the keys and values in a nested array**

**[[brand, Apple], [city, chennai],....]**

**const entries = Object.entries(apple);**

**console.log(entries);**

**console.log(entries[0]); only one value we need pass the index**

**4) Object.create()**

**5) Object.assign()---- Merge two objects and creates brand new object**

**const newObj = Object.assign(car, apple);**

**console.log(newObj);**

**6) Object.freeze();**

* **I don’t want modify and adding a values I want to lock the entire object we use freeze()**
* **It doesn't allow modifying or adding a new property into an object.**

**const count = {**

**a: 1,**

**b: 2,**

**c: 3**

**}**

**Object.freeze(count);**

**count.d = 4;///not allowed to add**

**count.a = 8;/////////// not allowed to modify**

**console.log(count);**

**7)Object.seal(); It is not going to add a property It allows only to modify the existing property**

**const count = {**

**a: 1,**

**b: 2,**

**c: 3**

**}**

**Object.seal(count);**

**count.d = 4;**

**count.a = 8;**

**console.log(count);**

**Array of objects:**

**const student1 = {**

**id: 101,**

**name: "Santosh",**

**city: "Chennai"**

**}**

**const data = [**

**{**

**id: 101,**

**name: "Santosh",**

**city: "Chennai",**

**friends: ['Yashas', "Murthy"],**

**acc: { a: 1, b: 2 }**

**},**

**{**

**id: 102,**

**name: "Praveen",**

**city: "Bangalore",**

**friends: ['Chetan'],**

**acc: { a: 3, b: 4 }**

**},**

**{**

**id: 103,**

**name: "Pooja",**

**city: "Delhi",**

**friends: ['Rakshitha'],**

**acc: { a: 5, b: 6 }**

**}**

**]**

**// console.log(data[0].name);**

**for (let i = 0; i < data.length; i++) {**

**console.log(data[i].acc.a);**

**}**

## **MATH and DATE objects**

**Math**

**The JS math Object provides several constants and methods to perfor, mathematical operation.**

**const price = -204.56; // 205 , 204**

**console.log(Math.PI);**

**console.log(`Round of ${price} is ${Math.round(price)}`);**

**console.log(`Ceil of ${price} is ${Math.ceil(price)}`);**

**console.log(`Floor of ${price} is ${Math.floor(price)}`);**

**console.log(`Absolute of ${price} is ${Math.abs(price)}`);**

**// console.log(price < 0 ? price \* -1 : price);**

**console.log(`Power of 10, 4 is ${Math.pow(10, 4)}`);**

**console.log(`Square root of 16 is ${Math.sqrt(16)}`);**

**console.log(`Maximum number of 22, 34, 12, 15 is ${Math.max(22, 34, 12, 15)}`);**

**console.log(`Minimum number of 22, 34, 12, 15 is ${Math.min(22, 34, 12, 15)}`);**

**console.log(`Truncate of 12.24 is ${Math.trunc(12.24)}`);**

**console.log(`Random number is ${Math.random()}`);**

**console.log(Math.trunc(Math.random() \* 10) + 5);**

**let max = 0;**

**const arr = [22, 34, 12, 15];**

**for (let i = 0; i < arr.length; i++) {**

**if (arr[i] > max || arr[i + 1] < arr[i]) {**

**max = arr[i]**

**}**

**}**

**console.log(max);**

**Date**

**const today = new Date();**

**console.log(`Today is ${today}`);**

**//getter methods of date object**

**console.log(`Today number is ${today.getDay()}`);**

**const days = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'];**

**console.log(days[today.getDay()]);**

**console.log(`Month number is ${today.getMonth()}`);**

**console.log(`Current full year is ${today.getFullYear()}`);**

**console.log(`Current time is ${today.getTime()}`);**

**console.log(`Current hour is ${today.getHours()}`);**

**console.log(`Current minutes is ${today.getMinutes()}`);**

**console.log(`Seconds is ${today.getSeconds()}`); \*/**

**setter**

**const date = new Date('2002.06.30').getTime();**

**console.log(date);**

**const date2 = new Date();**

**date2.setFullYear('2000');**

**console.log(date2);**

**const startDay = new Date('2023.06.30').getTime();**

**const endDay = new Date('2023.06.01').getTime();**

**if (startDay > endDay) {**

**console.log('Valid Selection');**

**} else {**

**console.log('Invalid date');**

**}**

**1. Generate 10 digit Alpha-numeric coupon AY7E35L3TI.**

**2. Wap to calculate the age of a person based on his DOB till today**

# **FUNCTIONS**

**It's a subroutine containing a set of statements / blocks of code, which perform some specific task. functions can be reusable.**

**Rules while creating functions**

**1. It shouldn't start with numbers**

**2. It shouldn't be a reserved keyword. var, if, for , switch, new...**

**3. It shouldn't contains any special characters other than underscore(\_)**

**Benefits of Using a Function**

* **Function makes the code reusable. You can declare it once and use it multiple times.**
* **Function makes the program easier as each small task is divided into a function.**
* **Function increases readability.**
* **Function expression is useful in passing callback function, creating a closure or IIFE**

**Function declaration without parameters:**

**syntax:**

**function <function\_name> () {**

**//code to be executed**

**}**

**<function\_name>() -> calling function**

**Example:**

**let count = 0;**

**function greet() {**

**count++;**

**alert(`${count} - Hello World`)**

**}**

**greet();**

**greet();**

**greet();**

**A function can have one or more parameters, which will be supplied by the calling code and can be used inside the function.**

**Argument object**

**all the functions in JS can use arguments object by default. An argument object includes the value of each parameter.**

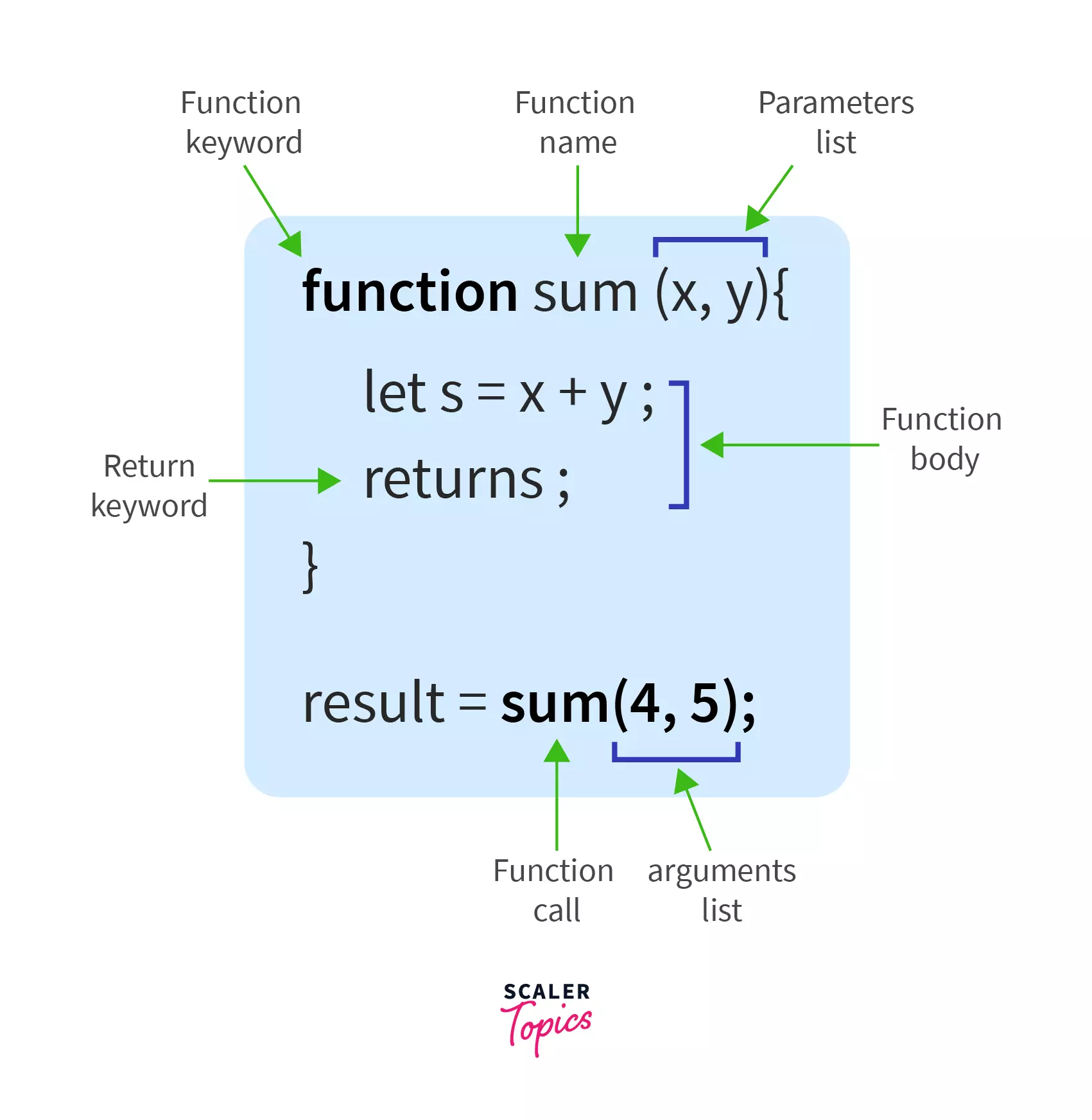
**Function declaration with parameters:**

**Syntax:**

**function <function\_name>(parameter1, parameter 2,.....){**

**//code to be executed**

**}**

**<function\_name>(argument1, argument2, .....)**

**When a value is passed when declaring a function, it is called a parameter. And when the function is called, the value passed is called argument.**

**EXAMPLE:**

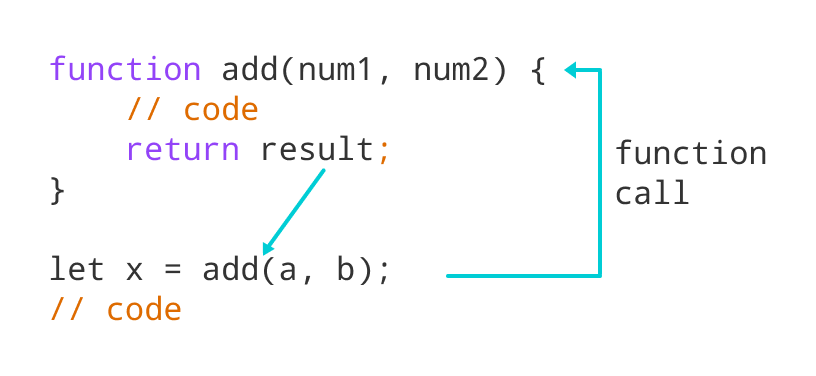
**function addition(a, b) {**

**const sum = a + b;**

**console.log(sum);**

**}**

**addition(10, 20);**

**Function with return Keyword:**

* **The return statement can be used to return the value to a function call.**
* **The return statement denotes that the function has ended. Any code after return is not executed.**
* **If nothing is returned, the function returns an undefined value.**

**function product(num1, num2) {**

**const result = num1 \* num2;**

**return result;**

**}**

**let x = product(10, 5); //50**

**let y = product(3, 2); //6**

**Function calling another function (Utility function / Helper function)**

**//Utility function**

**function cutFruit(fruit) {**

**return fruit \* 4;**

**}**

**function fruitProcessor(apples, oranges) {**

**const applePieces = cutFruit(apples); //12**

**const orangePieces = cutFruit(oranges); //20**

**console.log(`Juice with ${applePieces} pieces of apples and ${orangePieces} pieces of oranges.`);**

**}**

**fruitProcessor(3, 5);**

**2. Function expression / Anonymous function**

**Javascript allows us to assign a function to a variable and then that variable we can use as a function. It is called function expression.**

**Syntax:**

**const <variable\_name> = function (parameter1, parameter2,...) {**

**//code to be executed**

**}**

**<variable\_name>(argument1, argument2,..)**

**Example**

**const add = function (a, b) {**

**const sum = a + b;**

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

**return sum;**

**}**

**let x = add(3, 4);**

**console.log(x);**

**3. Arrow function /ES6 function**

**Arrow function is one of the features introduced in the ES6 version of JS. It allows us to create functions in a cleaner way.**

**Syntax:**

**const <variable\_name> = (parameter1, parameter2,...) => {**

**//code to be executed**

**}**

**<variable\_name>(argument1, argument2,...)**

**Example:**

**const product = (a, b) => {**

**const result = a \* b;**

**console.log(result);**

**return result**

**}**

**let y = product(3, 2);**

**console.log(y);**

**In Arrow function only one line that too return keyword no need to write curly braces and return keyword**

**const product = (a, b) => a \* b;**

**let y = product(3, 2);**

**console.log(y);**

**Nested functions:**

**function first() {**

**let str = 'Hello';**

**function second() {**

**let a = 'World'**

**let op = str + a;**

**console.log(op);**

**}**

**second();**

**}**

**first();**

**let a = 10;**

**function print() {**

**let b = 20;**

**console.log(a);**

**function test() {**

**console.log(a);**

**}**

**test()**

**}**

**print()**

**METHODS:**

**const person = {**

**name: "Jessica",// Properties**

**age: 25,**

**city: "Bangalore",**

**info: function () {**

**console.log(`My name is ${person.name}, and I live in ${person.city}`);**

**} //Method**

**}**

**let a = person.name;**

**let b = person['city'];**

**console.log(a);**

**console.log(b);**

**person.info(); // calling a method**

**person['info']()**

# **DOM(Document Object Model)**

**Whenever the browser passes an html element to our document from top to bottom it creates a tree-like structure. Each html element is an object representation which contains object properties and methods so this entire model we call Document Object Model.**

**DOM TREE:**

**EXAMPLE:**

**<html lang="en">**

**<head>**

**<title>My Document</title>**

**</head>**

**<body>**

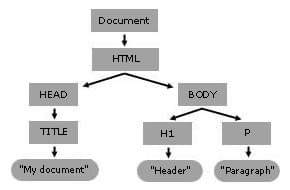
**<h1>Header</h1>**

**<p>Paragraph</p>**

**</body>**

**</html>**

**TREE LIKE STRUCTURE:**

****

**<h1 id="main-title">DOM Interaction</h1>**

**<ul>**

**<li>Item 1</li>**

**<li class="list-item">Item 2</li>**

**<li class="list-item">Item 3</li>**

**</ul>**

**There are 5 ways to select any HTML elements**

**1. getElementById()**

**2. getElementsByClassName()**

**3. getElementsByTagName()**

**4. querySelector()**

**5. querySelectorAll()**

**DOM Traversing**

**<figure id="figure">**

**<img src="https://www.achieversit.com/assets/images/logo-white.png" alt="ait">**

**<figcaption>AchieversIT</figcaption>**

**</figure>**

**<ul>**

**<li>Angular</li>**

**<li>React</li>**

**<li>Node</li>**

**</ul>**

**<ul>**

**<li>Angular</li>**

**<li>React</li>**

**<li>Node</li>**

**</ul>**

**<p class="para">Lorem ipsum dolor, sit amet consectetur adipisicing elit. Tenetur, fugit.</p>**

**<input type="text" value="Default">**

**<hr>**

**<button>Add color</button>**

**<h1></h1>**

**<script>**

**Selectors - Which helps us to select an element.**

**1) GetElementById() - we can select element id attribute**

**document.getElementById('figure').style.border = '1px solid red';**

**let isColor = false;**

**const btn = document.querySelector('button');**

**btn.addEventListener('click', () => {**

**if (!isColor) {**

**document.getElementById('figure').style.border = '1px solid red';**

**isColor = true;**

**} else {**

**document.getElementById('figure').style.border = 'none';**

**isColor = false;**

**}**

**})**

**2)GetElementsByTagName() - We can select an element by its tag name.**

**document.getElementsByTagName('ul')[0].style.border = '10px dotted green';**

**3) getElementsByClassName - We can select an element by its Class name.**

**document.getElementsByClassName('para')[0].style.color = 'green';**

**4) querySelector() - we can select elements by using any selectors Which you know in css. but it returns the first selected element.**

**document.querySelector('ul li').style.color = 'red';**

**document.querySelector('#figure').style.padding = '20px';**

**document.querySelector('#figure figcaption').style.padding = '20px';**

**document.querySelector('[src]').style.width = '200px';**

**5) querySelectorAll() - we can select elements by using any selectors Which you know in css. but it returns all selected elements.**

**console.log(document.querySelectorAll('ul li'));**

**document.querySelectorAll('ul li')[0].style.color = 'orange';**

**document.querySelectorAll('ul li')[1].style.color = 'purple';**

**document.querySelectorAll('ul li')[2].style.color = 'blue';**

**const num1 = 10;**

**const num2 = 20;**

**const res = num1 + num2;**

**document.querySelector('h1').textContent = res;**

**document.querySelectorAll('ul')[1].children[1].style.color = 'pink';**

### **Creating a 4 column layout in js**

* **How do convert string into a html element in js**
* **A browser have dom built in API which will convert string into an html element**
* **Built API given by the dom itself only**

**.box {**

**border: 1px solid #ccc;**

**padding: 20px;**

**text-align: center;**

**}**

**.container {**

**display: flex;**

**}**

**.flexItems {**

**flex: 1;**

**}**

**table {**

**font-family: arial, sans-serif;**

**border-collapse: collapse;**

**width: 100%;**

**}**

**td,**

**th {**

**border: 1px solid #dddddd;**

**text-align: left;**

**padding: 8px;**

**}**

**tr:nth-child(even) {**

**background-color: #dddddd;**

**}**

**</style>**

**</head>**

**<body>**

**<section class="container box"></section>**

**<!-- Table -->**

**<table>**

**<thead>**

**<tr>**

**<th>Sl No.</th>**

**<th>ID</th>**

**<th>Name</th>**

**<th>City</th>**

**<th>Salary</th>**

**</tr>**

**</thead>**

**<tbody>**

**</tbody>**

**</table>**

**<script>**

**const students = ['Ravi', 'Shashi', 'Pooja', 'Mahesh', 'Rajesh'];**

**creating a 4 column layout in js**

**let parentEle = `<section class="container box">**

**<div class="box flexItems"> ${students[0]}</div>**

**<div class="box flexItems"> ${students[1]}</div>**

**<div class="box flexItems"> ${students[2]}</div>**

**<div class="box flexItems"> ${students[3]}</div>**

**</section>`;**

**document.getElementsByTagName('body')[0].innerHTML = parentEle;**

**Dynamically adding the values using for loop**

**let output = '';**

**for (let i = 0; i < students.length; i++) {**

**output += ` <div class="box flexItems" data-id=${i}> ${students[i]}</div>`**

**}**

**console.log(output);**

**document.querySelector("section").innerHTML = output;**

**TABLE Creation:**

**const data = [**

**{ id: 101, name: "Gokul", city: 'Chennai', salary: 25000 },**

**{ id: 102, name: "Balu", city: 'Bangalore', salary: 50000 },**

**{ id: 103, name: "Surya", city: 'Pune', salary: 20000 },**

**{ id: 104, name: "Harish", city: 'Hyderabad', salary: 30000 },**

**{ id: 105, name: "Hunaid", city: 'Mumbai', salary: 40000 },**

**{ id: 106, name: "Akash", city: 'Delhi', salary: 50000 },**

**];**

**let res = '';**

**for (let i = 0; i < data.length; i++) {**

**res += `<tr>**

**<td>${i + 1}</td>**

**<td>${data[i].id}</td>**

**<td>${data[i].name}</td>**

**<td>${data[i].city}</td>**

**<td>${data[i].salary}</td>**

**</tr>`;**

**}**

**console.log(res);**

**document.querySelector('tbody').innerHTML = res;**

**Table structure using create Element method:**

**const parentEle = document.createElement('section') // <section></section>**

**const divEle1 = document.createElement('div');**

**const divEle2 = divEle1.cloneNode();**

**const divEle3 = divEle1.cloneNode();**

**Putting child elements into parent**

**parentEle.appendChild(divEle1); // <section><div></div></section>**

**parentEle.appendChild(divEle2);**

**parentEle.appendChild(divEle3);**

**Creating a text node to place inside the divisions**

**const txt1 = document.createTextNode('Peter');**

**const txt2 = document.createTextNode('Jessica');**

**const txt3 = document.createTextNode('Robert');**

**Appending text node into parent**

**divEle1.appendChild(txt1)**

**divEle2.appendChild(txt2)**

**divEle3.appendChild(txt3)**

**document.querySelector('body').appendChild(parentEle);**

**parentEle.setAttribute('class', 'container box');**

**divEle1.setAttribute('class', 'flexItems box');**

**divEle2.setAttribute('class', 'flexItems box')**

**divEle3.setAttribute('class', 'flexItems box')**

**parentEle.setAttribute('style', 'background-color : red')**

**parentEle.removeAttribute('style')**

# **EVENTS**

**Event is an action triggered by the end user to interact with a web or mobile page.**

**There are two ways to handle events**

**1. Inline event handler**

**<button onclick="clickHandler()">Click me</button>**

**2. addEventListner method**

**function buttonClickHandler() {**

**alert("Button was clicked!");**

**}**

**const btn = document.querySelector("button");**

**btn.addEventListener("click", buttonClickHandler);**

**Timers**

* **Timers are not a part of a javascript**
* **These timers are provided by the Web API just like DOM**
* **These timers are provided by the browser**

**SetTimeout:**

**settimeout will execute the function once after the timer has finished**

**syntax:**

**setTimeout(function(){**

**//code to be executed**

**}, time in milliseconds)**

**Example:**

**setTimeout(function () {**

**alert("Timer done!");**

**}, 3000);**

**SetInterval:**

**setInterval will execute the function at regular intervals of time.**

**syntax:**

**setInterval(function(){**

**//code to be executed**

**}, time in milliseconds)**

**Example:**

**let count = 0;**

**const timer = setInterval(() => {**

**count++;**

**console.log(count);**

**if (count > 5) {**

**clearInterval(timer);**

**}**

**}, 1000);**

**Remove Eventlistner**

**function buttonClickHandler() {**

**alert("Button was clicked!");**

**}**

**const btn = document.querySelector("button");**

**btn.addEventListener("click", buttonClickHandler);**

**setTimeout(() => {**

**btn.removeEventListener("click", buttonClickHandler);**

**}, 3000);**

**Event object:**

* **Whenever event occurs javascript provides an event object**
* **Event object contains a vital information at what time event occurs position,what type of event it is, target element**
* **In order to capture the event we need to pass the parameter for the call back functions**
* **You can pass any name like abc also can work**

**const btn = document.querySelector("button");**

**btn.addEventListener("click", function (e) {**

**console.log(e);**

**});**

**Other types of events**

**1)Global events**

**Click**

**Db click**

**2) Keyboard events**

**keyup, keydown, keypress**

**3) Mouse events**

**mousehover, mouseenter, mouseleave**

**4) Page events**

**load, scroll, resize**

**5) Form events**

**submit, blur, on change, focus**

**const form = document.querySelector("form");**

**const inp = document.querySelector("input");**

**form.addEventListener("submit", (e) => {**

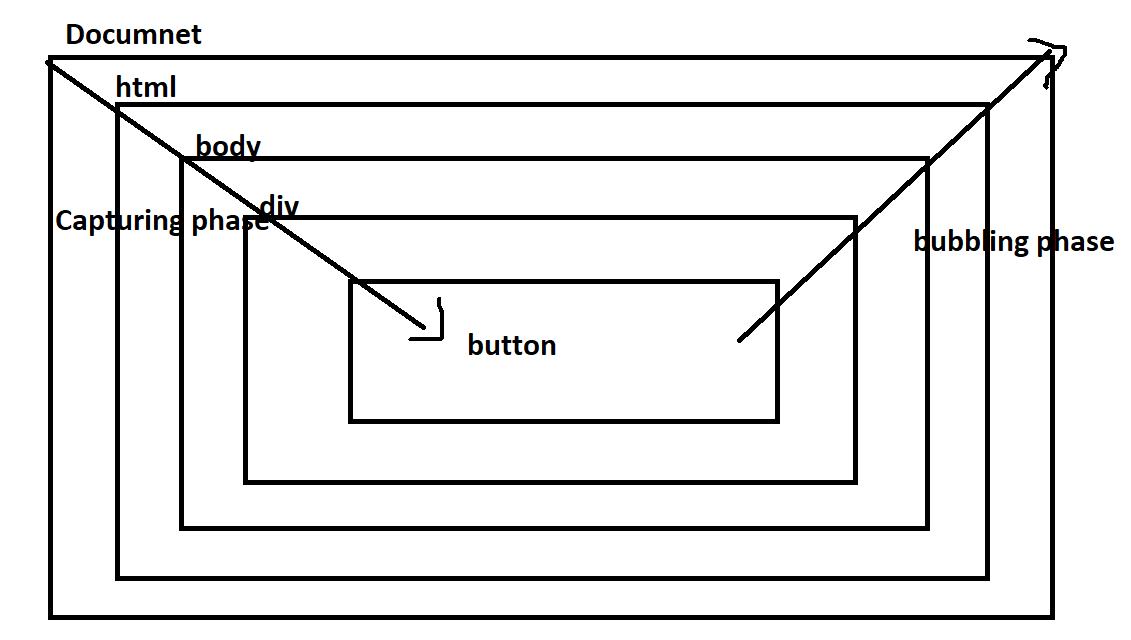
**e.preventDefault();**

**console.log(e);**

**console.log(inp.value);**

**});**

## **Event propagation(bubbling and capturing)**



* **Whenever event occurs on any element then events start propagating from the root element and it turns it to the target element**
* **Once handles the event at target element.so the signal never stays here**
* **Signal goes back to the root element,while going away if anyone of the parent having the same event handler it also handles the event at the parent also so this is called “bubbling”**
* **If you want to happens the event of the parent at the capturing phase we want pass a third argument for addeventlistener as a true called capturing phase**
* **If we want to stop the bubbling behavior of the target element pass the event object and use stoppropagation()**

**btn.addEventListener(“click”,(event)=>{**

**event.stoppropagation();**

**}**

#### **Regular expressions**

**Brackets - Defines the range of Characters**

**[abc] -> a,b or c**

**[^abc] -> any characters except a, b or c**

**[a-z] -> any characters from a to z**

**[A-Z] -> any characters from A to Z**

**[0-9]**

**[^0-9]**

**Quantifiers - Which gives info about repetition**

**[]? - occurs 0 or 1 times**

**[]+ - occurs 1 or more times**

**[]\* - occurs 0 or more times**

**[]{n} - occurs n times**

**[]{n,} - occurs n or more times**

**[]{x, y} - occurs at least x times and less than y times**

**modifiers**

**g- perform global match**

**i - it performs case insensitive match**

**m - it performs multiline matching**

**RegEx metacharacters**

**\d -> [0-9]**

**\D -> [^0-9]**

**\w ->[a-zA-Z0-9]**

**\W ->[^\w]**

**\s -> space**

**\ -> Escape character tell computer to treat following character as search character**

**\*/**

**// const pattern = new RegExp(pattern, attributes);**

**const str = `JavaScript Is best`;**

**const pattern = /is/g;**

**const res = pattern.test(str);**

**console.log(res);**

**//falsy values -> 0, "", undefined, NaN, null**

**const no = "9945118446";**

**// const op = no.match(/^[8,9][0-9]{9}$/)**

**const op = /^[8,9][0-9]{9}$/.test(no)**

**console.log(op);**

# Javascript features

**1. High-level- Language:**

**Every program that runs on your computer needs some hardware resources,such as memory and the CPU to do its work. low-level languages, such as C,where you have to manually manage these resources.you have high-level languages such as JavaScript and Python, where we do not have to manage resources at all because these languages have so-called abstractions that take all of that work away from us. The language is easier to learn and to use, but the downside is that programs will never be as fast or as optimized as, for example, C programs.**

**2. Garbage collection :**

**one of the powerful tools that takes memory management away from developers is garbage-collection,which is basically an algorithm inside the JavaScript engine, which automatically removes old, unused objects.from the computer memory in order not to clog it up with unnecessary stuff.which cleans memory from time to time so that we don't have to do it manually in our code.**

**3. Interpreted or JIT compiled:**

**JavaScript is an interpreted or just-in-time compiled language.the computer's processor only understands zeros and ones. which is also called machine code. This code eventually needs to be translated to machine code. And that step can be either compiling or interpreting. This happens inside the JavaScript engine.**

**4. MultiParadigm:**

**A paradigm is an approach and an overall mindset of structuring our code, which will ultimately direct the coding style and technique in a project that uses a certain paradigm. Three popular paradigms are procedural, object-oriented, and functional programming.So procedural programming is what we've been doing so far,which is basically just organizing the code in a very linear way, and then with some functions in between. we can classify paradigms as imperative or as declarative, many languages are only procedural or only object-oriented or only functional,but JavaScript does all of it. So it's really flexible and versatile. And so we can do really whatever we want with it.**

**5. Prototype-based object-oriented :**

**Everything in JavaScript is an object, except for primitive values such as numbers, strings, etc. But arrays are just objects. Now, we can create an array and then use the push method on it, for example?Well, it's because of prototypal inheritance. Basically, we create arrays from an array blueprint, which is like a template and this is called the prototype. This prototype contains all the array methods and the arrays that we create in our code then inherit the methods from the blueprint so that we can use them on the arrays.**

**6. First-class functions**

**first-class functions, which simply means that functions are treated just as regular variables. so, we can pass functions into other functions and we can even return functions from functions. and this allows for functional-programming, which is one of the paradigms and actually not all languages have first-class functions.**

**7. Dynamic**

**We don't assign data types to variables. Instead, they only became known when the JavaScript engine executes code. Also the type of variables can easily be changed as we reassign variables. And this is basically what dynamically-typed means.**

**8. Single-threaded & Non-blocking event loop**

**That means how the JavaScript engine handles multiple tasks happening at the same time.**

**const obj = {**

**name: "Peter",**

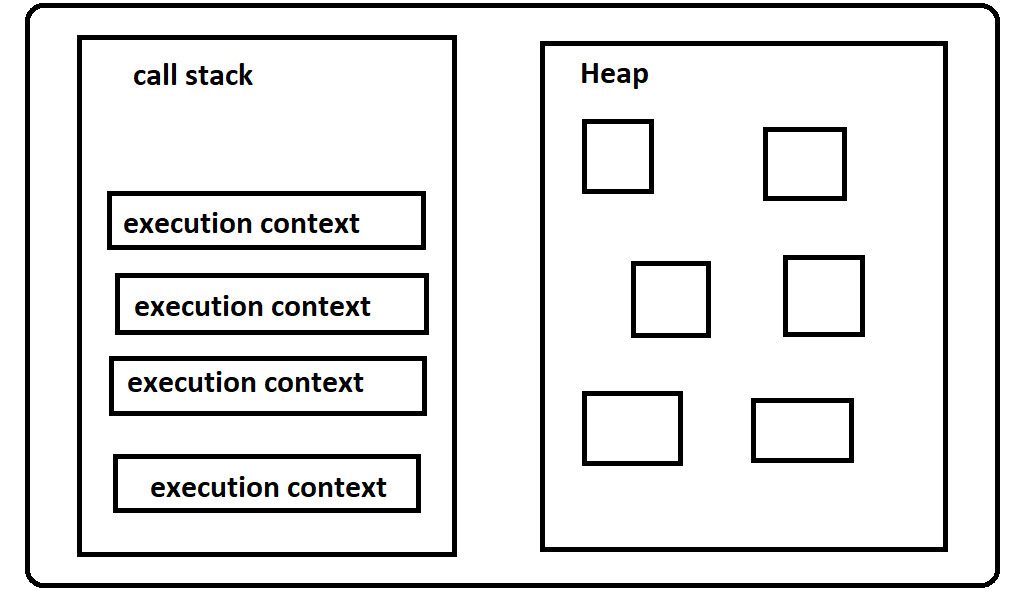
**};**

**console.log(obj);**

**let x = 10;**

**let add = function () {};**

# **Javascript engine**



**The Engine consists of two main components:**

**\* Memory Heap**

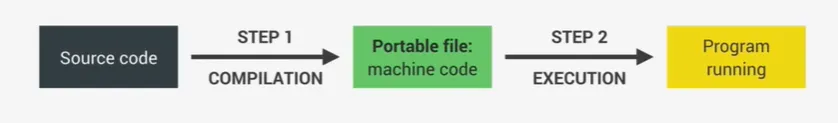
* **This is where the memory allocation happens**
* **Then the heap is an unstructured memory pool that stores all the objects that our application needs.**

**\* Call Stack**

* **This is where your stack frames are as your code executes**
* **The Call Stack is where our code is actually executed using something called execution contexts**

**Compilation:**

**So in Compilation, the entire source code is converted into machine code at once. And this machine code is then written into a portable file that can be executed on any computer. So we have two steps here. First, machine code is built and then it is executed in the CPU so in the processor. And the execution can happen way after the compilation. For example, any application that you are using on your computer right now has been compiled before and you are now executing it way after its compilation.**

****

**Interpretation**

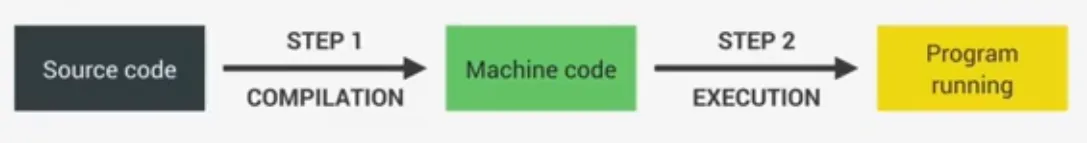
**On the other hand, in Interpretation, there is an interpreter which runs through the source code and executes it line by line. So we do not have the same two steps as before. Instead, the code is read and executed all at the same time. Of course, the source code still needs to be converted into machine code, but it simply happens right before it’s executed and not ahead of time.**

****

**Just-in-time Compilation**

**JavaScript used to be a purely interpreted language but the problem with interpreted languages is that they are much, much slower than compiled languages. This used to be okay for JavaScript, but now with modern JavaScript and fully-fledged web applications that we built and use today, low performance is no longer acceptable. Just imagine you were using Google Maps in your browser and you were dragging the map and each time you dragged it would take one second for it to move. That would be unacceptable, right?**

**How many people still think that JavaScript is an interpreted language, but that’s not really true anymore. So instead of simple interpretation, Modern JavaScript engine now uses a mix between compilation and interpretation which is called Just-in-time(JIT) compilation.**

****

**Just-in-time compilation**

**This approach basically compiles the entire code into machine code at once and then executes it right away. So we will have two steps of regular ahead of time compilation, but there is no portable file to execute. And the execution happens immediately after the compilation. And this is perfect for JavaScript as it’s really a lot faster than just execution code line by line. And now we know JavaScript has not interpreted language anymore.**

**Abstract Syntax tree:**

**Complete Architecture of browser:**

**Execution context and scope:**

**Execution context:**

* **It is an environment in which a piece of js code is executed**
* **It stores all the necessary information for some code to be executed**
* **Exactly one global context will be there in any execution context**
* **One execution context per each function**
* **Parser: A Parser or Syntax Parser is a program that reads your code line-by-line. It understands how the code fits the syntax defined by the Programming Language and what it (the code) is expected to do.**
* **JavaScript Engine: A JavaScript engine is simply a computer program that receives JavaScript source code and compiles it to the binary instructions (machine code) that a CPU can understand. JavaScript engines are typically developed by web browser vendors, and each major browser has one. Examples include the V8 engine for Google chrome, SpiderMonkey for Firefox, and Chakra for Internet Explorer.**
* **Function Declarations: These are functions that are assigned a name.**
* **function doSomething() { //here "doSomething" is the function's name**

**statements;**

**}**

* **Function Expressions: These are anonymous functions, that is functions without a function name like js function () { statements }. They are usually used in statements, like assigning a function to a variable. let someValue = function () { statements }.**
* **The Execution Context contains the code that's currently running, and everything that aids in its execution.**
* **During the Execution Context run-time, the specific code gets parsed by a parser, the variables and functions are stored in memory, executable byte-code gets generated, and the code gets executed.**
* **There are two kinds of Execution Context in JavaScript:**
  + **Global Execution Context (GEC)**
  + **Function Execution Context (FEC)**

**Global Execution Context (GEC)**

* **Whenever the JavaScript engine receives a script file, it first creates a default Execution Context known as the Global Execution Context (GEC).**
* **The GEC is the base/default Execution Context where all JavaScript code that is not inside of a function gets executed.**
* **For every JavaScript file, there can only be one GEC.**

**Function Execution Context (FEC)**

* **Whenever a function is called, the JavaScript engine creates a different type of Execution Context known as a Function Execution Context (FEC) within the GEC to evaluate and execute the code within that function.**
* **Since every function call gets its own FEC, there can be more than one FEC in the run-time of a script.**
* **How are Execution Contexts Created?**
* **Now that we are aware of what Execution Contexts are, and the different types available, let's look at how they are created.**
* **The creation of an Execution Context (GEC or FEC) happens in two phases:**
  + **Creation Phase**
  + **Execution Phase**

**Creation Phase**

* **In the creation phase, the Execution Context is first associated with an Execution Context Object (ECO). The Execution Context Object stores a lot of important data which the code in the Execution Context uses during its run-time.**

**Execution context is made up of three things**

1. **Variable Environment**
2. **Scope chain**
3. **This keyword**

**const name = 'Peter';**

**function first() {**

**let a = 1;**

**const b = second(); //2**

**a = a + b;**

**return a;**

**}**

**function second() {**

**let c = 2;**

**return c;**

**}**

**console.log(first());**

**Scope and Scope chain**

* **Scoping - How a program's variables are organized and accessed, "where do variables live? where we can access a certain variable."**
* **Lexical Scoping - Scoping is controlled by placement of functions and blocks in the code.**
* **Scope: Space or environment in which a certain variable is declared.**

**There is are 3 types of scopes**

**1. Global scope**

**2. Function scope**

**3. Block Scope(ES6)**

**Global scope**

* **Outside of any function or block**
* **Variables declared global scope are accessible everywhere**

**var x = 10;**

**let y = 20;**

**const z = 30;**

**Function scope**

* **Variables are accessible only inside function, NOT outside. local scope**

**Block scope(ES6)**

**Variables are declared with let and const can not be accessible outside the block.**

**var variables can accessible both inside and outside the block**

**if (true) {**

**var x = 10;**

**let y = 20;**

**const z = 30;**

**}**

**console.log();**

**//scope chain**

**const name = 'Peter';**

**function first() {**

**const age = 25;**

**if (age >= 20) {**

**const decade = 2;**

**var millennial = true;**

**}**

**const job = 'Doctor';**

**function second() {**

**const job = 'Engineer';**

**console.log(`${name} is a ${age}-old ${job}`);**

**}**

**second()**

**}**

**first()**

### **Hoisting & This keyword**

**Hoisting: Makes some types of variables accessible /usable in the code before they are actually declared. "Variables lifted to the top of their scope".**

**-------------------------------------------------------------------------------------------------------------**

**Hoisted Initial value Scope -------------------------------------------------------------------------------------------------------------**

**function declarations YES Actual function Block**

**var variables YES Undefined Function**

**let and const NO <Uninitialized>, TDZ Block**

**function expressions and <Depends if using var or let/const>**

**arrow function**

**console.log(x);**

**greet();**

**var x = 10;**

**function greet() {**

**console.log('Hello world');**

**}**

**console.log(a, b); // TDZ - temporal dead zone - region or space where we can't able to access variables**

**let a = 15;**

**const b = 20;**

**printMsg() // undefined()**

**var printMsg = () => {**

**console.log('Hello');**

**}**

**Recursive function - a function which calls itself until a certain condition fails.**

**let i = 0;**

**function callItself() {**

**i++;**

**console.log(i);**

**if (i < 5) {**

**callItself()**

**}**

**}**

**callItself();**

**Mutual recursion:**

**function a () {**

**b()**

**}**

**a()**

**function b () {**

**a()**

**}**

**var a = 10;**

**function greet() {**

**console.log(a);**

**var a = 20;**

**}**

**greet();**

**This Keyword:**

**Special variable that is created for every execution context. Takes the value of the owner of the function in which this keyword is used.**

**method**

**const apple = {**

**brand: "Apple",**

**model: 'iPhone 14',**

**info: function () {**

**console.log(this);**

**console.log(`${this.brand} ${this.model} was released on Sep, 2022.`);**

**}**

**};**

**apple.info();**

**simple function**

**function printMsg() {**

**console.log(this);**

**}**

**printMsg()**

**Arrow function**

**const apple = {**

**brand: "Apple",**

**model: 'iPhone 14',**

**info: function () {**

**console.log(this);**

**let arr = () => {**

**console.log(this);**

**}**

**arr()**

**console.log(`${this.brand} ${this.model} was released on Sep, 2022.`);**

**}**

**};**

**apple.info();**

**Eventlistener**

**const btn = document.querySelector('button');**

**btn.addEventListener('click', function () {**

**console.log(this);**

**})**

**Method —------> this = <Object that is calling the method>**

**Simple function ->this = undefined(strict mode) / window( sloopy mode)**

**Arrow function -> this = this of surrounding function(lexical this)**

**Event Listener -> this = <DOM element that handler is attached to>**

**new, call, apply, bind –-> It allows to set this manually**

# **Primitive vs reference data types**

**Primitive vs refs**

**let age = 30;**

**let oldAge = age;**

**age = 35;**

**console.log(age);**

**console.log(oldAge);**

**const person = {**

**name: "Peter",**

**age: 25,**

**friends: ['Charlie', 'Mark']**

**};**

**const newPerson = person;**

**Shallow copy**

**const newPerson = Object.assign({}, person)**

**newPerson.name = 'Jessica';**

**newPerson.friends.push('John')**

**console.log('Person', person);**

**console.log('New Person', newPerson); \*/**

**Deep copy**

**const newPerson = JSON.parse(JSON.stringify(person));**

**newPerson.friends.push('John')**

**console.log('Person', person);**

**console.log('New Person', newPerson);**

# **Modern Operators:**

**Destructuring arrays**

**Destructuring in JS is a simplified method of extracting multiple properties from an array by taking the structure and deconstructing it down into its own constituents through assignments by taking the syntax.**

**const city = ["Bangalore", 'Mumbai', 'Chennai', 'Pune'];**

**//old way**

**const a = city[0];**

**const b = city[1];**

**console.log(a, b);**

**ES6 version:**

**const [a, b] = ["Bangalore", 'Mumbai', 'Chennai', 'Pune'];**

**console.log(a, b);**

* **Original array doesn't effect**

**Ignoring values**

**const [x, , y] = ["Bangalore", 'Mumbai', 'Chennai', 'Pune'];**

**console.log(x, y);**

**nested array**

**const [m, [, n], , [, o]] = [1, [2, 3], 4, [5, 6, 7]];**

**console.log(m, n, o);**

**Default values**

**const [e, f, g = 3] = [1, 2, 5];**

**console.log(g);**

**Destructuring Objects**

**const person = {**

**name: "Praveen",**

**father: 'Ramesh',**

**hobbies: ["Sports", 'Reading', 'Travelling'],**

**friends: ["Sachin", 'Swathi']**

**};**

**old way**

**const a = person.name;**

**console.log(a);**

**const { name, friends, father } = person;**

**console.log(name, friends, father);**

**Renaming the variables**

**const { name: firstName, father: parent } = person;**

**console.log(firstName, parent);**

**Default values**

**const { name, father, mother = 'Laxmi' } = person;**

**console.log(name, father, mother);**

**Spread operator**

**Spread operator takes all the list of elements from an array or object and places them inside the another datatype.**

**const fruits = ["Orange", 'Mango', 'Apple'];**

**const nums = [1, 2, 3, ...fruits];**

**console.log(nums);**

**const newFruits = [...fruits];**

**console.log(fruits);**

**newFruits.push('Kiwi')**

**console.log(newFruits);**

**const apple = {**

**brand: "Apple",**

**model: 'iPhone 14'**

**};**

**const other = { a: 1, b: 2, ...apple };**

**console.log(other);**

**Rest pattern and parameters**

**const countries = ['India', 'UK', 'Japan', 'France', 'China'];**

**const [a, b, ...other] = ['India', 'UK', 'Japan', 'France', 'China'];**

**console.log(a, b);**

**console.log(other);**

**const num = { x: 1, y: 2, z: 3 }**

**const { x, ...left } = num;**

**console.log(x);**

**console.log(left);**

**const arr = [1, 2, 3, 4, 5];**

**function printMsg(...items) {**

**for (let i = 0; i < items.length; i++) {**

**console.log(items[i]);**

**}**

**}**

**printMsg(...arr)// printMsg(1,2,3,4,5)**

**Short circuiting**

**The || operators are used to combine two boolean values. it will return the first truthy value of all operands, or simply the last value if all of them are falsy.**

**falsy values -> 0, "", undefined, null, NaN**

**console.log(3 || "Hello");**

**console.log("" || "Welcome");**

**console.log(true || 0);**

**console.log(undefined || null);**

**console.log(undefined || 0 || "" || 'Hello' || 23 || null);**

**const car = {**

**make: "Honda",**

**// kms: 10**

**};**

**old way**

**//guard clause**

**if (car.kms) {**

**console.log(`The distance traveled by the car is ${car.kms}`);**

**}**

**console.log(`The distance traveled by the car is ${car.kms || '-NA-'}`);**

**AND**

**The && operator will return false as soon as it gets any falsy value and will return last true value if all values are truthy.**

**console.log(0 && "Hello");**

**console.log(7 && 'welcome');**

**console.log(23 && 'JS' && false && 100 && 0);**

**const car = {**

**make: "Honda",**

**kms: 10**

**};**

**car.make && console.log(`The car brand is ${car.make}`);**

**car.kms && console.log(`The distance traveled by car is ${car.kms}kms`);**

**NUllish coalescing(ES 2020) ??**

**only null and undefined is false.**

**console.log(`The distance traveled by car is ${car.kms ?? car.kms} kms`)**

**optional chaining(?.)**

**const restaurant = {**

**name: "Taj",**

**openingHours: {**

**thu: {**

**open: 12,**

**close: 22,**

**},**

**fri: {**

**open: 11,**

**close: 23,**

**},**

**sat: {**

**open: 0,**

**close: 24,**

**},**

**},**

**};**

**// if(resturant.openingHours.mon){**

**// console.log(resturant.openingHours.mon.open);**

**// }**

**console.log(resturant.openingHours?.mon?.open);**

**console.log('remaining code execu.....');**

**const days = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat']**

**for (let i = 0; i < days.length; i++) {**

**console.log(`${days[i]} ${resturant.openingHours[days[i]]?.open ?? 'Closed'}`);**

**}**

**Sets**

**A set is a collection of items which are unique, i.e no element can be repeated.**

**Set in ES6 is ordered: Elements of the set can be iterated in insertion order. Set can store any type of values.**

**const newSet = new Set(['India', 'Italy', 'UK', 'Australia', 'India', "Italy"]);**

**console.log(newSet);**

**size - It returns the number of unique elements**

**console.log(newSet.size);**

**has -It returns true if the specified value present in the set**

**console.log(newSet.has('UK'));**

**add - it adds the new element with a specified value at the end of the set**

**newSet.add('Japan');**

**console.log(newSet);**

**delete - It deletes an element with the specified value from the set object.**

**newSet.delete('Italy');**

**console.log(newSet);**

**remove all the elements from the set.**

**newSet.clear();**

**console.log(newSet);**

**Remove duplicates from an array**

**const arr = [10, 20, 10, 30, 40, 20, 50];**

**const unq = [...new Set(arr)];**

**// console.log(Array.from(unq));**

**console.log(unq);**

**Maps**

**Map is a collection of elements where each element is stored as a key, value pair.**

**Map objects can hold both objects and primitive values as either key or value.**

**const person = new Map();**

**person.set('name', 'Jessica');**

**person.set(23, 'age');**

**person.set('job', 'Engineer');**

**person.set(true, 'hasDL')**

**console.log(person);**

**size**

**console.log(person.size);**

**get**

**const a = person.get(23);**

**console.log(a);**

**delete**

**console.log(person.delete(true));**

**console.log(person);**

**clear**

**person.clear();**

**console.log(person);**

# Advanced Functions

**Default parameters**

**we can assign a parameter a default value that references to other default parameters.**

**function add(a, b, c = 2) {**

**return a + b + c;**

**}**

**console.log(add(5, 3, 1));**

**First-class functions**

**let x = 10;**

**let printMsg = function () { };**

* **JS has a fundamental property called first class functions. which is helps to write higher order functions.**
* **JS treats functions as first class citizens.**

**Functions are treated as simple values.**

* **functions are another type of object.**

**const obj = {**

**name: "peter",**

**info: function () { }**

**}**

**//Passing a function as an argument to another function**

**let x = 5;**

**const fn = function () {**

**console.log('Passed function');**

**}**

**function printNum(val) {**

**// console.log(val); // val = function () { }**

**val()**

**}**

**printNum(fn);**

**Function returns another new function**

**let x = 5;**

**const fn = function () {**

**console.log('Passed function');**

**}**

**function printNum(val) {**

**return val;**

**}**

**let res = printNum(fn);**

**res()**

**Higher-order function**

**A function that receives another function as an argument, that returns a new function, or both.**

**function clickHandler() {**

**console.log('button was clicked')**

**}**

**const btn = document.querySelector('button');**

**btn.addEventListener('click', clickHandler)**

**Callback function**

**The function which is passed as an argument into another function is called callback.**

**addEventListener is a Higher order function**

**clickHandler is callback function**

**'Javascript is best' -> "JAVASCRIPT is best"**

**//callback function**

**function wordToUpper(char) {**

**const [a, ...others] = char.split(' ')**

**return [a.toUpperCase(), ...others].join(' ')**

**}**

**//Higher-order function**

**function textTrans(str, fn) {**

**console.log(`Output is : ${fn(str)}`);**

**}**

**textTrans('Javascript is best', wordToUpper);**

# Closures ,call,apply,bind

**function parent(val) {**

**let x = 10;**

**function child() {**

**val++;**

**console.log(val);**

**console.log(x);**

**}**

**child();**

**}**

**parent(5);**

**Closure**

* **Any function always has access to the variable environment of the execution context in which the function was created even after the execution context is gone.**
* **The closure is then basically this variable environment to the function exactly at the time and place the function was created.**

**function count() {**

**let counter = 0;**

**return function () { counter++; console.log(counter); }**

**}**

**let getCount = count(); //function () { counter++; console.log(counter); }**

**getCount();**

**call, apply and bind**

**Call**

**This allows us to manually explicitly set the this keyword of any function that we want to call, then all the remaining arguments are the remaining arguments of the function**

**console.log(`-------Call--------`);**

**const samsung = {**

**brand: "Samsung",**

**model: "Galaxy S23",**

**};**

**const apple = {**

**brand: "Apple",**

**model: "iPhone 14",**

**};**

**const info = function (mon, year) {**

**console.log(**

**`${this.brand} ${this.model} was released on ${mon}, ${year}.`**

**);**

**};**

**info.call(apple, "Sep", 2022);**

**info.call(samsung, "Mar", 2023);**

**Apply**

**It's exactly the same as the call method, the only difference is that apply doesn't receive a list of arguments after the this keyword, but it's going to take any array of the argument, it will take arguments from array and pass it into functions**

**console.log(`-------APPLY--------`);**

**const appleArgs = ["Sep", 2022];**

**const samsungArgs = ["Mar", 2023];**

**info.apply(apple, appleArgs);**

**info.apply(samsung, samsungArgs);**

**info.call(apple, ...appleArgs);**

**info.call(samsung, ...samsungArgs);**

**Bind**

**Just like call method bind also allows to manually set this keyword for any function call, the difference is bind doesn't immediately call the function instead it returns a new function where the this keyword is bound so it is set to whatever the value we pass into bind.**

**// console.log(`-------Bind--------`);**

**info.call(apple, 'Sep', 2022)**

**info.call(apple, 'Aug', 2023)**

**info.call(apple, 'Oct', 2024) \*/**

**const appleBind = info.bind(apple);**

**appleBind('Sep', 2022);**

**appleBind('Aug', 2023);**

**appleBind('Oct', 2024);**

# IIFE(Immediately Invoked Function Expression)

**(function () {**

**alert('Function called')**

**})();**

**console.log('After function execution');**

**(function () {**

**var item = 0;**

**function addItem(qty) {**

**item += qty;**

**}**

**addItem(3);**

**console.log(item);**

**})();**

### **Function currying**

**function volume(a, b, c) {**

**return a \* b \* c;**

**}**

**console.log(volume(2, 3, 4)); \*/**

**function volume(a) {**

**return function (b) {**

**return function (c) {**

**return a \* b \* c;**

**};**

**};**

**}**

**let res = volume(2)(3)(4);**

**console.log(res);**

# Advanced Loops

**for-of loop**

**const fruits = ['Apple', 'Orange', 'Mango', 'Kiwi', 'Pineapple'];**

**/\* for (let i = 0; i < fruits.length; i++) {**

**console.log(i, fruits[i]);**

**} \*/**

**for (const item of fruits) {**

**console.log(item);**

**}**

**//for index**

**for (const [i, ele] of fruits.entries()) {**

**// console.log(item[0], item[1]);**

**// const item = [0, "Apple"]**

**// const [i, ele] = item;**

**if (i == 3) break;**

**console.log(i, ele);**

**}**

**for-in loop**

**const person = {**

**name: "peter",**

**age: 25,**

**city: "Bangalore",**

**country: "India"**

**};**

**/\* const keys = Object.keys(person)**

**for (let i = 0; i < keys.length; i++) {**

**console.log(person[keys[i]]);**

**} \*/**

**for (const key in person) {**

**console.log(key, person[key]);**

**}**

# Array Methods

**1. ForEach()**

**ForEach method input should be an array. It will execute the callback for each element of an array.**

**Syntax:**

**Array.forEach((current element, index, Array)=>{**

**//code to be executed**

**})**

**Example1:**

**const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];**

**const res = [];**

**nums.forEach((ele) => {**

**if (ele % 2 !== 0) {**

**res.push(ele)**

**}**

**})**

**console.log(res);**

**Example2:**

**const data = [**

**{ id: 101, val: 100 },**

**{ id: 102, val: 150 },**

**{ id: 103, val: 200 }**

**];**

**let total = 0**

**data.forEach((ele) => {**

**total += ele.val**

**})**

**console.log(total);**

**2. Map:**

**map is an array method the input takes an array as input, and returns a new array the each element of the array is mapped with new value.**

**Syntax:**

**const array = Array.map((ele, i, Array)=>{return new value})**

**Example1:**

**const nums = [1, 2, 3, 4, 5];**

**const res = nums.map((ele) => {**

**return ele \* 2;**

**})**

**console.log(res);**

**Example2:**

**const arr = ["a", 'b', 'c', 'd'] // [{a: 1}, {b : 2},{ c: 3}, {d: 4}]**

**const arrObj = arr.map((ele, i) => {**

**return { [ele]: i + 1 }**

**})**

**console.log(arrObj);**

**3. Filter:**

**Filter is an array method,it will take an array as an input and output is also going to return a brand new array.But filter method expects a boolean Expression,so it will iterate over each element of an array.**

**Whatever the elements which returns true for this expression those element s going to allow to pass remaining elements is filtered out**

**Syntax:**

**const array = Array.filter((ele, i, Array)=>{return boolean expression})**

**Example:**

**const nums = [10, 20, 30, 70, 50, 40, 80];**

**// const filRes = nums.filter((ele) => {**

**// return ele >= 50;**

**// })**

**const filRes = nums.filter((ele) => ele >= 50)**

**console.log(filRes);**

**4. Reduce**

**Syntax:**

**const <variable\_name> = Array.reduce((accumulator, current element, index, Array)=>{}, initial value of accumulator)**

**Example 1:**

**const nums = [10, 20, 30, 70, 50, 40, 80];**

**const res = nums.reduce((acc, ele) => {**

**return acc + ele**

**}, 0);**

**console.log(res);**

**Example 2:**

**const char = ['a', 'b', 'c', 'd']; // {a: 2, b: 3, c: 4, d: 5}**

**const obj = char.reduce((acc, ele, i) => {**

**acc[ele] = i + 2;**

**return acc;**

**}, {})**

**console.log(obj);**

**5.Find**

**Find method is going to find an element .It’s expect a boolean expression inside a callback function if the first element which returns for the boolean expression that element will be return as an output , if none of the element does not exists by the condition it returns undefined**

**const nums = [10, 20, 30, 70, 50, 40, 80];**

**const num = nums.find((ele) => { return ele > 50 });**

**console.log(num);**

**6.FindIndex**

**It returns the index of the find element**

**const index = nums.findIndex((ele) => ele > 50)**

**console.log(index);**

**7. Some**

**It takes input as an array,take call back function and boolean expression.If any one element satisfy the condition it returns true**

**const nums = [10, 20, 30, 70, 50, 40, 80];**

**const res = nums.some((ele) => ele > 50)**

**console.log(res);**

**8.Every**

**It takes input as an array,take call back function and boolean expression.If all the element satisfy the condition it returns true,if any one element not satisfy the given condition it returns false**

**const resEvery = nums.every((ele) => ele >= 30);**

**console.log(resEvery);**

**9.Flat**

**Here no need to pass the call back function**

**const arr = [[1, [2, 3]], [[4, 5], 6], [7, [8, 9]]]; //[1,2,3,4,5,6,7,8,9]**

**const res = arr.flat();//arr.flat(2),(3),(infinity)**

**Flat method will work only one level if the level increases deep increases then we need to provide an argument like**

**1—-----no need to pass**

**2—---pass 2**

**3—---pass 3**

**…..**

**Infinity levels we can pass the infinity**

**console.log(arr);**

**console.log(res);**

**Flat an array without using flat operator**

**const arrRes = [];**

**function recur(arr) {**

**arr.forEach((ele) => {**

**if (Array.isArray(ele)) {**

**recur(ele)**

**} else {**

**arrRes.push(ele)**

**}**

**})**

**}**

**recur(arr);**

**console.log(arrRes);**

**const temp = ['a', 'b', 'c', 'd'] // -> "1234"**

**const str = temp.reduce((acc, ele) => {**

**return acc + ele**

**}, '')**

**console.log(str);**

**10. FlatMap**

**const data = [**

**{ name: "Ramesh", transc: [10, 20, 30] },**

**{ name: "Hemanth", transc: [40, 50, 60] },**

**{ name: "Pooja", transc: [70, 80, 90] },**

**]; // [10, 20, 30, 40,...]**

**// const trans = data.map((ele) => ele.transc).flat()**

**const trans = data.flatMap((ele) => ele.transc)**

**console.log(trans);**

**11. Sort**

**const fruits = ["Kiwi", 'Apple', "Orange", 'Mango', 'Banana'];**

**fruits.sort();**

**console.log(fruits);**

**const nums = [500, -650, -230, 750, 54, -10, 75, 200]; nums.sort(function (a, b) {**

**if (a > b) {**

**return 1**

**}**

**if (a < b) {**

**return -1**

**}**

**});**

**nums.sort((a, b) => b - a)**

**console.log(nums);**

**method chaining**

**const data = [**

**{ name: "Ramesh", transc: [10, 20, 30] },**

**{ name: "Hemanth", transc: [40, 50, 60] },**

**{ name: "Pooja", transc: [70, 80, 90] },**

**];**

**const res = data.map((ele) => ele.transc).flat().reduce((acc, ele) => acc + ele, 0);**

**console.log(res);**

## 

## 