Tuesday(18/11/2025)

Identifier

- It is the name given to the variable by the programmer.

Rules :-

- It should not be start with number.
- Special character are not allowed only _ (underscore) and $ sign is allowed.
- Spaces are not allowed.
- We cannot use the reserved keyword as identifier.
- It is the case sensitive.

Operators :-

- It is used to perform some operation inside JavaScript.

1. Arithmetic Operator
2. Assignment Operator
3. Logical Operator
4. Relational Operator
5. Comparison Operator

==

- Double equal to it is used to checking the value only.

===

- Tripple equal to it is used to checking the value as well as type of the value it is also called strict type checking.

typeof :-

- It is used for checking the type of the data, which type of data you are using.
 Example :- console.log(typeof a);

Wednesday (19/11/2025)

datatype :-
---

1. Primitive Datatype :- Immutable
2. Non-primitive Datatype :- Mutable

1. Primitive Datatype

- It is immutable(which cannot be change).

```
        - number
        console.log(typeof 10);

        - String
        console.log(typeof "str");

        - Boolean
        console.log(typeof true);

        - null
        console.log(typeof null);

        - undefined
        console.log(typeof undefined);

        - bigInt
        console.log(typeof 17n);

        - Symbol
        console.log(typeof Symbol("a"));
```

2. Non-Primitive Datatype

- It is mutable(which can be change).

```
        - function
        function sum(){
        // body
}

        console.log(typeof sum);

        - array
        console.log(typeof arr = []);

        - object
        console.log(typeof obj = {});
```

null :-

- Null is treated as a value and can be store as a value to the variable.

undefined :-

- When we have not pass the data to the variable and then we try to console the variable then we will get undefined.

Symbol :-

- Symbol is used to create uniqueness in the variable.
	- Syntax :-
		let sym = Symbol("a");

Typecasting :-

- It is also known as type coiersion or conversion it means convert from one type of datatype into another type of datatype.

Type of typecasting :-
---
	1. implicit typecasting
	2. explicit typecasting

Implicit typecasting :-

- Those typecasting which will be done automatically by the JavaScript it is called implicit typecasting.
-- Example :- number and string

Explicit typecasting:-

- Those typecasting which will be done manually by the programmer, developer it is called explicit typecasting.
-- Example :- parseInt

Output methods :-

- console.log() :-
	- It is used for the testing purpose.

- document.writeln() :-
	- It is used to display the data on the UI.

- alert() :-
	- It is pop up method in JS.

- confirm() :-
	- It is pop up method to used to confirmed with user.

- prompt() :-
	- It is used to take the input from the user. Whatever the input added by the user, it converts the input to the "typecasting".

Scopes in JS :-

- It refers to current context of code.

- It specifies the accessibility and availability of the variable inside the JS code.

1. Global Scope
2. Script Scope
3. Block Scope
4. Local/Function Scope

Global Scope :-

- When we declare any variable by using var keyword then the scope of that variable will be in the global scope.

Script Scope :-

- When we declare any variable using let, const keyword then the scope of that variable will be in the script scope.

Block Scope :-

- It restricts the variable from accessing outside the block, When the variable is declare by using let, const keyword.

Local/Function Scope :-

- It restricts the variable that is declare inside the function from accessing outside the function either the variable is created by using var, let and const.

Thursday (20/11/2025)

Hoisting :-

- Moving variable and function declaration to the top before running the JS code.
- In other words, accessing the variable and function before declarartion and initialization, it is known as hoisting.

1. var :-

- If we try to access the variable before declaration and if the variable is declared by using var keyword, then we'll get undefined.

```
console.log(a);
var a = 10;
```

2. let & const :-

- If we try to access the variable before declaration and if the variable is declared by using let and const keyword, then we'll get uncaught Reference error, because it went into the TDZ(Temporal Dead Zone).

```
console.log(b);
let b = 20;
```

Friday(21/11/2025)

Decision Statement/ Conditional Statement :-
---

1. if

```
let data = true;
if(data){
        console.log("Data is accessed");
}
```

2. if-else

```
let age = parseInt(prompt("Enter your age "));
if(age >= 18){
        console.log("You can access the data!");
} else{
        console.log("You cannot access the data!");
}
```

3. else-if

```
let age = parseInt(prompt("Enter your age "));
if(age < 18){
        console.log("You cannot access the data!");
} else if(age >= 18){
        console.log("You can access the data!");
} else{
        console.log("You will not
eligible");
}
```

4. switch
5. ternary

```
(true ? console.log("condition is true") : console.log("condition is
false"));
```

falsy value :-

- 0,
-0,
null, undefined,
"", NaN,
false

Monday (24/11/2025)

Looping Statement :-

- It is block of code execute repeateadly untill condition is true.

Function :-

```
-------------
- It is reusable block of code and it is follows DRY (Do not repeat Yourself)
principal.
- It is block of code it's execute when we call it.

Argument :-
------------
- The values that are passed, while calling the function is called Argument.

Parameter :-
-------------
- The values that are accepted by the function, inside the parenthesis is called
parameter.

arguments :-
--------------
- It is object, it stores the data that are passed while calling the function.

Implicit return :-
--------------------
- If we are not returning anything from function, then function bydefault returns
undefined is known as implicit return.
Eg :-
function sum(a, b){
    console.log(a + b);
}

sum(10,20);

Explicit return :-
-------------------
- If we are explicitly returning something, from inside the function, it is called
explicit return.
Ex :-
function sum(a , b){
    console.log("Sum of Number :- ");
    return a + b;
}

sum(20,30);

Types of Function :-
---------------------
    1. Named Function :-
    ----------------------
    - A function which is having some name is known as named functioon.
        Ex :- function sum(){
        // body
}
```

```
    2. Anonymous Function :-
    -------------------------
    - A function which does not have a name is known as Anonymous function.
        Ex :- function (){
        // body
}

    3. Arrow Function :-
    ---------------------
    - It is a shorter way to writing a function code.
        Ex :- () => {}

    Properties of Arrow Function :-
    --------------------------------
        - If function having only one parameter , then there is no need of using the
parenthesis.
        - If function having more than one parameter, then we need of using the
parenthesis.
        - If function having only one Statement, then we don't need of curly braces.
        - If function return something and only one Statement returned then also we
don't need of curly braces.

Tuesday (25/11/2025)

Function with Expression :-
----------------------------
- When we store a function inside the variable, it is called function with
expression.
- The stored function that is passed to the variable is known first class variable.
        Ex :- let fun = (a, b) => {
                console.log("Function with expression");
}
fun(a)
Immediate Invoke Function Expression(IIFE) :-
-----------------------------------------------
- After creating the function immediately invoking the function is known as
Immediate Invoke Function Expression.
        Ex :- (() => {
        console.log("IIFE Function");
})()

Global Variable Voilation :-
-----------------------------

Higher Order Function :-
-------------------------
- When the function is accepting the another function and returning the function, it
is known as Higher order function.
        Ex :- function HOF(a, b, callback){
                return callback(a, b);
```

```
}

function callback(){
        console.log("hii")
}

        HOF(10,20, (a, b) => {
                console.log(a + b);
})
```

Callback Function :-

- A function which is passed as a argument to another function, it is known as Callback function.

Default Parameter :-

- when we not define the argument inside calling the function then we have to define bydefault value inside the formal argument where function created.

Nested Function :-

- A function in which accepts another function, it is called nested function.

```
        Ex :- function grandParent(){
                        console.log("Grand Parent Function");
                        function parent(){
                                console.log("Parent Function");
                                function child(){
                                        console.log("Child Function");
                }
                                child();
        }
                        parent();
}
                grandParent();
```

JavaScript Currying :-

- When a function is returning another function it is known as JavaScript currying.

```
        Ex :- function grandParent(){
                        console.log("Grand Parent Function");
                        return function parent(){
                                console.log("Parent Function");
                                return function child(){
                                        console.log("Child Function");
                }
        }
}
                grandParent()()();
```

Closure :-

- In the nested function when the inner function is trying to access the data which is present inside the parent function then closure will be created.
- It is an object.
- When the inner function access the variable which is present outside the inner function then closure will accept that necessary data which is used in inner function.

```
        Ex :- function grandParent(){
                        console.log("Grand Parent Function");
                        let a = 10;
                        return function parent(){
                                console.log("Parent Function");
                                console.log(a);
                                return function child(){
                                        console.log("Child Function");
                }
        }
}
                grandParent()()();
```

Recursion Function :-

- A function which calling itself.

```
        Ex :- function sum(n){
                if(n === 1){
                        return 1;
        } else{
                        return n + sum(n-1);
        }
}

        console.log(sum(5));
```

Generative Function :-

- A generative function returns multiples values.
- next() :- The generative function have next() method, It is an object stores two values, The values which is yielded by the generative function.
- done :- It return the boolean value indicating whether the generative function has the execution is completed or not.
- When we try to access with only function name then it will return one object "genFun {<suspended>}".
- When we try to access after storing the function inside the variable and then we get the right output.

```
        Ex :- function* genFun(){
    yield "Pradumn";
    yield "Soni";
    yield 100;
}
```

```
let res = genFun();
console.log(res.next());
console.log(res.next());
console.log(res.next());
console.log(res.next());

String :-
-----------
- It is sequence of character enclosed with single or double quotes.

Ways of creating the String :-
------------------------------
        1. let str = 'Pradumn';
        2. let str1 = "Pradumn";
        3. let str2 = `Pradumn`; :- backtick/ template literal
        4. let str3 = String(Pradumn);
        5. let str4 = new String(Verma);

String Interpolation :-
-------------------------
- Embedding variable with expression, it is called String interpolation.
 Ex :- let subject = "JavaScript";
                console.log(`${subject} is object based programming language.`);

Properties of Backtick :-
--------------------------
-

length :-
----------
- it is used find the length of String.

Method for extracting single character from string :-
-----------------------------------------------------
        Ex :- let str = "Hello World";

        - Property access
        console.log(str[6]);

        charAt() :-
        ------------
        - Return the character at the specified index.
        console.log(str.charAt(6));

        charCodeAt() :-
        -----------------
        - Returns the unicode value of the character at the specified location.
        console.log(str.charCodeAt(6));
```

```
        at() :-
        --------
        - Returns a new String consisting of the single UTF-16 code unit located at
the specified index.
        console.log(str.at(6));

Methods for extracting sequence of character from the string :-
---------------------------------------------------------------
        1. slice :-
        -------------
        - slice(start, end) :- Returns a section of a string.
        - Cannot change the original string.
                Ex :- console.log(str.slice(0, 5));

        2. substring
        - substring(start, end) :- Returns the substring at the specified location
within a string object.
                - Negative index converted into zero (0).
                - If start index, is greater than end index, then indexes are
swapped.

Methods for converting case of string :-
----------------------------------------
        1. toLowerCase() :- Convert all alphabetic character in a string to
lowercase.
                Ex :- console.log(str.toLowerCase());

        2. toUpperCase() :- Convert all alphabetic character in a string to
uppercase.
                Ex :- console.log(str.toUpperCase());

        3. concat() :- Returns a string that contains the concatenation of two or
more strings.
                Ex :- console.log(str.concat(str2, str3));

Methods for remove spaces in string :-
--------------------------------------
        1. trimStart() :- Removes the leading white space and line terminator
characters from starting of the string.
                Ex :- console.log(str.trimStart());

        2. trimEnd() :- Removes the trailing white space and line terminator
characters from the end of the string.
                Ex :- console.log(str.trimEnd());

        3. trim() :- Removes the leading and trailing white spaces and line
terminator characters from bothe ends of the string.
                Ex :- console.log(str.trim());

Method for adding string :-
```

```
---------------------------
        1. padStart(maxLength, fillString) :-
                Ex :- console.log(str.padStart(5, "a"));

        2. padEnd() :-
                Ex :- console.log(str.padEnd(5, "a"));

Method for index :-
--------------------
        1. indexOf() :- Returns the position of the first occurrence of a substring.
                                        - If string is not found then it returns -1.
                                        Ex :- console.log(str.indexOf("World"));

        2. lastIndexOf() :- Returns the last occurrence of a substring in the
string.
                Ex :- console.log(str.lastIndexOf("World"));

Method for checking contains or not :-
--------------------------------------
        1. includes() :- Returns true if search string appears as a substring.
                Ex :- console.log(str.includes("World"));

        2. replace() :- Replace text in a string, using a regular expression or
search string.
                Ex :- console.log(str.replace("Hello", "Hii"));

        3. replaceAll() :- Replace all instances of a substring in a string.
                Ex :- console.log(str.replaceAll("Hello", "Hii"));

        4. repeat() :- Used for repeating the string.
                Ex :- let str = "QSpiders"
                                console.log(str.repeat(5));

        5. match() :- Matches a string with a regular expression, and returns an
array containing the results of that search.
                                        If the matching character is not found then
it return null.
                        Ex :- let str2 = "I am Java Developer";
                                console.log(str2.match("am"));

        6. search() :- Find the first substring match in a regular expression
search.
                                        If the matching character is not found then
it return -1.
                        Ex :- let str2 = "I am Java Developer";
                                console.log(str2.match("am"));

Array :-
---------
- It is non primitive data type.
```

- It is used to store multiple value, in a single variable.
- It can be homogenous as well as heterogenous.

CRUD :-
----

1. Literal's way :-

- let arr = [1, 2, 3, 4] // Homogenous Array
- let arr2 = [1, "abc", null, undefined, true, ()=>{}] // heterogenous Array

2. Constructor's way :-

- let arr3 = new Array(10, 20, 30, 40)

```
console.log(arr3);

let arr4 = new Array(10);
console.log(arr4);
console.log(arr4.length);
```

3. Array.of

```
let arr5 = Array.of(10, 20, 30);
console.log(arr5);
```

Read :-

- let arr6 = [1, 2, "abc", null, "xyz"];

```
console.log(typeof arr[2]);
```

Update :-

- arr7[3] = "qwerty";

```
console.log(arr7);
```

Delete :-

- delete arr7[4];

```
console.log(arr7);
```

Methods of the array :-
---

push() :-

-> It is used for adding the element from the end of the aray
-> It affects the original array.
-> It return the length of the new updated array.

Ex:-

```
let arr = [10,20,30,40,-50]
let res = arr.push(500)
console.log(arr)
```

```
        console.log(res)

pop() :-
---------
-> It is used for removing the element from the end of the array.
-> It affects the original array.
-> It return the removed element

        Ex:-
        let arr = [10,20,30,40,-50]
        console.log(arr)        // [10,20,30,40,-50]
        ler res = arr.pop()
        console.log(arr)        // [10,20,30,40]
        console.log(res)        // -50

unshift() :-
---------------
-> It is used for adding the element from the start of the aray
-> It affects the original array.
-> It return the length of the new updated array.

        Ex:-
        let arr = [10,20,30,40,-50]
        console.log(arr)
        ler res = arr.unshift(50)
        console.log(arr)
        console.log(res)

shift() :-
-----------
- It is used for removing the element from the starting of the array.
- It affects the original array.
- It return the removed element.
        Ex :- let arr7 = [1, 20, 30, 50, 100];
        console.log(arr7);
        let res = arr7.shift(50);
        console.log(arr7);
        console.log(res);

slice() :-
-----------
- It doesn't affect the original array.
- It is used to remove elements from the array.
        Ex :- let arr = ["Pradumn", "Dinkar", "Ranjan", "Ankit", "Ayush"];
                console.log(arr);
                let res = arr.slice(2, 4);
                console.log(res);
                console.log(arr);
                let res2 = arr.slice(3);
                console.log(res2);
```

```
                console.log(arr);

splice() :-
-------------
- It is used for adding and removing as well as update the elements.
- It affects the original array.
- splice(start, deletecount, items1, items2..)
        Ex :- let arr2 = ["Pradumn", "Dinkar", "Ranjan", "Ankit", "Ayush"];

console.log(arr2);
let res3 = arr2.splice(2, 3);
console.log(res3);
console.log(arr2);

let res4 = arr2.splice(1, 1, "Mia");
console.log(res4);
console.log(arr2);

flat() :-
----------
- It is used for converting the multi-dimensional array into a single, dimensional
array.
        Ex :- let arr3 = [[[[[[1, 2, 3, "abc"]]]]]];
console.log(arr3);
let res5 = arr3.flat(Infinity);
console.log(res5);
console.log(arr3);

reverse() :-
-------------
- It is used to reversing the array. It affects the original array.
        Ex :- let arr4 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(arr4);
let res6 = arr4.reverse();
console.log(res6);
console.log(arr4);

find() :-
----------
- It return the first matching element. If the element is not found, it returns
undefined, It doesn't affect the original array.
- Returns the value of the first element in the array where condition is true,
undefined otherwise.
        Ex :- let arr5 = [10, 20, 30, 40, 50, 60];

console.log(arr5);
let res7 = arr5.find(m => m > 28)

console.log(res7);
console.log(arr5);
```

findIndex(callback) :-

- It returns the index of first matching element. If the element is not found, it returns -1. It doesn't affects the original array.
        Ex :- let arr6 = [10, 20, 30, 40, 50, 60, 70];

```
console.log(arr6);
let res8 = arr6.findIndex(m => m > 28)

console.log(res8);
console.log(arr6);
```

some(callback) :-

- Whether the specified callback function returns true for any element of an array.
- If condition was not matched by any element, it return false.
        Ex :- let arr7 = [10, 20, 30, 40, 50, 60, 70];

```
let res9 = arr7.some(m => m > 40)

console.log(res9);
```

every(callback) :-

- If all the element passed by the condition, then it return true, else if any one of the element failed to pass the condition , it return false.
        Ex :- let arr8 = [10, 20, 30, 40, 50, 60, 70];

```
let res10 = arr8.every(m => m > 40)

console.log(res10);
```

map(callback) :-

- Calls a defined callback function on each element of an array, and returns an array that contains the results.
- It doesn't modifies the original array.
        Ex :- let arr = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

```
// let res = arr.map(m => m * 10)
// console.log(res);

let res = arr.map(m => {
    return m * 10
});

console.log(res);
```

filter() :-

```
------------
- Return the element of an array that meet the condition specified in a callback
function.
- It doesn't modifies the original array.
        Ex :- let marks = [35, 40, 20, 45, 56, 79, 99];

let even = marks.filter((m) => {
    return m % 2 === 0
});

console.log(even);

reduce(callback, [initial value]) :-
-------------------------------------
- It returns single value
        Ex :- let num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let res1 = num.reduce((acc, currentVal) => {
    return acc + currentVal
}, 10)
console.log(res1);

sort() :-
----------
- It is used to sort the order of element of the array.
        Ex :- let arr2 = [10, 20, 11, 12, 23, 9, 4, 5];

let res2 = arr2.sort((a, b) => {
    return a - b
})
console.log(res2);

let res3 = arr2.sort((a, b) => {
    return b - a
})
console.log(res3)

Object :-
-----------
- It is non-primitive data type, It stores the data in the form of key-value pairs.
- Key-value pairs are seperated by colon (:).
- Key and value pair together it is known as property, two property are seperated by
comma (,).
        Ex :- //~ CRUD- Create

literals way :-
-----------------

let empDetails = {
    empId : 1,
    empName : "Pradumn",
```

```
    age : 26,
    isMarried : false,
    isDeveloper : undefined,
    skills : ["HTML", "CSS", "JS", "JAVA", {exp : "null"}],
    address : {
        pinCode : 273164,
        hNo : 423,
        street : "Sarojani Nagar",
        details : [{details : () => {
            console.log("My name is Pradumn Soni")
        }}]
    },
    printName : () => {
        console.log("Myself Pradumn Verma")
    }
}

//~ Read

console.log(empDetails);
console.log(empDetails.empName);
console.log(empDetails.skills[4].exp);
empDetails.address.details[0].details();
empDetails.printName();

//~ Update

console.log(empDetails.isMarried);
empDetails.isMarried = undefined;
console.log(empDetails.isMarried);

//~ Delete

console.log(empDetails.address);
delete empDetails.address
console.log(empDetails.address);

Constructor way :-
-------------------
- Creates an object using new keyword.
        Ex :- let obj = new Object({id : 1, sal : 20000, address : "Gorakhpur"});
console.log(obj);

Object.create() :-
-------------------
- Creates an object that has the specified prototype.
        Ex :- let obj2 = Object.create(obj);
console.log(obj2);
console.log(obj2.sal);
console.log(obj.sal);
```

```
console.log(obj2.address);
console.log(obj2.age);

Prototype :-
-------------
- It is an object. In JavaScript, every object has an internal link to another
object called it's prototype.

Prototypal Inheritance :-
--------------------------
- Prototypal inheritance is a feature in JavaScript where objects inherit properties
and methods directly from other objects, forming a "prototype chain".

        Ex :- console.log(obj2.__proto__.__proto__);
console.log(String.prototype);
console.log(Array.prototype);
console.log(Object.prototype);

Constructor Function :-
------------------------
- Creates an object with function.
        Ex :- function ConstructorFn(id, name, add) {
    console.log(this);
    this.id = id;
    this.name = name;
    this.add = add;
}

let obj3 = new ConstructorFn(101, "Pradumn", "Gurugram");
console.log(obj3);

use strict :-
--------------
- The purpose of "use strict" is to indicate that the code should be executed in
"strict mode".

this :-
--------
- Pointing mechanism. It will point to the object.
- Under non-strict, inside the function, it will point to the window.
- Under strict strict mode, inside the function, it will point to the undefined.

Object.defineProperty :-
-------------------------
- It is used to create a single object property only.
        Ex :- let obj = {}
Object.defineProperty(obj, "name", {
    value : "Pradumn"
})
console.log(obj);
```

Object.defineProperties() :-

- It is used to create a multiple object property.
        Ex :- let obj2 = {}
Object.defineProperties(obj2, {
    "name" : {value : "Pradumn Soni"},
    "age" : {value : 26},
    "sal" : {value : 22000}
})
console.log(obj2);
console.log(obj2.age);
console.log(typeof (obj2.age));
console.log(typeof obj2);
console.log(typeof (obj2.name));

Object.keys() :-

- It returns all the keys present inside the object in the form of array.
        Ex :- console.log(Object.keys(obj2));

Object.values() :-

- It returns all the values present inside the object in the form of array.
        Ex :- console.log(Object.values(obj2));

Object.entries() :-

- It returns keys as well as values, in the form of nested array.
        Ex :- console.log(Object.entries(obj2));

Object.fromEntries() :-

- Returns a object created by key-value entries for properties and methods.
        Ex :- let arr = [["id", 1], ["name", "Pradumn"], ["age", 26]];
                console.log(Object.fromEntries(arr));

Object.freeze() :-

- Prevents the modification of existing property attribute and values, and prevents the addition of new properties.
        Ex :- Object.freeze(obj3);

Object.isFrozen() :-

- Returns boolean value, returns true if the object is freezed and false when the object is not freezed.
        Ex :- console.log(Object.isFrozen(obj3));

Object.seal() :-

- The modification of attribute of existing properties is allowed, and prevent the addition od new property, and deletion of existing properties.

Object.isSealed() :-

- Returns boolean value, Returns true if the object is seal and false when the object is not seal.

Object.assign() :-

- Copy the values of all of the enumerable own properties from one or more source objects to a targets object. Return the targeted object.

```
        Ex :- let obj = {
    id : 1,
    name : "Pradumn"
}

let obj1 = Object.assign({}, obj);
console.log(obj1);

let arr = Object.assign([], obj);
console.log(arr);
```

Advanced Loop in JS :-

-

Destructing in JS :-

- is a shorthand syntax that alows you to extract values from arrays or properties from objects and assign them to distinct variables in a single, concise expression.

- The JS spread operator (...) is used to expand the elements of an iterable (like an array or a string) or the properties of an object into individual elements or key-value pair.

- The JS rest parameter is a features that allows a function to collect an idenfinite number of arguments into a single, real array.

Shallow Copy :-

- Shallow copy creates a new object but shares references to nested objecrts

```
        Ex :- let obj = {
    id : 1,
    name : "Karan",
    skills : "Java",
    sal : 22000,
    companyDetails : {
```

```
        cName : "Testoyantra",
        cId : 10283,
        cLocation : "Banglore"
    }
}

~! way-1
let newObj = Object.assign({}, obj);

~! way-2
let newObj = {...obj};

~! way-3
let newObj = {}
for(let keys in obj){
    newObj[keys] = obj[keys];
}

newObj.name = "Pradumn Soni";
newObj.companyDetails.cName = "QSpiders";

console.log(obj.name);
console.log(newObj.name);
console.log(obj.companyDetails.cName);
console.log(newObj.companyDetails.cName);

Deep Copy :-
--------------
- In JS, Deep copy creates a completely independent replica of the entire object
structure, including all nested elements.
        Ex :- let obj = {
    id : 1,
    name : "Karan",
    skills : "Java",
    sal : 22000,
    companyDetails : {
        cName : "Testoyantra",
        cId : 10283,
        cLocation : "Banglore"
    }
}
// way 1
let newObj1 = structuredClone(obj);

// way 2
let newObj1 = JSON.parse(JSON.stringify(obj));

newObj1.name = "Pradumn Soni";
newObj1.companyDetails.cName = "QSpiders";
```

```
console.log(obj.name);
console.log(newObj1.name);
console.log(obj.companyDetails.cName);
console.log(newObj1.companyDetails.cName);

call() :-
----------
- The call() method is used to invoke a function immediately while explicitely
specifying what the this keyword should refer to, It accepts arguments individually
(comma-seperated).
        Ex :- let person1 = {
    eName : "Pradumn"
}

let person2 = {
    eName : "Dinkar"
}

let person3 = {
    eName : "Ranjan"
}

let person4 = {
    eName : "Ankit"
}

function makeBill(sal, add){
    console.log(`Name is ${this.eName} and salary is ${sal} and address is ${add}`);
}

//~! call
makeBill.call(person1, 22000, "Gurugram");
makeBill.call(person2, 23000, "Noida");
makeBill.call(person3, 32000, "Gurgaon");
makeBill.call(person3, 25000, "Bajaj");

apply() :-
-----------
- The apply() method is similar to call(), but it invokes the function by passing
the arguments as an array ( or array-like object). It also executes the function
immediately and allows you to explicitely set the value of this.
        Ex :- let person1 = {
    eName : "Pradumn"
}

let person2 = {
    eName : "Dinkar"
}

let person3 = {
```

```
    eName : "Ranjan"
}

let person4 = {
    eName : "Ankit"
}

function makeBill(sal, add){
    console.log(`Name is ${this.eName} and salary is ${sal} and address is ${add}`);
}

//~! apply
makeBill.apply(person1, [ 22000, "Gurugram" ]);
makeBill.apply(person2, [ 23000, "Noida" ]);
makeBill.apply(person3, [ 32000, "Gurgaon" ]);
makeBill.apply(person3, [ 25000, "Bajaj" ]);

bind() :-
----------
- The bind() method does not executes the function immediately. Instead, it returns
a new function with a permanently assigned this value and optional preset arguments.
The returned function can be invoked later.
Ex :- let person1 = {
    eName : "Pradumn"
}

let person2 = {
    eName : "Dinkar"
}

let person3 = {
    eName : "Ranjan"
}

let person4 = {
    eName : "Ankit"
}

function makeBill(sal, add){
    console.log(`Name is ${this.eName} and salary is ${sal} and address is ${add}`);
}
//~! bind
let p1 = makeBill.bind(person1, [ 22000, "Gurugram" ]);
p1();
let p2 = makeBill.bind(person2, [ 23000, "Noida" ]);
p2();
let p3 = makeBill.bind(person3, [ 32000, "Gurgaon" ]);
p3();
let p4 = makeBill.bind(person3, [ 25000, "Bajaj" ]);
p4();
```

```
Math Object :-
---------------
- Math.PI
- Math.E
- Math.abs()
- Math.ceil()
- Math.floor()
- Math.trunc()
- Math.pow()
- Math.sqrt()
- Math.cbrt()
- Math.random()

JS Engine :-
--------------
.js file ----> parser ----> AST ----> Interpreter ----> profile ----> JIT Compiler


//! BOM - Bom stands for browser object model when we open the browser the browser
it self consider as an object this object is known as BOM. the another name of bom
is window. window is the global object in the fronted javascript
//~1.Document
//~2.Location
//~3 Navigator
//~4.History
//~5.Screen

//~function
//-window.location.reload
//-window.location
//-window.history
//-window.screen
//-window.navigator
//-window.document

//! navigator.onLine : is checking the internet is connected or not.
//if(navigator.onLine){
//in navigator.onLine : internet is connected then give true or not connected give
false.
//}

//! DOM - Dom stand for documnet object model
// when we execute html file inside the browser, browser create a tree like
structure of our html file this tree like structure is known as dom tree. the dom
tree is created to manipulate the html. manipulate it means the adding the element,
removing the element, adding the attribute, removing the attribute, adding the
style, removing the style.

// document.title
```

```
// document.body
// document.children

//! Method for accessing html elements :

//* 1.document.getElementById(" ")
//* 2.document.getElementsByClassName(" ")
//* 3.document.getElementsByTagName(" ")
//* 4.document.querySelector(" ")
//* 5.document.querySelectorAll(" ")

//! 1.document.getElementById : this method is used for targeting the html element
based on the id of the element.
// let data = document.getElementById("head")
// console.log(data)

//! 2.document.getElementByClassName(" ") : this method is used for targeting the
html element based on the class name and it return's all the matching element class
name in one array like object that is html collection.

// let data = document.getElementsByClassName("abc")
// console.log(data)

//! 3.document.getElementsByTagName(" ") : this method is used for targeting tha
html element based upon their tag name.it return's all the targeted elements in the
html collection

// let data = document.getElementsByTagName("h5")
// console.log(data)

//! 4.document.querySelector() : this method is used for targeting the html element
based upon id, class, tag name.it return's only the first matching element and we
have to write the symbol of id and class name.

// let data = document.querySelector("#head")
// console.log(data)

// let data = document.querySelector(".abc")
// console.log(data)

// let data = document.querySelector("h5")
// console.log(data)

//! 5.document.querySelectorAll(" ") : this method is used for targeting the html
element based upon id, class, tagname and it return's all the matching element in
the node list which is a array like object.

// let data = document.querySelectorAll("#head")
// console.log(data)
```

```
// let data = document.querySelectorAll(".abc")
// console.log(data)

// let data = document.querySelectorAll("h5")
// console.log(data)



16/12/2025
============

getAttribute() :-
------------------
- It is a method for accessing the html attributes.
        Ex :- console.log(inp.getAttribute("type"));
console.log(inp.getAttribute("placeholder"));

setAttribute() :-
------------------
- It is a method for adding the html attributes.
        Ex :- inp.setAttribute("type", "radio");

innerHTML :-
--------------
- It gives the content as well as the tags in which the content is present.
        Ex :- let head = document.querySelector("h1")

console.log(head.innerHTML)

innerText :-
-------------
- It gives only the content.
        Ex :- let head = document.querySelector("h1")
console.log(head.innerText)

textContent :-
---------------
- It gives the content with the extra spaces, as it is written in the HTML.
        Ex :- let head = document.querySelector("h1")
console.log(head.textContent)

classList.add() :-
--------------------
- Used for adding the class.
        Ex :- head.classList.add("heading")

classList.remove() :-
----------------------
- Used for removing the class.
        Ex :- head.classList.remove("heading")
```

classList.toggle() :-

- If class is already present, it removes the class, if class is not present, it adds the class.
    Ex :- head.classList.toggle("heading")

Event Listener :-

- Event Listener are the special attribute in the JavaScript, which allows us to call the function based on the user action. All the event handlers are prefix with (on).
- Some of the event handlers are :-
    1. clickEvent :- onclick, ondblclick
    2. mouse Event :- onmousehover, onmousemove, onmouseup, onmousedown
    3. Input Event :- onfocus, onblur, onchange, onselect
    4. Keyboard Event :- onkeyup, onkeydown, onkeypress
    5. Form Event :- onsubmit