JavaScript Basics

<script type="text/javascript">//works

//NOTE:Old JavaScript examples may use a type attribute: <script type="text/javascript">.
//The type attribute is not required. JavaScript is the default scripting language in HTML.
//Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
// The ECMAScript specification is a standardized specification of a scripting language developed by Brendan Eich of Netscape; initially named Mocha, then LiveScript, and finally JavaScript. In December 1995, Sun Microsystems and Netscape announced JavaScript in a press release.

JavaScript has 8 Datatypes

1. String 2.Number 3.Bigint 4.Boolean 5.Undefined 6.Null 7.Symbol 8.Object

NOTE: 1. String 2.Number 3.Bigint 4.Boolean 5.Undefined 6.Null 7.Symbol are 7 types of primitive data types. Primitive values are immutable (they are hardcoded and cannot be changed).

eg.if x = 3.14, you can change the value of x, but you cannot change the value of 3.14.//you need to reassign the new value but old one will be recessive but it would be still existing ,can't destroy that value.

JavaScript Objects are Mutable(Objects are mutable: They are addressed by reference, not by value and it can be deleted and updated)

NOTE:JavaScript evaluates expressions from left to right. Different sequences can produce different results:

let x = 16 + 4 + "Volvo";//20Volvo // JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".
let x = "Volvo" + 16 + 4;//Volvo164 //since the first operand is a string, all operands are treated as strings.

JavaScript has dynamic types. This means that the same variable can be used to hold different data types: Let can change datatype but const will give error

```
let x;       // Now x is undefined
x = 5;       // Now x is a Number
x = "John";  // Now x is a String
```

//Comparing two JavaScript objects always return false.all equator results false even for same value.

Note: below variables and functions have more priority in case of same name

This table shows the result of converting different JavaScript values to Number, String, and Boolean:

Original Value	**Converted to Number**	**Converted to String**	**Converted to Boolean**
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	**true**

"000"	0	"000"	**true**
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	**0**	""	**false**
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	**0**	""	true
[20]	**20**	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true
{ }	NaN	"[object Object]"	true
null	**0**	"null"	false
undefined	NaN	"undefined"	false

Values in quotes indicate string values.

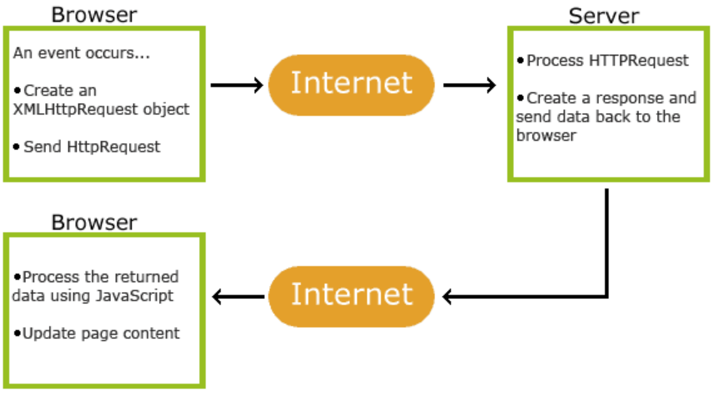
Red values indicate values (some) programmers might not expect.

NOTE:</script> can't be commented in javascript , it will have it's usual meaning even trying to comment it

Difference Between var, let and const

	Scope	Redeclare	Reassign	Hoisted	Binds this
var	No	Yes	Yes	Yes	Yes
let	Yes	No	Yes	No	No
const	Yes	No	No	No	No

How AJAX Works


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

XMLHttpRequest Object Properties

Property	Description
onload	Defines a function to be called when the request is received (loaded)
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

```
//numeric type variable
  var a = 10;
```

```
//string type variable
  var e = "hello world";
```

```
//array type variable
  var i = [1, 2, 3, 4, 5];
```

```
//null type variable
  var m = null;
var d = 40.5;
typeof 3.14           // Returns number
  typeof 33           // Returns number
  typeof (33 + 66)    // Returns number
  typeof NaN
//Returns number
```

//Integer Precision

//JavaScript has a limit on the precision of integers. This limit is 2^53 - 1. This means that any integer larger than this limit will be rounded to this limit. This is known as the safe integer limit.

```
let xs = 9007199254740992 === 9007199254740993;//true as it hasrounded the limit
```

MAX_SAFE_INTEGER is a constant that represents the maximum safe integer value in JavaScript. This value is +(2^53 - 1), which is 9007199254740991.

```
let max = Number.MAX_SAFE_INTEGER;
console.log(max);
```

MIN_SAFE_INTEGER is a constant that represents the minimum safe integer value in JavaScript. This value is -(2^53 - 1), which is -9007199254740991.

```
let min = Number.MIN_SAFE_INTEGER;
console.log(min);
```

//Number Size

//JavaScript Numbers are Always 64-bit Floating Point

//Value (aka Fraction/Mantissa) :52 bits (0 - 51)

//Exponent : 11 bits (52 - 62)

//Sign-1 bit (63)

//JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

//let x = 0xFF;

//JavaScript displays numbers as base 10 decimals.

//But you can use the toString()

```
var e = "Sudha Kumari" //updated
document.getElementById('name').innerText = e;
console.log(e)
// Sudha Kumari
typeof "Sudha"
// Returns string
typeof ("Sudha" + "Kumari") // Returns string
```

String() function

```
String(123); // "123"
```

1. `String(Date()); //Tue Aug 13 2024 10:28:39 GMT+0530 (India Standard Time)`

```
let xValid = 2 + 3 + "5";//5+'5'=55//check
```

NOTE: All string methods return a new string. They don't modify the original string.

Formally said:Strings are immutable: Strings cannot be changed, only replaced.

length

```
var mystring = "shanaya";
document.write("<br>", mystring.length);//7
```

indexOf

```
Syntax:str.indexOf(searchvalue,fromindex)
document.write("<br>", mystring.indexOf("a"));//2
 document.write("<br>", mystring.indexOf("a", 4));//4
document.write("<br>",
```

```
var j = ["hello", "world", "shanaya", "world"];
typeof [1, 2, 3]// Returns object
hence to recognize if it is array , typeof will not be useful so we use
```

Array.isArray(argumentArr)

```
console.log(Array.isArray([1, 2, 3])) // true
document.write("<br>", j);//[object Array]
```

For-in in an Array

```
var myarray = ["shanaya", 20, "mumbai"];
for (const key in myarray) {
    const element = myarray[key];

window.document.write("<br>", key, ":", myarray[key]);//same result as below
window.document.write("<br>", key, ":", element);//same both
  }
```

array length

```
document.write("<br>", myarray.length);//3
```

array index

```
document.write("<br>", myarray[2]);//mumbai
```

array push

```
myarray.push("Btech");
```

array pop

```
myarray.pop();//delete last element
```

array delete

```
delete myarray[0];//delete at an index
```

array shift

```
myarray.shift();
```

shift() removes the first element from an array and returns it,

It is not the same as undefined.

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

```
typeof null // Returns object
```

In JavaScript, null is a primitive value. However, typeof returns "object".

This is a well-known bug in JavaScript and has historical reasons.

```
//You can empty an object by setting it to null:
  var xDeclared = { a: 1 };
  xDeclared = null;//An object emptied by setting the value to null.
  typeof xDeclared; // Returns object , value is null means nothing
```

//**undefined type variable**

```
  var n ==undefined;
var xnotAssigned;
let xnotAssigned;
typeof xnotAssigned; // Returns undefined as value not assigned
var xDeclared = 10;
  xDeclared = undefined;//A variable emptied by setting the value to undefined.
  typeof xDeclared; // Returns undefined
```

An empty value has nothing to do with undefined because it has value

```
  var xDeclared = "";// An empty string has both a legal value and a type.
  typeof xDeclared; // Returns string
```

NOTE:The main difference between undefined and null is method to output numbers from base 2 to base 36.

//base 36 is for string//ascii//

Base36 is a binary-to-text encoding scheme that represents binary data in an ASCII string format by translating it into a radix-36 representation.

//Never write a number with a leading zero (like 07).

//Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

BigInt Hex, Octal and Binary

//hexadecimal=0x, octal=0o, bin=0b;

//The n at the end of the number

```
//bigInt
   var g = 10000000000000000000000000000000 console.log(g)//e+36
console.log(BigInt(g))//10000000000000000000000000000000
console.log(BigInt(434343344353423343432432432323))
let hex = 0x20000000000003n;
   let oct = 0o40000000000000000003n;
   let bin = 0b10000000000000000000000000000000000000000000000000011n;
 typeof 1234n         // Returns bigint
```

Exponential notation

```
   let yE = 123e5;    // 12300000//here e5 like 10^5
   let zE = 123e-5;   // 0.00123
//here e-5 like 10^-5
```

JavaScript Type Conversion

JavaScript is dynamically typed, which means you don't need to declare the type of a variable before using it. //By the use of a JavaScript function.However,

```
mystring.indexOf("a", 5));//6//after 5 index
```

lastIndexOf

```
lastIndexOf(item, start)
document.write("<br>",
mystring.lastIndexOf("a"));//6
```

concat

```
document.write("<br>",
mystring.concat(mystring))
//shanayashanaya

document.write("<br>",
mystring.concat(mystring, "mumbai"));//shanayashanayamumbai

document.write("<br>", mystring +
mystring +
"mumbai")//shanayashanayamumbai
```

String object

```
var mystring2 = new String("shanaya");
document.write("<br>", typeof
mystring2);//object
```

charAt

```
document.write("<br>", mystring.charAt
     (1));//h
```

charCodeAt

```
document.write("<br>",
mystring.charCodeAt
     (0));//115
```

codePointAt

```
document.write("<br>",
mystring.codePointAt
     (0));//115
```

push,pop,join,shift,unshift not for string

array unshift

unshift() adds one or more elements to the beginning of an array and returns the new length.

```
myarray.unshift("shanaya");
myarray.unshift();//no output change
```

array join

```
const elements = ['Fire', 'Air', 'Water'];
console.log(elements.join());
// Expected output: "Fire,Air,Water"
```

A string to separate each pair of adjacent elements of the array. If omitted, the array elements are separated with a comma (",").

```
console.log(elements.join(''));
// Expected output: "FireAirWater"
console.log(elements.join('-'));
// Expected output: "Fire-Air-Water"
```

Array.prototype.join recursively converts each element, including other arrays, to strings. Because the string returned by Array.prototype.toString (which is the same as calling join())

```
const matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9],
];
console.log(matrix.join()); // 1,2,3,4,5,6,7,8,9
console.log(matrix.join(";")); // 1,2,3;4,5,6;7,8,9

console.log([1, ,
```

that undefined is a value that is used when a variable has not been declared, or when a variable has been declared but has not been assigned a value. Null is a value that is used when a variable has been declared and assigned a value of null.

```
null === undefined // false //as data type is different , one is object and other is undefined respectively
  null == undefined // true //data type is different but in sense it has no value //has returns true
```

void Operator

The void operator returns undefined. It can be used to remove the value of an expression, preventing it from being logged to the console or assigned to a variable.//or any way printed

The void operator evaluates an expression and returns undefined. This operator is often used to obtain the undefined primitive value, using "void(0)" (useful when evaluating an expression without using the return value).

```
  document.write(void 5); // undefined
  document.write(void "Hello"); // undefined
  document.write(void function () { }); // undefined
  document.write(void (function () { }())); // undefined
  document.write(void (function () {
    return 5;
  }())); // undefined
//even if return is
```

JavaScript does have a type system, and it can automatically convert between different types. This is known as **type coercion**.//Automatically by JavaScript itself

Converting Strings to Numbers

```
Number(), parseInt(),parseFlo
at() covered in Number
datatype
toExponential(), toFixed(),to
Precision() covered in number
datatype
```

Converting Numbers to Strings

```
String() function
toString() method
already covered somewhere
```

Automatic Type Conversion

```
5 + null;  // returns 5
because null is converted to
0
("5" + null);// returns
"5null"   because null is
converted to "null"
("5" + 2);  // returns "52"
because 2 is converted to "2"
("5" - 2);  // returns 3
because "5" is converted to 5
("5" * 2);  // returns 10
because "5" and "2" are
converted to 5 and 2
```

NOTE: JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable:

```
document.write("<br>" + 5);
// outputs "5"
 document.write("<br>" +
true);  // outputs "true"
document.write("<br>" +
null);  // outputs "null"
document.write("<br>" +
undefined);  // outputs
"undefined"
document.write("<br>" +
(function () { return
"Hello"; }()));  //outputs
"Hello"//document.write("<br>
" + {toString:function()
{return "Hello";}});  //
outputs "Hello"
document.write("<br>" + [1,
2, 3]);  // outputs "1,2,3"
```

string slice

```
slice(start, end)
document.write("<br>
", mystring.slice(0,
2));//sh
```

string substr

```
substr(start,
length)
document.write("<br>
",
mystring.substr(0,
2));//sh
document.write("<br>
mystring.substr(-4):
",
mystring.substr(-4))
//substr() will
return an empty
string
```

substr() will return an empty string if you omit the second parameter and the first parameter is negative or greater than the length of the string.

substring

```
substring(start,
end)
```

Negative value is converted to 0

```
document.write("<br>
mystring.substring(0
,2):",
mystring.substring(0
,
    2))//sh
document.write("<br>
mystring.substring(-
2,2):",
mystring.substring(-
2,
    2))//sh  //-2
taken as 0
 document.write("<br>
mystring.substring(2
,-2):",
mystring.substring(2
,
    -2))//sh  //-2
taken as 0 and (2.0)
changed to (0,2)
```

```
3].join()); // '1,,3'
console.log([1,
undefined, 3].join());
// '1,,3'

const arrayLike = {
 length: 3,
 0: 2,
 1: 3,
 2: 4,
 3: 5, // ignored by
join() since length is 3
};
console.log(Array.protot
ype.join.call(arrayLike)
);
// 2,3,4
console.log(Array.protot
ype.join.call(arrayLike,
"."));
// 2.3.4

 When an array is cyclic
(it contains an element
that is itself),
browsers avoid infinite
recursion by ignoring
the cyclic reference.
const arr1 = [];
arr.push(1, [3, arr1,
4], 2);
console.log(arr.join(";"
)); // 1;3,,4;2
```

array concat

```
const array1 = ['a',
'b', 'c'];
const array2 = ['d',
'e', 'f'];
const array3 =
array1.concat(array2);
console.log(array3);
// Expected output:
Array ["a", "b", "c",
"d", "e", "f"]
document.write("<br>",ar
ray2
+array3);//shanaya,20,mu
mbaimumbai,pune,delhi
```

indexOf()

indexOf(searchElement)
indexOf(searchElement,

```
mentioned
  document.write( void
(function () { return 5; }
(5))); // undefined
```

JavaScript Objects

//Real Life Objects

// In real life, objects are things like: houses, cars, people, animals, or any other subjects.

Object Properties

// A real life car has properties like weight and color:
car.name = Fiat, car.model = 500, car.weight = 850kg,
car.color = white.
Car objects have the same properties, but the values differ from car to car.

```
Properties can be changed,
added, deleted, and some
are read only.
```

//object type variable

```
typeof { a: 1, b:
2 }//object
```

// student is object below
  var student = {
firstName:"Sana",
.lastName:"World",
    name: "shanaya",//name is property
    age: 20,
//JavaScript Object Methods by calling function as value of key.Methods are Functions stored as Properties.
 fullName: function () {
      return this.firstName + " " + this.lastName;//this refers to the student object
    }

```
 carsNested: { car1:
"Ford", car2: "BMW", car3:
"Fiat" } //nested
```

```
var myVar = { name: "Shanaya"
}  document.write("<br>" +
myVar);  // outputs "[object
Object]"
document.write(myVar.toString
()); // outputs
"name:Shanaya"
myVar.valueOf() //converts to
the actual object
{name:"Fjohn"}
myVar.valueOf().toString() //
converts to "[object
Object]"
```

and ***it is what exactly done in document.write (toString() is added)***

```
myVar.valueOf().toString().va
lueOf() //converts to the
actual object {name:"Fjohn"}
myVar.valueOf().toString().va
lueOf().toString() //converts
to "[object Object]"
```

Converting Binary to Decimal

We can convert a binary number to decimal using the parseInt() method with base 2.

```
var a = "101";
 document.writeln("<br>a in
decimal  with parseInt(a,
2) : " + parseInt(a, 2))//
Outputs:5
function bin2dec(bin)
{//bin=101
    return parseInt(bin, 2);
//return parseInt(bin,
2).toString(10);//it means
base $()_{10}$ which is same as
above
    // return parseInt(bin,
2).toString(2);//it converts
to base 2 which is binary $()_{2}$

  }
bin2dec("101")// Outputs:5
```

Converting Decimal to Binary

We can convert a decimal number to binary using the toString(2) method.

```
var a = 5;//101
  document.writeln("<br>a in
binary : " + a.toString(2))//
Outputs:101 //2 decides the
```

at()

```
document.write("<br>
mystring.at(0):",
mystring.at(0));//s
document.write("<br>
mystring.at(-2):",
mystring.at(-2));//h
//last character is
-1
```

indexing or Property Access [ ]

```
document.write("<br>
mystring[0]:",
mystring[0]);//s
document.write("<br>
mystring[-1]:",
mystring[-1]);//a
```

It is read only.

```
 str[0] = "A" gives
no error (but does
not work!) //array
is writeable but
property access is
only readable
```

toUpperCase

```
document.write("<br>
mystring.toUpperCase
():",
mystring.toUpperCase
());//SHANAYA
```

toLowerCase

```
document.write("<br>
mystring.toLowerCase
():",
mystring.toLowerCase
());//shanaya
```

trim()

```
text = "
HEllo   Sana    "
document.write("<br>
text:", text);//take
only one at start as
gap and one in
between  // text:
HEllo Sana
```

trimStart()

```
document.write("<br>
text:",
text.trimStart());//
take only one at
```

fromIndex)

```
document.write("<br>myar
ray3.indexOf('pune') :",
myarray3.indexOf("pune")
);
const array = [2, 9, 9];
array.indexOf(2); // 0
array.indexOf(7); // -1
array.indexOf(9, 2); //
2
array.indexOf(2, -1); //
-1
array.indexOf(2, -3); //
0

const array5 = [NaN];
array5.indexOf(NaN); //
-1
```

console.log([1, , 3].indexOf(undefined)); // -1

```
document.write( myarray3
.lastIndexOf("pune"));//
1 as only one pune
```

Array copyWithin()

```
The copyWithin() method
copies(overwrites) array
elements to another
position in an array and
does not change the
length of the array.
copyWithin(target,
start)
  copyWithin(target,
start, end)
// Copy to index 0 the
element at index 3
console.log(array1.copyW
ithin(0, 3, 4));
// Expected output:
Array ["d", "b", "c",
"d", "e"]
// Copy to index 1 all
elements from index 3 to
the end
console.log(array1.copyW
ithin(1, 3));
// Expected output:
Array ["d", "d", "e",
"d", "e"]
```

array slice

```
document.write("<br>",
myarray.slice(0,
2));//shanaya,20
```

  };

```
console.log( student
instanceof Object); //
true
```

console.log(student.name); `//objectName.property`

console.log(student[“name”]);`//objectName["property"]`

var ageExp="age"

```
console.log(ageExp);//
objectName[expression]
var ageProperty =
personCopyTry["age" +
"Name"];//undefined
var ageProperty =
student["age"] +
student["Name"];//defined
 console.log(student.carsN
ested.car1); // Output:
Ford
 console.log(student["cars
Nested"]["car1"]); //
Output: Ford
 var p1 = "carsNested";
  var p2 = "car2";
console.log(personCopyTry1
[p1][p2]);
```

//retrieving or Displaying data from object

It is a common practice to declare objects with the const keyword.

```
//updating or adding value
as objects muttable
```

student.name=”Sana”;

student.nameFunMet=`function () {`

```
    return (this.firstName
+ " " +
this.lastName).toUpperCase
();//toUpperCase() method
to convert a text to
uppercase
  }; //adding function as
property in object//this
refers to student(object
name);
console.log(student["nameF
unMet"]()); //SANA
WORLD //methods must have
()
console.log(student.nameFu
base value ()2
function dec2bin(dec)
{//dec=5
    return (dec >>>
0).toString(2);//5>>>0 is
Zero fill right shift
```

or

return parseInt(bin, 2).toString(2);//it converts to base 2 which is binary $()_2$

```
  }
dec2bin(-5);//
11111111111111111111111111111011
```

JavaScript Class Syntax

JavaScript classes are functions that use the class keyword and a constructor method. They are used to create blueprints for objects, which are instances of the class. Classes are a way to encapsulate data and behavior that is shared by multiple objects.

NOTE : A JavaScript class is not an object.It is a template for JavaScript objects.When you have a class, you can use the class to create objects:

//example

```
  class PersonClass {
    // constructor()
{ ... } //auto or default
//non parameterized//// If
you do not define a
constructor method,
JavaScript will add an empty
constructor method.
    constructor(name, age)
{//The class has two initial
properties: "name" and "age".
       this.name = name;
       this.age = age;
    }
  }
```

// Create a new object using the class. The new

```
start as gap and one
in between
   // text: HEllo
Sana
```

trimEnd()

```
document.write("<br>
text:",
text.trimEnd());
```

padding(): padStart() and padEnd() to support padding at the beginning and at the end of a string

```
document.write("<br>
",
mystring.padStart(10
,
'0'));//000shanaya"/
/here 0 is added at
start till the
length of mystring
become 10.
document.write("<br>
",
mystring.padEnd(10,
'x'));//shanayaxxx"/
/here x is added at
end till the length
of mystring become
10
```

Note : To pad a number, convert the number to a string first.

```
   var numb = 5;
   var numbtext =
numb.toString();
   let paddedStart =
numbtext.padStart(4,
"0");
   let paddedEnd =
numbtext.padEnd(4,
"x");
document.write("<br>
paddedStart:",
paddedStart);//outpu
t: padded: 0004
document.write("<br>
paddedEnd:",
paddedEnd);//output:
padded: 5xx
```

array splicing

The splice() method adds new items to an array.

```
document.write("<br>",
myarray.splice(2, 0,
"Lemon", "Kiwi"));//it
will not give output
here //shanaya,20,
Lemon,
Kiwi,mumbai//update the
old//The first parameter
(2) defines the position
where new elements
should be added (spliced
in).
  //The second parameter
(0) defines how many
elements should be
removed.
  //add item at index 2
onwards and others after
index will be updated
  document.write("",
myarray);//shanaya,20,
Lemon, Kiwi,mumbai
 document.write("<br>",
myarray.splice(2,
2));//Lemon,Kiwi//gives
output here//the deleted
one only
document.write("<br>",
myarray);//shanaya,20,
Kiwi,mumbai
```

The **spread operator (...)** is used to create a new array from the existing array.

```
const myarraytoSpliced =
["shanaya", 20,
"mumbai"];
  const
myarraytoSpliced1 =
[...myarraytoSpliced,
"Lemon",
"Kiwi"];//shanaya,20,mum
bai,Lemon,Kiwi//just
like concate here
```

toSpliced() method

The splice() method adds new items to an array.

The difference between the new toSpliced() method and the old splice() method is that the new method

```
nMet()); //SANA WORLD


//Deleting Properties
delete personCopyTry.age;
delete
personCopyTry["age"];


// Create a copy
  var studentCopyTry =
student;
 If you want to create a
copy of an object, you can
use the spread operator (…
  const x = { ...
student };
  or
  const x =
Object.assign({},
student);
  or
  const x =
JSON.parse(JSON.stringify(
person));
  or
  const x =
Object.create(student);
```

Displaying the Object using FOR-IN

```
 var textProp = ""
  for (var prop in
personMet) {
    textProp += prop + ":
" + student[prop] + "\n";
    textProp += prop + ":
" + student.prop +
"\n";//it will show error
as prop is an expression
not property
}

Object.values() creates an
array from the property
values:
 console.log(Object.values
(student)); // Output:
[ 'Sana', 'World',
'Shanaya',20, 'black' ]
```

```
keyword is used to
create a new object
from the class. The
object is an instance
of the class.
  var personOBJClass = new
PersonClass("Sudha Kumari",
20);
var personOBJClass2 =
new
PersonClass("Shanaya
Kumari", 18);
document.write(person
OBJClass2.name + " "
+ personOBJClass2.age
+ "<br>");//Shanaya
Kumari 18
```

The constructor method is called automatically when a new object is created.

It is used to initialize the properties of the object.It has to have the exact name "constructor"

NOTE : constructor of a class in java is with class name but in javascript construct keyword is used and its the method ,don't confused that it should have data type as it is method ,It happen is java not javaScript.

A class method is a method that belongs to the class rather than an instance of the class.Class methods are used to perform operations that are common to all instances of the class.

//instance is created by new keyword.

Class methods are created with the same syntax as object methods.//method() { ..... }

```
//Example of class method
  class CarClassMethodTry {
constructor(name, year) {
  this.name =name;
   this.year =year;
     }
```

NOTE: `put method with`

repeat()

```
string.repeat(count)
document.write("<br>
",
mystring.repeat(3));
//shanayashanayashan
aya
document.write("<br>
mystring.repeat(3 +'
' ) ",
mystring.repeat(3 +
"
"));//mystring.repea
t(3 +' ' )
shanayashanayashanay
a //it will not add
space in between
```

replace()

```
string.replace(searc
hvalue, newvalue)
```

The replace() method replaces only the first match

Returns a new string with matches of a pattern replaced by a replacement.

Case sensitive

```
document.write("<br>
",
mystring.replace("a"
, "A"));//shAnaya

let text2 = "Hello
Hello Shanaya"
document.write("<br>
",
text2.replace("Hello
", "Hi"));//Hi Hello
Shanaya
 document.write("<br>
",
text2.replace("hello
", "Hi"));//Hello
Hello Shanaya  //no
change
```

Using regular expression

```
document.write("<br>
",
text2.replace(/hello
```

creates a new array, keeping the original array unchanged, while the old method altered the original array.

```
toSpliced(start,
deleteCount, item1,
item2, /* …, */ itemN)
const months = ["Jan",
"Feb", "Mar", "Apr"];
  const spliced =
months.toSpliced(2, 1);
document.write("<br>
toSpliced : <br>",
spliced);//Jan,Feb,Apr
```

includes()

The includes() method returns true if a given element is found in a array, otherwise false

```
  array.includes(search-
item)
const fruitsInc =
["apple", "banana",
"cherry"];
  document.write("<br>",
fruitsInc.includes("appl
e"));//true
  document.write("<br>",
fruitsInc.includes("grap
es"));//false
```

find()

```
const fruitsFind =
["apple", "banana",
"cherry"];
  document.write("<br>",
fruitsFind.find(x => x
=== "banana"));//banana

const numbersFind = [4,
9, 16, 25, 29];
  let first =
numbersFind.find(myFunct
ionFind);
  function
myFunctionFind(value,
index, array) {
    return value > 18;
  }
  document.writeln("<br>
myFunctionFind" +
myFunctionFind);//return
whole function as a
value
```

```
//object constructor in
function
  document.write("<br>");
  function myobject3(name,
age, address) {
    this.name = name;
    this.age = age;
    this.address =
address;
  }
```

NOTE: wrong

```
   myobject3.myfunction =
function () {
      console.log("object
constructor");
    }//as you cannot add a
new property to an
existing object
constructor
```

To add a new property to a constructor, you must add it to the constructor function:

The `Object.prototype` inherit properties and methods from a prototype

To add a new property to a constructor

```
myobject3.prototype.myfunc
tion = function () {
    console.log("object
constructor");
    document.write("<br>",
"object constructor");
  }
var myobject4 = new
myobject3("shanaya", 20,
"mumbai");
  document.write("<br>",
myobject4);//[object
Object]
  for (const key in
myobject4) {
    const element =
myobject4[key];

window.document.write("<br
>", key, ":",
myobject4[key]);
  }
  myobject4.myfunction();
```

```
parameters at top and non
parameters below if method
name is same
    age(x)
{//car2InstanceTry.age() will
give error if parameter is
not given
      return x - this.year;
    }
    //OR
    age() {
const date = new Date();
      return
date.getFullYear() -
this.year;
    }
  }
 var car1InstanceTry = new
CarClassMethodTry('Toyota',
2015);
car1InstanceTry.age(); //
Output: 8
var yearPass = new
Date().getFullYear();//it is
passed in
car1InstanceTry.age(year)
 document.write("<br><br>car1
InstanceTry.age(yearPass) : "
+
car1InstanceTry.age(yearPass)
);//8
//consider this example
  class Animal {
    constructor(name) {
      this.name = name;
    }
  }
  class Dog extends Animal {
    constructor(name, age) {
      super(name);
      this.age = age;
    }
  }
  class Cat extends Animal {
    constructor(name, age) {
      super(name);
      this.age = age;
    }
  }

  document.write(
    "<br><br>Animal class
```

```
/ig, "Hi"));//Hi
Hello Shanaya  //
/i made it
insensitive //    /i
flag (insensitive):g
flag (global match)
```

replaceAll()

```
document.write("<br>
",
text2.replaceAll("He
llo", "Hi"));//Hi Hi
Shanaya
```

split()

```
A string can be
converted to an
array with the
split() method:
 text.split(",")
// Split on commas
  text.split(" ")
// Split on spaces
  text.split("|")
// Split on pipe

document.write("<br>
", text2.split("
"));//If the
separator is "", the
returned array will
be an array of
single
characters:
//Hello,Hello,Shanay
a
  //["Hello",
"Hello", "Shanaya"]

document.write("<br>
", text2.split(" ")
[2]);//Shanaya
```

search()

```
document.write("<br>
",
text2.search("hello"
));//0 //not found

document.write("<br>
",
text2.search(/hello/
));//searching text
with regular
```

```
//myFunctionFindfunction
myFunctionFind(value,
index, array) { return
value > 18; }
 document.writeln("<br>"
+ first); // Output: 25
```

findIndex() method

```
const fruitsFindIndex =
["apple", "banana",
"cherry"];
  document.write("<br>",
fruitsFindIndex.findInde
x(x => x ===
"banana"));//1
```

findLast() method

the findLast() method that will start from the end of an array and return the value of the first element that satisfies a condition.

```
const temp = [27, 28,
30, 40, 42, 35, 30];
  let high =
temp.findLast(x => x >
40);
 document.writeln("<br>f
indLast() high :" +
high);//findLast() high
:30
```

findLastIndex()

```
const tempLas = [27, 28,
30, 40, 42, 35, 30];
  let pos =
tempLas.findLastIndex(x
=> x > 40);
  document.write("<br>
findLastIndex :" +
pos);//4
```

reverse() method

```
const tempReverse = [27,
28, 30, 40, 42,
    35, 30];
  tempReverse.reverse();

document.writeln("<br>re
verse() :" +
tempReverse);//reverse()
:30,35,42,40,30,28,27
```

toReversed()

```
const monthstoReversed =
["Jan", "Feb", "Mar",
"Apr"];
```

const car = {type:"Fiat", model:"500", color:"white"};
name:value pairs are also called key:value pairs seperated by comma
Spaces and line breaks are not important. An object initializer can span multiple lines:object literals are also called object initializers.

The object data type can contain both built-in objects, and user defined objects:
Built-in object types can be: objects, arrays, dates, maps, sets, intarrays, floatarrays, promises, and more.
User defined object types can be: any object created by the user, such as a class or a function.

Date object:

JavaScript Stores Dates as zero time Milliseconds since 01 Jan 1970 00:00 and this date and time is considered as
One day (24 hours) is 86 400 000 milliseconds
new Date(milliseconds) creates a new date object as milliseconds plus zero time:

```
var date = new
Date(-1000000000000000)
  //1 January 01 1970
minus 1000000000000000
millisecond
//January 01 1970 plus 24
hours is:
  var date = new
Date(86400000);
  //or
  var date = new Date(24 *
60 * 60 * 1000);
  document.write(
    "<br>new date 24 * 60
* 60 * 1000 set :",
date);//new date 24 * 60 *
60 * 1000 set :Fri Jan 02
```

```
method: <br>",
Animal.prototype.constructor.name + "<br>",
    "Dog class method: <br>",
Dog.prototype.constructor.name + "<br>",
    "Cat class method: <br>",
Cat.prototype.constructor.name + "<br>"

  );
```

JavaScript JSON

JSON stands for JavaScript Object Notation.Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.JSON makes it possible to store JavaScript objects as text in string.JSON cannot be an object. JSON is a string format.
The data is only JSON when it is in a string format. When it is converted to a JavaScript variable, it becomes a JavaScript object.

JSON is a lightweight data interchange format and used for exchanging data between a server and web applications
SON is used for storing data in a file or database
The file type for JSON files is ".json"
The MIME type for JSON text is "application/json"

JSON is easy to read and write,self-describing ,language independent,text-based data format,subset of JavaScript

```
 var text = '{ "employees" :
[' + '{ "firstName":"Sudha" ,
"lastName": "Kumari" } ,
{ "firstName":"Sana" ,"lastName" : "W"} ,
{ "firstName":"Shanaya" ,
```

```
expression.//-1
//not found
document.write("<br>",
text2.search("l"));//2
```

NOTE: The search() method cannot take a second start position argument.
The indexOf() method cannot take powerful search values (regular expressions).

match()

The match() method searches a string for a match against a regular expression, returning an array matching the entire string, or null if there is no match:

```
document.write("<br>",
text2.match("Hello"));//["Hello"]

document.write("<br>",
text2.match("llo"));

document.write("<br>",
text2.match("Hello Hello
Shanaya"));//Hello
Hello Shanaya

document.write("<br>",
text2.match("Hello
Hello Shanaya")
[8]);//undefined

document.write("<br>",
text2.match("l"));//["l"]

document.write("<br>",
```

```
  const
reversedToReversed =
monthstoReversed.toReversed();

document.writeln("<br>toReversed() :" +
reversedToReversed);//toReversed() :Apr,Feb,Jan,Mar
```

sorting array

```
templas.sort();//it will be updated to sorted array,old array destroyed
  document.write("<br>templas.sort() :" +
templas);//templas.sort() :27,28,30,30,35,40,42
```

Decending sort()

```
document.writeln("<br>tempReverse.reverse after sort:" +
templas.reverse());//tempReverse.reverse after sort:42,40,35,30,30,28,27
```

toSorted() method

The difference between toSorted() and sort() is that the first method creates a new array, keeping the original array unchanged, while the last method alters the original array.

```
const monthstoSort =
["Jan", "Feb", "Mar",
"Apr"];
  const sortedToSorted =
monthstoSort.toSorted();
document.writeln("<br>toSorted() :" +
sortedToSorted);//toSorted() :Apr,Feb,Jan,Mar
document.writeln("<br>monthstoSort :" +
monthstoSort);//monthstoSort
:Jan,Feb,Mar,Apr//remained as it is
```

```
1970 05:30:00 GMT+0530
(India Standard Time)
//Previous Century
  //One and two digit
years will be interpreted
as 19xx
  var date = new Date(99,
11, 24);
  document.write("<br>new
date 99, 11, 24 set : ",
date)//new date 99, 11, 24
set : Fri Dec 24 1999
00:00:00 GMT+0530 (India
Standard Time)//1999 from
19xx and xx as 99

new
Date(year,month,day,hours,
minutes,seconds,ms)
```

By default, JavaScript will use the browser's time zone and display a date as a full text string.Date objects are static. The "clock" is not "running".use setInterval to make it ticking
If you supply only one parameter without string 2015 then it will be treated as milliseconds but with string it will be year "2015" year

const date = new Date("2022-03-25");

NOTE:string as a whole can be used only upto 3 data of dates and seperated by any either , or - but after 3 elements if required for time then no string as a whole should be used but individual elements can be string to avoid 02 as octata data hence use '02'and all elements seperated by ,

```
var date = new
Date('2023,02,24');
var date = new
Date('2023,02');//new
Date(year,month)
var date = new Date(2023);
//millisecond
```

```
"lastName" : "Singh" }]}';
we received this text from a web server in JSON string format and it need to parse
```

Data is in name/value pairs , just like JavaScript object properties and data separated by commas,Curly braces hold objects. Square brackets hold arrays
the object "employees" is an array. It contains three objects.Each object is a record of a person (with a first name and a last name).

```
In JSON, values must be one of the following data types:
   a string - {"name":"John"}
  a number -{"age":30} or {"age":30.43}
  an object (JSON object) - {"employee":{"name":"John", "age":30, "city":"New York"}}//Objects as values in JSON must follow the JSON syntax.
  an array -{"employees":["John", "Anna", "Peter"]}
  a boolean - {"sale":true}
  null - {"middlename":null}
  In JavaScript values can be all of the above, plus any other valid JavaScript expression but it need extra efforts, including:
  a function
  a date
  undefined
  In JSON, string values must be written with double quotes:{"name":"John"}
  In JavaScript, you can write string values with double or single quotes:
```

use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object
   const obj = JSON.parse(text);

```
document.getElementById("demo").innerHTML
```

```
text2.match("o"));//["o"]

document.write("<br>",
text2.match("hello"));//null //not found
document.write("<br>",
text2.match(/hello/ig));//Perform a global, case-insensitive search for hello
```

matchAll()

The matchAll() method returns an iterator(only no. of times it get matched return it and they are seperated by comma like ; Hello,Hello,Hello) of matches for a string against a regular expression, or an empty iterator if no match is found. It returns all matches, not just the first one

```
document.write("<br>",
text2.matchAll(/hello/ig));//Perform a global, case-insensitive search for hello
```

includes

```
Syntax:string.includes(searchvalue, start)
document.write("<br>",
text2.includes("Hello"));//true

document.write("<br>",
text2.includes("Hello", 10));//false
//Check if a string includes "Hello". Start checking from
```

Numeric sort()

The sort() method takes a comparison function as an argument.The comparison function defines the sort order.
The sorting is done by passing a comparison function to the sort method. This function takes two arguments, a and b, and returns the difference between them (a - b). If the result is negative, a will be placed before b in the sorted array; if positive, b will be placed before a. If the result is zero, their order remains unchanged.

```
const numbersSort = [25, 100, 1, 2];
document.writeln("<br>numbersSort.sort() :" + numbersSort.sort((a, b) => {
    document.write("<br>a-b inside function :" + (a - b));//at first b=25 and a=100
    return a - b;//value to be passed in sort//gives asscending order
  }));//The sorting is done by passing a comparison function to the sort method. This function takes two arguments, a and b, and returns the difference between them (a - b). If the result is negative, a will be placed before b in the sorted array; if positive, b will be placed before a. If the result is zero, their order remains unchanged.//combination of 2 formed from each.
  document.writeln("<br>
```

```
var date = new Date('2023,02,24,21');
  document.write("<br>new date '2023,02,24,21' set : ", date);//new date 2023,02,24,21 set : Invalid Date because only date is in string alone but if hour is to be combined then no string on any data allowed
var date = new Date(2023,02, 24, 4);//octal data type warning
var date = new Date(2023, '02', 24, 4);
  document.write("<br>new date 2023,02,24,4 set : ", date);//new date 2023,02,24,4 set : Fri Mar 24 2023 04:00:00 GMT+0530 (India Standard Time)//in 2023,02,24,4   02 is taken as octal hence put it in string but even if not put ,it will give no error on underline ,Octal literals are not allowed. Use the syntax '0o2'.
var date = new Date(2023, '02', 24, 4, 33, 30, 0);

document.write("<br>typeof new Date() :", typeof new Date());//object//new keyword creates date object
 document.write("<br>typeof new Date() :", typeof Date());//string
 const time = new Date();
  console.log(time instanceof Date); // true
  console.log(time instanceof Object); // true
 getDate()   Get the day as a number (1-31)
getUTCDate()
document.write("<br>
=obj.employees[1].firstName + " " + obj.employees[1].lastName;//obj.employees[1].firstName + obj.employees[1].lastName :Sana W

  //here obj is object of string input and JSON array becomes the property of obj and this array has further objects and their properties
```

You can create a JavaScript object from a JSON object literal:

```
 myJSON = '{"name":"John", "age":30, "car":null}';
 myObj = JSON.parse(myJSON);
//myObj = {"name":"John", "age":30, "car":null};
You can access object values by using dot (.) notation:
  xOBJ = myObj.name;//after parse
```

OR

```
by using bracket ([]) notation:
xOBJ = myObj["name"];
```

Looping an Object

You can loop through object properties with a for-in loop

```
 let text = "";
  for (const x in myObj) {
   text += x + ", "; //gives only key //name, age, car,
```

OR

```
text += myObj[x] + ", ";
  }
```

Array as JSON/`JSON Array Literals`

```
var text = '["Ford", "BMW", "Audi", "Fiat"]';
  var myArrJSON = JSON.parse(text);
myArrJSON[0]//myArrJSON[0] returns "Ford"
```

Objects can contain arrays:

```
var myObj={
  "name":"John",
```

```
position 10 //it does not means exactly h should be present at 10.
```

startsWith()

```
document.write("<br> text2.startsWith :", text2.startsWith("Hello"));//true
 document.write("<br> ", text2.startsWith("hello"));//false
document.write("<br> ", text2.startsWith("Hello", 10));//false //Check if a string starts with "Hello" at position 10
```

endsWith

```
document.write("<br> ", text2.endsWith("Shanaya"));//true
document.write("<br> ", text2.endsWith("a"));//true
```

localeCompare

```
let text2 = "Hello Hello Shanaya"
document.write("<br> ", text2.localeCompare("Hello"));//1//text 2 comes after Hello
document.write("<br> ", text2.localeCompare("ll"));//-1//h comes before ll
```

normalize

Returns a string containing the Unicode Normalization Form of the calling string's value. `String.prototype.normalize()` is correct in a technical sense, because normalize() is a dynamic method you call on instances, not the class

```
after sorting asscending numbersSort  : " + numbersSort);
document.writeln("<br>numbersSort.sort() :" + numbersSort.sort((a, b) => {
    document.write("<br> b-a inside function :" + (b - a));//at first b=25 and a=100
    return b - a;//value to be passed in sort//gives descending order
  }));
  document.writeln("<br> after sorting descending numbersSort  : " + numbersSort);
```

Sorting an Array in Random Order

The sort() method can sort the array in random order by using the Math.random() function in the comparison function.

```
let numbersRandomSort = [25, 100, 1, 2];
document.writeln("<br> <button onclick='myRandomSort()'> RandomSort </button></br>");
function myRandomSort() {
    //document.writeln("<br>numbersRandomSort.sort() :"+numbersRandomSort.sort((a, b )=> Math.random() - 0.5 * (a - b)));//always gives descending order ////100,25,2,1
    2007
    let numbersRandomSortafter = Math.random() - 0.5;
//numbersRandomSort.sort((a, b)=>{ return
```

```
date.getUTCDate() :", date.getUTCDate());//6//date.getUTCDate() will also give same result just by date.getDate()

  getDay()    Get the weekday a number (0-6). (day 0) is Sunday but Some countries in the world consider the first day of the week to be Monday.


  getFullYear()   Get the four digit year (yyyy)
  getHours()  Get the hour (0-23)
  getMilliseconds()   Get the milliseconds (0-999)
document.write("<br> date.getMilliseconds() :", date.getMilliseconds());//372

  getMinutes()    Get the minutes (0-59)
  getMonth()  Get the month (0-11)
You can use an array of names of month to return the month as a name
  const monthsNames = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"];
  var date = new Date();

document.write("<br>monthsNames[date.getMonth()] :", monthsNames[date.getMonth()]);//Get month as a name//it is just like getting index value from date.getMonth() and monthsNames[date.getMonth()] returns the element at that index
```

```
  "age":30,
  "cars":["Ford", "BMW",
"Fiat"]
  }
myObj.cars[0];
```

Looping Through an Array

```
 for (let i in myObj.cars) {
   x += myObj.cars[i];
  }
Or you can use a for loop:
for (let i = 0; i <
myObj.cars.length; i++) {
   x += myObj.cars[i];
  }
```

Parsing Dates

Date objects are not allowed in JSON.

If you need to include a date, write it as a string.

You can convert it back into a date object later:

```
var text = '{"name":"Sudha",
"birth":"1986-12-14",
"city":"Lucknow"}';
  var obj = JSON.parse(text);
  obj.birth = new
Date(obj.birth);// convert it
back into a date object
document.write("<br>myArrJSON
[0] : " + obj.name + ", " +
obj.birth);
```

Or, you can use the second parameter, of the JSON.parse() function, called reviver. The reviver parameter is a function that checks each property, before returning the value.

```
var obj = JSON.parse(text,
function (key, value) {//here
this function is reviver
function and it is passed as
the second parameter, of the
JSON.parse() function
    if (key == "birth") {
      return new
Date(value);//Convert a
string into a date, using the
reviver function:
    } else {
      return value;
    }
```

itself. The point of normalize() is to be able to compare Strings that look the same but don't consist of the same characters

```
document.write("<br>
",
text2.normalize());/
/Hello Hello Shanaya

const name1 =
'\u0041\u006d\u00e9\
u006c\u0069\u0065';
  const name2 =
'\u0041\u006d\u0065\
u0301\u006c\u0069\u0
065';
  //not jQuery as $
is used not for dom
elements tracing
//it is
interpolation if
takin value of a
variable
  console.log(`$
{name1}, ${name2}`);
  // expected
output: "Amélie,
Amélie"
  console.log(name1
=== name2);
  // expected
output: false

console.log(name1.le
ngth ===
name2.length);
  // expected
output: false
  const name1NFC =
name1.normalize('NFC
');
  const name2NFC =
name2.normalize('NFC
');
  console.log(`$
{name1NFC}, $
{name2NFC}`);
  // expected
output: "Amélie,
Amélie"
```

```
numbersRandomSortafter;}
);//100,25,2,1
numbersRandomSort.sort((
) => { return
numbersRandomSortafter }
);//numbersRandomSort.so
rt() method here is not
accurate. It will favor
some numbers over
others.//The most
popular correct method,
is called the Fisher
Yates shuffle
document.getElementById(
"RandomS").innerHTML =
"<br>numbersRandomSort
after :" +
numbersRandomSort
  };
document.writeln("number
sRandomSort after :" +
numbersRandomSort);//25,
100,1,2 at start//and no
change there after and
no updates
```

Fisher-Yates shuffle algorithm

```
document.writeln("<br>
<button
onclick='myRandomSortN()
'> RandomSortNew
</button></br>");
  document.writeln("<br>
<p id='RandomSN'>
RandomSNew </p></br>");

  function
myRandomSortN() {
    // Fisher-Yates
shuffle algorithm
    // which randomly
shuffles the elements in
an array by selecting
element from the portion
of the array that has
not yet been shuffled.
    //Looping through
the array: The for loop
starts with i set to the
last index of the points
array (points.length -
```

```
  getSeconds()    Get the
seconds (0-59)
  getTime()   Get the time
(milliseconds since
January 1, 1970)
NOTE:getUTCFullYear() same
as
getFullYear(),getUTCMonth(
) same as
getMonth(),getUTCDay()
same as
getDay(),getUTCHours()
same as
getHours(),getUTCMinutes()
same as
getMinutes(),getUTCSeconds
() same as
getSeconds(),getUTCMillise
conds() same as
getMilliseconds()
```

Time Zones

When setting or getting a date, without specifying the time zone, JavaScript will use the browser's time zone.

In other words: If a date/time is created in GMT (Greenwich Mean Time), the date/time will be converted to CDT (Central US Daylight Time) if a user browses from central US.

```
date.getTimezoneOffset();/
/-330//The
getTimezoneOffset() method
returns the difference (in
minutes) between local
time an UTC time  //local
time is according to
browser window
```

JavaScript Date Input

```
  });
```

Parsing Functions

If you need to include a function, write it as a string.

You can convert it back into a function later:

```
var text = '{"name":"John", "func":"function(){console.log(\"Hello World!\");}"    }';//error due to regular expression
  var text = '{"name":"John", "func":"function(){return 20;}"    }';
  var obj = JSON.parse(text);
obj.func = eval("(" + obj.func + ")"); // convert it back into a function object //obj.func(); // call the function object
document.write("<br>obj.name, obj.func :" + obj.name + ", " + obj.func);//not creates error as comma is not taken in different way //don't forget ()
```

NOTE: When sending data to a web server, the data has to be a string.

JSON.stringify() converts a JavaScript object into a JSON string.

```
var myJSON = JSON.stringify(obj);
  //The result will be a string following the JSON notation.
  //myJSON is now a string, and ready to be sent to a server:
  document.write("<br> myJSON :" + myJSON);
```

Stringify a JavaScript Array

```
var arrForStringify = ["John", "Peter", "Sally", "Jane"];
var myJSON = JSON.stringify(arrFor
```

```
console.log(name1NFC === name2NFC);
  // expected output: true

console.log(name1NFC.length === name2NFC.length); // expected output: true
```

NOTE:
the string Amélie has two different Unicode representations. With normalization, we can reduce the two forms to the same string.

The main problem is that you may have two strings which are semantically the same, but with two different representations: e.g. one with a accented character [one code point], and one with a character combined with accent [one code point for character, one for combining accent]. User may not be in control on how the input text will be sent, so you may have two different user names, or two different password. But also if you mangle data, you may get different results, depending on initial string. Users do not like it.

An other problem is about unique order of combining characters. You may have an accent, and a lower tail (e.g. cedilla): you may express this with several combinations: "pure

```
1) and continues until i is greater than 0. It decrements i by 1 in each iteration.
    //Generating a random index: Inside the loop, j is assigned a random integer between 0 and i (inclusive) using Math.floor(Math.random() * (i + 1)). This random index represents a position in the array that will be swapped with the current position i.
    //Swapping elements: The element at index i (stored in k) is swapped with the element at index j. This is done by temporarily holding the value at index i in k, then assigning the value from index j to index i, and finally assigning the value from k to index j.

    for (let i = numbersRandomSort.length - 1; i > 0; i--) {
      let j = Math.floor(Math.random() * (i + 1));
      let k = numbersRandomSort[i];//as it's index value will be updated in below hence its value is stored in k so that it can't be lost

numbersRandomSort[i] = numbersRandomSort[j];//value at index i updated to value at index j

numbersRandomSort[j] = k;//value at index j updated to previous
```

```
  JavaScript Date Input can be in the following formats:
  1. "MM/DD/YYYY"  which is Short Date//slash
var date = new Date("2024/01/01");
  document.write("<br>date : ", date);//date : Mon Jan 01 2024 00:00:00 GMT+0530 (India Standard Time) //it is acc to browser// but in some browser it returns NaN or undefined
  var date = new Date("01/01/2024");//same as above result but in some NaN or undefined////date : Mon Jan 01 2024 00:00:00 GMT+0530 (India Standard Time)

  2.(YYYY-MM-DDTHH:MM:SSZ): "2015-03-25"  which is ISO Date (The International Standard)//The ISO format follows a strict standard in JavaScript.// hyphen or dash//The other formats are not so well defined and might be browser specific.
  3. "25 Mar 2015" or "Mar 25 2015"  which is Long Date /Long dates are most often written with a "MMM DD YYYY" syntax like this and mont and date can be in any order And, month can be written in full (January), or abbreviated (Jan)

  no leading zeroes may produce an error in some browsers
  Note:UTC (Universal Time Coordinated) is the same
```

```
Stringify);
```

When ***storing data***, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

Storing data in local storage

```
var myObjStoring = { name: "Sudha", age: 31, city: "Lucknow" };
  var myJSON = JSON.stringify(myObjStoring);
localStorage.setItem("testJSON", myJSON);
document.write(myJSON + "<br>"+myJSON.name);//{"name":"Sudha","age":31,"city":"Luc know"}
 //myJSON.name :undefined
//it is undefined as it is string
// Retrieving data:
 var text = localStorage.getItem("testJSON");
 var obj = JSON.parse(text);
  document.write(obj + "<br>obj.name :" + obj.name);
//[object Object]
//obj.name :Sudha
```

Stringify Dates

```
var objDateJSON = { name: "John", today: new Date(), city: "New York" };
  var myJSON = JSON.stringify(objDateJSON);
document.write(myJSON);
{"name":"John","today":"2024-08-24T17:55:22.301Z", "city":"New York"}
```

Stringify Functions

The JSON.stringify() function will remove any functions from a JavaScript object, both the key and char, tail, accent", "pure char, accent, tail", "char+tail, accent", "char+accent, cedilla".

And you may have degenerate cases (especially if you type from a keyboard): you may get code points which should be removed (you may have a infinite long string which could be equivalent of few bytes.

In any case, for sorting strings, you (or your library) requires a normalized form: if you already provide the right, the lib will not need to transform it again.

So: you want that the same (semantically speaking) string has the same sequence of unicode code points.

Valid variable name

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

Let **_a4A_$** ;

```
// Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.
  Let $$$ = 2;
```

jQuery Selectors

```
In jQuery $("p");
```

```
stored value of index i in k variable
document.getElementById("RandomSN").innerHTML = "<br>numbersRandomSort after :" + numbersRandomSort;
    }
  }
```

Find the Lowest (or Highest) Array Value

first use sort() then select

```
document.write("minMaxSort after sorting:<br> " + minMaxSort[0] + " " + minMaxSort[minMaxSort.length - 1] + " " + minMaxSort);
```

Math.min.apply()/ Math.max.apply()

use Math.min.apply to find the lowest number in an array:

The Math.min() function returns the smallest of zero or more numbers.

The Math.min() function takes one or more arguments and returns the smallest one.

The Math.min() function is a static method of Math, therefore called on Math.

The Math.min() function is not generic, it doesn't work with objects other than numbers.

Math.min.apply(null, [1, 2, 3]) is equivalent to Math.min(1, 2, 3).

The apply() method calls a function with a given this value and arguments provided as an array.

The apply() method is a method of the Function prototype, therefore called on a function.

```
let minMaxMaths = [4, 2, 9, 6, 5, 1];
  function myArrayMin(minMaxMath) {
```

```
as GMT (Greenwich Mean Time).
```

NOTE: In ISO result vary from previous day to mention day

```
  var date = new Date("2024-01-01");
  document.write("<br>date : ", date);//date : Mon Jan 01 2024 05:30:00 GMT+0530 (India Standard Time) //always gives correct output//The computed date will be relative to your time zone.//Depending on your time zone, the result above will vary between 31 Dec and 1 Jan
```

Commas are ignored. Names are case insensitive:

```
  var date = new Date("25, MARCH, 2024");
  document.write("<br>date : ", date);
```

Date Methods

```
var date = new Date();
  console.log(Date);/*ƒ Date() { [native code] }*/
  console.log(date); /**/ console.log(date.toUTCString());
```

```
date.toUTCString()
```

Tue, 06 Aug 2024 04:48:19 GMT//here time is according to universal time zone not indian time zone but date is universal everywhere only timezone differ

```
date.toISOString()
```

The toISOString() method converts a date to a string using the ISO standard which is international

The ISO format follows a strict standard in JavaScript.

The other formats are not so well defined and might be browser specific

the value:
This can be omitted if you convert your functions into strings before running the JSON.stringify() function.

```
var objJSONfunStringify = { name: "Sudha", age: function () { return 30; }, city: "New York" };
  var myJSON1 = JSON.stringify(objJSONfunStringify);
document.write("<br> obj1 JSON.parse(myJSON1) : " + obj1 + "<br>obj1.name :" + obj1.name + "<br>obj1.age :" + obj1.age); //obj1 JSON.parse(myJSON1) : [object Object]
  // obj1.name :Sudha
  //obj1.age :undefined //as it is no more a function
  //obj1.age() show error as it is not object method
var objJSONfunStringify = { name: "John", age: function () { return 30; }, city: "New York" };
  objJSONfunStringify.age = objJSONfunStringify.age.toString();// Convert functions into strings to keep them in the JSON object.
 var myJSON = JSON.stringify(objJSONfunStringify);
  document.write("<br> myJSON JSON.stringify(objJSONfunStringify) :" + myJSON + "<br>myJSON.age :" + myJSON.age);
  //
```

```
means "select all p elements".
```

jQuery selectors are used to select DOM elements. They are similar to CSS selectors, but they are used to select elements in the DOM, not to style them.

```
 myElement = $("#id01");//# shows id
myElement = document.getElementById("id01");
```

Hence jQuery not need as we can use DOM

```
 myElements = $(".intro");
myElements = $("p.intro");
myElements = document.querySelectorAll("p.intro");
 myElement.text("Hello Sweden!");
myElement.textContent = "Hello Sweden!";
myText = $("#02").text();
 myElement.html("<p>Hello World</p>");
myElement.innerHTML = "<p>Hello World</p>";
content = myElement.html();
content = myElement.innerHTML;
 myElement.hide();
 myElement.style.display = "none";
 myElement.show();

myElement.style.display = "";
 $("#demo").css("font-size", "35px");//way of giving object in jQuery
document.getElementById("demo").style.fontSize = "35px";
  $("#id02").remove();
```

```
    let minMaths = Math.min.apply(null, minMaxMath);
let maxMaths = Math.max.apply(null, minMaxMath);
document.write(" Math.min.apply:" +
        minMaths + " is the lowest value in the array " + minMaxMaths + "<br> " +
        maxMaths + " is the lowest value in the array " + minMaxMaths);
 }
myArrayMin(minMaxMaths);
```

to find the lowest number is to use a home made method.

```
function myArrayMinHome(arr) {
    let len = arr.length;
    let min = Infinity;
    while (len--) {
      if (arr[len] < min) {
        min = arr[len];
      }
    }
    //return min;
document.write("
        min + " is the lowest value in the array " + arr + "<br> "
        );
  }
myArrayMinHome(minMaxMaths);
//OR
function myArrayMinHome(arr) {
    let lowest = arr[0];
    for (let i = 1; i < arr.length; i++) {
      if (arr[i] < lowest) {
        lowest = arr[i];
      }
```

```
date.toISOString()
2024-08-06T04:51:44.411Z//the ISO standard for date is 2024-08-06 and T for time and in universal time zone always//Date and time is separated with a capital T.//UTC time is defined with a capital letter Z.
var date = new Date("2024-01-01T12:00:00Z");//Omitting T or Z in a date-time string can give different results in different browsers.
```

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead://Modify the time relative to UTC by adding +HH:MM or subtraction -HH:MM to the time.

```
 var date = new Date("2024-01-01T12:00:00-06:30");//removing z becomes "2024-01-01T12:00:00" and now adding +HH:MM or -HH:MM and no gap in between
  document.write("<br>date 2024-01-01T12:00:00-06:30 : ", date);//date 2024-01-01T12:00:00 -06:30 : Tue Jan 02 2024 00:00:00 GMT+0530 (India Standard Time)
```

```
date.toString()
```

JavaScript will (by default) output dates using the toString() method

```
ue Aug 06 2024 10:19:07 GMT+0530 (India Standard Time)
```

```
date.toDateString()
date.toTimeString()
date.toLocaleString("en-US", {
```

```
{"name":"John","age":"function () {return 30;}","city":"New York"}
  // myJSON.age :undefined
```

Retrieving data:

```
var objT = JSON.parse(myJSON);
document.write("<br> objT JSON.parse(myJSON) :" + objT + "<br>objT.name :" + objT.name + "<br>objT.age :" + objT.age); //objT.age() gives error
  //objT JSON.parse(myJSON) : [object Object]
  // objT.name :John
  // objT.age :function () {return 30;}
  obj.age = eval("(" + objT.age + ")");
  document.write("<br>After eval obj.age :" + obj.age);
//now objT.age() can be used //After eval obj.age :function () {return 30;}
  document.write("<br>After eval obj.age() :" + obj.age());//After eval obj.age :30
```

JSON vs XML

XML is a markup language that is used to store and transport data between systems. It is more verbose than JSON and is often used for configuration files and data exchange between systems.

JSON is generally faster and more efficient than XML. JSON is also more widely supported than XML

JSON is Unlike XML Because

JSON doesn't use end tag

JSON is shorter

JSON is quicker to read and write

JSON can use arrays

The biggest difference is: XML has to be parsed with an XML parser. JSON can be parsed by a

```
document.getElementById("id02").remove();
myParent = $("#02").parent().prop("nodeName");;
myParent = document.getElementById("02").parentNode.nodeName;
```

Swapping JavaScript Variables

```
var firstName = "Sudha";
  var lastName = "Kumari";
// Destructing
  [firstName, lastName] = [lastName, firstName];

document.writeln("<br>firstName :" + firstName)// Outputs: Kumari

document.writeln("<br>lastName :" + lastName)// Outputs: Sudha
```

Escape Characters

```
let textManyDoubleString = "We are the so-called "Vikings" from the north.";
  //The string will be chopped to "We are the so-called ".
```

The backslash escape character (\) turns special characters into string characters://means whatever will be written after \ will become a string

```
 \" inserts a double quote in a string
\' inserts a single
```

```
if (arr[i] > highest) {
        highest = arr[i];
      }
    }
    // return lowest;
document.write(lowest + " is the lowest value in the array " + arr + "<br>");
  }
myArrayMinHome(minMaxMaths);
```

Sorting Object Arrays

```
const carsArrObj = [
    { type: "Volvo", year: 2016 },
    { type: "Saab", year: 2001 },
    { type: "BMW", year: 2010 },
    //{type:"bMW", year:2010}//here b will be consider largeer tha V , hence it will give error result without toLowerCase.
    //When toLowerCase is used then it gives proper result ignoring case sensitive
  ];
//Sorting the array of objects by year in ascending order
  carsArrObj.sort((a, b) => a.year - b.year);
 document.write("<br><br>Sorting the array of objects by year in ascending order: "
    + JSON.stringify(carsArrObj) + "<br>typeof JSON.stringify(carsArrObj) : " + typeof JSON.stringify(carsArrObj)//string
  );
  carsArrObj.sort((a, b) => a.year - b.year);
```

```
timeZone: "Asia/Kolkata" })
date.toLocaleDateString()
date.toLocaleTimeString()
new Date().toLocaleTimeString();//12:00:00 AM
var livetimer = new Date().toLocaleTimeString();
```

//live time only at the time of open then don't update , it gets update only after reopen

```
document.getElementById("livetimer").innerText = livetimer;

//using same in function //and it can be put in set interval for live
  function timer() {
    var livetimer = new Date().toLocaleTimeString();

document.getElementById("livetimer").innerText = livetimer;
  }
  const myInterval = setInterval(timer, 1000);//myInterval introduced to use clearInterval

//setInterval(timer,1000);//1 sec//update itself without reload after 1 sec as setinterval is used
  var StopFunction = function () {

window.clearInterval(myInterval);
  }
  StopFunction();
```

standard JavaScript function.

XML is much more difficult to parse than JSON.

JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

In XML,Fetch an XML document

Use the XML DOM to loop through the document

Extract values and store in variables

In JSON,Fetch a JSON string

JSON.Parse the JSON string

```
//XML Example
```

employees.xml

```
<!-- This XML file does not appear to have any style information associated with it. The document tree is shown below.These tags are case sensitive. Everyting else than tree must be commented -->
  <employees>
   <employee>
<firstName>John</firstName>
<lastName>Doe</lastName>
   </employee>
   <employee>
<firstName>Anna</firstName>
<lastName>Smith</lastName>
   </employee>
   <employee>
<firstName>Peter</firstName>
<lastName>Jones</lastName>
   </employee>
  </employees>
```

Similarity b/w JSON and XML

JSON is Like XML Because

Both JSON and XML are "self describing" (human readable)

Both JSON and XML are hierarchical (values within values)

Both JSON and XML can be parsed and used by lots of programming languages

Both JSON and XML can be fetched with an XMLHttpRequest

```
quote in a string:
\\ inserts a backslash in a string:
let textManyDoubleString = "We are the so-called\"Vikings\" from the north.It\'s alright.Location C:\\Windows\\System32\\drivers\\etc\\hosts";

document.write("<br>", textManyDoubleString);
  //The string will be written as "We are the so-called "Vikings" from the north."
```

Six other escape sequences are valid in JavaScript:

```
\n inserts a newline in a string:
let textNewLine = "Hello\r\nWorld";//gives same as above

document.write("<br> textNewLine :", textNewLine);
//Hello
  //World //new line for world

document.getElementById("escape characters").innerHTML = textNewLine;//it do not make sense to use escape in html
```

\t inserts a tab in a string: `//\t Horizontal Tabulator`

\b inserts a backspace

```
  );
//OR
document.write("<br><br>Sorting the array of objects by year in ascending order: <br> "
    + carsArrObj[0].type + " " + carsArrObj[0].year + "<br>" +
    carsArrObj[1].type + " " + carsArrObj[1].year + "<br>" +
    carsArrObj[2].type + " " + carsArrObj[2].year + "<br>"
  );//Saab 2001
  //BMW 2010
  //Volvo 2016
```

Sorting the array of objects by type

use the localeCompare() method to compare strings. The localeCompare() method returns a number indicating whether the string comes before, after or is the same as the given string in sort order.

```
carsArrObj.sort((a, b) => a.type.toLowerCase().localeCompare(b.type.toLowerCase()));
document.write("<br>Sorting the array of objects by type in asscending order using toLowerCase with localeCompare  : "
    + JSON.stringify(carsArrObj) + "<br>");
```

OR

```
carsArrObj.sort(function (a, b) {
    let x = a.type.toLowerCase();
    let y = b.type.toLowerCase();
    if (x < y) { return
```

Better representation

```
let timeid = document.getElementById("time"); //timidId should be global for outer function to work inside like mytimidfunction(), as it is called   inside setinverval and event onclick
  setInterval(function () {
    var date = new Date();
// let timeid=document.getElementById("time") ; //timidId should be global for outer function to work inside like mytimidfunction()
    //element called stored in seperate variable and then used again and again
timeid.innerText = (date.getHours()).toString().padStart(2, "0") + ":" + (date.getMinutes()).toString().padStart(2, "0") + ":" + (date.getSeconds()).toString().padStart(2, "0");
timeid.onclick = mytimidfunction //it works just like if wrapped in ()=>{} or function(){}  as you can see in above the last one
    //adviced to use this as it takes less no.of codes
}, 1000);
function mytimidfunction()
{

timeid.style.backgroundColor = "red"
    timeid.style.color =
```

NOTE:To see the PHP result in HTML fies, use PHP server:serve project instead of live server

AJAX

AJAX = Asynchronous JavaScript And XML.

AJAX is a developer's dream, because you can:

Read data from a web server - after the page has loaded

Update a web page without reloading the page

Send data to a web server - in the background

AJAX is a technique to send and receive data asynchronously from a web server.

AJAX is used to update parts of a web page without reloading the whole page.

AJAX is a combination of several technologies like JavaScript, HTML, and CSS.

Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means that both the web page and the XML file it tries to load, must be located on the same server.

It is done above using XMLHttpRequest in JSON section

NOTE: The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

async: true (asynchronous) or false (synchronous)

Note: By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:execute other scripts while waiting for server response .deal with the response after the response is ready

in a string://it means one character at back is deleted as per the escape

\f inserts a form feed in a string:

```
Note:Form feed means advance downward to the next "page". It was commonly used as page separators, but now is also used as section separators. Text editors can use this character when you "insert a page break". This is commonly escaped as "\f", abbreviated FF, and has ASCII value 12 or 0xC.
let textFormFeed = "Hello\fShanaya\fWorld";

document.write("<br> textFormFeed :", textFormFeed);
  //Hello
  //      Shanaya
  //
World //form feed is inserted in between
```

NOTE: \f is form feed, its use has become obsolete but it is used for giving indentation like //upper jaha khatam hua h next line me usi indentation kofollow karega.

\r inserts a carriage return in a string:

//Note:Carriage return means to return to the beginning of the current line without advancing downward. The name comes from a printer's carriage, as monitors were rare when the

```
-1; }
    if (x > y) { return 1; }
    return 0;
  });
  //this code sorts the carsArrObj array in ascending order based on the type property, ignoring case differences.
```

We can use the ***reverse()*** method to reverse the order of the array

```
  carsArrObj.sort((a, b) => a.type.localeCompare(b.type)).reverse();
NO lowercase
carsArrObj.sort((a, b) => a.type.localeCompare(b.type));
```

Stable Array sort()

Stable sort is a sorting algorithm that maintains the relative order of equal elements.

Stable sort is not as efficient as unstable sort for large arrays, but it is more predictable and easier to understand.

Stable sort is implemented using the merge sort algorithm, which has a time complexity of O(n log n) in the worst case.

```
const ArrObjStable = [
    { name: "X00", price: 100 },
    { name: "X01", price: 100 },
    { name: "X04", price: 110 },
    { name: "X06", price: 110 },
    { name: "X07", price: 110 },
    { name: "X08",
```

```
"yellow"
  }
```

```
date.valueOf()
date.getTime()
date.toISOString()
date.toJSON()
```

Date.parse()

If you have a valid date string, you can use the Date.parse() method to convert it to milliseconds.If the date string is not valid, Date.parse() returns NaN

```
let msecParse = Date.parse("March 21, 2024");
document.write(msecParse);
// 1710959400000
```

Date.now()

The Date.now() method returns the number of milliseconds elapsed since January 1, 1970

Date.now() is equivalent to new Date().getTime()

Date.now() is a static method of the Date object.

You cannot use it on a date object like myDate.now()

```
document.write("<br>Date.now() : ", Date.now());
```

Calculate the number of years since January 1, 1970:

```
  var now = Date.now();
  // Calculate milliseconds in a year
  const minute = 1000 * 60;
  const hour = minute * 60;
  const day = hour * 24;
  const year = day * 365;
  // Divide Date.now() with a year which is in millisecond
```

Hey : The default value for the async parameter is async = true.You can safely remove the third parameter from your code.

Caution: Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

xhttp.send();//you may get a **cached result**. To avoid this, add a unique ID to the URL:

```
   xhttp.open("GET",
"demo_get.asp?ID=12345");
```

If you want to send information with the GET method, add the information to the URL:

```
  xhttp.open("GET",
"demo_get2.asp?
fname=Sudha&lname=Kumari");
xhttp.send();
or
  xhttp.open("GET",
"demo_get.asp?t=" +
Math.random());
  xhttp.send();
```

Multiple Callback Functions

```
loadDoc("url-1",
myFunction1);
loadDoc("url-2",
myFunction2);
function loadDoc(url,
cFunction) {
 const xhttp = new
XMLHttpRequest();
 xhttp.onload = function()
{cFunction(this);}
 xhttp.open("GET", url);
 xhttp.send();
}
function myFunction1(xhttp) {
 // action goes here
}
function myFunction2(xhttp) {
 // action goes here
}
```

Handling XML file

name was coined. This is commonly escaped as "\r", abbreviated CR, and has ASCII value 13 or 0xD.

\uXXXX inserts a Unicode character in a string:

```
  let textUnicode =
"Hello\u{1F600}
World";
document.write("<br>
textUnicode : ",
textUnicode);
//Hello😊World
7.\v    Vertical
Tabulator
  8.\0    Null
character
  9.\xXX
Hexadecimal value
  10.\cX  Control
character
  11.\xXX Unicode
code point
  12.\uXXXX
Unicode code point
  13.\u{XXXXX....}
Unicode code point
```

Template Strings

Template Strings use back-ticks (``)(Back-tick is not the single quote it is ` above the tab not the ") rather than the quotes ("") to define a string:

Templates are strings enclosed in backticks (`This is a template string`).

Templates allow single and double quotes inside a string:

```
  var
textBackTickandDoubl
eIn = `He's often
called "Johnny"`;
  var
textBackTickandSingl
```

```
price: 120 },
    { name: "X09",
price: 120 },
    { name: "X19",
price: 140 }
  ];

ArrObjStable.sort(functi
on (a, b) {
    let x = a.price
    let y = b.price
    if (x < y) { return
-1; }
    if (x > y) { return
1; }
    return 0;
  });
 document.write( JSON.st
ringify(ArrObjStable));
```

to display in better way

```
let txt = "";
ArrObjStable.forEach(myA
rrObjStable);
  //value will be
complete an object at a
time passed in function
myArrObjStable
  function
myArrObjStable(value) {
    //
document.write("<br>valu
e :
"+value+"<br>");//value
: [object Object],need
to use JSON
document.write("<br>valu
e : " +
JSON.stringify(value));/
/value :
{"name":"X00","price":10
0}////value will be
complete an object at a
time passed in function
myArrObjStable
    // txt +=
JSON.stringify(value) +
"<br>";//used for exact
format
    txt += value.name +
" " + value.price +
"<br>"; //X00 100  //for
```

```
  let years =
Math.round(Date.now() /
year);
  document.write("<br>
Math.round(Date.now() /
year) : " + years)//55
 document.write("<br>now -
year : ", now -
year);//deduct 1 year
current year
```

Set Date Methods

```
setFullYear() setMonth()
setDate() setHours()
setMinutes() setSeconds()
setMilliseconds()
  setUTCDate()
setUTCMonth()
setUTCFullYear()
setUTCHours()
setUTCMinutes()
setUTCSeconds()
setUTCMilliseconds()
  setTime() setFullYear()
setMonth() setDay()
setHours() setMinutes()
setSeconds()
setMilliseconds()
  setUTCFullYear()
setUTCMonth() setUTCDay()
setUTCHours()setUTCMinutes
() setUTCSeconds()
setUTCMilliseconds()
document.write("<br>
setFullYear(2026) :" +
date.setFullYear(2026))//1
786004354610
  document.write("<br>date
after setFullYear(2026) :
", date);//date after
setFullYear(2026) : Thu
Aug 06 2026 13:51:01
GMT+0530 (India Standard
Time)
```

The setFullYear() method can optionally set month and day:

```
  document.write("<br>
setFullYear(2026, 8, 6) :"
+ date
    .setFullYear(2026, 8,
20))//1786004354610
  document.write("<br>
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
 const xmlDoc = this.responseXML;
 const x = xmlDoc.getElementsByTagName("employee");//case sensitive hence EMPLOYEE will give no result

 let txt = "";
 for (let i = 0; i < x.length; i++) {
  txt = txt + x[i].childNodes[0].nodeValue + "<br>";
 }
 document.getElementById("demoXML").innerHTML = txt;
}
xhttp.open("GET", "employees.xml");
xhttp.send();
```

JSON Server

```
var myObjServer = { name: "John", age: 31, city: "New York" };
  var myJSON = JSON.stringify(myObjServer);

window.location = "demo_json.php?x=" + myJSON;
//it will escape the main window and open demo_json.php
//http://localhost:3000/demo_json.php?x={%22name%22:%22Sudha%22,%22age%22:31,%22city%22:%22New%20York%22}
```

demo_json.php

```
<?php
echo "x is " . $_GET['x'] ;//x is {"name":"Sudha","age":31,"city":"New York"}
echo "name is " . $_GET['name'] ;//Warning:
```

```
eIn = `He's often called 'Johnny'`;//Templates allow backticks inside a string:means single quote inside single quote.
//Template literals are also useful when you need to create a string that contains multiple lines of text.
  //example:
  let textMultiline =
    `The quick
brown fox
jumps over
the lazy dog`;

document.write("<br> textMultiline : ", textMultiline);
  //The quick brown fox jumps over the lazy dog
```

jQuery as $ is used in back-tick

```
let textTemplate = `The value of x is ${xTemLit}`//not jQuery as $ is used not for dom elements tracing //it is interpolation if takin value of a variable
  var xTemLit = 5;
 document.write("<br> textTemplate : ", textTemplate); //The value of x is 5
```

Interpolation:

The syntax is:${...}//it can call any function variable or anything in the string directly.

Interpolation is the process of replacing

```
this format
  }
  document.write("<br>" + txt);

OR

function myArrObjStable(value, index, array) {// function takes 3 arguments:The item value//The item index//The array itself

Earlier codes will be as it is

}
```

JavaScript Array map()

```
map() method calls a function once for each element in an array and returns a new array with the results of applying that function to every element in the array.
const numbersMap = [45, 4, 9, 16, 25];
  const squareMap = numbersMap.map(myMapFunction);//squareMap is object of new array on which function myMapFunction will be applied.
  function myMapFunction(value, index, array) {
    // return value * value;
     return value * 2;
document.write("<br>value : " + JSON.stringify(value) + "<br>");//value : {"name":"X00","price":100}

document.write("index : " + index +
```

```
setFullYear(2026, 8, 6) :" + date);//setFullYear(2026, 8, 20) :Sun Sep 20 2026 13:57:17 GMT+0530 (India Standard Time)

date.setMonth(12);//1799223724039

date.setDate(12));//1799742889189
```

The setDate() method can also be used to add days to a date:

```
date.setDate(12
    + 10)//date.setDate(12 + 10) :1800118809189
//after setDate(12 + 10) : Mon Jan 22 2027 14
  //07:47:49 GMT+0530 (India Standard Time)
date.setDate(date.getDate()
    + 10));//date.setDate(date.getDate()+ 10) :1801471671600

date.setHours(12)
date.setHours(12, 30, 30, 100)//date.setHours(12, 30, 30, 100) : 180147
//date after setHours(12, 30, 30,100) : Mon Feb 01 2027 12:30:30.100 GMT+0530 (India Standard Time)
date.setMinutes(24);//date.setMinutes(24) :1801464870100
date.setSeconds(60)
```

NOTE:

```
compares today's date with May 24, 2200
  let textCompareDate = "";
  const today = new Date();
  const someday = new Date();
```

```
Undefined array key "name" in
C:
\Sudhadocuments\website\demo_
json.php on line 3 name is
?>
```

Receiving Data

_If you receive data in JSON format, you can easily convert it into a JavaScript object:

```
var myObjRecieve =
JSON.parse(myJSON);
myObjRecieve.name;
```

JSON From a Server

You can request JSON from the server by using an AJAX request.

As long as the response from the server is written in JSON format, you can parse the string into a JavaScript object.

Use the XMLHttpRequest to get data from the server:

```
document.write(`<br><div
id="myObjDataServer">
<button type="button"
onclick="loadDoc()">Change
Content with
AJAX</button><br>Let AJAX
change this text
</div> <br>`);
  function loadDoc() {
  var xmlhttp = new
XMLHttpRequest();
  xmlhttp.onload = function
() {
//this callback function
contain the code to execute
when the response is ready.
 var myObjDataServer=J
SON.parse(this.responseText);
document.getElementById("myOb
jDataServer").innerHTML +=
"<br>myObjDataServer.name :"
+ myObjDataServer.name;
document.write("<br><br>myObj
DataServer.name :" +
myObjDataServer.name);
//opens in main tab hence
create a innerhtml
```

placeholders in a string with actual values.

Template literals use interpolation to replace placeholders with actual values.

```
var MynameS =
"Sudha";
  var MyageS = 20;
hello = (val) =>
"Hello " + val;
  var
InterpolationText =
`Welcome ${MynameS},
${MyageS} ! ${hello}
`;
document.write("<br>
InterpolationText :
",
InterpolationText);
//Welcome Sudha,
20 ! (val) => "Hello
" + val
var
InterpolationText2 =
`Welcome ${MynameS},
${MyageS} ! $
{hello("Sana")}`;
document.write("<br>
InterpolationText2 :
",
InterpolationText2);
//InterpolationText2
: Welcome Sudha,
20 ! Hello Sana
```

Expression Substitution

```
var price = 10;
  var VAT = 0.25;
  var total =
`Total: ${(price *
(1 +
VAT)).toFixed(2)}`;
document.write("<br>
total : ", total);
//Total: 12.50
```

HTML Templates

Template literals can be

```
"<br>");//index : 0

document.write("array :
" +
JSON.stringify(array) +
"<br>");//array :
[{"name":"X00","price":1
00},
{"name":"X01","price":11
0}

document.write(value.nam
e + " " + value.price *
2 + "<br>");//X00 100

return
JSON.stringify(value) +
"=" + value.name + ": "
+ value.price * 2 + "
";//comlplete bojects
seperated by ,

  }
  //value will be passed
one by one in function
myMapFunction

document.write("<br>squa
reMap : " +
JSON.stringify(squareMap
) +
"<br>");//squareMap :
[2025,16,81,256,625]
  // [90,8,18,32,50] for
value * 2
```

the benefit of map() method is that it creates new array without affecting the original

array flat()

```
const myArrFlat = [[1,
2], [3, 4], [5, 6]];
  const newArr =
myArrFlat.flat();
  document.write("<br>",
newArr);//1,2,3,4,5,6//i
t is a flat array
```

array flatMap()

The flatMap() method first

```
someday.setFullYear(2200,
5, 24);
  if (someday > today) {
    textCompareDate =
"Today is before May 24,
2200.";
  } else {
    textCompareDate =
"Today is after May 24,
2200.";
  }
  document.write("<br> " +
textCompareDate); // Today
is before May 24,2200.
```

Objects are objects

const personNew = new Object();//now add properties//no need to use new Object build-in personNew.firstName = "Sudha";

```
Maths are objects

Functions are objects but
type of return function

Arrays are objects

Maps are objects

Sets are objects

All JavaScript values,
except primitives, are
objects
```

User defined object

//creates an empty JavaScript object, and then add properties:

const personEmpty = {};

// Using the new Keyword

personNew.firstName = "Sudha";

Properties can be primitive values, functions, or even other objects.

JavaScript Hoisting

JavaScript hoisting is a

```
   };
   xmlhttp.open("GET", "json_demo.txt");
Or
xmlhttp.open("GET", "json_demo.txt",true);//true shows that it is async ,
//Open the XMLHttpRequest object
   xmlhttp.send();//send the request to php file or server
}
```

steps of process:

An event occurs in a web page (the page is loaded, a button is clicked)

2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

json_demo.txt

```
{
   "name":"John",
   "age":31,
   "pets":[
      { "animal":"dog", "name":"Fido" },
      { "animal":"cat", "name":"Felix" },
      { "animal":"hamster", "name":"Lightning" }
   ]
}
```

NOTE: everywhere `XMLHttpRequest() , onload , open() then send()`

you can use **onreadystatechange** in the same way as onload is used

```
const xhttp = new
```

used to create HTML templates with placeholders for variables.

The placeholders are replaced with the actual values of the variables.

The resulting HTML string can be used to create a web page

The web page can be used to display data from a **database.**

```
var name = "Sudha";
  var age = 20;
  var htmlTemp = `<h1>Welcome, ${name}!</h1>
<p>You are ${age} years old.</p>`
document.write("<br> html : ", htmlTemp);
//html : <h1>Welcome, Sudha!</h1><p>You are 20 years old.</p>
Everything will be as bold as required like html tags
```

Example 2

```
var headerTemplateString = "Template Strings";
  var tagsTemplateString = ["template strings", "javascript", "es6"];
  var htmlTemplateString = `<h2>${headerTemplateString}</h2><ul>`;//ul started but not close here
  for (const x of tagsTemplateString) {
htmlTemplateString += `<li>${x}</li>`;
```

maps all elements of an array and then creates a new array by flattening the array.

```
const myArrflatMap = [1, 2, 3, 4, 5, 6];
const newArrflatMap = myArrflatMap.flatMap(x => [x + ":" + x * 10]);//1:10,2:20,3:30,4:40,5:50,6:60
```

JavaScript Array filter()

The filter() method creates a new array with all elements that pass the test implemented by the provided function.

Syntax:

```
arr.filter(callbackFn)
  arr.filter(callbackFn, thisValue)
```

`Parameters: callbackFn` - function that defines the test for each element in the array. The function takes three arguments: the current element, the index of the current element, and the array the callback was called upon.

`thisValue` - Value to use as this when executing callbackFn.

Return value: A new array with all elements that pass the test implemented by the provided function.

```
const numbersFilter1 = [1, 2, 3, 4, 5];
  const evenNumbers = numbersFilter1.filter(x => x % 2 === 0);

document.write("<br><br> numbersFilter1.filter(x => x % 2 === 0: " + evenNumbers + "<br>typeof evenNumbers :" + typeof evenNumbers + "<br>")//2,4//object
  //evenNumbers : [2, 4]//it returns a new
```

mechanism where variables and functions are moved to the top of their scope, regardless of where they are actually declared.Hoisting is JavaScript's default behavior of moving declarations to the top.

This means that variables and functions can be used before they are declared.

Hoisting is not limited to variables and functions, it also applies to function arguments and function parameters

Hoisting is a runtime behavior, not a compile-time behavior.

Hoisting is not the same as declaring a variable or function, it is a separate step that happens before the code is executed.

`xAssign = 11` // Assign 11 to xAssign //gives output even without declaration. but it is like global and default declared var and can be redeclared and assign , in that case it is not in terms of hosting

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

```
var xAssign; // Declare xAssign
  xAssign = 15;//reassign don't give error in case of var or no declaration
```

`let xDeclare; // Declare ReferenceError`

```
document.writeln("<br>xDeclare : " + xDeclare);
  // The variable xDeclare is declared with let, but
```

```
XMLHttpRequest();
xhttp.onreadystatechange =
function () {
      if (this.readyState ==
4 && this.status == 200)
{//When readyState is 4 and
status is 200, the response
is ready:
document.getElementById("demo
AJAX").innerHTML +=
this.responseText;//string
      }
   };//The
onreadystatechange event is
triggered four times (1-4),
one time for each change in
the readyState.
```

Array as JSON

When using the JSON.parse() on JSON derived from an array, the method will return a JavaScript array, instead of a JavaScript object.

```
  // JSON returned from a
server as an array:
  var xmlhttp = new
XMLHttpRequest();
  xmlhttp.onload = function
() {
    var myArr =
JSON.parse(this.responseText)
;

document.getElementById("myOb
jDataServer").innerHTML +=
"<br>myArr[0] :" + myArr[0];
  }
  xmlhttp.open("GET",
"json_demo_array.txt", true);
  xmlhttp.send();
```

json_demo_array.txt

["Ford", "BMW", "Audi", "Fiat"]

JSON PHP

A common use of JSON is to read data from a web server, and display the data in a web page.

To exchange JSON data between the client and a PHP server.

PHP has some built-in functions to

```
//adding list items
  }
  htmlTemplateString
+= `</ul>`;

document.getElementB
yId("demo").innerHTM
L =
htmlTemplateString;

document.write("<br>
htmlTemplateString
: ",
htmlTemplateString);
//it also gives
output without error
even with ,
```

boolean type variable

```
  var i = true;
  var j = false;
  if (i && true == 1) {
 console.log("yes")
 console.log("OK")}
  else {
console.log("no")}
 typeof true
// Returns boolean
  typeof false
// Returns boolean
```

Boolean Function

```
 Boolean(10 < 0)
// false
Boolean(0)
// false
 document.writeln("<
br>'the result of
10<0' :" + 10 < 0);
// false
```

NOTE:default value for empty string, undefined , null ,NaN is false default result of any text is true

Backslash

```
document.writeln("<b
r>'the result of
10\<0' :" + 10 < 0);
```

```
array with all elements
that pass the test
implemented by the
provided function.
  //filter() method does
not change the original
array.

const numbersFilter2 =
[45, 4, 9, 16, 25];
  const over18 =
numbersFilter2.filter(my
FilterFunction);
function
myFilterFunction(value)
{
    return value > 18;
  }

document.write("<br><br>
numbersFilter2.filter(my
FilterFunction): " +
over18 + "<br>")// [45,
25]
```

JavaScript Array reduce()

The reduce() method applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.

The reduce() method does not reduce the original array.

Syntax:

arr.reduce(callbackFn, initialValue)

```
const numbersReduce1 =
[1, 2, 3, 4, 5];
  const sum =
numbersReduce1.reduce((a
ccumulator,
currentValue) =>
accumulator +
currentValue,
0);//(accumulator,
currentValue) =>
accumulator +
currentValue   is
callback function and 0
is initial
```

```
not assigned a value
before it is used in the
document.writeln
statement, so it gives an
error ReferenceError.
 var xINi = 51; //
Initialize xINi
 document.writeln("<br><br
>JavaScript
Initializations are Not
Hoisted")

document.writeln("<br>xINi
= 51 + yINi : " + xINi +
yINi);// 51undefined
  var yINi = 12;//This is
because only the
declaration (var y), not
the initialization (=7) is
hoisted to the top.Because
of hoisting, y has been
declared before it is
used, but because
initializations are not
hoisted, the value of y is
undefined.
```

JavaScript Strict Mode

In strict mode, if you try to assign a value to a variable that has not been declared (either with var, let, const, or as a function parameter), you will get a ReferenceError.

Strict mode is enabled by adding the string "use strict"; at the top of a script or at the top of a function and Inside a function but using other than these 3 places are not recommanded like in block , in class , when function is called from different scope: "use strict"; (this is the most common way)

.This is a good thing, because it prevents you from accidentally creating global variables.reassigning is allowed in strict mode handle JSON.

Objects in PHP can be converted into JSON by using the PHP function json_encode():

PHP file:**demo_file.php**

The Client JavaScript

Here is a JavaScript on the client, using an AJAX call to request the PHP file then

Use JSON.parse() to convert the result into a JavaScript object:

NOTE: to see the result of php and html together in the page then use serve project not open with live server in the HTML page

```
document.write("<br><br><p id = 'myObjDataPHP'> myObjDataPHP</p>");
 var xmlhttp = new XMLHttpRequest();
  xmlhttp.onload = function () {
    var myObjClient = JSON.parse(this.responseText);

document.getElementById("myObjDataPHP").innerHTML += "<br>" + myObjClient.name;
  }
  xmlhttp.open("GET", "demo_file.php");
  xmlhttp.send();
```

demo_file.php

```
<?php
$myObj = new stdClass();
$myObj->name = "John";
$myObj->age = 30;
$myObj->city = "New York";

$myJSON = json_encode($myObj);
// Objects in PHP can be converted into JSON by using the PHP function json_encode():
echo $myJSON;
?>
```

PHP Array

_Arrays in PHP will also be converted into JSON when using the PHP function json_encode():

```
//check if the result of 10<0' : false
 Boolean('0'); // true as it takes string and anything other than 0 and false are considered true
Boolean(false == 0);//true
false == false)//false as both false shows object but only one time false shows comparision with 1,0 //object can never be compared hence false
typeof true);
//boolean
let booleanType = true;
  let booleanType1 = true;
document.writeln(booleanType1 == booleanType);//false as object is compared
typeof booleanType;
//boolean
let nullSet;//null
Boolean(nullSet));// false
Boolean(10 / 'Hello')); // false as gives NaN

var booleansObj = new Boolean(false);
typeof booleansObj); // object
booleansObj.toString()// 'false'
boolean(booleansObj == booleanType)); // error
booleansObj.valueOf(); // false
```

JavaScript Regular Expressions

Regular expressions are used for

```
value//accumulator gets updated every time acc to return value accumulator + currentValue //here sum is the single value that returns from the function

document.write("<br><br>numbersReduce1.reduce((accumulator, currentValue) => accumulator + currentValue: " + sum + "<br>")//1+2+3+4+5=15
  //15//it returns the sum of all elements in the array.
  //The reduce() method does not change the original array.
  //The accumulator is the initial value, if supplied, or the first element in the array.It updates in every iteration
  //The currentValue is the current element being processed in the array.

  const numbersReduce2 = [45, 4, 9, 16, 25];
  //let sum2 = numbersReduce2.reduce(myFunctionReduce);
  let sum2 = numbersReduce2.reduce(myFunctionReduce, 0);//initialized 0 is for total tostart
  function myFunctionReduce(total, value, index, array) {//The total (the initial value / previously returned value)
     //The item value
```

The "use strict" directive is only recognized at the beginning of a script or a function and inside function but not above variable

Strict mode makes it easier to write "secure" code by making some restrictions and warnings. In strict mode, you ***cannot use*** the following keywords as variable names: arguments, caller, error, eval, this, undefined, implements,interface,let,package,private,protected,public,static,yield

In "Strict Mode", undeclared variables are not automatically global.

```
"use strict"; // Enable strict mode for the current scope
Inside a function: "use strict"; (this is the most common way)
xstrictT = 3.14; // This will cause an error because x is not declared and "use strict"; declared at then beginning of script
  var xstrict = 5;// Declare and initialize xstrict

document.writeln(xstrict);//5
  xstrict = 10;//reassigning is allowed in strict mode

document.writeln(xstrict);//10
"use strict";//not works even when just above function
```

PHP file:demo_file_array.php

NOTE:make sure that no such statement should be present <!-- Use $_POST instead of $_GET: --> outside the php file and without any enclosure otherwise it gives parse error

PHP File explained:

Convert the request into an object, using the PHP function json_decode().

Access the database, and fill an array with the requested data.

Add the array to an object, and return the object as JSON using the json_encode() function.

Use **JSON.parse()** to convert the result into a JavaScript array:

```
 var xmlhttp = new XMLHttpRequest();
  xmlhttp.onload = function () {
    const myObj = JSON.parse(this.responseText);//Use the Data passed by php
document.getElementById("myObjDataPHP").innerHTML += "<br>" + myObj[2];
  }
  xmlhttp.open("GET", "demo_file_array.php", true);
  xmlhttp.send();
```

demo_file_array.php

```
<?php
$myArr = array("John", "Mary", "Peter", "Sally");
$myJSON = json_encode($myArr);
echo $myJSON;
{"name":"Sudha","age":30,"city":"Lucknow"}
//{"name":"Sudha","age":30,"city":"Lucknow"}
?>
```

Example explained for all php json:

Define an object containing a "limit" property and value.

Convert the object into a JSON string.

matching a pattern in a string.

They are used for searching, validating and extracting data from strings.

Syntax:

```
/pattern/modifiers;//eg : text.search(/shanaya/i)
var text = "Hello, world! Visit Shanaya Ecofriendly Bazaar and Get in touch of Sudha 's Shanaya shanaya ";
```

The pattern is the regular expression itself, and the modifiers are flags that can be used to modify the behavior of the regular expression.

The most common modifiers are

The most common modifiers are

'g' for global search (find all),// The 'g' flag is used to find all matches in a string, not just the first

```
    var regex = /hello/g;//global search
```

The search() method scans through a string looking for a match to the pattern, and returns an index of the match. If no match is found, it returns -1.

```
    var resultT = text.search(regex);
document.writeln("<br>" + resultT);//-1//-1 index means not found due to case sensitive
```

```
    //The item index
    //The array itself

document.write("total :" + total + " " + "value :" + value + "<br>");//total :0 value :45 total :45 value :4 total :49 value :9 total :58 value :16 total :74 value :25
    return total + value;
    //last value of total will be stored in sum and displayed at end
  }

document.write("<br><br> numbersReduce2.reduce(myFunctionReduce): " + sum2 + "<br>")
```

reduceRight()

The reduceRight() method applies a function against an accumulator and each element in the array (from right to left) to reduce it to a single value.

```
let sumReduceRight2 = numbersReduceRight2.reduceRight(myFunctionReduceRight);//it wil, start from last and take last as total and last second as value//here bydefault initial value is last index
let sumReduceRight2 = numbersReduceRight2.reduceRight(myFunctionReduceRight,-1);//total :-1 value :25
  total :24 value :16
  total :40 value :9
  total :49 value :4
  total :53 value :45

let sumReduceRight2 = numbersReduceRight2.redu
```

call ,it must be top statement of script tag

```
  mFunctionstrict();
  function myFunctionstrict() {
      y = 3.14;// This will cause an error (y is not defined)//ReferenceError: Cannot access 'y' before initialization at myFunctionstrict //this error not because of "use strict" but it is because of not declaring y anywhere.
//inside function
 "use strict"
  z = 3.14;  // This will cause an error (z is not defined, ReferenceError: Cannot access 'z' before initialization at myFunction
  }
```

Deleting a variable (or object) is not allowed in strict mode //SyntaxError: Delete of an unqualified identifier in strict mode.

```
  xOBJStrict = { p1: 10, p2: 20 };
//Using an object, without declaring it, is not allowed: Objects are variables too.
let xOBJStrictDelTry = 132;
delete xOBJStrictDelTry;
//Deleting a variable (or object) is not allowed in strict mode //SyntaxError: Delete of an unqualified identifier in strict mode.
function xFunD(p1, p2) { };
```

Send a request to the PHP file, with the JSON string as a parameter.

Wait until the request returns with the result (as JSON)

Display the result received from the PHP file.

PHP Database

PHP is a server side programming language, and can be used to access a database.

Imagine you have a database(shanayawork) on your server, and you want to send a request to it from the client where you ask for the 5 first rows in a table called "program".

On the client, make a JSON object that describes the numbers of rows you want to return.

NOTE: HTML as a client and server interacts

Before you send the request to the server, convert the JSON object into a string and send it as a parameter to the url of the PHP page:

```
 // PHP File explained: json_demo_db.php
```

// Convert the request into an object, using the PHP function json_decode().

// Access the database, and fill an array with the requested data.

// Add the array to an object, and return the object as JSON using the json_encode() function.

```
var limit = { "limit": 5 };
  var dbParam = JSON.stringify(limit);
  xmlhttp = new XMLHttpRequest();
  xmlhttp.onload = function () {
document.getElementById("myObjDataPHP").innerHTML += this.responseText;
console.log(typeof this.responseText);//string

  }
```

```
text.match(/Shanaya/g);// Outputs: ["Shanaya", "Shanaya"] //it finds all and it is case sensitive
```

The replace() method replaces a specified value with another value in a string. It returns a new string with the replacement made, and leaves the original string unchanged.

```
  Syntax : var resultReplace = text.replace(old,new);
var resultReplace = text.replace(/shanaya/i, "Shanaya World");
//Regular expression arguments (instead of string arguments) can be used in the methods above.Regular expressions can make your search much more powerful (case insensitive for example).
  var resultReplace = text.replace('Shanaya', "Shanaya World");

document.writeln("<br>text.replace(/Shanaya/,'Shanaya World') :" + resultReplace); // Output: Visit Shanaya World Bazaar

//Use String replace() With a
```

```
ceRight(myFunctionReduceRight, 0);
  function myFunctionReduceRight(total, value) {
//document.write("total :"+total +" "+"value :"+value+"<br>");//total :0 value :45 total :45 value :4 total :49 value :9 total :58 value :16 total :74 value :25
document.write("total :" + total + " " + "value :" + value + "<br>");
    return total + value;//total :25 value :16 total :41 value :9 total :50 value :4 total :54 value :45//it wil, start from last and take last as total and last second as value
    //last value of total will be stored in sum and displayed at end
  }
document.write("<br><br>numbersReduce2.reduce(myFunctionReduce): " + sumReduceRight2 + "<br>")
```

every() method

The every() method tests whether all elements in the array pass the test implemented by the provided function

```
const numbersEvery = [7, 4, 9, 5, 5];
  let resultEvery = numbersEvery.every(myFunctionEvery);//every method itself  calls the function for every element until return is false or 0.
function
```

```
delete xFunD;//Deleting a function is not allowed.
 function x(p1, p1) { };
// This will cause an error as Duplicating a parameter name is not allowed:
 let xOctal = 010;//error as Octal numeric literals are not allowed:
let xOctal2 = 0x10;//error as Hexadecimal numeric literals are not
let xOctal3 = 0b10;//error as Binary numeric literals are not allowed
let xOctal4 = 0o10;//error as Octal numeric literals are not allowed
let xOctal5 = 0x10.2;//error as Octal numeric
let xOctalEscape = "\010";//This will cause an error //Octal escape characters are not allowed:
const obj = {};This line initializes an empty object called obj. The const keyword means that obj cannot be reassigned to another value (though the contents of the object can still be modified, unless those properties are frozen).
Object.defineProperty(obj, "xRead", { value: 0, writable: false });The code provided defines a new object obj and then uses the Object.defineProperty method to create a property named xRead
 on that object obj. The method takes three parameters:
```

The ***first parameter*** is the object on which the property is to be defined (obj).

```
  xmlhttp.open("GET",
"json_demo_db.php?x=" +
dbParam);
  xmlhttp.send();
```

json_demo_db.php

```
<?php
header("Content-Type:
application/json;
charset=UTF-8");

// Get and decode JSON input,
ensure it's a valid
request //not necessary if no
error
if (!isset($_GET["x"])) {
  echo json_encode(["error"
=> "No input provided"]);
  exit();
}
$obj =
json_decode($_GET["x"],
false);
//not necessary if integer
passed in limit
if (!isset($obj->limit) || !
is_numeric($obj->limit)) {
  echo json_encode(["error"
=> "Invalid limit
parameter"]);
  exit();
}

// Sanitize and convert limit
to integer then use type "i"
in bind_param but if this
conversion not used the type
"s" or "i" both can be used
$limit = intval($obj->limit);

// Database connection setup
$conn = new
mysqli("localhost", "root",
"#sudha@2402",
"shanayawork");

// Check connection
if ($conn->connect_error) {
  echo json_encode(["error"
=> "Database connection
failed: " . $conn-
>connect_error]);
```

```
Regular Expression
and a Function
  var
resultReplaceFunctio
n =
text.replace(/shanay
a/i, function
(match) {//match
parameter is not in
use here but it can
be used to pass
value
    return "Shanaya
World";
  });
 document.writeln("<b
r>text.replace(/
shanaya/
i,function(match)
{return 'Shanaya
World'} : " +
resultReplaceFunctio
n); // Output: Visit
Shanaya World
```

'i' for case-insensitive search, and

```
   var regex =
/hello/i;//case
insensitive
    var resultT =
text.search(regex);
document.writeln("<b
r>" +
resultT);//0//0
index
```

'm' for multi-line search. The 'm' flag is used to perform start and end matching, so that ^ and $ match the start and end of each line, not just the start and end of the entire string

```
var regex =
/\nW/gm;//strM.match
(regex) : W
```

The match() method

```
myFunctionEvery(value,
index, array) {

document.write("value :"
+ value + " " +
"index :" + index + " "
+ "array :" + array +
"<br>");//item :1 index
:0 array :7, 2, 9, 4, 5
 return value > 0;//it
will execute everytime
until the value is
greater than zero//it
return true it executed
for upto last index
//incase of modulus when
it returns 0 then it
will be considered as
return false hence it
comes out of function.
}
document.write("numbersE
very.every(myFunctionEve
ry): " + resultEvery +
"<br>");//false
  //true for value>0
```

Array some()

```
The some() method tests
whether at least one
element in the array
passes the test
implemented by the
provided function
const numbersSome = [7,
4, 9, 5, 5];
  let resultSome =
numbersSome.some(myFunct
ionSome);
function
myFunctionSome(value,
index, array) {
return value%4;//it
returns true in the very
first by giving 7%4=3
and 3 is not false hence
a kind of (true) ,hence
stop here and returns
true.
 }
```

Array.from()

The Array.from() method

The ***second parameter*** is the name of the property as a string ("x").

The ***third parameter*** is a descriptor object that specifies the behavior of the property,contains several attributes that define the characteristics of the property:

```
 //value:0, 0 means that
the initial value of the
property xRead will be 0.
, writable:false are the
property of any property
in object.The value
property specifies the
value of the property.The
writable property
specifies whether the
value of the property can
be changed or not after it
has been defined.
obj.xRead = 3.14;//here we
are trying to assign the
value to the xRead
property of obj
object//xRead is the
property we want to add in
obj
//TypeError: Cannot assign
to read only property
'xRead' of object
'#<Object>' at obj.xRead =
3.14; // This will cause
an error (xRead is read-
only) as writeable is
false
console.log(obj.xRead); //
Outputs: 0
```

Checking if x is an own property of obj:

```
console.log(obj.hasOwnProp
erty('x')); // Outputs:
true
```

Non-enumerable and Non-configurable attributes:

By default, if you do not specify other attributes, properties defined with Object.defineProperty are: ***enumerable: false:*** The

```
  exit();
}

// Prepare SQL statement
$stmt = $conn->prepare("SELECT * FROM program LIMIT ?");
if (!$stmt) {
  echo json_encode(["error" => "Statement preparation failed: " . $conn->error]);
  exit();
}

// Bind parameters
$stmt->bind_param("i", $limit);//i used as intval is used to convert into integer but s can also be used
$stmt->bind_param("s", $limit);

// Execute and get result
$stmt->execute();//when statement is prepared then it must contain execute otherwise use sql directly

$result = $stmt->get_result();
$outp = $result->fetch_all(MYSQLI_ASSOC);

// Close connections
$stmt->close();
$conn->close();

// Output the result as JSON
echo json_encode($outp);
?>
```

Use the Data from database(no extra data ouside php otherwise it gives parse error)

```
var xmlhttp = new XMLHttpRequest();
  xmlhttp.onload = function () {
    const myObj = JSON.parse(this.responseText);
```

returns an array containing the matched text, or null if no match is found

```
text.match(/Shanaya/m);// Outputs: Shanaya
```

The 'y' flag is used to perform a sticky search, so that the search starts at the current position in the string, not at the start of the string

```
var regex = /hello/y;//sticky search
```

d :Perform start and end matching //start or end not start and end //it's like start and end together and at start and then at end //It only contain 3 element in output

```
var resultT = text.match(/(aa)(bb)/d);//null
var resultT = text.match(/(of)(r)/d);// ofr,of,r //it is same as predicted
```

-

The 'u' flag is used to perform a Unicode code point search, so that the search matches Unicode code points, not just characters

The 's' flag is used to perform a dotall search, so that the dot ( ) matches any character, including a newline

The 'x' flag is used to ignore

creates a new, shallow-copied Array instance from an array or an iterable object(like string,array or object etc).

```
Array.from("ABCDEFG");
  //Output: ["A", "B", "C", "D", "E", "F", "G"]
 typeof Array.from('ABCDEFG') :// object
  Array.from('ABCDEFG')[3] :// D
```

Arrays keys() Method

The keys() method returns a new Array Iterator object that contains the keys for each index in the array.

```
const fruitsKey = ["Banana", "Orange", "Apple", "Mango"];
const fruitsKeyIterator = fruitsKey.keys();//it is in form of array and [] is present //type =object
```

for of used to get elements

```
  let textsKey = "";
  for (let x of fruitsKeyIterator) {
    textsKey += x + " ";
  }
//here key is index no.
  document.write(
    "<br><br>fruitsKey : [" + fruitsKey + "]<br>" +
    "fruitsKeyIterator : [" + fruitsKeyIterator + "]<br>" +
    "textsKey : [" + textsKey + "]<br>" +
    "typeof fruitsKeyIterator : " + typeof fruitsKeyIterator
  ); //fruitsKey : [Banana,Orange,Apple,Mango]
  //fruitsKeyIterator : [[object Array
```

property will not show up in for...in loops or Object.keys().

```
Enumerating properties of obj:
  for (const key in obj) {
    console.log(key); // This won't log anything because x is non-enumerable
  }
```

configurable: false: The property cannot be deleted and its attributes cannot be changed to make it writable or enumerable.

```
 Attempting to change property attributes:
  // This will throw a TypeError in strict mode

Object.defineProperty(obj, "x", { writable: true });
// Error: Cannot redefine property: x

const obj = { get x() { return 0 } };
obj.x = 3.14;// This will cause an error as Writing to a get-only property is not allowed:
delete Object.prototype;
// This will cause an error because Deleting an undeletable property is not allowed:
All below variable name will cause an error.In strict mode, you cannot use the following keywords as variable names:
arguments, caller, error, eval, this, undefined.
let eval = 3.14;
let arguments = 3.14;
with (Math) { x = cos(2) }; // This will cause an error.The with statement is not allowed:
//For security reasons, eval() is not allowed to
```

```
console.log(typeof
myObj);//object
    let text = "";
    for (let x in myObj) {
      text +=
myObj[x].PROGRAM_NAME
 + "<br>";
    }

document.getElementById("myObjDataPHP").innerHTML +=
"<br>text" + text;
  }
  xmlhttp.open("GET",
"json_demo_db.php?x=" +
dbParam);
  xmlhttp.send();
```

PHP Method = POST

When sending data to the server, it is often best to use the HTTP POST method.

// To send AJAX requests using the POST method, specify the method, and the correct header.

// The data sent to the server must now be an argument to the send() method:

```
 //The only difference in the
PHP file is the method for
getting the transferred data.
  //Use $_POST instead of
$_GET:
```

In HTTP there are two ways to POST data: application/x-www-form-urlencoded and multipart/form-data. I understand that most browsers are only able to upload files if multipart/form-data is used

```
var dbParam =
JSON.stringify({ "limit":
5 });
    var xmlhttp = new
XMLHttpRequest();
    xmlhttp.onload = function
() {
      myObj =
JSON.parse(this.responseText)
```

whitespace in the regular expression, so that whitespace characters are treated as literal characters, not as part of the regular expression

The pattern can include special characters that have special meanings in regular expressions. Quantifiers define quantities For example, '.' matches any character,

'*' matches zero or more occurrences of the preceding character,

```
var resultTMu =
text.match(/a*/g);//
,,,,,,,,,,,,,,,,,,,,
,,a,,a,,a,,,,,,,,,,,
,,,,a,,aa,,,a,,,,,,,
,,,,,,,,,,,,,,,,,a,,
,,,,,a,,a,,a,,,,a,,a
,,a,,//between
two ,, is one letter
which is not a
```

n* Matches any string that contains zero or more occurrences of n

'+' matches one or more occurrences of the preceding character means atleast 1, //n+ Matches any string that contains at least one n

```
var resultTA =
text.match(/a+/g);//
a,a,a,a,aa,a,a,a,a,a
,a,a,a//Matches any
string that contains
at least one a
```

'?' matches zero

```
Iterator]]
  //textsKey : [0 1 2
3 ]
  //typeof
fruitsKeyIterator :
object
```

JavaScript Array entries()

The entries() method returns a new Array Iterator object that contains the key-value pairs for each element in the array.

```
const fruitsEnteries =
["Banana", "Orange",
"Apple", "Mango"];
  const
fruitsEnteriesIterator
    =
fruitsEnteries.entries()
; //returns an iterator
object with key-value
pairs
  textsEnteries = "";
  for (let x of
fruitsEnteriesIterator)
{
    //console.log(x);
    textsEnteries += x +
"<br> ";
  }
  document.write(

"<br><br>fruitsEnteries
: [" + fruitsEnteries +
"]<br>"
    +
"fruitsEnteriesIterator
: " +
fruitsEnteriesIterator +
"<br>" +
    "typeof
fruitsEnteriesIterator :
" + typeof
fruitsEnteriesIterator
    + "<br>
textsEnteries :
<br>[<br>" +
textsEnteries + "]<br>"

  ); //fruitsEnteries :
[Banana,Orange,Apple,Man
```

create variables in the scope from which it was called.In strict mode, a variable can not be used before it is declared:

```
eval("x = 2");
alert(x);// This will
cause an error as x not
declared
// In strict mode, eval()
can not declare a variable
using the var,let,const
keyword:
eval("var x = 2");
alert(x);// This will
cause an error
eval("let x = 2");
alert(x);// This will
cause an error
```

The this keyword refers to the object that called the function. If the object is not specified, functions in strict mode will return undefined and functions in normal mode will return the global object (window):

The this keyword in functions behaves differently in strict mode.

```
 "use strict"; //
functions f() in strict
mode
 function f() {
 alert(this); // will
alert "undefined"
return this;
}
alert(f()); // This will
cause an error. The
function f() returns
undefined in strict mode
document.writeln("<br>xOBJ
Strict = {p1:10,
p2:20}");
document.writeln("<br>xOBJ
StrictDelTry = 132 : " +
xOBJStrictDelTry);
document.writeln("<br>xFun
D = function(p1, p2) {} :
" + xFunD);
```

```
;
      let text = "";
      for (let x in myObj) {
        text +=
myObj[x].PROGRAM_NAME +
"<br>";
 //Computer Science
      // Mathematics
      // Biology
      // Chemistry
      // Physics
      }

document.getElementById("myObjDataPHP").innerHTML += text;
    }
    xmlhttp.open("POST",
"json_demo_db_post.php");

xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xmlhttp.send("x=" +
dbParam);
```

json_demo_db_post.php

```
<!-- Use $_POST instead of
$_GET: -->This will give
error when parsing hence
don't use it ouside the
php//remove it//even
commented < will be counted
and it gives error

<?php
header("Content-Type:
application/json;
charset=UTF-8");
$obj =
json_decode($_POST["x"],
false);
$conn = new
mysqli("localhost", "root",
"#sudha@2402",
"shanayawork");
$stmt = $conn-
>prepare("SELECT PROGRAM_NAME
FROM program LIMIT ?");
$stmt->bind_param("s", $obj-
>limit);
$stmt->execute();
$result = $stmt-
```

```
or one occurrence of
the preceding
character means
atmost 1,n? Matches
any string that
contains zero or one
occurrences of n
var resultTQ =
text.match(/a?/g);//
,,,,,,,,,,,,,,,,,,,,
,,a,,a,,a,,,,,,,,,,,
,,,,a,,a,a,,,a,,,,,,
,,,,,,,,,,,,,,,,,,a,
,,,,,,a,,a,,a,,,,a,,
a,,a,,
  '{n}' matches
exactly n
occurrences of the
preceding character,
 var resultTE =
text.match(/a{3}/g);
//null
  '{n,m}' matches
atleast n and atmost
m occurrences of the
preceding character
var resultTR =
text.match(/a{3,5}/g
);//null
  '{n,}' matches
atleast n
occurrences of the
preceding character
and here comma is
added to make
atleast otherwise it
would have become
exact no. ,
var resultTM =
text.match(/a{3,}/g)
;//null
 var text = "Hello,
world! Visit Shanaya
Ecofriendly Bazaar
and Get in touch of
Sudha 's Shanaya
shanaya 44332 ";
Other mstches
'[abc]' matches any
of the characters
'a', 'b', or 'c',
```

```
go]

//fruitsEnteriesIterator
: [[object Array
Iterator]]
  //typeof
fruitsEnteriesIterator :
object
    textsEnteries :
  [
  0,Banana
  1,Orange
  2,Apple
  3,Mango
  ]

  //note bydefault ,[ ]
is not shown on printing
the textsEnteries ,but
it gets stored in the
same format of array
```

Array with() Method

with() method as a safe way to update elements in an array without altering the original array.

```
 syntax:array.with(index
,"new Element")
 parameters are required
to update specific index
value with new value but
if it is not given any
parameter than it will
consider as to update 0
index with blank " "

const fruitsWith =
["Banana", "Orange",
"Apple", "Mango"];

  const
fruitsWithIterator =
fruitsWith.with();
//returns an iterator
object with key-value
pairs//it does not
affect the
original//fruitsWithIter
ator :  ,Orange,Apple,Man
go////,Orange,Apple,Mang
```

```
    }
z = 3.14;
  //
document.writeln("<br>z =
3.14 : " + z);//z = 3.14
//it is not in strict mode
for outside
```

JavaScript Performance

Reduce Activity in Loops

Statements or assignments that can be placed outside the loop will make the loop run faster.

Bad

```
  for (let i = 0; i <
arr.length; i++) {
    console.log(arr[i]);
  } // 1.5 seconds
  console.log(arr.length);
// 0.1 seconds
```

Good

```
let l =
arr.length;//reduce
activity in loop, it does
not need to check l again
in each loop
  for (let i = 0; i < l;
i++) {
    console.log(arr[i]);
  } // 0.05 seconds
```

Reduce Function Calls

Reduce DOM Access

Accessing the DOM is expensive. If you need to access the DOM inside a loop, try to reduce the number of times you access it.

The bad code accesses the DOM each time the loop is iterated.

The better code accesses the DOM outside the loop and makes the loop run faster.

If you expect to access a DOM element several times, access it once, and use it as a local variable:

Example: Accessing the DOM inside a loop

```
  let elements =
>get_result();
$outp = $result->fetch_all(MYSQLI_ASSOC);
echo json_encode($outp);
?>
```

JSON HTML

// JSON can very easily be translated into JavaScript.
// JavaScript can be used to make HTML in your web pages.
To Make an HTML table with data received as JSON:

```
document.write("<br><p id = 'myObjJSONHTML'>myObjJSONHTML</p>");
  var dbParam = JSON.stringify({ table: "program", limit: 3 });
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onload = function() {
    var myObj = JSON.parse(this.responseText);
    let text = "<table border='1'>"
    for (let x in myObj) {
      text += "<tr><td>" + myObj[x].PROGRAM_NAME + "</td></tr>";
    }
    text += "</table>"
document.getElementById("myObjJSONHTML").innerHTML = text;
  }
  xmlhttp.open("POST", "json_demo_html_table.php");
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xmlhttp.send("x=" + dbParam);
```

json_demo_html_table.php

```
<?php
header("Content-Type:application/json; charset=UTF-8");
$obj =
```

```
var resultT = text.match(/[SaB]/g);// S,a,a,a,B,a,a,a,a,S,a,S,a,a,a,a,a,a,a
var resultT = text.match(/[4]/g);// 4,4
  '[^abc]' matches any character that is not 'a', 'b', or 'c',
  '[a-z]' matches any character in the range 'a' to 'z',
var resultT = text.match(/[a-z]/g);//:H,e,l,l,o,w,o,r,l,d,i,s,i,t,h,a,n,a,y,a,c,o,f,r,i,e,n,d,l,y,a,z,a,a,r,a,n,d,e,t,i,n,t,o,u,c,h,o,f,u,d,h,a,s,h,a,n,a,y,a,s,h,a,n,a,y,a // it has ignored capital letters and symbols
  '[A-Z]' matches any character in the range 'A' to 'Z',
  '[0-9]' matches any digit in the range '0' to '9',
  (x|y)  Find any of the alternatives separated with |
same result as [abc] and [x|y]

var resultT = text.match(/(a|z)/g);//a,a,a,a,z,a,a,a,a,a,a,a,a,a,a
  var resultT = text.match(/[az]/g);//a,a,a,a,z,a,a,a,a,a,a,a,a,a,a
var resultT = text.match(/[a|z]/g);//a,a,a,a,z,a,a,a,a,a,a,a,a,a,a
```

```
o//here 0th index value is not shown as it by default take as fruitsWith.with(0,"")//hence first index element is not seen
  const fruitsWithUpdate = fruitsWith.with(2, "Pineapple");//araay elemnts in string must be updated in form of string "Pineapple" works but Pineapple not works//it update 2nd index value from apple to pineapple
```

Array Spread (...)

Spread syntax (...) is used to unpack the elements of an array into a new array.

Syntax: [...arrayName] or [...arrayName, ...arrayName2] or [...arrayName, ...arrayName2, ...arrayName3]

```
const q1 = ["Jan", "Feb", "Mar"];

  const q2 = ["Apr", "May", "Jun"];
  const q3 = ["Jul", "Aug", "Sep"];
  const q4 = ["Oct", "Nov", "May"];
  const q5 = [...q1, ...q2, ...q3, ...q4];
  document.write(
"q5 = [...q1, ...q2, ...q3, ...q4] <br>q5 : [" + q5 + "]<br>"

    + "typeof q5 : " + typeof q5//object
 );

document.write("<br>", typeof (myarray))//object
```

```
document.querySelectorAll('.my-class');//DOM value stored in element and now DOM need not to access for each iteration
  for (let i = 0; i < elements.length; i++) {
elements[i].style.color = 'red';
  } // 1.5 seconds
  const obj = document.getElementById("demo");
  obj.innerHTML = "Hello";
```

Reduce DOM Size

Keep the number of elements in the HTML DOM small.

This will always improve page loading, and speed up rendering (page display), especially on smaller devices.

Every attempt to search the DOM (like getElementsByTagName) will benefit from a smaller DOM.

Avoid Unnecessary Variables

Don't create new variables if you don't plan to save values.

Often you can replace code like this:

```
  let fullName = firstName + " " + lastName;
document.getElementById("demo").innerHTML = fullName;
  With this:
document.getElementById("demo").innerHTML = firstName + " " + lastName;//here fullname variable is not created unnecessary
```

Delay JavaScript Loading

If you have a lot of JavaScript code, consider delaying its execution until the page has finished loading

This can be done by putting the code in a function and

```
json_decode($_POST["x"], false);
$conn = new mysqli("localhost", "root", "#sudha@2402", "shanayawork");
$tableSelected=$obj->table;
$limitSet=$obj->limit;
$stmt = $conn->prepare("SELECT * FROM $tableSelected LIMIT $limitSet");
```

NOTE:bind_param is not only for one ? But also variables For more than one ?

```
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
FROM fruit
WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindParam(2, $colour, PDO::PARAM_STR);
$sth->execute();
```

Refer this to learn about bindParam more

PHP: PDOStatement::bindParam - Manual

/* just extra details

NOTE:if you are using where clause then forget the datatype of each column

```
$sql = "SELECT employee_id,first_name,last_name,position,salary FROM $tableSelected WHERE employee_id=$employeeIDSelected AND first_name=$UsernameSelected"
```

```
var resultT = text.match(/[a-zA-Z0-9]/g);//:H,e,l,l,o,w,o,r,l,d,V,i,s,i,t,S,h,a,n,a,y,a,E,c,o,f,r,i,e,n,d,l,y,B,a,z,a,a,r,a,n,d,G,e,t,i,n,t,o,u,c,h,o,f,S,u,d,h,a,s,S,h,a,n,a,y,a,s,h,a,n,a,y,a,4,4,3,3,2
```

Metacharacters are characters with a special meaning:

'\d' matches or find any digit

'\D matches any non-digit,

'\s' matches any whitespace character,

'\S' matches any non-whitespace character,

'\b' matches the start of a string, and it also matches the end of a string,Example : at the beginning of a word l like this: \bWORD, or at the end of a word like this: WORD\b

'\B' matches any position where the current character is not the start or end of a word//'\B' matches any position where the specified characters are not on the list of word characters,// '\B matches any character that is not a word boundary,

'\w' matches any alphanumeric character, and it also matches the

Const

A constant variable cannot be reassigned or changed once it is declared in same block but it can be updated with index

An array declared with const has Block Scope.

error if same name in same block even though elements are different

JavaScript const variables must be assigned a value when they are declared otherwise gives error even if it constant object is initialized below

```
//An array declared with const has Block Scope.
  const fruitsBlockTry = ["Banana", "Orange", "Apple", "Mango"];
  {
    const fruitsBlockTry = ["Ban", "Orange", "Apple", "Mango"];

document.write("<br>I am inside blook{} and my 0th index element is fruitsBlockTry[0] :" + fruitsBlockTry[0])//Ban
  }
  document.write("<br>I am outside blook{} and my 0th index element is fruitsBlockTry[0] :" + fruitsBlockTry[0])//Ban
```

Use const when you declare so that value should not be changed.

```
  A new Array
  A new Object
  A new Function
  A new RegExp
```

calling that function at the end of the body tag:

Putting your scripts at the bottom of the page body lets the browser load the page first.

While a script is downloading, the browser will not start any other downloads. In addition all parsing and rendering activity might be blocked.

The ***HTTP specification*** defines that browsers should not download more than two components in parallel.

An alternative is to use ***defer="true"*** in the script tag. The defer attribute specifies that the script should be executed after the page has finished parsing, but it only works for external scripts.

Use the defer attribute to load scripts asynchronously, but only if the script is not used to add event listeners to elements that are not yet available in the DOM.

```
 <script defer src="script.js"><script> // defer attribute is used here <script src="script.js"></cript> // defer attribute is not used here
```

If possible, you can add your script to the page by code, after the page has loaded:

Avoid Using with

Avoid using the with keyword. It has a negative effect on speed. It also clutters up JavaScript scopes.

The with keyword is not allowed in strict mode.

JavaScript Reserved Words

JavaScript has a number of reserved words that cannot be used as variable names,labels, or function names. These include:

```
;//not works due to no '' in $UsernameSelected as it gives ERROR 1054 (42S22): Unknown column 'Sudha' in 'where clause' . Hence use '' even after username is passed variable
$stmt = $conn->prepare($sql);

$stmt = $conn->prepare("SELECT * FROM $tableSelected WHERE employee_id =$employeeIDSelected AND first_name='$UsernameSelected'");//works
*/

$stmt->execute();
$result = $stmt->get_result();
$outp = $result->fetch_all(MYSQLI_ASSOC);
echo json_encode($outp);
?>
```

Using json php and Database together In ***json_demo_db.php*** instead of echo json_encode($outp); which print the data in json format in get html file , we can create table from the $outp and write html below code in the same ***json_demo_db.php*** file but note that this php file along with html in it can't be used in POST otherwise it gives < parse error

OR

create $outp = '[{"STUDENT_REF_ID":201,"PROGRAM_NAME":"Computer Science","PROGRAM_START_DATE":"2021-09-01 00:00:00"},{"STUDENT_REF_ID":202,"PROGRAM_NAME":"Mathematics","PROGRAM_START_DATE":"2021-09-01 00:00:00"},{"STUDENT_REF_ID":203,"PROGRA

underscore character,

'\n' matches a newline,
'\r' matches a carriage return,
'\t' matches a tab,
'\f' matches a form feed,
'\v matches a vertical tab,
'\0' matches a null character,

'[a-zA-Z]' matches any character in the range 'a' to 'z' or 'A' to 'Z',here no gap between
'[a-zA-Z0-9]' matches any character in the range 'a' to ' 'z' or 'A' to 'Z' or '0' to '9',
'[^0-9]' matches any character that is not a digit,
'\w' matches any word character (equivalent to [a-zA-Z])
'\W' matches any non-word character (equivalent to [^a-zA-Z0-9_
'\xhh' matches a character with the hexadecimal value hh,
'\xX' matches the character X, where X is a hexadecimal digit,
\uxxxx Find the Unicode character specified by the

var Array

```
fruitsConstNoValue = ["Banana", "Orange", "Apple", "Mango"];//no error even when type not specified but type can be declare later as var fruitsConstNoValue; but const fruitsConstNoValue; will gives error and const can't be emptied neither above nor below
```

Arrays declared with var can be initialized at any time. Arrays declared with var can be initialized at any time. Redeclaring an array declared with var is allowed anywhere in a program: `var is hoisted`.//You can use the variable before it is declared//Variables defined with var are hoisted to the top and can be initialized at any time.//carName = "Volvo";

```
  var carName;
```

var binds to this.

Variables declared with the var always have Global Scope.

Variables declared with varinside a { } block can be accessed from outside the block:

let variable

let and const must be declared before use.

Using a let variable before it is declared will result in a ReferenceError:

```
  carName = "Saab";
  let carName =
"Volvo";//error
```

const, let, var, function, class, export, import, default, yield,break, case, catch, continue, debugger, default, delete, do, else,finally, for, for...in, for...of, function, if, in,instanceof, new, return, switch, this, throw, try, typeof, void, while, with, export, import, let, static, super, yield, await,enum, extends, implements, interface, package, private, protected, public, static,abstract, boolean, byte, char, class, const, double, enum, export,extends, final, finally, float, for, goto, if, implements, import,in, instanceof, int, interface, long, native, new, package, private,protected, public, return, short, static, super, switch, synchronized,this, throw, throws, transient, try, void, volatile, while, boolean,byte, case, catch, char, class, const, continue, debugger, default,do, double, else, enum, export, extends, false, final, finally,float, for, function

abstract arguments

await* boolean break byte case catch char class* const* continue debugger default delete do double else enum* eval export* extends* false final finally float for function goto if implements import* in instanceof int interface let* long native new null package private protected public return short static super* switch synchronized this throw throws transient true try typeof var void volatile while with yield

Removed Reserved Words

The following reserved words have been removed from the ECMAScript 5/6 standard:

```
M_NAME":"Biology","PROGRAM_START_DATE":"2021-09-01 00:00:00"},
{"STUDENT_REF_ID":204,"PROGRAM_NAME":"Chemistry","PROGRAM_START_DATE":"2021-09-01 00:00:00"},
{"STUDENT_REF_ID":205,"PROGRAM_NAME":"Physics","PROGRAM_START_DATE":"2021-09-01 00:00:00"}]';
$data = json_decode($outp, true);
```

manually like this then dbms connection will not be needed not any GET or POST

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JSON Data to HTML Table</title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <!-- yhis bootstrap gives table structure bold and big -->
</head>
<body>
<div class="container mt-5">
  <h2 class="mb-4">JSON Data to HTML Table</h2>
  <table class="table table-bordered">
    <thead>
      <tr>
        <th>STUDENT_REF_ID</th>
        <th>PROGRAM_NAME</th>
        <th>PROGRAM_START_DATE</th>
      </tr>
    </thead>
    <tbody>
      <?php foreach ($outp as $row): ?>
```

hexadecimal number xxxx

'\uhhhh' matches a Unicode character with the hexadecimal value hhhh,

'\u{hhhh}' matches a Unicode character with the hexadecimal value hhhh

'\Uhhhhhhhh' matches a Unicode character with the hexadecimal value hhhhhhhh,

'\U{hhhhhhhh}' matches a Unicode character with the hexadecimal value hhhhhhhh,

'\cX' matches the control character X,

X, '\uX' matches the character X, '\UXX' matches the character XX

, '\x{X}' matches the character X, '\u{X}' matches the character

, '\U{XX}' matches the character XX, '\cX' matches the control character

, '\xX' matches the character X, '\uX' matches the character X,

'\UXX' matches the character XX, '\x{X}' matches the character X,

'\u{X}' matches the character X, '\U{XX}' matches the character XX

, '\cX' matches the control character X, '\xX'

```
  //function type variable
  var l = function () {
    console.log("hello world");
  };
typeof function () { };
// Returns function

typeof () => {} ; // Returns function//SyntaxError: Malformed arrow function parameter list
```

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

```
 function n2_$-aMe(parameter1, parameter2, parameter3) {
// code to be executed
}
 function toCelsius(fahrenheit) {
// fahrenheit -local variable
//fahrenheit -parameter which received the arguement passed
    return (5 / 9) * (fahrenheit – 32);
console.log("not execute after return statement");
  }
  var value = toCelsius(77);//toCelsius() refers to the function result from return and it is also calling and storing the return value in value variable
//77 is argument
  console.log(value);
  var value = toCelsius;//toCelsius refers to the function object
  console.log(value); //
```

abstract boolean byte char double final float goto int long native short synchronized throws transient volatile

Do not use these words as variables. ECMAScript 5/6 does not have full support in all browsers.

JavaScript Objects, Properties, and Methods

You should also avoid using the name of JavaScript built-in objects, properties, and methods:

Array,Boolean,Date,Error,Function,Number ,Object ,RegExp, **String** :constructor hasOwnProperty isPrototypeOf name String

Date : Infinity length Number toString Error,Function,Number,Object ,RegExp, String : arguments caller NaN undefined

eval : isFinite Math Object isNaN NaN Number undefined isInteger isSafeInteger parseFloat round toFixed toLocaleString toPrecision toExponential

function : isNaN NaN prototype valueOf call apply bind toString arguments caller NaN undefined length name prototype toString

Java Reserved Words

JavaScript is often used together with Java. You should avoid using some Java objects and properties as JavaScript identifiers:

getClass java JavaArray javaClass JavaObject JavaPackage

You should also avoid using the name of HTML and Window objects and

```
        <tr>
          <td><?php echo htmlspecialchars($row['STUDENT_REF_ID']); ?></td>
          <td><?php echo htmlspecialchars($row['PROGRAM_NAME']); ?></td>
          <td><?php echo htmlspecialchars($row['PROGRAM_START_DATE']); ?></td>
        </tr>
      <?php endforeach; ?>
    </tbody>
  </table>
</div>
</body>
</html>
```

Dynamic HTML Table

Make the HTML table based on the value of a drop down menu:

```
  Choose an option:
document.write('<br><h2>Make a table based on the value of a drop down menu.</h2><select id="myselect" onchange="change_myselect(this.value)"><option value="">Choose an option:</option><option value="orders">Orders</option><option value="parts">Parts</option><option value="supplier">Supplier</option></select><p id="demoDynamicHTMLTable">demoDynamicHTMLTable</p>');
    function change_myselect(sel) {
      const dbParam = JSON.stringify({ table: sel, limit: 20 });
      const xmlhttp = new XMLHttpRequest();
      xmlhttp.onload = function () {
        myObj = JSON.parse(this.responseText);
   text = "<table
```

```
matches the character X
```

The exec() method is a RegExp expression method.

```
The exec() method executes a search for a match in a specified string(""), or within a specified range of characters. Returns an object containing the matched text, or null if no match is found
var resultTest = /e/.exec("The best things in life are free!");// Output: ["e"] but seen only Output: e
```

The test() method performs a test for a match in a string. Returns true or false.

```
var resultTest = /Shanaya/.test("Bazaar Sh");//false //as it tests if regex=/Shanaya/ is present in Bazaar Sh //as it is not present hence results false.
```

The split() method splits a string into an array where each word is an array item.

javascript style guide

1. Use consistent naming conventions: Use camelCase for variable and function names, and PascalCase

undefined as it thinks toCelsius is some variable and its value is being asked but such vqriable is undefined

```
//here The object is object to test and Constructor is Constructor to test against.
  function CarInstanceofTry(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }
  const auto = new CarInstanceofTry('Honda', 'Accord', 1998);

console.log(auto instanceof CarInstanceofTry);
  // Expected output: true as new keyword shows that auto is an object and it is instance of CarInstanceofTry constructor doesn't have a [Symbol.hasInstance]() method hence false
console.log(auto instanceof Object);// Expected output: true but it is false actually
//  TypeError
  // Thrown if constructor( CarInstanceofTry) is not an object. If constructor doesn't have a [Symbol.hasInstance]() method, it must also be a function.
```

The constructor Property

The constructor property returns the function that

properties:

alert all anchor anchors area assign blur button checkbox clearInterval clearTimeout clientInformation close closed confirm constructor crypto decodeURI decodeURIComponent defaultStatus document element elements embed embeds encodeURI encodeURIComponent escape event fileUpload focus form forms frame innerHeight innerWidth layer layers link location mimeTypes navigate navigator frames frameRate hidden history image images offscreenBuffering open opener option outerHeight outerWidth packages pageXOffset pageYOffset parent parseFloat parseInt password pkcs11 plugin prompt propertyIsEnum radio reset screenX screenY scroll secure select self setInterval setTimeout status submit taint text textarea top unescape untaint window HTML Event Handlers

In addition you should ***avoid using the name of all HTML event handlers.***

onblur onclick onerror onfocus onkeydown onkeypress onkeyup onmouseover onload onmouseup onmousedown onsubmit

Key codes

Here are some codes of keyboard keys. They are normal virtual key codes from Win32API.

- 8 - Backspace
- 9 - Tab
- 12 - 5 in the numeric keypad when Num Lock is off

```
border='1'><thead>
<tr><th>pid</th><th>pname</th
><th>color</th><th>sid</th><t
h>quantity</th><th>sname</
th><th>city</th></tr></thead>
"
for (x in myObj) {
  text += "<tr><td
class='undefinethen'>" +
myObj[x].pid + "</td><td>" +
myObj[x].pname + "</td><td>"
+ myObj[x].color +
"</td><td>" + myObj[x].sid +
"</td><td>" +
myObj[x].quantity +
"</td><td>" + myObj[x].sname
+ "</td><td>" + myObj[x].city
+ "</td></tr>";}
 text += "</table>"
document.getElementById("demo
DynamicHTMLTable").innerHTML
= text; }
      xmlhttp.open("POST",
"json_demo_html_table.php",
true);
xmlhttp.setRequestHeader("Con
tent-type", "application/x-
www-form-urlencoded");
      xmlhttp.send("x=" +
dbParam);
    }
```

No change is made in ***json_demo_html_table file for dynamic dropdown options***

```
//To Make an HTML drop down
list with data received as
JSON:
let text = "<select>"
      for (let x in myObj) {
        text += "<option>" +
myObj[x].pid + "</option>";
      }
      text += "</select>"
```

it is used in onload after parse

Using .asp file

A . asp file or an active server page file is an ASP.NET typed webpage or web document that

for class names.

2. Use meaningful variable names: Use descriptive names for variables to make the code easier to understand

3. Use whitespace: Use whitespace to separate logical sections of code and make it easier to read

4. Use comments: Use comments to explain the purpose of code and make it easier to understand

5. Use functions: Use functions to encapsulate code and make it easier to reuse

6. Use objects: Use objects to encapsulate data and behavior

Always put spaces around operators (= + - * /), and after commas:

```
  var x = 5; // good
  var x=5; // bad
```

Always use 2 spaces for indentation of code blocks://Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently.

```
  if (x > 5) {
    console.log("x is
greater than 5");
  } // good
  if (x > 5) {
    console.log("x is
greater than 5");
  } // bad
```

Always end a simple statement with a semicolon. like declaration of variable,object

created an instance of the object. It returns the function that was used to create the object//as all complex datatype or primitive stored in javascript as an object as javascript is object oriented language hence stored in object form and constructor will work on almost all datatype or typeof results //, or null if the object was created with Object.create() or Object.assign() without a prototype object.

```
"Sudha".constructor //
function String()
{ [native code] }
(3.14).constructor//
function Number()
{ [native code] }
false.constructor
// function Boolean()
{ [native code] }
1234n.constructor
// function BigInt()
{ [native code] }
{}.constructor
// function Object()
{ [native code] }
[].constructor
// function Array()
{ [native code] }
new Date().constructor
// function Date()
{ [native code] }
new Set().constructor
// function Set()
{ [native code] }
new Map().constructor
// function Map()
{ [native code] }
function ()
{ }.constructor;
// function Function() {
[native code] }
var personConstructor =
{ name: 'Sudha', age: 20
};
{name:'Sudha',age:20}.co
nstructor; //error when
```

- 13 - Enter
- 16 - Shift
- 17 - Ctrl
- 18 - Alt
- 19 - Pause/Break
- 20 - Caps Lock
- 27 - Esc
- 32 - Space
- 33 - Page Up
- 34 - Page Down
- 35 - End
- 36 - Home
- 37 - Left arrow
- 38 - Up arrow
- 39 - Right arrow
- 40 - Down arrow
- 44 - Print Screen
- 45 - Insert
- 46 - Delete

- 48 - 0
- 49 - 1
- 50 - 2
- 51 - 3
- 52 - 4
- 53 - 5
- 54 - 6
- 55 - 7
- 56 - 8
- 57 - 9

- 65 - A
- 66 - B
- 67 - C
- 68 - D
- 69 - E
- 70 - F
- 71 - G
- 72 - H
- 73 - I
- 74 - J
- 75 - K
- 76 - L
- 77 - M
- 78 - N
- 79 - O
- 80 - P
- 81 - Q
- 82 - R
- 83 - S
- 84 - T
- 85 - U
- 86 - V
- 87 - W
- 88 - X
- 89 - Y
- 90 - Z

contains HTML codes, text, graphics, and XML.

JSONP

JSONP stands for JSON with Padding.JSONP is a method for sending JSON data without worrying about cross-domain issues.JSONP does not use the XMLHttpRequest object.

JSONP uses the <script> tag instead.

Requesting a file from another domain can cause problems, due to cross-domain policy.

Requesting an external script from another domain does not have this problem.

JSONP uses this advantage, and request files using the script tag instead of the XMLHttpRequest object.

```
<script>
document.write('<br><br><p id="demoJSONP">demoJSONP</p>');
function myFunc(myObj) {
document.getElementById("demoJSONP").innerHTML += "<br>myObj.name :" + myObj.name;//myObj.name :Sudha

}
</script>
```

NOTE:<script> tag inside <script> gives error

```
<script src="demo_jsonp.php">
document.write('<br><br>demo_jsonp.php is called');
</script>
```

// The script tag requests the file from the server, and the server returns the JSON data wrapped inside the function call. The browser executes the function, and the JSON data is available in the

```
  var x = 5; // good
  var x=5 // bad
```

General rules for complex (compound) statements:like function ,loop,conditionals

Put the opening bracket at the end of the first line.

Use one space before the opening bracket.

Put the closing bracket on a new line, without leading spaces.

Do not end a complex statement with a semicolon.

```
  example : for (let i = 0; i < 5; i++) {
     x += i; //ok
  }//; with }; bad
```

Always use semicolon after the last statement in a block of code.

like if,else,

```
  if (x > 5) {
    console.log("x is greater than 5");
  } // good
  if (x > 5) {
    console.log("x is greater than 5")
  } ;// bad
```

General rules for object definitions:

Place the opening bracket on the same line as the object name.

Use colon plus one space between each property and its value.

Use quotes around string values, not around numeric values.

Do not add a comma after the last property-value pair.

Place the closing

```
concatenated anything even gap, use in separate line for no error or use the variable name for same line
document.write({ name: 'Sudha', age: 20 }.constructor);// function Object() { [native code] }
document.write({ name: 'Sudha', age: 20 }.constructor === Object);//true
document.write(JSON.stringify(Object.prototype));//{}
document.write({ name: 'Sudha', age: 20 }.constructor === Object.prototype.constructor);//true
personConstructor.constructor//object
personConstructor.__proto__ //object
personConstructor.prototype //undefined as prototype is not property name , use __proto__
personConstructor.hasOwnProperty("name") + "<br>" + //true
personConstructor.hasOwnProperty("age") + "<br>" + //true
 personConstructor.hasOwnProperty("constructor") + "<br>" +//false//"constructor" is property of object not the respective datatype variable object
var myArrayConstructor = [4, 6, 23, 56];
document.write(myArrayConstructor.constructor === Array); //true
document.write(myArrayCo
```

- 91 - left Win
- 92 - right Win
- 93 - Popup

- 96 - 0 in the numeric keypad
- 97 - 1 in the numeric keypad
- 98 - 2 in the numeric keypad
- 99 - 3 in the numeric keypad
- 100 - 4 in the numeric keypad
- 101 - 5 in the numeric keypad
- 102 - 6 in the numeric keypad
- 103 - 7 in the numeric keypad
- 104 - 8 in the numeric keypad
- 105 - 9 in the numeric keypad
- 106 - * in the numeric keypad
- 107 - + in the numeric keypad
- 109 - - in the numeric keypad
- 110 - . in the numeric keypad
- 111 - / in the numeric keypad

- 112 - F1
- 113 - F2
- 114 - F3
- 115 - F4
- 116 - F5
- 117 - F6
- 118 - F7
- 119 - F8
- 120 - F9
- 121 - F10
- 122 - F11
- 123 - F12

- 144 - Num Lock
- 145 - Scroll Lock
- 160 - left Shift
- 161 - right Shift
- 162 - left Ctrl
- 163 - right Ctrl

```
function
```

demo_jsonp.php

```
<?php
$myJSON = '{ "name":"Sudha", "age":30, "city":"New York" }';
echo "myFunc(".$myJSON.");";
// The result returns a call to a function named "myFunc" with the JSON data as a parameter.The function named "myFunc" is must located on the client(html file), and ready to handle JSON data: //here JSON string of function is echo(print)
?>
```

To call the server file using JSONP by onclick of button

```
<script>
document.write('<br><button onclick="clickButton()">A script tag with a src attribute is created and placed in the document.</button><p id="demoDynamicScript">demoDynamicScript</p>');
function clickButton() {
//here script is created seperately for passing the php file, here in this script we are using the function to create script and call myFunc from here
 let s =document.createElement("script");//script tag is created for body element ... it would be above , not with the document.write
    s.src = "demo_jsonp.php";
document.body.appendChild(s);
    function myFunc(myObj) {
document.getElementById("demoDynamicScript").innerHTML = "<br>myObj.name" + myObj.name;
       // whatever passed in
```

bracket on a new line, without leading spaces.

Always end an object definition with a semicolon.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Short objects can be written compressed, on one line, using spaces only between properties, like this:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

For readability, avoid lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma.

```
document.getElementById("demo").innerHTML =
  "Hello Dolly.";//it is breaked after = operator
```

Variable and function names written as camelCase

Global variables written in UPPERCASE (We don't, but it's quite common)

Constants (like PI) written in UPPERCASE

HTML5 attributes can start with data- (data-quantity, data-price).

```
nstructor.constructor(3));//,, //these 2 commas shows that 3 inputs were given and since { _,_,_ } this way it forms object//here in constructor(3) ,3 shows the length of the object
.document.write(typeof myArrayConstructor.constructor(3));//object
 document.write(typeof myArrayConstructor.constructor(3).length);//number //here length will be 3
document.write(typeof myArrayConstructor.constructor(3, 4, 5, 6, 7, 8, 9));//object//it will not affect original array,it's just playing
document.write(new Date().constructor === Date);//true
```

JavaScript Function

normal function

```
  console.log("normal function")
  function myfunction()
{
    console.log("hello world");
    // alert("Hello World")
  };
  myfunction()//here function is called
```

function with parameter

```
  function hello(name) {
    console.log("hello " + name);
  };
  hello("shanaya");
```

function with return type

```
  function helloreturn(name) {
```

```
main php file will be passed here as parameter
    }
  }
```

Dynamic JSONP Database

```
sending JSON to the php file, and let the php file return a JSON object based on the information it gets.
document.write('<br><br><p id="jsonp_demo_db">jsonp_demo_db</p>');
var obj = { table: "program", limit: 10 };
var sD=document.createElement("script");
  sD.src = "jsonp_demo_db.php?x=" + JSON.stringify(obj);
document.body.appendChild(sD);
  function myFuncD(myObj) {
    let txt = "";
    for (let x in myObj) {
      txt += myObj[x].PROGRAM_NAME + "<br>";
    }
document.getElementById("jsonp_demo_db").innerHTML += "<br>txt :" + txt;
  }
```

jsonp_demo_db.php

```
<?php
header("Content-Type: application/json; charset=UTF-8");
$obj = json_decode($_GET["x"], false);
$conn = new mysqli("localhost", "root", "#sudha@2402", "shanayawork");
$result = $conn->query("SELECT PROGRAM_NAME FROM ".$obj->table." LIMIT ".$obj->limit);
$outp = array();//no need
```

CSS uses hyphens in property-names (font-size).

Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript names.

Many programmers prefer to use underscores (date_of_birth), especially in SQL databases.

Underscores are often used in PHP documentation.

`PascalCase`:PascalCase is often preferred by C programmers.

`camelCase:`camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries.

Do not start names with a $ sign. It will put you in conflict with many JavaScript library names.

Do not start names with a number. It will put you in conflict with many JavaScript library names

Do not use a mix of camelCase and underscore notation. It will put you in conflict with many JavaScript library names.

Loading JavaScript in HTML

Use simple syntax for loading external scripts (the type attribute is not necessary):

```
 <script
```

```
    return "hello " + name;
  };
console.log(helloreturn("shanaya"));
function add(a, b) {
    return a + b;
  }
  console.log(add(2, 3));
```

anonymous function

```
  (function () {
console.log("anonymous function");
    console.log("hello world");
  })();
```

function inside function inside function

```
  console.log("function inside function inside function")
  function myfunction3() {
    function myfunction4() {
      function myfunction5() {
console.log("hello world");
      }
      myfunction5();
    }
    myfunction4();
  }
  myfunction3();
```

Arrow function

```
var myFunctionArr = (a, b) => a * b;
myFunctionArr(5, 6); // Output: 30
//is equivalent to
  myFunctionArr();
  var myFunctionArr = function (a, b) {
    return a * b;
```

```
$outp = $result->fetch_all(MYSQLI_ASSOC);
echo
"myFuncD(".json_encode($outp).")";
?>
```

NOTE: noting extra should be here after php tag

Callback Function When you have no control over the server file

Sometimes the server file offers a callback function as a parameter:

```
document.write('<br><br><p id="jsonp_demo_dbCallback">jsonp_demo_dbCallback</p>');
var sCallback =document.createElement("script");
  s.src = "demo_jsonp2.php?callback=myDisplayFunction";
document.body.appendChild(s);
  function myDisplayFunction(myObj) {
document.getElementById("jsonp_demo_dbCallback").innerHTML = myObj + "<br> myObj.status :" + myObj.status+"<br> myObj.message :" + myObj.message+ "<br> myObj.data :" + myObj.data+ "<br> myObj.data.item1 :" + myObj.data.item1;
  }
```

demo_jsonp2.php

```
<?php
// Get the callback parameter from the URL
$callback = isset($_GET['callback']) ? $_GET['callback'] : 'myDisplayFunction';
// Create a sample response array
$responseData = array(
```

```
src="myscript.js">
</script>
```

but if it type module , then it must be specified

Accessing HTML Elements

```
//case sensitive string not tag name
const obj = getElementById("Demo")
  const obj = getElementById("demo")//different from above
```

Use Lower Case File Names

Most web servers (Apache, Unix) are case sensitive about file names: london.jpg cannot be accessed as London.jpg.

To avoid these problems, always use lower case file names (if possible).

Other web servers (Microsoft, IIS) are not case sensitive:london.jpg can be accessed as London.jpg or london.jpg.

Performance

Coding conventions are not used by computers. Most rules have little impact on the execution of programs.

Indentation and extra spaces are not significant in small scripts.

For code in development, readability should be preferred. Larger production

```
  }
hello = (val) => "Hello " + val;
hello("Sudha");
```

The handling of this is also different in arrow functions compared to regular functions.

In ***regular functions*** the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.

With ***arrow functions*** the this keyword always represents the object that defined the arrow function. not hat called the function

```
document.write("<br><p id='myHelloFunctionResult'>myHelloFunction Result : </p>");
helloRF = function () {
document.getElementById("myHelloFunctionResult").innerHTML += this;
  }
window.addEventListener("load", helloRF);//in hello this will refer to window// myHelloFunction Result : [object Window]
```

NOTE: all helloRF called onload

```
// A button object calls the regular function:
document.getElementById("myHelloFunction").addEventListener("click", helloRF);// [object HTMLButtonElement]
```

```
  'status' => 'success',
  'message' => 'Data retrieved successfully.',
  'data' => array(
    'item1' => 'value1',
    'item2' => 'value2'
  )
);
// Encode the array to JSON format
$jsonData = json_encode($responseData);
// Output the JSONP response
header('Content-Type: application/javascript');
echo $callback . '(' . $jsonData . ');';// as only 'myDisplayFunction' is in $callback
?>
```

Web Workers API

Web Workers API is based on the concept of worker threads, which are separate threads that run in parallel to the main thread.

Web Workers API provides a way to create and manage worker threads, as well as communicate with them through a message passing mechanism.

Web Workers API is useful for a variety of tasks, including:

1. Data processing: Web Workers API can be used to perform data processing tasks, such as data compression, encryption, or data transformation.
2. Scientific simulations: Web Workers API can be used to perform scientific simulations, such as weather
3. Machine learning: Web Workers API can be used to perform machine learning tasks, Graphics rendering tasks ,video processing tasks ,audio processing tasks ,video encoding and decoding ,cryptography tasks, such as encryption ,compression

scripts should be minimized.

Avoid global variables, avoid new, avoid ==, avoid eval()

Avoid global variables

Global variables are not good practice. They can be accessed from anywhere in the code, which can lead to unexpected behavior and bugs.

Instead, use local variables or encapsulate data in objects.

Global variables and functions can be overwritten by other scripts.

All variables used in a function should be declared as local variables.

Local variables must be declared with the var, the let, or the const keyword, otherwise they will become global variables.

It is a good coding practice to put all declarations at the top of each script or function.

This will:Give cleaner code

Provide a single place to look for local variables

Make it easier to avoid unwanted (implied) global variables

Reduce the possibility of unwanted re-declarations

Declare at the beginning

let firstName, lastName, price, discount, fullPrice;

```
var helloArrow = () => {
document.getElementById("myHelloArrowFunctionResult").innerHTML += this;
  }
  // The window object calls the function:
window.addEventListener("load", helloArrow);//[object Window]
  // A button object calls the function:
document.getElementById("myHelloArrowFunction").addEventListener("click", helloArrow);
  //myHelloArrowFunctionResult :[object Window] //the second example returns the window object because the window object is the "owner" of the function. not returns the element which called it
```

JavaScript Callbacks

Callbacks are functions passed as arguments to other functions, which are then executed after some operation has completed

Callbacks are used to handle asynchronous operations, such as reading files, making network requests, or executing functions that take a long time to complete and used to handle errors that occur during these operations.

JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined.

//lower call if same then it will overwrite the above call

```
document.write("<br><p
```

tasks ,network communication tasks, database operations , file I/O operations,web scraping tasks , web crawling tasks ,web automation tasks ,web testing tasks ,web development tasks , web deployment tasks ,web maintenance tasks , web security tasks , web optimization tasks, web analytics tasks , web monitoring tasks ,web debugging tasks , web profiling tasks, web performance optimization tasks, web accessibility tasks , web internationalization tasks, web localization tasks ,web content delivery tasks,web content caching tasks,

```
document.write(`<br><br><h2>JavaScript Web Workers API</h2>

<p id="resultWorkerOutput1"></p>
<button id='btnWorkHello'>Hello, worker!</button><br>`);
  var btnWorkHello = document.getElementById('btnWorkHello');

btnWorkHello.addEventListener('click', () => {
    //create a new web worker //in the event passed
    var workerO = new Worker('worker1.js');//Cannot GET /worker.js //it should be created worker.js new file
    //you must create worker.js file somewhere in this folder
    //post a message to the worker

workerO.postMessage('Hello, worker!');//in worker.js file from where you post message
    //listen for a message
```

```
  // Use later
  firstName = "Sudha";
  lastName = "Kumari";
  price = 19.90;
  discount = 0.10;
  fullPrice = price - discount;
```

This also goes for loop variables:

```
  for (let i = 0; i < 5; i++) {
    console.log(i);
    // i is a local variable, so it will not be overwritten by other scripts.
  }
```

Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

Give cleaner code

Provide a single place to initialize variables

Avoid undefined values

// Declare and initiate at the beginning

```
  let firstName = "";
  let lastName = "";
  let price = 0;
  let discount = 0;
  let fullPrice = 0,
  const myArray = [];
  const myObject = {};
```

Initializing variables provides an idea of the intended use (and intended data type).

Declaring objects with const will prevent any accidental change of type:

```
  let car =
```

```
id='demoMyDisplay'>demoMyDisplay </p>");
  function myDisplayer(some) {
document.getElementById("demoMyDisplay").innerHTML += "<br>" + some;
  }
  function myCalculator1(num) {
    let result = num * 9;
    myDisplayer("Hello Calculator1");// if it would not have been innerHTML += instead it would have been innerHTML =   , then lower will overwrite the above or call first
    myDisplayer(result);
  }
  function myCalculator2(num) {
    let result = num * 10;
    myDisplayer("Hello Calculator2");
    myDisplayer(result);
  }
  myCalculator2(5);
  myCalculator1(5);
```

Sequence Control

Sequence control is used to control the flow of a program. It determines the order in which the program's instructions are executed.

Sometimes you would like to have better control over when to execute a function.

Suppose you want to do a calculation, and then display the result.

You could call a calculator function (myCalculator), save the result, and then call another function (myDisplayer) to display the result:

```
from the worker object
    workerO.onmessage =
function (event) {//it should
be in worker.js // event or e
, it is used for handling the
event . Event Handler ,
function receives the event
(here message is received but
it that is the event
triggered)
      console.log('Received
message from worker in main
script:', event.data);//data
used not the value //value
only for input element

document.getElementById("resu
ltWorkerOutput1").innerHTML =
event.data;//Hello, worker!
    };//Received message from
worker: Hello, worker!
//when this.postMessage is
used in worker1.js
    //Received message from
worker in main script: when
postMessage  is used
    //
document.getElementById("resu
ltWorkerOutput1").innerHTML =
event.data;//undefined  //if
it is outside the
workerO.onmessage then event
will stand for this script
then put it inside
    //terminate the worker
    //
workerO.terminate(); //don't
use inside
  });
```

worker1.js

```
onmessage = function(event) {
  console.log('Received
message from worker in
worker1.js:',
event.data);//data used not
the value //value only for
input element////Received
message from worker in
worker.js : Hello, worker!
```

```
{type:"Fiat",
model:"500",
color:"white"};
  car = "Fiat";
// Changes object to
string
  const car =
{type:"Fiat",
model:"500",
color:"white"};
  car = "Fiat";
// Not possible
```

Declare Arrays with const

Declaring arrays with const will prevent any accidental change of type:

```
  let cars =
["Saab", "Volvo",
"BMW"];
  cars = 3;    //
Changes array to
number
  const cars =
["Saab", "Volvo",
"BMW"];
  cars = 3;    //
Not possible
```

Don't Use new Object()

```
  Use "" instead of
new String()
  Use 0 instead of
new Number()
  Use false instead
of new Boolean()
  Use {} instead of
new Object()
  Use [] instead of
new Array()
  Use /()/ instead
of new RegExp()
  Use function (){}
instead of new
Function()
  let x1 = "";
// new primitive
string
  let x2 = 0;
```

```
function
myDisplayer(some) {

document.getElementById(
"demoMyDisplay").innerHT
ML += "<br>" + some;
  }
  function
myCalculator(num1, num2)
{
    let result = num1 +
num2;
    //
myDisplayer(result);//Th
e problem is that you
cannot prevent the
calculator function from
displaying the
result. //now if we want
to control mydisplay
then this instruction
should be outside the
function to call it
seperately
     return result;
  }
  var Sumresult =
myCalculator(5, 10);
myDisplayer(Sumresult);/
/we have control ovewr
when to display function
should be called  //But
The problem  is that you
have to call two
functions to display the
result.
myCalculator(5,10) and
myDisplayer(Sumresult)
```

Now it is time to bring in a ***callback to solve above two problems*** mentioned that is of calling two function and calling inside has no control

BEST: solve both problems

```
function
myDisplayerCall(some) {
document.getElementById(
"demoMyDisplay").innerHT
ML += "<br>" + some;
```

```
 //
this.postMessage(event.data);
  postMessage(event.data);
//undefined for
document.write event.data
 //  postMessage(data);//data
is not defined
 // var data=event.data;
// postMessage(data);
  };
  //terminate the worker
  // worker.terminate();
```

example 2

__

```
document.write(`<br><br><h2>JavaScript Web Workers
API</h2><p>Count numbers:
<output
id="resultWorker"></output></p><button
onclick="startWorker()">Start
Worker</button> <button
onclick="stopWorker()">Stop
Worker</button><br>`);
  var wWorker;
  function startWorker() {
    if (typeof (wWorker) ==
"undefined") {
      wWorker = new
Worker("demo_workers.js");//creates new worker to write
javascript and run in
background
      //no need to post any
message or event in the
demo_workers.js
      //on click this
function will be called and
it autometically trigger
demo_workers.js
      //demo_workers.js file
should be in the same folder
as this html file
    }
    wWorker.onmessage =
function (event) {//receives
the postmessage of
demo_workers.js as it is the
the same function as what
```

```
// new primitive
number
  let x3 = false;
// new primitive
boolean
  const x4 = {};
// new object
  const x5 = [];
// new array object
  const x6 = /()/;
// new regexp object
  const x7 =
function(){}; // new
function object
```

Beware of Automatic Type Conversions

JavaScript is loosely typed.

A variable can contain all data types.

A variable can change its data type:

```
  let x = "Hello";
// typeof x is a
string
  x = 5;
// changes typeof x
to a number
```

Beware that numbers can accidentally be converted to strings or NaN (Not a Number).

When doing mathematical operations, JavaScript can convert numbers to strings:

```
   let x = 5 + 7;
// x.valueOf() is
12,  typeof x is a
number
  let x = 5 + "7";
// x.valueOf() is
57,  typeof x is a
string
  let x = "5" + 7;
// x.valueOf() is
57,  typeof x is a
string
  let x = 5 - 7;
```

```
  }
  function
myCalculatorCall(num1,
num2, myCallback) {
    let result = num1 +
num2;
myCallback(result);//As
myCallback =
myDisplayerCall
    //
myCallback(result)=myDisplayerCall(result);//if
callback is passed then
this will execute
otherwise not execute
the display
  }
  myCalculatorCall(5,
10, myDisplayerCall);
//only one call
sufficient to solve
above two problems
```

//here myCallback = myDisplayerCall ,it means myDisplayerCall() function will be called only when it is passed as arguement ,if we do't need the display then we can pass any other function as argument and it will not be called and it will not display anything

simply pass third arguement as undefined. or dropping third arguement gives **error**

```
 myCalculatorCall(5,10)//gives error
myCalculatorCall(5,10,)//gives error
if you don't to display
the result but only only
calculate then use
 myCalculatorCall(5,10,undefined);//undefined is
a callback function that
does nothing
You can also pass a
different function as
the callback, to display
the result in a
different way
  myCalculatorCall(5,
initiated the demo_workers.js
file  //demo_workers.js is
passed a message and that
message calls the function
taking event as parameter
      //
document.getElementById("resu
ltWorker").innerHTML +=
event.data;//it is in main
thread to print the
output//123456789 ..... due
to +=

document.getElementById("resu
ltWorker").innerHTML =
event.data;//count every 1
sec and gets updated
    };
  }
  function stopWorker() {
    wWorker.terminate();//new
demo_workers.js is stopped to
work in background
    wWorker =
undefined;//clear the
demo_workers.js space used
  }
```

demo_workers.js

```
//counting numbers using
timer
var count = 0;
setInterval(function(){
 count++;
 //
document.getElementById("resu
ltWorker").innerHTML =
count;//it will be not
defined here
 postMessage(count);//
works //it will be send as
event to the main script
//count is event.data
//setinterval will repaint
every 1 sec  //count value is
passed and the passed value
is called with event.data
//data=count   //it will work
fully
 }, 1000);
 //
```

```
// x.valueOf() is
-2,  typeof x is a
number
  let x = 5 - "7";
// x.valueOf() is
-2,  typeof x is a
number
  let x = "5" - 7;
// x.valueOf() is
-2,  typeof x is a
number
  let x = 5 - "x";
// x.valueOf() is
NaN, typeof x is a
number
```

Subtracting a string from a string, does not generate an error but returns `NaN` (`Not a Number`):

```
  "Hello" - "Dolly"
// returns NaN
```

`Use === Comparison`

`The == comparison operator always converts (to matching types) before comparison.`

`The === operator forces comparison of values and type:`

`Use Parameter Defaults`

If a function is called with a missing argument, the value of the missing argument is set to undefined.

Undefined values can break your code. It is a good habit to assign default values to arguments.

```
  function
myFunction(x, y) {
   if (y ===
undefined) {
     y = 0;
   }
```

```
10, hello);//any other
function when passed
then it will not call
the display function
```

NOTE: When you pass a function as an argument, remember not to use parenthesis.

Right: `myCalculator(5, 5, myDisplayer);`

Wrong: `myCalculator(5, 5, myDisplayer());`

Direct callback

```
  const myNumbers = [4,
1, -20, -7, 5, 9, -6];
 // Call removeNeg with
a callback
  const posNumbers =
removeNeg(myNumbers, (x)
=> x >= 0);//removeNeg
function is defined
below //here callback
function is directly
used in removeNeg
function call in form of
arrow function and one
line is a kind of return
true or false.
  // Display Result
document.getElementById(
"demoMyDisplay").innerHT
ML += "<br>" +
posNumbers;
  // Keep only positive
numbers
  function
removeNeg(numbers,
callback) {//here
callback function is (x)
=> x >= 0 //here
callback itself will
become the callback
function neme , now
parameter will be passed
in callback as
callback(x)
    const myArray = [];
    for (const x of
numbers) {
      if (callback(x)) {
```

```
postMessage(count);//count will be 0 always

 //or
 let i = 0;
function timedCount() {
 i ++;

postMessage(i);//postMessage() method - which is used to post a message back to the HTML page.

setTimeout("timedCount()",500);//Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.
}
timedCount();//called the function
```

Example 3

```
var btn1 = document.getElementById('btn1');
 btn1.addEventListener('click', () => {
    //on event click , event passed in button
    var workerObj = new Worker("worker.js");//take parameter of .js file which will receive the message with the help of workerObj  object , message is posted there in worker.js file
  workerObj.postMessage("Start worker");//it means iss object me message post karo //it also sends event click , event passed in button
    //  workerObj.onmessage = function(event) {} //onmessage used in worker.js file , means it will will receive message which is posted by this current script over there.
    //
```

```
  }
  default parameters in the function definition:
  function (a=1, b=1) { /*function code*/ }
```

Always end your switch statements with a default. Even if you think there is no need for it

Switch

```
var days = "";
    switch (new Date().getDay()) {
         case 0:
              days = "Sunday";
              break;
         case 1:
              days = "Monday";
              break;
         case 2:
              days = "Tuesday";
              break;
         case 3:
              days = "Wednesday";
              break;
         case 4:
              days = "Thursday";
              break;
         case 5:
              days = "Friday";
              break;
         case 6:
              days = "Saturday";
              break;
         default:
              days = "Unknown";
```

Avoid Number, String, and Boolean as Objects

Always treat numbers, strings, or booleans as primitive values. Not as objects.

Declaring these types as objects, slows down execution speed, and

```
//here filtering of positive numbers will take place.
          myArray.push(x);
        }
      }
      return myArray;
//1,5,9
   }
```

Asynchronous JavaScript

Functions running in parallel with other functions are called asynchronous

A good example is JavaScript `setTimeout()`

Asynchronous JavaScript is a type of JavaScript that runs in the background, without blocking the main execution

Asynchronous JavaScript is used to perform tasks that take a long time, like loading a file or retrieving data from a server.

`setTimeout()` is a function that executes a function after a specified delay and setTimeout() is asynchronous, meaning it runs in the background, without blocking the main execution.

```
setTimeout(myFunctionsetTimeout, 3000);//here myFunctionsetTimeout is used as a callback.myFunctionsetTimeout is passed to setTimeout() as an argument.
```

Advice : Instead of passing the name of a function as an argument to another function, you can always pass a whole function instead.

```
setTimeout(function () {
 document.getElementById("demoMyDisplay").innerHTML = "I love You !!";
  }, 3000); //here
```

```
document.getElementById("resultWorkerOutput").innerHTML = result;//it gives result = 0 //as par the onmessage global result variable
    workerObj.onmessage = function (e) { //e is event it takes from worker thread document.getElementById("resultWorkerOutput").innerHTML = e.data; } // it works and give proper result // workerObj is the object in the main script which is used to post and recive the message
  });
  var btn2 = document.getElementById('btn2');
 btn2.addEventListener('click', () => {//on event click , event passed in button
    document
.getElementById("resultWorkerH1").innerHTML += "Hi" + " ";//its not affecting the execution flow hence no need to put his hi in worker.js
  });
```

worker.js

```
onmessage = function(e) {//receieve the event// event click , event also passed here //here event is postmessage of .js file and object workerObj //workerObj.postMessage("Start worker");//data used not the value //value only for input element
  console.log('Received message', e.data);//it means event click par jo message post kiya uska data //e.data is Start worker which is passed in the Worker constructor//workerObj.postMessage("Start worker");
  // Do something with the
```

produces nasty side effects:

```
  let x = "John";
  let y = new String("John");
  (x === y) // is false because x is a string and y is an object.
  let x = new String("John");
  let y = new String("John");
  (x == y) // is false because you cannot compare objects.
```

`Avoid Using eval()`

The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.

Confusing Addition & Concatenation

```
  let x = 10;
  x = 10 + 5;
// Now x is 15

  let y = 10;
  y += "5";
// Now y is "105"
```

Misunderstanding Floats

All numbers in JavaScript are stored as 64-bits Floating point numbers (Floats).

All programming languages, including JavaScript, have difficulties with precise floating point values:

```
  let x = 0.1;
  let y = 0.2;
  let z = x + y
// the result in z will not be 0
```

```
function is passed as a callback to setTimeout() hence no function name is needed to be called
```

setInterval() is also asynchronous

```
setInterval(myFunctionsetInterval, 1000);
 function myFunctionsetInterval() {
document.getElementById("demoMyDisplay").innerHTML = "I love You !!";
  }
```

Callback Alternatives

callbacks are often replaced with promises or async/await syntax.

With asynchronous programming, JavaScript programs can start long-running tasks, and continue running other tasks in parallel. But, asynchronus programmes are difficult to write and difficult to debug. Because of this, most modern asynchronous JavaScript methods don't use callbacks. Instead, in JavaScript, asynchronous programming is solved using ***Promises*** instead.

JavaScript Promises

`"I Promise a Result!"`

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is an Object that links Producing code and Consuming code

A Promise contains both the producing code and calls to the consuming code

Promise Syntax

```
  let myPromise = new Promise(function(myResolve, myReject) {
  // "Producing Code"
```

```
received data
  // For example, update the
UI or send a response back
  // postMessage(e.data);
//undefined for
document.write event.data
  //
this.postMessage(e.data);//de
fined
  // postMessage(data);//data
is not defined
  var result = 0;
  for (let i = 0; i <
100000000; i++) {//it's very
timeconsuming
    result += i;
    }
    console.log(result);
    postMessage(result);//it
post the message to the
script(.js file) which
initialised the
workerconstructor .js
file //it must be received in
.js file workerObj with
onMessage //
workerObj.onmessage =
function(e)
    //workerObj object of the
script file will receive the
message
    //NOTE: if result is used
directly then it will receive
result=0  . hence onmessage
is used to receive the event
(postmessage) then for loop
result will send there
}
//console.log(btn1);
//error on click event as
what is passed to this
worker.js because btn1 object
is parent object and it is
not defined for this
worker.js  //it is seperate
thread error hence it will
keep executing and rest will
not execute as it raise error
//console.log(window);//
window not defined
//console.log(document);//
```

To solve the problem above, it helps to multiply and divide:

```
  let z = (x * 10 +
y * 10) / 10;
// z will be
0.3//each floating
no. multiplied with
its precise value ,
here it is 10
```

JavaScript will allow you to break a statement into two lines

```
var xBreak = "Hello
World!";//invalid
```

You must use a "backslash" if you must break a statement in a string:

```
var xBreak = "Hello\

World!"; //allowed
```

Misplacing Semicolon

Because of a misplaced semicolon, this code block will execute regardless of the value of x:

```
var xMistake = "";

  if (xMistake ==
19);//it will be
treated as some
alone part
  {//it will be like
a block
    // code block
  }
```

It is a default JavaScript behavior to close a statement automatically at the end of a line.

So, the semicolon is not needed here. But, if you want to make it clear that the if statement is not a

```
(May take some time)
   myResolve(); // when
successful //same name
only you can add value
in myResolve("value")
   myReject();  // when
error  //same name only
you can add value in
myReject("value")
```

LIKE

```
var x = 0;
if (x == 0) {
myResolve("OK");//value=
"OK" //this is value in
success case
    } else {
      myReject("Error");
    }

  });
  // "Consuming Code"
(Must wait for a
fulfilled Promise)
  //Here is how to use
the Promise:
  myPromise.then(
   function(value) {
//code if successful//
myDisplayerPromise(value
);
} //callback function
    ,
   function(error) { //
code if some error
myDisplayerPromise(error
);
 }//callback function
  );
```

Promise.then() takes two arguments, a callback for success and another for failure and Both are optional

```
function
myDisplayerPromise(some)
{
document.getElementById(
"demoMyPromise").innerHT
ML += some + "<br>";
  }
```

The Promise object supports

```
document not defined
console.log(self);//it is defined and it is the parent object of worker.js file and it heas its own properties and methods
//DedicatedWorkerGlobalScope {name: '', onmessageerror: null, onmessage: ƒ, cancelAnimationFrame: ƒ, close: ƒ, …}
```

Before creating a web worker, check whether the user's browser supports it:

```
  if (typeof (Worker) !== "undefined") {
    // Yes! Web worker support!
    // Some code.....
  } else {
    // Sorry! No Web Worker support..
  }
```

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
  if (typeof (w) == "undefined") {
    w = new Worker("demo_workers.js");
  }
```

Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the terminate() method:

w.terminate();

Reuse the Web Worker

If you set the worker variable to undefined, after it has been

standalone statement, you can use a semicolon like this:

```
function myFunction(a) {
    let power = 10;
    return a *
power;
  }//both above are same
function myFunction(a) {
    let power = 10
    return a * power
  }
```

JavaScript will also allow you to break a statement into two lines.

```
let
    power = 10;
//statement is breaked but string can't be without\
 //Note return if breaked give undefined
 return a * power;//OK
function myFunction(a) {
    let
      power = 10;
return//undefined Because JavaScript thought you meant: return;a * power;
    a * power;//not even get executed
  }
```

If a statement is incomplete like:let

JavaScript will try to complete the statement by reading the next line:

power = 10;

But since this statement is complete:

return

JavaScript will automatically close it like this:

two properties: state and result.

The **state** can be one of the following:

Pending(working), the result is undefined.

Fulfilled:the result is a value.

Rejected:the result is an error object.

The **result** can be one of the following:

A value (if the Promise is fulfilled)

An error (if the Promise is rejected)

Example Using Promise and Waiting for a Timeout

```
  var myPromise = new Promise(function (myResolve, myReject) {
    setTimeout(function () {
      myResolve("I love You !!");
      //it is only success case
    }, 3000);//print after 3 sec
  });

 myPromise.then(function (value) {//succes case only , for error case it should use error in place of value
document.getElementById("demoMyPromise").innerHTML += value + "<br>";
//I love You !!!
  });
```

The following example demonstrates how to use a ***promise to wait for a file to load before displaying its content***. The file is loaded using the XMLHttpRequest object.

```
  var myPromise = new
```

terminated, you can reuse the code:

w = undefined;

disadvantages of Web Workers and the DOM

Since web workers are in external files, they do not have access to the following JavaScript objects:

The window object

The document object

The parent object

Advantages of Web Workers

helps in complex computing.

does not block the UI

helps in multithreading

helps in parallel processing

Optimize performance of our program

JavaScript Fetch API

The Fetch API interface allows web browser to make HTTP requests to web servers.

😜 No need for XMLHttpRequest anymore.

The Fetch API provides a JavaScript interface for making HTTP requests and processing the responses.

Fetch is promise-based and is integrated with features of the modern web such as service workers and Cross-Origin Resource Sharing (CORS).

```
Fetch is based on async and await
```

.txt file fetch

```
let file = "learning.txt"
fetch(file)
  .then(x => x.text())
  .then(y => document.getElementById("demo").innerHTML = y);//[object Response]
```

Example

```
document.write(`<p id="resultgetText"></p>
<button id='btngetText' onclick=getText("learning.txt
```

return;

```
Caution : Never break a return statement.
```

JavaScript does not support arrays with named indexes.In JavaScript, ***arrays use numbered indexes***:

Accessing Arrays with Named Indexes

```
 Arrays with named indexes are called associative arrays (or hashes).//Note
const person = [];
  person[0] = "John";
  person[1] = "Doe";
  person[2] = 46;
  person.length;
// person.length will return 3
  person[0];
// person[0] will return "John"
```

In JavaScript, objects use named indexes.

NOTE:

If you use a named index, when accessing an array, JavaScript will redefine the array to a standard object.

After the automatic redefinition, array methods and properties will produce undefined or incorrect results:

```
var personA = [];
personA["firstName"] = "John";
  personA["lastName"] = "Doe";
  personA["age"] = 46;
  personA.length;
// person.length will return 0
  personA[0];
```

```
Promise(function (myResolve, myReject) {
//if promised not used then function getFile(myCallback){//myDisplayerCall is myCallback function which is passed in the function call and all below used XMLHttpRequest()
    var xhr = new XMLHttpRequest();
    xhr.onload = function () {
       if (xhr.status >= 200 && xhr.status < 300) {
      myResolve(xhr.responseText);
       } else {
      myReject(xhr.statusText);
       }
    };
    xhr.onerror = function () {  //onerror event is also covered here
      myReject(xhr.statusText);
    };
    xhr.open('GET', 'index2.html', true //true shows my resolve is true and it loads whole page //it will not throw any error
    );
    xhr.send();//
  });

myPromise.then(function (value) {
document.getElementById("demoMyPromise").innerHTML += value + "<br>";
  }).catch(function
") >Load file</button>`);
  async function
getText(file) {
    let myObject = await
fetch(file);
    let myText = await
myObject.text();
 document.getElementById("res
ultgetText").innerHTML =
myText;
    //   myDisplay(myText);
  }
```

learning.txt

Sudha

fetching .json file using .then

```
fetch('data.json')
    .then(function (response)
{
      return response.json();
    })
    .then(function (data) {
      appendData(data);
    })
    .catch(function (err) {
      console.log('error: ' +
err);
    });
  function appendData(data) {
    let mainContainer =
document.getElementById("myDa
ta");
    for (let i = 0; i <
data.length; i++) {
      let div =
document.createElement("div")
;
      div.innerHTML = 'Name:
' + data[i].firstName + ' ' +
data[i].lastName;

mainContainer.appendChild(div
);
    }
  }
```

body tag content

```
 <div id="myData"></div>
```

data.json (Array type)

```
[
  {
   "id": "1",
```

```
// person[0] will
return undefined
```

Trailing commas in object and array definition are legal in ECMAScript 5.

```
Object Example:
  person =
{firstName:"John",
lastName:"Doe",
age:46,}
  Array Example:
  points = [40, 100,
1, 5, 25, 10,];
WARNING !!

JSON does not allow
trailing commas.
```

Undefined is Not Null

JavaScript objects, variables, properties, and methods can be undefined.

In addition, empty JavaScript objects can have the value null.

This can make it a little bit difficult to test if an object is empty.

You can test if an object exists by testing if the type is undefined:

```
  if (typeof myObj
=== "undefined")
```

But you cannot test if an object is null, because this will throw an error if the object is undefined:

```
  Incorrect:
  if (myObj ===
null)
```

To solve this problem, you must test if an object is not null, and not undefined.

But this can still throw an error:

Incorrect:

```
  if (myObj !== null
```

```
(error) {
document.getElementById(
"demoMyPromise").innerHT
ML += "Error: " + error
+ "<br>"; //it print
error after searching
for file and is not
found
  });

//or the promise can be
handled by below method
also
  myPromise.then(
    function (value) {
document.getElementById(
"demoMyPromise").innerHT
ML += value +
"<br>"; },//comma is
must between two
functions in then , it
shows that one is for
myresolve callback and
other is for myreject
callback
    function (error) {

document.getElementById(
"demoMyPromise").innerHT
ML += "Error: " + error
+ "<br>";
    }//ERROR is catched
here from myReject
  );
```

The Promise object supports the following methods:

`then()`: Returns a new Promise object that is resolved or rejected based on the result of the original Promise object.

`catch()`: Returns a new Promise object that is resolved or rejected based on the result of the original Promise object. //already used above

`finally()`: Returns a new Promise object that is

```
    "firstName": "John",
    "lastName": "Doe"
  },
  {
    "id": "2",
    "firstName": "Mary",
    "lastName": "Peterson"
  },
  {
    "id": "3",
    "firstName": "George",
    "lastName": "Hansen"
  }
 ]
```

fetching .json file with fetch API and function without .then

```
fetchJSON("tryingJSON.json");
  async function fetchJSON(request) {
    try {
      const response5 = await fetch(request);
      // Checking headers
      const contentType = response5.headers.get("content-type");
      if (!contentType || !contentType.includes("application/json")) {
        throw new TypeError("Oops, we haven't got JSON!");
      }
      if (!response5.ok) {
        throw new Error(`Response status: ${response5.status}`);
      }
      // Otherwise, we can read the body as JSON
      const data = await response5.json();//object//fetch the response body content as JSON by calling the json() method of Response
console.log(data);//object dropdown
document.write(data);//[object Object]
document.write(JSON.stringify
```

```
&& typeof myObj !== "undefined")
```

Because of this, you must test for not undefined before you can test for not null:

Correct:

```
  if (typeof myObj !== "undefined" && myObj !== null)
//order of and matters
```

resolved or rejected based on the result of the original Promise object.

JavaScript Async

async makes a function return a Promise

await makes a function wait for a Promise

```
async function myAsyncFunction1() {
    return "Hello in myAsyncFunction1 ";
  }
myAsyncFunction1().then(function (result) {

document.getElementById("demoAsyncAwait").innerHTML += "result myAsyncFunction1(): " + result + "<br>";//result: Hello

document.getElementById("demoAsyncAwait").innerHTML += "Error: " + error + "<br>";//Reference error , error not defined//as it takes only resolve hence error is not defined

document.getElementById("demoAsyncAwait").innerHTML += "Error: " + result + "<br>";
  });

  //is same as below
  function myAsyncFunction2() {
    return Promise.resolve("Hello in myAsyncFunction2");
  }
 myAsyncFunction2().then(function (result) {
document.getElementById("demoAsyncAwait").innerHTML += "result
(data));//OUTPUT
      // Object
      // products: Array(2)0:
{id: 1, name: 'Product 1',
description: 'Description for
product 1', price: 29.99,
category: 'Electronics', …}1:
{id: 2, name: 'Product 2',
description: 'Description for
product 2', price: 49.99,
category: 'Home
Appliances', …}length: 2
    } catch (error) {
      console.error("Error:",
error);
console.error(error.message);
    }
  }
```

tryingJSON.json

```
{
  "products": [
   {
    "id": 1,
    "name": "Product 1",
    "description":
"Description for product 1",
    "price": 29.99,
    "category":
"Electronics",
    "stock": 100,
    "image_url": "title
logo.png"
   },
   {
    "id": 2,
    "name": "Product 2",
    "description":
"Description for product 2",
    "price": 49.99,
    "category": "Home
Appliances",
    "stock": 50,
    "image_url": "title
logo.png"
   }
  ]
}
```

Request Object / making clone

```
requestTry();
```

```
myAsyncFunction2(): " +
result + "<br>";//result
myAsyncFunction2():
Hello in
myAsyncFunction2
  });
```

Await Syntax

The await keyword can only be used inside an async function.

The await keyword makes the function pause the execution and wait for a resolved promise before it continues:

```
async function
myAsyncFunction3() {
    /*
      return new
Promise((resolve,
reject) => {
      // do something...
like an API call
      // if the API call
is successful, resolve
the promise with data
      resolve("Hello");
      });
      */
    var myPromise = new
Promise(function
(resolve) {
      resolve("Hello in
myAsyncFunction3()");
    });
    var result = await
myPromise;

document.getElementById(
"demoAsyncAwait").innerH
TML += "result
myAsyncFunction3(): " +
result + "<br >";
    //or

document.getElementById(
"demoAsyncAwait").innerH
TML += "result
myAsyncFunction3(): " +
await myPromise + "<br
```

```
  async function requestTry()
{
    const request7 = new 
Request("tryingJSON.json", {
      method: "POST",
      body: JSON.stringify({ 
username: "Sudha" }),
    });
    const request8 = 
request7.clone();
    const response7 = await 
fetch(request7);
console.log(response7.status)
;//200
    //  await is only valid 
in async functions and the 
top level bodies of modules
    const response8 = await 
fetch(request8);
console.log(response8.status)
;//200
  }
```

Using ajax and jquery from googleapis

```
$(function () {
    // var people = [];//no 
need
    $.getJSON('people.json', 
function (data) {
      $.each(data.person, 
function (i, f) {
        var tblRow = "<tr>" 
+ "<td>" + f.firstName + 
"</td>" + "<td>" + f.lastName 
+ "</td>" + "<td>" + f.job + 
"</td>" + "<td>" + f.roll + 
"</td>" + "</tr>"
        $
(tblRow).appendTo("#userdata 
tbody");
      });
    });
  });
```

head Script

```
 <script 
type="text/javascript" 
src="http://ajax.googleapis.c
om/ajax/libs/jquery/1.6.2/jqu
ery.min.js"> </script>
```

```
>";

  }

myAsyncFunction3();//res
ult myAsyncFunction3(): 
Hello in 
myAsyncFunction3()
  //result 
myAsyncFunction3(): 
Hello in 
myAsyncFunction3()
```

Waiting for a Timeout

```
  //The await keyword 
can also be used with 
the setTimeout function, 
which returns a promise:
  async function 
myAsyncFunction4() {
    // The setTimeout 
function returns a 
promise that resolves 
after a specified time
    var myPromise = new 
Promise(function 
(resolve) {

setTimeout(function () {
        resolve("Hello 
in myAsyncFunction4()");
      }, 10000);//
    });

document.getElementById(
"demoAsyncAwait").innerH
TML += "result 
myAsyncFunction4(): " + 
await myPromise + 
"<br>";//result 
myAsyncFunction4(): 
Hello in 
myAsyncFunction4()
  }

//myAsyncFunction4();//i
t will last for more 
time because of 
setTimeout , it wants to 
be executed after 10 sec 
but the below functions 
has taken it place hence
```

```
  <!-- This src is necessary
to use $ predefined on $
(function () {}  -->
```

body content

```
<div class="wrapper">
    <div class="profile">
      <table id="userdata"
border="2">
        <thead>
          <th>First
Name</th>
          <th>Last Name</th>
          <th>Email
Address</th>
          <th>City</th>
        </thead>
        <tbody>

        </tbody>
      </table>
    </div>
  </div>
```

people.json

```
{
  "person": [
    {
      "firstName": "Clark",
      "lastName": "Kent",
      "job": "Reporter",
      "roll": 20
    },
    {
      "firstName": "Bruce",
      "lastName": "Wayne",
      "job": "Player",
      "roll": 30
    },
    {
      "firstName": "Peter",
      "lastName": "Parker",
      "job": "Photographer",
      "roll": 40
    }
  ]
 }
```

More examples

```
// ajax/test.json is folder
directory , means ajax is
```

```
they will be removed and
it will be executed only
removing all aboves and
below in demoAsyncAwait
id
```

Waiting for a File

```
async function
myAsyncFunction5() {

    // Create a new file
reader
    //or use
XMLHttpRequest() //for
link or html files
    var fileReader = new
FileReader();
    // Read the file
fileReader.readAsText("l
earning.txt");
    var tryRead =
fileReader.readAsText("l
earning.txt");//for txt
file ,work good
    // When the file is
read, the result will be
available in the result
property
    fileReader.onload =
function () {
document.getElementById(
"demoAsyncAwait").innerH
TML += "result
myAsyncFunction5(): " +
fileReader +
fileReader.result +
tryRead + "<br >";
    };
  };
myAsyncFunction5();
```

waiting for html file

```
async function
myAsyncFunction6() {

    var myPromise = new
Promise(function
(resolve) {
      // Create a new
XMLHttpRequest object
```

```
folder name and test.json is
file name in it. To create
this directory just create
new file and paste
"ajax/test.json" directly and
it will create folder with
file
  $.getJSON("ajax/test.json",
function (data) {
    var items = [];
    $.each(data, function
(key, val) {
      items.push("<li id='" +
key + "'>" + val + "</li>");
    });
    $("<ul/>", {
      "class": "my-new-list",
      html: items.join("")
    }).appendTo("body");
  });
```

ajax/test.json

```
{
  "one": "Singular
sensation",
  "two": "Beady little eyes",
  "three": "Little birds
pitch by my doorstep"
}
```

jQuery

to load the content of `mypage.html` into the `div` with id `targetdiv`.

```
$.get('mypage.html',
function(data) {
  $('#targetdiv').html(data);
});
```

```
      var xhr = new
XMLHttpRequest();
      // Open the
request
      xhr.open('GET',
"Hair Spa Cap Steamer
For Women Beauty
Products.html");
      // Send the
request
      xhr.onload =
function () {
        if (xhr.status
== 200) {//the 200 OK
response is preferred to
this status.

resolve(xhr.response);

        } else {
          resolve("File
not Found");
        }
      };

xhr.send();//resolve
output value is send
    });

document.getElementById(
"demoAsyncAwait").innerH
TML += "result
myAsyncFunction6(): " +
await myPromise +
"<br>";//as soon it
exceed's it becomes
false and escaped
//await make it to
replace instead of
append

  }
  myAsyncFunction6();
//if it gets executed
and above function will
not be executed but if
it is not executed then
above will execute
```

JavaScript Scope

Scope determines the accessibility of variables, objects, and functions from different parts of the code. JavaScript has two types of scope: ***global and local***.

Block scope :kind of local but in javascript it is global scope. JavaScript does not have a concept of block scope like some other languages (e.g., C , Java, C#).But In javascript JavaScript keywords: ***let and const*** are two keywords which provide Block Scope in JavaScript,{}.JavaScript has function scope, not block scope. This means that variables declared inside a block statement are in the same scope as the variables declared outside the block statement.

Variables declared with the ***var keyword can NOT have block scope***.Variables declared inside a { } block can be accessed from outside the block.

Variables declared with the ***let and const keywords have block scope.***Variables declared inside a { }

Function scope:kind of local with var , let , const .Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.Local variables are created when a function starts, and deleted when the function is completed. Local variables are not shared between functions.In a web browser, global variables are deleted when you close the browser

window (or tab).
Function arguments (***parameters***) work as local variables inside functions.They are deleted when the function is completed.

```
var xGO = 10; // global
variable
var NameG;
  function
myFunctionScope() {
    var xINF = 20; //
local variable as in
function even it is var
 NameG = "Sudha
Kumari";//declared
outside the function
NameA = "Sudha Kumari";
// automatically global
variable as not declared
with var ,let, const in
function but function
must be called before
using it outside
otherwise it shows error

document.writeln("<br>Na
meG Inside function :" +
NameG); // NameG Inside
function :Sudha Kumari
    if (true) {
      var y = 30; //
local variable
document.writeln("<br>y
Inside function :" + y);
// outputs 30
document.writeln("<br>xG
O Inside function :" +
xGO); // outputs 10
    }
  }
document.writeln("<br>xG
O Outside function :" +
xGO); 10 //as global
variable
 document.writeln("<br>y
Outside
function :"+y);//y is
not defined here //Error
```

```
document.writeln("<br>NameG Outside function :"+NameG); //it would have been undefined if it is defined outside and no value assign to it like var NameG;//it is undefined if myFunctionAutoGlobal() is called below as js executes top to bottom hence show undefined
  myFunctionScope();
document.writeln("<br>NameG Outside function and below function call :" + NameG);// NameG Outside function and below function call :Sudha Kumari
```

Global scope is the scope of the global object (window in browsers, global in Node.js //what we used window.console() their window was default object of browser

- Variables declared outside a function or a block statement are global.

Local scope is the scope of a function or a block statement.

Variables declared inside a function or a block statement are local to that scope.

```
 {
    var xTG = 20; // like global variable
document.writeln("<br>Inside block: xTG = " + xTG); // outputs: 20
    let InX = 33;
document.writeln("<br>Inside block: InX = " + InX); // outputs: 33
    const InXL = 50;
document.writeln("<br>Inside block: InXL = " + InXL);// outputs: 50
```

```
  }

document.writeln("<br>Ou
tside block: xTG = " +
xTG); // outputs: 20
 document.writeln("<br>O
utside block: InX = " +
InX); // outputs:Error
 document.writeln("<br>O
utside block: InXL = " +
InXL); // outputs:Error
```

NOTE: Your global variables (or functions) can overwrite window variables (or functions).//but not object.

Any function, including the window object, can overwrite your global variables and functions.

The lifetime of a JavaScript variable starts when it is declared.

JavaScript this Keyword

The this keyword in JavaScript refers to the current execution context of a function. It is a reference to the current object. It is used to access the properties and methods of the current object.this is not a variable. It is a keyword. You cannot change the value of this.

```
//Object Method Binding
 var personThis = {
    firstName: "Sudha",
    lastName: "Kumari",
    id: 5334,
 name:this.firstName +"
"+ this.lastName,is
unexpected token in
object //this in an
object proprty refers to
global object not the
object defining, for
current object this must
```

```
be in method like
fullName
name:personpersonThis.fi
rstName +" "+
personShanaya.lastName//
Cannot read properties
of undefined (reading
'firstName')

    fullName: function
() {//here fullName is a
method of  personThis
object.
      return this + " "
+ JSON.stringify(this) +
"=  " + this.id + " " +
this.firstName + " " +
this.lastName;//here
this refers to object
personThis Because the
fullName method is a
method of the personThis
object.
      //this.firstName
is the firstName
property of this (the
person object).
 }

 sayHello: () => {
document.writeln("<br>He
llo, my name is " +
this.name + ".");//on
bind here
      //this will refer
to the xObjectBind2
    }

  };
 document.writeln("<br>F
ull Name: " +
personThis.fullName());/
/Full Name: [object
Object]
{"firstName":"Sudha","la
stName":"Kumari","id":53
34}= 5334 Sudha Kumari
var personShanaya = {
    firstName:
"Shanaya",
    lastName: "Singh",
    name: "Shanaya
Singh"
}
```

Methods like **call(), apply(), and bind()** can refer this to any object.The call() and apply() methods are predefined JavaScript methods.They can both be used to call an object method with another object as argument.

```
personThis.fullName.call
({ firstName:
"Shanaya" });//Hello, my
name is Shanaya(just
called not
updated).Shanaya is as
per the firstName
property value
passed,used to call an
object method (which is
fullName by personThis.
fullName syntax ) with
another object({ name:
"Shanaya" }) as
argument(more
precedence).//you can
define object to be
passed as an arguement
even seperatly.

PersonThis.fullName.call
(personShanaya);//Hello,
my name is Shanaya
Singh.
PersonThis.fullName.appl
y({ name:
"Sana" });//Hello, my
name is Sana
 var sayHelloBind =
PersonThis.fullName.bind
(personShanaya);//The
personShanaya object
borrows the  fullName
method from the
PersonThis object: and
stored in
sayHelloBind ,here to
call stores data it is
called as function as it
borrow
function(xObjectBind.say
Hello) hence data type
will also be function.
```

```
sayHelloBind();//Hello, my name is Shanaya.
```

The bind() method is ***not supported in arrow functions*** but the bind() method is supported in function expressions and function declarations.instead this in method will refer to global object .

```
var sayHelloBindArrow = PersonThis.sayHello
.bind(personShanaya);
sayHelloBindArrow()Hello, my name is Sudhais from global variable here window and it is defined in window with http://127.0.0.1:5500/learnjs.html?name=Sudha&age=20#link , from here this.name referes to the link part after question and it is stored in name property of window object somewhere.
```

<u>Precedence Object</u>

1 bind()
2 apply() and call()
3 Object method
4 Global scope

Alone, ***this refers to the global object*** even in strict mode but not in strict function

```
  var xGlobalObject = this;
document.writeln("<br>xGlobalObject: " + xGlobalObject);//xGlobalObject: [object Window]
```

In a function, this refers to the global object.but it should not be in strict mode

```
  function xFunction() {
    return this;[object Window]
 }
 "use strict";
```

```
  function
xFunctionStrictMode() {
   return this;
//undefined as function
in strict mode
  }
```

In an event, ***this refers to the element that received the event.***

```
document.write("<button
id='myDivThisEvent'>
myDivThisEvent</
button>");
var xEvent =
document.getElementById(
"myDivThisEvent");
xEvent.addEventListener(
"click", function () {
document.getElementById(
"DivThisEventResult").in
nerHTML +=
this;//myDivThisEvent
Result : [object
HTMLButtonElement]
  });
```

JavaScript Debugging

JavaScript debugging is the process of identifying and fixing errors in JavaScript code.

There are several tools and techniques available for debugging JavaScript code, including:

1. Browser developer tools: Most modern browsers have built-in developer tools that allow you to inspect the code, set breakpoints, and step through the code line by line.
2. Console.log(): This is a simple way to print out the value of a variable or expression to the console.A common method for debugging a problem like this is to insert a lot of console.log() statements into

the code, in order to inspect values as the script executes

//NOTE:The console.log() method may get the job done, but breakpoints can get it done faster.

3. Debugging libraries:

There are several libraries available that provide additional debugging features, such as the Chrome DevTools Debugger and the Firefox Debugger.

The Sources panel has three sections:

1. The ***left-hand side*** shows the file structure of your project.The Page tab : with the file tree. Every file that the page requests is listed here.

2. The ***middle section*** shows the code for the selected file.The Code Editor section: After selecting a file in the Page tab, the contents of that file are displayed here.

3. The ***right-hand side*** shows the call stack, which is a list of functions that were called to get to the current line of code.The Debugger section: Various tools for inspecting the page's JavaScript.

// A ***breakpoint*** lets you pause your code in the middle of its execution, and examine all values at that moment in time.With breakpoints, you can pause on the relevant code without even knowing how the code is structured.

// With ***console.log()***, you need to manually open the source code, find the relevant code, insert the console.log() statements, and then reload the page in order to see the messages in the Console. With breakpoints, you can pause on the relevant code without even knowing how the code is structured.

// In your console.log() statements you need to explicitly specify each value that you want to inspect. With breakpoints, DevTools shows you the values of all variables at that moment in time. Sometimes there are variables affecting your code that you're not even aware of.

//the incorrect sum (5 + 1 = 51) gets computed in the click event listener that's associated to the Add Number 1 and Number 2 button. Therefore, you probably want to pause the code around the time that the click listener executes. Event Listener Breakpoints let you do exactly that:

//1. Open the Debugger panel in DevTools.

//2. Click the Add Number 1 and Number 2 button.

//3. In the Debugger panel, click the Sources tab.

//4. In the left-hand side of the Debugger panel, click the file that contains the click//In the Debugger section, click Event Listener Breakpoints to expand the section.

DevTools reveals a list of expandable event categories, such as Animation and Clipboard.Next to the Mouse event category, click arrow_right Expand. DevTools reveals a list of mouse events, such as click and mousedown. Each event has a checkbox next to it.Check the click ***checkbox***. DevTools is now set up to automatically pause when any click event listener executes.

//5. Back on the demo, click Add Number 1 and Number 2 again. DevTools pauses the demo and highlights a line of code in the Sources panel. DevTools should be paused on this line of code, click the line of code that contains the function onClick() {}

// If you're paused on a different line of code, press resume ***Resume*** Script Execution until you're paused on the correct line.

//***To set a breakpoint,*** click in the left-hand margin of the code editor. A red dot will appear, indicating that a breakpoint has been set. When the code reaches the breakpoint, the debugger will pause execution and allow you to inspect the variables and expressions in the current scope.

//if (inputsAreEmpty()) {

//Rather than stepping through every line of code, you can use another type of breakpoint to pause the code closer to the probable location of the bug.

//***Line-of-code breakpoints*** are the most common type of breakpoint. When you've got a specific line of code that you want to pause on, use a line-of-code breakpoint:

//To step through the code, use the following buttons://it is represented with various symbol in right top

//1. *Step Over*: This button steps over the current line of code and continues execution.// Notice how DevTools skips a few lines of code. This is because inputsAreEmpty() evaluated to false, so the if statement's block of code didn't execute.

//2. ***Step Into***: This button steps into the current line of code and continues execution.

//3. ***Step Out:*** This button steps out of the current function and continues execution.

//4. ***Continue:*** This button continues execution until the next breakpoint is reached.

//5. ***Pause:*** This button pauses execution and allows you to inspect the variables and expressions in the

//current scope.

//6. ***Restart***: This button restarts the debugger and continues execution from the beginning.

//7. ***Disconnect:*** This button disconnects the debugger and stops execution.

//8. ***Resume***: This button resumes execution from the last breakpoint.

//9. ***Step Back:*** This button steps back to the previous line of code and continues execution.

//10. ***Step Forward:*** This button steps forward to the next line of code and continues execution.

//11. ***Step In:*** This button steps into the current line of code and continues execution.

//12. ***Step Out:*** This button steps out of the current function and continues execution.

//When you're paused on a line of code, the ***Scope tab shows you what local and global variables*** are defined at this point in execution, along with the value of each variable. It also shows closure variables, when applicable. When you're not paused on a line of code, the Scope tab is empty.
//given is what you exactly get in local
// Local
// this: Window
// addend1: 5
// addend2: 2
// sum: undefined
// Script
// Global   Window
//***Double-click a variable value to edit it.***

//The ***Watch tab*** lets you monitor the values of variables over time. Watch isn't just limited to variables. You can store any valid JavaScript expression in the Watch tab.

//Click the Watch tab.
// Click add(+) Add watch expression.
// Type "typeof sum" without " "
// Press Enter. DevTools shows typeof sum: "string". The value to the right of the colon is the result of your expression.//typeof sum:"undefined" //it is what you get
//As suspected, sum is being evaluated as a string/undefined in our case,

when it should be a number. You've now confirmed that this is the cause of the bug.

//In addition to viewing console.log() messages, you can also use the Console to evaluate arbitrary JavaScript statements. In terms of debugging, you can use the Console to test out potential fixes for bugs

//T***o evaluate a statement, type it into the Console and press Enter.*** The result of the statement will be displayed below the statement. You can also use the Console to test out potential fixes for bugs. For example, you can test out a potential fix for the bug you're currently debugging .To test out a potential fix, type the fix into the Console and press Enter. The result of the statement will be displayed below the statement. If the fix works, you can then apply the fix to your code.

// If you don't have the ***Console drawer open, press Escape*** to open it. It opens at the bottom of your DevTools window.//it shows console issues coverage what's new

// In the Console, type parseInt(addend1) + parseInt(addend2). This statement works because you are paused on a line of code where addend1 and addend2 are in scope.

// Press Enter. DevTools evaluates the statement and prints out 6, which is the result you expect the demo to produce.

// parseInt(addend1) +

parseInt(addend2).
//6
//it is exactly what you needed 7

//. You don't need to leave DevTools to apply the fix. You can edit JavaScript code directly within the DevTools UI.

// Now jump right in your file and edit your code. Google Chrome provides a very useful property list that helps you find the right line - press **Ctrl + Shift + O / Cmd + Shift + O:**//it will show Go to @Symbol //all id ,functions with code line no.

// Click resume Resume script execution.//or f8
//Navigate to the Source tab, and then click the Sources icon or press Ctrl + O to select your JavaScript file.

//NOTE:You cannot edit HTML pages in the Sources tab unless you have a Workspace set up. Check out Set Up Persistence with DevTools Workspaces for setting this up. Recommended if the code is yours.//https://developer.chrome.com/docs/devtools/workspaces?hl=en //this is what you need to get started with workspace

//However, you can set breakpoints on the JavaScript code within the <script> tags in the HTML page. This means that once you hit the breakpoint, you can run code into the Console to modify the state of your application, before continuing the execution as normal.

//https://developer.chrome.com/docs/devtools/overview?hl=en //it is complete reference of devtool or inspect function

// In the ***Code Editor,*** replace line 31,to do that navigate to the Elements tab. Locate the HTML element you want to edit, right-click on it, and choose “Edit as HTML” or double-click on the code, var sum = addend1 + addend2, with var sum = parseInt(addend1) + parseInt(addend2).

// Press Command + S (Mac) or Control + S (Windows, Linux) to save your change.

// Click label_off Deactivate breakpoints. Its color changes to blue to indicate that it's active. While this is set, DevTools ignores any breakpoints you've set.

//try this link

//https://stackoverflow.com/questions/14221579/how-do-i-add-comments-to-package-json-for-npm-install

//this github

//https://www.useblackbox.io/editor?id=61002366-6d70-4c1b-afdc-113b9e80fa36

//Now lets try this

//1. Open the DevTools by **pressing F12** or right-clicking on a page and

```
selecting Inspect or Inspect Element.
   //2. Click on the Sources tab.

   //code :
   // function updateLabel() {
   //   var addend1 = getNumber1();
   //   console.log('addend1:', addend1);
   //   var addend2 = getNumber2();
   //   console.log('addend2:', addend2);
   //   var sum = addend1 + addend2;
   //   console.log('sum:', sum);
   //   label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
   // }

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


document.write("<br><br><h1>Debugging JavaScript with Chrome DevTools</h1><br><label for='num1'>Number 1</label> <input placeholder='Number1' id='num1'> <br><label for='num2'>Number 2</label><input placeholder='Number 2' id='num2'><br><button onclick='updateLabel()'>Add Number 1 and Number 2</button><br><p id='demodebugresult'></p>");
  document.write(" ");
  function getNumber1() {
      var num1 =
document.getElementById(
"num1").value;
    return num1;
  }
  function getNumber2()
{
    var num2 =
document.getElementById(
"num2").value;
    return num2;
  }

function updateLabel() {
    var addend1 =
getNumber1();

console.log('addend1:',
addend1);
    var addend2 =
getNumber2();

console.log('addend2:',
addend2);
    // var sum = addend1
+ addend2;
    var sum = addend1 +
addend2;
    console.log('sum:',
sum);//typeof
sum:"undefined"
    //in local you will
get
    // addend1: "1"
    // addend2: "2"
    // sum: undefined

document.getElementById(
"demodebugresult").textC
ontent = addend1 + ' + '
+ addend2 + '=' + sum;

  }


label.textContent =
addend1 + ' + ' +
addend2 + '='
+sum;//Note the way of
writing.
```

4. Node.js debugging:

Node.js has its own set of debugging tools, including the built in debugger and the Node Inspector.

5. JavaScript debugging tools: There are several third-party tools available that provide additional debugging features, such as the JavaScript Debugger and the DebugBar.

6. Code editors: Many code editors, such as Visual Studio Code and Sublime Text,have built-in debugging tools.

7. JavaScript debugging frameworks: There are several frameworks available that provide additional debugging features, such as the Debug.js framework.
8. Browser extensions: There are several browser extensions available that provide additional debugging features, such as the Debugger extension for Chrome.
9. Node.js modules: There are several Node.js modules available that provide additional debugging features,such as the debug module.
10. JavaScript debugging plugins: There are several plugins available that provide additional debugging features, such as the JavaScript Debugger plugin for Visual Studio Code.
11. JavaScript debugging services: There are several services available that provide additional debugging features, such as the JavaScript Debugger

service for AWS Lambda.

12. JavaScript debugging tools for mobile devices: There are several tools available that provide additional debugging features

for mobile devices, such as the Chrome DevTools for Android and the Safari Web Inspector for iOS

13. JavaScript debugging tools for desktop applications: There are several tools available that provide additional debugging features

for desktop applications, such as the Chrome DevTools for Windows and the Safari Web Inspector for macOS

14. JavaScript debugging tools for server-side applications: There are several tools available that provide additional debugging

features for server-side applications, such as the Node.js Debugger and the Express.js Debugger

using the debugger keyword

debugger; // This will pause the execution of the code at this point and allow you to inspect the current state of the program.

The debugger keyword stops the execution of JavaScript, and calls (if available) the debugging function.

This has the same function as setting a breakpoint in the debugger.

If no debugging is available, the debugger statement has no effect.

With the debugger turned on, this code will stop executing before it executes the third line.

```
 var xDebug = 5;

  debugger;

  // The debugger statement will stop the execution of the code at this point, and allow you to inspect the current state of the program.
  // The value of xDebug is 5.

document.write("<br><br>The value of xDebug is " + xDebug + "<br>"); // This line will not be executed until the debugger is turned off.
//we will get the result
//The value of xDebug is 5
```

Complex Data Types

A complex data type can store multiple values and/or different data types together.All other complex types like arrays, functions, sets, and maps are just different types of objects.

Javascript Operator Precedence

The following operators are in order of precedence from highest to lowest:

1. ++, --, !, ~, typeof, void
2. +, -, *, /, %
3. <<, >>, >>>, <, >, <=, >=, instanceof, in
4. ==, !=, ===, !==, <>, !==
5. &
6. ^
7. |
8. ?:
9. =, +=, -=, *=, /=, %=, <<=, >>=, >>>
10. ,
11. ?: (conditional operator)
12. &&, ||
13. ?: (conditional operator)
14. =, +=, -=, *=, /=, %=, <<=, >>=, >>>
15. , (comma operator)

Maths

Maths Properties

```
Math.PI// 3.141592653589793
Math.E);// 2.718281828459045//The base of the natural logarithm, which is approximately 2.718
Math.SQRT2);//The square root of 2, which is approximately 1.4142135623730951
Math.LN2);// 0.6931471805599453//The natural logarithm of 2, which is approximately 0.6931471805599453 //base is e
Math.LN10);// 2.30258509299
```

Map

The Map object holds key-value pairs and remembers the original insertion order of the keys.

Any value (both objects and primitive values) may be used as either a key or a value

A Map object iterates its elements in insertion order (for fast and safe iteration).

```
var map = new Map()
  var map = new Map([[key, value],[key,value],[key,value]]);
  map.get(key);//gives value of key
var fruitsMap = new Map([["apples", 500],
```

Set

```
    // A JavaScript Set is an unordered collection of unique values. Each value can only occur once in a Set.
    // The values can be of any type, primitive values or objects.
    //Set is mutable,iterable, a subclass of Map and Object
    //list all Set elements (values) with a for..of loop
    console.log(typeof new Set());object

    //Set is a Map where keys are values and values are undefined
    //To create a JavaScript Set by:
    //-Passing an array
```

javascript arithmetic operators

```
   var num1 = 2
  var num2 = 3
      console.log(100 / "Apple");//NaN
   let xnantry = 100 / "Apple";
//isNaN() checks if a value is not a number
 console.log(isNaN(xnantry));// true
console.log(isNaN(44.5));//false

console.log(xnantry + num2);//NaN3//concatenation has take place

   console.log(num1 + num2);
   console.log(1 + 2);
   console.log(1 - 2);
   console.log(1 * 2);
   console.log(1 / 2);
   console.log(2 ** 3);//8
console.log(1 % 2);//gives remainder
```

Unary + Operator

```
A unary operation is an operation with only one operand.
//The unary + operator can be used to convert a everydatatype to a number. If it cannot be converted to a number, it will return NaN.
  document.write("<br>Unary + Operator<br>" + (+ "123")); // 123
  document.write("<br>Unary + Operator<br>" + typeof (+ "123")); // Unary + Operator //this kind of output is only seen in string to number and rest datatype to number will show number
  document.write("<br>" + (+ "99 88")); // NaN
var i = "5";      // i is a string
```

```
4046//The natural logarithm of 10, which is approximately 2//base is e
Math.LOG2E);// 1.4426950408889634//The base-2 logarithm of E, which is approximately
Math.LOG10E);// 0.4342944819032518//The base-10 logarithm of E, which is approximately
Math.SQRT1_2)// 0.7071067811865476//The square root of 1/2
```

Math Methods

```
Math.abs(-5.5)//5.5//Returns the absolute value of x ignoring sign
Math.round(-3.7);//-4// Returns the value of x rounded to the nearest integer like rounding off
Math.ceil(x): Returns the smallest integer that is greater than or equalto x
Math.ceil(3.3)//4
```

```
["bananas", 300],
["oranges", 200]]);//although var allows to create new object with same name but it is not allowed in map , it should be unique identifier //whole array of array neccessary
fruitsMap.set("bananas", 1000);//to update the last set value in map
var numb = fruitsMap.get("apples");
 var size = fruitsMap.size;

document.write("<br>There are " + numb + " apples"+ size + " fruits in the map")
fruitsMap.delete("apples");
fruitsMap.clear();
document.write("<br>There are fruitsMap.size :" + fruitsMap.size + " fruits in the fruitsMap after clear");
  // Expected output:
```

```
to new Set()

    //The new Set() method creates a new Set object from an array and returns it.
    // Syntax: new Set(array)

    var set1 = new Set([1, 2, 3]);

document.write(set1);// set1 : [object Set]// here for iterating for..of is used
set1[1];//undefined beacause all are converted into iterable objects hence can't get value with index like array,use for of for geting values

element.innerHTML += set1 instanceof Set;//true//adding anything here will make it false //check it because instanceof loves to be alone without any concatenation because it will change the name of set1
    //element.innerHTML = "<br>set1 instanceofCheck Settry :" + set1 instanceof Set; //false


    for (var i of set1) {

document.write("<br>set1 : " + i);//as i directly consider the value in of
        // document.write("<br>set
  var j = + i;    //j is number
  document.write("<br>" + j); // 5
document.write("<br>" + typeof (+123));  //number
document.write("<br>" + typeof (+true)); // number it will result after changing boolean to number
  document.write("<br>" + (+false)); // 0
document.write("<br>" + (+{ valueOf: function () { return 123; } }));// 123 //function is also an object
  document.write("<br>" + (+{ valueOf: function () { return "abc"; } })); //NaN
document.write("<br>" + (+new Date(2022, 0, 1))); //1640975400000
  document.write("<br>" + (+new Date())); //1723520852586 //acc to your window time
  document.write("<br>" + typeof (+new Date())); //number
document.write("<br>" + (+/abc/)); // NaN
  document.write("<br>" + (+/abc/.lastIndex)); // 0
```

```
 var inc = 5;
   console.log(inc++);
```

++ and -- only with variable not consonants

```
 //console.log(3++);//postfix Increment//error
```

Unary Increment (++)

```
console.log(inc++);//5
console.log(++inc);//6 prefix Increment
console.log(inc--);//postfix Decrement
   // console.log(--inc);//prefix Decrement
```

```
Math.ceil(-3.7)//-3
```

Math.floor(x): Returns the largest integer that is less than or equal

```
Math.floor(3.7)//3
Math.floor(1/2)//0
Math.floor(-3.7)//-4
Math.pow(2, -3);//2^-3 //0.125
```

Math.trunc(x): Returns the integer part of the number x ignoring decimals

```
Math.trunc(-3.7)//-3
Math.sign(-3.7)//-1//negative
Math.sign(0)//0//zero
Math.sign(3.7)//1//positive
Math.sqrt(16))//4
```

Math.sin(x) returns the sine (a value between -1 and 1) of the angle x (given in radians).

```
Math.sin(Math.PI / 2)//1
Math.sin(30 * Math.PI / 180 )//6.123233995736766e-17//Angle in radians = Angle in degrees x PI / 180.//30*Math.PI/180//here 30 is degree
Math.cos(Math.PI / 2)//0
Math.tan(Math.PI /2)//1.633123935319537e+16//tan(90) is Infinity
```

```
There are fruitsMap.size : 0 fruits in the fruitsMap after clear

var nameMap = new Map();
nameMap.set("FirstName", "Sudha");
document.write(nameMap.has("FirstName"));// returns true
Sudha.forEach(function (value, key, map) {

document.write("<br>forEach() method: key = " + key + ", value = "
        + value + ", map = " + map);//forEach() method: key = FirstName, value = Sudha, map = Map {"FirstName"...}
text += key + ' = ' + value + " ";

   });

document.write("<br>forEach() method: " + text);
//forEach() method: FirstName =Sudha
var entries = nameMap.entries
```

```
1[1] : " + set1[1]);//undefined

//document.write("<br>set1[i] : " + set1[i]);//undefined
    }
    // set1 : 1
    // set1 : 2
    // set1 : 3

    var set2 = new Set(["sudha", "shanaya", "sana"]);

document.write("<br>set2 : " + set2); set2 : sana,shanaya,sudha
```

Set methods

the add() method of object created by Set()

Create an empty set and use add() to add values
Set.prototype.add(value) //Don’t use prototype

```
    //The add() method adds a new element to the Set object.//here add is already a property defined in set and it is prototype chain here
    // Syntax: setObjVar.add(value)

    //to Create a Set and add values:
    var set3 = new Set();
    set3.add(1);
    set3.add(5);
    set3.add(10);
    set3.add("s");

document.write("<br>set3 : " + set3);//set3 : [object Set]
```

```
console.log(typeof NaN);//number
   console.log(100 / 0);//Infinity
console.log(-100 / 0);//-Infinity
 console.log(typeof Infinity);//number
```

Operator

let x=5;

```
==  equal to x == 8;false
x == 5 ; true
 x == "5";true
=== equal value and equal type  x === 5; true
x === "5"; false
!=  not equal x != 8; true
!== not equal value or not equal type x !== 5; false
 x !== "5";   true
 x !<== 8; true
>greater than x > 8; false
< less than  x < 8;   true
>= greater than or equal to x >= 8;  false
<=  less than or equal to x <= 8;  true
```

Logical Operators

Logical AND (&&) Operator

```
&&  and (x < 10 && y > 1) is true
||  or  (x == 5 || y == 5) is false
!   not !(x == y) is true
```

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax: condition ? value_if_true : value_if_false

```
let voteable = (age < 18) ? "Too young" : "Old enough";
```

When comparing a string with a number, JavaScript will convert the string to a number when doing the

```
Math.max(-3, 150, 30, 20, -8, -200)//150
Math.min(-3, 150, 30, 20, -8, -200)//-200
```

Math.random(): Returns a random number between 0 and 1

```
Math.random()// (Math.random() * 100) + 1 //0.20506587606408844
```

OTP

```
Math.floor((Math.random() * 1000000) + 1));//624334
 Math.log(x)// returns the natural logarithm of x.Math.E and Math.log() are twins.//base e//x is only positive
 Math.log(10))// 2.302585092994046
Math.log(-10))// NaN
Math.log2(10))// 3.321928094887362//returns the base 2 logarithm of x
Math.log10(10))// 1//base 10
Math.acos(x)// Returns the arccosine of x, in radians//arccosine (in radians)
//arccos x = cos^-1x. //arc means inverse like here cos inverse x
Math.acos(0.5)// 1.0471975511965976
Math.asin(-1)
Math.atan(1)//0
acosh(x)
```

```
();//form an array of each element in map
console.log(entries.next().value);
//['FirstName', 'Sudha']

for (var pair of entries) {//pair will take only one element array of key value in enteries into itself at a time then takes other element//pair = ['apples', 5]

document.write("<br>entries() method 1: key = " + pair[0] + " value = " + pair[1]);//works
    //pair[0]= FirstName and pair[1] =  Sudha
  }
  for (var [key, value] of entries) {//not work
document.write("<br>entries() method 2: key = " + key + ", value = " + value);
  }

  for (var [key, value] of map.entries()) {

document.write("<br>ent
```

```
 var text = "";
 for (var i of set3){
       text += i +" ";
docment.write("<br>set3 : " + text);//if used inside for loop it will form pattern hence use it outside for..of loop
        //     set3 : [object Set]
        // set3 : 1
        // set3 : 1 5
        // set3 : 1 5 10
        // set3 : 1 5 10 s

//document.write("<br>set3 : " + i);//not good idea to represent
        // set3 : 1
        // set3 : 5
        // set3 : 10
        // set3 : s
    }

document.write("<br>Nice approch set3 : " + text);//set3 : 1 5 10 s

    //Create a Set and add variables:
    var set4 = new Set();
    var aval = "a";
    var bval = "b";

set4.add(aval);//"a" added
    set4.add(bval);

document.write("<br>set4 : " + set4);
    // set4 : b,a
```

Use for..of to get values

has() method

The has() method returns true if the value exists in a Set,

comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false. //no need to use Number("10") when comparing number with string

```
2 < 12  true
  2 < "12"     true
  2 < "John"   false
//John from string converted into number NaN
  2 > "John"   false
  2 == "John"  false
  "2" > "12"   true  When comparing two strings, "2" will be greater than " 12 as it starts comparing from left to right digit to digit
Number("2") < Number("12") //true
  "2" == "12" false
```

The Nullish Coalescing Operator (??)

The ?? operator returns the first argument(defined above) if it is not nullish (null or undefined). Otherwise, it returns the second argument(after ?? which is default value in case of null).

```
let nameNoNullish = "Shanaya"
  let nameCoal = nameNoNullish ?? "Sudha";
  document.write("<br> nameCoal : " + nameCoal);//Shanaya
 let nameNullish = null;
  let nameCoalescing = nameNullish ?? "Sudha";
  document.write("<br> nameCoalescing : " + nameCoalescing);//Sudha//default passed if found null
```

bitwise operator

```
  console.log(1 & 2);
  console.log(1 | 2);
  console.log(1 ^ 2);
```

```
 Math.acosh()Returns the hyperbolic arccosine of x
//method returns a value between 0 and Infinity
 //The hyperbolic arc-cosine of the parameter,NaN if the parameter is less than 1 or not numeric.
Math.acosh(1))//0
Math.acosh(-1))// NaN
Math.asinh() method returns a value between -Infinity and Infinity
 //The hyperbolic arc-sine of the parameter,NaN if the parameter is not numeric.
Math.asinh(-1)// Math.asinh(-1) :-0.881373587019543
Math.atanh()
//method returns a value between -Infinity and Infinity
Math.atanh(1))// Infinity
Math.atanh(0))//0
atan2(y, x)// Returns the arctangent of the quotient of its arguments
  // The Math.atan2() method returns a value between -PI and PI
  // It is similar
```

```
ries() method 2: key = " + key + ", value = " + value);
    //it works when map.entries() used or defining var enteries just above //otherwise enteries worked for only upto one for loop and it forget enteries even like something is was defined in pair and [key,value] both.Hence it is needed to defined map.enteries either above for loop or with the for loop like var [key, value] of map.entries().
  }
var text = "";
  var entries = map.entries();
  for (var pair of entries) {
    text += pair[0] + ' = ' + pair[1] + " ";
  }

document.write("<br>entries() method 3: " + text);////now works only after defining var enteries again at above of for //entries() method: pair[0]= FirstName and pair[1] =  Sudha
var text = "";
  for (const x of map.keys()) {
    text += x + " ";
  }
  document.write(text);
firstname
for (const x of map.values()) {
    text += x + " ";
```

otherwise it returns false.

```
IsSudhaInSet2 = set2.has("sudha"); //true //no datatype required for IsSudhaInSet2 because it returns boolean due to has but it also works even when variable created
  IsSudhaInSet2 = set2.has("Sudha"); //false//it is case sensitive
```

forEach() Method

does not change the original Set.

```
var text = '';

set2.forEach(function (value) {
        text += value + " "
        //document.write("<br>set2.forEach() : " + value);
    }
   );

document.write("<br>set2.forEach() : " + text);//set2.forEach(): sudha shanaya sana
```

Values() Method

The values() method returns a Set Iterator object(then use for of to retrieve), which contains the values for each element in the Set object.The values() method is a block scope method, so the loop variable is local to

```
  console.log(1 << 2);
  console.log(1 >> 2);
```

comma operator

```
console.log(1, 2, 3, 4, 5);
```

The typeof operator returns "function" for function expressions and function declarations.

instanceof operator

The instanceof operator tests to see if the prototype property of a constructor appears anywhere in the prototype chain of an object. The return value is a boolean value.

Its behavior can be customized with Symbol.hasInstance.

Syntax : object instanceof constructor

object is created by you to use constructor function property ,it is same as the value to be passed in defined parameterised constructor and with new keword for object as reference

```
// setup instanceOf check that assumes that // anything with canEat property is an animal
class Animal { static [Symbol.hasInstance](obj) { if (obj.canEat) return true; } }

let obj = { canEat: true }; alert(obj instanceof Animal); // true: Animal[Symbol.hasInstance](obj) is called
```

An instance of a class is an object. It is also known as a class object or class instance. As

to calculating the arc tangent of y / x

```
  // tan^-1(y / x)
Math.atan2(- 1, -1)); // -PI/4 radians or -45 degrees//-0.785398 //-2.356194490192345
Math.cbrt(27
  )); // 3//cube root
math.exp()
//method returns E raised to the power of x (Ex). 'E' is the base of the natural system of logarithms (approximately 2.718282) and x is the number passed to it.
Math.exp(0.5)); // 1.648721271
```

JavaScript Bitwise Operations

&   AND Sets each bit to 1 if both bits are 1

| OR Sets each bit to 1 if one of two bits is 1

^   XOR Sets each bit to 1 if only one of two bits is 1

~   NOT Inverts all the bits

<< Zero fill left shift Shifts left by pushing zeros in from the right and let the leftmost bits fall off

>> Signed right

```
  }
```

// use the values() method to sum the values or find total in a map:

```
  var sumValueMethod = 0;//identifir sum has been already declared using either let or const ,suppose
```

fruitsMap not cleared

```
  for (var xValueMethod of fruitsMap.values()) {
      sumValueMethod += xValueMethod;
  }

document.write("<br><br>sum of values: " + sumValueMethod); //sum of values : 1000
```

//Objects as Keys

```
  //Being able to use objects as keys is an important Map feature.
  var apples = { name: 'Apples' };
  var bananas = { name: 'Bananas' };
  var map = new Map();
  map.set(apples, 500);
  map.set(bananas, 1000);

document.write("<br>")
  for (var i of map) {
    document.write(i + "<br>");

document.write("<br>map.get('apple')" +
```

the block

```
 var myIterator = set2.values();

var text = "";

  for (var i of myIterator) {
    text += i + " ";
  }
document.write(text);// sudha shanaya sana
```

Key() Method

The keys() method returns a Set Iterator object, which contains the keys for each element in the Set object.

```
   var myIterator = set2.keys();
```

Now use for..of loop

using key:value in set

```
  var keyValSet = new Set();
  keyValSet.add({ Name: "Sudha" });
  keyValSet.add({ Nickname: "Shanaya" });
 var myIteratorKeys = keyValSet.keys();

 var text = "";

  for (var i of myIteratorKeys) {
    // text += i + " ";
    //text += i.keys + " ";//undefined
    text += keyValSet.keys() + " ";//keyValSet.keys() : 0 1 2
  }
```

such, instantiation may be referred to as construction. Whenever values vary from one object to another, they are called instance variables. These variables are specific to a particular instance.

```
 // Execute until Infinity
   while (myNumberI != Infinity) {
   myNumberI = myNumberI * myNumberI;//itself stop for smooth flow
 console.log(myNumberI);  }
```

=	x = y	Same As x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Shift Assignment Operators

<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y

Bitwise Assignment Operators

&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

Logical Assignment Operators

&&=	x &&= y	x = x && (x = y)
||=	x ||= y	x = x || (x = y)
??=	x ??= y	x = x ?? (x = y)

toString(base) method

```
let myNumberT = 32.868;
myNumberT.hasOwnProperty("toString"); // true
  document.writeln(
    "Hexatrigesimal (base 36): " +
myNumberT.toString(36) +
"<br>" +
    "when no base given in toString() then it will not convert given no. into base of something, it will only convert number into 'number' : " +
myNumberT.toString() +
```

shift Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

>>> Zero fill right shift Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

In detail given in main notes

JavaScript Errors

JavaScript errors are thrown when something goes wrong in your code. They can be used to handle errors in your code, and to provide more information about what went wrong.Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Try,catch,finally ,throw

Note: Error is in form of object having properties: name and message.

The ***name*** property is the name of the error, and the ***message*** property is the description of the error.

```
 try {
    // code that might throw an error
```

```
map.get('Apples'));//undefined ,Remember: The key is an object (apples), not a string ("apples"):

document.write("<br>map.get(apples)" + map.get(apples) + '<br>');//500
    // [object Object],500  //Apples as key would have been shown when apples would have been declared directly instead of declaring as an object
    // [object Object],1000


   }

//Here is an example of how to use objects as keys in a Map:

  var map = new Map([
    ["apple", 5],
    ["banana", 10],
    ["orange", 15],
    ["apple", 20],
  ]);

  //The Map object will automatically ignore the duplicate key "apple" and keep the last value for the key "apple" which is 20.
  //The last key-value pair overwrites the previous one because the key is the same.
  //The map now contains the following key-value pairs:

document.write("<br><br>Map with objects as keys: " + map);//Map
```

```
document.write("<br>keyValSet.keys () : " + text);
//if added direct object

keyValSet.add({Name:"Sudha" , Age:20 , Priority:"Most"})

var myIteratorKeys = keyValSet.keys();

  var text = "";
  for (var i of myIteratorKeys) {
    text += i + " ";
  }

document.write("<br>keyValSet.keys () : " + text);//keyValSet.keys () : [object Object]
```

enteries() method

The entries() method returns an iterator that contains all key-value pairs in the Set object.

A Set has no keys, so the entries() method returns [value,value]

```
 var keyValSet = new Set([{ Name: "Sudha", Age: 20, Priority: "Most" }
])//keyEnteriesTry () : [object Object],[object Object]   //enteries


  var keyValSet = new Set(["Sudha", 20, "Most"])
  var text = " ";
  var keyEnteriesTry = keyValSet.entries();

document.write("<br>key
"<br>" +
    "Duotrigesimal (base 32): " +
myNumberT.toString(32) +
"<br>" +
    myNumberT.toString(16)
+ "<br>" +
    "Duodecimal (base 12): " + myNumberT.toString(12)
+ "<br>" +
    "Decimal (base 10): " +
myNumberT.toString(10) +
"<br>" +
    "Octal (base 8): " +
myNumberT.toString(8) +
"<br>" +
    "Binary (base 2): " +
myNumberT.toString(2)+"<br>"+(100+23).toString()//'123');
```

toExponential()

```
 let xnm = 9.656;
xnm.toExponential(4);//9.6560e+4
```

toFixed()

```
xnm.toFixed(2);// 9.66(round off after decimal length)
```

toPrecision()

```
xnm.toPrecision(2);// 9.7 (round off to specified length )
```

valueOf()

```
xnm.valueOf()//9.656
```

Returns a number as a number datatype defined

ParseInt() method

```
parseInt("123.45"); //123
```

parseInt() parses(analying string) a string and returns a whole number

```
parseInt("10 20 30"); //10
parseInt("123,45") //123
parseInt("10 years"); //10
parseInt("years 10");//NaN
```

ParseFloat() method

```
parseFloat("123.45");//123.45
parseFloat("10 20 30"); //10
```

```
adddlert("Welcome guest!");
    //  xTry = 1 / 0;//infinity
  } catch (e)
{//error object e is passed
    // code that will run if an error is thrown//JavaScript catches adddlert as an error, and executes the catch code to handle it.

document.writeln("<br>Caught exception: " + e);//Caught exception: ReferenceError: adddlert is not defined
e.message;// adddlert is not defined
e.name;// ReferenceError
 e["name"]);// ReferenceError
 }
```

The finally Statement

The finally statement is used to execute a block of code regardless of whether an exception is thrown or not. It is used to release any system resources that are allocated in the try block.

The throw Statement

The throw statement is used to create a custom error.

The exception can be a JavaScript String, a Number,

```
with objects as keys: [object Map]
  //Map(4) { 'apple' => 20, 'banana' => 10, 'orange' => 15 }

document.write("<br><br>Map with objects as keys map.apple : " + map.apple);//undefined//it would have worked only if  instead of ["apple", 5], {"apple": 5} would have present
  //The above code will return undefined because the map does not have a property called "apple"

document.write("<br><br>Map with objects as keys[0]: " + map[0]);//undefined
  //The above code will return undefined because the map does not have a property called "0"

document.write("<br><br>Map with objects as keys map.get('apple') : " + map.get('apple'));//20
```

The groupBy() method returns a Map object where each key is a value from the iterable and each value is an array of values that correspond to the key.

```
 // Create an Array of
```

```
EnteriesTry : " + text);//keyEnteriesTry : Set

  for (var i of keyEnteriesTry) {
    // document.write("<br>keyEnteriesTry : " + i);
    //keyEnteriesTry : [object Object],[object Object],[object Object]

    // keyEnteriesTry : Sudha,Sudha
    // keyEnteriesTry : 20,20
    // keyEnteriesTry : Most,Most

    text += i + "<br>";
    // text += i.values ;//function values() { [native code] }

  }

document.write("<br>keyEnteriesTry () : " + text);
```

Break and Continue

for all kind of loops

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

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

```
var iBreak = 0;
```

```
parseFloat("123,45"); //123.45
parseFloat("10 years"); //10
```

Number()

```
document.writeln(" Number(new Date(;1970-01-02')) :" + Number(new Date("1970-01-02")));
Number(true);//1
Number("10.33");//10.33
Number("10,33");//NaN
Number("10 33");//NaN
Number("Sudha");//NaN
let numObj = new Number(123);
//numObj can't be equated to num as one is object and other is 123 (number)
//object can't be equated with any
```

methods of Number

Number.isInteger()

```
Number.isInteger(10); //true
Number.isInteger(10.5); //false
```

Number.isSafeInteger()

```
Number.isSafeInteger(9007199254740991); //true
```

Number.isSafeInteger(9007199254740992); //false

Integer Precision

JavaScript has a limit on the precision of integers. This limit is 2^53 - 1. This means that any integer larger than this limit will be rounded to this limit. This is known as the safe integer limit.

let xs = 9007199254740992 === 9007199254740993;//true as it hasrounded the limit

MAX_SAFE_INTEGER is a constant that represents the maximum safe integer value in JavaScript. This value is +(2^53 - 1), which is 9007199254740991.

let max = Number.MAX_SAFE_INTEGE

a Boolean or an Object:

```
  throw "Too big"; // throw a text
  throw 500; // throw a number
  throw true; // throw a boolean
  throw {name: 'Error', message: 'Something went wrong'}; // throw an object
    throw new Error('Something went wrong'); // throw an object with a constructor
  throw new TypeError('Something went wrong'); // throw an object with a constructor
  throw new SyntaxError('Something went wrong'); // throw an object with a constructor
  throw new RangeError('Something went wrong'); // throw an object with a constructor
  throw new ReferenceError('Something went wrong'); // throw an object with a constructor
  throw new EvalError('Something went wrong'); // throw an object with a constructor
  throw new URIError('Something went wrong'); // throw an object with a constructor
  throw new AggregateError('So
```

objects as element

```
  var fruitsGroup =
[//must be an array
      { name: "apples", quantity: 300 },
      { name: "bananas", quantity: 500 },
      { name: "oranges", quantity: 200 },
      { name: "kiwi", quantity: 150 }
    ];
// Callback function to Group Elements
  function myCallback({ quantity }) {//take quantity:300 first//hence { quantity } = 300 at first //like first element in object form,it's key quantity's value.
      return quantity > 200 ? "ok" : "low";
      //string value returned from callback function is ok and low according to the parameter passed
    }

  // Group by Quantity
  var GroupByresult = Map.groupBy(fruitsGroup, myCallback);
  //final visualization is
var GroupByresult = new Map([["ok", [{ name: "apples", quantity: 300 }, { name: "bananas", quantity: 500 }]], ["low", [{ name: "oranges", quantity: 200 }, { name: "kiwi", quantity: 150 }]]]);//it is exactly the GroupByresult .....///Hurah! you did
```

```
while (iBreak < 10) {
    if (iBreak == 5) {
      break;//to stop from 5 and come out of while loop//break stops loop not related with if
    }
document.write("<br>iBreak : " + iBreak);
    iBreak++;
  }
  document.write("<br>I am out of while loop due to break");
  //break and continue
  // iBreak : 0
  // iBreak : 1
  // iBreak : 2
  // iBreak : 3
  // iBreak : 4
  // I am out of while loop due to break

document.write("<br>")
  for (let i = 0; i < 5; i++) {
    if (i == 1) {
      continue;//to skip 3 and continue with next iteration
    }
document.write("<br>i : " + i);
  }

document.write("<br>loop ended");
  // i : 0
  // i : 1,not printed due to continue , it starts for new one
  // i : 2
  // i : 3
  // i : 4
  // loop ended
```

JavaScript Labels

JavaScript has a feature

```
R;
    console.log(max);
```

MIN_SAFE_INTEGER is a constant that represents the minimum safe integer value in JavaScript. This value is -(2^53 - 1), which is -9007199254740991.

```
    let min = Number.MIN_SAFE_INTEGER;
    console.log(min);
```

Number Properties

```
Number.E; //2.71828... (Euler's number)
Number.LN2; //0.693147... (natural logarithm of 2)
Number.EPSILON; //2.220446049250313e-16 //2.220446... (The difference between 1 and the smallest number > 1)
Number.MAX_VALUE; //1.7976931348623157e+308
Number.MIN_VALUE; //5e-324
Number.POSITIVE_INFINITY//Infinity
Number.NEGATIVE_INFINITY//-Infinity
Number.NaN; //NaN
```

Iterable is an object which can be looped over or iterated over with the help of a for loop. Objects like lists(Array), tuples, sets, dictionaries, strings, etc. are called iterables.

NOTE: myobject is not iterable

//Technically, iterables must implement the Symbol.iterator method.

```
mething went wrong'); // throw an object with a constructor
    throw new DOMException('Something went wrong'); // throw an object with a constructor
    throw new DOMException('Something went wrong', 1); // throw an object with a constructor// throw new DOMException('Something went wrong', 1, 'myError'); // throw an
    throw new DOMException('Something went wrong', 1, 'myError', 'myMessage
```

Note the use of JS to create Html tags

```
document.writeln("<br><br>Please input a number between 5 and 10 :");
document.writeln("<br><input id='demoThrow' type='text'>");
document.writeln("<br><button type='button' onclick='myFunctionThrow()''>Test Input Throw</button>");

document.writeln("<br><p id='pThrow'>Input is :</p>");
    function
```

```
it
for (var i of GroupByresult) {

document.write("<br><br> i of GroupByresult :" + i + "<br>"); //ok
    //low

document.write("i.ok " + i.ok)//undefined

document.write("<br>i.value " + i.value)

document.write("<br>i[0] " + i[0])//low

document.write("<br>i[1] " + i[1])//i[1] [object Object],[object Object]

document.write("<br>i[2] " + i[2])//undefined

document.write("<br>i[1].name " + i[1].name)//undefined

document.write("<br>i[1][0] " + i[1][0])// [object Object]
    }

    var text = "<br>These fruits are Ok: <br>";

document.write("<br>GroupByresult.get('ok') :" + GroupByresult.get("ok"));//GroupByresult.get('ok') :[object Object],[object Object]
    for (var i of GroupByresult.get("ok")) {//identifier x is already defind because it is declared as
```

called labels, which allows you to break or continue a loop from any point in your code. A label is a word that you can use to identify a loop or block of code or a switch.

The syntax of a label is :

```
  label: statement
```

In JavaScript, labels are very limited: you can only use them with break and continue statements, and you can only jump to them from a statement contained within the labeled statement. You cannot jump to this label from anywhere in the program.

```
var iLabelTry = 0;

  outerT: while (iLabelTry < 5) {

document.write("<br>iLabelTry in OuterT  : " + iLabelTry);
    innerT: while (iLabelTry <= 3) {//if any no. will come here it will form infinite loop as there is no increment or decrement in this while loop even though if condition will not execute it and think as I have to form a loop ilabelTry=0 will stuck here and never stops and never ever enters into if to break the outerloop

document.write("<br>iLabelTry in innerT : " + iLabelTry);
      if (iLabelTry == 3) {
document.write("<br>iLa
```

//String, Array, TypedArray, Map and Set are all iterables, because their prototype objects have a Symbol.iterator method.

```
typeof Symbol()       //
Returns symbol
```

Destructuring

Destructuring is a feature in JavaScript that allows you to unpack values from arrays or objects into distinct variables. It is a shorthand way to assign values from an array or object to variables.Destructuring can be used with arrays and objects. *The values are assigned in the order they appear in the array and object*.

Array Destructuring

```
arrayDestructure = [223, 4,
32, , 432]
  var [var1, var2] =
arrayDestructure;
console.log(var1); //
Outputs: 223
  console.log(var2); //
Outputs: 4
var [var1, var2, var3,
var4, var5,var6,var7] =
[223, 4, 32, , 432]//blank
means undefined
document.writeln("var2); //
Outputs: 4
document.writeln(var4); //
Outputs: undefined
document.writeln(var6); //
Outputs: undefined
```

Skipping Array Values

```
var [a, , , d, e,f] =
arrayDestructure;//Here, a,
, , d, e and f are the
variables that will hold
the values from the array
document.writeln("<br>e :"
+ f)// Outputs: undefined
```

Array Position Values

```
var { [0]: First, [1]:
Middle, [2]: Last } =
arrayDestructure;
document.writeln(Last)//
Outputs:32
```

```
myFunctionThrow()
{
    var xThrow =
document.getElementById("demoThrow")
.value;;
    try {
      if
(xThrow.trim() ==
"") throw "empty";
      if
(isNaN(xThrow))
throw "not a
number";
      xThrow =
Number(xThrow);
      if (xThrow
< 5) throw "too
low";
      if (xThrow
> 10) throw "too
high";
    }
    catch (err) {

document.writeln("
<br>Input is : " +
err);
document.getElementById("pThrow").innerHTML = "Input
is : " + err;
    }
finally {
document.writeln("
<br>Finally block
executed");
    }
  }
```

The ***Error*** object is the base class for all exceptions in JavaScript. ErrorName : Description

EvalError : An error has occurred in the eval() function

RangeError: A number "out of

```
object.//returns
{name:"apples",
quantity:300} in i at
first then
    text += i.name + "
" + i.quantity +
"<br>";
  }
  text += "<br>These
fruits are low: <br>";
  for (var i of
GroupByresult.get("low"
)) {
    text += i.name + "
" + i.quantity +
"<br>";
  }
  document.write(text);
// Map.groupBy()
result: [object Map]
  // These fruits are
Ok:
  // apples 300
  // bananas 500

  // These fruits are
low:
  // oranges 200
  // kiwi 150
```

NOTE:Object.groupBy() groups elements into a JavaScript object.//[object Object]

Map.groupBy() groups elements into a Map object.// [object Map]

Here is an example of how to use the groupBy() method in JavaScript:

```
var numbers = [1, 2, 3,
4, 5, 6, 7, 8, 9, 10];

  var grouped =
numbers.reduce((acc, x)
=> {//here acc is
```

```
belTry in if of
innerT : " +
iLabelTry);
          break
outerT;//it will break
the outerT
        }
        iLabelTry++;//for
making this while loop
to stop and now no
error it will give

document.write("<br>iLabelTry in innerT at
last after  iLabelTry+
+; : " + iLabelTry);
     }
     //below
instructions will never
execute because inner
while loop execute
itself for all
iterations inside and
at last come out of
outerT and these
instructions will never
execute
document.write("<br>iLabelTry at last of
outerT : " +
iLabelTry);
     iLabelTry++;
document.write("<br>iLabelTry at last after
iLabelTry++ of outerT :
" + iLabelTry);
   }

document.write("<br>
Detailed explaination
of sumLabel and aLabel
below");

  outerloop: while
(sumLabel < 12) {

document.write("<br>IN
outerloop : sumLabel =
" + sumLabel + " aLabel
= " + aLabel);//IN
outerloop : sumLabel =
```

The Rest Property(spread ...name three dots inside [],{}, ())

The rest property is used to assign the rest of the array values to a variable.

```
var [a, b, ...rest] = arrayDestructuring4;//Here, a and b are the variables that will hold the first two values from the array,and rest will hold the rest of the values from the array
document.writeln("<br>rest :" + rest)// Outputs: 32, , 432
document.writeln("<br>typeof rest :" + typeof rest);//object
```

Object Destructuring

```
var object= {
    nameT: "Shanaya",
     age: 20,
Roll:30,
  };
  var { nameT: newName, age,Roll:RollNo = 25 ,occupation = "Student" } = objectPropertyAlias;
```

Object destructuring allows you to assign a new name to an existing property name.Now only new name will work and old one will be No more defined hence give error.Here nameT name changed to newName.

```
document.writeln("<br>newName : " + newName); // Outputs: Shanaya
document.writeln("<br>nameT : "+nameT); // ERROR as property name changed but you can call the old one without destructuring(using objectName.propertyName)
document.writeln("<br>Occupation : " + occupation);
//Outputs: Student (default value when not present in object. Default value will be written only when not
```

range" has occurred

```
function numRangeError() {
try {
if (xRangeInput < 0 || xRangeError > 500) throw "out of RangeError";
}
catch (err) {
if (err == "out of RangeError")
document.getElementById("pRangeError").innerHTML +=
            " out of RangeError";
     }
}
```

ReferenceError :An illegal reference has occurred.A ReferenceError is thrown if you try to access a variable that has not been declared.

```
xReferenceError = y + 1;//here y not declared
```

SyntaxError: A syntax error has occurred

```
eval("console.log('Hello World!'"); // Missing closing parenthesis //eval not working now//it will give error if no catch present with try
```

TypeError: A type error has occurred

```
var xTypeError = 5;
xTypeError.toUpperCase();   // You cannot convert a number to upper
```

URIError: An error in encodeURI() has occurred.A URIError

```
[object Object] and x is current element value
document.write(" JSON.stringify(acc) : " + JSON.stringify(acc) + " x :" + x);// here JSON.stringify(acc) used to change then object into string to see the values or data stored in object as it is and x is value stored in array //we are writing just to check in between
//JSON.stringify(acc) : {} x :1
    // JSON.stringify(acc) : {"1":[1]} x :2

//JSON.stringify(acc) : {"0":[2],"1":[1]} x :3

//JSON.stringify(acc) : {"0":[2],"1":[1,3]} x :4
JSON.stringify(acc) : {"0":[2,4],"1":[1,3]} x :5
JSON.stringify(acc) : {"0":[2,4],"1":[1,3,5]} x :6
JSON.stringify(acc) : {"0":[2,4,6],"1":[1,3,5]} x :7
JSON.stringify(acc) : {"0":[2,4,6],"1":[1,3,5,7]} x :8
JSON.stringify(acc) : {"0":[2,4,6,8],"1":[1,3,5,7]} x :9
JSON.stringify(acc) : {"0":[2,4,6,8],"1":[1,3,5,7,9]} x :10

if (!acc[x % 2]) {
//acc[1] not present then create the acc[1]=[]//x % 2 shows
```

```
0 aLabel = 1
     //// IN outerloop : sumLabel = 3 aLabel = 2 reseved values from continue in the outer loop as it is continued fron innerloop hence value of sumLabel = 3 aLabel = 2 because these values are send by innerloop to call this outerloop
     ///// IN outerloop : sumLabel = 5 aLabel = 2//////IN outerloop : sumLabel = 7 aLabel = 2////// IN outerloop : sumLabel = 9 aLabel = 2//////// IN outerloop : sumLabel = 11 aLabel = 2

     // Label for inner loop
     innerloop: while (aLabel < 3) {

document.write("<br>IN innerloop:  sumLabel = " + sumLabel + " aLabel = " + aLabel);// IN innerloop: sumLabel = 0 aLabel = 1
          //IN innerloop: sumLabel = 1 aLabel = 2//// IN innerloop: sumLabel = 3 aLabel = 2///// IN innerloop: sumLabel = 5 aLabel = 2////// IN innerloop: sumLabel = 7 aLabel = 2/////// IN innerloop: sumLabel = 9 aLabel = 2//////// IN innerloop: sumLabel = 11 aLabel = 2
          sumLabel += aLabel;

document.write("<br>IN innerloop after
defined in object property
)
```

here Roll:RollNo = 25 uses both default and alias

String Destructuring

String destructuring is used to extract values from a string using the spread operator (...var)

```
var stringDestructuring = "Hello, my name is Sudha and I am 20";
  var [greeting, name, occupation, age] = stringDestructuring.split(", ");//split at , and forms array
document.writeln("<br>Greeting : " + greeting); // Outputs: Hello
document.writeln("<br>Name : " + name); // Outputs:my name is Sudha and I am 20
document.writeln("<br>Occupation : " + occupation); // undefined
var [a1, a2, a3, a4, a5] = stringDestructuring;//destructure each letter when no split used
document.writeln("<br>a5 : " + a5); // Outputs: o
```

Destructuring Maps

```
var mapDestructuring = new Map([["name", "Sudha"], ["age", "25"], ["city", "Bangalore"]]);
var { nameM, age, city, ...rest } = mapDestructuring;
```

You can't destructure Map directly, you will have to either convert to object first or do something exotic, like using Proxy to intercept get calls. Something like this with helper function:

```
document.writeln("<br>nameM :" + nameM)// Outputs: undefined
```

is thrown if you use a function that is not allowed in a URI. URIError is a serializable object, so it can be cloned with structuredClone() or copied between Workers using postMessage(). It typically arises when decoding or encoding URIs (Uniform Resource Identifiers) with functions like ***decodeURIComponent() or encodeURI().***

URIError():Creates a new URIError object. A ***URI (Uniform Resource Identifier)*** is a sequence of characters that identifies a logical or physical resources The JavaScript URI() function is a function in JavaScript which is used to encode any URI (Uniform Resource Identifier) that substitutes certain instances of character with one, two, or three escape sequences representing UTF-8 patterns for the encoding of characters with the encoding URI functions.

Instance properties:Also inherits instance properties from its parent Error.

These properties are defined on

```
index no. of object //"1": [2,4,6,8,10]
      acc[x % 2] = [];//it creates new empty array for each unique key as value and itself become the key ,here unique values out of modulus 2 is 0 and 1
    };//if acc[x % 2] is not null it means it is always true for all elements in x as x starts from 1.Hence it
    acc[x % 2].push(x);//push element into acc[index either 0 or 1]//"1": [2,4,6,8,10]
      return acc;
   }, {});//in (function ,{}), {} shows the starting value of acc={} //but to see acc={ } ,we need to use JSON.stringify(acc) to get the data of [object Object]
```

```
JSON.stringify(value[, replacer[, space]]);
```

value: The JavaScript object or value that you want to convert to JSON.

replacer (optional): A function or an array that can be used to modify the behavior of the stringification. It can filter out properties or alter their values.

space (optional): A string or number that adds indentation, white space, and line breaks to the returned JSON string for readability.

```
sumLabel += aLabel  : sumLabel = " + sumLabel + " aLabel = " + aLabel);//IN innerloop after sumLabel += aLabel : sumLabel = 1 aLabel = 1// IN innerloop after sumLabel += aLabel : sumLabel = 3 aLabel = 2//// IN innerloop after sumLabel += aLabel : sumLabel = 5 aLabel = 2///// IN innerloop after sumLabel += aLabel : sumLabel = 7 aLabel = 2////// IN innerloop after sumLabel += aLabel : sumLabel = 9 aLabel = 2/////// IN innerloop after sumLabel += aLabel : sumLabel = 11 aLabel = 2//////// IN innerloop after sumLabel += aLabel : sumLabel = 13 aLabel = 2  ,it no more enters in if loop
      if (aLabel === 2 && sumLabel < 12) {

document.write("<br>IN if : sumLabel = " + sumLabel + " aLabel = " + aLabel);// IN if : sumLabel = 3 aLabel = 2//// IN if : sumLabel = 5 aLabel = 2///// IN if : sumLabel = 7 aLabel = 2
        ////// IN if : sumLabel = 9 aLabel = 2/////// IN if : sumLabel = 11 aLabel = 2
        continue outerloop;//value sumLabel = 3 aLabel = 2 will be passed in continue outerloop
```

```
document.writeln("<br>rest :" + rest);//rest :[object Object]
//map is destructured using for..of only
  var text = "";
  for (var [key, value] of mapDestructuring) {
    text += key + " is " + value + "<br> ";
  }
document.writeln("<br>text :" + text)
  //text :name is Sudha
  // age is 25
  // city is Bangalore
```

The yield operator is used in generators to produce a series of values over time, instead of computing them at once and returning them in an array, which would require storing them in memory.

The yield operator is used in the following way:

1. A **generator function** is defined with the function keyword, just like a regular function.
2. The generator function contains the yield keyword, which is used to produce a value.only one at a time and next time it gives next index value or simply next value of result.
3. The generator function is called, and it returns an iterator object.
4. The iterator object is used to retrieve the next value produced by the generator function.

```
function* myGenerator() {
    yield 1;
    yield 2;
    yield 3;
  }
  //To use the yield operator, you can use the
```

URIError.prototype and shared by all URIError instances.

URIError.prototype.constructor:The constructor function that created the instance object. For URIError instances, the initial value is the URIError constructor.

URIError.prototype.name:Represents the name for the type of error. For URIError.prototype.name, the initial value is "URIError".

Instance methods:Inherits instance methods from its parent Error.

```
  try {
decodeURIComponent("%E2%80%93");
//URIError: malformed URI sequence  // Incorrectly encoded character
decodeURIComponent("%");
decodeURI("%E2%80%93"); //no URIError
     decodeURI("%%%");//URIError // You cannot URI decode percent signs
     throw new URIError("Hello");
  }
  catch (e) {

e.name;//URIError
document.writeln(e instanceof URIError); // true
```

```
document.write("<br><br>Map with groupBy() method: " + JSON.stringify(grouped));
```

Summarize

The groupBy() method groups the numbers by their remainder when divided by 2.

The result is a Map object where each key is a remainder and each value is an array of numbers that correspond to the key.

Map with groupBy() method: {"0":[1,3,5,9],"1":[2,4,6,8,10]}

```
//Here is an example of how to use the groupBy() method in JavaScript with objects as


  var fruitsReduce = [
    { name: "apples", quantity: 300 },
    { name: "bananas", quantity: 500 },
    { name: "oranges", quantity: 200 },
    { name: "kiwi", quantity: 150 },
  ];
  var grouped = fruitsReduce.reduce((acc, x) =>{
   document.write(" acc : " + JSON.stringify(acc) + " x :" + JSON.stringify(x));

    //acc : {} x : {"name":"apples","quantity":300}
    // acc : {"apples":[300]} x : {"name":"bananas","quantity":500}
```

```
      }
document.write("<br>IN innerloop at last : sumLabel = " + sumLabel + " aLabel = " + aLabel);// IN innerloop at last : sumLabel = 1 aLabel = 1//////// IN innerloop at last : sumLabel = 13 aLabel = 2
      aLabel++;

document.write("<br>IN innerloop at last after aLabel++ : sumLabel = " + sumLabel + " aLabel = " + aLabel);//IN innerloop at last after aLabel++ : sumLabel = 1 aLabel = 2 and this value will be the start of 2nd iteration of innerloop//////// IN innerloop at last after aLabel++ : sumLabel = 13 aLabel = 3
    }
  }
```

for loop with label

```
document.write("<br>");

  let strLabel = '';
  loop1: for (let i = 0; i < 5; i++) {
    if (i === 1) {
      continue loop1;
    }
    strLabel = strLabel + i;
  }
document.write("<br>strLabel" + strLabel + " ");//0 2 3 4//works as
```

label with block

const carsLabel = ["BMW", "Volvo", "Saab", "Ford"];

```
following code:
  let generator = myGenerator();//The generator function is called, and it returns an iterator object. //here iterator object is generator
  document.writeln("<br>" + generator.next().value); // Output: 1
  document.writeln("<br>" + generator.next().value); // Output: 2
  document.writeln("<br>" + generator.next().value); // Output: 3
  //The yield operator is useful when you need to produce a series of values over time, and you don't want to store them in memory all at once.
```

Parameteric function for yield

```
function* foo(index) {
    while (index < 2) {
      yield index;
      index++;
    }
  }
  const iterator = foo(0);
iterator.next().value;
  // Expected output: 0
iterator.next().value;
  // Expected output: 1
function* countAppleSales() {
    const saleList = [3, 7, 5];
    for (let i = 0; i < saleList.length; i++) {
      yield saleList[i];
    }
  }
  const appleSales = countAppleSales();
document.writeln("<br>" + appleSales.next().value); // Output: 3
function* counter(value) {
    while (true) {
```

```
//false when "<br>" is used as it changes variable name because of concatenation
document.writeln("<br>" + e.message); // " URI malformed "
e.name//"URIError"
document.writeln("<br>" + e.stack); //URIError: URI malformed at decodeURIComponent () at http://127.0.0.1:5500/learnjs.html?name=Sudha&age=20:5045:13// Stack of the error
  }
```

HTML validation

The HTML Validation is used to validate the input data.

The HTML Validation can be used with the try-catch statement to handle errors.

```
document.writeln("<br><input id='demoHtml' type='number' min='5' max='10' step='1'>");//step and max min is very useful here.

document.writeln("<br><button type='button' onclick='myFunctionHtml()''>Test Input</button>");
```

```
    // acc : {"apples":[300],"bananas":[500]} x : {"name":"oranges","quantity":200} acc : {"apples":[300],"bananas":[500],"oranges":[200]} x : {"name":"kiwi","quantity":150}
  if (!acc[x.name]) acc[x.name] = [];//here x.name value will be the key and its value will be array

acc[x.name].push(x.quantity);//In the value array , quantity is pushed
    return acc;
  }, {});

document.write("<br><br>Map with groupBy() method: " + JSON.stringify(grouped));
 //The groupBy() method groups the fruits by their name.

  //Map with groupBy() method: {"apples":[300],"bananas":[500],"oranges":[200],"kiwi":[150]}

 typeof map;//returns object

  typeof fruitsMap.get("apples");//returns value of key//number
  typeof fruitsMap.set("apples",
```

```
 listLabel: {
    textLabel += "<br>" + carsLabel[0] + "<br>";
    textLabel += carsLabel[1] + "<br>";
    break listLabel;
    textLabel += carsLabel[2] + "<br>";
    textLabel += carsLabel[3] + "<br>";
  }

document.write("<br>textLabel :" + textLabel);
```

if statement

Uppercase letters (If or IF) will generate a JavaScript error.

```
  if (condition) {
  //  block of code to be executed if the condition is true
  }

var world = document.getElementById("worldclass").innerHTML;
  if (world == "Shanaya's World") {
    console.log("yes it's shanaya's world");

  }
  else {
    console.log("no it's not shanaya's world");
  }
```

JavaScript Switch Statement

Use the switch statement to select one of many code blocks to be executed.

```
  switch(expression) {
   case x:
    // code block
    break;
```

```
      const step = yield value++;

      if (step) {
        value += step;
      }
    }
  }

  const generatorFunc = counter(0);

console.log(generatorFunc.next().value); // 0

console.log(generatorFunc.next().value); // 1

console.log(generatorFunc.next().value); // 2

console.log(generatorFunc.next().value); // 3

console.log(generatorFunc.next(10).value); // 14
//after 10 yields from current yield

console.log(generatorFunc.next().value); // 15

console.log(generatorFunc.next(10).value); // 26
```

The delete operator is used to delete a property from an object.

```
//The delete operator is used in the following way:
  //1. The delete operator is used with the object name and the property name, separated by .
  //2. The delete operator returns true if the property is deleted successfully, and false otherwise.
  var objT = { a: 1, b: 2, c: 3 };
  document.writeln("<br>" + delete objT.a); // Output:
```

```
document.writeln("<br><p id='pHtml'>Input is :</p>");
  function myFunctionHtml() {
    var xHtml = document.getElementById("demoHtml").value;

//document.writeln("<br>Input is : " + xHtml);

document.getElementById("pHtml").innerHTML = "Input is : " + xHtml;
  }
```

```
10);//returns object
  typeof fruitsMap.delete("apples");//returns boolean
  typeof fruitsMap.has("apples");//returns boolean
  typeof fruitsMap.clear();//returns undefined
  typeof fruitsMap.keys();//returns iterator object
  typeof fruitsMap.values();//returns iterator object
  typeof fruitsMap.entries();//returns iterator object
  typeof fruitsMap.size;//returns number
  typeof fruitsMap.forEach(callback);//returns undefined
  typeof fruitsMap.forEach(callback, thisValue);//returns undefined
document.write("<br><br>fruitsMap2 instanceof Map : " + fruitsMap2 instanceof Map);//false


document.write("<br>");

document.write(fruitsMap2 instanceof Map);//true//it should not contain any other concatenate string before fruitsMap2
```

Differences between JavaScript Objects and Maps:

```
  Object
```

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

switch done with getDay

When JavaScript reaches a break keyword, it breaks out of the switch block. This will stop the execution inside the switch block.

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The default Keyword specifies the code to run if there is no case match

If no default label is found, the program continues to the statement(s) after the switch.

default case does not have to be the last case in a switch block

If default is not the last case in the switch block, remember to end the default case with a break.

Switch cases use strict comparison (===)

```
  //A strict comparison can only be true if the operands are of the same type and same value.
  let textSwitchComparison = "0";
  let textSwitchComparisonValue;
  switch (textSwitchComparison) {
```

```
true
  document.writeln("<br>" + objT); // [object Object]
  document.writeln("<br>" + JSON.stringify(objT)); // Output: {b: 2, c: 3}
```

for of is only for iterable objects like array, string, map, set, etc.

);//new line for each value in iterable in console

```
//Iterating Over an Array
    var myarray = ["shanaya", "sudha", "sneha"];//string array
    var myarray = [2, 5, 3, 5, 3, 5]//number array
    for (var value of myarray) {
        console.log(value
    }
    //Iterating Over a String
    var myString = "Shanaya"
    for (var value of
```

```
  Not directly iterable
  Do not have a size property
  Keys must be Strings (or Symbols)
  Keys are not well ordered
  Have default keys

  Map
  Directly iterable
  Have a size property
  Keys can be any datatype
  Keys are ordered by insertion
  Do not have default keys
```

```
    case 0:
textSwitchComparisonValue = "Off";
      break;
    case 1:
textSwitchComparisonValue = "On";
      break;
    default:
textSwitchComparisonValue = "No value found";
  }
  document.write("<br> The switch result textSwitchComparison textSwitchComparisonValue : " + WeekendBlock);//No value found //as here 0 with case is number and 0 in textSwitchComparison is string //hence different type //The strict comparison results false while matching with case 0
```

.toString() with array

```
const cars = [
    "Saab",
    "Volvo",
    "BMW"
  ];//Spaces and line breaks are not important. A declaration can span multiple lines

console.log("cars.toString() :" + cars.toString());//Saab,Volvo,BMW
  console.log("typeof cars :" + typeof cars);
  console.log("typeof cars.toString() :" + typeof cars.toString());//string
```

```
myString) {
        console.log(value);
        //S
        //h
        //a
        //n
        //a
        //y
        //a
    }
    //Iterating Over a Set
    var mySet = new
Set(["shanaya", "sudha",
"sneha"]);
    for (var value of
mySet) {
        console.log(value);
        //shanaya
        //sudha
        //snaha
    }
    //Iterating Over a Map
    var myMap = new Map([
        ["key1", "value1"],
        ["key2", "value2"],
        ["key3", "value3"]
    ]);
    for (var [key, value]
of myMap) {
        console.log(key + "
= " + value);
        // key1= value1
        // key2= value2
        // key3= value3

    }
```

for in if as for of .for in is for all objects,array,string and everything with index

```
    for (var value in
myarray) {
        console.log(value+
" : " +
myarray[value]//myarray[val
ue] is wrong only value is
used in array
);
```

.at() method for array

```
var p = [1, 2, 3, 4,
5];
  var q = ["hello",
"world", "shanaya",
"world"];
  for (let index = 0;
index < p.length;
index++) {
    const element =
p[index];

console.log(element);
    for (let i = 0; i <
q.length; i++) {
      const element =
q[i];

console.log(element);
    }
  }
  //for accessing last
array
  console.log
(q.at(q.length -
1));//world
  //.at() method  for
array

console.log(q.at(2));//
shanaya
```

```
    }
    var myobject = {
        name: "shanaya",
        age: 20,
        city: "pune"
    };

    // for (var value of
myobject) {
    //
console.log(value);
    // }
    //TypeError: myobject
is not iterable
```

Creating javascript
iterator for object

```
//JavaScript Iterators
    //The iterator protocol
defines how to produce a
sequence of values from an
object.
    //An object becomes an
iterator when it implements
a next() method //to call
its next value from
iterables
    //The next() method
returns an object with two
properties: done and value.
    // The done
property //done (true or
false)//true if the
iterator has completed and
false if the iterator has
produced a new value
    //The value
property //value (the next
value in the sequence)//The
value returned by the
iterator(Can be omitted if
done is true)
    //Creating an Iterator

    //Home Made Iterable

    //    function*
myIterable() {
    //     yield 1;
    //     yield 2;
    //     yield 3;
    //    }
    function myIterable() {
        let m = 0;
        return {
            nextTry:
function () {  //here
nextTry is the label for
function block and it
returns value and done//
nextTry is not function
name //nextTry can also be
taken as object method
     m += 10;//update m to
10 from 0
    return { value: m,
done: false };//done false
shows that not to stop
updating m on each call of
n. nextTry()
NOTE  : for multiple
returns in a function use
block with return like
return{ }
}   };   }

    // Create Iterable
myIterable().nextTry();
[object Object]
myIterable().nextTry;//
function() {} whole
function code ,just like
properties value is asked
myIterable.nextTry;//
undefined as () necessary
for functi9on name
myIterable().nextTry().valu
e;// 10
myIterable().nextTry().valu
e);// 10//it wll restart
looking from the top of
function as myIterable() is
used
```

//just like printing the return value after each call from start

```
 NOTE:
myIterable().nextTry();//it
will not update the value
hence n object is declared
below to store the last
```

data , always m=10 when called directly with this

```
var n = myIterable();//for
storing the last return
value and at start it takes
0
    n.nextTry(); // 10 //m
= 0+10 ,and this updated
value of m gets stored in n
for next time//just like
myIterable().next();//next
method returns only one
next value at a time//here
myIterable() holds the
return values of the
function  myIterable() with
labelled function next
    n.nextTry(); // 20// m
= 10+10
    n.nextTry(); // 30// m
= 20+10 //only update but
to see changes use value
property also
n.nextTry().value;//40
```

The problem with a home made iterable is that It does not support the JavaScript for..of statement. for..of is only supported in JavaScript iterable

Solution to iterate over custom iterator

The code provided below defines a custom iterator for the `myNumbersIterable` object, allowing you to iterate over a series of numbers.

```
    myNumbersIterable = {};

    // Make it Iterable

    //NOTE : It is wrong
that Symbol.iterator is
name of function in the
```

```
object
    //The
`myNumbersIterable[Symbol.i
terator]` function defines
how the object should be
iterated over using a
`for...of` loop.
    //In order to be
iterable, an object must
implement the [Symbol.
iterator]() method, meaning
that the object (or one of
the objects up its
prototype chain) must have
a property with a [Symbol.
iterator] key which is
available via constant
Symbol.
    //prototype chain are
the the further
functionality or value
added using the prototype
properties//[Symbol.
iterator] is a property key
along with [Symbol.
iterator]:value ,something
like this. or think as
symbols are used to call
the iterator property of
prototype. and symbol is
only present in [] boxes.

myNumbersIterable[Symbol.it
erator] = function ()
{//here [Symbol. iterator]
key is a property of
prototype used////here
myNumbersIterable is
function and
[Symbol.iterator] is
property of prototype as
key
        //
myNumbersIterable =
function Symbol.iterator ()
{ //wrong
        //  let n = 0;
        var n = 0;
        done = false;//by
default
        return { //- The
function returns an object
that contains a `next`
method.
      next() {
        n += 10;
        if (n ==
50) { done = true }//now
done gets updated to true
from false
        return
{ value: n, done:
done };//value:n, done:true
      }
    };//the iterator
takes only the values from
return hence in for..of
nij.value is not used
  }
  //iterating over the
JavaScript iterable
  for (const nij of
myNumbersIterable) {

document.write("<br>x : " +
nij);
    // x : 10
    // x : 20
    // x : 30
    // x : 40

    //
document.write("<br>x : " +
nij.value);//NOTE: here
nij.value is undefined
because next here is
function and here values
are returned by itself as
Symbol.iterator is used
    // x : undefined
    // x : undefined
    // x : undefined
    // x : undefined
Executed 4 times
  }
```

Explaination

```
//  The code provided
defines a custom iterator
for the `myNumbersIterable`
```

object, allowing you to iterate over a series of numbers. Here’s a breakdown:

```
    // 1. **Custom Iterator**:
    //    - The `myNumbersIterable[Symbol.iterator]` function defines how the object should be iterated over using a `for...of` loop.
    //In order to be iterable, an object must implement the [Symbol. iterator]() method, meaning that the object (or one of the objects up its prototype chain) must have a property with a [Symbol. iterator] key which is available via constant Symbol.//here myNumbersIterable is function and [Symbol.iterator] is property of prototype as key

    //    - Inside this function, a variable `n` is initialized to `0` to track the current number, and `done` is initialized to `false` to indicate whether the iteration is complete.

    // 2. **Next Method**:
    //    - The function returns an object that contains a `next` method.
    //    - Each time the `next` method is called, it increments `n` by `10`.
    //    - It checks if `n` has reached `50`. If it has, it sets `done` to `true`, indicating that there are no more values to iterate.
    // 3. **Return Value**:
    //    - The `next`
method returns an object
with two properties:
    //      - `value`: the
current value of `n`.
    //      - `done`: a
boolean indicating whether
the iterator has completed
(i.e., whether `n` has
reached `50`).

    // 4. **Looping Through
myNumbersIterable**:
    //    - The code uses a
`for...of` loop to iterate
over `myNumbersIterable`.
    //    - For each number
generated by the iterator,
it appends the number
followed by a line break
(`"<br>"`) to the `text`
string.

    // **Summary**: The
code creates a custom
iterator for
`myNumbersIterable` that
generates multiples of `10`
(from `10` to `40`) in a
loop, appending each number
to a string until it
reaches `50`.
```

for each

used in php
file ,set,array,map but not object

```
 var myarray = ["shanaya",
"sudha", "sneha"];
  myarray.forEach(function
(value) {
    console.log(value);
  });
```

Using arrow function

```
myarray.forEach((value) =>
{
    console.log(value);
  });
```

```
numbers = [1, 2, 3, 4, 5];
  numbers.forEach((number,
index) => {
    console.log('Index: ' +
index +
      ', Value: ' +
number);
  });
  numbers.forEach((number,
index, numbers) => {
    console.log('Index: ' +
index + ', Value: ' +
number + ',Array: ' +
numbers);
  });

myobject.forEach((value,
index, myobject) => {
    console.log('Index: ' +
index + ', Value: ' + value
+ ',Array: '+myobject);
   //  Uncaught TypeError:
myobject.forEach is not a
function
  });

  myobject.forEach(function
(value) {
    console.log(value);
    //Uncaught TypeError:
myobject.forEach is not a
function
  });
```

Loop

for - loops through a block of code a number of times

for/in - loops through the properties of an object

for/of - loops through the values of an iterable object

while - loops through a block of code while a specified condition is true

do/while - also loops through a block of code while a specified condition is true

```
var loop =
document.getElementById("id").innerHTML
    //alert(loop);
```

JavaScript Conditional Statements

There are three types of conditional statements in JavaScript:

1. if statement
2. if...else statement
3. switch statement

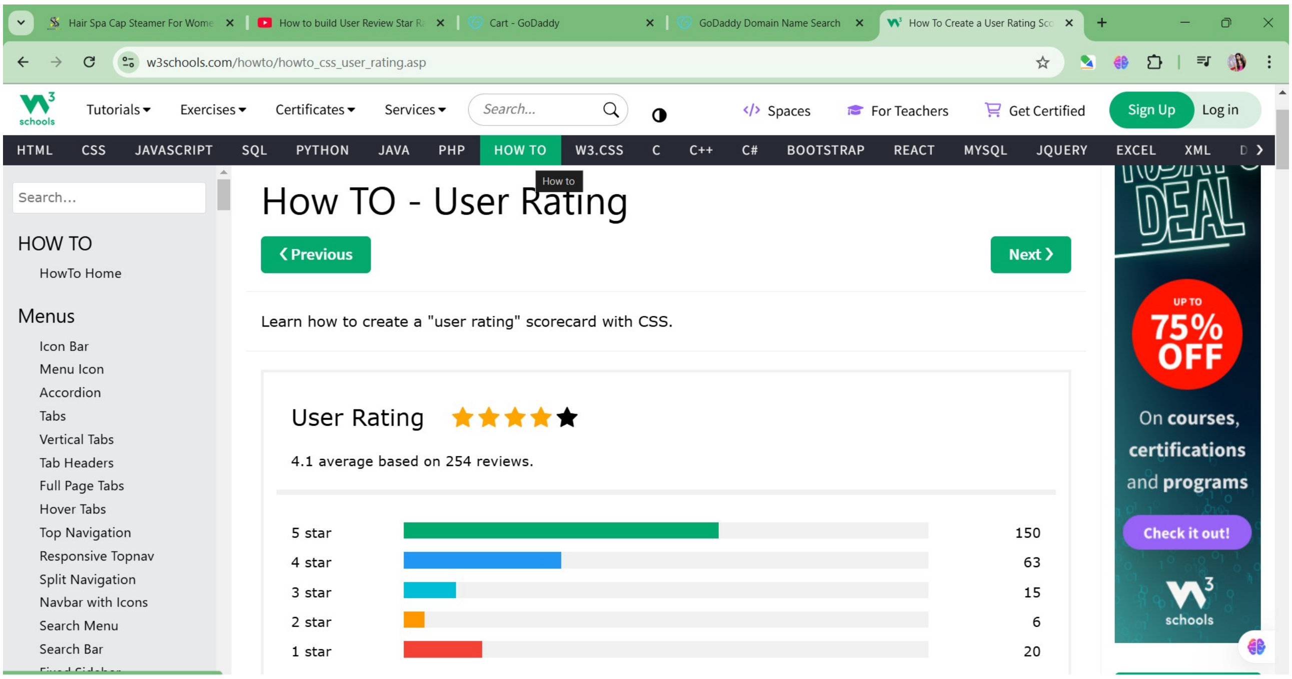
already covered all in codes

```
    for (var i = 0; i < 10; i++) {
        console.log(i);

document.getElementById("loop").innerHTML
= loop + i;//shows only last 9 Shanaya's
World9 as loop id is only once and value
is updating everytime to same position
and loop stores only Shanaya's World as
per loop defined above.
    }

var i = 0;//initialized for not setting
again and again inside for loop
    while (i < 10) {
        console.log(i);

document.getElementById("while").innerHTM
L = whiletry + i;
        i++;//i f not used then the
condition never evaluates to false, an
infinite loop can occur.
    }

do {
        console.log(i);
        document.getElementById("do
while").innerHTML = dowhile + i;
        result += i;
        i++;//if not used then it will
form infinite loop

    } //while (i<10);
    while (i > 10);
    console.log(result);
```

To add js-confetti
https://www.npmjs.com/package/js-confetti

https://www.cssscript.com/tag/confetti/

Learn:



How TO - User Rating

❮ Previous Next ❯

Learn how to create a "user rating" scorecard with CSS.

User Rating ★★★★★

4.1 average based on 254 reviews.

5 star	🟩	150
4 star	🟦	63
3 star	🟦	15
2 star	🟧	6
1 star	🟥	20

How to has many effects to apply.