Advanced

sssssssss

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

Q1:Array Introduction

In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements.

Declaration of an Array

There are basically two ways to declare an array.

```
var a = [];
var b = new Array()

1.let a = "Hello World!"
alert(a);

2.typeof "hello";
typeof (8);
typeof true
typeof ninja: "turtle"

3.let str="welcome";
str.length;
str.toUpperCase()
str.charAt(1);
str.indexOf("c");
str.concat("Ninja");
"JavaScript".concat("Ninja");

4.Array
var myArray = [1,2];
var myArray = new Array(24,5);
var pizzas = ["Margherita", "Mushroom", "Spinach & Rocket"];

function isEqual()
// if length is not equal
if(a.length 1=5. length)
return "False",
else
// comapring each element of array
for(var arr = 'False')
return "False",
return "True",
var arr= isEqual();
document.write(arr);
```

Q2:Array Destructuring

```
let animals =['Dog','Cat','Rat'];
let a = animals[0];
let b = animals[1];
let c = animals[2];
```

```
console.log(a,b,c);
```
o/p: *Dog Cat Rat*

after destructuring
```
let animals =['Dog','Cat','Rat'];
let a,b,c;
[a,b,c] = animals;
console.log(a,b,c);
```
o/p:*Dog Cat Rat*

And
```
let animals =['Dog','Cat','Rat'];
let [a,b,c] = animals;
console.log(a,b,c);
```
o/p: *Dog Cat Rat*

Again
```
let animals =['Dog','Cat','Rat'];
let a,b,c;
[a, ,b] = animals;
console.log(a,b);
```
o/p: *Dog Cat*

Array reverse()

arr.reverse() is used for in place reversal of the array. The first element of the array becomes the last element and vice versa.
Syntax:arr.reverse()
This function does not take any argument,This function returns the reference of the reversed original array.

```
function func() (
var arr = [34, 234, 567,4];
console.log(arr);

var new_arr = arr.reverse();
console.log(new_arr);
}
func();
```

Basic Array Methods

1.Array.push() : Adding Element at the end of an Array. As array in JavaScript are mutable object, we can easily add or remove elements from the Array. And it dynamically changes as we modify the elements from the array.

```
var number_arr = [ 10, 20, 30, 40, 50 ];
var string_arr = [ "piyush", "gourav", "smruti", "ritu" ];
number_arr.push(60);
number_arr.push(70, 80, 90);
string_arr.push("sumit", "amit");
console.log("After push number_arr " + number_arr);
console.log("After push string_arr " + string_arr);
```
o/p:
After push number_arr 10,20,30,40,50,60,70,80,90
After push string_arr piyush,gourav,smruti,ritu,sumit,amit

2.Array.unshift() : Adding elements at the front of an Array
```
var number_arr = [ 20, 30, 40 ];
var string_arr = [ "amit", "sumit" ];
number_arr.unshift(10, 20);
string_arr.unshift("sunil", "anil");
console.log("After unshift number_arr " + number_arr);
console.log("After unshift string_arr " + string_arr);
```
o/p
After unshift number_arr 10,20,20,30,40
After unshift string_arr sunil,anil,amit,sumit

3.Array.pop() : Removing elements from the end of an array
```
var number_arr = [ 20, 30, 40, 50 ];
var string_arr = [ "amit", "sumit", "anil" ];
number_arr.pop();
string_arr.pop();
```

```
console.log("After pop number_arr " + number_arr);
console.log("After popo string_arr " + string_arr);
```

4.Array.shift() : Removing elements at the beginning of an array

```
var number_arr = [ 20, 30, 40, 50, 60 ];
var string_arr = [ "amit", "sumit", "anil", "prateek" ];
number_arr.shift();
string_arr.shift();
console.log("After shift number_arr " + number_arr);
console.log("After shift string_arr " + string_arr);
```

5.Array.splice(): Splice is very useful method as it can remove and add elements from the particular location.

```
var number_arr = [ 20, 30, 40, 50, 60 ];
var string_arr = [ "amit", "sumit", "anil", "prateek" ];
number_arr.splice(1, 3);
number_arr.splice(1, 0, 3, 4, 5);
string_arr.splice(1, 2, "xyz", "geek 1", "geek 2");
console.log("After splice number_arr " + number_arr);
console.log("After splice string_arr " + string_arr);
```

o/p:*After splice number_arr 20,3,4,5,60*
After splice string_arr amit,xyz,geek 1,geek 2,prateek

Array sort()

arr.sort() function is used to sort the array in place in a given order according to the compare() function. If the function is omitted then the array is sorted in ascending order.
compareFunction(a,b) is less than 0
Then a comes before b in the answer.
compareFunction(a,b) > 0
Then b comes before a in the answer.
compareFunction(a,b) = 0
Then the order of a and b remains unchanged.

```
function func() {
//Original string
var err
//sorting the array
document.write(arr.sort());
document.write(arr);
func();
```

Array.findIndex() Method

It returns index of the first element in a given array that satisfies the provided testing function. Otherwise -1 is returned.
It does not execute the function once it finds an element satisfying the testing function.
It does not change the original array.
Parameters:
function:- A function to be run for each element in the array.
currentValue:- The value of the current element.
index:- The array index of the current element.
array:- The array object with current element belongs to.
thisValue:- A value to be passed to the function to be used as its “this” value.
If this parameter is empty, the value “undefined” will be passed as its “this” value.

```
var array = [ 10, 20, 30,110, 60 ];
function finding_index(element) {
return element > 25
}
console.log(array.findlndex(finding_index));
```

This Array.findIndex() Method can be used to find the index of an element in the array that is a prime number (or returns -1 if there is no prime number).

```
function isPrime(n) ( if (n === 1) {
return false;
} else if (n === 2) { return true;
} else {
for (var x = 2; x < n; x++) (
if (n % x === 0) { return false;
}
```

```
}
return true;
}
}
// Printing -1 because prime number is not found.
console.log([ll 4, 6, 8,12].(isPrime));
// Printing 2 the index of prime number (7) found.
console.log([ 4, 6, 7,12 ].findlndex(isPrime));
```

array.entries()

The array.entries() is an inbuilt function in JavaScript which is used to get a new Array that contains the key and value pairs for each index of an array.
It returns an array of index and values of the given array on which array.entries() function is going to work.
Here array.entries() method in javaScript used to find out key and value pairs for each index in any given array.

```
var array = ['geeksforgeeks', 'gfg', 'Jhon'];
var iterator = array.entries();
console.log(iterator.next().value);
console.log(iterator.next().value);
console.log(iterator.next().value);
```

o/p:
Array [0, "geeksforgeeks"]
Array [1, "gfg"]
Array [2, "Jhon"]
Whenever we need to get key and value pair for each index in any array thet time we use array.entries() Method

```
var array = ['geeksforgeeks', 'gfg', 'Jhon'];
var iterator = array.entries();
// printing key and value pair from the given array using for loop.
for (let e of iterator) (
console.log(e);
}
```

Array every()

Array.every() function checks whether all the elements of the array satisfy the given condition or not that is provided by a function passed to it as the argument. The syntax of the function is as follows:
arr.every(function[, This_arg])

ArgumentThe argument to this function is another function that defines the condition to be checked for each element of the array. This function argument itself takes three arguments:

array (optional)
This is the array on which the .every() function was called.

index (optional)
This is the index of the current element being processed by the function.

element (Compulsory)
This is the current element being processed by the function.
This function returns Boolean value true if all the elements of the array follow the condition implemented by the argument function. If one of the elements of the array does not satisfy the argument function, then this function returns false.

```
function ispositive(element, index, array) {
return element > 0;
}
function func() (
var arr = [ 11, 89, 23, 7, 98 ];
// check for positive number
var value = arr.every(ispositive);
document.write(value);
}
func();
```

Array forEach()

arr.forEach() function calls the provided function once for each element of the array. The provided function may perform any kind of operation on the elements of the given array.
The syntax of the function is as follows:
arr.forEach(function callback(currentValue[, index[, array]])

[, thisArg]);

ArgumentsThe argument to this function is another function that defines the condition to be checked for each element of the array. This function itself takes three arguments:
arrayThis is the array on which the .forEach() function was called.
indexThis is the index of the current element being processed by the function.
elementThis is the current element being processed by the function.
Another argument thisValue is used to tell the function to use this value when executing argument function.

Return value The return value of this function is always undefined. This function may or may not change the original array provided as it depends upon the functionality of the argument function.

```
function func() (
const items = [1, 29,47];
const copy = [];
items.forEach(function(item){
copy.push(item*item);
}
document.write(copy);
}
func();
```

Array lastIndexOf

arr.lastIndexOf() function is used to find the index of the last occurrence of the search element provided as the argument to the function. The syntax of the function is as follows:
arr.lastIndexOf(searchElement[,index])

Arguments The first argument to this function is the searchElement which is the value to be searched in the array. Second argument to this function is the optional index argument which defines the starting index in the array from where the element is to be searched backwards. If this argument is not provided then index arr.length – is taken as the starting index to begin the backward search as it is the default value.
Return value This function returns the index of the last occurrence of the searchElement. If the element cannot be found in the array, then this function returns -1

```
var array = [2, 98, 12, 45];
print(array.lastIndexOf(98,2));
console.log(array.lastIndexOf(98,2));
```

o/p: 1

copyWithin

The copyWithin() method copies part of an array to the same array and returns it, without modifying its size
i.e, copies array element of a array within the same array.

```
var array = [ 1, 2, 3, 4, 5, 6, 7 ];
console.log(array.copyWithin(0, 4, 5));
```

Output:[5, 2, 3, 4, 5, 6, 7]

array.values

The array.values() function used to returns a new array Iterator object that contains the values for each index in the array
i.e, it prints all the elements of the array.
It returns a new array iterator object.
i.e, elements of the given array.

```
var array = [ 'a', 'gfg', 'c', 'n' ];
var iterator = array.values();
for (let elements of iterator)
console.log("elements")
```

Array slice

arr.slice() function returns a new array containing a portion of the array on which it is implemented. The original remains unchanged.
arr.slice(begin, end)

```
var arr = [23,56,87,32,75,13];
var new_arr = arr.slice(3,10);
console.log(arr);
console.log(new_arr);
```

Output:
[23,56,87,32,75,13]
[32,75,13]

array.toLocaleString

The array.toLocaleString() is used to convert the element of the given array to string.

arr.toLocaleString();
It return a string representing the elements of the array.
var name = ["Ram", "Sheeta", "Geeta"];
var number1 = 3.45;
var number2 = [23, 34, 54];
var A = [name, number1, number2];
var string = A.toLocaleString();
console.log(string);
Output:"Ram, Sheeta, Geeta, 3.45, 23, 34, 54"

Array.of()

The array.of() function creates a new array instance with variables present as the argument of the function.
Array.of(element0, element1, ...)
Whenever we need to get elements of any array that time we take help of Array.of() method in JavaScript.
console.log(Array.of(['Ram', 'Rahim', 'Geeta', 'Sheeta']));
Output: Array [Array ["Ram", "Rahim", "Geeta", "Sheeta"]]

ArrayBuffer Object

An ArrayBuffer object is used to represent a generic, fixed-length raw binary data buffer. The contents of an ArrayBuffer cannot be directly manipulated and can only be accessed through a DataView Object or one of the typed array objects. These Objects are used to read and write the contents of the buffer. More than one DataView or typed array objects can be added to one ArrayBuffer and any changes to one object can be easily seen by the other objects view.
The following are the typed arrays:
Float32Array, Float64Array, Int8Array, Int16Array, Int32Array, Uint8Array, Uint8ClampedArray, Uint16Array, Uint32Array.
Syntax:new ArrayBuffer(byteLength)
Parameters: It accepts one parameter i.e. bytelength which denotes the size, in bytes, of the array buffer to be created.
Return value: It returns a new ArrayBuffer object of the specified size and the content is initialized to 0.

```
//Create a 16byte buffer
var buffer = new Array8uffer(16);
//Create a DataView referring to the buffer
var viewl = new DataView(buffer);
var view2 = new Int8Array(buffer);
//Put value of 32bits viewl.
setInt32(0, 0x76543210);
console.log(viewl.getInt32(0).toString(16));
//prints only 8bit value
console.log(viewl.getInt8(0).toString(16));
```

Properties : ArrayBuffer.byteLength: The byteLength property returns the length of the buffer in bytes.
ArrayBuffer.prototype: This property allows to add properties to all ArrayBuffer objects.
Methods:
ArrayBuffer.isView(arg): If arg is one of the ArrayBuffer views (typed array objects or a DataView) then true is returned otherwise, false is returned.

ArrayBuffer.transfer(oldBuffer [, newByteLength]): The contents from the oldbuffer specified is either truncated or zero-extended by the specified newByteLength and is returned as a new ArrayBuffer.

Instance Methods:ArrayBuffer.slice() and ArrayBuffer.prototype.slice():A new ArrayBuffer is returned whose contents are a copy of this ArrayBuffer's bytes from begin, inclusive, up to end, exclusive.

Array some()

arr.some() function check whether at least one of the elements of the array satisfies the condition checked by the argument function. The syntax of the function is as follows:
arr.some(arg_function(element, index, array), thisArg)

ArgumentsThe argument of this function is another function that defines the condition to be checked for each element of the array. This function itself takes three arguments:

arrayThis is the array on which the .some() function was called.
indexThis is the index of the current element being processed by the function.
elementThis is the current element being processed by the function.

Another argument this Value is used to tell the function to use this value when executing argument function.
This function returns true even if one of the elements of the array satisfies the condition(and does not check the remaining values) implemented by the argument function. If no element of the array satisfies the condition then it returns false.

```
function checkAvailability(arr, val)
return arr.some(function(arrVal) {
return val === arrVal;
});
}

function func() (
var arr = [2, 5, 8,1, 4]
// Checking for condition
document.write(checkAvailability(arr, 2));
document.write(checkAvailability(arr, 87));
}
func();
```

join

Array.join() function is used to join the elements of the array together into a string.
Array.join([separator])
Argument: (separator)A string to separate each elements of the array. If leave it by default array element separate by comma(,).
This function returns a string created by joining all the elements of the array using the separator. If the separator is not provided then the array elements are joined using comma (,) as it is the default separator for this function. If an empty string is provided as the separator then the elements are joined directly without any character in between them. If array is empty then this function returns an empty string.
var a = [1, 2, 3, 4, 5, 6]; print(a.join('|'));

fill

Array.fill() function is used to fill the array with a given static value. The value can be used to fill the entire array or it can be used to fill a part of the array.
arr.fill(value, start, end)
Here arr is the array to be filled with the static value.
ArgumentsThis function takes three arguments.
valueIt defines the static value with which the array elements are to be replaced.**start (Optional)**it defines the starting index from where the array is to be filled with the static value. If this value is not defined the the starting index is taken as 0. If start is negative then the net start index is length+start.**end (Optional)**This argument defines the last index up to which the array is to be filled with the static value. If this value is not defined then by default the last index of the i.e arr.length – 1 is taken as the end value. If the end is negative, then the net end is defined as length+end.
This function does not return a new array. Instead of it modifies the array on which this function is applied.

```
function func()
var arr = [1, 23, 46, 58];
arr.fill(87, 1, 3);
document.write(arr);
}
func0;
```

o/p:[1, 87, 87, 58]

find

arr.find() function is used to find the first element from the array that satisfies the condition implemented by a function. If more than one element satisfies the condition then the first element satisfying the condition is returned. Suppose that you want to find the first odd number in the array. The argument function checks whether the argument passed to it is odd number or not . The find() function calls the argument function for every element of the array. The first odd number for which argument function returns true is reported by find() function as the answer. The syntax of the function is as follows:
arr.find(function(element, index, array), thisValue)
ArgumentsThe argument to this function is another function that defines the condition to be checked for each element of the array. This function itself takes three arguments:

array: This is the array on which the .filter() function was called.
index: This is the index of the current element being processed by the function.
element: This is the current element being processed by the function.
Another argument thisValue is used to tell the function to use this value when executing argument function.
Return value This function returns the first value from the array that satisfies the given condition. If no value satisfies the given condition, then it returns undefined as its answer.

```
function isOdd(element, index, array)
{
return (element % 2=1);
}
```

```
function func()
{
var array = [ 4, 5, 8, 11];
document.write(array.find( IsOdd));
}
func();
```

concat

Array.concat() function is used to merge two or more arrays together. This function does not alter the original arrays passed as arguments

```
var num1 = [[23]];
var num2 = [89, [67]];
print(num1.concat(num2));
```

filter

arr.filter() function is used to create a new array from a given array consisting of only those elements from the given array which satisfy a condition set by the argument function.

```
function isEven(value) {
return value%2 = 0;
}
function func() {
var filtered = [11, 98, 31, 23, 944].filter(isEven);
document.write(filtered);
}
funco;
```

arrayBuffer.slice

The arrayBuffer.slice is a property in JavaScript which return the another arrayBuffer containing the contents of previous arrayBuffer from begin inclusive, to end, exclusive in bytes. ArrayBuffer is an object which is used to represent fixed-length binary data.
Difference between property and function in javascript.
Property in JavaScript is nothing but a value whereas method is a function.

arraybuffer.slice(begin[, end])Parameters Used:
begin : The slicing begins from Zero-based byte index.
end : The slicing ends at this index byte. The new ArrayBuffer will contain all the contents, if the end is found unspecified. It must be valid index range specified for the current array. If the new ArrayBuffer length is found negative then it is fixed with zero.
The property returns a new ArrayBuffer object.

```
var myBuffer = new ArrayBuffer(16);
var uint32View = new Uint32Array(myBuffer);
uint32View[1] = 30;
var sliced_buf = new Uint32Array(myBuffer.slice(4, 12));
console.log(sliced_buf[0]);
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

clearTimeout() & clearInterval() Method

clearTimeout()
The clearTimeout() function in javascript clears the timeout which has been set by setTimeout()function before that. setTimeout() function takes two parameters. First a function to be executed and second after how much time (in ms). setTimeout() executes the passed function after given time. The number id value returned by setTimeout() function is stored in a variable and it’s passed into the clearTimeout() function to clear the timer.

```
var t;
function color() (
if(document.getElementbyldebtnihstyle.color=='bluel {
document.getElementhyldebtn'hstyle.color&green';
}
else
{
document.getElementhyldrbtn'hstyle.color&lalue'n
}
function fun() {
t =setTimeout(color, 3000);
}
function stop() ( clearTimeout(t);
```

clearInterval()
The clearInterval() function in javascript clears the interval which has been set by setInterval() function before that. setInterval() function takes two parameters. First a function to be executed and second after how much time (in ms). setInterval() executes the passed function for the given time interval. The number id value returned by setInterval() function is stored in a variable and it's passed into the clearInterval() function to clear the interval.

```
var t; function color() (
if(document.getElementhyldebtnihstyle.color=='bluel {
document.getElementhyldebtn'hstyle.color&green';
}
else
{
document.getElementhyldebtn'hstyle.color=tlue'n
}
function fun() {
t =setlnterval(color, 3000);
}
function stop() { clearinterval(t); }
```

Date.now()

The Date.now() is an inbuilt function in JavaScript which returns the number of milliseconds elapsed since January 1, 1970, 00:00:00 UTC. Since now() is a static method of Date, it will always be used as Date.now().

```
var A = Date.now();
var d = Date(Date.now());
a = d.toString()
document.write("The current date is: " + a)
```

date.toLocaleString()

The date.toLocaleString() is used to convert a date and time to a string.
dateObj.toLocaleString(locales, options)
dateObj should be a valid Date object.
Parameters:
locales –This parameter is an array of locale strings that contain one or more language or locale tags. Note that it is an optional parameter. If you want to use a specific format of the language in your application then specify that language in the locales argument.
Options –It is also an optional parameter and contains properties that specify comparison options.Some properties are localeMatcher, timeZone, weekday, year, month, day, hour, minute, second etc.

```
var date = new Date(Date.UTC(2018, 5, 26, 7, 0, 0));
var options = { hourl2: false };
document.write(date.toLocaleString("enUS"));
document.write("");
document.write(date.toLocaleString("en-Us", options));
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

Q1:Introduction

There are certain features or mechanisms which makes a Language Object Oriented like:
Object
Classes
Encapsulation
Inheritance
Object– An Object is a unique entity which contains property and methods. For example "car" is a real life Object, which have some characteristics like color, type, model, horsepower and performs certain action like drive. The characteristics of an Object are called as Property, in Object Oriented Programming and the actions are called methods. An Object is an instance of a class. Objects are everywhere in JavaScript almost every element is an Object whether it is a function,arrays and string.

Classes-Classes are blueprint of an Object. A class can have many Object, because class is a template while Object are instances of the class or the concrete implementation.
JavaScript is a prototype based object oriented language, which means it doesn't have classes rather it define behaviors using constructor function and then reuse it using the prototype

```
class Vehicle {
constructor(name, maker, engine) {
this.name = name;
this.maker = maker;
this.engine = engine;
```

```
}
getDetails(){
return (The name of the bike is {this.name}.°)
}
}

// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');
console.log(bike1.name); // Hayabusa
console.log(bike2.maker); // Kawasaki
console.log(bikel.getDetails();
```

Encapsulation-The process of wrapping property and function within a single unit is known as encapsulation.

```
class person{
constructor(name,id){
this.name = name;
this.id = id;
}
add_Address(add){
this.add = add;
}
getDetails(){
console.log( Name is {this.name},Address is: {this.add});
}
}
let person1 = new person('Mukul',21);
person1.add_Address('Delhi');
person1.getDetails();
```

Most of the OOP languages provide access modifiers to restrict the scope of a variable, but their are no such access modifiers in JavaScript but their are certain way by which we can restrict the scope of variable within the Class/Object.

```
function person(fname,Iname){
let firstname = fname;
let lastname = Iname;

let getDetails_noaccess = function(){
return ('First name is: {firstname}
Last name is: {lastname});
}

this.getDetails_access = function(){
return ('First name is: {firstname},
Last name is: {lastname});
}
}

let person1 = new person('Mukul','Latiyan');
console.log(person1.firstname);
console.log(personi.getDetails_noaccess);
console.log(personi.getDetails_access(});
```

In the above example we try to access some property(person1.firstname) and functions(person1.getDetails_noaccess) but it returns undefine while their is a method which we can access from the person object(person1.getDetails_access()), by changing the way to define a function we can restrict its scope.

Inheritance-
It is a concept in which some property and methods of an Object is being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Object inherits Object i.e. certain features (property and methods)of one object can be reused by other Objects.

```
class person{
constructor(name){
this.name = name;
}
//method to return the string
toString(){
return ("Name of person: {this.name});
```

```
} }

class student extends person{
constructor(name,id){
//super keyword to for calling above class constructor
super(name);
this.id = id;
}
toString(){
return (“S{super.toString()},Student ID: S{this.id}’);
}
}
let student1 = new student(‘Mukul’,22);
console.log(student1.toString());
```

In the above example we define an Person Object with certain property and method and then we inherit the Person Object in the Student Object and use all the property and method of person Object as well define certain property and methods for Student.
Note: The Person and Student object both have same method i.e toString(), this is called as Method Overriding. Method Overriding allows method in a child class to have the same name and method signature as that of a parent class.
In the above code, super keyword is used to refer immediate parent class instance variable.

class traversy

```
class Person__ {
constructor(firstName, lastName, dob) {
this.firstName = firstName;
this.lastName = lastName;
this.dob = new Date(dob);
}

getBirthYear() {
return this.dob.getFullYear();
}

getFullName() {
return this.firstName this.lastName
}
}

const person1 = new Person__('John', 'Doe', '7-8-80');
console.log(person1.getBirthYear());
```


String substr

str.substr() function returns the specified number of characters from the specified index from the given string.
str.substr(start , length)
Arguments:
The first argument to the function start defines the starting index from where the substring is to be extracted from the base string. The second argument to the function length defines the number of characters to be extracted starting from the start in the given string. If the second argument to the function is undefined then all the characters from the start till the end of the length is extracted.
Return value: This function returns a string that is the part of the given string. If the length is 0 or negative value then it returns an empty string.

```
var str = 'It is a great day.'
print(str.substr(5,-7));
```

string.toString

The string.toString() is an inbuilt function in JavaScript which is used to return the given string itself.
string.toString()
Parameters: It does not accept any parameter.
Return Values: It returns a new string represent the given string object.

```
var a = new String("GeeksforGeeks");
document.write(a.toString());
```

String trim

str.trim() function is used to remove the white spaces from both the ends of the given string.

```
var str = " GeeksforGeeks";
```

```
var st = str.trim();
print(st);
```

str.trimLeft()str.trimLeft() function is used to remove the white spaces from the start of the given string. It does not affect the trailing white spaces.

```
var str = " GeeksforGeeks";
var st = str.trim();
print(st);
```

str.trimRight()str.trimRight() function is used to remove the white spaces from the end of the given string. It does not affect the white spaces at the start of the string.

```
var str = " GeeksforGeeks";
var st = str.trimRight();
print(st);
```

string.prototype.charCodeAt()

str.charCodeAt() function returns a Unicode character set code unit of the character present at the index in the string specified as the argument. The syntax of the function is as follows:

str.charCodeAt(index)

ArgumentsThe only argument to this function is the index of the character in the string whose Unicode is to be used. The range of the index is from 0 to length – 1.

Return value

This function returns the Unicode (ranging between 0 and 65535) of the character whose index is provided to the function as the argument. If the index provided is out of range the this function returns NaN.

```
var str = 'ephemeral';
print(str.charCodeAt(4));
```

Output:109

String.fromCharCode

String.fromCharCode() function is used to create a string from the given sequence of UTF-16 code units. The syntax of this function is as follows:

String.fromCharCode(n1, n2, ..., nX)

ArgumentsThe function takes the UTF-16 Unicode sequences as its argument. The number of arguments to this function depends upon the number of characters to be joined as a string. The range of the numbers is between 0 and 65535

The return value of this function is a string containing the characters whose UTF-16 codes were passed to the function as arguments.

```
print(String.fromCharCode(65, 66, 67));
```

Output:ABC

String startsWith()

The str.startsWith() method is used to check whether the given string start with the characters of the specified string or not.

str.startsWith(searchString , position)

Parameters: This method accepts two parameters as mentioned above and described below:

searchString: It is required parameter. It stores the string which needs to search.

start: It determines the position in the given string from where the searchString is to be searched. The default value is zero.

Return value This function returns the Boolean value true if the searchString is found else returns false.

```
var str = 'It is a great day.'
var value = str.startsWith('great',8);
print(value);
```

Output:true

String toUpperCase()

str.toUpperCase() function converts the entire string to Upper case. This function does not affect any of the special characters, digits and the alphabets that are already in upper case.

```
var str = 'It iS a Great Day.';
var string = str.toUpperCase();
print(string);
```

String toLowerCase()

str.toLowerCase() function converts the entire string to lower case. This function does not affect any of the special characters, digits and the alphabets that are already in the lower case.

```
var str = 'It iS a Great Day.';
var string = str.toLowerCase();
print(string);
```

Output:it is a great day.

string.repeat()

The string.repeat() is an inbuilt function in JavaScript which is used to build a new string containing a specified number of copies of the string on which this function has been called.

```
A = "gfg";
a = A.repeat(5);
document.write(a);
```

string.normalize()

The string.normalize() is an inbuilt function in javascript which is used to return a Unicode normalisation form of a given input string. If the given input is not a string, then at first it will be converted into string then this function will work.

string.normalize([form])

```
var a = "GeeksForGeeks";
b = a.normalize('NFC')
c = a.normalize('NFD')
d = a.normalize('NFKC')
e = a.normalize('NFKD')
console.log(b)
console.log(c)
console.log(d)
console.log(e)
```

Output:

```
GeeksForGeeks
GeeksForGeeks
GeeksForGeeks
GeeksForGeeks
```

string.replace()

The string.replace() is an inbuilt function in JavaScript which is used to replace a part of the given string with some another string or a regular expression. The original string will remain unchanged.

str.replace(A, B)

```
var re = /GeeksForGeeks/;
var string = 'GeeksForGeeks is a CS portal';
var newstring = string.replace(re, 'gfg');
document.write(newstring);
```

string.search()

The string.search() is an inbuilt function in JavaScript which is used to search for a match in between regular expressions and a given string object.

string.search(A)

Parameters: Here parameter “A” is a regular expression object.

Return Value: This function returns the index of the first match in between regular expression and the given string object and returns -1 if no match found.

String includes()

In JavaScript, includes() method determines whether a string contains the given characters within it or not.

This method returns true if the string contains the characters, otherwise, it returns false.

Note: The includes() method is case sensitive i.e, it will treat the Uppercase characters and Lowercase characters differently.

string.includes(searchvalue, start)

Parameters Used:

search value: It is the string in which the search will take place.

start: This is the position from where the search will be processed
(although this parameter is not necessary if this is not mentioned
the search will begin from the start of the string).

Returns either a Boolean true indicating the presence or it returns a false indicating the absence.

```
var str = "Welcome to GeeksforGeeks.";
var check = str.includes("o", 17);
document.getElementById("GFG").innerHTML = check;
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback

Q1: What is JavaScript(JS)?

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into otherwise static HTML pages.

Q2: What are the features of JavaScript?

- JavaScript is a lightweight, interpreted programming language.
- JavaScript is designed for creating network-centric applications.

jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

- JavaScript is complementary to and integrated with Java.
- JavaScript is complementary to and integrated with HTML.
- JavaScript is open and cross-platform.

Q3: What are the advantages of JavaScript?

- Less server interaction? You can validate user input before sending the page off to the server.
- Immediate feedback to the visitors? They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity? You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.

Q4: Why is javascript called Richer Interface?

You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Q5: How can we read the properties of an object in js?

Can write and read properties of an object using the dot(.) notation.

Q6: How many types of functions JS support?

A function in JavaScript can be either named or anonymous

Q7: Which built-in method calls a function for each element in the array?

forEach method calls a function for each element in the array.

Q8: Difference between "undefine" and "NULL" Keywords?

When you define a var but not assign any value. typeof(undefine)=> undefine Null- manually done. typeof(null)=> object

Q9: What is prototypal Inheritance?

Every object has a property called a prototype, where we can add methods to it and when you create another object from these the newly created object will automatically inherit its parent's property.

Q10: What is SetTimeout()?

When you setTimeout it becomes asynchronous and it has to wait on the stack to get everything got finished

Q11: What is closure and how do you use it?

When a function returns the other function the returning function will hold its environment and this is known as closure.

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

Q31: Variables and Datatypes in JavaScript

There are majorly two types of languages. First, one is Statically typed language where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types.Example: C, C++, Java.

int x = 5 // variable x is of type int and it will not store any other type.
string y = 'abc' // type string and will only accept string values

Dynamically typed languages:
These languages can receive different data types over time. For example- Ruby, Python, JavaScript etc.
var x = 5; // can store an integer
var name = 'string'; // can also store a string.
JavaScript is dynamically typed (also called loosely typed) scripting language. That is, in javascript variables can receive different data types over time. Datatypes are basically typed of data that can be used and manipulated in a program.
seven data types:
Out of which six data types are Primitive(predefined).
Numbers: 5, 6.5, 7 etc.
String: "Hello GeeksforGeeks" etc.
Boolean: Represent a logical entity and can have two values: true or false.
Null: This type has only one value : null.
Undefined: A variable that has not been assigned a value is undefined.
Object: It is the most important data-type and forms the building blocks for modern JavaScript.
Variables in JavaScript are containers which hold reusable data. It is the basic unit of storage in a program.

We can initialize the variables either at the time of declaration or also later when we want to use them. Below are some examples of declaring and initializing variables in JavaScript:

```
var name;
var name, title, num;
var name = "Harsh";
name = "Rakesh";
```

Javascript is also known as untyped language. This means, that once a variable is created in javascript using the keyword var, we can store any type of value in this variable supported by javascript. Below is the example for this:

```
var num = 5;
num = "GeeksforGeeks";
```

Q2: let and const

```
let x;
let name = 'Mukul';
let a=1,b=2,c=3;
let a = 3;
a = 4; // works same as var.
```

Const is another variable type assigned to data whose value cannot and will not change throught the script.

```
const name = 'Mukul';
name = 'Mayank'; // will give Assignment to constant variable error.
```

Q3: Variable Scope in Javascript

Scope of a variable is the part of the program from where the variable may directly be accessible. In JavaScript, there are two types of scopes:

Global Scope – Scope outside the outermost function attached to Window.
Local Scope – Inside the function being executed.
Let's look at the code below. We have a global variable defined in first line in global scope. Then we have a local variable defined inside the function fun().

```
let globalVar = "This is a global variable";
function fun() (
let localVar = "This is a local variable";
console.log(globalVar);
console.log(localVar); }
funn:
```

o/p: This is a global variable
This is a local variable
This shows that inside the function we have access to both global variables (declared outside the function) and local variables (declared inside the function).Let's move the console.log statements outside the function and put them just after calling the function.

```
let globalVar = "This is a global variable";
function fun() (
let localVar = "This is a local variable";
}
fun();
console.log(globalVar);
console.log(localVar);
```

o/p: This is a global variable
RefferenceError: localVar is not defined

This is because now the console.log statements are present in global scope where they have access to global variables but cannot access the local variables. Also, any variable defined in a function with the same name as a global variable takes precedence over the global variable,

Q32: Objects

Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript's primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.
An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.

Creating Objects
1.literal syntax Object literal syntax uses the notation to initialize an object an its methods/properties directly.

```
var obj = {
memberl : valuel,
member2 : value2,
};
```

These members can be anything – strings, numbers, functions, arrays or even other objects. An object like this is referred to as an object literal.

2.Object Constructor :
The Object constructor creates an object wrapper for the given value. This, used in conjunction with the "new" keyword allows us to initialize new objects.

```
const school = new Object();
school.name = 'Vivekanada school';
school.location = 'Delhi';
school.established = 1971;
school.displaylnfo = function(){
console.log( was established in at ');
}
school.displayinfo();
```

The two methods mentioned above are not well suited to programs that require the creation of multiple objects of the same kind, as it would involve repeatedly writing the above lines of code for each such object. To deal with this problem, we can make use of two other methods of object creation in JavaScript that reduces this burden significantly,

3.Constructors
Constructors in JavaScript, like in most other OOP languages, provides a template for creation of objects. In other words, it defines a set of properties and methods that would be common to all objects initialized using the constructor.

```
function Vehicle(name, maker) {
this.name = name;
this.maker = maker;
}
let cart = new Vehicle('Fiesta', 'Ford');
let cart = new Vehicle('Santa Fe', 'Hyundai')
console.log(carl.name); // Output: Fiesta console.log(car2.name);
```

Notice the usage of the "new" keyword before the function Vehicle. Using the "new" keyword in this manner before any function turns it into a constructor. What the "new Vehicle()" actually does is :

It creates a new object and sets the constructor property of the object to schools (It is important to note that this property is a special default property that is not enumerable and cannot be changed by setting a "constructor: someFunction" property manually).

Then, it sets up the object to work with the Vehicle function's prototype object (Each function in JavaScript gets a prototype object, which is initially just an empty object but can be modified.The object, when instantiated inherits all properties from its constructor's prototype object).

Then calls Vehicle() in the context of the new object, which means that when the "this" keyword is encountered in the constructor(vehicle()), it refers to the new object that was created in the first step.

Once this is finished, the newly created object is returned to car1 and car2

Inside classes, there can be special methods named constructor().

```
class people {
constructor() {
this.name = "Adam";
}
}
let personl = new people(); // Output : Adam console.log(personl.name);
```

Having more than one function in a class with the name of constructor() results in an error.

4.Prototypes
Accessing Object Members
Object members(properties or methods) can be accessed using the
1.dot notation
(objectName.memberName)

```
let school = (
name : "Vivekanada",
location : "Delhi",
established :1971, 20 :1000,
displayinfo : function() (
console.logn{school.name}
was established in at $(school.location}');
}
}
console.log(school.name);
console.log(school.established);
```

2.Bracket Notation :
objectName["memberName"]

```
let school = (
name : "Vivekanada School",
location : "Delhi",
established :1995, 20 : 1000,
displayinfo : function() {
document.write
was established in at');
} }
// Output : Vivekanada School console.log(schoolrnamel);
// Output: 1000 console.log(school[201);
```

Unlike the dot notation, the bracket keyword works with any string combination, including, but not limited to multi-word strings. For example:
somePerson.first name // invalid
somePerson["first name"] // valid
Unlike the dot notation, the bracket notation can also contain names which are results of any expressions variables whose values are computed at run-time. For instance :
let key = "first name" somePerson[key] = "Name Surname"
Similar operations are not possible while using the dot notation.
Iterating over all keys of an object

```
let person = (
gender : "male"
var personl = Object.create(person);
personl.name = "Adam";
personl.age =45;
personl.nationality = "Australian";
for (let key in personl) (
// Output : name, age, nationality // and gender
console.log(key);
```

Q33: There are 3 ways to create objects:

1.Using the Object() Constructor

```
var obj = new Object();
obj.firstName = "Geeks";
obj.middleName = "for";
obj.lastName = "Geeks";
obj.isTeaching = "Javascipt";
obj.greet = function() {
console.log("Hi");
```

2.Using Object literal

```
var obj = {} //obj is an empty object
obj.firstName = "Geeks";
obj.middleName = "for";
```

```
obj.IastName = "Geeks";
obj.isTeaching = "Javascipt";
obj.greet = function() {
console.Iog("Hi");
}
```

3. Directly specifying the values

```
var obj ={
obj.firstName = "Geeks";
obj.middleName = "for";
obj.lastName = "Geeks";
obj.isTeaching = "Javascipt";
obj.greet = function() {
console.log("Hi");
}
```

Coercion: What we call typecasting in C, C++, Java, it is called coercion in JavaScript.
Coercion is of two types:
Explicit Coercion
Explicit coercion is the process by which we explicitly define a variable to a data type. var x = 42;
var explicit = String(x); // explicit is set to "42"

Implicit Coercion
Implicit Coercion is the process by which the interpreter dynamically type casts the variable under certain conditions.
var x = 42; var implicit = x + " "; // interpreter automatically sets implicit as "42"

Q34: Scope

Variable lifetime: – The variable lifetime is from where they are declared until their function ends. If no function is defined then scope of the variable is global.
Hoisting: – Function definitions are hoisted but not variable declarations. This means that when a function is declared, it is usable from anywhere within your code.
hoisting

```
greet();
function greet(){
console.log("Hi");
```

The above function is executed successfully when called. But if we call the function first and then define it, then also it is successfully executed. This is called function hoisting.
However variable initialization are not hoisted.

```
x(); // calling x
var x = function(){
console.log("Hi");
i /* the above is an error as x is not a function because due to the '=' operator,
x is a variable initialization to which a function is assigned.
```

The above example is the part which proves that variable hoisting is not possible in JavaScript.
But look at the next example which is another type of hoisting defined in JavaScript.

```
console.log(x);
var x = 42;
```

Output ==> undefined

Q35: if-else Statement in JavaScript

JavaScript's conditional statements:
if
if-else
nested-if
if-else-if
These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

```
var i = 10; if (i > 15)

console.log("10 is less than 15");

console.log("I am Not in if");


var i = 10; if (i > 15)
```

```
console.log("10 is less than 15");

else

console.log("I am Not in if");
```

nested-if: A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, JavaScript allows us to nest if statements within if statements.

```
var i= 10;
if (i == 10) {
// First if statement if (i< 15)
document.write("i is smaller than 15"); // Nested - if statement Will only be executed if statement above itis true
if (i< 12)
document.write("i is smaller than 12 too");
else
document.write("i is greater than 15");
```

if-else-if ladder: Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.
var i = 20; if (i == 10) document.wrte("i is 10"); else if (i == 15) document.wrte("i is 15"); else if (i == 20) document.wrte("i is 20"); else document.wrte("i is not present");

Q35: Switch Case in JavaScript

```
vari=9;
switch (i)
{
case 0:
document.write("i is zero.");
break;
case 1:
document.write("i is one.");
break;
case 2:
document.write("i is two.");
break;
default:
document.write("i is greater than 2.");
}
```

Q36: Loops in JavaScript

```
var languages = {
first :"C",
second : "Java",
third : "Python", fourth : "PHP",
fifth : "JavaScript"
}; // iterate through every property of the object languages and print all of them using for..in loops
for (itr in languages) {
document.write(languages[itr] + "
");
}
```

do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

```
var x=21;
do {
// The line while be printer even if the condition is false
document.write("Value of x:" +x + "
");
x++;
} while (x < 20);
```

Q36: this Identifier

Global Scope: Whenever ‘this’ keyword is used in the global context i.e. not as a member of function or object declaration, it always refers to the Global object.
var a = "GFG";

```
console.log(a);
this.a = "GeeksforGeeks";
console.log(a);
```

Functional Scope :
If a function has a ‘this’ reference inside it, it can be said that the this refers to an object, not the function itself
To determine which object the ‘this’ points to depends on how the function was called in the first place.

```
function myFunc() {
console.log( this.a };
i
var a="Global";
// Owner of the function.
var myObj1 = {
a: "“myObj1",
myFunc: myFunc
I;
// Object other than the owner.
var myObj2 = {
a: "“myObj2"
I;
// Call the funtion in Global Scope.
myFunc()};
// Call the funtion from the reference of owner.
myObj1.myFunc();
// Call the function from the reference
// of object other than the owner.
myFunc.call( myObj2 )};
// Create anew undefined object.
new myFunc();
```

Seeing the above example, we can see four different ways we can determine what this points to. There are four rules for how this gets set, let us explain these four for ourselves.
In the first call, myFunc() ends up setting this to the Global object in non-strict mode. Whereas, in strict mode this would be undefined and JavaScript will throw an error in accessing the property.
myObj1.myFunc() sets this to the myObj1 object, here myObj1 is the owner of the Function myFunc and we are calling the function with the reference of the object itself, Thus in such cases this will refer to the owner object.
myFunc.call(myObj2) sets this to the myObj2 object. This proves that this doesn’t always point to the owner object, it rather points to the object under whose scope the function was called.
new myFunc() sets this to a brand new empty object thus we get undefined in the console log.
Note: We can determine whom ‘this’ refers by following this simple technique. Whenever a function containing ‘this’ is called, we should look at the immediate left of the parentheses pair “()”. If on the left side of the parentheses there is a reference, then “this” refers to the object it belongs to, otherwise, it refers to the global object.

Inside an Event Handler
‘this’ inside of an event handler always refers to the element it was triggered on.

```
function clickedMe() {
console.log(this.innerHTML);
}
clickedMe(); // undefined because global object.
var myElem = document.getElementByld({‘clickMe');
myElem.onclick = clickedMe;
myElem.onclick(); // Welcome to GFG!
```

Q37: Functions in JavaScript

A function is a set of statements that take inputs, do some specific computation and then returns the result to the user.
The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call that function.

```
function welcomeMsg(name) {
document.write("Hello " + name +" welcome to GeeksforGeeks");
}
// creating a variable
var nameVal = "Admin";
// calling the function
welcomeMsg(nameVal);
```

Return Statement: There are some situations when we want to return some values from a function after performing some operations. In such cases, we can make use of the return statement in JavaScript. This is an optional statement and most of the

times the last statement in a JavaScript function.

Q38: Closure

```
/*1*/
function foo()
f2*f
{*3*/
varb=1; [*4*/
function inner(){
{*5*/
return b;
f6*/
} {*7*/
return n;
f*3*/
} /*9*/ var get_func_inner = foo();
/*10*/ — console.log(get_func_inner()});
/*11*/ — console.log(get_func_inner()});
/*12*/ — console.log(get_func_inner())
```

:Interesting thing to note here is from line number 9 to line number 12 . At line number 9 we are done with the execution of function foo() but we can access the variable b which is defined in function foo() through function inner() i.e in line number 10, 11, 12. and as desired it logs the value of b. This is closure in action that is inner function can have access to the outer function variables as well as all the global variables.
In order to see the variable and function bound within closure we can write as:
console.dir(get_func_inner);

```
function foo{outer_arg) {
function inner{inner_arg) {
return outer_arg + inner_arg;
}
return inner;
}
var get_func_inner = foo(5);
console.log(get_func_inner(4});
console.log(get_func_inner(3));
```

Note even when we are done with the execution of foo(5) we can access the outer_arg variable from the inner function. And on execution of inner function produce the summation of outer_arg and inner_arg as desired.
Now let's see an example of closure within a loop.
In this example we would to store a anonymous function at every index of an array.

```
function outer() {
var arr =[];
var i;
for (i =0; 1 <4; i++) {
// storing anonymus function
arr[i] = function () { return i;
} }
// returning the array.
return arr;
}
var get_arr = outer();
console.log(get_arr{0]({});
console.log(get_arr{1]({});
console.log(get_arr{2]({});
console.log(get_arr{3]({});
```

In the above code we have created four closure which point to the variable i which is local variable to the function outer. Closure don't remember the value of the variable it only points to the variable or stores the reference of the variable and hence, returns the current value. In the above code when we try to update the value of it gets reflected to all because the closure stores the reference.
Lets see an correct way to write the above code so as to get different values of i at different index.

```
function outer() {
function create_Closure(val) {
return function() {
return val;
}
}
```

```
var arr =[];
var i;
for (i =0; 1 <4; i++) {
arr[i] = create_Closure(i);
}
return arr;
}
var get_arr = outer();
console.log(get_arr{0]({});
console.log(get_arr{1]({});
console.log(get_arr{2]({});
console.log(get_arr{3]({});
```

In the above code we are updating the argument of the function create_Closure with every call. Hence, we get different values of i at different index.
Closure is one of the most important yet most misunderstood concepts in Javascript. The closure is a methodology in which a child function can keep the environment of its parent scope even after the parent function has already been executed,

Q39: Modules

Here, the Rectangle() function serves as an outer scope that contains the variables required i.e. length, width, as well as the functions create(), getArea(), and getPerimeter(). All these together are the private details of this Rectangle module that cannot be accessed/modified from the outside. On the other hand, the publicAPI as the name suggests is an object that consists of three functional members and is returned when the Rectangle function execution is complete. Using the API methods we can create and get the value of area and perimeter of the rectangle.

Q40: Difference between var and let

var and let are both used for function declaration in javascript but the difference between them is that var is function scoped and let is block scoped.
Input:console.log(x);
var x=5;
console.log(x);
Output:
undefined
5
Input:console.log(x);
let x=5;
console.log(x);
Output:
Error
Example: var x = 5;
document.write(x, "\n");
let y = 10;
document.write(y, "\n");
document.write(z, "\n");
var z = 2;
document.write(a);
let a = 3;
Example2:
In the following code, clicking start will call a function that changes the color of the two headings every 0.5sec. The color of first heading is stored in a var and the second one is declared by using let.
Both of them are then accessed outside the function block. Var will work but the variable declared using let will show an error because let is block scoped.

Q41: Window innerWidth and innerHeight Properties

The innerWidth property in JavaScript returns the width and innerHeight property returns the height of window content area.
window.innerWidth
window.innerHeight

Q42: Callbacks**modules**

```
{
length =1;
width =w;
}
function getArea() {
return (length * width);
}
function getPerimeter() {
```

```
return (2 * (length + width));
}
var publicAPI = {
create : create,
getArea : getArea,
getPerimeter : getPerimeter
hi
return publicAPI;
}
var myRect = Rectangle();
myRect.create(5, 4); console.log("Area: " + myRect.getArea(});
console.log("Perimeter: " + myRect.getPerimeter());
```

varLet

callBach

```
function add(a, b , callback){
document.write( The sum of and S{b} is .” +"
");
callback();
}
// disp() function is called just after the ending of add({) function function disp(){
document.write('This must be printed after addition’);
}
// Calling add{) function add(5,6,disp);
```

Callbacks are a great way to handle something after something else has been completed(function execution). If we want to execute a function right after the return of some other function, then callbacks can be used.
avaScript functions have the type of Objects. So, much like any other objects (String, Arrays etc.), They can be passed as an argument to any other function while calling.

```
let person = (
gender : "male"
var personl = Object.create(person);
personl.name = "Adam";
personl.age =45;
personl.nationality = "Australian";
for (let key in personl) (
// Output : name, age, nationality // and gender
console.log(key);
```

Here are the two functions – add(a, b, callback) and disp(). Here add() is called with the disp() function i.e. passed in as the third argument to the add function along with two numbers.
As a result, the add() is invoked with 1, 2 and the disp() which is the callback. The add() prints the addition of the two numbers and as soon as that is done, the callback function is fired! Consequently, we see whatever is inside the disp() as the output below the addition output.

Q43: Type Conversion

JavaScript is loosely typed language and most of the time operators automatically convert a value to the right type but there are also cases when we need to explicitly do type conversions.
While JavaScript provides numerous ways to convert data from one type to another but there are two most common data conversions :
Converting Values to String
Converting Values to Numbers
Converting Values to Strings:
String() or toString() function can be used in JavaScript to convert a value to a string.
Input:var v = 1555;
var s = String(v);
Output:
now s contains "1555".

Code #2:
Below code going to convert the number to string, boolean value to string and dates to string.

```
var v = 123;
var d = new Date('1995-12-17T03:24:00');
document.write(" String(v) = " + String(v));
document.write(" String(v + 11) = " + String(v + 11));
document.write(" String( 10 + 10) = " + String(10 + 10));
document.write(" String(false) = " + String(false));
```

```
document.write(" String(d) = " + String(d));
```

Converting Values to Numbers: We can use Number() function in JavaScript to convert a value to a Number. It can convert any numerical text and boolean value to a Number. In case of strings of non-numbers it will convert it to a NaN(Not a Number).

```
var v = "144";
var d = new Date('1995-12-17T03:24:00');
document.write(" Number(v) = " + Number(v));
document.write(" Number(false) = " + Number(false));
document.write(" Number(true) = " + Number(true));
document.write(" Number(d) = " + Number(d));
```

code #4:

If the string is non-number then it converts it to NaN and strings of white spaces or empty strings will convert to 0.

```
var v = "";
var d = " ";
var s = "GeeksforGeeks";
document.write(" Number(v) = " + Number(v));
document.write(" Number(d) = " + Number(d));
document.write(" Number(s) = " + Number(s));
```

o/p Number(v) = 0
Number(d) = 0
Number(s) = NaN

Q44: Strict mode

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a “strict” operating context. This strict context prevents certain actions from being taken and throws more exceptions. The statement “use strict”; instructs the browser to use the Strict mode, which is a reduced and safer feature set of JavaScript.

Benifits of using ‘use strict’
Strict mode eliminates some JavaScript silent errors by changing them to throw errors.
Strict mode fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that’s not strict mode.
Strict mode prohibits some syntax likely to be defined in future versions of ECMAScript.
It prevents, or throws errors, when relatively “unsafe” actions are taken (such as gaining access to the global object).
It disables features that are confusing or poorly thought out.
Strict mode makes it easier to write “secure” JavaScript.

How to use strict mode
Strict mode can be used in two ways – used in global scope for the entire script and can be applied to individual functions.
Strict mode doesn’t work with block statements enclosed in braces.
Using Strict mode for the entire script
To invoke strict mode for an entire script, put the exact statement “use strict”; (or ‘use strict’;) before any other statements.

```
'use strict';
let v = "strict mode script!";


function strict() {
// Function-level strict mode syntax
‘use strict’;
function nested() {
return Javascript on GeeksforGeeks’; }
return "strict mode function! "+nested();
function notStrict() {
return "non strict function";
}
```

Examples of using Strict mode

- In normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable
- Using strict mode, don’t allow to use a variable without declaring it

Using a variable, without declaring it, is not allowed:

```
‘use strict’;
x = 3.14;
```

Using an object, without declaring it, is not allowed:
Octal numeric literals are not allowed

```
‘use strict’;
let x = 010;
```

Escape characters are not allowed

```
‘use strict’;
```

let eval = 3.14;
he string “arguments” cannot be used as a variable
‘use strict’;
let arguments = 3.14;

- In function calls like f(), the this value was the global object. In strict mode, it is now undefined

Q45: Error and Exceptional Handling With Examples

There are three types of error in programming which are discusses below :
Syntax error
Logical error
Runtime error

Syntax error: According to computer science, a syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language or it is also the compile-time error if the syntax is not correct then it will give an error message.
As the syntax is not correct of the JavaScript it will affect only the thread that is under this JavaScript and the rest of the code in other threads gets executed as nothing in them depends on the code containing the error.

Logical error: It is the most difficult error to be traced as it is the error on the logical part of the coding or logical error is a bug in a program that causes it to operate incorrectly and terminate abnormally (or crash).

Runtime Error: A runtime error is an error that occurs during the running of the program, also known as the exceptions. In the example that is given below the syntax is correct, but at runtime, it is trying to call a method that does not exist.
window.printme();
As in runtime error, there are exceptions and this exception is correct by the help of the try and catch method

try ___ catch method: JavaScript uses the try catch and finally to handle the exception and it also used the throw operator to handle the exception. try have the main code to run and in the catch, give the exception statement all the things that are related to the exception.

```
function First() {
var a= 123;
var b = 145;
varc=at+b;
alert("Value of a:" +a);
alert("Value of b: "+b );
alert("Sum of a and b: " +c);
}
```

In this example, use the finally method which will always execute unconditionally after the try/catch.

```
function First() {
var a= 123;
var b = 145;
varc=at+b;
try {
alert("Value of a:" +a);
alert("Value of b: "+b );
alert("Sum of a and b: " +c);
}
catch (e){
alert("Error: " + e.description );
}
finally {
alert("Finally block will always execute!" );
}
}
```

Q46: Arrow functions

, provides a concise way to write functions in JavaScript.
it offers is the fact that it does not bind its own this. In other words, the context inside arrow functions is lexically or statically defined.
What do we mean by that?
Unlike other functions, the value of this inside arrow functions is not dependent on how they are invoked or how they are defined.It depends only on its enclosing context.

```
let People = function(person, age) {
this.person = person;
this.age = age;
```

```
this.info = function() {
// logs People
document.write(this);
setTimeout(function() {
// here this!=People
document.write(this.person +" is " + this.age + " years old");
}, 3000);
}
}
let person1 = new People(‘John', 21);
// logs : undefined is undefined years old after 3 seconds
person1.info()};
```

o/p:[object Object]
undefined is undefined years old
The reason that we get undefined outputs instead of the proper info as output happens because the function() defined as the callback for setTimeout has a normal function invocation and as we know, this means that its context is set to the global context or in other words the value of this is set to the window object.
This happens because every regular, non-arrow function defines its own this or context depending on their invocation.The context of the enclosing objects/function does not affect this tendency to automatically define their own context.
How do we solve this?
One obvious solution that comes to mind is, what if the function did not define its own context?What if it inherited the context from the info(), because that would mean this function() gets the this as was defined in info()
Well, that is exactly what arrow functions do.They retain the value of this from their enclosing context.
That is, in the above example, if the function defined as callback for setTimeout() were an arrow function it would inherit the value of this from it’s enclosing context – info()

```
let People = function(person, age) {
this.person = person;
this.age = age;
this.info = function() {
// logs People
document.write(this);
setTimeout(() => {
// arrow function to make lexical "this" binding
// here this=People."this" has been inherited
document.write(this.person +" is " + this.age
+" years old");
}, 3000);
}
}
let person1 = new People(‘John', 21);
// logs : John is 21 years old after 3 seconds
person1.info()};
```

o/p: [object Object]
John is 21 years old
Thus, regardless of whether the arrow function was called using function invocation or method invocation, it retains the]value of this from its enclosing context.In other words, an arrow function’s this value is the same as it was immediately outside it.
If used outside any enclosing function, an arrow function inherits the global context, thereby setting the value of this to the global object.
this in separated methods: When a method from any object is separated from it, or stored in a variable, eg : let separated = People.info, it loses the reference to its calling object.
Notice the lack of opening and closing parentheses after info. This indicates that we are not calling the method immediately.

```
let People = function(person, age) {
this.person = person;
this.age = age;
this.info = function() {
// logs People
document.write(this +");
// here this=People
document.write(this.person +" is " + this.age + "years old" +'
');
}
}
let person1 =new People(‘John', 21);
// logs : John is 21 years old
person1.info()};
```

```
// separating the method info() from its
// object by storing it in a variable
let separated = person1.info;
// logs : undefined is undefined years old
separated();
```

o/p:[object Object] John is 21 years old
[object Window] undefined is undefined years old
Once we separate the info() from the person1 object by storing it in separated, we lose all references to the person1 object. We can no longer access the parent object by using this inside the separated method because the context of the separated method gets reset to the global context.
Thus, when we call separated() we see that this is now set to the global window object.
How do we solve this?
One way is to bind the value of an object with the method when storing the method in separated. This ensures that all references to this refers to this bound object even in the separated method.
This can be done using bind() like so:

```
let People = function(person, age) {
this.person = person;
this.age = age;
this.info = function() {
// logs People
document.write(this +");
// here this=People
document.write(this.person +" is " + this.age + "years old" +'
');
}
}
let person1 =new People(‘John', 21);
person1.info()};
let separated = person1.info.bind(person1); {*
the bind({person1) statement ensures that "this" always
refers to personi inside the bound method- info()
*f
// logs : undefined is undefined years old
separated();
```

o/p:[object Object] John is 21 years old
[object Object] John is 21 years old
Note: We could have used any object to bind() to the method info(), instead of, code.person1, and the outputs would have changed accordingly.It is not mandatory to bind a method to it’s parent object.
If you have come this far you would have noticed that we mentioned bind() a couple of times.**bind, call and apply**

bind(), call() and apply() are all used to modify the context of a function. All three of these help explicitly specify what the value of this should be inside a function. But, there are certain differences in how each of them work.
bind()
bind() allows us to explicitly define what value this will have inside a function by binding an object to that function.
The bound object serves as the context(this value) for the function that it was bound to.
To use it, we need primarily two things – An object to bind to a function and a function that this object is to be bound to.

Syntex:boundfunction = someFunction.bind(someObject, additionalParams)
The first argument used inside the bind() directive serves as the this value and the arguments that follow it are optional and serve as the arguments for the bound function.

o/p Orange is orange
Banana is Yellow

Notice that we call displayInfo() on fruit1 using method invocation : fruit1.displayInfo and therefore might expect it to have the context of fruit1. However this does not seem to be the case as “Banana is Yellow” gets logged instead of “Orange is orange”.
This happens because we explicitly bind the value of this inside displayInfo() using the bind() command to bindingObj.
As we can see, binding an object explicitly to a function overrides its normal context rules and forcefully sets all this values to the bound object inside it.
Often, when passing functions around, it loses its context.For instance:

o/p[object Window] undefined
To prevent resetting of context inside the function passed as callback, we explicitly bind the value of this to be the same as it was inside passAround(), i.e:to the binding object :

```
class people {
constructor() {
this.name = “Adam”;
I
I
let person1 = new people();
// Output : Adam
console.log(personi.name);
```

o/p[object Object] John

call() and apply()
call() and apply() perform a task similar to bind by explicitly specifying what value this should store inside a function. However, one major difference between them and bind() is that call() and apply() immediately calls the function, as opposed to simply preparing a copy of the function with a bound this value for future use.
The syntax:

call:function.call(thisValue, arg1, arg2, ...)
apply:function.apply(thisValue, [arg1, arg2, ...])
callApply The first argument in both cases is the value of this that we wish for the called function to have.
Essentially, the only difference between these two methods is the fact that in apply , the second argument is an array object of arguments while in call, all arguments are sent in a comma separated format.

o/p[object Object] Banana is yummy
[object Object] Orange is sour

this with event listeners Inside functions used as callbacks for event listeners, this holds the value of the element that fired the event.

```
let elem = document.querySelector(‘btn')
elem.addEventListener(‘click’, function() {
// ogs: btn
document.write(this)
1)
```

If the callback function used here is an arrow function and our event listener is nested inside another method, this would refer to the context of the outer nesting method, and we can no longer access the element that the event listener is added to using this, like we had in the previous code example.
Fortunately, this can be easily resolved by using the currentTarget method on the element like so:

```
function outerfunc(elem){
this.clickHandler = function() {
elem.addEventListener(‘click', (e) => {
// logs

'<'button />element
document.write(e.currentTarget);
// this has the value of outerfunc
this.displayInfo();
)
}

this.displayInfo = function() {
document.write(' Correctly identified!");
}
}

let button = document.body.querySelector(‘button’);
let test = new outerfunc(button);
test.clickHandler()
```

o/p:[object HTMLButtonElement] Correctly identified!
As we can see, using the currentTarget allows us to access the element that the event listener is added on while this allows us to access the context of the enclosing function – i.e: this allows us to successfully call the displayInfo() method.

arrayjs
date
oops
advancedJSGeeksForGeeks

Q1. Name the different types of JavaScript data?

- 1. String
- 2. Function

- 3. Boolean
- 4. Object
- 5. Number
- 6. Undefined

int x =5
string y = sabe

Q2. What is the definition of global variables? In what way, these variables are declared?

Global variable is a special kind of variable in JavaScript. This kind of variable is something that is easy to use and also available across the entire length of the JavaScript code. Mainly, the var keyboard is used whether to declare a global or local variable.

Q3. Name the problems that are associated with the use of global variables?

Even though global variables are easy to use, these have some shortfalls. While using this type of variables, the problem of clashing of the variable names of different global and local scope occurs. The code that is often relied on the global variable also gets difficult to be tested and debugged.

Q4. Name the different type of groups of data types that are used in JavaScript and define them?

1. Reference type - These are complex types of data which can mainly include dates and strings.
2. Primitive type - These are types of data includes number data.

Q5. Mention the advantages of using JavaScript programming language?

The different kinds of advantages of using JavaScript are -
1. JavaScript provides the users to get immediate feedback. They can check the data which they entered without waiting for the page to be reloaded.
2. The server interacting is lesser and the load on the server is also much lesser. JavaScript helps in saving the traffic on the server.
3. Users can use various items to provide a rich interface to their own sites like they can use sliders and various drag-and-drop components.
4. JavaScript also increased interactivity between various users.

Q6. JavaScript is a case sensitive language - Explain?

JavaScript programming language is case sensitive. This basically means those different types of variables, language keyboards, functions and various identities, all these should be consistently used with the help of capitalized letters.

Q7. Mention the different types of functions are supported by JavaScript? Define each of them?

There are two types of functions which are supported by JavaScript. They are -
1. Anonymous function - This type of function generally has no name and this is the difference between it and a normal function.
2. Named function - On the other hand, this function is something which is named properly and specifically.

Q8. What are the different scopes of a variable? Define local variable?

The scope of a variable is generally defined as the region which acts as the place of the program. Local variables are those types of variable which are visible across the specifically defined function. The parameter used is also local to the function.

Q9. What is the meaning of the word 'callback'?

Callback is a typical function of the JavaScript which can be passed as an option or argument of JavaScript. Sometimes, callbacks can also be termed as simple events. Users are given calls to react to different kind of triggered situations.

Q10. Define the mechanism of the typeof operator?

The typeof operator is basically just the unary operator which is used before the single operand. The value indicates the operand's data type. The type includes Boolean, number, and string.

Q11. How to create an object?

An object in JavaScript can be created using two was:
New Key word: To create a student object from the above student class we can call the Student function using new keyword.
var student1 = new Student('santosh',2)
Anonymous Object:
Anonymous objects can be created using pair of curls' braces containing property name and value pairs.

Var rose = ('color': 'red')

Q12. What are the meanings of undefined variables and undeclared variables?

Undefined variables are the types of variables which gets declared in the program. However, the value of undefined variables is not provided in JavaScript programming language. The value gets returned if one tries to read the undefined value. Undeclared variables are the types of variables which do not even exist in a program and are also not declared properly. The value also can't be read properly when tried to be read and hence a runtime error is shown.

Q13. Define the various types of errors which occur in JavaScript programming language?

Run time errors - This type of error is the outcome of the misuse of the use of command within the HTML language.
Load time errors - Load time errors are basically syntax errors which are improper and arise when a web page is tried to be loaded. This type of error is generated dynamically.
Logical errors - A function often has a different operation and this type of error arises when the logic of the function is badly performed.

Q14. What is constructor property?

Constructor property of an object maintains a reference to its creator function.
ex. Creating a student object and calling the constructor property

Q15.What Is Scope In JavaScript?

The scope determines the accessibility of variables, objects, and functions in particular part of your code. In JavaScript, the scope is of two types.

- Global Scope
- Local Scope

Q16. How to call other class methods?

Using call () and apply () method we can use methods from different context to the current context. It is really helpful in reusability of code and context binding.
call (): It is used to calls a function with a given this value and arguments provided individually.
apply (): It is used to call a function with a given this value and arguments provided as an array.

Q17. How many types of objects are there in JavaScript? Define them?

There are two types of objects in JavaScript -
1. Date object - This type of object is built within the JavaScript programming language. The date objects are created with the help of new date. It can be operated with the help of a bunch of methods once it is created. The methods allow the inclusion of the year, the month, day, hour and even minutes, seconds and millisecond of the date object. These are set with the help of local standard time of universal time.
2. Number object - This type of number object also includes the dates as it solely numerical dates are represented by it like integers and fractions. The literals of the numbers get converted to the number class automatically.

Q18. Explain method overriding with an Example?

We can override any inbuilt method of JavaScript by declaring its definition again. The existing definition is accessible ot override by the Prototype property. Consider the below example, Split () class. But we have overridden its definition using its prototype property.
Below screen shows the inbuilt behavior of split () method. It has divided the string into an array of element.

Q19. What is the meaning of continuing and break statements?

Continue statements are the ones which continue from the next loop with the next set of statements. On the other hand, break statements are the ones that start from the current existing loop.

Q21. How to inherit form a class?

Inheritance can be achieved in JavaScript using Prototype property.
tep1: Child class prototype should point to parent class object. .prototype = new ();
Step2: Reset the child class prototype constructor to point self. .prototype . constructor = student;

Q22. How are JavaScript and ECMA Script related?

ECMA Script are like rules and guideline while Javascript is a scripting language used for web development.

Q23: What is the significance, and what are the benefits, of including 'use strict' at the beginning of a JavaScript source file?

The short and most important answer here is that use strict is a way to voluntarily enforce stricter parsing and error handling on your JavaScript code at runtime. Code errors that would otherwise have been ignored or would have failed silently will now generate errors or throw exceptions. In general, it is a good practice.
Some of the key benefits of strict mode include:
Makes debugging easier. Code errors that would otherwise have been ignored or would have failed silently will now generate errors or throw exceptions, alerting you sooner to problems in your code and directing you more quickly to their source.
Prevents accidental globals. Without strict mode, assigning a value to an undeclared variable automatically creates a global variable with that name. This is one of the most common errors in JavaScript. In strict mode, attempting to do so throws an

error.
Eliminates this coercion. Without strict mode, a reference to a this value of null or undefined is automatically coerced to the global. This can cause many headfakes and pull-out-your-hair kind of bugs. In strict mode, referencing a a this value of null or undefined throws an error. Disallows duplicate parameter values. Strict mode throws an error when it detects a duplicate named argument for a function (e.g., function foo(val1, val2, val1)), thereby catching what is almost certainly a bug in your code that you might otherwise have wasted lots of time tracking down. Note: It used to be (in ECMAScript 5) that strict mode would disallow duplicate property names but as of ECMAScript 2015 this is no longer the case.
Makes eval() safer. There are some differences in the way eval() behaves in strict mode and in non-strict mode. Most significantly, in strict mode, variables and functions declared inside of an eval() statement are not created in the containing scope (they are created in the containing scope in non-strict mode, which can also be a common source of problems). Throws error on invalid usage of delete. The delete operator (used to remove properties from objects) cannot be used on non-configurable properties of the object. Non-strict code will fail silently when an attempt is made to delete a non-configurable property, whereas strict mode will throw an error in such a case.

Q25: What is the difference between test () and exec () methods?

Both test () and exec () are RegExp expression methods.
Using test (), we will search a string for a given pattern, if it finds the matching text then it returns the Boolean value 'true' and else it returns 'false'.
But in exec (), we will search a string for a given pattern, if it finds the matching text then it returns the pattern itself and else it returns 'null' value.

Q26: What is the use of 'debugger' keyword in JavaScript code?

Using the 'debugger' keyword in the code is like using breakpoints in the debugger.
To test the code, the debugger must be enabled for the browser. If debugging is disabled for the browser, the code will not work. During debugging the code below should stop executing, before it

Q27: What is JavaScript Hoisting?

Using 'JavaScript Hoisting' method, when an interpreter runs the code, all the variables are hoisted to the top of the original /current scope. If you have a variable declared anywhere inside the JavaScript code then it is brought to the top.
This method is only applicable for the declaration of a variable and is not applicable for initialization of a variable. Functions are also hoisted to the top, whereas function explanations are not hoisted to the top.
Basically, where we declared the variable inside the code doesn't matter much.

Q28: What is the difference between 'var' and 'let' keyword?

Var	let
'var' keyword was introduced in JavaScript code from the beginning Stage itself.	'let' keyword is introduced in 2015 only.
'Var' keyword has function scope. The variable defined with var is available anywhere within the function	A variable declared with 'let' keyword has a scope only with in that block. So, let has a Block Scope.
The variable declared with 'var' be hoisted	The variable declared with 'let' be hoisted

Q29: What is the difference between 'let' and 'const'?

let	const
using 'let' we can change the value of variable any number of times	using 'const', after the first assignment of the value we cannot redefine the value again

Q30: What is the difference between 'null' and 'undefined'?

Both the keywords represent empty values.
In 'undefined', we will define a variable, but we won't assign a value to that variable. On the other hand, in 'null' we will define a variable and assign the 'null' value to the variable.
type of (undefined) and type of (null) object.

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2

callback

```
const second = () => {
console.log('Hello there!');
}

const first = () => {
console.log('Hi there!');
second();

console.log('The End');
}
```

DOM3
DOM4

first();

Micro_Task_queue

```
console.log('Script start');
setTimeout(() => {
console.log('setTimeout');
}, 0);

new Promise((resolve, reject) => {
resolve('Promise resolved'); }).

then(res => console.log(res))
.catch(err => console.log(err));

console.log('Script End');
```

Micro_Task_queue2
this time with two promises and two setTimeout.

```
console.log('Script start');
setTimeout(() => {
console.log('setTimeout 1');
}, 0);

setTimeout(() => {
console.log('setTimeout 2');
}, 0);

new Promise((resolve, reject) => {
resolve('Promise 1 resolved'); })

.then(res => console.log(res))
.catch(err => console.log(err));

new Promise((resolve, reject) => {
resolve('Promise 2 resolved'); })

.then(res => console.log(res))
.catch(err => console.log(err));

console.log('Script End');
```

non-blocking_asynchronous_way

```
const networkRequest = () => {
setTimeout(() => {

console.log('Async Code');
}, 2000);
};

console.log('Hello World');
networkRequest();

console.log('The End');
```

non-blocking

```
const networkRequest = () => {
setTimeout(() => {

console.log('Async Code');
}, 2000);
};

console.log('Hello World');
networkRequest();
```

synch

```
const second = () => {
console.log('Hello there!');
}

const first = () => {
console.log('Hi there!');

second();
console.log('The End');
}

first();
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

callback

```
var fruits = ['apple','bannana','orange'];
fruits.forEach(function(fruit, index){

console.log(index+1 +" . "+ fruit)
});
fruits;
```

Use named functions as callbacks

```
function namedCallback(){
alert("namedCallback()")
}

function testFunction(callback){
callback()
}

testFunction(namedCallback);
```

callback2

```
var author = 'FF';
function namedCallback(param){
alert('namedCallback() called by '+param)
}

function testFunction(callback){
callback()
}

testFunction(namedCallback(author))
```

Multiple callback functions can be used as parameters of another function.

```
function printList(callback) {
// do your printList work
console.log('printList is done');
callback();
}

function updateDB(callback) {
// do your updateDB work
console.log('updateDB is done');

callback()
}

function getDistanceWithLatLong(callback) {
// do your getDistanceWithLatLong work
console.log('getDistanceWithLatLong is done');

callback();
}
```

```
function runSearchInOrder(callback) {
getDistanceWithLatLong(function() {

updateDB(function() {

printList(callback);
});
});
}

runSearchInOrder(function(){console.log('finished')});
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

class onjects

```
class Vehicle {
constructor(name, maker, engine) {
this.name = name;
this.maker = maker;
this.engine = engine;
}
}
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');
console.log(bike1.name);
console.log(bike2.maker);
```

functional object

```
function funObject(name, salery){
this.name=name;
this.salery=salery;
this.printInfo = function(){
console.log(this.name);
console.log(this.salery);
}
}
var obj = new funObject("mukesh", 4657789);
obj.printInfo();
```

nested objects

```
const user = {
id: 101,
email: 'jack@dev.com',
personalInfo: {
name: 'Jack',
address: {
line1: 'westwish st',
line2: 'washmasher',
city: 'wallas',
state: 'WX'
}
}
}
const name = user.personalInfo.name;
const userCity = user.personalInfo.address.city;
console.log(userCity);
```

object literals

```
var obj={
name: "",
salery: "",
printInfo: function(){
console.log(this.name);
console.log(this.salery);
}
}
obj.name="vinnet";
```

```
obj.salery=12346;
obj.printInfo()
```

singleton object

```
var obj = new function() {
this.name = "";
this.age = "";
this.printInfo = function() {
console.log(this.name);
console.log(this.age);
};
}
// Initializing object.
obj.name = "Vineet";
obj.age = 20;
// Calling method of the object.
obj.printInfo();
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

block scope

```
let x=77;
{
let x=23; // legal
console.log(x);
}
let x=67;// illegal
console.log(x);
```

function scope

```
function fun(){
let num=10;
console.log(num);
}
fun();
console.log(num);
```

Global scope
let is a keyword used to declare variables in javascript that are block scoped.
A variable can be either of global or local scope. A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.

```
let num=10;
console.log(num);
function fun(){
console.log(num);
}
fun();
```

Lexical declaration

```
x=12;
console.log(x);
let x;
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2

constructor invocation*Constructor invocation is performed when new keyword is followed by an function name, and a set of opening and closing parentheses(with or without arguments).*
Firstly, an empty object is created that is an instance of the function name used with new (i.e : people(name, age)).
Then, it links the prototype of the constructor function(people) to the newly created object, thus ensuring that this object can inherit all properties and methods of the constructor function
Then, the constructor function is called on this newly created object.If we recall the method invocation, we will see that is similar.Thus, inside the constructor function, this gets the value of the newly created object used in the call.
Finally, the created object, with all its properties and methods set, is returned to person1

```
let people = function(name, age) {
this.name = name;
this.age = age;

this.displayInfo = function() {
```

DOM3
DOM4

```
console.log(this.name + " is " + this.age + " years old");
}
}

let person1 = new people('John', 21);


person1.displayInfo();
```

function invocation*Function invocation refers to the process of invoking a function using its name or an expression that evaluates to the function object.*
if it is invoked through the function invocation without strict mode, has the value of the global object, which is the window object in the browser environment.
However, this is not always the case. If the doSomething() function were running in strict mode, it would log undefined instead of the global window object.This is because, in strict mode, the default value of this, for any function object is set to undefined instead of the global object.

```
function doSomething() {
//'use strict';

console.log(this)

function innerFunction() {
// Also logs undefined, indicating that strict mode permeates to inner function scopes

console.log(this)
}

innerFunction();
}

doSomething();
```

method invocation*logInfo() is a method of the person object and we invoked it using the object invocation pattern.*
Inside such a method, that has been invoked using the property accessors, this will have the value of the invoking object, that is this will point to the object that was used in conjunction with the property accessor to make the call.

```
let person = {
name : "John",
age : 31,
logInfo : function() {

console.log(this.name + " is " + this.age + " years old ");
}
}

person.logInfo()
```

method invocation2*logInfo() is a method of the person object and we invoked it using the object invocation pattern.*
Inside such a method, that has been invoked using the property accessors, this will have the value of the invoking object, that is this will point to the object that was used in conjunction with the property accessor to make the call.
when method invocation patterns are used, the value of this is set to the calling object.

```
let add = {
num : 0,

calc : function() {
console.log(this + ' ')
this.num += 1;

return this.num;
}
};

console.log(add.calc() + '
'); console.log(add.calc());
```

method invocation3

What happens to this in a function nested inside a method of an object?

innerfunc() is called from within the calc() method using a simple function invocation. This means, inside innerfunc() this is set to the global object, which does not have a num property, and hence the NaN outputs are obtained.

```
let add = {
num : 0,
calc : function() {
console.log(this + ' ')
```

```
function innerfunc() {
this.num += 1;

console.log(this + ' ');
return this.num


} return innerfunc();
}
};

console.log(add.calc());
```

How can we retain the value of this from the outer method inside the nested function?

```
let adding = {
num : 0,

calc : function() {
console.log(this + ' ')
thisreference = this;

function innerfunc() {
thisreference.num += 1;

console.log(thisreference + ' ');
return thisreference.num;
}
return innerfunc();
}
};

console.log(adding.calc());
```

Other solutions to this problem involve using bind(), call() or apply()

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons
jsgeeksforgeeks
asynchronous
callback
jsObjects
scope
this
DOM
DOM2
DOM3
DOM4

DOM

EXAMINE THE DOCUMENT OBJECT

```
console.dir(document);
console.log(document.domain);
console.log(document.URL);
console.log(document.title);

document.title = 123;

console.log(document.doctype);
console.log(document.head);
console.log(document.body);
console.log(document.all);
console.log(document.all[10]);
document.all[10].textContent = 'Hello';

console.log(document.forms[0]);
console.log(document.links);
console.log(document.images);
```

GETELEMENTBYID

```
console.log(document.getElementById('header-title'));

var headerTitle = document.getElementById('header-title');
var header = document.getElementById('main-header');

console.log(headerTitle);

headerTitle.textContent = 'Hello';
headerTitle.innerText = 'Goodbye';

console.log(headerTitle.innerText);

headerTitle.innerHTML = '
```

Hello

';
header.style.borderBottom = 'solid 3px #000';

GETELEMENTSBYCLASSNAME

var items = document.getElementsByClassName('list-group-item');

console.log(items);
console.log(items[1]);

items[1].textContent = 'Hello 2';
items[1].style.fontWeight = 'bold';
items[1].style.backgroundColor = 'yellow';
Gives error

for(var i = 0; i < items.length; i++) *{
items[i].style.backgroundColor = '#f4f4f4' }

GETELEMENTSBYTAGNAME
var li = document.getElementsByTagName('li');
console.log(li);
console.log(li[1]);

li[1].textContent = 'Hello 2';
li[1].style.fontWeight = 'bold';
li[1].style.backgroundColor = 'yellow';

for(var i = 0; i < li.length; i++) {
li[i].style.backgroundColor = '#f4f4f4'
}

QUERYSELECTOR
var header = document.querySelector('#main-header');
header.style.borderBottom = 'solid 4px #ccc';

var input = document.querySelector('input');
input.value = 'Hello World'

var submit = document.querySelector('input[type="submit"]');
submit.value="SEND"

var item = document.querySelector('.list-group-item');
item.style.color = 'red';

var lastItem = document.querySelector('.list-group-item:last-child');
lastItem.style.color = 'blue';

var secondItem = document.querySelector('.list-group-item:nth-child(2)');
secondItem.style.color = 'coral';

QUERYSELECTORALL
var titles = document.querySelectorAll('.title');

console.log(titles);
titles[0].textContent = 'Hello';

var odd = document.querySelectorAll('li:nth-child(odd)');
var even= document.querySelectorAll('li:nth-child(even)');

for(var i = 0; i < odd.length; i++){
odd[i].style.backgroundColor = '#f4f4f4';
even[i].style.backgroundColor = '#ccc';
}

DOM

createElement
Create a div

```
var newDiv = document.createElement('div');

Add class
newDiv.className= 'hello';

Add id
newDiv.id = 'hello1';

Add attr
newDiv.setAttribute('title', 'Hello Div');

Create text node
var newDivText = document.createTextNode('Hello World');

Add text to div
newDiv.appendChild(newDivText);

var container = document.querySelector('header .container');
var h1 = document.querySelector('header h1');

console.log(newDiv);

newDiv.style.fontSize = '30px';

container.insertBefore(newDiv, h1);

for(var i = 0; i < odd.length; i++){
odd[i].style.backgroundColor = '#f4f4f4';
even[i].style.backgroundColor = '#ccc';
}
```


DOM

EVENTS

```
var button = document.getElementById('button').addEventListener('click', buttonClick);

function buttonClick(e){
console.log('Button clicked');

document.getElementById('header-title').textContent = 'Changed';
document.querySelector('#main').style.backgroundColor = '#f4f4f4';
console.log(e);

console.log(e.target);
console.log(e.target.id);
console.log(e.target.className);
console.log(e.target.classList);

var output = document.getElementById('output');
output.innerHTML = '

'+e.target.id+'

';

console.log(e.type);

console.log(e.clientX);
console.log(e.clientY);

console.log(e.offsetX);
console.log(e.offsetY);

console.log(e.altKey);
```

```
console.log(e.ctrlKey);
console.log(e.shiftKey);
}

var button = document.getElementById('button');
var box = document.getElementById('box');

button.addEventListener('click', runEvent);
button.addEventListener('dblclick', runEvent);
button.addEventListener('mousedown', runEvent);
button.addEventListener('mouseup', runEvent);

box.addEventListener('mouseenter', runEvent);
box.addEventListener('mouseleave', runEvent);

box.addEventListener('mouseover', runEvent);
box.addEventListener('mouseout', runEvent);

box.addEventListener('mousemove', runEvent);

var itemInput = document.querySelector('input[type="text"]');
var form = document.querySelector('form');
var select = document.querySelector('select');
itemInput.addEventListener('keydown', runEvent);
itemInput.addEventListener('keyup', runEvent);
itemInput.addEventListener('keypress', runEvent);

itemInput.addEventListener('focus', runEvent);
itemInput.addEventListener('blur', runEvent);

itemInput.addEventListener('cut', runEvent);
itemInput.addEventListener('paste', runEvent);

itemInput.addEventListener('input', runEvent);

select.addEventListener('change', runEvent);
select.addEventListener('input', runEvent);

form.addEventListener('submit', runEvent);

function runEvent(e){
e.preventDefault();
console.log('EVENT TYPE: '+e.type);

console.log(e.target.value);
document.getElementById('output').innerHTML = '

'+e.target.value+'

';

output.innerHTML = '

MouseX: '+e.offsetX+'

MouseY: '+e.offsetY+'

';

document.body.style.backgroundColor = "rgb("+e.offsetX+","+e.offsetY+", 40)";
}
```

arrayjs
date
oops
advancedJSGeeksForGeeks
String
javascriptInterviews
javascriptQuestuons

DOM4

```
const person_ = {
firstName: 'John',
age: 30,
```

```
hobbies: ['music', 'movies', 'sports'],
address: {
street: '50 Main st',
city: 'Boston',
state: 'MA'
}
}
```

Get single value

```
console.log(person_.name)
```

Get array value

```
console.log(person_.hobbies[1]);
```

Get embedded object

```
console.log(person_.address.city);
```

Add property

```
person.email = 'jdoe@gmail.com';
```

Array of objects

```
const todos = [
{
id: 1,
text: 'Take out trash',
isComplete: false
},
{
id: 2,
text: 'Dinner with wife',
isComplete: false
},
{
id: 3,
text: 'Meeting with boss',
isComplete: true
}
];
```

Get specific object value

```
console.log(todos[1].text);
```

Format as JSON

```
console.log(JSON.stringify(todos));
```

For

```
for(let i = 0; i <= 10; i++){
console.log(For Loop Number: i);
}
```

While

```
let i = 0
while(i <= 10) {
console.log(While Loop Number: i);
i++;
}
```

Loop Through Arrays For Loop

```
for(let i = 0; i < todos.length; i++){
console.log( Todo i1: todos[i].text);
}
```

For...of Loop

```
for(let todo of todos) {
console.log(todo.text);
}
```

HIGH ORDER ARRAY METHODS (show prototype)
forEach() - Loops through array

```
todos.forEach(function(todo, i, myTodos) {
console.log(i1: todo.text);
console.log(myTodos);
});
```

map() - Loop through and create new array

```
const todoTextArray = todos.map(function(todo) {
return todo.text;
console.log(todoTextArray);
});
```

filter() - Returns array based on condition

```
const todo1 = todos.filter(function(todo) {
return todo.id === 1;
});
```

Mouse Event

```
btn.addEventListener('click', e => {
e.preventDefault();
console.log(e.target.className);
document.getElementById('my-form').style.background = '#ccc';
document.querySelector('body').classList.add('bg-dark');
ul.lastElementChild.innerHTML = '
```

Changed

```
';
});
```

USER FORM SCRIPT
Put DOM elements into variables

```
const myForm = document.querySelector('#my-form');
const nameInput = document.querySelector('#name');
const emailInput = document.querySelector('#email');
const msg = document.querySelector('.msg');
const userList = document.querySelector('#users');
```

Listen for form submit

```
myForm.addEventListener('submit', onSubmit);

function onSubmit(e) {
e.preventDefault();

if(nameInput.value === '' || emailInput.value === '') {
msg.classList.add('error');
msg.innerHTML = 'Please enter all fields';

setTimeout(() => msg.remove(), 3000);
}
else {
const li = document.createElement('li');

li.appendChild(document.createTextNode(nameInput.value: emailInput.value));
userList.appendChild(li);

nameInput.value = '';
emailInput.value = '';
}
}
```