1. Java script Asynchronous or Synchronous. - >Syncronous and single threaded
2. <https://medium.com/@madasamy/15-javascript-concepts-that-every-nodejs-programmer-must-to-know-6894f5157cb7>
3. Rest parameters:  
   The **rest parameter** syntax allows a function to accept an indefinite number of arguments as an array,  
   function sum(...theArgs) {

}

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters>

1. Execution Context : ( How java script works)  
   - has two parts memory and code  
   - variables and functios initialize into memory
2. Call stack/ control stack/ program stack /runtime stack/ machine stack/ execution context stack:  
    Call stack contain the execution context. At bottom there is global execution context. Whenever execution context is created ,it’s pushed into the call stack. After completing the execution this execution context is poped up and controller moves to global or next execution context.
3. Hoisting :  
    Access the variables and functions before initializing is known as hoisting.
4. Undefined Vs not defined
5. Let and const
6. Block scop
7. Closures:
   1. Function with his lexical scope know as closure,
   2. function a(){

let a=5

function b(){

return a=a+5

}

return b

}

let z=a()

console.log(z)

1. Map vs filter vs for loop vs reduce
   1. Reduce:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce>  
  
  
const message = [1, 2,3,4];

// reduce join each element of the string

let joinedString = message.reduce(function (previous, current,currentIndex,array) {

console.log(previous,current)

console.log("currentIndex",currentIndex)

console.log(array)

return previous + current;

},2);

console.log("final----",joinedString);

1. Lexical environment
2. Temporal Dead Zone
3. If functions
4. Difference between call and apply
5. Primitive and non-primitive data types
6. Let is block scoped
7. **Call , Apply and bind**

All these three are used for burrow the function from other object.  
For example we have object and there is an function , we want to access this function but we can’t directly. For that we can use call, apply or bind methods.  
var car = {

registrationNumber: "GA12345",

brand: "Toyota",

displayDetails: function(){

console.log(this.registrationNumber + " " + this.brand);

}

}

var myCarDetails = car.displayDetails;

myCarDetails()

* 1. Call  
       
     var car = {   
      registrationNumber: "GA12345",

brand: "Toyota"

}

function displayDetails(ownerName) {

console.log(ownerName + ", this is your car: " + this.registrationNumber + " " + this.brand);

}displayDetails.call(car, "Vivian"); // Vivian, this is your car: GA12345 Toyota

displayDetails.call(car, "Vivian"); // Vivian, this is your car: GA12345 Toyota

* 1. Apply
     1. Apply is similar to Call method , but it will take perameters into array

var car = {

registrationNumber: "GA12345",

brand: "Toyota"

}

function displayDetails(ownerName) {

console.log(ownerName + ", this is your car: " + this.registrationNumber + " " + this.brand);

}

displayDetails.apply(car, ["Vivian"]);

* 1. Bind
     1. bind method is similar to call but it create copy of function and assign to variable.

var car = {

registrationNumber: "GA12345",

brand: "Toyota",

displayDetails: function(ownerName){

console.log(ownerName + ", this is your car: " + this.registrationNumber + " " + this.brand);

}

}

var myCarDetails = car.displayDetails.bind(car, "Vivian");

myCarDetails()

1. CORS
   1. Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.

function hoist() { a = 20; var b = 100; } hoist(); console.log(a);

**Magic:**

Problem :1 --------------  
function ab(){

for(var a=0;a<=5;a++){

setTimeout(()=>{

console.log(a)

},1000)

}

}

ab()   
output: 666666  
var replace their values at the same refrence , and in settime out a refferes to reference not value

Problem :2 --------------  
function ab(){

for(let a=0;a<=5;a++){

setTimeout(()=>{

console.log(a)

},1000)

}

}

ab()   
output: 0123456  
because let is block scope , every time new a is created and reffers to new value.

Problem :1-3 --------------  
function ab(){

for(var a=0;a<=5;a++){

close(x){

setTimeout(()=>{

console.log(x)

},1000)

}

}

close(a)

}

ab()   
  
 output: 0123456  
It was just because of memory referece now every a refers to new memory.