https://blog.risingstack.com/node-js-interview-questions-and-answers-2017/

https://medium.com/@vigowebs/frequently-asked-node-js-interview-questions-and-answers-b74fa1f20678

<http://www.tutorialsteacher.com/nodejs/nodejs-modules>

# Node.js Module

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

## Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

### Node.js Core Modules

Node.js is a lightweight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

The following table lists some of the important core modules in Node.js.

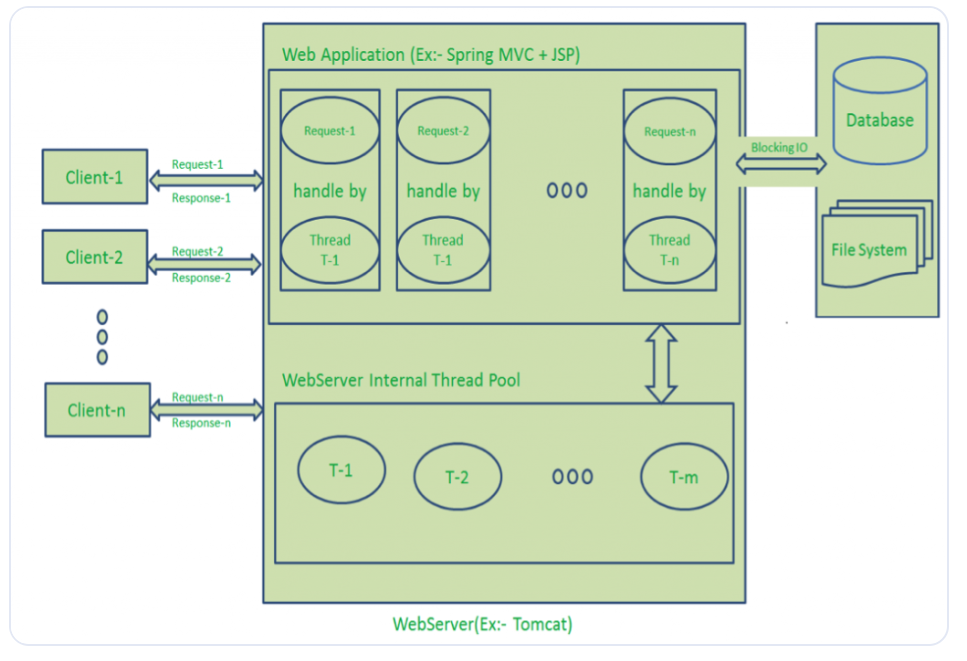
| Core Module | Description |
| --- | --- |
| [http](https://nodejs.org/api/http.html) | http module includes classes, methods and events to create Node.js http server. |
| [url](https://nodejs.org/api/url.html) | url module includes methods for URL resolution and parsing. |
| [querystring](https://nodejs.org/api/querystring.html) | querystring module includes methods to deal with query string. |
| [path](https://nodejs.org/api/path.html) | path module includes methods to deal with file paths. |
| [fs](https://nodejs.org/api/fs.html) | fs module includes classes, methods, and events to work with file I/O. |
| [util](https://nodejs.org/api/util.html) | util module includes utility functions useful for programmers. |

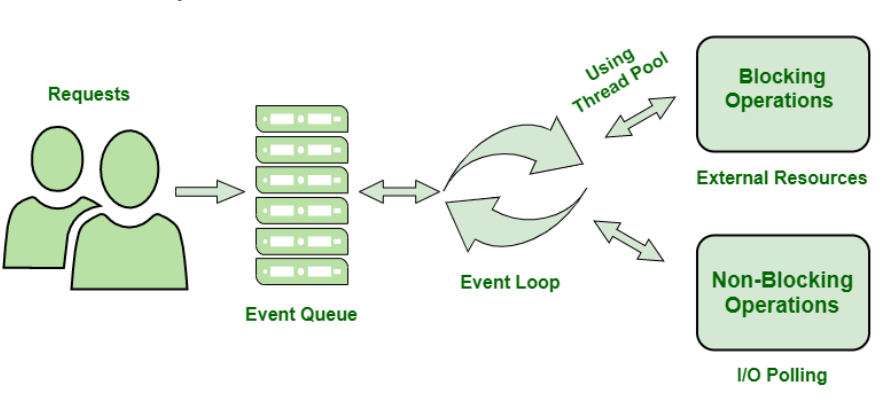
# https://community.esri.com/thread/75046-->call back return value

This is impossible as you cannot return from an asynchronous call inside a synchronous method.

In this case you need to pass a callback to foo that will receive the return value

# https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop-->Node js architecture.





# **Q. Return value from nested callback instead of the parent function?**

Have addNewUser accept a callback as its last argument and have the innermost function call the callback with the value.

Alternatively, you could look into having addNewUser return a promise. RSVP or Q are implementations of the Promises/A :

function addNewUser(hash,name,number,time,syncTime) {

var deferred = Q.defer();

connection.query("SELECT ...", function(err, rows, fields) {

if(err) { deferred.reject(err); }

if(rows[0]) { deferred.reject("some reason"); }

connection.query("INSERT INTO ...", function(err, rows, fields) {

if(err) { deferred.reject(err); }

deferred.resolve(rows[0]); // Whatever addNewUser would return normally

});

});

return deferred.promise;

}

Then the caller would use it like this:

addNewUser(...).then(function(newUserAdded) {

// Do something with newUserAdded here

}, function(err) {

// Do something with the error here

});

# **Difference between Json.Parse() and Json.Stringify()?**

a.Json.Stringify() is used for convert Json to String. ==>Json.Stringify(null)//'null';

Json.Parse is used for converting string to json. ==>Json.Parse('null')//null;

# **1.what is error first call back?**

a. passing error as first argument to the function is known as error first call here error will check first after that data checks.

# Two types of function creation will be there

functionName (parameter){

new promise((resolve,reject)=>{

try{}

catch(error){

reject error;

}

});

return value;

}

const functionName=parameters=>{

return value;

}

# **2.How can you reduce consistency?**

using tools like eslint and standard.

# 3.**How does Node.js support multi-processor platforms, and does it fully utilize all processor resources?**

a. using cluster module which is one of module in node js, it can support to run multiple worker processors in node js.

-->function addAfterTwoSeconds(x){

return new Promise(resolve=>{

setTimeout(()=>{

resolve(x\*2);

},2000)

})

}

async function addPromise(x){

const a=await addAfterTwoSeconds(10);

const b=await addAfterTwoSeconds(20);

const c=await addAfterTwoSeconds(30);

return (x+a+b+c);

}

addPromise(10).then(result=>{

Promise.resolve(1)

.then((x) => x + 1)

.then((x) => { throw new Error('My Error') })

.catch(() => 1)

.then((x) => x + 1)

.then((x) => console.log(x))

.catch(console.error)

console.log(result);

})

# **benefits of node js:**

-->Asynchronous

-->Single Threading

-->No Buffer

# **Node package installation**

-->node packages can be installed in two ways global installation and local installation here globally installed files are stored in

user<directory> these are no need to require and locally installed packages are stored in our api package.json we need to use require statement to get the local package.

-->package.json represent the root directory of any node modules.

# **4.what is blocking code?**

a. blocking code is nothing, but synchronous calling means second block not executes until first block code is executed.

# **5.what is event emitter?**

a. It is nothing but fire the event whenever it is required.

const event=require('event');

const event\_instance=new event.EventEmitter();

-->Through above event instance we can fire the respective event which supports event.on etc....

# **6.How many types of streams are present in node?**

a. 1. Readable 2. Writable 3. Duplex 4. Transform.

--->fs.open(path,flags[,mode],callback);

# **7.How can you avoid callback hell?**

a. callback hell is avoided by using promises, modularization, asyn/await function calling.

# **8.Advantages of Node js?**

-->Asynchronous

-->Very Fast

-->No Blocking

# **9.What is event Looping?**

-->To Process the external events and convert them into callback invocations we use event loops.

# **Pros and cons of node js?**

-->if there is no intensive CPU computation [mathematical calculation] then node give quick responsiveness.

-->if

**Blocking Code:**

var data=fs.readFile('input.txt');

console.log(data);

console.log('code completed');

Output:

reading text file

code completed

**Non-Blocking Code:**

fs.readFile('input.txt',function(err,data){

if(err){

console.log(err.message);

}

console.log(data);

});

console.log('code completed');

Output:

code completed

reading text file

# ===>

add(1)() return 1

add(1)(2)() return 2

add(1,2,3,4..n) return n

var add=(param1)=>({

return function()

{

}

})

# **what is module in node js?**

module is a simple or complex functionality organized in single or multiple file which can be used in throughout the application.

# **-->Diff b/w general n arrow functions?**

a. this behavior is changes in both arrow and general functions'

-->Named expression functions.

# **what is the difference between method and functions?**

1. Functions and methods both are functions in JavaScript. A method is just a function which is a property of an object.

--->function fun(param1, param2){

// some code...

}

-->var obj = {

name : "John snow",

work : function someFun(paramA, paramB) {

// some code..

}

}

# **--->For js event loop properties**

1.Heap

2.Stack

3.Web Api

4.Queue

5.Event Loop.

# **--->For Node js Architecture**

client req ===>

client request =====> event queue ==> event loop ===>[]==>thread pool==>D.B

# **--->What is event loop?**

1. Event loop can perform non blocking i/o operations.

setTimeout(()=>{

console.log('');

})

# **-->what is object destructuring?**

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

https://stackoverflow.com/questions/15349733/setimmediate-vs-nexttick

# **ES6 or ES2015 additional features:**

-->Object destructuring

-->arrow functions

-->template literals(`${err}` stringify in our snr)

-->classes

-->Spread Operator

-->Types of object declaration.

**Difference:**

var a = Object.create(null); is not the same as var a = {};

var a = Object.create(null); sets the prototype of a as null where as

var a = {}; sets the prototype of a as the Object object

# **Event Loop Internal Properties:**

1.Timers-->Used for setTimeOut() and setInterval()

2.Pending CallBacks-->I/O executions will be perform here.

3.Ideal--->Internal Operations are performed here.

4.poll-->Get the new I/O operations and executions performed here and timer can be set here.

5.check-->it will check the setTimeout threshold period.

6.close callback-->callbacks can close here.

**Timers:**

1. **setTimeout()**-->it can run only once.

ex:

for(i=1;i<5;i++)

{

setTimeout(()=>{

console.log(i);

},1000)

}

o/p:5,5,5,5

2.**setInterval()-->**it can run infinity rimes.

ex:

for(i=1;i<5;i++)

{

setInterval(()=>{

console.log(i);

},1000)

}

o/p:5,5,5,5 and so on......

3. **setImmediate()-->**It can return the value immediately after current event execution is done.

function check(){

setTimeout(()=>{

console.log('setTimeOut');

},0)

setImmediate(()=>{

console.log('setImmediate')

})

}

o/p:

setImmediate

setTimeOut

**prosess.nextTick():**

-->You should use nextTick when you want to ensure that your code is executed in next event loop instead of after a specified time.

nextTick is more efficient than timers and when you want to ensure the function you call is executed asynchronously.

# **-->Difference b/w arrow and normal function?**

a. For General function if we want to access the current function value this is not directly referenced, we need to do external referencing, but in arrow function is self-reference no need to use external referencing.

ref url:http://exploringjs.com/es6/ch\_arrow-functions.html

**Solution1**

function Prefixer(prefix) {

this.prefix = prefix;

}

Prefixer.prototype.prefixArray = function (arr) {

var that = this; // (A)

return arr.map(function (x) {

return that.prefix + x;

});

};

Now Prefixer works as expected:

> var pre = new Prefixer('Hi ');

> pre.prefixArray(['Joe', 'Alex'])

[ 'Hi Joe', 'Hi Alex' ]

**solution2:**

function Prefixer(prefix) {

this.prefix = prefix;

}

Prefixer.prototype.prefixArray = function (arr) {

return arr.map(x=>this.prefix + x);

};

var pre = new Prefixer('Hi ');

pre.prefixArray(['Joe', 'Alex'])

# **core modules in node js?**

http

path

url

querystring

fs

utils

# **1.how many streams are existed in node js?**

there are four types of streams are existed in node js.

1.Readable 2. Writable 3. Duplex 4. Transform

# **call back hell**

call back hell is reduced by using modularization

# **exporting module**

So, there are two types of exporting module will be there.

function method(){}

# **what is event loop?**

-->To Process the external events and convert them into callback invocations we use event loop.

# **-->Blocking Code:**

var data=fs.readFile('input.txt');

console.log(data);

console.log('code completed');

o/p:

# **---->Non Blocking:**

fs.readFile('input.txt',(err,data)=>{

if(err){

console.log(err.message);

}

else{console.log(data);}

})

console.log('code completed');

# **var**

-->var is function scope see below example:

function test()

{

var t=20;

}

test()

t = 5 \* t;

console.log(t)

VM5476:7 Uncaught ReferenceError: t is not defined

at <anonymous>:7:1

**Reason**: Var is function scope t is not assessable outside of te function.

function test()

{

t=20;

}

test()

t = 5 \* t;

console.log(t)

o/p:100

**Reason**: Here t will be assessable to outside of the function because it is take it as window object.

# **'use strict'**

-->Here if we put 'use strict' then it will restrict the window object and delete object option is also not work see below example.

1)'use strict'

function test()

{g

t=20;

}

test()

t = 5 \* t;

console.log(t)

VM5476:7 Uncaught ReferenceError: t is not defined

at <anonymous>:7:1

2)var a=10;

delete a;//error occur

**use strict will restrict below operations:**

1. variable using without declaration
2. Delete a function or object
3. Deleting an undeletable property
4. Duplicating a parameter name
5. Octal numeric literals/escape are not allowed
6. Writing to a read-only property
7. Writing to a get-only property
8. The string "eval" cannot be used as a variable

# **-->Spread Operator:**

var mid=[3,4];

var st=[1,2,mid,5,6]

console.log(st);

o/p:[ 1, 2, [ 3, 4 ], 5, 6 ]//Not expected o/p

var mid=[3,4];

var st=[1,2,...mid,5,6]

console.log(st);

o/p:[ 1, 2, 3, 4,5, 6 ]//expected o/p

-->Spread operator solution for Math.max();

var arr=[1,2,3,4,5];

function math(arr)

{

return Math.max.apply(null,arr);

}

console.log(math(arr))

o/p:5

It is simplified using spread operator:

var arr=[1,2,3,4];

console.log(Math.max(...arr));

o/p:

# **What is variable Hoisting?**

-->The variables and functions declarations are moved to top of their scope before code execution.

https://scotch.io/tutorials/understanding-hoisting-in-javascript

<https://scotch.io/tutorials/understanding-hoisting-in-javascript>

# **Session maintenance**

--->Session maintenance is not there in javascript. only cookies are worked in js because js can work on current page. if we want to transfer value

from one page to another page then we can use dotnet, java, php, etc...

# **maintain cookie in js :**

-->When we enter a site(fb, gmail) with username and password browser ask us to save user data that is cookie.

Creating coockie:document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

# **Template Literal:**

Template literals is a additional way to create and handle dynamic strings/string templates

usages:

# **1.MultiLIne Strings**

-->console.log('First Line\n'+'second Line') //Normal syntax

-->console.log(`First Line

second Line`); //Template literal syntax

# **2.Expression Interpolation**

var a=10;

var b=20;

var c=b?(a+b):a; //Normal ternary operator

var c=`{b?a+b:a}` //Template literal syntax

**Function declaration:**

function sample(){

return 'return sample';

}

sample();//return sample

**Arrow Function Expressions:**

var sample=()=>{

return 'return arrow sample'

}

sample();//return arrow sample

**class Declaration[Es6 Feature]**

class sample{

constructor(height,weight){

this.height=height;

this.weight=weight;

}

print(){return 'print method calling';}

}

var obj=new sample();

obj.height=100;

obj.weight=200;

console.log(obj);//sample { height: 100, weight: 200 }

obj.print();//print method calling

# **Object Literals:**

https://blog.kevinchisholm.com/javascript/difference-between-object-literal-and-instance-object/

-->If the object declared with properties and methods known as object literal and that properties and methods are public. the function declared inside object capable of private scope.

var obj={

name:'gopi',

Class:'first',

subjects:function(){

return {lang1:'telugu',lang2:'english'}

}

}

var {lang1,lang2}=obj.subjects();

console.log(lang1)//telugu

# **Instance Object:**

Here instance object in the sense creating instance for the function and the properties, methods declared inside the function are private only.

by using 'this' keyword we make them as public see below code.

var fun=function(){

color='blue';

this.getcolor=function(){

return color;

}

this.setcolor=function(newcolor){

color=newcolor;

}

}

var inst\_obj=new fun();

console.log(inst\_obj.getcolor());//blue

inst\_obj.setcolor('yellow');

console.log(inst\_obj.getcolor());//yellow

console.log(inst\_obj.color);//o/p is undefined but we access color by put it as this keyword.

# **//Regarding window object:**

function sample(){

st='same';

}

sample();

var obj=new sample();

console.log(obj.st);// o/p is undefined but we access st by put it as this keyword.

# **Rest parameters:**

https://javascript.info/rest-parameters-spread-operator

function sum(...args)

{

var sum=0;

for(let arg of args)

{

sum+=arg;

}

return `Total Sum is ${sum} and passing arguments lengthis ${args.length}`

}

sum(1,2,3,4,5,5);

# **-->call, apply, bind methods difference**

**bind**: if a function needs to invoke later in certain events then we use bind.

var obj={

name:'Gopi'

}

var sample=function(a,b,c){

console.log(this.name+' is '+ a +' and '+b +' and '+c);

}

var args=['sam','dom','ram'];

var bound=sample.bind(obj)

bound('sam','dom','ram');===>o/p Gopi is sam and dom and ram

-->Here if we declare bound function with new keyword then the value of first parameter is ignored see below example.

var obj={

name:'Gopi'

}

var sample=function(a,b,c){

console.log(this.name+' is '+ a +' and '+b +' and '+c);

}

var args=['sam','dom','ram'];

var bound=sample.bind(obj)

new bound('sam','dom','ram');==>o/p undefined is sam and dom and ram

**call()**;

-->If we want to invoke a function then we use call and apply but their usage is different see below example.

var obj={

name:'Gopi'

}

var sample=function(a,b,c){

console.log(this.name+' is '+ a +' and '+b +' and '+c);

}

sample.call(obj,'sam','dom','ram');==>o/p Gopi is sam and dom and ram

**apply();**

var obj={

name:'Gopi'

}

var sample=function(a,b,c){

console.log(this.name+' is '+ a +' and '+b +' and '+c);

}

var args=['sam','dom','ram'];

sample.apply(obj,args);==>o/p Gopi is sam and dom and ram

-->Shrinkwrap:

if we shrinkwrap in node then it will create a new file called shrinkwrap.json in that all package.json dependencies and dev dependencies will be maintained. and if we install the node module next time then packages can be taken

-->

var a=[1,2,3];

console.log(a)

a=[...a,4];

console.log(`using spread operator [${a}]`);

a.splice(1,0,'sample');

console.log(`using splice [${a}]`);

--->console.log([1,2,3]+[4,5,6])

o/p:1,2,34,5,6

# **--->what is the use of clouser**

-->console.log(5<6<7)//true

-->console.log(7>6>5)//false

because 5<6<7 7>6>5 5<6 is true and true equal to one so 1<7 true

7>6>5 here 7>6 is true and true equal to one so 1>5 false.

**-->https://www.youtube.com/watch?v=qsNxdukPc2U-->Node Logics**

# **-->regarding iife means selfinvoking function**

(()=>{

console.log('hi')

})();

# console.log(NaN===NaN)//o/p false

# -->var a=[1,2,2,3,4]--Remove duplicates from array.

var b=[...new Set(a)]--o/p-{1,2,3,4}

-->function a(){return arguments;};

console.log(a('hi'));

o/p-->Arguments ["hi", callee: ƒ, Symbol(Symbol.iterator): ƒ]

-->var a=()=>{return arguments;};

console.log(a('hi'));

VM313:1 Uncaught ReferenceError: arguments is not defined

so for above code we need to define like var a=(n)=>{return n;}

-->var obj={

name:'gopi'

}

# **Object.freeze(obj);**//here it can restrict the insertion of new object properties.

obj.age=26;

console.log(obj);

g

var obj={

name:'gopi'

}

# **Object.seal(obj);**//here it will not allow to add new properties but it will modify the existed properties.

obj.name=26;

console.log(obj);

-->if we want to modify one object property and seaz another property use code like below.

var obj={

name:'gopi'

}

# Object.defineProperty

**Object.defineProperty**(obj,'age',{

value:26,

writable:false

})

obj.name='sam';

obj.age=27

console.log(obj);

->fun1(one){

fun2(two){

fun3(three){

one+Two+three;

}

}

}

==>function celebrityName (firstName) {

var nameIntro = "This celebrity is ";

function secondlastName (theLastName) {

function thirdlastName (thirdLastName) {

return nameIntro + firstName + " " + theLastName+" "

+thirdLastName;

}

return thirdlastName;

}

return secondlastName;

}

var mjName = celebrityName ("Michael");

var third=mjName ("Jackson");

third("sample");

-->for(let i=0;i<3;i++)

{

setTimeout(()=>{

console.log(i);

},1000)

}

o/p:0,1,2

for(var i=0;i<3;i++)

{

setTimeout(()=>{

console.log(i);

},1000)

}

o/p:3,3,3-->see solution in below

for(var i=0;i<3;i++)

{

(setTimeout(()=>{

console.log(i);

},1000))();-->immediate invoking with iify function

}

o/p:0,1,2

# **-->what are javascript types?**

-->Number, boolean, string, object, null, undefined

# **-->How do you create an object in js?**

var obj={};var obj=new obj();

# **-->How do you assign object properties?**

obj['age']=16;obj.age=16;

# **-->How to append value to an array?**

arr[arr.length]=value;

# **--->Reverse String in js**

var str='hi'

str = str.split("");

str = str.reverse();

str = str.join("");

console.log(str);

-->const obj={

a:1,

b:2,

getA:function(){

console.log(this.a);

//return this;---->uncomment this code give good result

},

getB:function(){

console.log(this.b);

}

}

obj.getA().getB();

---->var obj={

a:{

b:{

c:'gopi'

}

}

}

var clone=JSON.parse(JSON.stringify(obj))

console.log(clone.a.b.c)

====>

var a=[1,2,3,4,5];

var b=[2,3,4,5,6,7,9];

var c=a.concat(b).sort((a,b)=> //here we can use spread operator also (...a,...b)

{

return a > b;

});

console.log(c);

# **-->What is shallowcopy and deepcopy?**

https://stackoverflow.com/questions/184710/what-is-the-difference-between-a-deep-copy-and-a-shallow-copy

**-->shallocopy:**

var a={1,2,4};

var b=a;

===>means a=b={1,2,3} here both a and b can reference to same memory location.

--->**Deep copy**:

var a={1,2,3};

var b=json.parse(Json.Stringify(a));

==>means a={1,2,3} and b={1,2,3} here both a and b can referencing to different memory location.

# **-->Callback function:**

A function can passed as parameter to another function then it will be a callback function.

functionOne(err,daat=>{

if(err){

return err;}

console.log(data);

})

functionTwo(callback)

{

callback(null,'Hi')

}

functionTwo(functionOne);

# **-->Event Bubbling:**

If two events are there event1 and event2 then firing will start from event2 to event1 nothing but bubbling up.

**-->Event Capturing/Event delegation:**

If two events are there event1 and event2 then firing will start from event1 to event2 nothing but capturing down.

# **Generators**

-->Generators are used to pause and resume the block of code, here generators can have two methods which are **yield()** and **next()**.

-->if we give run next method then code can run until first yield finding.

function\* Add(x) {

console.log("first yield");//first next

yield x + 1;

console.log("second yield");//second next

var y = yield(null);

y = 6

console.log("third yield");//third next

return x + y;

}

var gen = Add(5);

gen.next();

gen.next();

gen.next();

# **==>Difference between local and session storage?**

-->Here localstorage can be stored without any expiration time if we want to delete localstorage doen with javascript or clear cache

-->Session storage only available duration of browser session

https://www.c-sharpcorner.com/UploadFile/cd7c2e/difference-between-local-storage-session-storage-ans-cookie/

# **-->Difference b/w http and http2?**

# **===>What is token authentication.**

# **==>Difference b/w exports and module.export**

==>array some

==>create sample listener

# **--->some() method:**

-->The return value of some method is true/false here it will check weather the item exist or not in the array if item will exist then return

true else return false.

var a=[1,2,3,4,5];

var b=[];

function ageCount(age){

b.push(age);

return age>4;

}

var some=a.some(ageCount);

console.log(b);

console.log(some);

https://www.simplilearn.com/sap-modules-sap-fi-sap-co-sap-sd-sap-hcm-and-more-rar111-article

# **--> What are exit codes in Node.js? List some exit codes?**

Exit codes are specific codes that are used to end a “process” (a global object used to represent a node process).

ex:unused,Fatal Error

# **-->what is buffer class in Node js?**

Node provides Buffer class which provides instances to store raw data similar to an array of integers but corresponds to a raw memory

allocation outside the V8 heap. Buffer class is a global class that can be accessed in an application without importing the buffer module.

-->buf = new Buffer(256);

len = buf.write("Simply");

console.log("Octets written : "+ len);

# **---->stdout and stderr:**

console.log("I will goto the STDOUT");

console.error("I will goto the STDERR");

process.stdout.write("I will goto the STDOUT")

process.stderr.write("I will goto the STDERR")

# **--->Reduce():**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/Reduce

it is used to reduce the single value from an array.

# **--->shift():**

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_shift

--->It can remove the first element of an array.

# **-->unshift():**

-->it is used to add the elements from beginning of array.

# **-->pop():**

-->It is used to get the last element of an array.

-->MongolClient.connect(url,function(err,db)=>{

dbo=db.db('myDb');

dbo.collection('collectionName').inserOne(myobj,(err,data)=>{

})

})

# **--->REST-->Representation state transfer**

# **-->CAP theorem**

# -->sql-->C+A

# -->NoSql-->C+P

# **REST-->Representation state transfer.**

# **SOAP-->Simple Object Access Protocol**

# **CAP -->COns**

-->function fst(a,b){

const name=a;

const age=b;

return {name,age};

}

function scnd(lacation){

const a=fst.apply('',['gopi',26]);

console.log('addres is',a.name,a.age,lacation);

}

scnd('hyderabad');

-->function fst(a,b){

this.name=a;

this.age=b;

}

fst.prototype.scnd=function(location){

console.log('addres is',this.name,this.age,location)

}

var obj=new fst('gopi',26);

obj.scnd('hyderabad');

-->function fst(a,b){

this.name=a;

this.age=b;

}

function scnd(location){

const obj=new fst('gopi',26);

console.log('address is',obj.name,obj.age,location);

}

scnd('hyd');

-->const obj={

a:'Hi',

b:20,

sayName(st){

console.log(this.a,st)

}

}

**//bind:**

const st=obj.sayName.bind(obj);

st('how r u');

**//call:**

const st=obj.sayName.call(obj,'Hw r u');

**//apply**

const st=obj.sayName.apply(obj,['Hw r u']);

# **-->Object comparing**

http://adripofjavascript.com/blog/drips/object-equality-in-javascript.html

function add(a){

return function sum1(b){

return function sum2(c){

return function sum3(d){

return a+b+c+d;

}

}

}

}

add(10)(2)(3)(4)