**1. What is JavaScript?**

JavaScript is a client-side as well as server side scripting language that can be inserted into HTML pages and is understood by web browsers. JavaScript is also an Object based Programming language.

**Object**-**based language** doesn't support all the features of OOPs like Polymorphism and Inheritance

**2. What are JavaScript Data Types?**

Following are the JavaScript Data types:

* Number
* String
* Boolean
* Function
* Object
* Undefined

**3. What is negative infinity?**

dividing negative number by zero.

**4. Which company developed JavaScript?**

Netscape is the software company who developed JavaScript.

**5. What is the difference between ViewState and SessionState?**

'ViewState' is specific to a page in a session.

'SessionState' is specific to user specific data that can be accessed across all pages in the web application.

**6.**  **3+2+"7"=57**

**“7”-3=4  
 "6"+5+10+"10"=651010  
 “6” \* 5 = 30  
 “6”/6 =1**

**"7"+3+3=733**

**2 \*\*3=8**

**7. how to detect the operating system on the client machine?** navigator.platform

**8. What is the function of delete operator?**

var student= {age:20, batch:"ABC"};

delete student.age;

**9. How generic objects can be created?**

var I = new object();

**10**. **Explain the for-in loop?**

for (variable name in object){

statement or block to execute

}

**11. anonymous function in JavaScript?**

var anon = function() {

alert('I am anonymous');

};

anon();

**12. What is the difference between .call() and .apply()?**

.call() is used when the number of the function's arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.

***TO change dynamically value of pi we can use call and apply.***

var cylinder = {  
 pi: 3.14,  
 volume: function(r, h) {  
 return this.pi \* r \* r \* h;  
 }  
};  
  
cylinder.volume.call({pi: 3.14159}, 2, 6);  
cylinder.volume.apply({pi: 3.14159}, [2, 6]);

it contain two argument:-  
1.Context: A context is an object that replaces **this**keyword inside the area function.   
2.Function Argument

**Bind -: Bind**attaches a brand new **this** to a given function.  
What is the use of **Bind?** It allows us to inject a context into a function which returns a new function with updated context. It means **this** variablewill be user supplied variable.

var newVolume = cylinder.volume.bind({pi: 3.14159}); // This is not instant call

// After some long time, somewhere in the wild

newVolume(2,6); // Now pi is 3.14159

**13. Define unescape() and escape() functions?**The escape () function is responsible for coding a string so as to make the transfer of the information from one computer to the other, across a network.

document.write(escape("Hello? How are you!"));

Output: Hello%3F%20How%20are%20you%21

The unescape() function is very important as it decodes the coded string.  
document.write(unescape("Hello%3F%20How%20are%20you%21"));

Output: Hello? How are you!

**14. What are the decodeURI() and encodeURI()?**EncodeURl() is used to convert URL into their hex coding. And DecodeURI() is used to convert the encoded URL back to normal.

var uri="my test.asp?name=ståle&car=saab";

document.write(encodeURI(uri)+ "<br>");

document.write(decodeURI(uri));

Output -

my%20test.asp?name=st%C3%A5le&car=saab

my test.asp?name=ståle&car=saab

**15. var myArray = [[[]]]**

3d array

declaration

**16.Function Closure:-**The self-invoking function only runs once. The self-invoking function only runs once.  
var add = (function () {  
    var counter = 0;  
    return function () {counter += 1; return counter}  
})();  
  
add();  
add();  
add();

**17.Object.freeze and object.seal**

**Object.freeze**allows us to freeze an object so that existing properties cannot be modified.var marks = {physics: 98, maths:95, chemistry: 91};  
finalizedMarks = Object.freeze(marks);  
finalizedMarks["physics"] = 86; // throws error in strict mode  
console.log(marks); // {physics: 98, maths: 95, chemistry: 91}

Object.isFrozen(finalizedMarks); // returns true

**Object.seal**is slightly different from the **freeze**. It allows configurable properties but won’t allow new property addition or deletion or properties.  
var marks = {physics: 98, maths:95, chemistry: 91};  
Object.seal(marks);

delete marks.chemistry; // returns false as operation failed  
marks.physics = 95; // Works!  
marks.greek = 86; // Will not add a new property

Object.isSealed(marks); // returns true

18. CallBack function

A callback function is called at the completion of a given task. This allows other code to be run in the meantime and prevents any blocking.

To make it nonblocking state we use callback

//mongodb connectivity using callback

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url,(function(err, db) {

db.collection('Employee').updateOne({

"EmployeeName": "Martin"

}, {

$set: {

"EmployeeName": "Mohan"

}

});

});

$("#btn\_1").click(function() {

alert("Btn 1 Clicked");

});

that is passed to another function (let’s call this other function “otherFunction”) as a parameter, and the callback function is called (or executed) inside the otherFunction.

**19.Promise**

During the development lifecycle, there may be an instance where you would need to nest multiple callback functions together. This can get kind of messy and difficult to maintain at a certain point in time. In short, a promise is an enhancement to callbacks that looks towards alleviating these problems.

//mongodb connectivity using promise

var Promise = require('promise');

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url)

.then(function(err, db) {

db.collection('Employee').updateOne({

"EmployeeName": "Martin"

}, {

$set: {

"EmployeeName": "Mohan"

}

});

});

**It is asynch.**let promise = new Promise(function(resolve, reject) {

setTimeout(() => reject(new Error("Whoops!")), 1000);

});

// reject runs the second function in .then

promise.then(

result => alert(result), // doesn't run

error => alert(error) // shows "Error: Whoops!" after 1 second

);

When the executor finishes the job, it should call one of the functions that it gets as arguments:

* resolve(value) — to indicate that the job finished successfully:
  + sets state to "fulfilled",
  + sets result to value.
* reject(error) — to indicate that an error occurred:
  + sets state to "rejected",
  + sets result to error.

#### 20. what is Hoisting ? Hoisting is a process of pushing the declared variables to the top of the program while running it. For Ex:

catName("Chloe");

function catName(name) {

console.log("My cat's name is " + name);

}

#### 21 what is nodeJs ?

#### Node.js is a server side scripting based on Google’s V8 JavaScript engine. It is used to build scalable programs especially web applications

**22. Why use Node.js?**

Node.js makes building scalable network programs easy. Some of its advantages include:

* It is generally fast
* It almost never blocks
* It offers a unified programming language and data type
* Everything is asynchronous

**23. What are “streams” in Node.js? Explain the different types of streams present in Node.js.**

Streams are objects that allow reading of data from the source and writing of data to the destination as a continuous process.

There are four types of streams.

* to facilitate the reading operation
* to facilitate the writing operation
* to facilitate both read and write operations
* is a form of Duplex stream that performs computations based on the available input

#### 24. What is an error-first callback?

Error-first callbacks are used to pass errors and data as well. You have to pass the error as the first parameter, and it has to be checked to see if something went wrong. Additional arguments are used to pass data.

#### fs.readFile(filePath, function(err, data) { if (err) { // handle the error, the return is important here // so execution stops here return console.log(err) } // use the data object console.log(data) })

#### 25. How can you avoid callback hells?

There are lots of ways to solve the issue of callback hells:  
1.modularization: break callbacks into independent functions  
2.use generators with Promises

#### 26. What's a stub? Name a use case!

Stubs are functions/programs that simulate the behaviors of components/modules. Stubs provide canned answers to function calls made during test cases.

An example can be writing a file, without actually doing so.

**var** fs = require('fs')

**var** writeFileStub = sinon.stub(fs, 'writeFile', **function** (path, data, cb) {

**return** cb(null)

})

expect(writeFileStub).to.be.called

writeFileStub.restore()