**https://www.techbeamers.com/javascript-interview-questions-answers/**

[**https://www.guru99.com/javascript-interview-questions-answers.html**](https://www.guru99.com/javascript-interview-questions-answers.html)

**How to create a JavaScript object?**

A JavaScript object is an entity having state and behavior (properties and method). Since JavaScript is an object-based language, it treats everything as an object.

JavaScript is a template-based language not class based. It supports to create the object directly, there is no need to define a class for this.

The JavaScript [**object**](https://www.code-sample.com/2015/06/object-in-javascript.html) is a collection of properties and the each property associated with the **name-value pairs**. The object can contain **any data types** (numbers, arrays, object etc.)

The example looks like,

Var myObject= {empId : “001”, empCode :”X0091”};

In the above example, here are two properties one is empId and other is empCode and its values are “001” and “X0091”.

**Types of creating an object (**There are two types**)**

1.      Object literals

2.      Object constructor

**Object Literals**: This is the most common way to create an object with object literal and the example as given below.

The empty object initialized using object literal i.e.

var emptyObj= {};

This is an object with 4 items using object literal i.e.

var emptyObj ={

  empId:”Red”,

  empCode: “X0091”,

 empDetail : function(){

    alert(“Hi”);

};

};

**Object Constructor**: The second way to create object using object constructor and the constructor is a function used to initialize new object.

The example looks like,

Var obj = new Object();

Obj.empId=”001”;

Obj.empCode=”X0091”;

Obj.empAddressDetai = function(){

       Console.log(“Hi, I Anil”);

};

**Summary:**

There is no best way to create an object. It’s depending on your uses case. You can choose all one as per your case. All have some benefits.

The date objects inherit from **Date.prototype** and the array objects inherit from **Array.prototype** and also the Person objects inherit from **Person.prototype**.

Sometimes we have a requirement to add new **properties**and **methods** in the existing objects or constructors.

Example for add new properties to object constructors –

//Person

function Person(firstName, lastName, age) {

  this.firstName = firstName,

  this.lastName = lastName;

  this.age = age;

}

//Person prototype

Person.prototype.nationality = "Indian";

Example for add new methods objects constructors –

//Person Peroperty class

function Person(firstName, lastName, age) {

  this.firstName = firstName,

  this.lastName = lastName;

  this.age = age;

}

//Person prototype

Person.prototype.fullName = function() {

  return this.firstName + " " + this.lastName;

};

[What is Prototype in JavaScript?](https://www.code-sample.com/2015/06/prototype-in-javascript.html)  
[How to create prototype in JavaScript?](https://www.code-sample.com/2015/06/prototype-in-javascript.html)

The prototype is a fundamental concept of JavaScript and its must to known JavaScript developers and all the [JavaScript objects](https://www.code-sample.com/2015/06/prototype-in-javascript.html) have an object and its property called prototype and its used to add and the custom functions and property.

Every JavaScript function has a prototype property (by default this property is null), that is mainly used for implementing inheritance. We add methods and properties to a function’s prototype so that it becomes available to instances of that function

The example looks like,

**var** employee = **function** () {

*//This is a constructor function.*

}

*//Crate the instance of above constructor function and assign in a variable*

**var** empInstance = new employee();

empInstance.deportment = "IT";

console**.log**(empInstance.deportment);*//The output of above is IT.*

The example with prototype as given below.

**var** employee = **function** () { *//This is a constructor function.}*

**employee**.prototype.deportment = "IT";*//Now, for every instance employee will have a deportment.*

*//Crate the instance of above constructor function and assign in a variable*

**var** empInstance = new employee();

empInstance.deportment = "HR";

console**.log**(empInstance.deportment);*//The output of above is HR not IT.*

**What is ‘this’ in JavaScript?**

The 'this' keyword behaves a little differently in JavaScript compared to other languages.

In most of the other languages, '[this](https://www.code-sample.com/2015/06/what-is-this-in-javascript.html)' keyword is a reference to the current object instantiated by the classes and methods.

In the JavaScript [languages](https://www.code-sample.com/2015/06/what-is-this-in-javascript.html), 'this' keyword refers to the object which 'owns' the method, but it depends on how a function is called.

The examples in details as given below.

*//Global Scope in JavaScript*

*//In the below example, ‘this’ keyword refers to the global object.*

**window**.sms = "Hi, I am window object";

console**.log**(**window**.sms);

console**.log**(this.sms); *// Hi, I am window object.*

console**.log**(**window** === this); *// Its return true.*

*//Calling a Function in JavaScript*

*//In the below example, ‘this’ keyword remains the global object if we are calling a function.*

**window**.sms = "Hi, I am window object";

*// Creating a function*

**function** thisMethod() {

console**.log**(this.sms); *// Hi, I am window object.*

console**.log**(**window** === this); *//Its return true.*

}

*// Calling a function*

thisMethod();

*//Calling Object Methods in JavaScript*

*//In the below example, when we calling an object constructor or any of its methods,*

*//‘this’ keyword refers to the instance of the object.*

**window**.sms = "Hi, I am window object";

**function** objectTestMethod() {

this.sms = "Hi, I am a test object.";

**this**.method1 = **function** () {

console**.log**(this.sms); *// Hi, I am a test object.*

};

}

objectTestMethod.prototype. method2 = **function** () {

console**.log**(this.sms); *// Hi, I am a test object.*

};

*// Creating object and calling methods*

**var** v = new objectTestMethod();

v. method1();

v. method2();

#### 1. Global Scope.

A variable defined outside a function comes under the Global scope. Variables defined inside the Global scope are accessible from any part of the code. Let’s see an example.

#### 2. Local Scope.

Variables defined inside a function comes under the Local scope. Different functions can use a variable with the same name. It is because these variables are strictly bound to the function that defines it (each having different scopes) and is not accessible in other functions. Let’s see an example.

// Global Scope

function sampleFunction() {

// Local Scope #1

function sample2Function() {

// Local Scope #2

}

}

### Q-5. What Is Closure In JavaScript?

### Answer.

A closure is a JavaScript function defined inside another function. And that’s why it gets a special privilege to access three types of scope which are as follows.

* Internal Scope i.e. the variables defined between its curly brackets.
* Outer Function Scope i.e. the variables of the enclosing function.
* Global Scope i.e. variables defined as globals.

Please note that a closure can not only access the outer function variables but also see its parameters.

Here is a code example describing closure by adding a function inside another function.

function outerFunc(arg1, arg2) {

​var param = "I'm closure. ";

// Inner function accessing outer function variables and parameter​s

​function innerFunc() {

​ return arg1 + arg2 + " " + param;

}

​ ​return innerFunc();

}

​

outerFunc("arg1", "arg2");

While you create the JavaScript function within another function and the inner function  freely access all the variable of outer function. i.e.

### Q-6. Why Is “Self” Needed Instead Of “This” In JavaScript?

Inner functions in JavaScript have access to all of the variables defined in the outer function. However, “this” variable is an exception. Since the nested function is just a regular function and not an object method, it’s “this” refers to the global namespace. To make it more clear, let’s look at the following example.

var aProperty = 'global';

var myObject = {

outerFun: function() {

this.aProperty = 'local';

setTimeout(function() {

console.log(this.aProperty); // outputs 'global'

}, 1);

}

};

Thus, we see that inside “setTimeout” function, “this” refers to the global object. We need a way to get a reference to the object, that is available inside the nested function. We assign the object from “this”, to another(non-special) variable, “self”. It is not a special variable and hence cannot be overwritten by other functions(like “this”). Thus on using “self” inside the inner function, we can refer to the local object. Following is the sample code.

var myObject = {

outerFun: function() {

var self = this;

this.aProperty = 'local';

setTimeout(function() {

console.log(self.aProperty); // outputs 'local'

}, 1);

}

};

### What Is An Anonymous Function And When Should You Use It?

Anonymous functions are functions that are dynamically declared at runtime. They’re called anonymous functions because they don’t have a name like normal functions.

We use function operator to declare an anonymous function, instead of the function declaration. Also, function operator can be used to create a new function, wherever it’s valid to put an expression. For example, we declare a new function to be supplied as an argument to a function call or to assign a property of another object.

var testFunction= function()

{

alert("Zoom! Zoom! Zoom!");

}

testFunction();

### What Is The Difference Between “==” And “===”?

The double equals (==) are used for check only value of its variables but triple equals (===) are used for check value and type as well of its variables.

1.  The double equal “==” is an [auto-type](https://www.code-sample.com/2014/06/difference-between-and-in-javascript.html) conversion and it checks only value not type.

2.  The triple equal “===” is not auto-type conversion and it check value and type both.﻿

### What Are JavaScript Data Types?

JavaScript supports three Primary, two Composite and two Special data types. Next, we list down the data types in each of the categories. 1) Primary Data Types – String,Number,Boolean 2) Composite Data Types – Object,Array 3) Special Data Types – Null, Undefined.

**What is variable typing?**

Variable typing is used to assign a number to a variable and then assign string to the same variable. Example is as follows:

i= 8;

i="john";

### Q-13. What Are The Different Ways To Create An Array In JavaScript?

There are two main ways to create an array in JavaScript.

#### 1. Using An Array Initializer (Array Literal).

The array initializer (array literal) syntax is simple. It is a comma-separated list of values in square brackets.

Let’s see some examples.

var myArray1 = [1,2,3,4,5] // an array with 5 elements

var myArray2 = [5] // an array with 1 element

var myArray3 = [true,'Hi',[7]] // element types need not be the same.

#### 2. Using The Array Constructor.

The Array constructor method has three different syntaxes. If we call the constructor with two or more arguments, it declares an array with array elements also initialized. If we provide only one argument to the Array constructor, it refers to the length of the new array with, elements not initialized. Lastly, the constructor without any argument creates an array with its length set to zero with elements not initialized.

Let’s see some examples.

var myArray4 = new Array(1,2,3,4,5) // an array with 5 elements

var myArray5 = new Array(20) // an empty array of length 20

var myArray6 = new Array() // an empty array of length 0

### What Are JavaScript Cookies?

Cookies are the small test files stored in a computer and it gets created when the user visits the websites to store information that they need. Example could be User Name details and shopping cart information from the previous visits.

A cookie is a piece of data which is sent from a website (that owns the requested web page) and gets stored locally by the browser at the user end. Cookies are needed because HTTP protocol which arranges for the transfer of web pages to your browser, is stateless. It means that HTTP has no way to keep track of the activities performed by the user at an earlier point in time. One way to resolve this issue is by using cookies. It contains the following data.

* A name-value pair containing the actual data.
* An expiry date after which the cookie is no longer valid.
* The domain and path of the server it should be sent to.

When a request arrives at the server for a web page that maintains a cookie, the server appends the cookie to the HTTP header to send it across. The server-side programs can then read out the information included in it and decide that you have the right to view the page or not and other user preferences.

Thus, every time you visit the site that maintains the cookies, your information is available there.

**51. What is the use of Push method in JavaScript?**

The push method is used to add or append one or more elements to the end of an Array. Using this method, we can append multiple elements by passing multiple arguments

**52. What is unshift method in JavaScript?**

Unshift method is like push method which works at the beginning of the array. This method is used to prepend one or more elements to the beginning of the array.

### How Will You Create A Cookie Using JavaScript?

The simplest way to create a cookie is to assign a string value to the <**document.cookie**> object.

document.cookie = "key1 = value1; key2 = value2; expires = date";

Here, “expires” attribute is optional. We have to provide a date or time value for this attribute.

If we provide a valid value for the date or time, then the cookie will expire at the given date or time and it will not be accessible after that.

**How To Read A Cookie Using JavaScript?**

To read a Cookie, we have to access the value of the <document.cookie> object. This <document.cookie> string maintains a list of <name = value> pairs that is separated with semicolons.

"name" is the name of a cookie and

"value" is its string value.

We use String <split()> function to break the <document.cookie> object to sub-strings. Each of these sub-strings contains a key-value pair which represents the information related to a Cookie.

**How To Delete A Cookie Using JavaScript?**

To delete a Cookie, we have to set its expiry date to a time that occurred in the past. If attempts are made to read a deleted Cookie then, nothing is returned.

**Why Should You Not Prefer To Use Global Variables In JavaScript And How Can You Prevent It?**

The principal issue in using a global variable is that someone else can create another variable

with the same name. And you may not know it until the duplicate could overwrite the value of your variable.

To avoid using globals, follow any of the following approaches.

1. Create a single global variable that holds all your other variables.

var myGlobalList = {}; myGlobalList.first = "test";

2. Enclose all of your code in a self-executing method/function so that any variable declared inside remain in the function scope.

(function(){

var test = "myvar";

})()

What Are The Different Objects Used In JavaScript?

**Window Object. -**It is the topmost object in the hierarchy. It refers to the content area of the browser window that consists of HTML documents. Each frame is also a window that has some actions inside it.

**Document Object.-**A Document object represents the HTML document that the window will display. It has various properties that refer to other objects, which allow access to and modification of content in the document.

#### Form Object. - A form object is used to take user data as input for processing. It corresponds to an HTML input form constructed with the ****<FORM>…</FORM>**** tag.

### How Will You Replace All Occurrences Of A String In JavaScript?

We can use String’s **<replace()>** method to substitute any string. There are the following two ways to use this method.

#### Pass The Input String As A Regular Expression.

str = "ghktestlllltest-sdds"

str = str.replace(/test/g, '');

alert(str)

#### Use RegExp Class To Create A Regular Expression And Pass It.

String.prototype.replaceAll = function(find, replace) {

var target = this;

return target.replace(new RegExp(find, 'g'), replace);

};

str = "ghktestlllltest-sdds"

str = str.replaceAll('test', '');

alert(str)

[**What is Hoisted in JavaScript?**](https://www.code-sample.com/2015/06/hoisted-in-javascript.html)

In the JavaScript, the variables can be used before declared, this kinds of mechanism is called [Hoisted](https://www.code-sample.com/2015/06/hoisted-in-javascript.html). It's a default behavior of [JavaScript](https://www.code-sample.com/2015/06/hoisted-in-javascript.html).

You can easily understanding  in the below example in detail.

//The variable declaration look like.

var emp;

//The variable initialization look like.

emp = "Anil Singh";

var emp; //The declaration of emp is hoisted but the value of emp is  undefined.

emp = 10; //The Assignment still occurs where we intended (The value of emp is 10)

function getEmp() {

    var emp; //The declaration of a different variable name emp is hoisted but the value of emp is  undefined.

    console.log(emp); //The output is undefined

    emp = 20; //The assignment values is 20.

    console.log(emp); //The output is 20.

}

getEmp();

### How Does The <Array()> Differ From <[]> While Creating A JavaScript Array?

Both the **<Array()>** and **<[]>** works almost the same in JavaScript.

If we use them as is (i.e. without any argument) to create an array object, then they will result in an array object of zero length. Also, if we pass a string or a list of strings as arguments, even then the result will be similar.

There is a difference, but there is no difference in that example.

Using the more verbose method: new Array() does have one extra option in the parameters: if you pass a number to the constructor, you will get an array of that length:

x = new Array(5);

alert(x.length); // 5

To illustrate the different ways to create an array:

var a = [], // these are the same

b = new Array(), // a and b are arrays with length 0

c = ['foo', 'bar'], // these are the same

d = new Array('foo', 'bar'), // c and d are arrays with 2 strings

**How to empty an array in JavaScript?**

var userList =  ['anil','kumar','singh','kushinagar','up','india'];

1) userList =[]; 2) userList.length=0; 3)  userList.splice(0, userList.length);

# How To Convert a string to Lowercase

let testStr ='This is test string';

testStr = testStr.toLowerCase();

# How To convert JSON Object to String?

let myObject=['A','B','C','D']

JSON.stringify(myObject);

# How To modify the URL of page without reloading the page?

To modify the URL of page without reloading the page using the “push State” function.

Example –

window.history.pushState('page1', 'This is page1 Title', '/index.htm');

**What Is an Associative Array in JavaScript?**

**What is array?**   
Array is a collection of index items and it is a number indexes.  
  
Some of programming language support array as named indexes and the JavaScript not support the array as name indexes and its provide only number indexes but provide this feature using the associative array.

The array with name indexes are called associative array and the associative array is provide a ways to store the information.  
The number index array example as given below -

var users = new Object();

users["Name1"] = "Anil 1";

users["Name2"] = "Anil 2";

users["Age"] = 33;

alert(Object.keys(users).length); //output is 3.

var length = Object.keys(users).length;  // 3

The name index array example as given below -

var users = [];

users["Name1"] = "Anil 1";

users["Name2"] = "Anil 2";

users["Age"] = 33;

var length = users.length;         // users.length will return 0

var detail = users[0];             // users[0] will return undefined

**Where To Use the Associate Array?**

The **empName** as text type, **empAge** as number type and **enpDOB** as date type  
If we need to find the type of a column that time we can create the associate array i.e.

var empDetailType = new Array();

empDetailType["empName"] = "ANil";

empDetailType["empAge"] = 30;

empDetailType["empDOB"] = "10/03/1984";

console.log("Find the emp age type :" + empDetailType["empAge"]);

**Why it is not advised to use innerHTML in JavaScript?**

innerHTML content is refreshed every time and thus is slower. There is no scope for validation in innerHTML and, therefore, it is easier to insert rouge code in the document and, thus, make the web page unstable.

**How are JavaScript and ECMA Script related?**

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

### Explain “use strict” ?

“use strict” is a javascript directive that is introduced in Es5. The purpose of using “use strict” directive is to enforce the code is executed in strict mode. In strict mode we can’t use a variable without declaring it. “use strict” is ignored by earlier versions of Javascript.

Strict Mode imposes a layer of constraint on JavaScript. It provides following enhancements.

* JavaScript will throw an error if we try to use the elements of a deprecated language.
* To use a variable, it has become mandatory to declare it.
* It disallows duplicate property and parameter names.
* JavaScript will throw an error if we try to assign a value to a read-only property.
* It decreases the global namespace pollution.

To enable strict mode, we have to add, “use strict” directive to the code. The physical location of the “strict” directive determines its scope. If used at the beginning of the js file, its scope is global. However, if we declare strict mode at the first line in the function block, its scope restricts to that function only.

**What is the 'Strict' mode in JavaScript and how can it be enabled?**

Strict Mode adds certain compulsions to JavaScript. Under the strict mode, JavaScript shows errors for a piece of codes, which did not show an error before, but might be problematic and potentially unsafe. Strict mode also solves some mistakes that hamper the JavaScript engines to work efficiently.

Strict mode can be enabled by adding the string literal "use strict" above the file. This can be illustrated by the given example:

function myfunction() {

"use strict";

var v = "This is a strict mode function";

}

**Define event bubbling?**

JavaScript allows DOM elements to be nested inside each other. In such a case, if the handler of the child is clicked, the handler of parent will also work as if it were clicked too

Bubbling just works like the bubbles, the event gets handled by the innermost element and then propagated to the outer element. suppose the click event did occur in the ‘li’ element in bubbling model the event will be handled first by ‘li’ then by ‘ul’ and at last by ‘div’ element. - <div><ul><li></li></ul></div>

**Explain function hoisting in JavaScript?**

JavaScript’s default behavior that allows moving declarations to the top is called Hoisting.

The 2 ways of creating functions in JavaScript are Function Declaration and Function Expression. Let’s find out more about these:

Function Declaration- A function with the specific parameters is known as function declarations. To create a variable in JavaScript is called declarations.

hoisted (); // logs "foo"

function hoisted() { console.log('foo');}

Function Expression- When a function is created by using an expression it is called function expression.

notHoisted(); // TypeError: notHoisted is not a function

var notHoisted = function() { console.log('bar');};

**Explain Arrow functions?**

An arrow function is consise and short way to write function expressions in Es6 or above .Arrow functions

cannot be used as constructors and also does not supports this, arguments, super, or new.target keywords.

It is best suited for non-method functions .In general an arrow function

looks like const function\_name= ()=>{}

const greet=()=>{console.log('hello');}

greet();

**3. What are exports and imports?**

Imports and exports helps us to write modular javascript code. Using Imports and exports we can split our code in to multiple files. Imports allows to take only some specific variables or methods of a file. We can import methods or variables that are exported by a module.See the below example for more detail.

//index.js

import name,age from './person';

console.log(name);

console.log(age);

//person.js

let name ='Sharad',

occupation='developer'

age =26;

export { name, age};

## Question 17. What is the difference between declaring a function in the formats listed below?

var foo = function() {

// Some code

};

function bar () {

// Some code

};

The main difference is function foo is defined at run-time and is called function expression, whereas function bar is defined at parse time and is called function statement. To understand in better, let's see below code :

// Run-Time function declaration

<script>

foo(); // Call foo function here, It will give an error

var foo = function() {

console.log("Hi I am inside Foo");

};

</script>

// Parse-Time function declaration

<script>

bar(); // Call bar function here, It will not give an Error

function bar() {

console.log("Hi I am inside Foo");};</script>

## What’s the difference between typeof and instanceof?

typeof is an operator that returns a string with the type of whatever you pass.

The typeof operator checks if a value belongs to one of the seven basic types: number, string, boolean, object, function, undefined or Symbol.

typeof(null) will return object.

instanceof is much more intelligent: it works on the level of prototypes. In particular, it tests to see if the right operand appears anywhere in the prototype chain of the left. instanceof doesn’t work with primitive types. It instanceof operator checks the current object and returns true if the object is of the specified type, for example:

var dog = new Animal();

dog instanceof Animal; // Output : true

Here dog instanceof Animal is true since dog inherits from Animal.prototype

var name = new String("xyz");

name instanceof String; // Output : true

## What is the difference between a method and a function in javascript?

In JS, that difference is quite subtle. A function is a piece of code that is called by name and function itself not associated with any object and not defined inside any object. It can be passed data to operate on (i.e. parameter) and can optionally return data (the return value).

// Function statement

function myFunc() {

// Do some stuff;

}

// Calling the function

myFunc();

Here myFunc() function call is not associated with object hence not invoked through any object.

A function can take a form of immediately invoked function expression (IIFE):

// Anonymous Self-invoking Function

(function() {

// Do some stuff;

})();

Finally there are also arrow functions:

const myFunc = arg => {

console.log("hello", arg)

}

A method is a piece of code that is called by its name and that is associated with the object. Methods are functions. When you call a method like this obj1.myMethod(), the reference to obj1 gets assigned (bound) to this variable. In other words, the value of this will be obj1 inside myMethod.

Here are some examples of methods:

##### Example 1

var obj1 = {

attribute: "xyz",

myMethod: function () { // Method

console.log(this.attribute);

}

};

// Call the method

obj1.myMethod();

Here obj1 is an object and myMethod is a method which is associated with obj1.

##### Example 2

In ES6 we have classes. There the methods will look like this:

class MyAwesomeClass {

myMethod() {

console.log("hi there");

}

}

const obj1 = new MyAwesomeClass();

obj1.myMethod();

Understand: the method is not some kind of special type of a function, and it's not about how you declare a function. It's the way we **call** a function. Look at that:

var obj1 = {

prop1: "buddy"

};

var myFunc = function () {

console.log("Hi there", this);

};

// let's call myFunc as a function:

myFunc(); // will output "Hi there undefined" or "Hi there Window"

obj1.myMethod = myFunc;

//now we're calling myFunc as a method of obj1, so this will point to obj1

obj1.myMethod(); // will print "Hi there" following with obj1.

## What are the ways of creating objects in JavaScript ?

#### Method 1: Function based

This method is useful if we want to create several similar objects. In the code sample below, we wrote the function Employeeand used it as a constructor by calling it with the new operator.

function Employee(fName, lName, age, salary){

this.firstName = fName;

this.lastName = lName;

this.age = age;

this.salary = salary;

}

// Creating multiple object which have similar property but diff value assigned to object property.

var employee1 = new Employee('John', 'Moto', 24, '5000$');

var employee1 = new Employee('Ryan', 'Jor', 26, '3000$');

var employee1 = new Employee('Andre', 'Salt', 26, '4000$');

#### Method 2: Object Literal

Object Literal is best way to create an object and this is used frequently. Below is code sample for create employee object which contains property as well as method.

var employee = {

name : 'Nishant',

salary : 245678,

getName : function(){

return this.name;

}

}

The code sample below is Nested Object Literal, Here address is an object inside employee object.

var employee = {

name : 'Nishant',

salary : 245678,

address : {

addressLine1 : 'BITS Pilani',

addressLine2 : 'Vidya Vihar'.

phoneNumber: {

workPhone: 7098889765,

homePhone: 1234567898

}

}}

#### Method 3: From Object using new keyword

In the code below, a sample object has been created using Object's constructor function.

var employee = new Object(); // Created employee object using new keywords and Object()

employee.name = 'Nishant';

employee.getName = function(){

return this.name;

}

#### Method 4:\*\* Using Object.create

Object.create(obj) will create a new object and set the obj as its prototype. It’s a modern way to create objects that inherit properties from other objects. Object.create function doesn’t run the constructor. You can use Object.create(null) when you don’t want your object to inherit the properties of Object.

**. What is constructor property?**

Constructor property of an object maintains a reference to its creator function.

Example:

Let us checkout an example by creating a student object and calling the constructor property on it.

function Student( name, mark){  
this. name = name; this. mark =mark;  
}  
var student 1 = new Student (’sandeep’, 123);  
console.log (student1.constructor);  
function Student(name, mark){  
this.name = name;  
this.mark = mark;  
}  
var student1 = new Student(’sandeep’ ,123);  
console.log(student1.constructor);

Checkout the following screen shot for above code in chrome console. The console log is printing the referenced function by student1 object.

**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.

1. Global Scope
2. Local Scope

**What is the different type of features of JavaScript?**

JavaScript has five types of features -

* It is an open platform.
* It is a lightweight programming language which is easily interpreted.
* It is a programming language which is well integrated and complements well with Java.
* It is designed in such a way which is supported for creating every type of network-centric application.
* JavaScript is also integrated well with HTML and also compliments with it.

**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.

**Explain how to detect the operating system on the client machine?**

In order to detect the operating system on the client machine, the navigator.platform string (property) should be used.

**What do mean by NULL in Javascript?**

The NULL value is used to represent no value or no object. It implies no object or null string, no valid boolean value, no number and no array object.

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

The delete keyword is used to delete the property as well as its value.

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

**What are the different types of errors in JavaScript?**

There are three types of errors:

* **Load time errors**: Errors which come up when loading a web page like improper syntax errors are known as Load time errors and it generates the errors dynamically.
* **Run time errors**: Errors that come due to misuse of the command inside the HTML language.
* **Logical Errors**: These are the errors that occur due to the bad logic performed on a function which is having different operation.