Top UI-DEVELOPER Interview Questions & Answers

**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

**2. Enumerate the differences between Java and JavaScript?**

Java is a complete programming language. In contrast, JavaScript is a coded program that can be introduced to HTML pages. These two languages are not at all inter-dependent and are designed for the different intent. Java is an object - oriented programming (OOPS) or structured programming language like C++ or C whereas JavaScript is a client-side scripting language.

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

Following are the JavaScript Data types:

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

**4. What is the use of isNaN function?**

isNan function returns true if the argument is not a number otherwise it is false.

**5. Between JavaScript and an ASP script, which is faster?**

JavaScript is faster. JavaScript is a client-side language and thus it does not need the assistance of the web server to execute. On the other hand, ASP is a server-side language and hence is always slower than JavaScript. Javascript now is also a server side language (nodejs).

**6. What is negative infinity?**

Negative Infinity is a number in JavaScript which can be derived by dividing negative number by zero.

**7. Is it possible to break JavaScript Code into several lines?**

Breaking within a string statement can be done by the use of a backslash, '\', at the end of the first line

Example:

document.write("This is \a program");

And if you change to a new line when not within a string statement, then javaScript ignores break in line.

Example:

var x=1, y=2,

z=

x+y;

The above code is perfectly fine, though not advisable as it hampers debugging.

**8. Which company developed JavaScript?**

Netscape is the software company who developed JavaScript.

**9. What are undeclared and undefined variables?**

Undeclared variables are those that do not exist in a program and are not declared. If the program tries to read the value of an undeclared variable, then a runtime error is encountered.

Undefined variables are those that are declared in the program but have not been given any value. If the program tries to read the value of an undefined variable, an undefined value is returned.

**10. Write the code for adding new elements dynamically?**

<script type="text/javascript">

function addNode()

{

var newP = document.createElement("p");

var textNode = document.createTextNode(" This is a new text node");

newP.appendChild(textNode);

document.getElementById("firstP").appendChild(newP);

}

</script>

<p id="firstP">firstP<p>

**11. What are global variables? How are these variable declared and what are the problems associated with using them?**

Global variables are those that are available throughout the length of the code, that is, these have no scope. The var keyword is used to declare a local variable or object. If the var keyword is omitted, a global variable is declared.

Example:

// Declare a global globalVariable = "Test";

The problems that are faced by using global variables are the clash of variable names of local and global scope. Also, it is difficult to debug and test the code that relies on global variables.

**12. What is a prompt box?**

A prompt box is a box which allows the user to enter input by providing a text box. Label and box will be provided to enter the text or number.

**13. What is 'this' keyword in JavaScript?**

'This' keyword refers to the object from where it was called.

**14. Explain the working of timers in JavaScript? Also elucidate the drawbacks of using the timer, if any?**

Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functions **setTimeout, setInterval**and**clearInterval**.

The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay. The **setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled. The **clearInterval(id)**function instructs the timer to stop.

Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

**15. Which symbol is used for comments in Javascript?**

// for Single line comments and

/\* Multi

Line

Comment

\*/

**16. 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.

**17. What is === operator?**

=== is called as strict equality operator which returns true when the two operands are having the same value without any type conversion.

**18. Explain how can you submit a form using JavaScript?**

To submit a form using JavaScript use document.form[0].submit();

document.form[0].submit();

**19. Does JavaScript support automatic type conversion?**

Yes JavaScript does support automatic type conversion, it is the common way of type conversion used by JavaScript developers

**20. How can the style/class of an element be changed?**

It can be done in the following way:

document.getElementById("myText").style.fontSize = "20?;

or

document.getElementById("myText").className = "anyclass";

**21. Explain how to read and write a file using JavaScript?**

There are two ways to read and write a file using JavaScript

* Using JavaScript extensions
* Using a web page and Active X objects

**22. What are all the looping structures in JavaScript?**

Following are looping structures in Javascript:

* For
* While
* do-while loops

**23. What is called Variable typing in Javascript?**

Variable typing is used to assign a number to a variable and the same variable can be assigned to a string.

Example

i = 10;

i = "string";

This is called variable typing.

**24. How can you convert the string of any base to integer in JavaScript?**

The parseInt() function is used to convert numbers between different bases. parseInt() takes the string to be converted as its first parameter, and the second parameter is the base of the given string.

In order to convert 4F (of base 16) to integer, the code used will be -

parseInt ("4F", 16);

**25. Explain the difference between "==" and "==="?**

"==" checks only for equality in value whereas "===" is a stricter equality test and returns false if either the value or the type of the two variables are different.

**26. What would be the result of 3+2+"7"?**

Since 3 and 2 are integers, they will be added numerically. And since 7 is a string, its concatenation will be done. So the result would be 57.

**27. 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.

**28. 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.

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

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

Example

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

delete student.age;

**30. What is an undefined value in JavaScript?**

Undefined value means the

* Variable used in the code doesn't exist
* Variable is not assigned to any value
* Property doesn't exist

**31. What are all the types of Pop up boxes available in JavaScript?**

* Alert
* Confirm and
* Prompt

**32. What is the use of Void(0)?**

Void(0) is used to prevent the page from refreshing and parameter "zero" is passed while calling.

Void(0) is used to call another method without refreshing the page.

**33. How can a page be forced to load another page in JavaScript?**

The following code has to be inserted to achieve the desired effect:

<script language="JavaScript" type="text/javascript" >

<!-- location.href="http://newhost/newpath/newfile.html"; //--></script>

**34. What is the data type of variables of in JavaScript?**

All variables in the JavaScript are object data types.

**35. What is the difference between an alert box and a confirmation box?**

An alert box displays only one button which is the OK button.

But a Confirmation box displays two buttons namely OK and cancel.

**36. What are escape characters?**

Escape characters (Backslash) is used when working with special characters like single quotes, double quotes, apostrophes and ampersands. Place backslash before the characters to make it display.

Example:

document.write "I m a "good" boy"

document.write "I m a \"good\" boy"

**37. 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.

**38. Explain what is pop()method in JavaScript?**

The pop() method is similar as the shift() method but the difference is that the Shift method works at the start of the array. Also the pop() method take the last element off of the given array and returns it. The array on which is called is then altered.

Example:

var cloths = ["Shirt", "Pant", "TShirt"];

cloths.pop();

//Now cloth becomes Shirt,Pant

**39. Whether JavaScript has concept level scope?**

No. JavaScript does not have concept level scope. The variable declared inside the function has scope inside the function.

**40. Mention what is the disadvantage of using innerHTML in JavaScript?**

If you use innerHTML in JavaScript the disadvantage is

* Content is replaced everywhere
* We cannot use like "appending to innerHTML"
* Even if you use +=like "innerHTML = innerHTML + 'html'" still the old content is replaced by html
* The entire innerHTML content is re-parsed and build into elements, therefore its much slower
* The innerHTML does not provide validation and therefore we can potentially insert valid and broken HTML in the document and break it

**41. What is break and continue statements?**

Break statement exits from the current loop.

Continue statement continues with next statement of the loop.

**42. What are the two basic groups of dataypes in JavaScript?**

They are as –

* Primitive
* Reference types.

Primitive types are number and Boolean data types. Reference types are more complex types like strings and dates.

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

Generic objects can be created as:

var I = new object();

**44. What is the use of type of operator?**

'Typeof' is an operator which is used to return a string description of the type of a variable.

**45. Which keywords are used to handle exceptions?**

Try… Catch---finally is used to handle exceptions in the JavaScript

Try{

Code

}

Catch(exp){

Code to throw an exception

}

Finally{

Code runs either it finishes successfully or after catch

}

**46. Which keyword is used to print the text in the screen?**

document.write("Welcome") is used to print the text – Welcome in the screen.

**47. What is the use of blur function?**

Blur function is used to remove the focus from the specified object.

**48. 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";

**49. How to find operating system in the client machine using JavaScript?**

The '**Navigator.appversion'** is used to find the name of the operating system in the client machine.

**50. 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.

**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.

**53. What is the difference between JavaScript and Jscript?**

Both are almost similar. JavaScript is developed by Netscape and Jscript was developed by Microsoft .

**54. How are object properties assigned?**

Properties are assigned to objects in the following way -

obj["class"] = 12;

or

obj.class = 12;

**55. 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";

}

**56. What is the way to get the status of a CheckBox?**

The status can be acquired as follows -

alert(document.getElementById('checkbox1').checked);

If the CheckBox will be checked, this alert will return TRUE.

**57. How can the OS of the client machine be detected?**

The navigator.appVersion string can be used to detect the operating system on the client machine.

**58. Explain window.onload and onDocumentReady?**

The onload function is not run until all the information on the page is loaded. This leads to a substantial delay before any code is executed.

onDocumentReady loads the code just after the DOM is loaded. This allows early manipulation of the code.

**59. How will you explain closures in JavaScript? When are they used?**

Closure is a locally declared variable related to a function which stays in memory when the function has returned.

For example:

function greet(message) {

console.log(message);

}

function greeter(name, age) {

return name + " says howdy!! He is " + age + " years old";

}

// Generate the message

var message = greeter("James", 23);

// Pass it explicitly to greet

greet(message);

This function can be better represented by using closures

function greeter(name, age) {

var message = name + " says howdy!! He is " + age + " years old";

return function greet() {

console.log(message);

};

}

// Generate the closure

var JamesGreeter = greeter("James", 23);

// Use the closure

JamesGreeter();

**60. How can a value be appended to an array?**

A value can be appended to an array in the given manner -

arr[arr.length] = value;

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

The for-in loop is used to loop through the properties of an object.

The syntax for the for-in loop is -

for (variable name in object){

statement or block to execute

}

In each repetition, one property from the object is associated to the variable name, and the loop is continued till all the properties of the object are depleted.

**62. Describe the properties of an anonymous function in JavaScript?**

A function that is declared without any named identifier is known as an anonymous function. In general, an anonymous function is inaccessible after its declaration.

Anonymous function declaration -

var anon = function() {

alert('I am anonymous');

};

anon();

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

The function .call() and .apply() are very similar in their usage except a little difference. .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.

The basic difference between .call() and .apply() is in the way arguments are passed to the function. Their usage can be illustrated by the given example.

var someObject = {

myProperty : 'Foo',

myMethod : function(prefix, postfix) {

alert(prefix + this.myProperty + postfix);

}

};

someObject.myMethod('<', '>'); // alerts '<Foo>'

var someOtherObject = {

myProperty : 'Bar'

};

someObject.myMethod.call(someOtherObject, '<', '>'); // alerts '<Bar>'

someObject.myMethod.apply(someOtherObject, ['<', '>']); // alerts '<Bar>'

**64. 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.

**65. Is JavaScript case sensitive? Give an example?**

Yes, JavaScript is case sensitive. For example, a function parseInt is not same as the function Parseint.

**66. What boolean operators can be used in JavaScript?**

The 'And' Operator (&&), 'Or' Operator (||) and the 'Not' Operator (!) can be used in JavaScript.

\*Operators are without the parenthesis.

**67. How can a particular frame be targeted, from a hyperlink, in JavaScript?**

This can be done by including the name of the required frame in the hyperlink using the 'target' attribute.

<a href="/newpage.htm" target="newframe">>New Page</a>

**68. What is the role of break and continue statements?**

Break statement is used to come out of the current loop while the continue statement continues the current loop with a new recurrence.

**69. Write the point of difference between web-garden and a web-farm?**

Both web-garden and web-farm are web hosting systems. The only difference is that web-garden is a setup that includes many processors in a single server while web-farm is a larger setup that uses more than one server.

**70. How are object properties assigned?**

Assigning properties to objects is done in the same way as a value is assigned to a variable. For example, a form object's action value is assigned as 'submit' in the following manner - Document.form.action="submit"

**71. What is the method for reading and writing a file in JavaScript?**

This can be done by Using JavaScript extensions (runs from JavaScript Editor), example for opening of a file -

fh = fopen(getScriptPath(), 0);

**72. How are DOM utilized in JavaScript?**

DOM stands for Document Object Model and is responsible for how various objects in a document interact with each other. DOM is required for developing web pages, which includes objects like paragraph, links, etc. These objects can be operated to include actions like add or delete. DOM is also required to add extra capabilities to a web page. On top of that, the use of API gives an advantage over other existing models.

**73. How are event handlers utilized in JavaScript?**

Events are the actions that result from activities, such as clicking a link or filling a form, by the user. An event handler is required to manage proper execution of all these events. Event handlers are an extra attribute of the object. This attribute includes event's name and the action taken if the event takes place.

**74. Explain the role of deferred scripts in JavaScript?**

By default, the parsing of the HTML code, during page loading, is paused until the script has not stopped executing. It means, if the server is slow or the script is particularly heavy, then the webpage is displayed with a delay. While using Deferred, scripts delays execution of the script till the time HTML parser is running. This reduces the loading time of web pages and they get displayed faster.

**75. What are the various functional components in JavaScript?**

The different functional components in JavaScript are-

**First-class functions:** Functions in JavaScript are utilized as first class objects. This usually means that these functions can be passed as arguments to other functions, returned as values from other functions, assigned to variables or can also be stored in data structures.

**Nested functions:** The functions, which are defined inside other functions, are called Nested functions. They are called 'everytime' the main function is invoked.

**76. Write about the errors shown in JavaScript?**

JavaScript gives a message if it encounters an error. The recognized errors are -

* Load-time errors: The errors shown at the time of the page loading are counted under Load-time errors. These errors are encountered by the use of improper syntax, and thus are detected while the page is getting loaded.
* Run-time errors: This is the error that comes up while the program is running. It is caused by illegal operations, for example, division of a number by zero, or trying to access a non-existent area of the memory.
* Logic errors: It is caused by the use of syntactically correct code, which does not fulfill the required task. For example, an infinite loop.

**77. What are Screen objects?**

Screen objects are used to read the information from the client's screen. The properties of screen objects are -

* AvailHeight: Gives the height of client's screen
* AvailWidth: Gives the width of client's screen.
* ColorDepth: Gives the bit depth of images on the client's screen
* Height: Gives the total height of the client's screen, including the taskbar
* Width: Gives the total width of the client's screen, including the taskbar

**78. Explain the unshift() method ?**

This method is functional at the starting of the array, unlike the push(). It adds the desired number of elements to the top of an array. For example -

var name = [ "john" ];

name.unshift( "charlie" );

name.unshift( "joseph", "Jane" );

console.log(name);

The output is shown below:

[" joseph "," Jane ", " charlie ", " john "]

**79. 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.

For Example:

<script>

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

</script>

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

The unescape() function is very important as it decodes the coded string.

It works in the following way. For example:

<script>

document.write(unescape("Hello%3F%20How%20are%20you%21"));

</script>

Output: Hello? How are you!

**80. 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.

<script>

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

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

document.write(decodeURI(uri));

</script>

Output -

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

my test.asp?name=ståle&car=saab

**81. 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.

**82. What does the following statement declares?**

var myArray = [[[]]];

It declares a three dimensional array.

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

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

**84. What is namespacing in JavaScript and how is it used?**

Namespacing is used for grouping the desired functions, variables etc. under a unique name. It is a name that has been attached to the desired functions, objects and properties. This improves modularity in the coding and enables code reuse.

**85. How can JavaScript codes be hidden from old browsers that don't support JavaScript?**

For hiding JavaScript codes from old browsers:

Add "<!--" without the quotes in the code just after the <script> tag.

Add "//-->" without the quotes in the code just before the <script> tag.

Old browsers will now treat this JavaScript code as a long HTML comment. While, a browser that supports JavaScript, will take the "<!--" and "//-->" as one-line comments.

**JavaScript Unit Testing Frameworks**

**What is JavaScript?**

* JavaScript is a programming language which is defined as high level, dynamic and interpreted language used with HTML web applications
* JavaScript is also used for other than web documents such as PDFs and desktop widgets and became popular for server-side web application
* JavaScript is object-based script and follows the prototype

**JavaScript Unit Testing**

1. JavaScript Unit tests are usually run in the browser or on the frontend. Some JavaScript code is written for running a test for a page of the website or a single module of an application, and then this code is combined with HTML as an inline event handler
2. These unit tests are organized one by one in a Suite. Each and every suite contains number of tests separately performed for a separate module
3. Most importantly they don't conflict with any other module and runs with fewer dependencies on each other (some critical situation may cause dependencies)

**Challenges in JavaScript Unit Testing**

There are certain problems one can find while performing JavaScript Unit Testing such as;

1. Many other languages support unit testing in browsers, in the stable as well as in runtime environment but JavaScript can not
2. You can understand some system actions with other languages, but this is not the case with JavaScript
3. Some JavaScript are written for a web application may have multiple dependencies
4. JavaScript is good to use in combination with HTML and CSS rather than on the web
5. Difficulties with page rendering and DOM manipulation

To avoid such issues what you can do is;

1. Do not use global variables
2. Do not manipulate predefined objects
3. Design core functionalities based on library
4. Try to create small pieces of functionalities with lesser dependencies

**JavaScript Unit Testing Frameworks**

Sometimes you find the error message on your screen regarding such as 'Unable to load example.js' or any other JavaScript error regarding version control, these vulnerabilities comes under JavaScript Unit Testing.

We will review some tools and frameworks that are being used to perform JavaScript Unit Testing.

1. [JSUnit](http://www.jsunit.net/): JSUnit is an open source unit testing framework for client-side JavaScript running in the browser. This framework runs on different platforms and browsers. It fulfills server needs such as;
   1. Logging results in XML format
   2. Running JSUnit from JUnit and Ant and more than one or more browser and remote machines as well

2. [Unit.js](http://unitjs.com/): It is known as an open source assertion library running on browser and Node.js. It is extremely compatible with other JavaScript Unit Testing frameworks like Mocha, Karma, Jasmine, QUnit, Protractor, etc. Provides the full documented API of assertion list

3. [QUnit](https://qunitjs.com/): It is used for both client-side as well as server-side JavaScript Unit Testing. This Free framework is used for jQuery projects. It follows Common JS unit testing Specification for unit testing. It supports the Node Long-term Support Schedule.

4. [Jasmine](https://jasmine.github.io/): Jasmine is the behavior-driven development framework for JavaScript unit Testing. It is used for testing both synchronous and asynchronous JavaScript Code. It does not require DOM and comes with the easy syntax that can be Written for any test.

5. [Karma](https://karma-runner.github.io/2.0/index.html): Karma is an open source productive testing environment. Easy workflow control Running on the command line. Offers the freedom to write the tests with Jasmine, Mocha, and QUnit. You can run the test on real devices with easy debugging.

6. [Mocha](https://mochajs.org/): Mocha runs on Node.js and in the browser. Mocha performs asynchronous Testing in a simpler way. Provides accuracy and flexibility in reporting. Provides tremendous support of rich features such as test-specific timeouts, JavaScript APIs etc.

7. [Jest](https://facebook.github.io/jest/): Jest is used by Facebook so far to test all of the JavaScript code. It provides the 'zero-configuration' testing experience. Supports independent and non-interrupting running test without any conflict. Do not require any other setup configuration and libraries.

8. [AVA](https://github.com/avajs): AVA is simple JavaScript Unit Testing Framework. Tests are being run in parallel and serially. Parallel tests run without interrupting each other. AVA Supports asynchronous testing as well. AVA uses subprocesses to run the test.

**Summary**

* JavaScript Unit Testing may become tedious and tricky sometimes as it is performed for the front-end basically. One can use the JS libraries to for adding little ease. The challenge might become bigger as JavaScript is getting incorporated with Node.js and TypeScript.
* You should keep three things in mind while performing the test such as; The feature that needs to be tested, the final output and the expected output. Some tools and framework may help you in performing this task. Above mentioned tool lists is mentioned with most popular and useful frameworks used for JavaScript Unit Testing.
* More than these with upcoming challenges in performing testing there, some more powerful frameworks and tools may get evolved in future.
* **Explain what is Javascript? List some data types supported by Javascript?**

**Javascript**

Javascript is an object-oriented computer programming language commonly used to create interactive effects within web browsers.It is first used by the Netscape browser, that provides access to the HTML document object model (DOM), provides access to the browser object model (BOM). Javascript syntax looks a lot like java, c or c++ syntax.

Below is the list of data types supported by Javascript:-

* + Undefined
  + Null
  + Boolean
  + String
  + Symbol
  + Number
  + Object
* **Q2.**

**What close() does in Javascript?**

In Javascript close() method is used to close the current window. You must write window.close() to ensure that this command is associated with a window object and not some other JavaScript object.

* **Q3.**

**What is the difference between let and var?**

Both var and let are used for variable/ method declaration in javascript but the main difference between let and var is that **var** is function scoped whereas **let** is block scoped.

* **Q4.**

**Explain Closures in JavaScript?**

Closures are the combination of lexical environment and function within which the function was declared. This allows JavaScript programmers to write better, more creative, concise and expressive codes. The closure will consist of all the local variables that were in-scope when the closure was created.

Sure, closures appear to be complex and beyond the scope, but after you read this article, closures will be much more easy to understand and more simple for your everyday [JavaScript](https://www.onlineinterviewquestions.com/advanced-javascript-interview-questions/)programming tasks. JavaScript is  a very function-oriented language it gives the user freedom to use functions as the wish of the programmer.

* **Q5.**

**Explain JavaScript Event Delegation Model?**

In JavaScript, there is some cool stuff that makes it the best of all. One of them is Delegation Model. When capturing and bubbling, allow functions to implement one single handler to many elements at one particular time then that is called event delegation. Event delegation allows you to add event listeners to one parent instead of specified nodes. That particular listener analyzes bubbled events to find a match on the child elements. Many people think it to be complicated but in reality, it is very simple if one starts understanding it.

* **Q6.**

**Describe negative infinity in JavaScript?**

NEGATIVE\_INFINITY property represents negative infinity and is a number in javascript, which is derived by ‘dividing negative number by zero’. It can be better understood as a number that is lower than any other number. Its properties are as follows:  
– A number of objects need not to be created to access this static property.  
– The value of negative infinity is the same as the negative value of the infinity property of the global object.

The values behave differently than the mathematical infinity:

* + Any positive value, including POSITIVE\_INFINITY, multiplied by NEGATIVE\_INFINITY is NEGATIVE\_INFINITY.
  + Any negative value, including NEGATIVE\_INFINITY, multiplied by NEGATIVE\_INFINITY is POSITIVE\_INFINITY.
  + Zero multiplied by NEGATIVE\_INFINITY is NaN.
  + NaN multiplied by NEGATIVE\_INFINITY is NaN.
  + NEGATIVE\_INFINITY, divided by any negative value except NEGATIVE\_INFINITY, is POSITIVE\_INFINITY.
  + NEGATIVE\_INFINITY, divided by any positive value except POSITIVE\_INFINITY, is NEGATIVE\_INFINITY.
  + NEGATIVE\_INFINITY, divided by either NEGATIVE\_INFINITY or POSITIVE\_INFINITY, is NaN.
  + Any number divided by NEGATIVE\_INFINITY is zero.

**Q7.**

**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.

**e.g:**

hoisted(); // logs "foo"

function hoisted() {

 console.log('foo');

}

**Function Expression**

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

e.g:

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

var notHoisted = function() {

  console.log('bar');

};

**Q8.**

**What is the use of let & const?**

In modern javascript let & const are different ways of creating variables.Earlier in javascript, we use the var keyword for creating variables. let & const keyword is introduced in version ES6 with the vision of creating two different types of variables in javascript one is immutable and other is mutable.const: It is used to create an immutable variable. Immutable variables are variables whose value is never changed in the complete life cycle of the program.  
let: let is used to create a mutable variable. Mutable variables are normal variables like var that can be changed any number of time.

**Q9.**

**Explain Arrow functions?**

An arrow function is a 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();

**Q10.**

**What are exports and imports?**

Imports and exports help us to write modular javascript code. Using Imports and exports we can split our code into multiple files. Imports allow taking 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};

**Q11.**

**What is difference between module.exports and export?**

The module is a plain JavaScript object with an exports property. Exports is a plain JavaScript variable that happens to be set to module.exports. At the end of your file, node.js will basically ‘return’ module.exports to the require function. A simplified way to view a JS file in Node could be this:

var module = { exports: {} };

var exports = module.exports;

// your code

return module.exports;

If you set a property on exports, like exports.a = 9;, that will set module.exports.a as well because objects are passed around as references in JavaScript, which means that if you set multiple variables to the same object, they are all the same object; so then exports and module.exports are the same objects.  
But if you set exports to something new, it will no longer be set to module.exports, so exports and module.exports are no longer the same objects.

**Q12.**

**How to import all exports of a file as an object.**

import \* as object name from ‘./file.js’ is used to import all exported members as an object. You can simply access the exported variables or methods using dot (.) operator of the object.

Example:

objectname.member1;

objectname.member2;

objectname.memberfunc();

**Q13.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.

**Q14.**

**In Javascript are calculations with fractional numbers guaranteed to be precise?**

NO, calculations with fractional numbers are not guaranteed to be precise in Javascript

**Q15.**

**List the comparison operators supported by Javascript?**

Javascript supports below comparison operators

* + > Greater than
  + < Less than
  + <= Less than or equal to
  + >= Greater than or equal to
  + == Equal to
  + != Not Equal to
  + === Equal to with datatype check
  + !== Not equal to with datatype check

**Q16.How do you declare variables in Javascript?**

In Javascript variable are declared using the var keyword.A variable must begin with A **letter**, **$** or \_.

**eg.** var myVar=”Online Interview Questions”;

**PS:** All variables in Javascript are Case sensitive.

**Q17.What will happen if an infinite while loop is run in Javascript?**

The program will crash the browser.

**Q18.List HTML DOM mouse events?**

HTML DOM mouse events

* + onclick
  + ondblclick
  + mousemove
  + mousedown
  + mouseover
  + mouseout
  + mouseup

**Q19.How to get the last index of a string in Javascript?**

**string.length-1** is used to get the last index of a string in Javascript

**Example Usage:-**

var myString="JavascriptQuestions";

console.log(myString.length-1);

**Q20.How to get the primitive value of a string in Javascript?**

In Javascript **valueOf()** method is used to get the primitive value of a string.

**Example Usage:**

var myVar= "Hi!"

console.log(myVar.valueOf())

**Q21.What are the primitive data types in JavaScript?**

A primitive is a basic data type that’s not built out of other data types. It can only represent one single value. All primitives are built-in data types by necessity, (the compiler has to know about them,) but not all built-in data types are primitives.

In JavaScript there are 5 primitive data types are available they are **undefined**, **null**, **boolean**, **string** and **number** are available.Everything else in Javascript is an object.

**Q22.Explain Event bubbling and Event Capturing in JavaScript?**

**Event Capture and Bubbling**: In HTML DOM API there are two ways of event propagation and determines the order in which event will be received. The two ways are Event Bubbling and Event Capturing. The first method event bubbling directs the event to its intended target, and the second is called event capture in which the event goes down to the element.

**Event Capture**

The capture procedure is rarely used but when it’s used it proves to be very helpful. This process is also called ‘trickling’. In this process, the event is captured first by the outermost element and then propagated to the innermost element. For example:

From the above example, suppose the click event did occur in the ‘li’ element, in that case capturing event it will be first handled ‘div’, then ‘ul’ and at last the target element will be hit that is ‘li’

**Event Bubbling**

Bubbling just works like the bubbles, the event gets handled by the innermost element and then propagated to the outer element.

From the above example, 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.

**Q23.What does the instanceof operator do?**

In Javascript **instanceof** operator checks whether the object is an instance of a class or not:

**Example Usage**

Square.prototype = new Square();

console.log(sq instanceof Square); // true

**Q24.What is Javascript BOM?**

BOM stands for “Browser Object Modal” that allows Javascript to ‘talk’ to the browser, no standards, modern browsers implement similar BOMS – window, screen, location, history, navigator, timing, cookies.

**Q25.What are different types of Popup boxes available in Javascript?**

In Javascript there are 3 types of Popup Boxes are available, they are

* + Alert
  + Confirm
  + Prompt

**Q26.**

**How can you create an array in Javascript?**

There are 3 different ways to create an array in Javascript. They are

* + By array literal  
    **usage:**

var myArray=[value1,value2...valueN];

* + By creating instance of.Array  
    **usage:**

var myArray=new Array();

* + By usingan Array constructor  
    **usage:**

var myArray=new Array('value1','value2',...,'valueN');

**Q27.**

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

**Strict mode** is a way to introduce better error-checking into your code. When you use strict mode, you cannot, for example, use implicitly declared variables, or assign a value to a read-only property, or add a property to an object that is not extensible.

You can enable strict mode by adding **“use strict”**; at the beginning of a file, a program, or a function. This kind of declaration is known as a directive prologue. The scope of a strict mode declaration depends on its context. If it is declared in a global context (outside the scope of a function), all the code in the program is in strict mode. If it is declared in a function, all the code in the function is in strict mode.

**Q28.How to calculate Fibonacci numbers in JavaScript?**

**Calculating Fibonacci series in JavaScript**

Fibonacci numbers are a sequence of numbers where each value is the sum of the previous two, starting with 0 and 1. The first few values are 0, 1, 1, 2, 3, 5, 8, 13 ,…,

function fib(n) {

var a=0, b=1;

for (var i=0; i < n; i++) {

var temp = a+b;

a = b;

b = temp;

}

return a;

}

**Q29.What is the difference between the substr() and substring() functions in JavaScript?**

**Difference between the substr() and substring() functions in JavaScript.**

The substr() function has the form substr(startIndex,length). It returns the substring from startIndex and returns ‘length’ number of characters.

var s = "hello";

( s.substr(1,4) == "ello" ) // true

The substring() function has the form substring(startIndex,endIndex). It returns the substring from startIndex up to endIndex – 1.

var s = "hello";

( s.substring(1,4) == "ell" ) // true

**Q30.What are different types of Inheritence? Which Inheritance is followed in Javascript.**

There are two types of Inherientence in OOPS Classic and Prototypical Inheritance. Javascript follows Prototypical Inheritance.

**Q31.What is output of undefined \* 2 in Javascript?**

nan is output of undefined \* 2.

**Q32.How to add/remove properties to object dynamically in Javascript?**

You can add a property to an object using object.property\_name =value , delete object.property\_name is used to delete a property.

**Example:**

let user = new Object();

// adding a property

user.name='Anil';

user.age =25;

console.log(user);

delete user.age;

console.log(user);

**Q33.How to convert Javascript date to ISO standard?**

**toISOString()** method is used to convert javascript date to ISO standard. It converts JavaScript Date object into a string, using the ISO standard.

**Usage:**

var date = new Date();

var n = date.toISOString();

console.log(n);

// YYYY-MM-DDTHH:mm:ss.sssZ

**Q34.How to get inner Html of an element in JavaScript?**

InnerHTML property of HTML DOM is used to get inner Html of an element in JavaScript.

**Example Usage:**

This is inner Element

<script type="text/javascript">

var inner= document.getElementById("inner").innerHTML ;

console.log(inner); // This is inner Element

document.getElementById("inner").innerHTML = "Html changed!";

var inner= document.getElementById("inner").innerHTML ;

console.log(inner); // Html changed!

</script>

**Q35.How to clone an object in Javascript?**

Object.assign() method is used for cloning an object in Javascript.Here is sample usage

var x = {myProp: "value"};

var y = Object.assign({}, x);

**Q36.List different ways of empty an array in Javascript?**

In Javascript, there are many ways to empty an array in Javascript, below we have listed 4 major

* + By assigning an empty array.
  + var arr1 =[1,4,5,6];
  + arr1=[];
  + By assigning array length to 0.
  + var arr2 =[1,4,5,6];
  + arr2.length=0;
  + By poping the elements of the array.
  + var arr2 =[1,4,5,6];
  + while(arr.length > 0) {
  + arr.pop();
  + }
  + By using .splice() .
  + var arr =[1,4,5,6];
  + arr.splice(0,arr.length)

**Q37.** **How to get an element by class in JavaScript ?**

**document.getElementsByClassName()** method is used in Javascript to get an element with a class name.

|  |  |
| --- | --- |
| **getElementsByClassName()** | |
| **Method Name** | getElementsByClassName |
| **Syntax** | document.getElementsByClassName('className') |
| **Parameter** | String (name of class) |
| **Output** | Array of HTMLCollection that have inputted className |

**Q38.** **Explain Typecasting in Javascript?**

In Programming whenever we need to convert a variable from one data type to another Typecasting is used. In Javascript, we can do this via library functions. There are basically 3 typecasts are available in Javascript Programming, they are:

* + Boolean(value): Casts the inputted value to a Boolean
  + Number(value): Casts the inputted value to an Integer or Floating point Number.
  + String(value) : Casts the inputted value value a string

**Q39.**

**How to encode and decode a URL in JavaScript?**

**encodeURI()** function is used to encode an URL in Javascript.It takes a url string as parameter and return encoded string. Note: encodeURI() did not encode characters like **/ ? : @ & = + $ #**, if you have to encode these characters too please use encodeURIComponent(). Usage:

var uri = "my profile.php?name=sammer&occupation=pāntiNG";

var encoded\_uri = encodeURI(uri);

**decodeURI()** function is used to decode an URL in Javascript.It takes a encoded url string as parameter and return decoded string. Usage:

var uri = "my profile.php?name=sammer&occupation=pāntiNG";

var encoded\_uri = encodeURI(uri);

decodeURI(encoded\_uri);

**Q40.How do you change the title of the page by JavaScript?**

You can change the title of a webpage using setting the title property of the document object.

**Example usage**

document.title="My New Title";

**Q41.**

**What is difference between deep and shallow object coping in JavaScript?**

**Q42.**

**List some Unit Testing Frameworks JavaScript**

Below is the list of few most Popular Javascript Unit Testing Frameworks:

* + Unit.js
  + Jasmine
  + Karma
  + Chai
  + AVA
  + Mocha
  + JSUnit
  + QUnit
  + Jest

**Q43How to add a new property in existing function JavaScript?**

**Q44.Explain JavaScript Accessors ?**

JavaScript Accessors

**Q45.List few difference between primitive and non primitive JavaScript data types?**

**Q46.Explain higher-order functions in JavaScript?**

**Q47.Explain few difference between null, undefined or undeclared JavaScript variable?**

**Q48.How host objects are different from native objects in JavaScript?**

**Q49.What is difference between var x =1; and x=1;?**

**Q50.Explain spread operator in JavaScript?**

**Q51.How to call a function in every x seconds in JavaScript?**

**Q52.Explain Promise in JavaScript?**

**Q53.What is difference between Array.splice() and Array.slice() method in JavaScript?**

**Q54.Is JavaScript multi-threaded or single-threaded?**

JavaScript is single-threaded.

**Q55.Explain JavaScript Debounce Function?**

**Q56.List some Design patterns in JavaScript?**

**Q57.What is console.time() and console.timeEnd()? What is its syntax, and why is it used?**

**Q58.What are different types of Scope Chain available in JavaScript?**

**Q59.How to remove duplicate values from a JavaScript array?**

We can use array.indexOf method to check a value exists or not. See below example to remove duplicate values.

let duplicates = ['delhi','kanpur','kanpur','goa','delhi','new york'];

function removeDuplicatesValues(arr){

let unique\_array = [];

for(let i = 0;i < arr.length; i++){

if(unique\_array.indexOf(arr[i]) == -1){

unique\_array.push(arr[i])

}

}

return unique\_array

}

console.log(removeDuplicatesValues(duplicates));

* **Q60.How to redirect a page to another page in Javascript?**

* **Q61.Is it possible to do 301 redirects in Javascript ?**

JavaScript entirely runs on the client machine. 301 is response code that is sent by the server as a response. So it is not possible to do 301 Redirects In JavaScript.

* **Q62.Write a program to reverse a string in pure JavaScript?**

**Q63.Write program to remove duplicate in an array ?**

**Q64.List few Difference between JAVA and JavaScript?**

**Q65.Explain MUL function in Javascript?**

**Q66.List few advantages of using JavaScript?**

Few advantage of Javascript

* + Javascript is executed on user's computer, the meaning is that whatever you do in Javascript will not add any processing strain on the server. and that's why it is called as the client-side programming language. And this feature makes your sites responsive for the end user and less expensive for you in terms of server traffic.
  + With the help of Javascript, you can create highly responsive interfaces which will improve the user experience and provide dynamic functionality, without waiting for the server to show another page.
  + If you want to make online systems available offline and sync automatically once the computer goes online, then Javascript is the best technology you can use. you can do this using the right browser add-ons (Such as Google or Yahoo Browser Plus).
  + Content loading and changing it dynamically. Using Ajax in Javascript you can load content into the document if and when the user needs it, without reloading the entire page.
  + Using the Principles of unobtrusive JavaScript(defensive Scripting), JavaScript can test for what is possible in your browser and react accordingly.
  + Javascript is an object-oriented programming language that supports the creation of both client and server side applications. It is dynamic, weakly typed, prototype-based and multi-paradigm high-level programming language

# JavaScript Interview Questions:

### 1) What is JavaScript?

**JavaScript** is a scripting language. It is different from Java language. It is object-based, lightweight, cross-platform translated language. It is widely used for client-side validation. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

### 2) List some features of JavaScript.

Some of the features of JavaScript are:

* Lightweight
* Interpreted programming language
* Good for the applications which are network-centric
* Complementary to Java
* Complementary to HTML
* Open source
* Cross-platform

### 3) List some of the advantages of JavaScript.

Some of the advantages of JavaScript are:

* Server interaction is less
* Feedback to the visitors is immediate
* Interactivity is high
* Interfaces are richer

### 4) List some of the disadvantages of JavaScript.

Some of the disadvantages of JavaScript are:

* No support for multithreading
* No support for multiprocessing
* Reading and writing of files is not allowed
* No support for networking applications.

### 5) Define a named function in JavaScript.

The function which has named at the time of definition is called a named function. For example

function msg()

{

   document.writeln("Named Function");

}

msg();

### 6) Name the types of functions

The types of function are:

* Named - These type of functions contains name at the time of definition. For Example:

function display()

{

document.writeln("Named Function");

}

display();

* Anonymous - These type of functions doesn't contain any name. They are declared dynamically at runtime.

var display=function()

{

document.writeln("Anonymous Function");

}

display();

### 7) Define anonymous function

It is a function that has no name. These functions are declared dynamically at runtime using the function operator instead of the function declaration. The function operator is more flexible than a function declaration. It can be easily used in the place of an expression. For example:

var display=function()

{

 alert("Anonymous Function is invoked");

}

display();

### 8) Can an anonymous function be assigned to a variable?

Yes, you can assign an anonymous function to a variable.

### 9) In JavaScript what is an argument object?

The variables of JavaScript represent the arguments that are passed to a function.

### 10) Define closure.

In JavaScript, we need closures when a variable which is defined outside the scope in reference is accessed from some inner scope.

var num = 10;

function sum()

{

document.writeln(num+num);

}

sum();

### 11) If we want to return the character from a specific index which method is used?

The JavaScript string charAt() method is used to find out a char value present at the specified index. The index number starts from 0 and goes to n-1, where n is the length of the string. The index value can't be a negative, greater than or equal to the length of the string. For example:

var str="Javascriptbynagaraj";

document.writeln(str.charAt(4));    Ans:s

### 12) What is the difference between JavaScript and JScript?

Netscape provided the JavaScript language. Microsoft changed the name and called it JScript to avoid the trademark issue. In other words, you can say JScript is the same as JavaScript, but Microsoft provides it.

### 13) How to write a hello world example of JavaScript?

A simple example of JavaScript hello world is given below. You need to place it inside the body tag of HTML.

**<script** type="text/javascript"**>**

document.write("JavaScript Hello World!");

**</script>**

### 14) How to use external JavaScript file?

I am assuming that js file name is message.js, place the following script tag inside the head tag.

**<script** type="text/javascript" src="message.js"**></script>**

### 15) Is JavaScript case sensitive language?

Yes, JavaScript is a case sensitive language. For example:

Var msg = "JavaScript is a casesensitive language";

//Here, var should be used to declare a variable

function display()

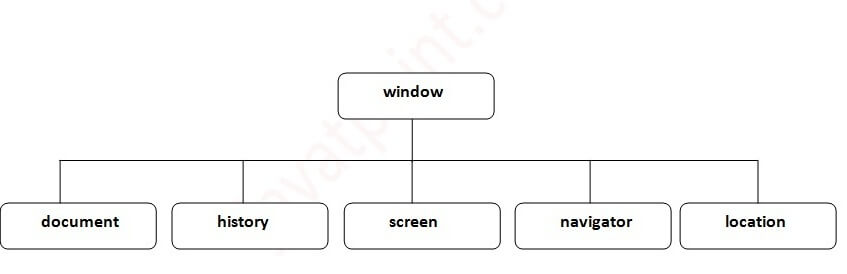
{

document.writeln(msg); // It will not display the result.

}   display();

### 16) What is BOM?

**BOM** stands for Browser Object Model. It provides interaction with the browser. The default object of a browser is a window. So, you can call all the functions of the window by specifying the window or directly. The window object provides various properties like document, history, screen, navigator, location, innerHeight, innerWidth,



### 17) What is DOM? What is the use of document object?

**DOM** stands for Document Object Model. A document object represents the HTML document. It can be used to access and change the content of HTML.

### 18) What is the use of window object?

The window object is created automatically by the browser that represents a window of a browser. It is not an object of JavaScript. It is a browser object.

The window object is used to display the popup dialog box. Let's see with description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| alert() | displays the alert box containing the message with ok button. |
| confirm() | displays the confirm dialog box containing the message with ok and cancel button. |
| prompt() | displays a dialog box to get input from the user. |
| open() | opens the new window. |
| close() | closes the current window. |
| setTimeout() | performs the action after specified time like calling function, evaluating expressions. |

### 19) What is the use of history object?

The history object of a browser can be used to switch to history pages such as back and forward from the current page or another page. There are three methods of history object.

1. history.back() - It loads the previous page.
2. history.forward() - It loads the next page.
3. history.go(number) - The number may be positive for forward, negative for backward. It loads the given page number.

### 20) How to write a comment in JavaScript?

There are two types of comments in JavaScript.

1. Single Line Comment: It is represented by // (double forward slash)
2. Multi-Line Comment: Slash represents it with asterisk symbol as /\* write comment here \*/

### 21) How to create a function in JavaScript?

To create a function in JavaScript, follow the following syntax.

function function\_name(){

//function body

}

### 22) What are the JavaScript data types?

There are two types of data types in JavaScript:

1. Primitive Data Types - The primitive data types are as follows:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| String | represents a sequence of characters, e.g., "hello" |
| Number | represents numeric values, e.g., 100 |
| Boolean | represents boolean value either false or true |
| Undefined | represents an undefined value |
| Null | represents null, i.e., no value at all |

1. Non-primitive Data Types - The non-primitive data types are as follows:

|  |  |
| --- | --- |
| **DataType** | **Description** |
| Object | represents an instance through which we can access members |
| Array | represents a group of similar values |
| RegExp | represents regular expression |

### 23) What is the difference between == and ===?

The == operator checks equality only whereas === checks equality, and data type, i.e., a value must be of the same type.

### 24) How to write HTML code dynamically using JavaScript?

The innerHTML property is used to write the HTML code using JavaScript dynamically. Let's see a simple example:

document.getElementById('mylocation').innerHTML="<h2>This is heading using JavaScript</h2>";

### 25) How to write normal text code using JavaScript dynamically?

The innerText property is used to write the simple text using JavaScript dynamically. Let's see a simple example:

document.getElementById('mylocation').innerText="This is text using JavaScript";

### 26) How to create objects in JavaScript?

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

There are 3 ways to create an object in JavaScript.

1. By object literal
2. By creating an instance of Object
3. By Object Constructor

**1).The syntax of creating object using object literal is given below:**

**o**bject={property1:value1,property2:value2.....propertyN:valueN} . As you can see, property and value is separated by : (colon).

Let’s see the simple example of creating object in JavaScript.

**<script>**

emp={id:102,name:"Shyam Kumar",salary:40000}

document.write(emp.id+" "+emp.name+" "+emp.salary);

**</script>**

## 2) By creating instance of Object

The syntax of creating object directly is given below:

var objectname=new Object();  Here, **new keyword** is used to create object.

Let’s see the example of creating object directly.

**<script>**

var emp=new Object();

emp.id=101;

emp.name="Ravi Malik";

emp.salary=50000;

document.write(emp.id+" "+emp.name+" "+emp.salary);

**</script>**

## 3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

**<script>**

function emp(id,name,salary)

{

this.id=id;

this.name=name;

this.salary=salary;

}

e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

**</script>**

## Defining method in JavaScript Object

We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.

The example of defining method in object is given below.

**<script>**

function emp(id,name,salary){

this.id=id;

this.name=name;

this.salary=salary;

this.changeSalary=changeSalary;

function changeSalary(otherSalary){

this.salary=otherSalary;

}

}

e=new emp(103,"Sonoo Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);

e.changeSalary(45000);

document.write("**<br>**"+e.id+" "+e.name+" "+e.salary);

**</script>**

## JavaScript Object Methods

The various methods of Object are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No Methods** | | **Methods** | **Description** |
| 1 | [Object.assign()](https://www.javatpoint.com/javascript-object-assign-method) | This method is used to copy enumerable and own properties from a source object to a target object |
| 2 | [Object.create()](https://www.javatpoint.com/javascript-object-create-method) | This method is used to create a new object with the specified prototype object and properties. |
| 3 | [Object.defineProperty()](https://www.javatpoint.com/javascript-object-defineproperty-method) | This method is used to describe some behavioral attributes of the property. |
| 4 | [Object.defineProperties()](https://www.javatpoint.com/javascript-object-defineproperties-method) | This method is used to create or configure multiple object properties. |
| 5 | [Object.entries()](https://www.javatpoint.com/javascript-object-entries-method) | This method returns an array with arrays of the key, value pairs. |
| 6 | [Object.freeze()](https://www.javatpoint.com/javascript-object-freeze-method) | This method prevents existing properties from being removed. |
| 7 | [Object.getOwnPropertyDescriptor()](https://www.javatpoint.com/javascript-object-getownpropertydescriptor-method) | This method returns a property descriptor for the specified property of the specified object. |
| 8 | [Object.getOwnPropertyDescriptors()](https://www.javatpoint.com/javascript-object-getownpropertydescriptors-method) | This method returns all own property descriptors of a given object. |
| 9 | [Object.getOwnPropertyNames()](https://www.javatpoint.com/javascript-object-getownpropertynames-method) | This method returns an array of all properties (enumerable or not) found. |
| 10 | [Object.getOwnPropertySymbols()](https://www.javatpoint.com/javascript-object-getownpropertysymbols-method) | This method returns an array of all own symbol key properties. |
| 11 | [Object.getPrototypeOf()](https://www.javatpoint.com/javascript-object-getprototypeof-method) | This method returns the prototype of the specified object. |
| 12 | [Object.is()](https://www.javatpoint.com/javascript-object-is-method) | This method determines whether two values are the same value. |
| 13 | [Object.isExtensible()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is extensible |
| 14 | [Object.isFrozen()](https://www.javatpoint.com/javascript-objects) | This method determines if an object was frozen. |
| 15 | [Object.isSealed()](https://www.javatpoint.com/javascript-objects) | This method determines if an object is sealed. |
| 16 | [Object.keys()](https://www.javatpoint.com/javascript-objects) | This method returns an array of a given object's own property names. |
| 17 | [Object.preventExtensions()](https://www.javatpoint.com/javascript-object-preventextensions-method) | This method is used to prevent any extensions of an object. |
| 18 | [Object.seal()](https://www.javatpoint.com/javascript-object-seal-method) | This method prevents new properties from being added and marks all existing properties as non-configurable. |
| 19 | [Object.setPrototypeOf()](https://www.javatpoint.com/javascript-object-setprototypeof-method) | This method sets the prototype of a specified object to another object. |
| 20 | [Object.values()](https://www.javatpoint.com/javascript-object-values-method) | This method returns an array of values. |

### 27) How to create an array in JavaScript?

There are 3 ways to create an array in JavaScript.

1. By array literal
2. By creating an instance of Array
3. By using an Array constructor

## 1) JavaScript array literal

The syntax of creating array using array literal is given below:

var arrayname=[value1,value2.....valueN];

As you can see, values are contained inside [ ] and separated by , (comma).

Let’s see the simple example of creating and using array in JavaScript.

**<script>**

var emp=["Sonoo","Vimal","Ratan"];

for (i=0;i**<emp.length**;i++){

document.write(emp[i] + "**<br/>**");

}  **</script>**

**2) JavaScript Array directly (new keyword)**

The syntax of creating array directly is given below:

var arrayname=new Array();  Here, new keyword is used to create instance of array.

Let’s see the example of creating array directly.

**<script>**

var i;

var emp = new Array();

emp[0] = "Arun";

emp[1] = "Varun";

emp[2] = "John";

for (i=0;i**<emp.length**;i++){

document.write(emp[i] + "**<br>**");

}

**</script>**

## 3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

**<script>**

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i**<emp.length**;i++){

document.write(emp[i] + "**<br>**");

}

**</script>**

### 28) What does the isNaN() function?

The isNan() function returns true if the variable value is not a number. For example:

function number(num) {

 if (isNaN(num)) {

return "Not a Number";

}

  return "Number";

}

Eg:console.log(number('1000F'));  // expected output: "Not a Number"

Eg:console.log(number('1000'));  // expected output: "Number"

### 29) What is the output of 10+20+"30" in JavaScript?

3030 because 10+20 will be 30. If there is numeric value before and after +, it treats as binary + (arithmetic operator).

function display()

{

  document.writeln(10+20+"30");

}

display();

### 30) What is the output of "10"+20+30 in JavaScript?

102030 because after a string all the + will be treated as string concatenation operator (not binary +).

function display()

{

document.writeln("10"+20+30);

}

display();

### 31) Difference between Client side JavaScript and Server side JavaScript?

**Client-side JavaScript** comprises the basic language and predefined objects which are relevant to running JavaScript in a browser. The client-side JavaScript is embedded directly by in the HTML pages. The browser interprets this script at runtime.

**Server-side JavaScript** also resembles client-side JavaScript. It has a relevant JavaScript which is to run in a server. The server-side JavaScript are deployed only after compilation.

### 32) In which location cookies are stored on the hard disk?

The storage of cookies on the hard disk depends on the OS and the browser.

The Netscape Navigator on Windows uses a cookies.txt file that contains all the cookies. The path is c:\Program Files\Netscape\Users\username\cookies.txt

The Internet Explorer stores the cookies on a file username@website.txt. The path is: c:\Windows\Cookies\username@Website.txt.

### 33) What is the real name of JavaScript?

The original name was **Mocha**, a name chosen by Marc Andreessen, founder of Netscape. In September of 1995, the name was changed to LiveScript. In December 1995, after receiving a trademark license from Sun, the name JavaScript was adopted.

### 34) What is the difference between undefined value and null value?

**Undefined value:** A value that is not defined and has no keyword is known as undefined value. For example:

int number;//Here, a number has an undefined value.

**Null value:** A value that is explicitly specified by the keyword "null" is known as a null value. For example:

String str=null;//Here, str has a null value.

### 35) How to set the cursor to wait in JavaScript?

The cursor can be set to wait in JavaScript by using the property "cursor". The following example illustrates the usage:

**<script>**

window.document.body.style.cursor = "wait";

**</script>**

### 36) What is this [[[]]]?

This is a three-dimensional array. Eg:var myArray = [[[]]];

### 37) Are Java and JavaScript same?

No, Java and JavaScript are the two different languages. Java is a robust, secured and object-oriented programming language whereas JavaScript is a client-side scripting language with some limitations.

### 38) What is negative infinity?

Negative Infinity is a number in JavaScript which can be derived by dividing the negative number by zero. For example:

var num=-5;

function display()

{

document.writeln(num/0);

}

display();  //expected output: -Infinity

### 39) What is the difference between View state and Session state?

"View state" is specific to a page in a session whereas "Session state" is specific to a user or browser that can be accessed across all pages in the web application.

### 40) What are the pop-up boxes available in JavaScript?

* Alert Box
* Confirm Box
* Prompt Box

#### Example of alert() in JavaScript

**<script** type="text/javascript"**>**

function msg(){

 alert("Hello Alert Box");

}

**</script>**

**<input** type="button" value="click" onclick="msg()"**/>**

#### Example of confirm() in JavaScript

**<script** type="text/javascript"**>**

function msg(){

var v= confirm("Are u sure?");

if(v==true){

alert("ok");

}

else{

alert("cancel");

}   }

**</script>**

**<input** type="button" value="delete record" onclick="msg()"**/>**

#### Example of prompt() in JavaScript

**<script** type="text/javascript"**>**

function msg(){

var v= prompt("Who are you?");

alert("I am "+v);

}  **</script>**

**<input** type="button" value="click" onclick="msg()"**/>**

### 41) How can we detect OS of the client machine using JavaScript?

The **navigator.appVersion** string can be used to detect the operating system on the client machine.

### 42) How to submit a form using JavaScript by clicking a link?

Let's see the JavaScript code to submit the form by clicking the link.

**<form** name="myform" action="index.php"**>**

Search: **<input** type='text' name='query' **/>**

**<a** href="javascript: submitform()"**>**Search**</a>**

**</form>**

**<script** type="text/javascript"**>**

function submitform()

{

document.myform.submit();

}

**</script>**

### 43) Is JavaScript faster than ASP script?

Yes, because it doesn't require web server's support for execution.

### 44) How to change the background color of HTML document using JavaScript?

**<script** type="text/javascript"**>**

document.body.bgColor="pink";

**</script>**

### 45) How to handle exceptions in JavaScript?

By the help of try/catch block, we can handle exceptions in JavaScript. JavaScript supports try, catch, finally and throw keywords for exception handling.

### 46) How to validate a form in JavaScript?

**<script>**

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;

if (name==null || name==""){

alert("Name can't be blank");

return false;

}else if(password.length**<6**){

 alert("Password must be at least 6 characters long.");

return false;

}  }  **</script>**

**<body>**

**<form** name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" **>**

Name: **<input** type="text" name="name"**><br/>**

Password: **<input** type="password" name="password"**><br/>**

**<input** type="submit" value="register"**>**

**</form>**

### 47) How to validate email in JavaScript?

**<script>**

function validateemail()

{

var x=document.myform.email.value;

var atposition=x.indexOf("@");

var dotposition=x.lastIndexOf(".");

if (atposition**<1** || dotposition**<atposition**+2 || dotposition+2**>**=x.length)

{

alert("Please enter a valid email address \n atpostion:"+atposition+"\n dotposition:"+dotposition);

   return false;

}

}

**</script>**

**<body>**

**<form** name="myform"  method="post" action="#" onsubmit="return validateemail();"**>**

Email: **<input** type="text" name="email"**><br/>**

**<input** type="submit" value="register"**>**

**</form>**

### 48) What is this keyword in JavaScript?

The this keyword is a reference variable that refers to the current object. For example:

var address=

{

company:"Javaworld",

city:"Noida",

state:"UP",

fullAddress:function()

{

return this.company+" "+this.city+" "+this.state;

}

};

var fetch=address.fullAddress();

document.writeln(fetch);

### 49) What is the requirement of debugging in JavaScript?

JavaScript didn't show any error message in a browser. However, these mistakes can affect the output. The best practice to find out the error is to debug the code. The code can be debugged easily by using web browsers like Google Chrome, Mozilla Firebox.

To perform debugging, we can use any of the following approaches:

* Using console.log() method
* Using debugger keyword

### 50) What is the use of debugger keyword in JavaScript?

JavaScript debugger keyword sets the breakpoint through the code itself. The debugger stops the execution of the program at the position it is applied. Now, we can start the flow of execution manually. If an exception occurs, the execution will stop again on that particular line.. For example:

function display()

{

x = 10;

y = 15;

z = x + y;

debugger;

document.write(z);

document.write(a);

}

display();

### 51) What is the role of a strict mode in JavaScript?

The JavaScript strict mode is used to generates silent errors. It provides "use strict"; expression to enable the strict mode. This expression can only be placed as the first statement in a script or a function. For example:

"use strict";

x=10;

console.log(x);

### 52) What is the use of Math object in JavaScript?

The JavaScript math object provides several constants and methods to perform a mathematical operation. Unlike date object, it doesn't have constructors. For example:

function display()

{

document.writeln(Math.random());

}

display();

### 53) What is the use of a Date object in JavaScript?

The JavaScript date object can be used to get a year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

function display()

{

var date=new Date();

var day=date.getDate();

var month=date.getMonth()+1;

var year=date.getFullYear();

document.write("**<br>**Date is: "+day+"/"+month+"/"+year);

}

display();

### 54) What is the use of a Number object in JavaScript?

The JavaScript number object enables you to represent a numeric value. It may be integer or floating-point. JavaScript number object follows the IEEE standard to represent the floating-point numbers.

function display()

{

var x=102;//integer value

var y=102.7;//floating point value

var z=13e4;//exponent value, output: 130000

var n=new Number(16);//integer value by number object

document.write(x+" "+y+" "+z+" "+n);

}

display();

### 55) What is the use of a Boolean object in JavaScript?

The JavaScript Boolean is an object that represents value in two states: true or false. You can create the JavaScript Boolean object by Boolean() constructor.

function display()

{

document.writeln(10**<20**);//true

document.writeln(10**<5**);//false

}

display();

### 56) What is the use of a TypedArray object in JavaScript?

The JavaScript TypedArray object illustrates an array like a view of an underlying binary data buffer. There is any number of different global properties, whose values are TypedArray constructors for specific element types.

function display()

{

var arr1= [1,2,3,4,5,6,7,8,9,10];

arr1.copyWithin(2) ;

document.write(arr1);

}

display();

### 57) What is the use of a Set object in JavaScript?

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references. For example:

function display()

{

var set = new Set();

set.add("jQuery");

set.add("AngularJS");

set.add("Bootstrap");

for (let elements of set) {

document.writeln(elements+"**<br>**");

}

}

display();

### 58) What is the use of a WeakSet object in JavaScript?

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values. For example:

function display()

{

var ws = new WeakSet();

var obj1={};

var obj2={};

ws.add(obj1);

ws.add(obj2);

//Let's check whether the WeakSet object contains the added object

document.writeln(ws.has(obj1)+"**<br>**");

document.writeln(ws.has(obj2));

}

display()

### 59) What is the use of a Map object in JavaScript?

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key. For example:

function display()

{

var map=new Map();

map.set(1,"jQuery");

map.set(2,"AngularJS");

map.set(3,"Bootstrap");

document.writeln(map.get(1)+"**<br>**");

document.writeln(map.get(2)+"**<br>**");

document.writeln(map.get(3));

}

display();

### 60) What is the use of a WeakMap object in JavaScript?

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values. For example:

function display()

{

var wm = new WeakMap();

var obj1 = {};

var obj2 = {};

var obj3= {};

wm.set(obj1, "jQuery");

wm.set(obj2, "AngularJS");

wm.set(obj3,"Bootstrap");

document.writeln(wm.has(obj2));

}

display();

**jQuery Interview Question:**

1) What is jQuery?

jQuery is a fast, lightweight, feature-rich client-side JavaScript library. It is cross-platform and supports different types of browsers. It has provided a much-needed boost to JavaScript. Before jQuery, JavaScript codes were lengthy and bigger, even for smaller functions. It makes a website more interactive and attractive.

2) Is jQuery a programming language?

jQuery is not a programming language but a well-written JavaScript code. It is used to traverse documents, event handling, Ajax interaction, and Animation.

3) What is the difference between JavaScript and jQuery?

The simple difference is that JavaScript is a language while jQuery is a built-in library built for JavaScript. jQuery simplifies the use of JavaScript language.

4) Is jQuery replacement of JavaScript?

No, jQuery is not the replacement of JavaScript. jQuery is written on the top of JavaScript, and it is a different library. jQuery is a lightweight JavaScript library which is used to interact with JavaScript and HTML.

5) Why do we use jQuery?

* It is very easy to learn and use.
* It is used to develop browser compatible web applications.
* It improves the performance of an application.
* It is very fast and extensible.
* It facilitates you to write minimal lines of codes for UI related functions.
* It provides cross-browser support.

6) What is $() in jQuery library?

The $() function is an alias of jQuery() function. It is used to wrap any object into jQuery object which later facilitates you to call the various method defined jQuery object. You can pass a selector string to $() function, and it returns a jQuery object which contains an array of all matched DOM elements.

Syntax:

$(document).ready(function() {

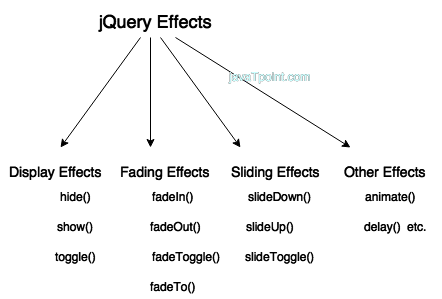
$("p").css("background-color", "pink");

});

7) What are the effects methods used in jQuery?

These are some effects methods used in jQuery:

* show() - It displays or shows the selected elements.
* hide() - It hides the matched or selected elements.
* toggle() - It shows or hides the matched elements. In other words, it toggles between the hide() and shows() methods.
* fadeIn() - It shows the matched elements by fading it to opaque. In other words, it fades in the selected elements.
* fadeOut() - It shows the matched elements by fading it to transparent. In other words, it fades out the selected elements.



8) What is the use of toggle() method in JQuery?

The jQuery toggle() is a particular type of method which is used to toggle between the hide() and show() method. It shows the hidden elements and hides the shown element.

Syntax:

$(selector).toggle();

$(selector).toggle(speed, callback);

$(selector).toggle(speed, easing, callback);

$(selector).toggle(display);

**speed**: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

**easing**: It specifies the easing function to be used for transition.

**callback**: It is also an optional parameter. It specifies the function to be called after completion of toggle() effect.

**display**: If true, it displays an element. If false, it hides the element

9) What is the purpose of fadeToggle() method in JQuery?

The jQuery fadeToggle() method is used to toggle between the fadeIn() and fadeOut() methods. If the elements are faded in, it makes them faded out, and if they are faded out, it makes them faded in.

Syntax:

$(selector).fadeToggle();

$(selector).fadeToggle(speed,callback);

$(selector).fadeToggle(speed, easing, callback);

**speed**: It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

**easing**: It specifies the easing function to be used for transition.

**callback**: It is also an optional parameter. It specifies the function to be called after completion of fadeToggle() effect.

10) What is the use of delay() method in JQuery?

The jQuery delay() method is used to delay the execution of functions in the queue. It is the best method to make a delay between the queued jQuery effects. The jQUery delay () method sets a timer to delay the execution of the next item in the queue.

Syntax:

$(selector).delay (speed, queueName)

**speed:** It is an optional parameter. It specifies the speed of the delay. Its possible values are slow, fast and milliseconds.

**queueName:** It is also an optional parameter. It specifies the name of the queue. Its default value is "fx" the standard queue effect.

11) Is it possible that jQuery HTML work for both HTML and XML document?

No, jQuery HTML only works for HTML document. It doesn't work for XML documents.

12) What is the use of html() method in JQuery?

The jQuery html() method is used to change the entire content of the selected elements. It replaces the selected element content with new contents.

Syntax:

$(document).ready(function(){

 $("button").click(function(){

 $("p").html("Hello **<b>**JQuerylibrary.com**</b>**");

});

});

13) What is the use of css() method in JQuery?

The jQuery CSS() method is used to get (return)or set style properties or values for selected elements. It facilitates you to get one or more style properties. The jQuery CSS() provides two ways:

Return a CSS property

It is used to get the value of a specified CSS property.

$(document).ready(function(){

$("button").click(function(){

alert("Background color = " + $("p").css("background-color"));

});

});

Set a CSS property

This property is used to set a specific value for all matched element.

$(document).ready(function(){

    $("button").click(function(){

        $("p").css("background-color", "violet");

    });

});

14) Is jQuery library used for server scripting or client scripting?

It is a library for client-side Scripting.

15) Is jQuery a W3C standard?

No, jQuery is not a W3C standard.

16) What is the starting point of code execution in jQuery?

$(document).ready() function is the starting point of jQuery code. It is executed when DOM is loaded.

17) What is the basic requirement to start with the jQuery?

You need refer to its library to start with jQuery. You can download the latest version of jQuery from jQuery.com.

18) Can you use any other name in place of $ (dollar sign) in jQuery?

Yes, instead of $ (dollar sign) we can use jQuery as a function name. For example:

jQuery(document).ready(function() {

jQuery("p").css("background-color", "pink");

});

19) Can you use multiple document.ready() function on the same page?

Yes. You can use any number of document.ready() function on the same page. For example:

$(document).ready(function() {

$("h1").css("background-color", "red");

});

$(document).ready(function() {

$("p").css("background-color", "pink");

 });

20) What is the difference between find and children methods?

Find method is used to find all levels down the DOM tree while children method is used to find single level down the DOM tree.

21) What is a CDN?

CDN stands for Content Delivery Network or Content Distribution Network. It is a large distributed system of servers deployed in multiple data centers across the internet. It provides the files from servers at a higher bandwidth that leads to faster loading time. These are several companies that provide free public CDNs:

* Google
* Microsoft
* Yahoo

22) What is the goal of CDN and what are the advantages of using CDN?

The primary goal of the CDN is to provide content to the end-users with high availability and high performance.

**Advantages of using CDN:**

* It reduces the load from the server.
* It saves bandwidth. jQuery framework is loaded faster from these CDN.
* If a user regularly visits a site which is using jQuery framework from any of these CDN, it will be cached.

23) How can you use a jQuery library in your project?

You can use a jQuery library in the ASP.Net project from downloading the latest jQuery library from jQuery.com and include the references to the jQuery library file in your HTML/PHP/JSP/Aspx page.

**<script** src="\_scripts/jQuery1.2.6.js" type="text/javascript"**></script>**

**<script** language="javascript"**>**

$(document).ready(function() {

alert('test');

});

**</script>**

24) What are the selectors in jQuery? How many types of selectors in jQuery?

If you want to work with an element on the web page, first you need to find it. Selectors find the HTML elements in jQuery. There are many types of selectors. Some basic selectors are:

* **Name:** It is used to select all elements which match with the given element Name.
* **#ID:** It is used to select a single element which matches with the given ID
* **.Class:** It is used to select all elements which match with the given Class.
* **Universal (\*):** It is used to select all elements available in a DOM.
* **Multiple Elements E, F, G:** It is used to selects the combined results of all the specified selectors E, F or G.
* **Attribute Selector:** It is used to select elements based on its attribute value.

25) What is a use of jQuery filter?

jQuery filter is used to filter the specific values from the object. It filters the result of your original query into specific elements.

26) What are the different types of selectors in jQuery?

There are three types of selectors in jQuery:

* CSS Selector
* Custom Selector
* XPath Selector

27) What is the difference between the ID selector and class selector in jQuery?

ID selector and class selector are the same as they are in CSS. ID selector uses ID while class selector uses a class to select elements.

You use an ID selector to select just one element. If you want to select a group of elements, having the same CSS class, use class selector.

28) What is the use of serialize() method in JQuery?

The jQuery serialize() method is used to create a text string in standard URL-encoded notation. It serializes the form values so that its serialized values can be used in the URL query string while making an AJAX request.

Syntax:

$(document).ready(function(){

$("button").click(function(){

 $("div").text($("form").serialize());

});

});

29) What is the use of val() method in JQuery?

The jQuery val() method is used:

* To get the current value of the first element in the set of matched elements.
* To set the value of every matched element.

Syntax:

$(document).ready(function(){

 $("button").click(function(){

 $("div").text($("form").serialize());

});

});

30) How to add and remove CSS classes to an element using jQuery?

You can use addclass() jQuery method to add CSS class to an element and removeclass() jQuery method to remove CSS class from an element.

CSS addClass() Example

<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">

 </script>

<script type="text/javascript" language="javascript">

$(document).ready(function()  {

$("#btn").click(function()  {

 $("#para").addClass("change");

});

});  </script>

<style>

.change

    {

     color:blue;

 }  </style>

<p id="para">This method adds CSS **class** from an element</p>

<input type="button" id="btn" value="Click me">

CSS removeClass() Example

 <script type="text/javascript" language="javascript">

$(document).ready(function()  {

$("#btn").click(function()  {

     $("p").removeClass("change");

});

});

 </script>

<style>

     .change

     {

         color:blue;

     }

</style>

<p **class**="change">This method removes CSS **class** to an element</p>

<input type="button" id="btn" value="Click me">

31) Can you write a jQuery code to select all links inside the paragraph?

Yes. You can use <a> tag nested inside paragraph <p> tag to select all links. For example:

 <script type="text/javascript" language="javascript">

$(document).ready(function()  {

     $("p a").attr("href", "https://www.javatpoint.com");

});

  </script>

     <p><a>Learn JavaScript</a></p>

     <p><a>Learn jQuery</a></p>

32) What is the difference between prop and attr?

**attr():** It gets the value of an attribute for the first element in the set of matched element.

**prop():** it gets the value of a property for the first element in the set of matched elements. It is introduced in jQuery 1.6.

33) What are the two types of CDNs?

There are two types of CDN:

* **Microsoft:** It loads jQuery from AJAX CDN.
* **Google:** It loads jQuery from Google libraries API.

34) What is the use of the animate() method in jQuery?

The animate function is used to apply the custom animation effect to elements. Syntax:

$(selector).animate({params}, [duration], [easing], [callback])

Here,

* "param" defines the CSS properties on which you want to apply the animation.
* "duration" specify how long the animation run. It can be one of the following values: "slow," "fast," "normal" or milliseconds
* "easing" is the string which specifies the function for the transition.
* "callback" is the function which we want to run once the animation effect is complete.

35) How can you disable jQuery animation?

By using jQuery property "jQuery.fx.off" and setting it to true, you can disable jQuery animation.

# SET- III JavaScript-Interview-Questions

## Question 1. What's the difference between undefined and not defined in JavaScript

In JavaScript if you try to use a variable that doesn't exist and has not been declared, then JavaScript will throw an error var name is not defined and the script will stop execute thereafter. But If you use typeof undeclared\_variable then it will return undefined.

Before starting further discussion let's understand the difference between declaration and definition.

var x is a declaration because you are not defining what value it holds yet, but you are declaring its existence and the need for memory allocation.

var x; // declaring x

console.log(x); // output: undefined

var x = 1 is both declaration and definition (also we can say we are doing initialisation), Here declaration and assignment of value happen inline for variable x, In JavaScript every variable declaration and function declaration brings to the top of its current scope in which it's declared then assignment happen in order this term is called **hoisting**.

A variable can be declared but not defined. When we try to access it, It will result undefined.

var x; // Declaration

typeof x ==='undefined'; // Will return true

A variable can be neither declared nor defined. When we try to reference such variable then the result will be not defined.

console.log(y); // Output: ReferenceError: y is not defined

### Ref Link:

<http://stackoverflow.com/questions/20822022/javascript-variable-definition-declaration>

## Question 2. For which value of x the results of the following statements are not the same?

// if( x <= 100 ) {...}

if( !(x >100) ) {...}

### Answer

NaN <= 100 is false and NaN > 100 is also false, so if the value of x is NaN, the statements are not the same.

The same holds true for any value of x that being converted to Number, returns NaN, e.g.: undefined, [1,2,5], {a:22} , etc.

This is why you need to pay attention when you deal with numeric variables. NaN can’t be equal, less than or more than any other numeric value, so the only reliable way to check if the value is NaN, is to use isNaN() function.

## Question 3. What is the drawback of declaring methods directly in JavaScript objects?

### Answer

One of the drawback of declaring methods directly in JavaScript objects is that they are very memory inefficient. When you do that, a new copy of the method is created for each instance of an object. Let's see it on example:

varEmployee=function (name, company, salary) {

this.name= name ||"nagaraj";

this.company= company ||"IMR";

this.salary= salary ||5000;

// We can create a method like this:

this.formatSalary=function () {

return"$ "+this.salary;

};

};

// we can also create method in Employee's prototype:

Employee.prototype.formatSalary2=function() {

return"$ "+this.salary;

}

//creating objects

var emp1 =newEmployee('Yuri Garagin', 'Company 1', 1000000);

var emp2 =newEmployee('Dinesh Gupta', 'Company 2', 1039999);

var emp3 =newEmployee('Erich Fromm', 'Company 3', 1299483);

Here each instance variable emp1, emp2, emp3 has own copy of formatSalary method. However the formatSalary2 will only be added once to an object Employee.prototype.

## Question 4. What is “closure” in javascript? Can you provide an example?

A closure is a function defined inside another function (called parent function) and has access to the variable which is declared and defined in parent function scope.

The closure has access to the variable in three scopes:

* Variable declared in his own scope
* Variable declared in parent function scope
* Variable declared in the global namespace

var globalVar ="abc";

// Parent self invoking function

(functionouterFunction (outerArg) {

// begin of scope outerFunction

// Variable declared in outerFunction function scope

var outerFuncVar ='x';

// Closure self-invoking function

(functioninnerFunction (innerArg) {

// begin of scope innerFunction

// variable declared in innerFunction function scope

var innerFuncVar ="y";

console.log("outerArg = "+ outerArg +"\n"+"outerFuncVar = "+ outerFuncVar +"\n"+"innerArg = "+ innerArg +"\n"+"innerFuncVar = "+ innerFuncVar +"\n"+"globalVar = "+ globalVar);

// end of scope innerFunction

})(5); // Pass 5 as parameter

// end of scope outerFunction

})(7); // Pass 7 as parameter

innerFunction is closure which is defined inside outerFunction and has access to all variable which is declared and defined in outerFunction scope. In addition to this function defined inside the function as closure has access to the variable which is declared in global namespace.

Output of above code would be:

outerArg =7

outerFuncVar = x

innerArg =5

innerFuncVar = y

globalVar = abc

## Question 5. Write a mul function which will work properly when invoked with following syntax.

console.log(mul(2)(3)(4)); // output : 24

console.log(mul(4)(3)(4)); // output : 48

### Answer

Below is the code followed by the explanation of how it works:

functionmul (x) {

returnfunction (y) { // anonymous function

returnfunction (z) { // anonymous function

return x \* y \* z;

};

};

}

Here the mul function accepts the first argument and returns the anonymous function which takes the second parameter and returns the anonymous function which takes the third parameter and returns the multiplication of arguments which is being passed in successive

In Javascript function defined inside has access to outer function variable and function is the first class object so it can be returned by the function as well and passed as an argument in another function.

* A function is an instance of the Object type
* A function can have properties and has a link back to its constructor method
* A function can be stored as variable
* A function can be pass as a parameter to another function
* A function can be returned from another function

## Question 6. How to empty an array in JavaScript?

For instance:

var arrayList = ['a', 'b', 'c', 'd', 'e', 'f'];

How can we empty the array above?

### Answer

There are a couple of ways by which we can empty an array, So let's discuss all the possible way by which we can empty an array.

#### Method 1

arrayList = [];

The code above will set the variable arrayList to a new empty array. This is recommended if you don't have **references to the original array** arrayList anywhere else because It will actually create a new empty array. You should be careful with this way of empty the array, because if you have referenced this array from another variable, then the original reference array will remain unchanged, Only use this way if you have only referenced the array by its original variable arrayList.

For instance:

var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array

var anotherArrayList = arrayList; // Referenced arrayList by another variable

arrayList = []; // Empty the array

console.log(anotherArrayList); // Output ['a', 'b', 'c', 'd', 'e', 'f']

#### Method 2

arrayList.length=0;

The code above will clear the existing array by setting its length to 0. This way of emptying an array will also update all the reference variables that point to the original array.

For instance:

var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array

var anotherArrayList = arrayList; // Referenced arrayList by another variable

arrayList.length=0; // Empty the array by setting length to 0

console.log(anotherArrayList); // Output []

#### Method 3

arrayList.splice(0, arrayList.length);

Above implementation will also work perfectly. This way of empty the array will also update all the references of the original array.

var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array

var anotherArrayList = arrayList; // Referenced arrayList by another variable

arrayList.splice(0, arrayList.length); // Empty the array by setting length to 0

console.log(anotherArrayList); // Output []

#### Method 4

while(arrayList.length) {

arrayList.pop();

}

Above implementation can also empty the array. But not recommended to use often.

## Question 7. How to check if an object is an array or not?

### Answer

The best way to find whether an object is instance of a particular class or not using toString method from Object.prototype

var arrayList = [1 , 2, 3];

One of the best use cases of type checking of an object is when we do method overloading in JavaScript. To understand this, let's say we have a method called greet which can take a single string and also a list of strings. To make our greet method workable in both situation we need to know what kind of parameter is being passed: is it single value or list of values?

functiongreet(param) {

if() {

// here have to check whether param is array or not

}

else {

}

}

However, in the above implementation it might not necessary to check the type of the array, we can check for single value string and put array logic code in else block, let see below code for the same.

functiongreet(param) {

if(typeof param ==='string') {

}

else {

// If param is of type array then this block of code would execute

}

}

Now it's fine we can go with the previous two implementations, but when we have a situation like a parameter can be single value, array, and object type then we will be in trouble.

Coming back to checking the type of an object, As we mentioned that we can use Object.prototype.toString

if(Object.prototype.toString.call(arrayList) ==='[object Array]') {

console.log('Array!');

}

If you are using jQuery then you can also used jQuery isArray method:

if($.isArray(arrayList)) {

console.log('Array');

} else {

console.log('Not an array');

}

FYI jQuery uses Object.prototype.toString.call internally to check whether an object is an array or not.

In modern browser, you can also use:

Array.isArray(arrayList);

Array.isArray is supported by Chrome 5, Firefox 4.0, IE 9, Opera 10.5 and Safari 5

## Question 8. What will be the output of the following code?

var output = (function(x) {

delete x;

return x;

})(0);

console.log(output);

### Answer

The code above will output 0 as output. delete operator is used to delete a property from an object. Here x is not an object it's **local variable**. delete operator doesn't affect local variables.

## Question 9. What will be the output of the following code?

var x =1;

var output = (function() {

delete x;

return x;

})();

console.log(output);

### Answer

The code above will output 1 as output. delete operator is used to delete a property from an object. Here x is not an object it's **global variable** of type number.

## Question 10. What will be the output of the following code?

var x = { foo :1};

var output = (function() {

deletex.foo;

returnx.foo;

})();

console.log(output);

### Answer

The code above will output undefined as output. delete operator is used to delete a property from an object. Here x is an object which has foo as a property and from a self-invoking function, we are deleting the foo property of object x and after deletion, we are trying to reference deleted property foo which result undefined.

## Question 11. What will be the output of the following code?

var Employee = {

company:'xyz'

}

var emp1 =Object.create(Employee);

deleteemp1.company

console.log(emp1.company);

### Answer

The code above will output xyz as output. Here emp1 object got company as **prototype** property. delete operator doesn't delete prototype property.emp1 object doesn't have **company** as its own property. you can test it console.log(emp1.hasOwnProperty('company')); //output : false. However, we can delete company property directly from Employee object using delete Employee.company or we can also delete from emp1 object using \_\_proto\_\_ property delete emp1.\_\_proto\_\_.company.

## Question 12. What is undefined x 1 in JavaScript

var trees = ["redwood", "bay", "cedar", "oak", "maple"];

delete trees[3];

### Answer

* When you run the code above and do console.log(trees); in chrome developer console then you will get ["redwood", "bay", "cedar", undefined × 1, "maple"].
* In the recent versions of Chrome you will see the word empty of undefined x 1.
* When you run the same code in Firefox browser console then you will get ["redwood", "bay", "cedar", undefined, "maple"]

Clearly we can see that Chrome has its own way of displaying uninitialized index in arrays. However when you check trees[3] === undefined in any browser you will get similar output as true.

**Note:** Please remember that you need not check for the uninitialized index of the array in trees[3] === 'undefined × 1' it will give an error because 'undefined × 1' this is just way of displaying an uninitialized index of an array in chrome.

## Question 13. What will be the output of the following code?

var trees = ["xyz", "xxxx", "test", "ryan", "apple"];

delete trees[3];

console.log(trees.length);

### Answer

The code above will output 5 as output. When we used delete operator for deleting an array element then, the array length is not affected by this. This holds even if you deleted all elements of an array using delete operator.

So when delete operator removes an array element that deleted element is no longer present in the array. In place of value at deleted index undefined x 1 in **chrome** and undefined is placed at the index. If you do console.log(trees) output ["xyz", "xxxx", "test", undefined × 1, "apple"] in Chrome and in Firefox ["xyz", "xxxx", "test", undefined, "apple"].

## Question 14. What will be the output of the following code?

var bar =true;

console.log(bar +0);

console.log(bar +"xyz");

console.log(bar +true);

console.log(bar +false);

### Answer

The code above will output 1, "truexyz", 2, 1 as output. Here's a general guideline for the plus operator:

* Number + Number -> Addition
* Boolean + Number -> Addition
* Boolean + Boolean -> Addition
* Number + String -> Concatenation
* String + Boolean -> Concatenation
* String + String -> Concatenation

## Question 15. What will be the output of the following code?

var z =1, y = z =typeof y;

console.log(y);

### Answer

The code above will print string "undefined" as output. According to associativity rule operator with the same precedence are processed based on their associativity property of operator. Here associativity of the assignment operator is Right to Left so first typeof y will evaluate first which is string "undefined" and assigned to z and then y would be assigned the value of z. The overall sequence will look like that:

var z;

z =1;

var y;

z =typeof y;

y = z;

## Question 16. What will be the output of the following code?

// NFE (Named Function Expression)

varfoo=functionbar() { return12; };

typeofbar();

### Answer

The output will be Reference Error. To fix the bug we can try to rewrite the code a little bit:

**Sample 1**

varbar=function() { return12; };

typeofbar();

or

**Sample 2**

functionbar() { return12; };

typeofbar();

The function definition can have only one reference variable as a function name, In **sample 1** bar is reference variable which is pointing to anonymous function and in **sample 2** we have function statement and bar is the function name.

varfoo=functionbar() {

// foo is visible here

// bar is visible here

console.log(typeofbar()); // Works here :)

};

// foo is visible here

// bar is undefined here

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

varfoo=function() {

// Some code

}

functionbar () {

// Some code

}

### Answer

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

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

varfoo=function() {

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

};

// Parse-Time function declaration

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

functionbar() {

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

}

## Question 18. In which case the function definition is not hoisted in JavaScript?

### Answer

Let's take the following **function expression**

varfoo=functionfoo() {

return12;

}

In JavaScript var-declared variables and functions are hoisted. Let's take function hoisting first. Basically, the JavaScript interpreter looks ahead to find all the variable declaration and hoists them to the top of the function where it's declared. For example:

foo(); // Here foo is still undefined

varfoo=functionfoo() {

return12;

};

The code above behind the scene look something like this:

var foo =undefined;

foo(); // Here foo is undefined

foo=functionfoo() {

// Some code stuff

}

var foo =undefined;

foo=functionfoo() {

// Some code stuff

}

foo(); // Now foo is defined here

## Question 19. What will be the output of the following code?

var salary ="1000$";

(function () {

console.log("Original salary was "+ salary);

var salary ="5000$";

console.log("My New Salary "+ salary);

})();

### Answer

The code above will output: undefined, 5000$ because of hoisting. In the code presented above, you might be expecting salary to retain it values from outer scope until the point that salary was re-declared in the inner scope. But due to hoisting salary value was undefined instead. To understand it better have a look of the following code, here salary variable is hoisted and declared at the top in function scope. When we print its value using console.log the result is undefined. Afterwards the variable is redeclared and the new value "5000$" is assigned to it.

var salary ="1000$";

(function () {

var salary =undefined;

console.log("Original salary was "+ salary);

salary ="5000$";

console.log("My New Salary "+ salary);

})();

## Question 20. 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 =newAnimal();

dog instanceof Animal; // Output : true

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

var name =newString("xyz");

name instanceofString; // Output : true

Ref Link: <http://stackoverflow.com/questions/2449254/what-is-the-instanceof-operator-in-javascript>

## Question 21. Calculate the length of the associative array

var counterArray = {

A:3,

B:4

};

counterArray["C"] =1;

### Answer

First of all, in case of JavaScript an associative array is the same as an object. Secondly, even though is no built-in function or property available to calculate the length/size an object, we can write such function ourselves.

#### Method 1

Object has keys method which can we used to calculate the length of object.

Object.keys(counterArray).length; // Output 3

#### Method 2

We can also calculate the length of object by iterating through the object and by doing a count of own property of object. This way we will ignoge the properties that came from the object's prototype chain:

functiongetLength(object) {

var count =0;

for(key in object) {

// hasOwnProperty method check own property of object

if(object.hasOwnProperty(key)) count++;

}

return count;

}

#### Method 3

All modern browsers (including IE9+) support the [getOwnPropertyNames](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames) method, so we can calculate the length using the following code:

Object.getOwnPropertyNames(counterArray).length; // Output 3

#### Method 4

[Underscore](https://underscorejs.org/#size) and [lodash](https://lodash.com/docs/4.17.10#size) libraries have the method size dedicated to calculate the object length. We don't recommend to include one of these libraries just to use the size method, but if it's already used in your project - why not?

\_.size({one:1, two:2, three:3});

=>3

## Question22.Difference between Function, Method and Constructor

## calls in JavaScript.

If your are familiar with Object-oriented programming, More likely familiar to thinking of functions, methods, and class constructors as three separate things. But In JavaScript, these are just three different usage patterns of one single construct.

functions : The simplest usages of function call:

functionhelloWorld(name) {

return"hello world, "+ name;

}

helloWorld("JS Geeks"); // "hello world JS Geeks"

Methods in JavaScript are nothing more than object properties that are functions.

var obj = {

helloWorld:function() {

return"hello world, "+this.name;

},

name:'John Carter'

}

obj.helloWorld(); // // "hello world John Carter"

Notice how helloWorld refer to this properties of obj. Here it's clear or you might have already understood that this gets bound to obj. But the interesting point that we can copy a reference to the same function helloWorld in another object and get a difference answer. Let see:

var obj2 = {

helloWorld :obj.helloWorld,

name:'John Doe'

}

obj2.helloWorld(); // "hello world John Doe"

You might be wonder what exactly happens in a method call here. Here we call the expression itself determine the binding of this this, The expression obj2.helloWorld() looks up the helloWorld property of obj and calls it with receiver object obj2.

The third use of functions is as constructors. Like function and method, constructors are defined with function.

functionEmployee(name, age) {

this.name= name;

this.age= age;

}

var emp1 =newEmployee('John Doe', 28);

emp1.name; // "John Doe"

emp1.age; // 28

Unlike function calls and method calls, a constructor call new Employee('John Doe', 28) creates a brand new object and passes it as the value of this, and implicitly returns the new object as its result.

The primary role of the constructor function is to initialize the object.

## Question 23. What would be the output of the following code?

functionUser(name) {

this.name= name ||"JsGeeks";

}

var person =newUser("xyz")["location"] ="USA";

console.log(person);

### Answer

The output of above code would be "USA". Here new User("xyz") creates a brand new object and created property location on that and USA has been assigned to object property location and that has been referenced by the person.

Let say new User("xyz") created an object called foo. The value "USA" will be assigned to foo["location"], but according to [ECMAScript Specification](http://www.ecma-international.org/ecma-262/6.0/#sec-assignment-operators-runtime-semantics-evaluation) , pt 12.14.4 the assignment will itself return the rightmost value: in our case it's "USA". Then it will be assigned to person.

To better understand what's going on here, try to execute this code in console, line by line:

functionUser(name) {

this.name= name ||"JsGeeks";

}

var person;

var foo =newUser("xyz");

foo["location"] ="USA";

// the console will show you that the result of this is "USA"

## Question 24. What are Service Workers and when can you use them?

### Answer

It’s a technology that allows your web application to use cached resources first, and provide default experience offline, before getting more data from the network later. This principle is commonly known as Offline First.

Service Workers actively use promises. A Service Worker has to be installed,activated and then it can react on fetch, push and sync events.

As of 2017, Service Workers are not supported in IE and Safari.

## Question 25. 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

functionmyFunc() {

// 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:

constmyFunc=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:

classMyAwesomeClass {

myMethod() {

console.log("hi there");

}

}

constobj1=newMyAwesomeClass();

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"

};

varmyFunc=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.

## Question 26. What is IIFE (Immediately Invoked Function Expression) and how it can be useful?

#### Definition

IIFE a function that runs as soon as it's defined. Usually it's anonymous (doesn't have a function name), but it also can be named. Here's an example of IIFE:

(function() {

console.log("Hi, I'm IIFE!");

})();

// outputs "Hi, I'm IIFE!"

#### Explanation

So, here's how it works. Remember the difference between function statements (function a () {}) and function expressions (var a = function() {})? So, IIFE is a function expression. To make it an expression we surround our function declaration into the parens. We do it to explicitly tell the parser that it's an expression, not a statement (JS doesn't allow statements in parens).

After the function you can see the two () braces, this is how we run the function we just declared.

That's it. The rest is details.

* The function inside IIFE doesn't have to be anonymous. This one will work perfectly fine and will help to detect your function in a stacktrace during debugging:
* (functionmyIIFEFunc() {
* console.log("Hi, I'm IIFE!");
* })();

// outputs "Hi, I'm IIFE!"

* It can take some parameters:
* (functionmyIIFEFunc(param1) {
* console.log("Hi, I'm IIFE, "+ param1);
* })("Yuri");

// outputs "Hi, I'm IIFE, Yuri!"

Here there value "Yuri" is passed to the param1 of the function.

* It can return a value:
* var result = (functionmyIIFEFunc(param1) {
* console.log("Hi, I'm IIFE, "+ param1);
* return1;
* })("Yuri");
* // outputs "Hi, I'm IIFE, Yuri!"

// result variable will contain 1

* You don't have to surround the function declaration into parens, although it's the most common way to define IIFE. Instead you can use any of the following forms:
  + ~function(){console.log("hi I'm IIFE")}()
  + !function(){console.log("hi I'm IIFE")}()
  + +function(){console.log("hi I'm IIFE")}()
  + -function(){console.log("hi I'm IIFE")}()
  + (function(){console.log("hi I'm IIFE")}());
  + var i = function(){console.log("hi I'm IIFE")}();
  + true && function(){ console.log("hi I'm IIFE") }();
  + 0, function(){ console.log("hi I'm IIFE") }();
  + new function(){ console.log("hi I'm IIFE") }
  + new function(){ console.log("hi I'm IIFE") }()

Please don't use all these forms to impress colleagues, but be prepared that you can encounter them in someone's code.

#### Applications and usefulness

Variables and functions that you declare inside an IIFE are not visible to the outside world, so you can:

* Use the IIFE for isolating parts of the code to hide details of implementation.
* Specify the input interface of your code by passing commonly used global objects (window, document, jQuery, etc.) IIFE’s parameters, and then reference these global objects within the IIFE via a local scope.
* Use it in closures, when you use closures in loops.
* IIFE is the basis of in the module pattern in ES5 code, it helps to prevent polluting the global scope and provide the module interface to the outside.

## Question 27. Describe Singleton Pattern In JavaScript

### Answer

The singleton pattern is an often used JavaScript design pattern. It provides a way to wrap the code into a logical unit that can be accessed through a single variable. The Singleton design pattern is used when only one instance of an object is needed throughout the lifetime of an application. In JavaScript, Singleton pattern have many uses, they can be used for NameSpacing, which reduce the number of global variables in your page (prevent from polluting global space), organizing the code in a consistent manner, which increase the readability and maintainability of your pages.

There are two important points in the traditional definition of Singleton pattern:

* There should be only one instance allowed for a class and
* We should allow global point of access to that single instance

Let me define singleton pattern in JavaScript context:

It is an object that is used to create namespace and group together a related set of methods and attributes (encapsulation) and if we allow to initiate then it can be initiated only once.

In JavaScript, we can create singleton though object literal. However, there is some another way but that I will cover in next post.

A singleton object consists of two parts: The object itself, containing the members (Both methods and attributes) within it, and global variable used to access it. The variable is global so that object can be accessed anywhere in the page, this is a key feature of the singleton pattern.

**JavaScript: A Singleton as a Namespace**

As I have already stated above that singleton can be used to declare Namespace in JavaScript. NameSpacing is a large part of responsible programming in JavaScript. Because everything can be overwritten, and it is very easy to wipe out variable by mistake or a function, or even a class without even knowing it. A common example which happens frequently when you are working with another team member parallel,

functionfindUserName(id) {

}

/\* Later in the page another programmer

added code \*/

var findUserName =$('#user\_list');

/\* You are trying to call :( \*/

console.log(findUserName())

One of the best ways to prevent accidentally overwriting variable is to namespace your code within a singleton object.

/\* Using Namespace \*/

var MyNameSpace = {

findUserName:function(id) {},

// Other methods and attribute go here as well

}

/\* Later in the page another programmer

added code \*/

var findUserName =$('#user\_list');

/\* You are trying to call and you make this time workable \*/

console.log(MyNameSpace.findUserName());

### Singleton Design Pattern Implementation

/\* Lazy Instantiation skeleton for a singleton pattern \*/

var MyNameSpace = {};

MyNameSpace.Singleton= (function() {

// Private attribute that holds the single instance

var singletonInstance;

// All of the normal code goes here

functionconstructor() {

// Private members

var privateVar1 ="Nishant";

var privateVar2 = [1,2,3,4,5];

functionprivateMethod1() {

// code stuff

}

functionprivateMethod1() {

// code stuff

}

return {

attribute1 :"Nishant",

publicMethod:function() {

alert("Nishant");// some code logic

}

}

}

return {

// public method (Global access point to Singleton object)

getInstance:function() {

//instance already exist then return

if(!singletonInstance) {

singletonInstance =constructor();

}

return singletonInstance;

}

}

})();

// getting access of publicMethod

console.log(MyNamespace.Singleton.getInstance().publicMethod());

The singleton implemented above is easy to understand. The singleton class maintains a static reference to the lone singleton instance and return that reference from the static getInstance() method.

## Question 28. What are the ways of creating objects in JavaScript ?

### Answer

#### 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.

functionEmployee(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 =newEmployee('John', 'Moto', 24, '5000$');

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

var employee1 =newEmployee('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(){

returnthis.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 =newObject(); // Created employee object using new keywords and Object()

employee.name='Nishant';

employee.getName=function(){

returnthis.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.

## Question 29. Write a function called deepClone which takes an object and creates a object copy of it.

var newObject =deepClone(obj);

Solution:

functiondeepClone(object){

var newObject = {};

for(var key in object){

if(typeof object[key] ==='object'&& object[key] !==null )

{

newObject[key] =deepClone(object[key]);

}

else{

newObject[key] = object[key];

}

}

return newObject;

}

**Explanation:** We have been asked to do deep copy of object so What's basically it's mean ??. Let's understand in this way you have been given an object personalDetail this object contains some property which again a type of object here as you can see address is an object and phoneNumber in side an address is also an object. In simple term personalDetail is nested object(object inside object). So Here deep copy means we have to copy all the property of personalDetail object including nested object.

var personalDetail = {

name :'Nishant',

address : {

location:'xyz',

zip :'123456',

phoneNumber : {

homePhone:8797912345,

workPhone :1234509876

}

}

}

So when we do deep clone then we should copy every property (including the nested object).

## Question 30. Best way to detect undefined object property in JavaScript.

Suppose we have given an object person

var person = {

name:'Nishant',

age :24

}

Here the person object has a name and age property. Now we are trying to access the **salary** property which we haven't declared on the person object so while accessing it will return undefined. So how we will ensure whether property is undefined or not before performing some operation over it?

**Explanation:**

We can use typeof operator to check undefined

if(typeof someProperty ==='undefined'){

console.log('something is undefined here');

}

Now we are trying to access salary property of person object.

if(typeofperson.salary==='undefined'){

console.log("salary is undefined here because we haven't declared");

}

## Question 31. Write a function called Clone which takes an object and creates a object copy of it but not copy deep property of object.

var objectLit = {foo :'Bar'};

var cloneObj =Clone(obj); // Clone is the function which you have to write

console.log(cloneObj ===Clone(objectLit)); // this should return false

console.log(cloneObj ==Clone(objectLit)); // this should return true

**solution:**

functionClone(object){

var newObject = {};

for(var key in object){

newObject[key] = object[key];

}

return newObject;

}

## Question 32. What are promises and how they are useful?

We use promises for handling asynchronous interactions in a sequential manner. They are especially useful when we need to do an async operation and THEN do anotherasync operation based on the results of the first one. For example, if you want to request the list of all flights and then for each flight you want to request some details about it. The promise represents the future value. It has an internal state (pending, fulfilled and rejected) and works like a state machine.

A promise object has then method, where you can specify what to do when the promise is fulfilled or rejected.

You can chain then() blocks, thus avoiding the callback hell. You can handle errors in the catch() block. After a promise is set to fulfilled or rejected state, it becomes immutable.

Also mention that you know about more sophisticated concepts:

* async/await which makes the code appear even more linear
* RxJS observables can be viewed as the recyclable promises

Be sure that you can implement the promise, read [one of the articles on a topic](https://opensourceconnections.com/blog/2014/02/16/a-simple-promise-implementation-in-about-20-lines-of-javascript/), and learn the source code of the [simplest promise implementation](https://gist.github.com/softwaredoug/9044640).

## Question 33. How to check whether a key exist in a JavaScript object or not.

Let say we have person object with property **name** and **age**

var person = {

name:'Nishant',

age:24

}

Now we want to check whether name property exist in person object or not ?

In JavaScript object can have own property, in above example name and age is own property of person object. Object also have some of inherited property of base object like toString is inherited property of person object.

So how we will check whether property is own property or inherited property.

Method 1: We can use in operator on objet to check own property or inherited property.

console.log('name'in person); // checking own property print true

console.log('salary'in person); // checking undefined property print false

in operator also look into inherited property if it doesn't find property defined as own property. For instance If I check existence of toString property as we know that we haven't declared this property on person object so in operator look into there base property.

Here

console.log('toString'in person); // Will print true

If we want to test property of object instance not inherited properties then we will use hasOwnProperty method of object instance.

console.log(person.hasOwnProperty('toString')); // print false

console.log(person.hasOwnProperty('name')); // print true

console.log(person.hasOwnProperty('salary')); // print false

## Question 34. What is NaN, why do we need it, and when can it break the page?

NaN stands for “not a number.” and it can break your table of numbers when it has an arithmetic operation that is not allowed. Here are some examples of how you can get NaN:

Math.sqrt(-5);

Math.log(-1);

parseFloat("foo"); /\* this is common: you get JSON from the server, convert some strings from JSON to a number and end up with NaN in your UI. \*/

NaN is not equal to any number, it’s not less or more than any number, also it's not equal to itself:

NaN!==NaN

NaN<2// false

NaN>2// false

NaN===2// false

To check if the current value of the variable is NaN, you have to use the isNaN function. This is why we can often see NaN in the webpages: it requires special check which a lot of developers forget to do.

Further reading: [great blogpost on ariya.io](https://ariya.io/2014/05/the-curious-case-of-javascript-nan)

## Question 35. Fix the bug using ES5 only

var arr = [10, 32, 65, 2];

for (var i =0; i <arr.length; i++) {

setTimeout(function() {

console.log('The index of this number is: '+ i);

}, 3000);

}

For ES6, you can just replace var i with let i.

For ES5, you need to create a function scope like here:

var arr = [10, 32, 65, 2];

for (var i =0; i <arr.length; i++) {

setTimeout(function(j) {

returnfunction () {

console.log('The index of this number is: '+ j)

};

}(i), 3000);

}

## Question 36. How to check if the value of a variable in an array?

We always encounter in such situation where we need to know whether value is type of array or not.

For instance : the code below perform some operation based value type

function(value){

if("value is an array"){

// Then perform some operation

}else{

// otherwise

}

}

Let's discuss some way to detect an array in JavaScript.

**Method 1:**

Juriy Zaytsev (Also known as kangax) proposed an elegant solution to this.

functionisArray(value){

returnObject.prototype.toString.call(value) ==='[object Array]';

}

This approach is most popular way to detecting a value of type array in JavaScript and recommended to use. This approach relies on the fact that, native toString() method on a given value produce a standard string in all browser.

**Method 2:**

Duck typing test for array type detection

// Duck typing arrays

functionisArray(value){

returntypeofvalue.sort==='function';

}

As we can see above isArray method will return true if value object have sort method of type function. Now assume you have created a object with sort method

var bar = {

sort:function(){

// Some code

}

}

Now when you check isArray(bar) then it will return true because bar object has sort method, But the fact is bar is not an array.

So this method is not a best way to detect an array as you can see it's not handle the case when some object has sort method.

**Method 3:**

ECMAScript 5 has introduced **Array.isArray()** method to detect an array type value. The sole purpose of this method is accurately detecting whether a value is an array or not.

In many JavaScript libraries you may see the code below for detecting an value of type array.

function(value){

// ECMAScript 5 feature

if(typeofArray.isArray==='function'){

returnArray.isArray(value);

}else{

returnObject.prototype.toString.call(value) ==='[object Array]';

}

}

## Question 37. Best way to detect reference values of any type in JavaScript ?

In Javascript Object are called as reference type, Any value other then primitive is definitely a reference type. There are several built-in reference type such as **Object**, **Array**, **Function**, **Date**, **null** and **Error**.

Detecting object using typeof operator

console.log(typeof {}); // object

console.log(typeof []); // object

console.log(typeofnewArray()); // object

console.log(typeofnull); // object

console.log(typeofnewRegExp()); // object

console.log(typeofnewDate()); // object

But the downside of using typeof operator to detect an object is that typeof returns object for null (However this is fact that null is an object in JavaScript).

The best way to detect an object of specific reference type using instanceof operator.

Syntax : **value** instanceof **constructor**

//Detecting an array

if(value instanceofArray){

console.log("value is type of array");

}

// Employee constructor function

functionEmployee(name){

this.name= name; // Public property

}

var emp1 =newEmployee('John');

console.log(emp1 instanceof Employee); // true

instanceof not only check the constructor which is used to create an object but also check it's prototype chain see below example.

console.log(emp1 instanceofObject); // true

## Question 38. How does Object.create method works JavaScript?

The ECMAScript 5 **Object.create()** method is the easiest way for one object to inherit from another, without invoking a constructor function.

**For instance:**

var employee = {

name:'Nishant',

displayName:function () {

console.log(this.name);

}

};

var emp1 =Object.create(employee);

console.log(emp1.displayName()); // output "Nishant"

In the example above, we create a new object emp1 that inherits from employee. In other words emp1's prototype is set to employee. After this emp1 is able to access the same properties and method on employee until new properties or method with the same name are defined.

**For instance:** Defining displayName() method on emp1 will not automatically override the employee displayName.

emp1.displayName=function() {

console.log('xyz-Anonymous');

};

employee.displayName(); //Nishant

emp1.displayName();//xyz-Anonymous

In addition to this **Object.create()** method also allows to specify a second argument which is an object containing additional properties and methods to add to the new object.

**For example**

var emp1 =Object.create(employee, {

name: {

value:"John"

}

});

emp1.displayName(); // "John"

employee.displayName(); // "Nishant"

In the example above, emp1 is created with it's own value for name, so calling **displayName()** method will display "John"instead of "Nishant".

Object created in this manner give you full control over newly created object. You are free to add, remove any properties and method you want.

## Question 39. How to use constructor functions for inheritance in JavaScript?

Let say we have Person class which has name, age, salary properties and **incrementSalary()** method.

functionPerson(name, age, salary) {

this.name= name;

this.age= age;

this.salary= salary;

this.incrementSalary=function (byValue) {

this.salary=this.salary+ byValue;

};

}

Now we wish to create Employee class which contains all the properties of Person class and wanted to add some additional properties into Employee class.

functionEmployee(company){

this.company= company;

}

//Prototypal Inheritance

Employee.prototype=newPerson("Nishant", 24,5000);

In the example above, **Employee** type inherits from **Person**. It does so by assigning a new instance of Person to Employeeprototype. After that, every instance of Employee inherits its properties and methods from Person.

//Prototypal Inheritance

Employee.prototype=newPerson("Nishant", 24,5000);

var emp1 =newEmployee("Google");

console.log(emp1 instanceof Person); // true

console.log(emp1 instanceof Employee); // true

Let's understand Constructor inheritance

//Defined Person class

functionPerson(name){

this.name= name ||"Nishant";

}

var obj = {};

// obj inherit Person class properties and method

Person.call(obj); // constructor inheritance

console.log(obj); // Object {name: "Nishant"}

Here we saw calling **Person.call(obj)** define the name properties from Person to obj.

console.log(name in obj); // true

Type-based inheritance is best used with developer defined constructor function rather than natively in JavaScript. In addition to this also allows flexibility in how we create similar type of object.

## Question 40. How we can prevent modification of object in JavaScript ?.

ECMAScript 5 introduce several methods to prevent modification of object which lock down object to ensure that no one, accidentally or otherwise, change functionality of Object.

There are three levels of preventing modification:

**1: Prevent extensions :**

No new properties or methods can be added to the object, but one can change the existing properties and method.

For example:

var employee = {

name:"Nishant"

};

// lock the object

Object.preventExtensions(employee);

// Now try to change the employee object property name

employee.name="John"; // work fine

//Now try to add some new property to the object

employee.age=24; // fails silently unless it's inside the strict mode

**2: Seal :**

It is same as prevent extension, in addition to this also prevent existing properties and methods from being deleted.

To seal an object, we use **Object.seal()** method. you can check whether an object is sealed or not using **Object.isSealed();**

var employee = {

name:"Nishant"

};

// Seal the object

Object.seal(employee);

console.log(Object.isExtensible(employee)); // false

console.log(Object.isSealed(employee)); // true

deleteemployee.name// fails silently unless it's in strict mode

// Trying to add new property will give an error

employee.age=30; // fails silently unless in strict mode

when an object is sealed, its existing properties and methods can't be removed. Sealed object are also non-extensible.

**3: Freeze :**

Same as seal, In addition to this prevent existing properties methods from being modified (All properties and methods are read only).

To freeze an object, use Object.freeze() method. We can also determine whether an object is frozen using Object.isFrozen();

var employee = {

name:"Nishant"

};

//Freeze the object

Object.freeze(employee);

// Seal the object

Object.seal(employee);

console.log(Object.isExtensible(employee)); // false

console.log(Object.isSealed(employee)); // true

console.log(Object.isFrozen(employee)); // true

employee.name="xyz"; // fails silently unless in strict mode

employee.age=30; // fails silently unless in strict mode

deleteemployee.name// fails silently unless it's in strict mode

Frozen objects are considered both non-extensible and sealed.

**Recommended:**

If you are decided to prevent modification, sealed, freeze the object then use in strict mode so that you can catch the error.

For example:

"use strict";

var employee = {

name:"Nishant"

};

//Freeze the object

Object.freeze(employee);

// Seal the object

Object.seal(employee);

console.log(Object.isExtensible(employee)); // false

console.log(Object.isSealed(employee)); // true

console.log(Object.isFrozen(employee)); // true

employee.name="xyz"; // fails silently unless in strict mode

employee.age=30; // fails silently unless in strict mode

deleteemployee.name; // fails silently unless it's in strict mode

## Question 44. Write a log function which will add prefix (your message) to every message you log using console.log ?

For example, If you log console.log("Some message") then output should be **(your message) Some message.**

Logging error message or some informative message is always required when you dealing with client side JavaScript using console.log method. Some time you want to add some prefix to identify message generated log from your application hence you would like to prefix your app name in every console.log.

A general way to do this keep adding your app name in every console.log message like

console.log('your app name'+'some error message');

But doing in this way you have to write your app name everytime when you log message using console.

There are some best way we can achieve this

functionappLog() {

var args =Array.prototype.slice.call(arguments);

args.unshift('your app name');

console.log.apply(console, args);

}

appLog("Some error message");

//output of above console: 'your app name Some error message'

## Question 45 . Write a function which will test string as a literal and as an object ?

For example: We can create string using string literal and using String constructor function.

// using string literal

var ltrlStr ="Hi I am string literal";

// using String constructor function

var objStr =newString("Hi I am string object");

We can use typeof operator to test string literal and instanceof operator to test String object.

functionisString(str) {

returntypeof(str) =='string'|| str instanceofString;

}

var ltrlStr ="Hi I am string literal";

var objStr =newString("Hi I am string object");

console.log(isString(ltrlStr)); // true

console.log(isString(objStr)); // true

## Question 46 . What is typical use case for anonymous function in JavaScript ?

Anonymous functions basically used in following scenario.

1. No name is needed if function is only used in one place, then there is no need to add a name to function.Let's take the example of setTimeout function

setTimeout(function(){

alert("Hello");

},1000);

Here there is no need of using named function when we are sure that function which will alert hello would use only once in application.

1. Anonymous functions are declared inline and inline functions have advantages in the case that they can access variable in the parent scopes.

Let's take a example of event handler. Notify event of particular type (such as click) for a given object.

Let say we have HTML element (button) on which we want to add click event and when user do click on button we would like to execute some logic.

<buttonid="myBtn"></button>

Add Event Listener

var btn =document.getElementById('myBtn');

btn.addEventListener('click', function () {

alert('button clicked');

});

Above example shows used of anonymous function as a callback function in event handler.

1. Passing anonymous function as a parameter to calling function.

Example:

// Function which will execute callback function

functionprocessCallback(callback){

if(typeof callback ==='function'){

callback();

}

}

// Call function and pass anonymous function as callback

processCallback(function(){

alert("Hi I am anonymous callback function");

});

The best way to make a decision for using anonymous function is to ask the following question:

Will the function which I am going to define, be used anywhere else?

If your answer is yes then go and create named function rather anonymous function.

**Advantage of using anonymous function:**

1. It can reduce a bit of code, particularly in recursive function and in callback function.
2. Avoid needless global namespace pollutions.

## Question 47 . How to set a default parameter value ?

If you are coming from python/c# you might be using default value for function parameter incase value(formal parameter) has not been passed. For instance :

// Define sentEmail function

// configuration : Configuration object

// provider : Email Service provider, Default would be gmail

defsentEmail(configuration, provider='Gmail'):

# Your code logic

**In Pre ES6/ES2015**

There are a lot of ways by which you can achieve this in pre ES2015.

Let's understand the code below by which we achieved setting default parameter value.

**Method 1: Setting default parameter value**

functionsentEmail(configuration, provider) {

// Set default value if user has not passed value for provider

provider =typeof provider !=='undefined'? provider :'Gmail'

// Your code logic;

}

// In this call we are not passing provider parameter value

sentEmail({

from:'xyz@gmail.com',

subject:'Test Email'

});

// Here we are passing Yahoo Mail as a provider value

sentEmail({

from:'xyz@gmail.com',

subject:'Test Email'

}, 'Yahoo Mail');

**Method 2: Setting default parameter value**

functionsentEmail(configuration, provider) {

// Set default value if user has not passed value for provider

provider = provider ||'Gmail'

// Your code logic

;

}

// In this call we are not passing provider parameter value

sentEmail({

from:'xyz@gmail.com',

subject:'Test Email'

});

// Here we are passing Yahoo Mail as a provider value

sentEmail({

from:'xyz@gmail.com',

subject:'Test Email'

}, 'Yahoo Mail');

## Question 48. Write code for merge two JavaScript Object dynamically.

Let say you have two objects

var person = {

name :'John',

age :24

}

var address = {

addressLine1 :'Some Location x',

addressLine2 :'Some Location y',

city :'NewYork'

}

Write merge function which will take two object and add all the own property of second object into first object.

merge(person , address);

/\* Now person should have 5 properties

name , age , addressLine1 , addressLine2 , city \*/

**Method 1: Using ES6, Object.assign method**

constmerge= (toObj, fromObj) =>Object.assign(toObj, fromObj);

**Method 2: Without using built-in function**

functionmerge(toObj, fromObj) {

// Make sure both of the parameter is an object

if (typeof toObj ==='object'&&typeof fromObj ==='object') {

for (var pro in fromObj) {

// Assign only own properties not inherited properties

if (fromObj.hasOwnProperty(pro)) {

// Assign property and value

toObj[pro] = fromObj[pro];

}

}

}else{

throw"Merge function can apply only on object";

}

}

## Question 49. What is non-enumerable property in JavaScript and how you can create one?

Object can have properties that don't show up when you iterate through object using for...in loop or using Object.keys() to get an array of property names. This properties is know as non-enumerable properties.

Let say we have following object

var person = {

name:'John'

};

person.salary='10000$';

person['country'] ='USA';

console.log(Object.keys(person)); // ['name', 'salary', 'country']

As we know that person object properties name, salary ,country are enumerable hence it's shown up when we called Object.keys(person).

To create a non-enumerable property we have to use **Object.defineProperty()**. This is a special method for creating non-enumerable property in JavaScript.

var person = {

name:'John'

};

person.salary='10000$';

person['country'] ='USA';

// Create non-enumerable property

Object.defineProperty(person, 'phoneNo',{

value :'8888888888',

enumerable:false

})

Object.keys(person); // ['name', 'salary', 'country']

In the example above phoneNo property didn't show up because we made it non-enumerable by setting **enumerable:false**

Now let's try to change value of phoneNo

person.phoneNo='7777777777';

Changing non-enumerable property value will return error in strict mode. In non-strict mode it won't through any error but it won't change the value of phoneNo.

**Bonus**

**Object.defineProperty()** is also let you create read-only properties as we saw above, we are not able to modify phoneNo value of a person object.

## Question 50. What is Function binding ?

Function binding falls in advance JavaScript category and this is very popular technique to use in conjunction with event handler and callback function to preserve code execution context while passing function as a parameter.

Let's consider the following example:

var clickHandler = {

message:'click event handler',

handleClick:function(event) {

console.log(this.message);

}

};

var btn =document.getElementById('myBtn');

// Add click event to btn

btn.addEventListener('click', clickHandler.handleClick);

Here in this example clickHandler object is created which contain message properties and handleClick method.

We have assigned handleClick method to a DOM button, which will be executed in response of click. When the button is clicked, then handleClick method is being called and console message. Here console.log should log the click event handlermessage but it actually log undefined.

The problem of displaying undefined is because of the execution context of clickHandler.handleClick method is not being saved hence this pointing to button btn object. We can fix this issue using bind method.

var clickHandler = {

message:'click event handler',

handleClick:function(event) {

console.log(this.message);

}

};

var btn =document.getElementById('myBtn');

// Add click event to btn and bind the clickHandler object

btn.addEventListener('click', clickHandler.handleClick.bind(clickHandler));

bind method is available to all the function similar to call and apply method which take argument value of this.

# Coding Questions

## Passing values by reference vs by value

For a JS developer, it's crucially important to understand which values are passed by reference, and which ones are passed by value. Remember that objects, including arrays are passed by reference while strings, booleans and numbers are passed by value.

### 1. What would be the output of following code?

var strA ="hi there";

var strB = strA;

strB="bye there!";

console.log (strA)

The output will 'hi there' because we're dealing with strings here. Strings are passed by value, that is, copied.

### 2. What would be the output of following code?

var objA = {prop1:42};

var objB = objA;

objB.prop1=90;

console.log(objA)

The output will {prop1: 90} because we're dealing with objects here. Objects are passed by reference, that is, objA and objBpoint to the same object in memory.

### 3. What would be the output of following code?

var objA = {prop1:42};

var objB = objA;

objB = {};

console.log(objA)

The output will {prop1: 42}.

When we assign objA to objB, the objB variable will point to the same object as the objB variable.

However, when we reassign objB to an empty object, we simply change where objB variable references to. This doesn't affect where objA variable references to.

### 4. What would be the output of following code?

var arrA = [0,1,2,3,4,5];

var arrB = arrA;

arrB[0]=42;

console.log(arrA)

The output will be [42,1,2,3,4,5].

Arrays are object in JavaScript and they are passed and assigned by reference. This is why both arrA and arrB point to the same array [0,1,2,3,4,5]. That's why changing the first element of the arrB will also modify arrA: it's the same array in the memory.

### 5. What would be the output of following code?

var arrA = [0,1,2,3,4,5];

var arrB =arrA.slice();

arrB[0]=42;

console.log(arrA)

The output will be [0,1,2,3,4,5].

The slice function copies all the elements of the array returning the new array. That's why arrA and arrB reference two completely different arrays.

### 5. What would be the output of following code?

var arrA = [{prop1:"value of array A!!"}, {someProp:"also value of array A!"}, 3,4,5];

var arrB = arrA;

arrB[0].prop1=42;

console.log(arrA);

The output will be [{prop1: 42}, {someProp: "also value of array A!"}, 3,4,5].

Arrays are object in JS, so both varaibles arrA and arrB point to the same array. Changing arrB[0] is the same as changing arrA[0]

### 6. What would be the output of following code?

var arrA = [{prop1:"value of array A!!"}, {someProp:"also value of array A!"}];

var arrB =arrA.slice();

arrB[0].prop1=42;

console.log(arrA);

The output will be [{prop1: 42}, {someProp: "also value of array A!"}, 3,4,5].

The slice function copies all the elements of the array returning the new array. However, it doesn't do deep copying. Instead it does shallow copying. You can imagine slice implemented like this:

functionslice(arr) {

var result = [];

for (i =0; i<arr.length; i++) {

result.push(arr[i]);

}

return result;

}

Look at the line with result.push(arr[i]). If arr[i] happens to be a number or string, it will be passed by value, in other words, copied. If arr[i] is an object, it will be passed by reference.

In case of our array arr[0] is an object {prop1: "value of array A!!"}. Only the reference to this object will be copied. This effectively means that arrays arrA and arrB share first two elements.

This is why changing the property of arrB[0] in arrB will also change the arrA[0].

## Hoisting

### 1. console.log(employeeId);

1. Some Value
2. Undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 4) ReferenceError: employeeId is not defined

### 2. What would be the output of following code?

console.log(employeeId);

var employeeId ='19000';

1. Some Value
2. undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

### 3. What would be the output of following code?

var employeeId ='1234abe';

(function(){

console.log(employeeId);

var employeeId ='122345';

})();

1. '122345'
2. undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

### 4. What would be the output of following code?

var employeeId ='1234abe';

(function() {

console.log(employeeId);

var employeeId ='122345';

(function() {

var employeeId ='abc1234';

}());

}());

1. '122345'
2. undefined
3. '1234abe'
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

### 5. What would be the output of following code?

(function() {

console.log(typeof displayFunc);

vardisplayFunc=function(){

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

}

}());

1. undefined
2. function
3. 'Hi I am inside displayFunc'
4. ReferenceError: displayFunc is not defined

Answer: 1) undefined

### 6. What would be the output of following code?

var employeeId ='abc123';

functionfoo(){

employeeId ='123bcd';

return;

}

foo();

console.log(employeeId);

1. undefined
2. '123bcd'
3. 'abc123'
4. ReferenceError: employeeId is not defined

Answer: 2) '123bcd'

### 7. What would be the output of following code?

var employeeId ='abc123';

functionfoo() {

employeeId ='123bcd';

return;

functionemployeeId() {}

}

foo();

console.log(employeeId);

1. undefined
2. '123bcd'
3. 'abc123'
4. ReferenceError: employeeId is not defined

Answer: 3) 'abc123'

### 8. What would be the output of following code?

var employeeId ='abc123';

functionfoo() {

employeeId();

return;

functionemployeeId() {

console.log(typeof employeeId);

}

}

foo();

1. undefined
2. function
3. string
4. ReferenceError: employeeId is not defined

Answer: 2) 'function'

### 9. What would be the output of following code?

functionfoo() {

employeeId();

var product ='Car';

return;

functionemployeeId() {

console.log(product);

}

}

foo();

1. undefined
2. Type Error
3. 'Car'
4. ReferenceError: product is not defined

Answer: 1) undefined

### 10. What would be the output of following code?

(functionfoo() {

bar();

functionbar() {

abc();

console.log(typeof abc);

}

functionabc() {

console.log(typeof bar);

}

}());

1. undefined undefined
2. Type Error
3. function function
4. ReferenceError: bar is not defined

Answer: 3) function function

## Objects

### 1. What would be the output of following code ?

(function() {

'use strict';

var person = {

name:'John'

};

person.salary='10000$';

person['country'] ='USA';

Object.defineProperty(person, 'phoneNo', {

value:'8888888888',

enumerable:true

})

console.log(Object.keys(person));

})();

1. Type Error
2. undefined
3. ["name", "salary", "country", "phoneNo"]
4. ["name", "salary", "country"]

Answer: 3) ["name", "salary", "country", "phoneNo"]

### 2. What would be the output of following code ?

(function() {

'use strict';

var person = {

name:'John'

};

person.salary='10000$';

person['country'] ='USA';

Object.defineProperty(person, 'phoneNo', {

value:'8888888888',

enumerable:false

})

console.log(Object.keys(person));

})();

1. Type Error
2. undefined
3. ["name", "salary", "country", "phoneNo"]
4. ["name", "salary", "country"]

Answer: 4) ["name", "salary", "country"]

### 3. What would be the output of following code ?

(function() {

var objA = {

foo:'foo',

bar:'bar'

};

var objB = {

foo:'foo',

bar:'bar'

};

console.log(objA == objB);

console.log(objA === objB);

}());

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

### 4. What would be the output of following code ?

(function() {

var objA =newObject({foo:"foo"});

var objB =newObject({foo:"foo"});

console.log(objA == objB);

console.log(objA === objB);

}());

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

### 5. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB =Object.create({

foo:'foo'

});

console.log(objA == objB);

console.log(objA === objB);

}());

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

### 6. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB =Object.create(objA);

console.log(objA == objB);

console.log(objA === objB);

}());

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

### 7. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB =Object.create(objA);

console.log(objA.toString() ==objB.toString());

console.log(objA.toString() ===objB.toString());

}());

1. false true
2. false false
3. true false
4. true true

Answer: 4) true true

### 8. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB = objA;

console.log(objA == objB);

console.log(objA === objB);

console.log(objA.toString() ==objB.toString());

console.log(objA.toString() ===objB.toString());

}());

1. true true true false
2. true false true true
3. true true true true
4. true true false false

Answer: 3) true true true true

### 9. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB = objA;

objB.foo='bar';

console.log(objA.foo);

console.log(objB.foo);

}());

1. foo bar
2. bar bar
3. foo foo
4. bar foo

Answer: 2) bar bar

### 10. What would be the output of following code ?

(function() {

var objA =Object.create({

foo:'foo'

});

var objB = objA;

objB.foo='bar';

deleteobjA.foo;

console.log(objA.foo);

console.log(objB.foo);

}());

1. foo bar
2. bar bar
3. foo foo
4. bar foo

Answer: 3) foo foo

### 11. What would be the output of following code ?

(function() {

var objA = {

foo:'foo'

};

var objB = objA;

objB.foo='bar';

deleteobjA.foo;

console.log(objA.foo);

console.log(objB.foo);

}());

1. foo bar
2. undefined undefined
3. foo foo
4. undefined bar

Answer: 2) undefined undefined

## Arrays

### 1. What would be the output of following code?

(function() {

var array =newArray('100');

console.log(array);

console.log(array.length);

}());

1. undefined undefined
2. [undefined × 100] 100
3. ["100"] 1
4. ReferenceError: array is not defined

Answer: 3) ["100"] 1

### 2. What would be the output of following code?

(function() {

var array1 = [];

var array2 =newArray(100);

var array3 =newArray(['1',2,'3',4,5.6]);

console.log(array1);

console.log(array2);

console.log(array3);

console.log(array3.length);

}());

1. [] [] [Array[5]] 1
2. [] [undefined × 100] Array[5] 1
3. [] [] ['1',2,'3',4,5.6] 5
4. [] [] [Array[5]] 5

Answer: 1) [] [] [Array[5]] 1

### 3. What would be the output of following code?

(function () {

var array =newArray('a', 'b', 'c', 'd', 'e');

array[10] ='f';

delete array[10];

console.log(array.length);

}());

1. 11
2. 5
3. 6
4. undefined

Answer: 1) 11

### 4. What would be the output of following code?

(function(){

var animal = ['cow','horse'];

animal.push('cat');

animal.push('dog','rat','goat');

console.log(animal.length);

})();

1. 4
2. 5
3. 6
4. undefined

Answer: 3) 6

### 5. What would be the output of following code?

(function(){

var animal = ['cow','horse'];

animal.push('cat');

animal.unshift('dog','rat','goat');

console.log(animal);

})();

1. [ 'dog', 'rat', 'goat', 'cow', 'horse', 'cat' ]
2. [ 'cow', 'horse', 'cat', 'dog', 'rat', 'goat' ]
3. Type Error
4. undefined

Answer: 1) [ 'dog', 'rat', 'goat', 'cow', 'horse', 'cat' ]

### 6. What would be the output of following code?

(function(){

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

console.log(array.indexOf(2));

console.log([{name:'John'},{name :'John'}].indexOf({name:'John'}));

console.log([[1],[2],[3],[4]].indexOf([3]));

console.log("abcdefgh".indexOf('e'));

})();

1. 1 -1 -1 4
2. 1 0 -1 4
3. 1 -1 -1 -1
4. 1 undefined -1 4

Answer: 1) 1 -1 -1 4

### 7. What would be the output of following code?

(function(){

var array = [1,2,3,4,5,1,2,3,4,5,6];

console.log(array.indexOf(2));

console.log(array.indexOf(2,3));

console.log(array.indexOf(2,10));

})();

1. 1 -1 -1
2. 1 6 -1
3. 1 1 -1
4. 1 undefined undefined

Answer: 2) 1 6 -1

### 8. What would be the output of following code?

(function(){

var numbers = [2,3,4,8,9,11,13,12,16];

var even =numbers.filter(function(element, index){

return element %2===0;

});

console.log(even);

var containsDivisibleby3 =numbers.some(function(element, index){

return element %3===0;

});

console.log(containsDivisibleby3);

})();

1. [ 2, 4, 8, 12, 16 ] [ 0, 3, 0, 0, 9, 0, 12]
2. [ 2, 4, 8, 12, 16 ] [ 3, 9, 12]
3. [ 2, 4, 8, 12, 16 ] true
4. [ 2, 4, 8, 12, 16 ] false

Answer: 3) [ 2, 4, 8, 12, 16 ] true

### 9. What would be the output of following code?

(function(){

var containers = [2,0,false,"", '12', true];

var containers =containers.filter(Boolean);

console.log(containers);

var containers =containers.filter(Number);

console.log(containers);

var containers =containers.filter(String);

console.log(containers);

var containers =containers.filter(Object);

console.log(containers);

})();

1. [ 2, '12', true ] [ 2, '12', true ] [ 2, '12', true ] [ 2, '12', true ]
2. [false, true] [ 2 ] ['12'] [ ]
3. [2,0,false,"", '12', true] [2,0,false,"", '12', true] [2,0,false,"", '12', true] [2,0,false,"", '12', true]
4. [ 2, '12', true ] [ 2, '12', true, false ] [ 2, '12', true,false ] [ 2, '12', true,false]

Answer: 1) [ 2, '12', true ] [ 2, '12', true ] [ 2, '12', true ] [ 2, '12', true ]

### 10. What would be the output of following code?

(function(){

var list = ['foo','bar','john','ritz'];

console.log(list.slice(1));

console.log(list.slice(1,3));

console.log(list.slice());

console.log(list.slice(2,2));

console.log(list);

})();

1. [ 'bar', 'john', 'ritz' ] [ 'bar', 'john' ] [ 'foo', 'bar', 'john', 'ritz' ] [] [ 'foo', 'bar', 'john', 'ritz' ]
2. [ 'bar', 'john', 'ritz' ] [ 'bar', 'john','ritz ] [ 'foo', 'bar', 'john', 'ritz' ] [] [ 'foo', 'bar', 'john', 'ritz' ]
3. [ 'john', 'ritz' ] [ 'bar', 'john' ] [ 'foo', 'bar', 'john', 'ritz' ] [] [ 'foo', 'bar', 'john', 'ritz' ]
4. [ 'foo' ] [ 'bar', 'john' ] [ 'foo', 'bar', 'john', 'ritz' ] [] [ 'foo', 'bar', 'john', 'ritz' ]

Answer: 1) [ 'bar', 'john', 'ritz' ] [ 'bar', 'john' ] [ 'foo', 'bar', 'john', 'ritz' ] [] [ 'foo', 'bar', 'john', 'ritz' ]

### 11. What would be the output of following code?

(function(){

var list = ['foo','bar','john'];

console.log(list.splice(1));

console.log(list.splice(1,2));

console.log(list);

})();

1. [ 'bar', 'john' ] [] [ 'foo' ]
2. [ 'bar', 'john' ] [] [ 'bar', 'john' ]
3. [ 'bar', 'john' ] [ 'bar', 'john' ] [ 'bar', 'john' ]
4. [ 'bar', 'john' ] [] []

Answer: 1. [ 'bar', 'john' ] [] [ 'foo' ]

### 12. What would be the output of following code?

(function(){

var arrayNumb = [2, 8, 15, 16, 23, 42];

arrayNumb.sort();

console.log(arrayNumb);

})();

1. [2, 8, 15, 16, 23, 42]
2. [42, 23, 26, 15, 8, 2]
3. [ 15, 16, 2, 23, 42, 8 ]
4. [ 2, 8, 15, 16, 23, 42 ]

Answer: 3. [ 15, 16, 2, 23, 42, 8 ]

## Functions

### 1. What would be the output of following code ?

functionfuncA(){

console.log("funcA ", this);

(functioninnerFuncA1(){

console.log("innerFunc1", this);

(functioninnerFunA11(){

console.log("innerFunA11", this);

})();

})();

}

console.log(funcA());

1. funcA Window {...} innerFunc1 Window {...} innerFunA11 Window {...}
2. undefined
3. Type Error
4. ReferenceError: this is not defined

Answer: 1)

### 2. What would be the output of following code ?

var obj = {

message:"Hello",

innerMessage:!(function() {

console.log(this.message);

})()

};

console.log(obj.innerMessage);

1. ReferenceError: this.message is not defined
2. undefined
3. Type Error
4. undefined true

Answer: 4) undefined true

### 3. What would be the output of following code ?

var obj = {

message:"Hello",

innerMessage:function() {

returnthis.message;

}

};

console.log(obj.innerMessage());

1. Hello
2. undefined
3. Type Error
4. ReferenceError: this.message is not defined

Answer: 1) Hello

### 4. What would be the output of following code ?

var obj = {

message:'Hello',

innerMessage:function () {

(function () {

console.log(this.message);

}());

}

};

console.log(obj.innerMessage());

1. Type Error
2. Hello
3. undefined
4. ReferenceError: this.message is not defined

Answer: 3) undefined

### 5. What would be the output of following code ?

var obj = {

message:'Hello',

innerMessage:function () {

var self =this;

(function () {

console.log(self.message);

}());

}

};

console.log(obj.innerMessage());

1. Type Error
2. 'Hello'
3. undefined
4. ReferenceError: self.message is not defined

Answer: 2) 'Hello'

### 6. What would be the output of following code ?

functionmyFunc(){

console.log(this.message);

}

myFunc.message="Hi John";

console.log(myFunc());

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 3) undefined

### 7. What would be the output of following code ?

functionmyFunc(){

console.log(myFunc.message);

}

myFunc.message="Hi John";

console.log(myFunc());

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 2) 'Hi John'

### 8. What would be the output of following code ?

functionmyFunc() {

myFunc.message='Hi John';

console.log(myFunc.message);

}

console.log(myFunc());

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 2) 'Hi John'

### 9. What would be the output of following code ?

functionmyFunc(param1,param2) {

console.log(myFunc.length);

}

console.log(myFunc());

console.log(myFunc("a","b"));

console.log(myFunc("a","b","c","d"));

1. 2 2 2
2. 0 2 4
3. undefined
4. ReferenceError

Answer: a) 2 2 2

### 10. What would be the output of following code ?

functionmyFunc() {

console.log(arguments.length);

}

console.log(myFunc());

console.log(myFunc("a","b"));

console.log(myFunc("a","b","c","d"));

1. 2 2 2
2. 0 2 4
3. undefined
4. ReferenceError

Answer: 2) 0 2 4

## Object Oriented

### 1. What would be the output of following code ?

functionPerson(name, age){

this.name= name ||"John";

this.age= age ||24;

this.displayName=function(){

console.log(this.name);

}

}

Person.name="John";

Person.displayName=function(){

console.log(this.name);

}

var person1 =newPerson('John');

person1.displayName();

Person.displayName();

1. John Person
2. John John
3. John undefined
4. John John

Answer: 1) John Person

## Scopes

### 1. What would be the output of following code ?

functionpassWordMngr() {

var password ='12345678';

this.userName='John';

return {

pwd: password

};

}

// Block End

var userInfo =passWordMngr();

console.log(userInfo.pwd);

console.log(userInfo.userName);

1. 12345678 Window
2. 12345678 John
3. 12345678 undefined
4. undefined undefined

Answer: 3) 12345678 undefined

### 2. What would be the output of following code ?

var employeeId ='aq123';

functionEmployee() {

this.employeeId='bq1uy';

}

console.log(Employee.employeeId);

1. Reference Error
2. aq123
3. bq1uy
4. undefined

Answer: 4) undefined

### 3. What would be the output of following code ?

var employeeId ='aq123';

functionEmployee() {

this.employeeId='bq1uy';

}

console.log(newEmployee().employeeId);

Employee.prototype.employeeId='kj182';

Employee.prototype.JobId='1BJKSJ';

console.log(newEmployee().JobId);

console.log(newEmployee().employeeId);

1. bq1uy 1BJKSJ bq1uy undefined
2. bq1uy 1BJKSJ bq1uy
3. bq1uy 1BJKSJ kj182
4. undefined 1BJKSJ kj182

Answer: 2) bq1uy 1BJKSJ bq1uy

### 4. What would be the output of following code ?

var employeeId ='aq123';

(functionEmployee() {

try {

throw'foo123';

} catch (employeeId) {

console.log(employeeId);

}

console.log(employeeId);

}());

1. foo123 aq123
2. foo123 foo123
3. aq123 aq123
4. foo123 undefined

Answer: 1) foo123 aq123

## Call, Apply, Bind

### 1. What would be the output of following code ?

(function() {

var greet ='Hello World';

var toGreet = [].filter.call(greet, function(element, index) {

return index >5;

});

console.log(toGreet);

}());

1. Hello World
2. undefined
3. World
4. [ 'W', 'o', 'r', 'l', 'd' ]

Answer: 4) [ 'W', 'o', 'r', 'l', 'd' ]

### 2. What would be the output of following code ?

(function() {

var fooAccount = {

name:'John',

amount:4000,

deductAmount:function(amount) {

this.amount-= amount;

return'Total amount left in account: '+this.amount;

}

};

var barAccount = {

name:'John',

amount:6000

};

varwithdrawAmountBy=function(totalAmount) {

returnfooAccount.deductAmount.bind(barAccount, totalAmount);

};

console.log(withdrawAmountBy(400)());

console.log(withdrawAmountBy(300)());

}());

1. Total amount left in account: 5600 Total amount left in account: 5300
2. undefined undefined
3. Total amount left in account: 3600 Total amount left in account: 3300
4. Total amount left in account: 5600 Total amount left in account: 5600

Answer: 1) Total amount left in account: 5600 Total amount left in account: 5300

### 3. What would be the output of following code ?

(function() {

var fooAccount = {

name:'John',

amount:4000,

deductAmount:function(amount) {

this.amount-= amount;

returnthis.amount;

}

};

var barAccount = {

name:'John',

amount:6000

};

varwithdrawAmountBy=function(totalAmount) {

returnfooAccount.deductAmount.apply(barAccount, [totalAmount]);

};

console.log(withdrawAmountBy(400));

console.log(withdrawAmountBy(300));

console.log(withdrawAmountBy(200));

}());

1. 5600 5300 5100
2. 3600 3300 3100
3. 5600 3300 5100
4. undefined undefined undefined

Answer: 1) 5600 5300 5100

### 4. What would be the output of following code ?

(function() {

var fooAccount = {

name:'John',

amount:6000,

deductAmount:function(amount) {

this.amount-= amount;

returnthis.amount;

}

};

var barAccount = {

name:'John',

amount:4000

};

varwithdrawAmountBy=function(totalAmount) {

returnfooAccount.deductAmount.call(barAccount, totalAmount);

};

console.log(withdrawAmountBy(400));

console.log(withdrawAmountBy(300));

console.log(withdrawAmountBy(200));

}());

1. 5600 5300 5100
2. 3600 3300 3100
3. 5600 3300 5100
4. undefined undefined undefined

Answer: 2) 3600 3300 3100

### 5. What would be the output of following code ?

(functiongreetNewCustomer() {

console.log('Hello '+this.name);

}.bind({

name:'John'

})());

1. Hello John
2. Reference Error
3. Window
4. undefined

Answer: 1) Hello John

### 6. Suggest your question!

## Callback Functions

### 1. What would be the output of following code ?

functiongetDataFromServer(apiUrl){

var name ="John";

return {

then:function(fn){

fn(name);

}

}

}

getDataFromServer('www.google.com').then(function(name){

console.log(name);

});

1. John
2. undefined
3. Reference Error
4. fn is not defined

Answer: 1) John

### 2. What would be the output of following code ?

(function(){

var arrayNumb = [2, 8, 15, 16, 23, 42];

Array.prototype.sort=function(a,b){

return a - b;

};

arrayNumb.sort();

console.log(arrayNumb);

})();

(function(){

var numberArray = [2, 8, 15, 16, 23, 42];

numberArray.sort(function(a,b){

if(a == b){

return0;

}else{

return a < b ?-1:1;

}

});

console.log(numberArray);

})();

(function(){

var numberArray = [2, 8, 15, 16, 23, 42];

numberArray.sort(function(a,b){

return a-b;

});

console.log(numberArray);

})();

1. [ 2, 8, 15, 16, 23, 42 ] [ 2, 8, 15, 16, 23, 42 ] [ 2, 8, 15, 16, 23, 42 ]
2. undefined undefined undefined
3. [42, 23, 16, 15, 8, 2] [42, 23, 16, 15, 8, 2] [42, 23, 16, 15, 8, 2]
4. Reference Error

Answer: 1) [ 2, 8, 15, 16, 23, 42 ] [ 2, 8, 15, 16, 23, 42 ] [ 2, 8, 15, 16, 23, 42 ]

## Return Statement

### 1. What would be the output of following code ?

(function(){

functionsayHello(){

var name ="Hi John";

return

{

fullName: name

}

}

console.log(sayHello().fullName);

})();

1. Hi John
2. undefined
3. Reference Error
4. Uncaught TypeError: Cannot read property 'fullName' of undefined

Answer: 4) Uncaught TypeError: Cannot read property 'fullName' of undefined

### 2. What would be the output of following code ?

functiongetNumber(){

return (2,4,5);

}

var numb =getNumber();

console.log(numb);

1. 5
2. undefined
3. 2
4. (2,4,5)

Answer: 1) 5

### 3. What would be the output of following code ?

functiongetNumber(){

return;

}

var numb =getNumber();

console.log(numb);

1. null
2. undefined
3. ""
4. 0

Answer: 2) undefined

### 4\*\*. What would be the output of following code ?

functionmul(x){

returnfunction(y){

return [x\*y, function(z){

return x\*y + z;

}];

}

}

console.log(mul(2)(3)[0]);

console.log(mul(2)(3)[1](4));

1. 6, 10
2. undefined undefined
3. Reference Error
4. 10, 6

Answer: 1) 6, 10

### 5\*\*. What would be the output of following code ?

functionmul(x) {

returnfunction(y) {

return {

result: x \* y,

sum:function(z) {

return x \* y + z;

}

};

};

}

console.log(mul(2)(3).result);

console.log(mul(2)(3).sum(4));

1. 6, 10
2. undefined undefined
3. Reference Error
4. 10, 6

Answer: 1) 6, 10

### 6. What would be the output of following code ?

functionmul(x) {

returnfunction(y) {

returnfunction(z) {

returnfunction(w) {

returnfunction(p) {

return x \* y \* z \* w \* p;

};

};

};

};

}

console.log(mul(2)(3)(4)(5)(6));

1. 720
2. undefined
3. Reference Error
4. Type Error

Answer: 1) 720

…………………………………………………………………………………………………………………………………………………………

Angular JS-

* **Q1.What is Angular JS?**

**AngularJS** is a JavaScript framework that is used for making rich, extensible web applications. It runs on plain JavaScript and HTML, so you don’t need any other dependencies to make it work.AngularJS is perfect for Single Page Applications (SPA). It is basically used for binding JavaScript objects with HTML UI elements.

* **Q2.Explain the architecture of AngularJS?**

AngularJS is architecture on 3 components. They are

* + The Template (View)
  + The Scope (Model)
  + The Controller (Controller)

AngularJS extends HTML attributes with Directives and binds data to HTML with Expressions.

* **Q3.What is the Template in AngularJS?**

The template is the HTML portion of the angular app. It is exactly like a static HTML page, except that templates contain additional syntax which allows data to be injected in it in order to provide a customized user experience.

* **Q4.What is the scope in AngularJS?**

The scope is the object that represents the “model” of your application. It contains fields that store data which is presented to the user via the template, as well as functions which can be called when the user performs certain actions such as clicking a button.

**Q5.What is the controller in AngularJS?**

The controller is a function which generally takes an empty scope object as a parameter and adds to it the fields and functions that will be later exposed to the user via the view.

* **Q6.Explain Directives in AngularJs?**

AngularJS directives are extended HTML attributes with the prefix ng-  
The 3 main directives of angular js are

* + **ng-app:-**directive is used to flag the HTML element that Angular should consider to be the root element of our application. Angular uses spinal-case for its custom attributes and camelCase for the corresponding directives which implement them.
  + **ng-model:-** directive allows us to bind values of HTML controls (input, select, textarea) to application data. When using ngModel, not only change in the scope reflected in the view, but changes in the view are reflected back into the scope.
  + **ng-bind:-**directive binds application modal data to the HTML view.

**Q7.List some tools for testing AngularJS applications?**

For testing AngularJS applications there are certain tools that you should use that will make testing much easier to set up and run.

**Karma**

Karma is a JavaScript command line tool that can be used to spawn a web server which loads your application’s source code and executes your tests. You can configure Karma to run against a number of browsers, which is useful for being confident that your application works on all browsers you need to support. Karma is executed on the command line and will display the results of your tests on the command line once they have run in the browser.

Karma is a NodeJS application and should be installed through NPM/YARN. Full installation instructions are available on the Karma website.

**Jasmine**

Jasmine is a behavior-driven development framework for JavaScript that has become the most popular choice for testing AngularJS applications. Jasmine provides functions to help with structuring your tests and also making assertions. As your tests grow, keeping them well structured and documented is vital, and Jasmine helps achieve this.

Jasmine comes with a number of matches that help you make a variety of assertions. You should read the Jasmine documentation to see what they are. To use Jasmine with Karma, we use the karma-jasmine test runner.

**angular-mocks**

AngularJS also provides the ngMock module, which provides mocking for your tests. This is used to inject and mock AngularJS services within unit tests. In addition, it is able to extend other modules so they are synchronous. Having tests synchronous keeps them much cleaner and easier to work with. One of the most useful parts of ngMock is $httpBackend, which lets us mock XHR requests in tests and return sample data instead.

Source:<https://docs.angularjs.org/guide/unit-testing> 0

**Q8.How do you share data between controllers in AngularJs?**

We can share data by creating a service, **Services** are easiest, fastest and cleaner way to share data between controllers in AngularJs.  
There are also other ways to share data between controllers, they are

* + Using **Events**
  + **$parent, nextSibling, controllerAs**
  + Using the **$rootScope**
* **Q9.Explain AngularJS digest cycle?**

AngularJS digest cycle is the process behind Angular JS data binding.  
In each digest cycle, Angular compares the old and the new version of the scope model values. The digest cycle is triggered automatically. We can also use $apply() if we want to trigger the digest cycle manually.

* **Q10.What is internationalization in Angularjs?**

**Internationalization** is a way to show locale-specific information on a website.It is used to create multilingual language websites.

* **Q11.Difference between AngularJS and JavaScript Expressions?**

Below are some major difference between AngularJS and JavaScript Expressions

* + Both can contain literals, operators, and variables.
  + AngularJS expressions can be written in HTML but JavaScript expressions are not.
  + AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.
  + AngularJS expressions support filters, while JavaScript expressions do not.

**Q12.Explain basic steps to set up an Angular app?**

* + Create an angular.module
  + Assign a controller to the module
  + Link your module to HTML with ng-app
  + Link the controller to HTML with ng-controller directive
* **Q13.What are Angular Modules?**

Angular Modules are place where we write code of our Angular application.Writing Modules makes our code more maintainable, testable, and readable. All dependencies for our app are defined in modules.

**Q14.Explain Directive scopes?**

There are three types of directive scopes available in Angular.

* + **Parent Scope**: is default scope
  + **Child Scope**:  If the properties and functions you set on the scope are not relevant to other directives and the parent, you should probably create a new child scope.
  + **Isolated Scope**: Isolated Scope is used if the directive you are going to build is self-contained and reusable. Does not inherit from parent scope, used for private/internal use.
* **Q15.How to isolate a directive’s Scope in Angular?**

You can isolate a directive’s Scope by passing an object to the scope optionof directive.  
This tells the directive to keep scope inside of itself and not to inherit or share with other scopes.

* **Q16.What is the difference between one-way binding and two-way binding ?**

In One-Way data binding, view (UI part) not updates automatically when data model changed. We need to write custom code to make it updated every time.  
ng-bind has one-way data binding.

While in two way binding scope variable will change it’s value every time its data model changed is assigned to a different value.

* **Q17.How would you make an Angular service return a promise? Write a code snippet as an example**

To add promise functionality to a service, we inject the “$q” dependency in the service, and then use it like so:

angular.factory('testService', function($q){

return {

getName: function(){

var deferred = $q.defer();

//API call here that returns data

testAPI.getName().then(function(name){

deferred.resolve(name)

})

return deferred.promise;

}

}

})

The $q library is a helper provider that implements promises and deferred objects to enable asynchronous functionality

Source: [https://docs.angularjs.org/api/ng/service/$q](https://docs.angularjs.org/api/ng/service/%24q)

* **Q18.Explain the role of $routeProvider in AngularJS?**

The $routeProvider is used to configure roots within an AngularJS application. It can be used to link a URL with a corresponding HTML page or template, and a controller (if applicable).

**Q19.Explain how does Angular implement two-way binding?**

Data-binding in Angular apps is the automatic synchronization of data between the model and view components. The way that Angular implements data-binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. When the model changes, the view reflects the change and vice versa.

* **Q20.What is dependency injection and how does it work?**

AngularJS was designed to highlight the power of dependency injection, a software design pattern that places an emphasis on giving components their dependencies instead of hardcoding them within the component. For example, if you had a controller that needed to access a list of customers, you would store the actual list of customers in a service that can be injected into the controller instead of hardcoding the list of customers into the code of the controller itself. In AngularJS you can inject values, factories, services, providers, and constants.

* **Q21.What is $rootscope? How is it different from the scope?**

In Angular JS $rootscope and $scope both are an object which is used for sharing data from the controller to view.  
The main difference between $rootscope and $scope is that $rootscope is available globally to across all the controllers whereas $scope is available only in controllers that have created it along with its children controllers.

**Q22.What is the difference between $scope and scope?**

In Angular js $scope is used whenever we have to use dependency injection (D.I) whereas as the scope is used for directive linking.

**Q23.What are Angular Expressions?**

Angular js Expression is JavaScript alike code snippets used to bind expression data in view or HTML. Angular expressions are written inside two curly braces.

{{a+b}}

**Q24.List the major browsers Supported by Angular js?**

Below are some major browsers supported by Angular js

* + Mozilla Firefox
  + Google Chrome
  + Microsft Edge
  + IE 10,11
  + IE Mobile,
  + Safari, iOS
  + Android: Nougat (7.0) Marshmallow (6.0)

**Q25.How to enable HTML5 mode in Angular 1.x?**

**html5Mode** method of $locationProvider module is used to enable HTML5 mode in Angular 1.x. For creating pretty URLs and removing # from URLs html5Mode need to true.

Enabling html5Mode to true in Angular 1.x.

angular.module('myApp', [])

.config(function($routeProvider, $locationProvider) {

$routeProvider

.when('/', {

templateUrl : 'partials/home.html',

controller : mainController

})

// use the HTML5 History API

$locationProvider.html5Mode(true);

});

**Q26.List some of the built-in validators in Angular JS?**

Angular js supports all standard HTML5 attributes to validate input.Below are few built-in validators in Angular js.

* + required
  + min
  + max
  + type=”number” OR type=”email”

**Q27.What is Angular’s prefixes $ and $$?**

Angular uses these prefixes to prevent accidental code collision with users code.  
$ prefix is used with public objects whereas $$ prefix is used with a private object.

* **Q28.What are directives? How to create and use custom Directive in Angular js.**

In angular Directives are used to extend the attributes of Html elements.  
Creating custom directive in Angular js.  
Angular js Directives are restricted to element and attribute and created using a directive function.Here is sample code to create a directive in Angular js.

var app = angular.module("myApp", []);

app.controller('AppController', function($scope) {

var users=[];

var user1={};

user1.firstName="Satyam";

user1.lastName="Kumar";

users.push(user1);

var user2={};

user2.firstName="Ravi";

user2.lastName="Sankar";

user2.push(user2);

$scope.users=students;

});

app.directive('user', function() {

//define the directive object

var directive = {};

//restrict = E, implies that directive is Element directive

directive.restrict = 'E';

//element will be replaced by this text/html

directive.template = "First Name: **{{user.firstName}}** , Last Name: **{{user.lastName}}**";

var linkFunction = function($scope, element, attributes) {

element.css("background-color", "#e1e1e1");

}

directive.link=linkFunction;

return directive;

});

As Above directive is restricted to Element directive, so you can use this directive as an element only.  
Usage:

<div ng-app="app">

<h1>Custom Directive Demo</h1>

<div ng-controller="UserController">

<div ng-repeat="user in users">

<user></user>

</div>

</div>

</div>

* **Q29.How to access parent scope from child controller in Angular JS?**

In angular there is a scope variable called $parent (i.e. $scope.$parent). $parent is used to access parent scope from child controller in Angular JS.  
Example:

<div ng-controller="ParentCtrl">

<h1>{{ name }}</h1>

<p>{{ address }}</p>

<div ng-controller="ChildCtrl">

<h1>{{ title }}</h1>

<input type="text" ng-model="$parent.address" />

</div>

**Q30.How to do email and Phone no. validation in Angular JS?**

Angular form validation and ng-pattern directive can be used to validate the email and phone number in Angular JS.

* **Q31.What is the difference between a link and compile in Angular JS?**

**Compile function**: To template DOM manipulation and to gather all the directives, the compile function is used.  
**Link function**: To register DOM listeners as well as for the instance DOM manipulation, the Link function is used.

* **Q32.How can you get URL parameters in Controller?**

The RouteProvider and the RouteParams can be used to get the URL parameters in the controller.  
As the route wires up the URL to the controller and RouteParams can be passed to the controller to get the URL parameters.

* **Q33.How to enable caching in Angular 1.x?**

Caching can be enabled by setting the config.cache value or the default cache value to TRUE or to a cache object that is created with $cacheFactory.In case you want to cache all the responses, then you can set the default cache value to TRUE.  
And, if you want to cache a specific response, then you can set the config.cache value to TRUE.

* **Q34.Explain the use of Ng-If, Ng-Switch, And Ng-Repeat directives?**

**ng-if** – This directive removes a portion of the DOM tree, which is based on the expression.  
In case the expression is assigned to ng-if, it evaluates to a false value, and then the element is deleted from the DOM tree, or else a clone of the element is reinserted into the DOM.  
**ng-switch** – This directive is used based on a scope expression to conditionally swap DOM structure on the template.  
The ng-switch default directive will be preserved at the specific location in a template.  
**ng-repeat**– This directive is used to instantiate the template once per item from a collection.  
Each template which is instantiated gets its own scope where the given loop variable is set to the current collection of item.

* **Q35.How to change start and end symbols used for AngularJS expressions?**

Passing the $interpolateprovider in the config can help us change the start and end symbols used in our Angular JS expressions.

* **Q36.List some difference between Angularjs and Angular 2?**

|  |  |
| --- | --- |
| **Difference between AngularJS and Angular 2** | |
| Angular js or Angular 1.x is based on MVC Architecture | Angular 2 is based on service/ components |
| Javascript is used to write applications in Angular js. | Typescript (superset of javascript) is used to write application in Angular2. |
| Controller are used to write logics and interact with Model and view. | In Angular 2 Controllers are totally eliminated  and Components take its place. |
| Angular 1 is created for developing Single page web-applications. | In Angular 2 can used for developing  native applications for mobile platform  like React Native as well  interative web applications. |
| Angularjs is easy to setup, you just need to include angular.js library to start. | Angular 2 is dependent on other  modules/ package.It give little brainstrom  to install and run Angular 2. |
| In Angularjs ng-app directive is used bootstrap the app. | ng-app is removed in Angular 2.  You need to call Angular2 bootstrap  method to bootstrap your application. |

* **Q37.How to validate an URL in Angular JS?**

Adding the regex directly to the ng pattern to the attribute can help you validate the URL in Angular JS.

* **Q38.What is the use of $locale service in Angular JS?**

locale service provides with the localization rules for Angular JS components.

* **Q39.What is transclusion in Angular JS?**

The transclusion in Angular JS will allow you to move the original children of a directive to a specific location inside a new template.The ng directive marks the insertion point for the transcluded DOM of the very near parent directive that uses transclusion.  
**ng-transclude** or **ng-transclude-slot** attribute directives are used for transclusion in Angular JS.

* **Q40.What is a locale ID in Angular JS?**
* **Q41.What is the difference between a stateful and stateless component in AngularJs?**
* **Q42.List some methods supported by $http service?**
* **Q43.What is “track by” in AngularJS and how does it work?**
* **Q44.Explain $templateCache in Angular Js?**

**Q45.What is the difference between a unit and end-to-end (e2e) test?**

* **Q46.Explain .config() and .run() methods in AngularJS?**
* **Q47.What is a Mocked Service in Angular Js? How to use it?**
* **Q48.How to implement caching in Angular 1.x?**
* **Q49.What are Compile, Pre, and Post linking in AngularJS?**
* **Q50.**

**How to use $scope.$watch and $scope.$apply in AngularJS?**

* **Q51.How to use Multiple ng-app within a page in AngularJS?**
* **Q52.How to set focus on the input field in Angular 1.x?**
* **Q53.Write a code to upload an image in Angular js?**
* **Q54.How to send an Http post request in Angular js?**

**Q55.What is the difference between ‘this’ vs $scope in AngularJS controllers?**

* **Q56.How $location is different from window.location?**
* **Q57.What is the use of ng-cloak directive?**
* **Q58.How to do Animation in AngularJS?**
* **Q59.How to format a date in Angular js?**
* **Q60.What is $emit, $broadcast and $on in AngularJS?**

$broadcast, $emit, and $on are services in Angular Js. Below we have listed why they are used In Angular JS.

**$broadcast()**: $broadcast() service of Angular is used to propagate the event to all of his child controllers and it’s registered parent $rootscope.scope listeners.

$rootScope.$broadcast('SummaryEvent', {

priority: priority

});

**$on()**: AngularJS $on() service is used to listen any type of event raised by $broadcast and $emit.

$scope.$on(''SummaryEvent'', function (event, args) {

Vm.priority=args.priority

});

**$emit**: $emit is similar to $broadcast service but it is used to propagate the event to upwards through the scope hierarchy and notify to the registered $rootScope.Scope listeners.

* **Q61.What are scopeless controller? why to use them?**
* **Q62.What is Isolate Scope and why it is required?**
* **Q63.What is the difference between $evalAsync and $timeout?**
* **Q64.How to disable right click event in Angular js?**
* **Q65.What is a provider in Angular JS?**
* **Q66.Explain Bootstrapping in Angular Js?**
* **Q67.How Angular.js is different from Jquery?**
* **Q68.List and explain different types of directive available in Angular Js?**
* **Q69.What is a dynamic route in Angular js and how do you implement it?**
* **Q70.What is different between ng routing and UI routing?**
* **Q71.Explain UI Routing in AngularJS?**
* **Q72.Explain Routing in Angular Js?**

Routing is one the core feature of AngularJs Framework that is useful in building a single page web application with multiple views. In Angular ngRoute Module is used to implement Routing. **ngView**,**$routeProvider**,**$route** and $routeParams are different components of the **ngRoute** Module that helps to configure and mapping Url to views.

* **Q73.Explain Event in Angular js. List some events you have worked in AngularJs?**
* **Q74.How to create a service in Angular js?**
* **Q75.How to disable a control or element in Angular js?**

[0](https://www.onlineinterviewquestions.com/how-to-disable-a-control-or-element-in-angular-js/)

* **Q76.**

**Explain ng-include directive?**

[0](https://www.onlineinterviewquestions.com/explain-ng-include-directive/)

* **Q77.**

**How to hide or show a div in Angular js?**

[0](https://www.onlineinterviewquestions.com/how-to-hide-or-show-a-div-in-angular-js/)

* **Q78.**

**How to create a service using Factory?**

[0](https://www.onlineinterviewquestions.com/how-to-create-a-service-using-factory/)

* **Q79.**

**List some inbuilt services in AngularJs ?**

There are 30 inbuilt **services in AngularJs**. Below are few most used services in AngularJs.

* + $location
  + $scope
  + $http
  + $timeout
  + $interval
  + $window

[0](https://www.onlineinterviewquestions.com/list-some-inbuilt-services-in-angularjs/)

* **Q80.**

**What are filters in AngularJS ? Can you list some?**

[0](https://www.onlineinterviewquestions.com/what-are-filters-in-angularjs-can-you-list-some/)

* **Q81.**

**List some Cons or disadvantages of using AngularJS?**

[0](https://www.onlineinterviewquestions.com/list-some-cons-or-disadvantages-of-using-angularjs/)

* **Q82.**

**Explain DDO in AngularJs?**

DDO Stands for Directive Definition Object. DDO is used while creating a custom directive in AngularJs.

[0](https://www.onlineinterviewquestions.com/explain-ddo-in-angularjs/)

* **Q83.**

**What is an interceptor in Angular? Why it is used?**

An interceptor is a middleware code in AngularJs where all the $http requests go through. It is attached with $httpProvider service and able to intercept request and response objects. Interceptor Middleware is useful for error handling, authentication and other filters you want to apply on request and response.

* **Q84.Explain the functionality angular.copy() method?**

### Q85.What is the difference between observable and promises?

The differences between observable and promises are:

1. Observable is a more powerful way of handling HTTP asynchronous requests. Whereas, A promise handles a single event when an asynchronous operation completes or fails.
2. An observable is like a stream which allows passing zero or more events where the callback is called for each event. Whereas, A promise eventually calls the success or failed callback even when you don’t need the notification or the result it provides anymore.
3. Observable works with multiple values for a particular time. Whereas, Promises works with and even returns a single value at a time.
4. Observables can be canceled. Whereas, Promises cannot be canceled.
5. Observable supports map, filter, reduce and similar operators. Whereas, Promises have more readable codes with try/catch and async/await.
6. In observable, one operator ‘retry’ can be used to retry whenever needed. Whereas, Promises cannot be retried. A promise should have access to the original function that returned the promise in order to have a retry capability.

### 86. List the differences between Angular 2 components vs. directives.

Apart from components, directives are also used in Angular 2 which allows us to attach behavior to elements in DOM. There are certain differences between the components and directives in Angular 2. They are:

1. In Angular 2, a component is a directive with a view whereas a directive is a decorator with no view. Components are the specific type of directive that allows us to utilize web component functionality throughout our application. Whereas, Directive is the mechanism by which we attach behavior to elements.
2. A component is used to break up the application into smaller components. Whereas, Directive is used to design the re-usable components.
3. Components can be used to define pipes. Whereas, We cannot define pipes using directives.
4. Components can be present per DOM element. Whereas, Directive is used to add behavior to an existing DOM element.