JAVASCRIPT

1.Who developed Javascript and when was it introduced ?
Javascript was introduced in 1995 to add programs to web pages by Netscape.

2.Is Javascript a single threaded or multithreaded?
Javascript is single threaded but we can make it as multi threaded using HTML5 web-workers.

3.Is Javascript Asynchronous ?
Javascript is not entirely asynchronous only certain parts of Javascript are asynchronous.The difference between synchronous code and asynchronous code is that synchronous code executes from the top of a code block to the bottom in the order it was written.
In case of asynchronous code,the file sends the request to the server for a resource.

4.In Javascript,how many ways an object can be created ?
(i)Using an object literal
(ii)Using new keyword
(iii)Using an objet constructor

5.In Javascript,what are the three types of errors ?
The three types of errors in Javascript are :
(i) Syntax errors,also called as parsing errors,occur "at compile time in traditional programming languages" and "at interpret time in Javascript"
(ii) Runtime errors,also called as exceptions,occur during execution (that is after compilation or interpretation)
(iii) Logic errors are the errors occur when you mistake in the logic that drives your script and you do not get the result you expected.

6.What is a isNan function in JavaScript ?
isNaN function determines whether a value is Not- a -Number.It returns true if the value is Not-a-Number,otherwise false.

7.Can you explain the concept of truthy and falsy in Javascript ?
Nan returns “true" when it is not a number.The truthy values are the values other than falsy values.
Nan returns “false" when it is a number.The falsy values are false,0,empty string,null,undefined,Nan.

8.What is JsHint ?
Js Hint is a community Driven tool to detect errors and potential problems in Javascript code (that is to find some red flags and bugs).

9.Can you explain eventCapturing and eventBubbling ,which one of the two is prefered and why ?
Event bubbling and EventCapturing are two ways of event propagation in the HTML DOM API.
With EventBubbling,the event is first captured and handled by the inner most element and then propagated to the outer elements.
With EventCapturing,the event is first captured by the outermost element and propagated to the inner elements.
EventBubbling is preferred over EventCapturing as it supports in all IE browsers.

10.How can you register an event handler ?
addEventListener (type, listener, useCapture) - is used to register an event handler.
To use Event capturing, set “useCapture" to “true”

11.How can you stop an Event bubbling or Event capturing ?
event.cancelBubble = true - for IE
event.stopPropagation( ) - for other browsers

12.What is on-ready function ?
on-ready function only waits for DOM tree,and doesn’t wait for images or external resource loading,that means on-ready function is faster.
we can have on-ready function multiple times in a page.

13.What is on-load function ?
on-load function not only waits for DOM to be created but also waits until all external resources are fully loaded including heavy images,audios and videos.
we can have only one "on-load function" in a page.

14.What is the difference between Call method and Apply method in JavaScript ?
In Call method,we have to pass comma separated arguments.In Apply method,we have to pass an array.
Call method is used, if you know the number of arguments or there are no arguments to be passed.
Apply method is used,if you don’t know the number of arguments to be passed or the argument object is already an array.

15.What is Hoisting in JavaScript ?
Hoisting is Javascript’s default behavior of moving all declarations to the top of the current scope,that is,to the top of the current script or current function.

16.In Javascript,what is the difference between Undefined and Null ?
type of undefined - returns undefined.
type of null - returns object
null = = = undefined - returns false
null = = undefined - returns true

17.What is the difference between = = and = = = operators ?
= = operator is used for comparing the values.
= = = operator is used for comparing the values as well as data types.

18.What is the difference between pass by value and pass by reference ?
Primitives (Numbers,Boolean) are passed by value.
Objects are passed by reference.

19.What is the difference between undeclared,undefined and null ?
undeclared: variables don't even exist
undefined: variables exist,but nothing is assigned to them
null: variables exist and have null assigned to them

20.What are the data types in Javascript ?
Basically,there are two types of data types in Javascript,they are:
Primitive values and Reference values
Primitive values: String,Number,Boolean,Null,Undefined
Reference values : Array,Object
Primitive values : are stored in a stack
Reference values : are stored in a heap
(i)Number
(ii)Null
(iii)Undefined
(iv)String
(v)Array
(vi)Object
(vii)Boolean

21.What is positive infinity in Javascript ?
If the value returned is higher than the value that Javascript can handle then the value is called as positive infinity.

22.What is negative infinity in Javascript ?
If the value returned is smaller than the value that Javascript can handle then the value is called as negative infinity.

23.What are truthy and falsy values in Javascript ?
The truth values are:
"0"
"false"
empty function
empty object
empty array

The falsy values are:
0
null
false
undefined
NAN
“ “

24.What is a prototype in Javascript ?
A prototype is an object from which other objects can be created.Javascript doesn’t support classical inheritance.Javascript supports only prototype-based inheritance.

25.What is the difference between AngularJs expressions and Javascript expressions ?
Both AngularJs expressions and Javascript expressions can contain literals,operators,and variables.
Unlike Javascript expressions,AngularJs expressions can be written inside HTML.
Unlike Javascript expressions,AngularJs expressions don’t support conditionals,loops.
Unlike Javascript expressions,AngularJs expressions support filters.

26.What is push( ) method in Javascript ?
push( ) method is used to add new items to the end of an array,and returns the new length.
array.push (item1,item2,…..)

27.What is unshift( ) method in Javascript ?
unshift( ) method is used to add new items to the beginning of an array,and returns the new length.
array.unshift (item1, item2,…..)

28.What is pop( ) method in Javascript ?
pop( ) method is used to remove the last item of an array,and returns that item.
array.pop( )

29.What is shift( ) method in Javascript ?
shift( ) method is used to remove the first item of an array,and returns that item.
array,shift( )

30.What is the difference between shift( ) and unshift( ) methods ?
unshift( ) method adds one or more elements to the beginning of an array and returns the new length of the array.
shift( ) method removes the first element from an array and returns that element.

31.What are Javascript design Patterns ?
Adapter Pattern

Bridge Pattern
Command Pattern
Composite Pattern
Decorator Pattern
Factory Pattern
Facade Pattern
Observer Pattern
Proxy Pattern
Singleton Pattern

32.What is a singleton design pattern ?
Singletons reduce "the need for global variables" which is very important because it avoids the risk of name collisions.

33.What is Strict mode in Javascript ?
Strict mode makes several changes to normal Javascript semantics.
First,strict mode eliminates some Javascript silent errors by changing them to throw errors.
Second,strict mode fixes mistakes that make it difficult for Javascript engines to perform optimizations(that is,strict mode can sometimes be made to run faster than identical code that is not strict mode)
Third,strict mode prohibits some syntax likely to be defined in future versions of ECMAScript.

Strict mode applies to entire scripts or individual functions.Strict mode doesn't apply to block statements enclosed in { } braces;attempting to apply strict mode to such contexts does nothing.

To apply strict mode for an entire script,put "use strict" before any other statements.
Concatenation of strict mode scripts with each other is fine,and concatenation of non-strict mode scripts with each other is fine,but concatenating strict mode script and non-strict mode script is problematic,so it is recommended to enable strict mode on a function-by-function basis(at least during the transition period),otherwise it is recommended to wrap the entire contents of a script in a function and having that outer function use strict mode,this eliminates the concatenation problem.

To apply strict mode for a function,put "use strict" in the function's body before any other statements.
If strict mode is applied for a function,then it is no longer possible to reference the "window" object through "this" keyword.

34.What is a closure ?
A closure is a special kind of object that combines two things: a function, and the environment in which that function was created.The environment consists of any local variables that were in-scope at the time that the closure was created.
A closure lets you associate some data(the environment) with a function that operates on that data.
A closure is a function having access to the parent scope,even after the parent function has closed.
Closures make it possible for a function to have private variables.
Using closures,we can emulate private methods in Javascript.Private methods aren't just useful for restricting access to code but they also provide a powerful way of managing your global namespace,keeping non-essential methods from cluttering up the public interface to your code.

35.What are the problems if I use closures ?
Closures affect script performance both in terms of processing speed and memory consumption.

36.What are the issues with a Closure ?
A Closure can cause memory leaks in IE.Javascript is a garbage collected language - that is,objects are allocated memory upon their creation and that memory is reclaimed by the browser when no reference to an object remain. IE uses its own garbage collection scheme for this,separate from the mechanism used for Javascript.It is the interaction between the two that can cause memory leaks.
A memory leak in IE occurs any time a circular reference is formed between a Javascript object and a native object. The "circular reference" is created between a Javascript object(that is,a function) and a native object.This "circular reference" introduced by a Closure can be broken by adding another closure.

37.How to avoid closures ?
Closures can be avoided by breaking "circular references" during the “window.onunload” event.Many event libraries will generally do this for us,but doing so disables “back-forward cache in FireFox”,so we should not register an “unload” listener in FireFox.

38.How can we create a cookie in Javascript ?
Cookies are invented by Netscape to give “memory” to web servers and browsers.HTTP is a stateless,this problem can be overcome by using “cookies”.
A cookie can be created,read,and deleted using the property “document.cookie”.We can add expiry date for a cookie,but by default a cookie will be deleted when the browser is closed.We can add a path parameter which tells the browser what path the cookie belongs to,but by default a cookie belongs to the current page.

39.What is a promise ?
A promise represents the result of an asynchronous operation.A promise can be in one of the three different states:
pending - is the initial state of a promise
fulfilled - represents the successful operation
rejected - represents the failed operation
once a promise is fulfilled or rejected,it is immutable(that is,it can never be changed)

40.What does a promise do ?
promises help you naturally handle errors,and write cleaner code by not having “callback” parameters,and without modifying the underlying architecture(that is,we can implement them in pure Javascript and use them to wrap the existing asynchronous operations)

41.How to construct a promise ?
To construct a promise,we use "new promise”.We give the constructor a "factory function” which does the actual work.This "factory function” is called immediately with two arguments.The first argument fulfills the promise and the second argument rejects the promise.Once the operation is completed,we call the appropriate function.

42.What is “promise.done" ?
Inorder to use a promise,we must somehow be able to wait for the promise to be fulfilled or rejected.The way to do this is using “promise.done”.To support “promise.done” in our browser,we need to polyfill the following:
<script src = “ https : / / www.promisejs.org / polyfills / promise - 7.0.1.min.js “ > </script>

43.How can we achieve transformation and chaining of a promise ?
Promises have a method for transformation and chaining of a promise.
Put simply, .then is to .done as .map is to .forEach,that is,we use ".then” whenever we want to do something with the result and we use “.done” whenever we want to do nothing with the result.

44.What kind of loops does Javascript support ?
Javascript supports:
for loop
for….. in loop
for….. of
while loop
do….. while loop
break
continue

45.What is the difference between break statement and continue statement ?
(i)break statement breaks the loop and continues executing the code after the loop.
continue statement breaks one iteration(in the loop),if a specified condition occurs,and continues with the next iteration in the loop.

(ii)When you use break statement without a label,it terminates the innermost enclosing loop (that is,“while,do-while,for,or switch”) immediately and transfers control to the following statement.
When you use break statement with a label,it terminates the specified labeled statement.

(iii)When you use continue statement without a label,it terminates the current iteration of the innermost enclosing loop and continues executing of the loop with the next iteration.
When you use continue statement with a label,it applies to the looping statement identified with that label.

46.What are the different types of Popup boxes you can create in Javascript ?
window.alert (“some text”) - Alert box - when an Alert box pops up,user will have to click “OK” to proceed
window.confirm(“some text) - confirm box - when a Confirm box pops up,user will have to click either “OK” or “cancel” to proceed.If user clicks “OK”,the box returns true.If user clicks “cancel”,the box returns false.
window.prompt(“some text”) - prompt box - when a prompt box pops up,user will have to click either “OK” or “cancel” to proceed after entering an input value.if the user clicks “OK”,the box returns the input value.If the user clicks “cancel”,the box returns null.

47.What happens if VAR isn’t declared in Javascript ?
Assigning a value to an undeclared variable implicitly creates it as a global variable (that is,it becomes a property of the global object)when the assignment is executed.

48.What is the difference between declared variables and undeclared variables ?
Declared variables are constrained in the execution context in which they are declared,while undeclared variables are always global.
Declared variables are created before any code is executed,while undeclared variables don't exist(that is,not created) until the code assigning to them is executed.
Declared variables are a non-configurable property of their execution context,while undeclared variables are configurable(that is,they can be deleted)

49.What are Timing events in Javascript ?
Both setInterval( ) and setTimeout( ) methods are window objects.
setInterval( ) method executes a function,over and over again,at specified time intervals.
window.setInterval ("function", milliseconds)
To stop the further execution of the function specified in the setInterval( ) method,use clearInterval( ) method.
window.clearInterval (intervalVariable)

setTimeout( ) method executes a function once,after waiting a specified number of milliseconds.
window.setTimeout ("function", milliseconds)
To stop the further execution of the function specified in the setInterval( ) method,use clearInterval( ) method.
window.clearTimeout (timeoutVariable)

setImmediate( ) method executes a function immediately after the browser has completed other operations,such as events and display updates.
window.setImmediate("function", 0)
That is,setImmediate( ) method is used instead of setTimeout("function", 0)
To stop the immediate actions,use clearImmediate( ) method.
window.clearImmediate(immediateID)
where immediateID is an ID returned by "setImmediate( )" method.

50.What is the difference between Dot notation and Bracket notation when accessing an object's property ?
Dot notation is used :
when we know the name of the property.
when the property name contains a valid Javascript identifier,that is,a sequence of alphanumerical characters,also including the underscore and dollar sign,that cannot start with a number.For an instance,object.$1 is valid, object.1 is invalid

Bracket notation is used :
when the name of the property is contained in a variable.
when the property name contains "characters not permitted in identifiers",such as starting with a digit,containing a space or dash.
obj.foo - is valid
obj [foo] - is valid
obj.45 - is invalid
obj [45] - is valid
obj.foo-bar - is invalid
obj [foo-bar] - is valid

The only way to access the object's property using the for-in loop,is using a bracket notation,you can't use dot notation.

51.How can you handle runtime errors in Javascript ?
The runtime errors(also called as exceptions) can be handled using try / catch / finally exception handling statements.
The runtime errors occur due to an illegal operation,that is,referencing an undefined variable or calling a non-existent method.
try - tests a block of code for errors
catch - handles the errors
throw - creates the custom errors
finally - executes the code,after try and catch,regardless of the result.

52.How can you debug Javascript ?
Debugging is the process of testing,finding,and reducing the bugs or errors in computer programs.
Activate debugging in your browser with F12,and select "console" in the debugger menu.

53.What is a Debugger statement in Javascript ?
we can place a debugger statement anywhere in a program to stop execution.Using the debugger statement is similar to setting a breakpoint in the code.

54.What is a global variable and what are the problems associated with them ?
A global variable is a variable that is available through out the length of the code,that is,a global variable has no scope.
The problem associated with the global variables is: the clash of variable names of local and global scope.
It is difficult to debug and test the code that relies on global variables.

55.What is 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".
i = 10;
i = "string"
This is called variable typing

56.How can you convert the string of any base to integer in Javascript ?
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.

57.What is NULL in Javascript ?
NULL represents no value.

58.What is the function of delete operator ?
The delete operator deletes all variables and objects in a program,but it cannot delete variables declared with VAR keyword.

59.Which symbol is used for comments in Javascript ?
For single line comments : //
For multi line comments : /* ….*/

60.What is a wrapper object in Javascript ?
whenever we try to access a property of a string,Javascript forces the value of a string to an object by “new String (str)".This object is called a “wrapper object”.

```
var str = ‘hello’ ;
str.custom = 1;
```

Here,Javascript creates a wrapper string object,sets its custom property to 1,and then discards it.

```
var str = ‘hello’;
var temp = new String(str);          // wrapper object
temp.custom = 1;
```

For the three primitives values,that is Boolean type,Number type, and String type, three special references are created.
Every time a primitive value is read,the following happens :
(i) create an instance of the string type
(ii) call the specified method on the instance
(iii) destroy the instance

61.What is the difference between session state and view state in Javascript ?
view state is maintained in page level only.
view state of one page isn’t visible in another page.
view state information is stored in client only.
view state persist the values of particular page in the client(browser) when post operation is done.
session state is maintained in session only.
session state value is available in all pages within a user session.
session state information is stored in server.
session state persist the data of particular user in the server.This data will be available till user closes the browser or session time completes.

62.Is Javascript case sensitive ?
Yes Javascript is case sensitive.For an instance,a function parseInt isn’t same as Parseint.

63.What is Namespacing in Javascript ?
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.

Using global variables in Javascript is evil and a bad practice. That being said, namespacing is used to bundle up all your functionality using a unique name. In JavaScript, a namespace is really just an object that you've attached all further methods, properties and objects. It promotes modularity and code reuse in the application.

64.What is the use of void(0) ?
void(0) is used to prevent the page from refreshing and parameter "0" is passed while calling.
void(0) is used to call another method without refreshing the page.

65.What is the type of variables in Javascript ?
All variables available in Javascript are "object data types".

66.What is the difference between an alert box and a confirmation box ?
An alert box displays only one button which is "OK button".
A confirmation box displays two buttons namely "OK" and "cancel".

67.How can you find the operating system in the client machine using Javascript ?
"navigator.appversion" is used to find the OS in the client machine using Javascript.

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

69.What are the two basic datatypes in Javascript ?
Primitive type and Reference types are the two basic datatypes in Javascript.

70.What is the difference between Javascript and Jscript ?
Both Javascript and Jscript are similar.Javascript was developed by Netscape whereas Jscript was developed by Microsoft.

71.What is the disadvantage of using innerHTML in Javascript ?
(i) content is replaced every time
(ii) we cannot use like "appending to innerHTML"
(iii) Even if we use "innerHTML = innerHTML + 'html' " still the old content is replaced by html.
(iv) The entire innerHTML content is re-parsed and build into elements,therefore its much slower.
(v) The innerHTML doesn't provide validation and therefore we can potentially insert invalid and broken HTML in the document and break it.

72.How are object properties assigned in Javascript ?

Assigning "properties to an object" is similar to assigning "values to a variable".

73.What is the difference between a break and continue statements ?
A break statement is used to come out of the current loop while the continue statement is used to continue the current loop with a new recurrence.

74.What are screen objects in Javascript ?
screen objects are used to read the information from the client's screen.The properties of a screen object 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 clients' screen,including the taskbar
Width - gives the total width of the client's screen,including the task bar

75.What is the difference between escape( ) and unescape( ) functions ?
escape( ) function is responsible for coding a string so as to transfer the information from one computer to other computer,across a network.
unescape( ) function is responsible for decoding a "coded string".

76.What is the difference between decodeURI( ) and encodeURI( ) ?
encodeURI( ) is used to convert URI into their Hex coding.
decodeURI( ) is used to convert the encoded URI back to normal.

77.What are window objects and navigator objects in Javascript ?
window object is the top level object in Javascript.window object contains several other objects such as document,history,location,name,menu bar,etc.,in itself.window object is the global object for Javascript that is written at client-side.
navigator.appName - gives the browser name
navigator.appVersion - gives the browser version
navigator.appCodeName - gives the browser code name
navigator.platform - gives the platform on which the browser is running

78.How can you read and write a file in javascript ?
A file can be read and written by using the following javascript functions : fopen( ), fread( ), fwrite( )
fopen( ) takes two parameters : path and mode (0 for reading and 3 for writing)
fopen( ) returns 1,if the file is successfully opened.
file = fopen(getScriptPath( ),0)
str = fread(file,flength(file))
file = fopen ("c:\MyFile.txt", 3)
fwrite(file,str)

Using ActiveXObject

var fso = new ActiveXObject (“Scripting.FileSystemObject”)
var s = fso.OpenTextFile(“C:\\example.txt”,1,true)

79.What is this keyword ?
“this" is the object that owns the current code.
The value of “this",when used in a function,is the object that owns the function.

80.How many ways we can invoke a function ?
A function can be invoked in four ways :
(i)Invoking a function as a function
(ii)Invoking a function as a method
(iii)Invoking a function with a function constructor (that is,using a new keyword before a function name)

(iv)Invoking a function with a function method(that is,using call method and apply method)
Both the call method and apply method takes an owner object as the first argument.The only difference is that call method takes the function arguments separately,and apply method takes the function arguments in an array.
In Javascript strict mode,the first argument becomes the value of “this" in the invoked function,even if the argument isn’t an object.
In non-strict mode,if the value of first argument is null or undefined,it is replaced with the global object.

81.What is anonymous function ?
An anonymous function is a function that is declared without a name.Once an anonymous function is created,it can’t be accessible.

82.What is a self invoking function ?
A self invoking function is invoked automatically without being called.
A self invoking function is a function,which is enclosed in parenthesis and followed by an empty parenthesis.

83.How can we convert a variable into a number in Javascript ?
In Javascript,a variable can be converted into a number by using three methods:
Number( ) - is used to convert Javascript variables into numbers
parseInt( ) - parses a string and returns a whole number.spaces are allowed.only the first number is returned.If the number cannot be converted,parseInt( ) returns NaN (Not a Number)
parseFloat( ) - parses a string and returns a number.spaces are allowed.only the first number is returned.If the number cannot be converted,parseFloat( ) returns NaN (Not a Number)
valueOf( ) - returns a number.
valueOf( ) is used internally in Javascript to convert "number objects" to "primitive values".
In Javascript,a number can be a primitive value or an object.

84.What are "number methods" in Javascript ?
toString( ) - returns a number as a string
toExponential( ) - returns a string,with a number rounded and written using exponential notation.
toFixed( ) - returns a string,with a number rounded and written with a specified number of decimals.

toPrecision( ) - returns a string,with a number written with a specified length.
valueOf( ) - returns a number as a number.

85.What is a regular expression in Javascript ?
A regular expression is a sequence of characters that forms a "search pattern".
A regular expression is used to perform all types of "text search" and "text replace" operations.
syntax: / pattern / modifiers
var a = / w3schools / i ;
/w3schools / i - is a regular expression
w3schools - is a pattern (to be used in a search)
i - is a modifier (modifies the search to be case-insensitive)
A regular expression is often used with two string methods :
search( ) - uses an expression to search for a match,and returns the position of the match
replace( ) - returns a modified string where the pattern is replaced.

86.What are regular expression methods ?
The regular expression methods are :
test( ) - searches a string for a pattern, and returns "true" or "false",depending on the result.
exec( ) - searches a string for a specified pattern,and returns the found text.

87.What are Array methods ?
concat( ) - returns a new array comprised of this array joined with other array.
every( ) - returns true if every element in this array satisfies the provided testing function
filter( ) - creates a new array with all of the elements of this array for which the provided filtering function returns true.
forEach( ) - calls a function for each element in the array.
indexOf( ) - returns the first (least) index of an element with an array equal to the specified value, or -1 if none is found.
join( ) - joins all elements of an array into a string.
lastIndexOf ( )- returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
map( ) - creates a new array with the results of calling a provided function on every element in this array.
pop( ) - removes the last element from an array and returns that element.
push( ) - adds one or more elements to the end of an array and returns the new length of the array.
reduce( ) - applies a function simultaneously against two values of the array (from left to right) as to reduce it to a single value.
reduceRight( ) - applies a function simultaneously against two values of the array (from right to left) as to reduce it to a single value.
reverse( ) - reverses the order of the element of an array (that is,the first becomes the last,and the last becomes the first).
shift( ) - removes the first element from an array and returns that element.
slice( ) - extracts a section of an array and returns a new array.
some( ) - returns true if at least one element in this array satisfies the provided testing function.
toSource( ) - represents the source code of an object
sort( ) - sorts the element of an array.
splice( ) - adds or removes elements from an array.

toString( ) - returns a string representing the array and its elements.
unshift( ) - adds one or more elements to the front of an array and returns the new length of the array.

88.What are screen objects in Javascript ?
screen objects are used to read the information from the client's screen.The properties of a screen object 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 clients' screen,including the taskbar
Width - gives the total width of the client's screen,including the task bar

89.What is the difference between escape( ) and unescape( ) functions ?
escape( ) function is responsible for coding a string so as to transfer the information from one computer to other computer,across a network.
unescape( ) function is responsible for decoding a "coded string".

90.What is the difference between decodeURI( ) and encodeURI( ) ?
encodeURI( ) is used to convert URI into their Hex coding.
decodeURI( ) is used to convert the encoded URI back to normal.

91.Why it isn't advised to use innerHTML in Javascript ?
innerHTML content is refreshed every time and thus is slower.There is no scope for validation in innerHTML therefore it is easy to insert rouge code in the document thus making the web page unstable.

92.What boolean operators are available in Javascript ?
&& - AND operator
|| - OR operator
! - NOT operator

93.What is Namespacing ?
using global variables in Javascript is a bad practice.Namespacing is used to bundle up all your functionality using a unique name.In Javascript,a namespace is really just an object that you've attached all further methods,properties and objects.Namespace promotes modularity and code reuse in the application.

94.What happens if VAR is not defined ?
If you use var the variable is declared within the scope you are in (e.g. of the function). If you don't use var, the variable bubbles up through the layers of scope until it encounters a variable by the given name or the global object (window, if you are doing it in the browser), where it then attaches. It is then very similar to a global variable. However, it can still be deleted with delete (most likely by someone else's code who also failed to use var). If you use var in the global scope, the variable is truly global and cannot be deleted.

95.What is the difference between innerHTML and append( )
InnerHTML is not standard, and its a String. The DOM is not, and although innerHTML is faster and less verbose, its better to use the DOM methods like appendChild(), firstChild.nodeValue, etc to alter innerHTML content.

96.Does Javascript has "concept level scope" ?
Javascript doesn't have concept level scope.The variable declared inside the function has scope inside the function.
concept level scope is also called as block level scope.
JavaScript does not have block level scope, all the variables declared inside a function possess the same level of scope.

97.Is Javascript asynchronous ?
Web programming essentials like DOM event handlers and AJAX calls are asynchronous, and because these common tasks make up a great bulk of JavaScript programming, JavaScript itself is often labeled "asynchronous."

98.What is the problem with asynchronous ?
The main problem with asynchronous programming is how to call asynchronous functions that depend on each other. How do you ensure function A has finished before calling function B.

99.How to avoid the problem of asynchronous ?
callbacks can be used to avoid the problem of asynchronous.But,the heavy usage of callbacks lead to callback hell,the main problem of callback hell is difficult to read the code.
callback hell can be avoided by naming your functions and avoiding nested callbacks.
promises can be used to avoid the problem of asynchronous.promises give nice (looking) abstraction.
AsyncJS avoids the callback hell and the complexity of promises.

100.When callbacks are preferred ?
callbacks are preferred for low-level APIs because everything is explicit and visible.

101.What is the difference between Function declaration and Function Expression ?
(i) function funcDeclaration() {
  return 'A function declaration';
}

(ii)var funcExpression = function () {
  return 'A function expression';
}

Similar to the "var" statement, Function declarations are hoisted to the top of other code whereas Function expressions aren't hoisted, which allows them to retain a copy of the local variables from the scope where they were defined.

102.What is the difference between creating an object by object constructor and object literal ?

If an object is created using an object literal and if that object has some properties and methods,and if we are trying to create a new object from that parent object and if we want to add any properties and methods to the child object then the property and the methods of the parent object will get polluted or will get overridden ,in order to avoid this problem we will create an object using an object constructor.

103.What happens if you don't specify values for the function parameters when calling a function ?
If you don't specify values for the function parameters when calling a function,then the values for the function parameters are set to undefined.

104.What is the default return value of a function ?
The default return value of function is "undefined".

105.What is the result a function without return statement ?
A function without a "return statement" (or one that ends its execution without hitting one) will return "undefined".

106.What is document.write ?
The advantage of document.write is that ,document.write is the fastest way to add a script-generated text into the page(HTML document).
syntax:
document.write('<script src="' + src + '" type="text/javascript"><\/script>');
document.write works only while the page is loading,if you call document.write after the page has finished loading then "document.write" will overwrite the whole page.

107.What is an alternative to document.write ?
The alternative to "document.write" is "document.createElement".

108.What is the main difference between Dot notation and Bracket notation ?
The main difference between Dot notation and Bracket notation is that,Bracket notation allows for dynamic setting of properties.

109.How to keep the scope of "this" keyword lexical or How to set the scope of "this" keyword to lexical scope?
We can keep the scope of "this" keyword lexical or we can keep the scope of "this" keyword to lexical scope :
(i)By using bind( )
(ii)By using call( )
(iii)By using apply( )
(iv)By using "var self = this" or "var that = this" or "var _this = this"
(v)By using Arrow functions

110.What is bind( ) ?
we have an object with a method,and we are extracting the method from that object,then the method will become a plain method, and to overcome this problem we make use of bind( ).

syntax:
bind(object) - we pass an object into bind( )method.

111.What is the difference between call( ) and apply( ) ?
In case of call( ),the first argument is an object,which sets the value of “this” and the second and other arguments are passed individually.
In case of apply( ),the first argument is an object,which sets the value of “this” and the second and other arguments are passed in an array.

112.What is Lexical Scope ?
Whenever you see a function within another function, the inner function has access to the scope in the outer function, this is called Lexical Scope or Closure or Static Scope.

113.Can we have multiple event handlers on the same event ?
We can have multiple event handlers on the same event,and the event handlers are independent of each other and hence the browser doesn’t guarantee the order of execution of these event handlers.
For an instance,if there are two “onclick event handlers” on the same “link event”,then stopping bubbling in one of the event handlers has no effect on the other one.Also,the browser doesn’t guarantee the order in which they trigger.

114.What are the event handlers which cannot be registered as event bubbling ?
onfocus and onblur are the two event handlers which cannot be registered as event bubbling.

115 What should we use self-invoking functions ?
(i)To achieve encapsulation
(ii)To achieve block scoping
(iii)To prevent global namespace pollution
(iv)To help GC understand that you don't need any of referenced objects anymore when function is done.

116.What is the drawback of self-invoking anonymous functions ?
The drawback of self-invoking anonymous functions is that you cannot execute the same function twice (unless you put it inside a loop or another function). This makes the anonymous self-invoking functions best suited for one-off or initialization tasks.

117.Why should we use Anonymous functions ?
(i)To achieve code brevity ,which we can observe in callbacks and event handlers.
(ii)To achieve scope management,that is to create temporary/private scope.
(iii)To achieve code brevity,which we can observer in closures and recursions.
(iv)To prevent global namespace pollution

118.What is the result of 3 || “ “ || 2 || undefined || null ?
3

Here,|| prints the value if it is “Truthy value”.

119.What is the result of 3 && “ “ && 2 && undefined && null ?
empty string
Here,&& prints the value if it is “Falsy value”.

120.What are Falsy values ?
The Falsy values are :
(i) false
(ii) 0 (zero)
(iii) “ ” (empty string without space)
(iv) null
(v) undefined
(vi) NaN (a special Number value meaning Not-a-Number!)

Note(i):The falsy values false, 0 (zero), and "" (empty string) are all equivalent and can be compared against each other:
var c = (false == 0); // true
var d = (false == ""); // true
var e = (0 == ""); // true

Note(ii):The falsy values null and undefined are not equivalent to anything except themselves:
var f = (null == false); // false
var g = (null == null); // true
var h = (undefined == undefined); // true
var i = (undefined == null); // true

Note(iii):Finally, the falsy value NaN is not equivalent to anything — including NaN!
var j = (NaN == null); // false
var k = (NaN == NaN); // false
You should also be aware that typeof(NaN) returns "number". Fortunately, the core JavaScript function isNaN() can be used to evaluate whether a value is NaN or not.

121.What are Truthy values ?
The Truthy values are:
(i) “0” (zero in quotes)
(ii) “false" (false in quotes)
(iii) empty functions
(iv) empty arrays
(v) empty objects

122.What is the drawback of using Private methods in Javascript ?
One of the drawback of creating true private method in javascript is that they are very memory inefficient because a

new copy of the method would be created for each instance.

123.How to initiate the HTTP requests ?
HTTP requests are initiated when a script sets the location property of a window object or when a script calls the submit( ) method of a Form object.

124.What is an alternative method for the location property of a window object ?
submit( ) method is an alternative method for the location property of a window object.

125.What are the four different parts of HTTP Request ?
The four different parts of HTTP Request are :
(i)The HTTP request method or "verb"
(ii)The URL being requested
(iii)An optional set of request headers, which may include authentication information
(iv)An optional request body

126.What are the three different parts of HTTP Response sent by a server ?
The three different parts of HTTP Response sent by a server are :
(i)A numeric and textual status code that indicates the success or failure of the request
(ii)A set of response headers
(iii)The response body

127.What are the various forms of client-side storage ?
The various forms of client-side storage are web databases, filesystem API, Offline web applications and cookies.

128.Which client-side storage allows the caching of web pages and their associated resources ?
Offline web applications allows the caching of web pages and their associated resources

129.What is the Microsoft's own proprietary client-side storage?
Microsoft implements its own proprietary client-side storage mechanism, known as "userData," in IE5 and later. userData enables the storage of medium amounts of string data and can be used as an alternative to Web Storage in versions of IE before IE8.

130.How many ways the client-side Javascript code can be embedded within an HTML document ?
The client-side Javascript code can be embedded within an HTML document in four ways :
(i) Inline, between a pair of "script" tags
(ii) From an external file specified by the src attribute of a "script" tag
(iii) In an HTML event handler attribute, such as onclick or onmouseover
(iv) In a URL that uses the special javascript: protocol.

131.How nested documents in the HTML can be done ?
HTML documents may contain nested documents using an iframe element. An iframe creates a nested browsing context represented by a Window object of its own.

132.What does “1”+2+4 evaluate to ?
Since 1 is a string, everything is a string, so the result is 124.

133.What does 5 + 4 + “3” evaluate to ?
93.

134.How do you change the style/class on any element?
document.getElementById(“myText”).style.fontSize = “20”;
or
document.getElementById(“myText”).className = “anyclass”;

135.Which Attribute is used to include External JS code inside your HTML Document
src Attribute

136.Which attribute is used to specify that the script is executed when the page has finished parsing (only for external scripts) ?
defer Attribute

137.When there is an indefinite or an infinity value during an arithmetic value computation,what does Javascript display ?
“Infinity” gets displayed

138.What does the escape sequence ‘\f’ stands for ?
\f is the JavaScript escape sequence that stands for Form feed (\u000C).

139.How to convert a non-string “false” to string without invoking (using) the “new” operator ?
A non-string “false” can be converted into string without invoking (using) the “new” operator in two ways :
(i) false.toString()
(ii) String(false)

140.What is a Ternary operator ?
JavaScript supports one Ternary operator,that is,the Conditional operator ?: , which combines three expressions into a single expression.

141.What is the purpose of “toLocaleString( )” ?

The basic purpose of "toLocaleString( )" is to return a localized string representation of the object

142.What is the result of var count = [1,,3]; ?
The omitted value takes "undefined"

143.What happens if reverse() and join() methods are used simultaneously ?
The reverse() followed by a join() will reverse the respective array and will store the reversed array in the memory.

144.What method or operator is used to identify the Array ?
"typeof" method or property is used to identify the Array type.

145.What is the purpose of a "return" statement in a function?
The purpose of a "return" statement in a function is to stop executing the function and return the value.

146.What happens if a "return" statement does not have an associated expression?
If a "return" statement does not have an associated expression,then it returns the "undefined value".

147.What happens if you omit the LANGUAGE attribute in <script> tag ?
If you omit the LANGUAGE attribute, both Navigator and Internet Explorer default to the value "JavaScript". Nonetheless, because there are now multiple scripting languages available it is a good habit to always use the LANGUAGE attribute to specify exactly what language (or what version) your scripts are written in.

For an instance :
(i)<SCRIPT LANGUAGE="JavaScript"> // JavaScript code goes here
</SCRIPT>

(ii)<SCRIPT LANGUAGE="VBScript">
// VBScript code goes here (' is a comment character like // in JavaScript)
</SCRIPT>

148.Why do we need both client-side and server-side validations ?
Client side validation can be easily bypassed(bypassed or disabled) by disabling Javascript on a client browser.
If Javascript is disabled and if we don't have any server side validation, there could be different threats ranging from storing invalid data to security vulnerabilities.

149.What is window object ?
window object represents the browser's window, and all global variables are the properties of the window object and all global functions are the methods of the window object.

150.How to determine the size of a browser window ?
we can determine the size of a browser window using :
(i)window.innerHeight - return the inner height of the browser in pixels (not including toolbars and scrollbars).
(ii)window.innerWidth - returns the inner width of the browser in pixels (not including toolbars and scrollbars).
In case of IE,we can determine the size of a browser window using :
(i)document.documentElement.clientHeight
(ii)document.documentElement.clientWidth
or
(i)document.body.clientHeight
(ii)document.body.clientWidth

151.How to open a new window ,close the current window ,move the current window and resize the current window ?
window.open( ) - opens a new window
window.close( ) - closes the current window
window.moveTo( ) - moves the current window
window.resizeTo( ) - resizes the current window

152.How to calculate the users screen width and screen height ?
screen.width - returns the users' screen width
screen.height - returns the users' screen height
screen.availWidth - returns the users' screen width minus windows task bar
screen.availHeight - returns the users' screen height minus windows task bar
screen.colorDepth - returns the number of bits used to display one color.
screen.pixelDepth - returns the pixel depth of the screen.

153.What are the window.location properties and methods ?
(i)window.location.href property - returns the URL of the current page.
(ii)window.location.hostname property -returns the name of the current page.
(iii)window.location.pathname property - returns the pathname of the current page.
(iv)window.location.protocol property - returns the web protocol of the current page.
(v)window.location.assign( ) method - loads a new document.

154.What are window.history methods ?
(i)window.history.back( ) method - loads the previous URL in the history list.
window.history.back( ) method is same as clicking the Back button in the browser.

(ii)window.history.forward( ) method - loads the next URL in the history list.
window.history.forward( ) method is same as clicking the Forward button in the browser.

155.What RegExp Object Methods have you used ?
(i)RegExpObject.exec(string) - tests for a match in a string and returns the matched text if it finds a match,otherwise it returns null.
(ii)RegExpObject.test(string) - tests for a match in a string and returns true if it finds a match,otherwise it returns

false.
(iii)RegExpObject.toString( ) - returns the string value of the regular expression.

156.What RegExp brackets have you used ?
(i) [abc] - finds any character between the brackets
(ii) [^abc] - finds any character not between the brackets
(iii) [0-9] - finds any digits between the brackets
(iv) [^0-9] - finds any digits not between the brackets
(v) (x|y) - finds any of the alternatives specified

157.What is replace( ) method ?
replace( ) method searches a string for a specified value,or a regular expression, and returns a new string where the specified values are replaced.
If you are replacing a value (and not a regular expression) , only the first instance of the value will be replaced.To replace all the occurrences of a specified value,use the global("g") modifier.
string.replace(searchValue,newValue)

158.What is the result of ?
x = ~-y;
w = x = y = z;
q = a?b:c?d:e?f:g;

Result:
x = ~(-y); w = (x = (y = z));
q = a?b:(c?d:(e?f:g));

159.What is array map( ) ?
The map() method passes each element of the array on which it is invoked to the function you specify, and returns an array containing the values returned by that function.

160.Which common operation do the reduce and reduceRight methods follow ?
The reduce and reduceRight methods follow a common operation called inject and fold

161.What is the result of ?
if (!a[i]) continue;

Result :
(i)Skips the undefined elements
(ii)Skips the non existent elements
(iii)Skips the null elements

162.What is the result of ?

```
var a = [1,2,3,4,5];
a.slice(0,3);
```

Result:
[1,2,3]

163.Which of the following is the correct code for invoking a function without this keyword at all, and also to determine whether the strict mode is in effect?

a. var strict = (function { return this; });
b. mode strict = (function() { return !this; }());
c. var strict = (function() { return !this; }());
d. mode strict = (function { });

164.Which is an equivalent code to invoke a function m of class o that expects two arguments x and y?

a. o(x,y);
b. o.m(x) && o.m(y);
c. m(x,y);
d. o.m(x,y);

Result:
Answer : d
Explanation : The code above is an invocation expression: it includes a function expression o.m and two argument expressions, x and y.

165.Consider the following code snippet

```
o.m(x,y);
```

Which is an equivalent code for the above code snippet?

a. o.m(x) && o.m(y);
b. o["m"](x,y);
c. o(m)["x","y"];
d. o.m(x && y);

Result:
Answer : b
Explanation : Another way to write o.m(x,y) is o[“m”](x,y).

166.For the below mentioned code snippet:

```
var o = new Object();
```

The equivalent statement is:

a. var o = Object();
b. var o;
c. var o= new Object;
d. Object o=new Object();

Result:

Answer : c
Explanation : You can always omit a pair of empty parentheses in a constructor invocation.

167.What is the difference between the two lines given below ?
!!(obj1 && obj2);
(obj1 && obj2);

a. Both the lines result in a boolean value “True”
b. Both the lines result in a boolean value “False”
c. Both the lines checks just for the existence of the object alone
d. The first line results in a real boolean value whereas the second line merely checks for the existence of the objects

Result:
Answer : d
Explanation : None.

168.What is the property of JSON() method ?
a. it can be invoked manually as object.JSON()
b. it will be automatically invoked by the compiler
c. it is invoked automatically by the JSON.stringify() method
d. it cannot be invoked in any form

Result:
Answer : c
Explanation : The JSON format is intended for serialization of data structures and can handle JavaScript primitive values, arrays, and plain objects. It does not know about classes, and when serializing an object, it ignores the object’s prototype and constructor. If you call JSON.stringify() on a Range or Complex object, for example, it returns a string like {“from”:1, “to”:3} or {“r”:1, “i”:-1}.

169.How to implement AJAX using Javascript ?

```
<!DOCTYPE html>
<html>
<body>

<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
```

```
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

170.How to implement Ajax using jQuery ?

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $.ajax({url: "demo_test.txt", success: function(result){
         $("#div1").html(result);
      }});
   });
});
</script>
</head>
<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>

</body>
</html>
```

Setting	Description	Default Value
cache	Set to true to cache the response from the server, or false to always make the request. A value of false also makes jQuery append a random timestamp parameter to the request, as it does with $.getScript(), to ensure the browser doesn't cache the response.	true (false when used with the 'script' and 'jsonp' dataType setting)
complete	Specifies a callback function to run when the request completes (whether successful or not).	None
data	Any data to be sent to the server in the request. This works in the same way as $.get()'s data argument.	None
dataType	The type of data to expect in the response. This works in the same way as $.get()'s fourth argument. You can also specify a value of "jsonp" to make a JSONP request.	Intelligent guess
error	Specifies a callback function to run if the request fails.	None
headers	Additional HTTP headers to send with the request, as a map of key/value pairs.	{}

password	A password to use if the server requires HTTP authentication.	None
success	Specifies a callback function to run if the request succeeds. This is the same as the third argument passed to `$.get()`.	None
timeout	How long, in milliseconds, to wait for the Ajax request to complete. A value of zero means jQuery will wait indefinitely.	`0`
type	The type of request to make: `"GET"` or `"POST"`.	`"GET"`
username	A username to use if the server requires HTTP authentication.	None

171.How many we can define a function ?
Function Declaration
Function Expression
Function Expression is of three types :
Named Function Expressions
Anonymous Function Expressions
Self Invoking Function Expressions

172.what is delete operator ?
delete operator replaces the value of a key to undefined but delete operator doesn't delete the key
obj =[1,2,3];
delete obj.1
obj.length;
console.log(obj.1) // undefined
console.log(obj.length) // still the length is 3,as the key didn't get deleted

173.What is the difference between Function parameters and Function arguments ?
Function parameters are the names listed in the Function definition.
Function arguments are the real values passed to the function.
If a function is called with missing arguments,the missing values are set to "undefined".

174.what is the purpose of arguments object in a function ?
Arguments object is a local variable available within all functions.It contains all the function parameters that are passed to the function and can be indexed like an array.The length property of the arguments object returns the number of arguments passed to the function.

175.Is it possible to pass variable number of arguments to a Javascript function ?
we can pass as many arguments as we want to a Javascript function.All the parameters will be stored in the argument object.

176.Can we use sort( ) method on arguments object ?
arguments object is not similar to Arrays,so we cannot use sort( )method on arguments object.To use sort( )method,we need to convert the arguments object into an Array and the we can use sort( )method on this Array.

```
var argsArray = Array.prototype.slice.call(arguments);
argsArray.sort( );
```

or

```
var argsArray = [ ].slice.call(arguments);
argsArray.sort( );
```

177.what is recursive function ?
A recursive function is a function that calls itself.
When writing a recursive function there must be a definite break condition,otherwise we risk creating infinite loops.

178.what is DOM?
DOM is an application programming interface,and is used to add content to an HTML document,to delete content from an HTML document,and to change content on an HTML document.
The document is just a web page.
The objects are the HTML elements like <head>, <body>,
The model is how the HTML elements are laid out in a tree structure.

179.what is a prototype property ?
The prototype property is used to add properties and methods to an object.

```
object.prototype.name = value;
```

180.why do we need a prototype object ?
In any Object-Oriented programming language,we don’t want to create the copies of functions instead we want to share the same function code across all objects and this is possible using prototype object.

181.what are the advantages of prototype object?
(i)No matter how many objects you create,functions are loaded only once into memory.

(ii)Allows you to override functions if required.

182.What is Namespace ?
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.

Using global variables in Javascript is evil and a bad practice. That being said, namespacing is used to bundle up all your functionality using a unique name. In JavaScript, a namespace is really just an object that you've attached all further methods, properties and objects. It promotes modularity and code reuse in the application.

183.What is Nested Namespace ?
Nested Javascript Namespace is nothing but a Namespace inside a Namespace,which provides better modularization for complex javascript applications.

```
var MYAPPLICATION = {
        MODEL : {
                PRODUCTS : {

                                                }
                        }
}
```

The above code is similar to :
MYAPPLICATION.createNS("MYAPPLICATION.MODEL.PRODUCTS");

How to check if a Namespace is already exists ?
Before creating Namespace ,it is good practice to check if it already exists,and the syntax for that is :
var MYAPPLICATION = MYAPPLICATION | | { }

184.What is Namespace Aliasing ?
when your Namespace hierarchy becomes deeper,you will notice that it becomes cumbersome since you always have to type the complete reference to a function or a variable.This can easily be solved by "aliasing a namespace".

```
var MYAPPLICATION = {
        MODEL : {
                PRODUCTS : {

                                                }
                        }
}
```

The above code can be aliased to :
var p = MYAPPLICATION.MODEL.PRODUCTS;

185.what is Inheritance ?
Javascript has prototype based inheritance,that is,an object inherits from another object.

```
var a = function () {}
a.prototype.sayHello = function () { alert('hello') }

var b = function () {}
b.prototype = new a();
//b.prototype.sayHello();

var c = new b();
c.sayHello(); // Alerts "hello"
```

186.What is instanceOf operator ?
instanceOf operator is used to check if the object is created by given constructor.

```
function Animal( ) { }
var dog = new Animal
alert( dog  instanceOf  Animal )     // true

//( here,the logic that happens behind is : dog._proto_ = = Animal.prototype )

function Animal( ) { }
var dog = new Animal( )
alert(dog  instanceOf  Animal)

var dog = Object.create(Animal)

Object.create(Animal)
new Animal()


Object.getPrototypeOf(dog) = = Animal
dog  instanceof  Animal
```

187.What is typeof property ?
typeof operator takes a value ad returns its type.
(i)typeof undefined - returns “undefined”
(ii)typeof 0 - returns “number”
(iii)typeof true - returns “boolean”
(iv)typeof “foo” - returns “string”
(v)typeof { } - returns “object”
(vi)typeof null - returns “object”
(vii)typeof function( ) { } - returns “function”

(viii)typeof NaN - returns “number”

188.what is hasOwnProperty( ) ?
hasOwnProperty() is used to determine if a property is defined on the actual object or its prototype.

```
function Animal (name) {
        this.name = name
}

Animal.prototype = {
        eats : true
}

var dog = new Animal (‘tommy’)

alert( dog.hasOwnProperty(‘eats’) )     // false ,  in prototype

alert( dog.hasOwnProperty(‘name’) )   // true,   in object
```

How to associate the child object to parent object ?
var childObject = new parentObject( );

what is polymorphism ?

A for..in loop outputs all properties from the object and its prototype:

```
function Rabbit(name) {
          this.name = name
        }

        Rabbit.prototype = { eats: true }

        var rabbit = new Rabbit('John')

        for(var p in rabbit) {
          alert (p + " = " + rabbit[p])
        }
```

output : name = john
                eats = true

To get a list of properties, which are not in prototype, filter them through hasOwnProperty:

```
function Rabbit(name) {
          this.name = name
        }

        Rabbit.prototype = { eats: true }

        var rabbit = new Rabbit('John')

        for(var p in rabbit) {
        if (!rabbit.hasOwnProperty(p)) continue

          alert (p + " = " + rabbit[p])
        }
```

output : name = john

```
var animal = {
          eats: true
        }

        rabbit = Object.create(animal)

        alert( Object.getPrototypeOf(rabbit) === animal )
```

```
var animal = {
          eats: true
        }

        rabbit = Object.create(animal)

        alert(rabbit instanceof animal)
```

```
        function Rabbit() { }
        var rabbit = new Rabbit

        alert(rabbit instanceof Rabbit)
```

```
function Rabbit() { }
        var rabbit = new Rabbit

        alert(Object.getPrototypeOf(rabbit) === Rabbit )
```

189.what is the result of the following ?
Number("Hello") - returns NaN
Number("") - returns 0
Number("000010") - returns 10
Number(true) - returns 1
Number(NaN) - returns NaN

190.How to check if a variable is an Array ?
we use "instanceofArray" to check if a variable is an Array.

```
if (i instanceofArray) {
        alert('i is an Array');
}else {
        alert(i isn't an Array);
}
```

191.How to set the cursor to wait in Javascript ?
window.document.body.style.cursor = "wait" ;

192.How to submit a form using Javascript ?
document.forms[0].submit();
0 refers to the index of the form.if you have more than one form in a page,the the first form has the index of 0,the second form has the index of 1,and so on.

193.For var a = (2,3,5) ; what is the value of a ?
The value of a is 5.
The comma operator evaluates each of its operands (from left to right) and returns the value of the last operand.

194.For var a = (1, 5 - 1) * 2 what is the value of a?

The value of a is 8.

195.What is the value of !'bang' ?
false. ! is NOT. If you put ! in front of truthy values, it will return false.

196.For 2 in [1,2] ; what is the result ?
false. Because "in" returns whether a particular property/index available in the Object. In this case object has index 0 and 1 but don't have 2. Hence you get false.

197.what is 2&&3 ?
3

198.what is 3&&2 ?
2

199.what is 2 || 3 ?
2

200.what is 3 || 2 ?
3

201.what is 3 instanceof Number ?
false

202.What is typeof arguments ?
Object. arguments are array like but not array. it has length, can access by index but can't push pop, etc.

203.What is typeof [] ?
Object. Actually Array is derived from Object. If you want to check array use Array.isArray(arr)

204.What is 2+true ?
3. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 2+1

205.What is the result of ?
```
var text = 'outside';
function logIt(){
   console.log(text);
```

```
    var text = 'inside';
};
logIt();
```

The result is :
//undefined
//undefined

206.How to detect mobiles including ipad using navigator.useragent in javascript?

```
 if(navigator.userAgent.match(/Android/i) || navigator.userAgent.match(/webOS/i) ||  navigator.userAgent.match(/
BlackBerry/i) || navigator.userAgent.match(/iPhone/i)){
        console.log('Calling from Mobile');
    }else{
    console.log('Calling from Web');
}
```

207.How to open URL in new tab in Javascript ?

```
window.open('http://www.web-technology-experts-notes.in/','_blank');
```

208.How to convert JSON Object to String?

```
var myobject=['Web','Technology','Experts','Notes']
JSON.stringify(myobject);
```

209.How to convert JSON String to Object?

```
var jsonData = '{"name":"web technology","year":2015}';
var myobject = JSON.parse(jsonData);
console.log(myobject);
```

210.How to modify the URL of page without reloading the page?
use pushState javascript function.

```
window.history.pushState('page2', 'This is page Title', '/newpage.php');
```

211.Why do we need to check if DOM is ready OR NOT?
Some times, we need to do some modification in html tags like p,div and image etc.
We can change the DOM only after if DOM is loaded.

212.How can you check the DOM is fully loaded or ready ?

```
 jQuery(document).ready(function(){
     //console.log('DOM IS FULLY loaded');

  });
```

```
if ( jQuery.isReady ) {
   //console.log('DOM IS FULLY loaded');
}
```

213.what is this keyword

Global Scope

If there's no current object, 'this' refers to the global object. In a web browser, that's 'window' — the top-level object which represents the document, location, history and a few other useful properties and methods.

```
window.WhoAmI = "I'm the window object";
alert(window.WhoAmI);
alert(this.WhoAmI); // I'm the window object
alert(window === this); // true
```

Calling a Function

'this' remains the global object if you're calling a function:

```
window.WhoAmI = "I'm the window object";

function TestThis() {
    alert(this.WhoAmI); // I'm the window object
    alert(window === this); // true
}

TestThis();
```

Calling Object Methods

When calling an object constructor or any of its methods, 'this' refers to the instance of the object — much like any class-based language:

```
window.WhoAmI = "I'm the window object";

function Test() {

    this.WhoAmI = "I'm the Test object";

    this.Check1 = function() {
        alert(this.WhoAmI); // I'm the Test object
    };

}

Test.prototype.Check2 = function() {
    alert(this.WhoAmI); // I'm the Test object
};

var t = new Test();
t.Check1();
t.Check2();
```

Using Call or Apply

In essence, call and apply run JavaScript functions as if they were methods of another object. A simple example demonstrates it further:

```
function SetType(type) {
    this.WhoAmI = "I'm the "+type+" object";
}

var newObject = {};
SetType.call(newObject, "newObject");
alert(newObject.WhoAmI); // I'm the newObject object

var new2 = {};
SetType.apply(new2, ["new2"]);
alert(new2.WhoAmI); // I'm the new2 object
```

Local Scope

Inside of a function, the value of `this` depends on how the function is called. There are three main variations:

`this` Used in a Simple Function Call

The first variation is a standalone function invocation where we call a function directly.

```
function simpleCall(){
  console.log(this);
}

simpleCall();
// output: the Window object
```

In this case, the value of `this` is not set by the call. Since the code is not running in strict mode, the value of `this` must always be an object so it defaults to the global object.

In strict mode, the value of `this` remains at whatever it's set to when entering the execution context. If it's not defined, it remains undefined, as we can see in the following example:

```
function simpleCall(){
  "use strict";
  console.log(this);
}

simpleCall();
// output: undefined
```

`this` Used in an Object's Method

We can store a function in a property of an object, which turns it into a method that we can invoke via that object. When a function is called as a method of an object, its `this` value is set to the object the method is called on.

```
var message = {
  content: "I'm a JavaScript Ninja!",
  showContent: function() {
    console.log(this.content);
  }
};

message.showContent();   // output: I'm a JavaScript Ninja!
```

Here, `showContent()` is a method of the `message` object, and thus `this.content` is equal to `message.content`.

`this` Used in Constructor Functions

We can invoke a function via the `new` operator. In this case the function becomes a constructor—a factory for objects. Unlike the simple function calls and method calls discussed above, a constructor call passes a brand new object as the value of `this`, and implicitly returns the new object as its result.

When a function is used as a constructor (with the `new` keyword), its `this` value is bound to the newly constructed object. If we miss the `new` keyword, then it will be a regular function and `this` will point to the `window` object.

```
function Message(content){
  this.content = content;
  this.showContent = function(){
    console.log(this.content);
  };
}

var message = new Message("I'm JavaScript Ninja!");

message.showContent();
// output: I'm JavaScript Ninja!
```

In the above example, we have a constructor function named `Message()`. By using the `new` operator we create a brand new object named `message`. We also pass the constructor function a string, which it sets as the `content` property of our new object. In last line of code we see that this string is successfully output, because `this` is pointing to the newly created object, and not to the constructor function itself.

How `this` Can Be Successfully Manipulated

In this section, we'll examine some built-in mechanisms for controlling the behavior of `this`.

In JavaScript, all functions are objects, and therefore they can have methods. Two of these methods, which all functions have, are apply() and call(). We can use these methods to change the context to whatever we need and thus, explicitly set the value of `this`.

The `apply()` method takes two arguments: an object to set `this` to, and an (optional) array of arguments to pass to the function.

The `call()` method works exactly the same as `apply()`, but we pass the arguments individually rather than in an array.

Let's see it in action:

```
function warrior(speed, strength){
  console.log(
    "Warrior: " + this.kind +
    ", weapon: " + this.weapon +
    ", speed: " + speed +
    ", strength: " + strength
  );
}

var warrior1 = {
  kind: "ninja",
  weapon: "shuriken"
};

var warrior2 = {
  kind: "samurai",
  weapon: "katana"
};

warrior.call(warrior1, 9, 5);
// output: Warrior: ninja, weapon: shuriken, speed: 9, strength: 5
warrior.apply(warrior2, [6, 10]);
// output: Warrior: samurai, weapon: katana, speed: 6, strength: 10
```

Here, we have a factory function `warrior()`, which is used to create different types of warriors by using different warrior objects. So, in that factory function, `this` will point to the different objects we pass in using `call()` and/or `apply()`.

In the first function call, we use the `call()` method to set `this` to the `warrior1` object, and pass the other arguments we need, separated by commas. In the second function call, we do almost the same, but this time we pass in the `warrior2` object and the necessary arguments are put in an array.

Besides `apply()` and `call()` ECMAScript 5 added the bind() method, which also allows us to set which specific object will be bound to `this` when a function or method is invoked. Let's consider the following example:

```
function warrior(kind){
  console.log(
    "Warrior: " + kind +
    ". Favorite weapon: " + this.weapon +
    ". Main mission: " + this.mission
  );
}

var attributes = {
  weapon: "shuriken",
  mission: "espionage"
};

var ninja = warrior.bind(attributes, "ninja");

ninja();
// output: Warrior: ninja. Favorite weapon: shuriken. Main mission: espionage
```

In this example, the `bind()` method is used in similar way, but unlike the `call()` and `apply()` methods, `warrior.bind()` creates a new function (with the same body and scope as `warrior()`) rather than modifying the original `warrior()` function. The new function behaves just like the old one, but with its receiver bound to the `attributes` object, while the old one remains unchanged.

1. Using `this` in Extracted Methods

One of the most common mistakes that people make is when trying to assign an object's method to a variable and expecting that `this` will still point to the original object. As we can see from the following example, that simply doesn't work.

- Building Faster Websites with Grav, a Modern Flat-file CMS
- 7 MORE Photoshop Master Tips to Speed up Your Workflow

```
var car = {
  brand: "Nissan",
  getBrand: function(){
    console.log(this.brand);
  }
};

var getCarBrand = car.getBrand;

getCarBrand();   // output: undefined
```

Even though `getCarBrand` appears to be a reference to `car.getBrand()`, in fact, it's just another reference to `getBrand()` itself. We already know that the call-site is what matters in determining the context, and here, the call-site is `getCarBrand()`, which is a plain and simple function call.

To prove that `getCarBrand` points to a baseless function (one which isn't bound to any specific object), just add `alert(getCarBrand);` to the bottom of the code and you'll see the following output:

```
function(){
  console.log(this.brand);
}
```

`getCarBrand` holds just a plain function, which is no longer a method of the `car` object. So, in this case, `this.brand` actually translates to `window.brand`, which is, of course, `undefined`.

If we extract a method from an object, it becomes a plain function again. Its connection to the object is severed, and it no longer works as intended. In other words, an extracted function is not bound to the object it was taken from.

So how can we remedy this? Well, if we want to keep the reference to the original object, we need to explicitly bind the `getBrand()` function to the `car` object when we assign it to the `getCarBrand` variable. We can do this by using the bind() method.

```
var getCarBrand = car.getBrand.bind(car);
getCarBrand();   // output: Nissan
```

Now, we get the proper output, because we successfully redefine the context to what we want it to be.

2 `this` Used in Callbacks

The next issue occurs when we pass a method (that uses `this` as a parameter) to be used as a callback function. For example:

```
<button id="btn" type="button">Get the car's brand</button>

var car = {
  brand: "Nissan",
  getBrand: function(){
    console.log(this.brand);
  }
};

var el = document.getElementById("btn");
el.addEventListener("click", car.getBrand);
```

Even though we use `car.getBrand`, we actually only get the function `getBrand()` which is attached to the `button` object.

Passing a parameter to a function is an implicit assignment, so what happens here is almost the same as in the previous example. The difference is that now `car.getBrand` is not explicitly assigned, but implicitly. And the result is pretty much the same—what we get is a plain function, bound to the `button` object.

In other words, when we execute a method on an object, which is different from the object upon which the method was originally defined, the `this` keyword no longer refers to the original object, rather to the object that invokes the method.

With reference to our example: we are executing `car.getBrand` on `el` (the button element), not the `car` object, upon which it was originally defined. Consequently, `this` no longer refers to `car`, rather to `el`.

If we want to keep the reference to the original object intact, again, we need to explicitly bind the `getBrand()` function to the `car` object by using the `bind()` method.

```
el.addEventListener("click", car.getBrand.bind(car));
```

Now, everything works as expected.

3 `this` Used Inside Closures

Another instance when `this`'s context can be mistaken is when we use `this` inside of a closure. Consider the following example:

```
var car = {
  brand: "Nissan",
  getBrand: function(){
    var closure = function(){
      console.log(this.brand);
    };
    return closure();
  }
};

car.getBrand();   // output: undefined
```

Here, the output we get is `undefined`, because closure functions (inner functions) don't have access to the `this` variable of outer functions. The net result is that `this.brand` is equal to `window.brand`, because `this` in inner functions is bound to the global object.

To fix this issue, we need to keep `this` bound to the `getBrand()` function.

```
var car = {
  brand: "Nissan",
  getBrand: function(){
    var closure = function(){
      console.log(this.brand);
    }.bind(this);
    return closure();
  }
};

car.getBrand();   // output: Nissan
```

This binding is equivalent to `car.getBrand.bind(car)`.

Another popular method to fix closures, is to assign the `this` value to another variable, thus preventing the unwanted change.

```
var car = {
  brand: "Nissan",
  getBrand: function(){
    var self = this;
    var closure = function(){
      console.log(self.brand);
    };
    return closure();
  }
};

car.getBrand();   // output: Nissan
```

Here, the value of `this` can be assigned to `_this`, `that`, `self`, `me`, `my`, `context`, an object's pseudo name, or whatever else works for you. The main point is to keep a reference to the original object.

ECMAScript 6 to the Rescue

In the previous example we saw a primer on what is known as "lexical `this`"—when we set the `this` value to another variable. In ECMAScript 6 we can use the similar, but more elegant, technique, applicable via the new arrow functions.

Arrow-functions are created not by the `function` keyword, but by the so-called "fat arrow" operator (`=>`). Unlike regular functions, arrow functions take the `this` value from their immediate enclosing scope. The lexical binding of an arrow function can't be overridden, even with the `new` operator.

Let's now see how arrow function can be used to substitute the `var self = this;` statement.

```
var car = {
  brand: "Nissan",
  getBrand: function(){
    // the arrow function keeps the scope of "this" lexical
    var closure = () => {
      console.log(this.brand);
    };
    return closure();
  }
};

car.getBrand();   // output: Nissan
```

214.Javascript String functions

1. charAt(x) Returns the character at the "x" position within the string.

```
//charAt(x)
var myString = 'jQuery FTW!!!';
console.log(myString.charAt(7));
//output: F
```

2. charCodeAt(x) Returns the Unicode value of the character at position "x" within the string.

```
//charAt(position)
var message="jquery4u"
//alerts "q"
alert(message.charAt(1))
```

3. concat(v1, v2,...) Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.

```
//concat(v1, v2,..)
var message="Sam"
var final=message.concat(" is a"," hopeless romantic.")
//alerts "Sam is a hopeless romantic."
alert(final)
```

4. fromCharCode(c1, c2,...) Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance. For example: String.fromCharCode().

```
//fromCharCode(c1, c2,...)
console.log(String.fromCharCode(97,98,99,120,121,122))
//output: abcxyz
console.log(String.fromCharCode(72,69,76,76,79))
//output: HELLO
//(PS - I have no idea why you would use this? any ideas?)
```

Also see: Full List of JavaScript Character Codes

5. indexOf(substr, [start]) Searches and (if found) returns the index number of the searched character or substring within the string. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is 0.

```
//indexOf(char/substring)
var sentence="Hi, my name is Sam!"
if (sentence.indexOf("Sam")!=-1)
alert("Sam is in there!")
```

6. lastIndexOf(substr, [start]) Searches and (if found) returns the index number of the searched character or substring within the string. Searches the string from end to beginning. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is string.length-1.

```
//lastIndexOf(substr, [start])
var myString = 'javascript rox';
console.log(myString.lastIndexOf('r'));
//output: 11
```

7. match(regexp) Executes a search for a match within a string based on a regular expression. It returns an array of information or null if no match is found.

```
//match(regexp) //select integers only
var intRegex = /[0-9 -()+]+$/;

var myNumber = '999';
var myInt = myNumber.match(intRegex);
console.log(isInt);
//output: 999

var myString = '999 JS Coders';
var myInt = myString.match(intRegex);
console.log(isInt);
//output: null
```

Also see: jQuery RegEx Examples to use with .match()

8. replace(regexp/substr, replacetext) Searches and replaces the regular expression (or sub string) portion (match) with the replaced text instead.

```
//replace(substr, replacetext)
var myString = '999 JavaScript Coders';
console.log(myString.replace(/JavaScript/i, "jQuery"));
//output: 999 jQuery Coders

//replace(regexp, replacetext)
var myString = '999 JavaScript Coders';
console.log(myString.replace(new RegExp( "999", "gi" ), "The"));
//output: The JavaScript Coders
```

9. search(regexp) Tests for a match in a string. It returns the index of the match, or -1 if not found.

```
//search(regexp)
var intRegex = /[0-9 -()+]+$/;

var myNumber = '999';
var isInt = myNumber.search(intRegex);
console.log(isInt);
//output: 0

var myString = '999 JS Coders';
var isInt = myString.search(intRegex);
console.log(isInt);
//output: -1
```

10. slice(start, [end]) Returns a substring of the string based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.

```
//slice(start, end)
var text="excellent"
text.slice(0,4) //returns "exce"
text.slice(2,4) //returns "ce"
```

11. split(delimiter, [limit]) Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return.

```
//split(delimiter)
var message="Welcome to jQuery4u"
//word[0] contains "We"
//word[1] contains "lcome to jQuery4u"
var word=message.split("l")
```

12. substr(start, [length]) Returns the characters in a string beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, to)
var text="excellent"
text.substring(0,4) //returns "exce"
text.substring(2,4) //returns "ce"
```

13. substring(from, [to]) Returns the characters in a string between "from" and "to" indexes, NOT including "to" inself. "To" is optional, and if omitted, up to the end of the string is assumed.

```
//substring(from, [to])
var myString = 'javascript rox';
myString = myString.substring(0,10);
console.log(myString)
//output: javascript
```

14. toLowerCase() Returns the string with all of its characters converted to lowercase.

```
//toLowerCase()
var myString = 'JAVASCRIPT ROX';
myString = myString.toLowerCase();
console.log(myString)
//output: javascript rox
```

15. toUpperCase() Returns the string with all of its characters converted to uppercase.

```
//toUpperCase()
var myString = 'javascript rox';
myString = myString.toUpperCase();
console.log(myString)
//output: JAVASCRIPT ROX
```

216.what is delete operator ?
The delete operator is used to delete properties from an object.
delete operators don't affect local variables.

var trees = ["redwood","bay","cedar","oak","maple"];
delete trees[3];
console.log(trees);
Answer : ["redwood","bay","cedar",undefined,"maple"];

var trees = ["redwood","bay","cedar","oak","maple"];
delete trees[3];
console.log(trees);
console.log(trees===undefined);
Answer : true

```
var trees = ["redwood","bay","cedar","oak","maple"];
delete trees[3];
console.log(trees);
console.log(trees.length);
```
Answer : 5

217.what is Boolean + Number evaluate to ?
Boolean + Number = Addition

```
var bar = true;
console.log(bar + 0);          // Answer : 1          // Explanation : Boolean + Number = Addition
```

218.what is Boolean + String evaluate to ?
Boolean + String = Concatenation

```
var bar = true;
console.log(bar + "xyz");      // Answer : “truexyz”  // Explanation : Boolean + String = Concatenation
```

219.what is Boolean + Boolean evaluate to ?
Boolean + Boolean = Addition

```
var bar = true;
console.log(bar + true);       // Answer : 2          // Explanation : Boolean + Boolean = Addition
```

220.what is the value of False(Boolean) ?
The value of False(Boolean) = 0

221.what is the value of True(Boolean) ?
The value of True(Boolean) = 1

222.what is the difference between Function Expression and Function Declaration ?
Function Expression

```
var foo = function(){
   // Some code
};
```

Function Declaration

```
function bar(){
   // Some code
};
```

The main difference is the Function Expression ,that is, function foo is defined at run-time whereas the Function Declaration ,that is,function bar is defined at parse time. To understand this in better way, let's take a look at the code below:

Run-Time function declaration

```
<script>
foo(); // Calling foo function here will give an Error
 var foo = function(){
       console.log("Hi I am inside Foo");
 };
 </script>
```

```
<script>
Parse-Time function declaration
bar(); // Calling bar function will not give an Error
 function bar(){
   console.log("Hi I am inside Foo");
 };
 </script>
```

Another advantage of Run-Time function declaration is that you can declare functions based on certain conditions.

223.Is Javascript asynchronous ?
Web programming essentials like DOM event handlers and AJAX calls are asynchronous, and because these common tasks make up a great bulk of JavaScript programming, JavaScript itself is often labeled "asynchronous."

224.What is the problem with asynchronous ?
The main problem with asynchronous programming is how to call asynchronous functions that depend on each other. How do you ensure function A has finished before calling function B.

225.How to avoid the problem of asynchronous ?
callbacks can be used to avoid the problem of asynchronous.But,the heavy usage of callbacks lead to callback hell,the main problem of callback hell is difficult to read the code.
callback hell can be avoided by naming your functions and avoiding nested callbacks.
promises can be used to avoid the problem of asynchronous.promises give nice (looking) abstraction.
AsyncJS avoids the callback hell and the complexity of promises.

226.When callbacks are preferred ?
callbacks are preferred for low-level APIs because everything is explicit and visible.

227.what is object.assign() ?
object.assign( ) is used to copy the values of all enumerable own properties from one or more source objects to a target object,and it (object.assign( )) will return the target object.

```
var obj = { a: 1 };
var copy = Object.assign({}, obj);
console.log(copy); // { a: 1 }
```

228.what is Javascript used for ?
Javascript is used to change HTML content,DOM elements,HTML attributes,HTML styles,and to validate input data forms.

```
var z = 1, y = z = typeof y;
console.log(y);
```

Answer : undefined
Explanation : Here, the associativity of the assignment operator is Right to Left, so typeof y will evaluate first , which is undefined. It will be assigned to z, and then y would be assigned the value of z and then z would be assigned the value 1.

```
var foo = function bar(){ return 12; };
 typeof bar();
```

Answer : Reference Error

```
 var bar = function(){ return 12; };
 typeof bar();
```

Answer : number

```
function bar(){ return 12; };
 typeof bar();
```

Answer : number

```
 var foo = function bar(){
 };
 console.log(typeof bar());
```

Answer : number

```
 var foo = function bar(){
          console.log(typeof bar());
 };
```

Answer : undefined

How to reverse a string ?

```
function reverseString(str) {
   return str.split("").reverse().join("");
}
reverseString("hello");
```

Result is : "olleh"

How to find the largest number in an Array ?

```
function getmax() {
    var numbers = [11, 22, 33, 99, 44, 55, 66, 77, 88],
       max = Math.max.apply(null,numbers);
  console.log(max);
}
getmax();
```

How to find the smallest number in an Array ?

```
function getmin() {
    var numbers = [11, 22, 33, 99, 44, 55, 66, 77, 88],
       min = Math.min.apply(null,numbers);
  console.log(min);
}
getmin();
```

How to sort an Array alphabetically ?

```
function ascendingOrder(){
          var fruits = ["Banana", "Orange", "Apple", "Mango"]
          ascending = fruits.sort();
          console.log(ascending);
}
ascendingOrder();
```

How to sort an Array in descending order ?

```
function descendingOrder(){
        var fruits = ["Banana", "Orange", "Apple", "Mango"]
        descending = fruits.sort().reverse();
        console.log(descending);
}
descendingOrder();
```

How to sort an Array in ascending order ?

```
function ascendingOrder() {
        var numbers = [40, 100, 1, 5, 25, 10];
        ascending = numbers.sort(function(a,b) {
        return a-b;
});
console.log(ascending);
}
ascendingOrder();
```

How to sort an Array in descending order ?

```
function descendingOrder() {
        var numbers = [40, 100, 1, 5, 25, 10];
        descending = numbers.sort(function(a,b) {
        return b-a;
});
console.log(descending);
}
descendingOrder();
```

How to sort an Array in random order ?

```
function randomOrder() {
        var numbers = [40, 100, 1, 5, 25, 10];
        random = numbers.sort(function(a,b) {
        return 0.5 - Math.random()
});
console.log(random);
}
randomOrder();
```

How to print the number of even numbers in an Array ?

```
var a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var b =[ ];
for (var i = 0; i < a.length; i++) {
   if(i % 2 === 0) {
      b.push(a[i]);
   }
}
```

How to print the even numbers of an Array ?

```
var a = [4,5,7,8,14,45,76];

function even(a){
  return a.filter(function(val){return val%2===0})
}

console.log((even(a)));
```

How to print the odd numbers of an Array ?

```
var a = [4,5,7,8,14,45,76];

function odd(a){
  return a.filter(function(val){return val%2!==0})
}

console.log((odd(a)));
```

How to print the even numbers in an Array ?

How to print the number of even numbers in an Array ?

How to find odd numbers in an Array ?

How to print the number of odd numbers in an Array ?

How to find the duplicates in an Array ?

How to remove the duplicates in an Array ?

palindrom
fatorial
reverse string
fibonacci series

how to remove 3rd element from an Array ?

how to add new element at 4th place in an Array ?

How to find the largest number in an Array ?

How to find the smallest number in an Array ?

How to find the duplicates in an Array ?

How to remove the duplicates in an Array ?

How to achieve the ascending order of an Array ?

How to achieve the descending order of an Array ?

How to find the longest string in an Array ?

Explain prototype property of function ?

how will you create a namespace in jAVASCRIPT ?

how will you create a module in javascript ?

Explain JSON and JSONP ?

what is the cross domain reference issue ?

what are recursive functions ?

what are callback functions ?

how will you implement inheritance in Javascript ?

can we assign primitives to the prototypes ?

can we assign objects and Arrays to the prototypes ?

what is prototype chaining ?

where do we use event bubbling and event capturing ?