JavaScript Interview Questions & Answers

> Click ⭐ if you like the project. Pull Requests are highly appreciated. Follow me @SudheerJonna for technical updates.

Go to Coding Exercise for coding specific questions

Download PDF/Epub formats

You can download the PDF and Epub version of this repository from the latest run on the actions tab.

1. Take this JavaScript Projects course to go from a JavaScript beginner to confidently building your own projects
2. Take this coding interview bootcamp if you're serious about getting hired and don't have a CS degree

3. Take this Advanced JavaScript Course to learn advanced JS concepts and become a top JS developer

JavaScript Interview Questions | Top JavaScript Interview Questions and Answers.

Table of Contents

No.	Questions
1	What are the possible ways to create objects in JavaScript
2	What is prototype chain
3	What is the difference between Call, Apply and Bind
4	What is JSON and its common operations
5	What is the purpose of the array slice method
6	What is the purpose of the array splice method
7	What is the difference between slice and splice
8	How do you compare Object and Map
9	What is the difference between == and === operators
10	What are lambda or arrow functions
11	What is a first class function
12	What is a first order function
13	What is a higher order function
14	What is a unary function

No.	Questions
15	What is the currying function
16	What is a pure function
17	What is the purpose of the let keyword
18	What is the difference between let and var
19	What is the reason to choose the name let as a keyword
20	How do you redeclare variables in switch block without an error
21	What is the Temporal Dead Zone
22	What is IIFE(Immediately Invoked Function Expression)
23	How do you decode or encode a URL in JavaScript?
24	What is memoization
25	What is Hoisting
26	What are classes in ES6
27	What are closures
28	What are modules
29	Why do you need modules
30	What is scope in javascript
31	What is a service worker
32	How do you manipulate DOM using a service worker
33	How do you reuse information across service worker restarts
34	What is IndexedDB
35	What is web storage
36	What is a post message
37	What is a cookie
38	Why do you need a Cookie
39	What are the options in a cookie

No.	Questions
40	How do you delete a cookie
41	What are the differences between cookie, local storage and session storage
42	What is the main difference between localStorage and sessionStorage
43	How do you access web storage
44	What are the methods available on session storage
45	What is a storage event and its event handler
46	Why do you need web storage
47	How do you check web storage browser support
48	How do you check web workers browser support
49	Give an example of web worker
50	What are the restrictions of web workers on DOM
51	What is a promise
52	Why do you need a promise
53	What are the three states of promise
54	What is a callback function
55	Why do we need callbacks
56	What is a callback hell
57	What is server-sent events
58	How do you receive server-sent event notifications
59	How do you check browser support for server-sent events
60	What are the events available for server sent events
61	What are the main rules of promise
62	What is callback in callback
63	What is promise chaining
64	What is promise.all

No.	Questions
65	What is the purpose of race method in promise
66	What is a strict mode in javascript
67	Why do you need strict mode
68	How do you declare strict mode
69	What is the purpose of double exclamation
70	What is the purpose of delete operator
71	What is typeof operator
72	What is undefined property
73	What is null value
74	What is the difference between null and undefined
75	What is eval
76	What is the difference between window and document
77	How do you access history in javascript
78	How do you detect caps lock key turned on or not
79	What is isNaN
80	What are the differences between undeclared and undefined variables
81	What are global variables
82	What are the problems with global variables
83	What is NaN property
84	What is the purpose of isFinite function
85	What is an event flow
86	What is event bubbling
87	What is event capturing
88	How do you submit a form using JavaScript
89	How do you find operating system details

No.	Questions
90	What is the difference between document load and DOMContentLoaded events
91	What is the difference between native, host and user objects
92	What are the tools or techniques used for debugging JavaScript code
93	What are the pros and cons of promises over callbacks
94	What is the difference between an attribute and a property
95	What is same-origin policy
96	What is the purpose of void 0
97	Is JavaScript a compiled or interpreted language
98	Is JavaScript a case-sensitive language
99	Is there any relation between Java and JavaScript
100	What are events
101	Who created javascript
102	What is the use of preventDefault method
103	What is the use of stopPropagation method
104	What are the steps involved in return false
105	What is BOM
106	What is the use of setTimeout
107	What is the use of setInterval
108	Why is JavaScript treated as Single threaded
109	What is an event delegation
110	What is ECMAScript
111	What is JSON
112	What are the syntax rules of JSON
113	What is the purpose JSON stringify

No.	Questions
114	How do you parse JSON string
115	Why do you need JSON
116	What are PWAs
117	What is the purpose of clearTimeout method
118	What is the purpose of clearInterval method
119	How do you redirect new page in javascript
120	How do you check whether a string contains a substring
121	How do you validate an email in javascript
122	How do you get the current url with javascript
123	What are the various url properties of location object
124	How do get query string values in javascript
125	How do you check if a key exists in an object
126	How do you loop through or enumerate javascript object
127	How do you test for an empty object
128	What is an arguments object
129	How do you make first letter of the string in an uppercase
130	What are the pros and cons of for loop
131	How do you display the current date in javascript
132	How do you compare two date objects
133	How do you check if a string starts with another string
134	How do you trim a string in javascript
135	How do you add a key value pair in javascript
136	Is the '!--' notation represents a special operator
137	How do you assign default values to variables
138	How do you define multiline strings

No.	Questions
139	What is an app shell model
140	Can we define properties for functions
141	What is the way to find the number of parameters expected by a function
142	What is a polyfill
143	What are break and continue statements
144	What are js labels
145	What are the benefits of keeping declarations at the top
146	What are the benefits of initializing variables
147	What are the recommendations to create new object
148	How do you define JSON arrays
149	How do you generate random integers
150	Can you write a random integers function to print integers with in a range
151	What is tree shaking
152	What is the need of tree shaking
153	Is it recommended to use eval
154	What is a Regular Expression
155	What are the string methods available in Regular expression
156	What are modifiers in regular expression
157	What are regular expression patterns
158	What is a RegExp object
159	How do you search a string for a pattern
160	What is the purpose of exec method
161	How do you change style of a HTML element
162	What would be the result of 1+2+'3'
163	What is a debugger statement

No.	Questions
164	What is the purpose of breakpoints in debugging
165	Can I use reserved words as identifiers
166	How do you detect a mobile browser
167	How do you detect a mobile browser without regexp
168	How do you get the image width and height using JS
169	How do you make synchronous HTTP request
170	How do you make asynchronous HTTP request
171	How do you convert date to another timezone in javascript
172	What are the properties used to get size of window
173	What is a conditional operator in javascript
174	Can you apply chaining on conditional operator
175	What are the ways to execute javascript after page load
176	What is the difference between proto and prototype
177	Give an example where do you really need semicolon
178	What is a freeze method
179	What is the purpose of freeze method
180	Why do I need to use freeze method
181	How do you detect a browser language preference
182	How to convert string to title case with javascript
183	How do you detect javascript disabled in the page
184	What are various operators supported by javascript
185	What is a rest parameter
186	What happens if you do not use rest parameter as a last argument
187	What are the bitwise operators available in javascript
188	What is a spread operator

No.	Questions
189	How do you determine whether object is frozen or not
190	How do you determine two values same or not using object
191	What is the purpose of using object is method
192	How do you copy properties from one object to other
193	What are the applications of assign method
194	What is a proxy object
195	What is the purpose of seal method
196	What are the applications of seal method
197	What are the differences between freeze and seal methods
198	How do you determine if an object is sealed or not
199	How do you get enumerable key and value pairs
200	What is the main difference between Object.values and Object.entries method
201	How can you get the list of keys of any object
202	How do you create an object with prototype
203	What is a WeakSet
204	What are the differences between WeakSet and Set
205	List down the collection of methods available on WeakSet
206	What is a WeakMap
207	What are the differences between WeakMap and Map
208	List down the collection of methods available on WeakMap
209	What is the purpose of uneval
210	How do you encode an URL
211	How do you decode an URL
212	How do you print the contents of web page
213	What is the difference between uneval and eval

No.	Questions
214	What is an anonymous function
215	What is the precedence order between local and global variables
216	What are javascript accessors
217	How do you define property on Object constructor
218	What is the difference between get and defineProperty
219	What are the advantages of Getters and Setters
220	Can I add getters and setters using defineProperty method
221	What is the purpose of switch-case
222	What are the conventions to be followed for the usage of swtich case
223	What are primitive data types
224	What are the different ways to access object properties
225	What are the function parameter rules
226	What is an error object
227	When you get a syntax error
228	What are the different error names from error object
229	What are the various statements in error handling
230	What are the two types of loops in javascript
231	What is nodejs
232	What is an Intl object
233	How do you perform language specific date and time formatting
234	What is an Iterator
235	How does synchronous iteration works
236	What is an event loop
237	What is call stack
238	What is an event queue

No.	Questions
239	What is a decorator
240	What are the properties of Intl object
241	What is an Unary operator
242	How do you sort elements in an array
243	What is the purpose of compareFunction while sorting arrays
244	How do you reversing an array
245	How do you find min and max value in an array
246	How do you find min and max values without Math functions
247	What is an empty statement and purpose of it
248	How do you get meta data of a module
249	What is a comma operator
250	What is the advantage of a comma operator
251	What is typescript
252	What are the differences between javascript and typescript
253	What are the advantages of typescript over javascript
254	What is an object initializer
255	What is a constructor method
256	What happens if you write constructor more than once in a class
257	How do you call the constructor of a parent class
258	How do you get the prototype of an object
259	What happens If I pass string type for getPrototype method
260	How do you set prototype of one object to another
261	How do you check whether an object can be extendable or not
262	How do you prevent an object to extend
263	What are the different ways to make an object non-extensible

No.	Questions
264	How do you define multiple properties on an object
265	What is MEAN in javascript
266	What Is Obfuscation in javascript
267	Why do you need Obfuscation
268	What is Minification
269	What are the advantages of minification
270	What are the differences between Obfuscation and Encryption
271	What are the common tools used for minification
272	How do you perform form validation using javascript
273	How do you perform form validation without javascript
274	What are the DOM methods available for constraint validation
275	What are the available constraint validation DOM properties
276	What are the list of validity properties
277	Give an example usage of rangeOverflow property
278	Is enums feature available in javascript
279	What is an enum
280	How do you list all properties of an object
281	How do you get property descriptors of an object
282	What are the attributes provided by a property descriptor
283	How do you extend classes
284	How do I modify the url without reloading the page
285	How do you check whether an array includes a particular value or not
286	How do you compare scalar arrays
287	How to get the value from get parameters
288	How do you print numbers with commas as thousand separators

No.	Questions
289	What is the difference between java and javascript
290	Does javascript supports namespace
291	How do you declare namespace
292	How do you invoke javascript code in an iframe from parent page
293	How do get the timezone offset from date
294	How do you load CSS and JS files dynamically
295	What are the different methods to find HTML elements in DOM
296	What is jQuery
297	What is V8 JavaScript engine
298	Why do we call javascript as dynamic language
299	What is a void operator
300	How to set the cursor to wait
301	How do you create an infinite loop
302	Why do you need to avoid with statement
303	What is the output of below for loops
304	List down some of the features of ES6
305	What is ES6
306	Can I redeclare let and const variables
307	Is const variable makes the value immutable
308	What are default parameters
309	What are template literals
310	How do you write multi-line strings in template literals
311	What are nesting templates
312	What are tagged templates
313	What are raw strings

No.	Questions
314	What is destructuring assignment
315	What are default values in destructuring assignment
316	How do you swap variables in destructuring assignment
317	What are enhanced object literals
318	What are dynamic imports
319	What are the use cases for dynamic imports
320	What are typed arrays
321	What are the advantages of module loaders
322	What is collation
323	What is for...of statement
324	What is the output of below spread operator array
325	Is PostMessage secure
326	What are the problems with postmessage target origin as wildcard
327	How do you avoid receiving postMessages from attackers
328	Can I avoid using postMessages completely
329	Is postMessages synchronous
330	What paradigm is Javascript
331	What is the difference between internal and external javascript
332	Is JavaScript faster than server side script
333	How do you get the status of a checkbox
334	What is the purpose of double tilde operator
335	How do you convert character to ASCII code
336	What is ArrayBuffer
337	What is the output of below string expression
338	What is the purpose of Error object

No.	Questions
339	What is the purpose of EvalError object
340	What are the list of cases error thrown from non-strict mode to strict mode
341	Do all objects have prototypes
342	What is the difference between a parameter and an argument
343	What is the purpose of some method in arrays
344	How do you combine two or more arrays
345	What is the difference between Shallow and Deep copy
346	How do you create specific number of copies of a string
347	How do you return all matching strings against a regular expression
348	How do you trim a string at the beginning or ending
349	What is the output of below console statement with unary operator
350	Does javascript uses mixins
351	What is a thunk function
352	What are asynchronous thunks
353	What is the output of below function calls
354	How to remove all line breaks from a string
355	What is the difference between reflow and repaint
356	What happens with negating an array
357	What happens if we add two arrays
358	What is the output of prepend additive operator on falsy values
359	How do you create self string using special characters
360	How do you remove falsy values from an array
361	How do you get unique values of an array
362	What is destructuring aliases
363	How do you map the array values without using map method

No.	Questions
364	How do you empty an array
365	How do you rounding numbers to certain decimals
366	What is the easiest way to convert an array to an object
367	How do you create an array with some data
368	What are the placeholders from console object
369	Is it possible to add CSS to console messages
370	What is the purpose of dir method of console object
371	Is it possible to debug HTML elements in console
372	How do you display data in a tabular format using console object
373	How do you verify that an argument is a Number or not
374	How do you create copy to clipboard button
375	What is the shortcut to get timestamp
376	How do you flattening multi dimensional arrays
377	What is the easiest multi condition checking
378	How do you capture browser back button
379	How do you disable right click in the web page
380	What are wrapper objects
381	What is AJAX
382	What are the different ways to deal with Asynchronous Code
383	How to cancel a fetch request
384	What is web speech API
385	What is minimum timeout throttling
386	How do you implement zero timeout in modern browsers
387	What are tasks in event loop
388	What are microtasks

No.	Questions
389	What are different event loops
390	What is the purpose of queueMicrotask
391	How do you use javascript libraries in typescript file
392	What are the differences between promises and observables
393	What is heap
394	What is an event table
395	What is a microTask queue
396	What is the difference between shim and polyfill
397	How do you detect primitive or non primitive value type
398	What is babel
399	Is Node.js completely single threaded
400	What are the common use cases of observables
401	What is RxJS
402	What is the difference between Function constructor and function declaration
403	What is a Short circuit condition
404	What is the easiest way to resize an array
405	What is an observable
406	What is the difference between function and class declarations
407	What is an async function
408	How do you prevent promises swallowing errors
409	What is deno
410	How do you make an object iterable in javascript
411	What is a Proper Tail Call
412	How do you check an object is a promise or not
413	How to detect if a function is called as constructor

No.	Questions
414	What are the differences between arguments object and rest parameter
415	What are the differences between spread operator and rest parameter
416	What are the different kinds of generators
417	What are the built-in iterables
418	What are the differences between for...of and for...in statements
419	How do you define instance and non-instance properties
420	What is the difference between isNaN and Number.isNaN?
421	How to invoke an IIFE without any extra brackets?
422	Is that possible to use expressions in switch cases?
423	What is the easiest way to ignore promise errors?
424	How do style the console output using CSS?
425	What is nullish coalescing operator(??)?
426	How do you group and nest console output?
427	What is the difference between dense and sparse arrays?
428	What are the different ways to create sparse arrays?
429	What is the difference between setTimeout, setImmediate and process.nextTick?
430	How do you reverse an array without modifying original array?
431	How do you create custom HTML element?
432	What is global execution context?
433	What is function execution context?
434	What is debouncing?
435	What is throttling?
436	What is optional chaining?

1. What are the possible ways to create objects in JavaScript

There are many ways to create objects in javascript as below

i. **Object constructor:**

The simplest way to create an empty object is using the Object constructor. Currently this approach is not recommended.

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

ii. **Object's create method:**

The create method of Object creates a new object by passing the prototype object as a parameter

```
var object = Object.create(null);
```

iii. **Object literal syntax:**

The object literal syntax (or object initializer), is a comma-separated set of name-value pairs wrapped in curly braces.

```
var object = {
     name: "Sudheer"
     age: 34
};

Object literal property values can be of any data type, including array, fu
```

Note: This is an easiest way to create an object

iv. **Function constructor:**

Create any function and apply the new operator to create object instances,

```
function Person(name) {
  this.name = name;
  this.age = 21;
}
var object = new Person("Sudheer");
```

v. **Function constructor with prototype:**

This is similar to function constructor but it uses prototype for their properties and methods,

```
function Person() {}
Person.prototype.name = "Sudheer";
var object = new Person();
```

This is equivalent to an instance created with an object create method with a function prototype and then call that function with an instance and parameters as arguments.

```
function func {};

new func(x, y, z);
```

(OR)

```
// Create a new instance using function prototype.
var newInstance = Object.create(func.prototype)

// Call the function
var result = func.call(newInstance, x, y, z),

// If the result is a non-null object then use it otherwise just use the ne
console.log(result && typeof result === 'object' ? result : newInstance);
```

vi. **ES6 Class syntax:**

ES6 introduces class feature to create the objects

```
class Person {
  constructor(name) {
    this.name = name;
  }
}

var object = new Person("Sudheer");
```

vii. **Singleton pattern:**

A Singleton is an object which can only be instantiated one time. Repeated calls to its constructor return the same instance and this way one can ensure that they don't accidentally create multiple instances.

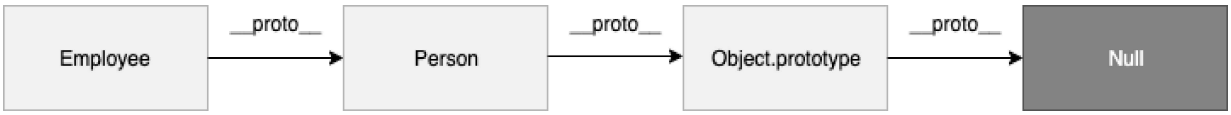
```
var object = new (function () {
  this.name = "Sudheer";
})();
```

⬆ Back to Top

2. What is a prototype chain

Prototype chaining is used to build new types of objects based on existing ones. It is similar to inheritance in a class based language.

The prototype on object instance is available through **Object.getPrototypeOf(object)** or ****proto**** property whereas prototype on constructors function is available through **Object.prototype**.



⬆ Back to Top

3. What is the difference between Call, Apply and Bind

The difference between Call, Apply and Bind can be explained with below examples,

Call: The call() method invokes a function with a given `this` value and arguments provided one by one

```
var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

function invite(greeting1, greeting2) {
  console.log(
    greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
  );
}

invite.call(employee1, "Hello", "How are you?"); // Hello John Rodson, How are y
invite.call(employee2, "Hello", "How are you?"); // Hello Jimmy Baily, How are y
```

Apply: Invokes the function with a given `this` value and allows you to pass in arguments as an array

```
var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

function invite(greeting1, greeting2) {
  console.log(
    greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
  );
}

invite.apply(employee1, ["Hello", "How are you?"]); // Hello John Rodson, How ar
invite.apply(employee2, ["Hello", "How are you?"]); // Hello Jimmy Baily, How ar
```

bind: returns a new function, allowing you to pass any number of arguments

```
var employee1 = { firstName: "John", lastName: "Rodson" };
var employee2 = { firstName: "Jimmy", lastName: "Baily" };

function invite(greeting1, greeting2) {
  console.log(
    greeting1 + " " + this.firstName + " " + this.lastName + ", " + greeting2
  );
}

var inviteEmployee1 = invite.bind(employee1);
var inviteEmployee2 = invite.bind(employee2);
inviteEmployee1("Hello", "How are you?"); // Hello John Rodson, How are you?
inviteEmployee2("Hello", "How are you?"); // Hello Jimmy Baily, How are you?
```

Call and apply are pretty interchangeable. Both execute the current function immediately. You need to decide whether it's easier to send in an array or a comma separated list of arguments. You can remember by treating Call is for **comma** (separated list) and Apply is for **Array**.

Whereas Bind creates a new function that will have `this` set to the first parameter passed to bind().

⬆ Back to Top

4. What is JSON and its common operations

JSON is a text-based data format following JavaScript object syntax, which was popularized by `Douglas Crockford` . It is useful when you want to transmit data across a network and it is basically just a text file with an extension of .json, and a MIME type of application/json

Parsing: Converting a string to a native object

```
JSON.parse(text);
```

Stringification: converting a native object to a string so it can be transmitted across the network

```
JSON.stringify(object);
```

↑ Back to Top

5. What is the purpose of the array slice method

The **slice()** method returns the selected elements in an array as a new array object. It selects the elements starting at the given start argument, and ends at the given optional end argument without including the last element. If you omit the second argument then it selects till the end.

Some of the examples of this method are,

```
let arrayIntegers = [1, 2, 3, 4, 5];
let arrayIntegers1 = arrayIntegers.slice(0, 2); // returns [1,2]
let arrayIntegers2 = arrayIntegers.slice(2, 3); // returns [3]
let arrayIntegers3 = arrayIntegers.slice(4); //returns [5]
```

Note: Slice method won't mutate the original array but it returns the subset as a new array.

↑ Back to Top

6. What is the purpose of the array splice method

The **splice()** method is used either adds/removes items to/from an array, and then returns the removed item. The first argument specifies the array position for insertion or deletion whereas the optional second argument indicates the number of elements to be deleted. Each additional argument is added to the array.

Some of the examples of this method are,

```
let arrayIntegersOriginal1 = [1, 2, 3, 4, 5];
let arrayIntegersOriginal2 = [1, 2, 3, 4, 5];
let arrayIntegersOriginal3 = [1, 2, 3, 4, 5];

let arrayIntegers1 = arrayIntegersOriginal1.splice(0, 2); // returns [1, 2]; ori
let arrayIntegers2 = arrayIntegersOriginal2.splice(3); // returns [4, 5]; origir
let arrayIntegers3 = arrayIntegersOriginal3.splice(3, 1, "a", "b", "c"); //retur
```

Note: Splice method modifies the original array and returns the deleted array.

⬆ Back to Top

7. What is the difference between slice and splice

Some of the major difference in a tabular form

Slice	Splice
Doesn't modify the original array(immutable)	Modifies the original array(mutable)
Returns the subset of original array	Returns the deleted elements as array
Used to pick the elements from array	Used to insert or delete elements to/from array

⬆ Back to Top

8. How do you compare Object and Map

Objects are similar to **Maps** in that both let you set keys to values, retrieve those values, delete keys, and detect whether something is stored at a key. Due to this reason, Objects have been used as Maps historically. But there are important differences that make using a Map preferable in certain cases.

i. The keys of an Object are Strings and Symbols, whereas they can be any value for a Map, including functions, objects, and any primitive.

ii. The keys in Map are ordered while keys added to Object are not. Thus, when iterating over it, a Map object returns keys in order of insertion.

iii. You can get the size of a Map easily with the size property, while the number of properties in an Object must be determined manually.

iv. A Map is an iterable and can thus be directly iterated, whereas iterating over an Object requires obtaining its keys in some fashion and iterating over them.

v. An Object has a prototype, so there are default keys in the map that could collide with your keys if you're not careful. As of ES5 this can be bypassed by using map = Object.create(null), but this is seldom done.

vi. A Map may perform better in scenarios involving frequent addition and removal of key pairs.

⬆ Back to Top

9. What is the difference between == and === operators

JavaScript provides both strict(===, !==) and type-converting(==, !=) equality comparison. The strict operators take type of variable in consideration, while non-strict operators make type correction/conversion based upon values of variables. The strict operators follow the below conditions for different types,

i. Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.

ii. Two numbers are strictly equal when they are numerically equal. i.e, Having the same number value. There are two special cases in this,
   a. NaN is not equal to anything, including NaN.
   b. Positive and negative zeros are equal to one another.

iii. Two Boolean operands are strictly equal if both are true or both are false.

iv. Two objects are strictly equal if they refer to the same Object.

v. Null and Undefined types are not equal with ===, but equal with ==. i.e, null===undefined --> false but null==undefined --> true

Some of the example which covers the above cases,

```
0 == false   // true
0 === false  // false
1 == "1"     // true
1 === "1"    // false
null == undefined // true
null === undefined // false
'0' == false // true
'0' === false // false
[]==[] or []===[] //false, refer different objects in memory
{}=={} or {}==={} //false, refer different objects in memory
```

⬆ Back to Top

10. What are lambda or arrow functions

An arrow function is a shorter syntax for a function expression and does not have its own **this, arguments, super, or new.target**. These functions are best suited for non-method functions, and they cannot be used as constructors.

⬆ Back to Top

11. What is a first class function

In Javascript, functions are first class objects. First-class functions means when functions in that language are treated like any other variable.

For example, in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable. For example, in the below example, handler functions assigned to a listener

```
const handler = () => console.log("This is a click handler function");
document.addEventListener("click", handler);
```

⬆ Back to Top

12. What is a first order function

First-order function is a function that doesn't accept another function as an argument and doesn't return a function as its return value.

```
const firstOrder = () => console.log("I am a first order function!");
```

⬆ Back to Top

13. What is a higher order function

Higher-order function is a function that accepts another function as an argument or returns a function as a return value or both.

```
const firstOrderFunc = () =>
  console.log("Hello, I am a First order function");
```

```
const higherOrder = (ReturnFirstOrderFunc) => ReturnFirstOrderFunc();
higherOrder(firstOrderFunc);
```

⬆ Back to Top

14. What is a unary function

Unary function (i.e. monadic) is a function that accepts exactly one argument. It stands for a single argument accepted by a function.

Let us take an example of unary function,

```
const unaryFunction = (a) => console.log(a + 10); // Add 10 to the given argumer
```

⬆ Back to Top

15. What is the currying function

Currying is the process of taking a function with multiple arguments and turning it into a sequence of functions each with only a single argument. Currying is named after a mathematician **Haskell Curry**. By applying currying, a n-ary function turns it into a unary function.

Let's take an example of n-ary function and how it turns into a currying function,

```
const multiArgFunction = (a, b, c) => a + b + c;
console.log(multiArgFunction(1, 2, 3)); // 6

const curryUnaryFunction = (a) => (b) => (c) => a + b + c;
curryUnaryFunction(1); // returns a function: b => c =>  1 + b + c
curryUnaryFunction(1)(2); // returns a function: c => 3 + c
curryUnaryFunction(1)(2)(3); // returns the number 6
```

Curried functions are great to improve **code reusability** and **functional composition**.

⬆ Back to Top

16. What is a pure function

A **Pure function** is a function where the return value is only determined by its arguments without any side effects. i.e, If you call a function with the same arguments 'n' number of times and 'n' number of places in the application then it will always return the same value.

Let's take an example to see the difference between pure and impure functions,

```
//Impure
let numberArray = [];
const impureAddNumber = (number) => numberArray.push(number);
//Pure
const pureAddNumber = (number) => (argNumberArray) =>
  argNumberArray.concat([number]);

//Display the results
console.log(impureAddNumber(6)); // returns 1
console.log(numberArray); // returns [6]
console.log(pureAddNumber(7)(numberArray)); // returns [6, 7]
console.log(numberArray); // returns [6]
```

As per the above code snippets, the **Push** function is impure itself by altering the array and returning a push number index independent of the parameter value. . Whereas **Concat** on the other hand takes the array and concatenates it with the other array producing a whole new array without side effects. Also, the return value is a concatenation of the previous array.

Remember that Pure functions are important as they simplify unit testing without any side effects and no need for dependency injection. They also avoid tight coupling and make it harder to break your application by not having any side effects. These principles are coming together with **Immutability** concept of ES6 by giving preference to **const** over **let** usage.

⬆ Back to Top

17. What is the purpose of the let keyword

The `let` statement declares a **block scope local variable**. Hence the variables defined with let keyword are limited in scope to the block, statement, or expression on which it is used. Whereas variables declared with the `var` keyword used to define a variable globally, or locally to an entire function regardless of block scope.

Let's take an example to demonstrate the usage,

```
let counter = 30;
if (counter === 30) {
  let counter = 31;
  console.log(counter); // 31
}
console.log(counter); // 30 (because the variable in if block won't exist here)
```

Back to Top

18. What is the difference between let and var

You can list out the differences in a tabular format

var	let
It is been available from the beginning of JavaScript	Introduced as part of ES6
It has function scope	It has block scope
Variables will be hoisted	Hoisted but not initialized

Let's take an example to see the difference,

```
function userDetails(username) {
  if (username) {
    console.log(salary); // undefined due to hoisting
    console.log(age); // ReferenceError: Cannot access 'age' before initializati
    let age = 30;
    var salary = 10000;
  }
  console.log(salary); //10000 (accessible to due function scope)
  console.log(age); //error: age is not defined(due to block scope)
}
userDetails("John");
```

Back to Top

19. What is the reason to choose the name let as a keyword

`let` is a mathematical statement that was adopted by early programming languages like **Scheme** and **Basic**. It has been borrowed from dozens of other languages that use `let` already as a traditional keyword as close to `var` as possible.

⬆ Back to Top

20. How do you redeclare variables in switch block without an error

If you try to redeclare variables in a `switch block` then it will cause errors because there is only one block. For example, the below code block throws a syntax error as below,

```
let counter = 1;
switch (x) {
  case 0:
    let name;
    break;

  case 1:
    let name; // SyntaxError for redeclaration.
    break;
}
```

To avoid this error, you can create a nested block inside a case clause and create a new block scoped lexical environment.

```
let counter = 1;
switch (x) {
  case 0: {
    let name;
    break;
  }
  case 1: {
    let name; // No SyntaxError for redeclaration.
    break;
  }
}
```

⬆ Back to Top

21. What is the Temporal Dead Zone

The Temporal Dead Zone is a behavior in JavaScript that occurs when declaring a variable with the let and const keywords, but not with var. In ECMAScript 6, accessing a `let` or `const` variable before its declaration (within its scope) causes a ReferenceError. The time span when that happens, between the creation of a variable's binding and its declaration, is called the temporal dead zone.

Let's see this behavior with an example,

```
function somemethod() {
  console.log(counter1); // undefined
  console.log(counter2); // ReferenceError
  var counter1 = 1;
  let counter2 = 2;
}
```

⬆ Back to Top

22. What is IIFE(Immediately Invoked Function Expression)

IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined. The signature of it would be as below,

```
(function () {
  // logic here
})();
```

The primary reason to use an IIFE is to obtain data privacy because any variables declared within the IIFE cannot be accessed by the outside world. i.e, If you try to access variables with IIFE then it throws an error as below,

```
(function () {
  var message = "IIFE";
  console.log(message);
})();
console.log(message); //Error: message is not defined
```

⬆ Back to Top

23. How do you decode or encode a URL in JavaScript?

`encodeURI()` function is used to encode an URL. This function requires a URL string as a parameter and return that encoded string. `decodeURI()` function is used to decode an URL. This function requires an encoded URL string as parameter and return that decoded string.

Note: If you want to encode characters such as `/ ? : @ & = + $ #` then you need to use `encodeURIComponent()`.

```
let uri = "employeeDetails?name=john&occupation=manager";
let encoded_uri = encodeURI(uri);
let decoded_uri = decodeURI(encoded_uri);
```

Back to Top

24. What is memoization

Memoization is a programming technique which attempts to increase a function's performance by caching its previously computed results. Each time a memoized function is called, its parameters are used to index the cache. If the data is present, then it can be returned, without executing the entire function. Otherwise the function is executed and then the result is added to the cache. Let's take an example of adding function with memoization,

```
const memoizAddition = () => {
  let cache = {};
  return (value) => {
    if (value in cache) {
      console.log("Fetching from cache");
      return cache[value]; // Here, cache.value cannot be used as property name
    } else {
      console.log("Calculating result");
      let result = value + 20;
      cache[value] = result;
      return result;
    }
  };
};
// returned function from memoizAddition
const addition = memoizAddition();
console.log(addition(20)); //output: 40 calculated
console.log(addition(20)); //output: 40 cached
```

Back to Top

25. What is Hoisting

Hoisting is a JavaScript mechanism where variables, function declarations and classes are moved to the top of their scope before code execution. Remember that JavaScript only hoists declarations, not initialisation. Let's take a simple example of variable hoisting,

```
console.log(message); //output : undefined
var message = "The variable Has been hoisted";
```

The above code looks like as below to the interpreter,

```
var message;
console.log(message);
message = "The variable Has been hoisted";
```

In the same fashion, function declarations are hoisted too

```
message("Good morning"); //Good morning

function message(name) {
  console.log(name);
}
```

This hoisting makes functions to be safely used in code before they are declared.

⬆ Back to Top

26. What are classes in ES6

In ES6, Javascript classes are primarily syntactic sugar over JavaScript's existing prototype-based inheritance. For example, the prototype based inheritance written in function expression as below,

```
function Bike(model, color) {
  this.model = model;
  this.color = color;
}

Bike.prototype.getDetails = function () {
  return this.model + " bike has" + this.color + " color";
};
```

Whereas ES6 classes can be defined as an alternative

```
class Bike {
  constructor(color, model) {
    this.color = color;
```

```
    this.model = model;
  }

  getDetails() {
    return this.model + " bike has" + this.color + " color";
  }
}
```

⬆ Back to Top

27. What are closures

A closure is the combination of a function and the lexical environment within which that function was declared. i.e, It is an inner function that has access to the outer or enclosing function's variables. The closure has three scope chains

i. Own scope where variables defined between its curly brackets
ii. Outer function's variables
iii. Global variables

Let's take an example of closure concept,

```
function Welcome(name) {
  var greetingInfo = function (message) {
    console.log(message + " " + name);
  };
  return greetingInfo;
}
var myFunction = Welcome("John");
myFunction("Welcome "); //Output: Welcome John
myFunction("Hello Mr."); //output: Hello Mr.John
```

As per the above code, the inner function(i.e, greetingInfo) has access to the variables in the outer function scope(i.e, Welcome) even after the outer function has returned.

⬆ Back to Top

28. What are modules

Modules refer to small units of independent, reusable code and also act as the foundation of many JavaScript design patterns. Most of the JavaScript modules export an object literal, a function, or a constructor

⬆ Back to Top

29. Why do you need modules

Below are the list of benefits using modules in javascript ecosystem

i. Maintainability
ii. Reusability
iii. Namespacing

⬆ Back to Top

30. What is scope in javascript

Scope is the accessibility of variables, functions, and objects in some particular part of your code during runtime. In other words, scope determines the visibility of variables and other resources in areas of your code.

⬆ Back to Top

31. What is a service worker

A Service worker is basically a script (JavaScript file) that runs in the background, separate from a web page and provides features that don't need a web page or user interaction. Some of the major features of service workers are Rich offline experiences(offline first web application development), periodic background syncs, push notifications, intercept and handle network requests and programmatically managing a cache of responses.

⬆ Back to Top

32. How do you manipulate DOM using a service worker

Service worker can't access the DOM directly. But it can communicate with the pages it controls by responding to messages sent via the `postMessage` interface, and those pages can manipulate the DOM.

⬆ Back to Top

33. How do you reuse information across service worker restarts

The problem with service worker is that it gets terminated when not in use, and restarted when it's next needed, so you cannot rely on global state within a service worker's `onfetch` and `onmessage` handlers. In this case, service workers will have access to IndexedDB API in order to persist and reuse across restarts.

⬆ Back to Top

34. What is IndexedDB

IndexedDB is a low-level API for client-side storage of larger amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data.

⬆ Back to Top

35. What is web storage

Web storage is an API that provides a mechanism by which browsers can store key/value pairs locally within the user's browser, in a much more intuitive fashion than using cookies. The web storage provides two mechanisms for storing data on the client.

i. **Local storage:** It stores data for current origin with no expiration date.
ii. **Session storage:** It stores data for one session and the data is lost when the browser tab is closed.

⬆ Back to Top

36. What is a post message

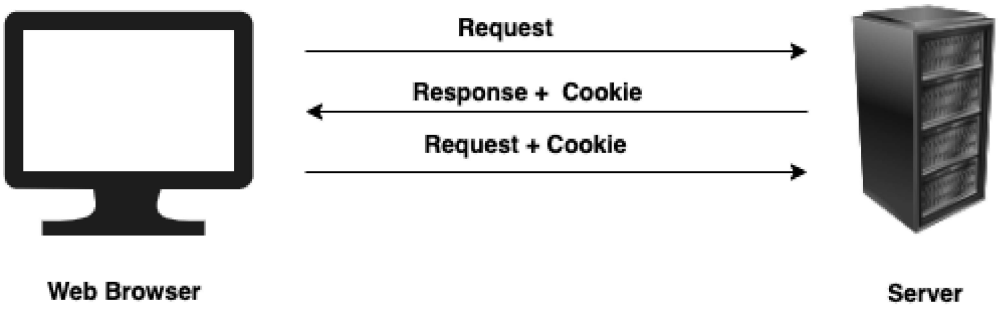
Post message is a method that enables cross-origin communication between Window objects.(i.e, between a page and a pop-up that it spawned, or between a page and an iframe embedded within it). Generally, scripts on different pages are allowed to access each other if and only if the pages follow same-origin policy(i.e, pages share the same protocol, port number, and host).

⬆ Back to Top

37. What is a Cookie

A cookie is a piece of data that is stored on your computer to be accessed by your browser. Cookies are saved as key/value pairs. For example, you can create a cookie named username as below,

```
document.cookie = "username=John";
```



⬆ Back to Top

38. Why do you need a Cookie

Cookies are used to remember information about the user profile(such as username). It basically involves two steps,

i. When a user visits a web page, the user profile can be stored in a cookie.
ii. Next time the user visits the page, the cookie remembers the user profile.

⬆ Back to Top

39. What are the options in a cookie

There are few below options available for a cookie,

i. By default, the cookie is deleted when the browser is closed but you can change this behavior by setting expiry date (in UTC time).

```
document.cookie = "username=John; expires=Sat, 8 Jun 2019 12:00:00 UTC";
```

i. By default, the cookie belongs to a current page. But you can tell the browser what path the cookie belongs to using a path parameter.

```
document.cookie = "username=John; path=/services";
```

⬆ Back to Top

40. How do you delete a cookie

You can delete a cookie by setting the expiry date as a passed date. You don't need to specify a cookie value in this case. For example, you can delete a username cookie in the current page as below.

```
document.cookie =
  "username=; expires=Fri, 07 Jun 2019 00:00:00 UTC; path=/;";
```

Note: You should define the cookie path option to ensure that you delete the right cookie. Some browsers doesn't allow to delete a cookie unless you specify a path parameter.

⬆ Back to Top

41. What are the differences between cookie, local storage and session storage

Below are some of the differences between cookie, local storage and session storage,

Feature	Cookie	Local storage	Session storage
Accessed on client or server side	Both server-side & client-side	client-side only	client-side only
Lifetime	As configured using Expires option	until deleted	until tab is closed
SSL support	Supported	Not supported	Not supported
Maximum data size	4KB	5 MB	5MB

⬆ Back to Top

42. What is the main difference between localStorage and sessionStorage

LocalStorage is the same as SessionStorage but it persists the data even when the browser is closed and reopened(i.e it has no expiration time) whereas in sessionStorage data gets cleared when the page session ends.

⬆ Back to Top

43. How do you access web storage

The Window object implements the `WindowLocalStorage` and `WindowSessionStorage` objects which has `localStorage` (window.localStorage) and `sessionStorage` (window.sessionStorage) properties respectively. These properties create an instance of the Storage object, through which data items can be set, retrieved and removed for a specific domain and storage type (session or local). For example, you can read and write on local storage objects as below

```
localStorage.setItem("logo", document.getElementById("logo").value);
localStorage.getItem("logo");
```

⬆ Back to Top

44. What are the methods available on session storage

The session storage provided methods for reading, writing and clearing the session data

```
// Save data to sessionStorage
sessionStorage.setItem("key", "value");

// Get saved data from sessionStorage
let data = sessionStorage.getItem("key");

// Remove saved data from sessionStorage
sessionStorage.removeItem("key");

// Remove all saved data from sessionStorage
sessionStorage.clear();
```

⬆ Back to Top

45. What is a storage event and its event handler

The StorageEvent is an event that fires when a storage area has been changed in the context of another document. Whereas onstorage property is an EventHandler for processing storage events. The syntax would be as below

```
window.onstorage = functionRef;
```

Let's take the example usage of onstorage event handler which logs the storage key and it's values

```
window.onstorage = function (e) {
  console.log(
    "The " +
      e.key +
      " key has been changed from " +
      e.oldValue +
      " to " +
      e.newValue +
      "."
  );
};
```

⬆ Back to Top

46. Why do you need web storage

Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. Also, the information is never transferred to the server. Hence this is a more recommended approach than Cookies.

⬆ Back to Top

47. How do you check web storage browser support

You need to check browser support for localStorage and sessionStorage before using web storage,

```
if (typeof Storage !== "undefined") {
  // Code for localStorage/sessionStorage.
} else {
  // Sorry! No Web Storage support..
}
```

⬆ Back to Top

48. How do you check web workers browser support

You need to check browser support for web workers before using it

```
if (typeof Worker !== "undefined") {
  // code for Web worker support.
} else {
```

```
  // Sorry! No Web Worker support..
}
```

⬆ Back to Top

49. Give an example of a web worker

You need to follow below steps to start using web workers for counting example

i. Create a Web Worker File: You need to write a script to increment the count value. Let's name it as counter.js

```
let i = 0;

function timedCount() {
  i = i + 1;
  postMessage(i);
  setTimeout("timedCount()", 500);
}

timedCount();
```

Here postMessage() method is used to post a message back to the HTML page

i. Create a Web Worker Object: You can create a web worker object by checking for browser support. Let's name this file as web_worker_example.js

```
if (typeof w == "undefined") {
  w = new Worker("counter.js");
}
```

and we can receive messages from web worker

```
w.onmessage = function (event) {
  document.getElementById("message").innerHTML = event.data;
};
```

i. Terminate a Web Worker: Web workers will continue to listen for messages (even after the external script is finished) until it is terminated. You can use the terminate() method to terminate listening to the messages.

```
w.terminate();
```

i. Reuse the Web Worker: If you set the worker variable to undefined you can reuse the code

```
w = undefined;
```

⬆ Back to Top

50. What are the restrictions of web workers on DOM

WebWorkers don't have access to below javascript objects since they are defined in an external files

i. Window object
ii. Document object
iii. Parent object

⬆ Back to Top

51. What is a promise

A promise is an object that may produce a single value some time in the future with either a resolved value or a reason that it's not resolved(for example, network error). It will be in one of the 3 possible states: fulfilled, rejected, or pending.

The syntax of Promise creation looks like below,

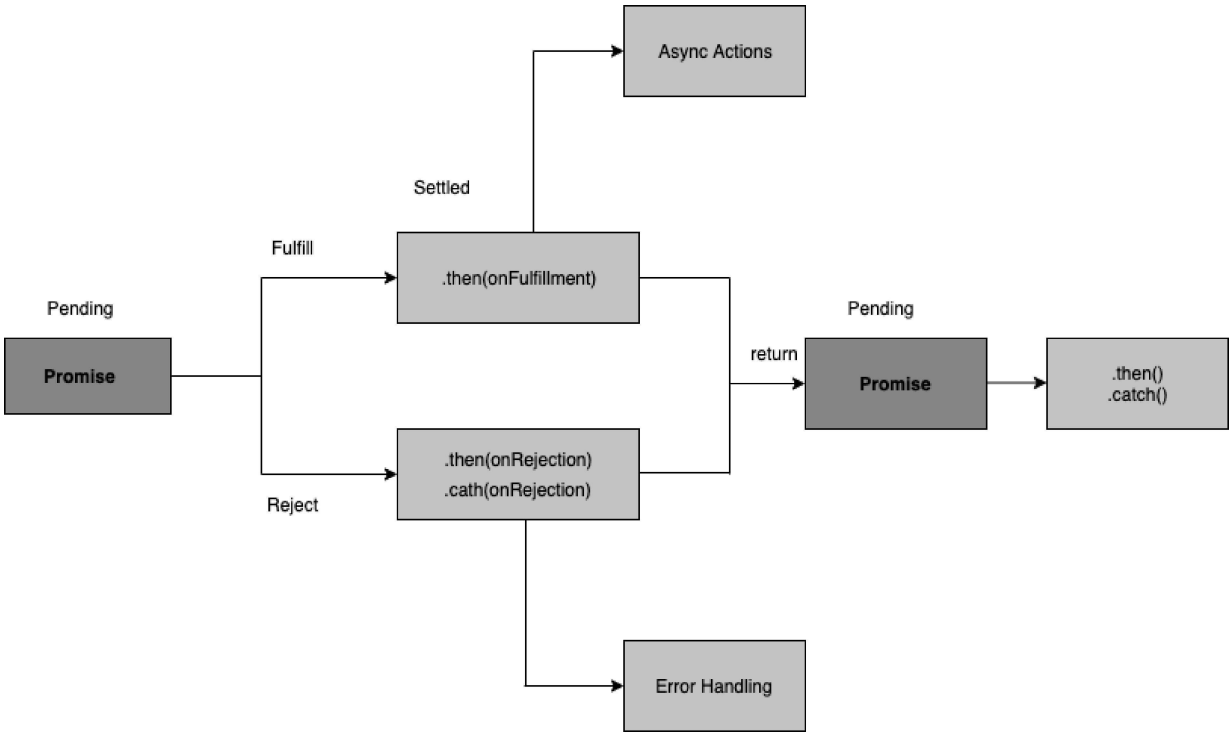
```
const promise = new Promise(function (resolve, reject) {
  // promise description
});
```

The usage of a promise would be as below,

```
const promise = new Promise(
  (resolve) => {
    setTimeout(() => {
      resolve("I'm a Promise!");
    }, 5000);
  },
  (reject) => {}
);

promise.then((value) => console.log(value));
```

The action flow of a promise will be as below,



⬆ Back to Top

52. Why do you need a promise

Promises are used to handle asynchronous operations. They provide an alternative approach for callbacks by reducing the callback hell and writing the cleaner code.

⬆ Back to Top

53. What are the three states of promise

Promises have three states:

i. **Pending:** This is an initial state of the Promise before an operation begins
ii. **Fulfilled:** This state indicates that the specified operation was completed.
iii. **Rejected:** This state indicates that the operation did not complete. In this case an error value will be thrown.

⬆ Back to Top

54. What is a callback function

A callback function is a function passed into another function as an argument. This function is invoked inside the outer function to complete an action. Let's take a simple example of how to use callback function

```
function callbackFunction(name) {
  console.log("Hello " + name);
}

function outerFunction(callback) {
  let name = prompt("Please enter your name.");
  callback(name);
}

outerFunction(callbackFunction);
```

⬆ Back to Top

55. Why do we need callbacks

The callbacks are needed because javascript is an event driven language. That means instead of waiting for a response javascript will keep executing while listening for other events. Let's take an example with the first function invoking an API call(simulated by setTimeout) and the next function which logs the message.

```
function firstFunction() {
  // Simulate a code delay
  setTimeout(function () {
    console.log("First function called");
  }, 1000);
}
function secondFunction() {
  console.log("Second function called");
}
firstFunction();
secondFunction();

Output;
// Second function called
// First function called
```

As observed from the output, javascript didn't wait for the response of the first function and the remaining code block got executed. So callbacks are used in a way to make sure that certain code doesn't execute until the other code finishes execution.

⬆ Back to Top

56. What is a callback hell

Callback Hell is an anti-pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic. The callback hell looks like below,

```
async1(function(){
    async2(function(){
        async3(function(){
            async4(function(){
                ....
            });
        });
    });
});
```

⬆ Back to Top

57. What are server-sent events

Server-sent events (SSE) is a server push technology enabling a browser to receive automatic updates from a server via HTTP connection without resorting to polling. These are a one way communications channel - events flow from server to client only. This has been used in Facebook/Twitter updates, stock price updates, news feeds etc.

⬆ Back to Top

58. How do you receive server-sent event notifications

The EventSource object is used to receive server-sent event notifications. For example, you can receive messages from server as below,

```
if (typeof EventSource !== "undefined") {
  var source = new EventSource("sse_generator.js");
  source.onmessage = function (event) {
    document.getElementById("output").innerHTML += event.data + "<br>";
  };
}
```

⬆ Back to Top

59. How do you check browser support for server-sent events

You can perform browser support for server-sent events before using it as below,

```
if (typeof EventSource !== "undefined") {
  // Server-sent events supported. Let's have some code here!
} else {
  // No server-sent events supported
}
```

⬆ Back to Top

60. What are the events available for server sent events

Below are the list of events available for server sent events

Event	Description
onopen	It is used when a connection to the server is opened
onmessage	This event is used when a message is received
onerror	It happens when an error occurs

⬆ Back to Top

61. What are the main rules of promise

A promise must follow a specific set of rules,

i. A promise is an object that supplies a standard-compliant `.then()` method
ii. A pending promise may transition into either fulfilled or rejected state
iii. A fulfilled or rejected promise is settled and it must not transition into any other state.
iv. Once a promise is settled, the value must not change.

⬆ Back to Top

62. What is callback in callback

You can nest one callback inside in another callback to execute the actions sequentially one by one. This is known as callbacks in callbacks.

```
loadScript("/script1.js", function (script) {
  console.log("first script is loaded");

  loadScript("/script2.js", function (script) {
    console.log("second script is loaded");

    loadScript("/script3.js", function (script) {
      console.log("third script is loaded");
      // after all scripts are loaded
    });
  });
});
```

⬆ Back to Top

63. What is promise chaining

The process of executing a sequence of asynchronous tasks one after another using promises is known as Promise chaining. Let's take an example of promise chaining for calculating the final result,

```
new Promise(function (resolve, reject) {
  setTimeout(() => resolve(1), 1000);
})
  .then(function (result) {
    console.log(result); // 1
    return result * 2;
  })
  .then(function (result) {
    console.log(result); // 2
    return result * 3;
  })
  .then(function (result) {
    console.log(result); // 6
    return result * 4;
  });
```

In the above handlers, the result is passed to the chain of .then() handlers with the below work flow,

i. The initial promise resolves in 1 second,

ii. After that `.then` handler is called by logging the result(1) and then return a promise with the value of result * 2.

iii. After that the value passed to the next `.then` handler by logging the result(2) and return a promise with result * 3.

iv. Finally the value passed to the last `.then` handler by logging the result(6) and return a promise with result * 4.

⬆ Back to Top

64. What is promise.all

Promise.all is a promise that takes an array of promises as an input (an iterable), and it gets resolved when all the promises get resolved or any one of them gets rejected. For example, the syntax of promise.all method is below,

```
Promise.all([Promise1, Promise2, Promise3]) .then(result) => {   console.log(res
```

Note: Remember that the order of the promises(output the result) is maintained as per input order.

⬆ Back to Top

65. What is the purpose of the race method in promise

Promise.race() method will return the promise instance which is firstly resolved or rejected. Let's take an example of race() method where promise2 is resolved first

```
var promise1 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 500, "one");
});
var promise2 = new Promise(function (resolve, reject) {
  setTimeout(resolve, 100, "two");
});

Promise.race([promise1, promise2]).then(function (value) {
  console.log(value); // "two" // Both promises will resolve, but promise2 is fa
});
```

⬆ Back to Top

66. What is a strict mode in javascript

Strict Mode is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This way it prevents certain actions from being taken and throws more exceptions. The literal expression `"use strict";` instructs the browser to use the javascript code in the Strict mode.

⬆ Back to Top

67. Why do you need strict mode

Strict mode is useful to write "secure" JavaScript by notifying "bad syntax" into real errors. For example, it eliminates accidentally creating a global variable by throwing an error and also throws an error for assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object.

⬆ Back to Top

68. How do you declare strict mode

The strict mode is declared by adding "use strict"; to the beginning of a script or a function. If declared at the beginning of a script, it has global scope.

```
"use strict";
x = 3.14; // This will cause an error because x is not declared
```

and if you declare inside a function, it has local scope

```
x = 3.14; // This will not cause an error.
myFunction();

function myFunction() {
  "use strict";
  y = 3.14; // This will cause an error
}
```

⬆ Back to Top

69. What is the purpose of double exclamation

The double exclamation or negation(!!) ensures the resulting type is a boolean. If it was falsey (e.g. 0, null, undefined, etc.), it will be false, otherwise, true. For example, you can test IE version using this expression as below,

```
let isIE8 = false;
isIE8 = !!navigator.userAgent.match(/MSIE 8.0/);
console.log(isIE8); // returns true or false
```

If you don't use this expression then it returns the original value.

```
console.log(navigator.userAgent.match(/MSIE 8.0/)); // returns either an Array c
```

Note: The expression !! is not an operator, but it is just twice of ! operator.

⬆ Back to Top

70. What is the purpose of the delete operator

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

```
var user = { name: "John", age: 20 };
delete user.age;

console.log(user); // {name: "John"}
```

⬆ Back to Top

71. What is the typeof operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable. It returns the type of a variable or an expression.

```
typeof "John Abraham"; // Returns "string"
typeof (1 + 2); // Returns "number"
```

⬆ Back to Top

72. What is undefined property

The undefined property indicates that a variable has not been assigned a value, or not declared at all. The type of undefined value is undefined too.

```
var user; // Value is undefined, type is undefined
console.log(typeof user); //undefined
```

Any variable can be emptied by setting the value to undefined.

```
user = undefined;
```

⬆ Back to Top

73. What is null value

The value null represents the intentional absence of any object value. It is one of JavaScript's primitive values. The type of null value is object. You can empty the variable by setting the value to null.

```
var user = null;
console.log(typeof user); //object
```

⬆ Back to Top

74. What is the difference between null and undefined

Below are the main differences between null and undefined,

Null	Undefined
It is an assignment value which indicates that variable points to no object.	It is not an assignment value where a variable has been declared but has not yet been assigned a value.
Type of null is object	Type of undefined is undefined
The null value is a primitive value that represents the null, empty, or non-existent reference.	The undefined value is a primitive value used when a variable has not been assigned a value.
Indicates the absence of a value for a variable	Indicates absence of variable itself
Converted to zero (0) while performing primitive operations	Converted to NaN while performing primitive operations

⬆ Back to Top

75. What is eval

The eval() function evaluates JavaScript code represented as a string. The string can be a JavaScript expression, variable, statement, or sequence of statements.

```
console.log(eval("1 + 2")); //  3
```

⬆ Back to Top

76. What is the difference between window and document

Below are the main differences between window and document,

Window	Document
It is the root level element in any web page	It is the direct child of the window object. This is also known as Document Object Model(DOM)
By default window object is available implicitly in the page	You can access it via window.document or document.
It has methods like alert(), confirm() and properties like document, location	It provides methods like getElementById, getElementsByTagName, createElement etc

⬆ Back to Top

77. How do you access history in javascript

The window.history object contains the browser's history. You can load previous and next URLs in the history using back() and next() methods.

```
function goBack() {
  window.history.back();
}
function goForward() {
  window.history.forward();
}
```

Note: You can also access history without window prefix.

⬆ Back to Top

78. How do you detect caps lock key turned on or not

The `mouseEvent getModifierState()` is used to return a boolean value that indicates whether the specified modifier key is activated or not. The modifiers such as CapsLock, ScrollLock and NumLock are activated when they are clicked, and deactivated when they are clicked again.

Let's take an input element to detect the CapsLock on/off behavior with an example,

```
<input type="password" onmousedown="enterInput(event)" />

<p id="feedback"></p>

<script>
  function enterInput(e) {
    var flag = e.getModifierState("CapsLock");
    if (flag) {
      document.getElementById("feedback").innerHTML = "CapsLock activated";
    } else {
      document.getElementById("feedback").innerHTML =
        "CapsLock not activated";
    }
  }
</script>
```

⬆ Back to Top

79. What is isNaN

The isNaN() function is used to determine whether a value is an illegal number (Not-a-Number) or not. i.e, This function returns true if the value equates to NaN. Otherwise it returns false.

```
isNaN("Hello"); //true
isNaN("100"); //false
```

⬆ Back to Top

80. What are the differences between undeclared and undefined variables

Below are the major differences between undeclared(not defined) and undefined variables,

undeclared	undefined
These variables do not exist in a program and are not declared	These variables declared in the program but have not assigned any value
If you try to read the value of an undeclared variable, then a runtime error is encountered	If you try to read the value of an undefined variable, an undefined value is returned.

⬆ Back to Top

81. What are global variables

Global variables are those that are available throughout the length of the code without any scope. The var keyword is used to declare a local variable but if you omit it then it will become global variable

```
msg = "Hello"; // var is missing, it becomes global variable
```

⬆ Back to Top

82. What are the problems with global variables

The problem with global variables is the conflict of variable names of local and global scope. It is also difficult to debug and test the code that relies on global variables.

⬆ Back to Top

83. What is NaN property

The NaN property is a global property that represents "Not-a-Number" value. i.e, It indicates that a value is not a legal number. It is very rare to use NaN in a program but it can be used as return value for few cases

```
Math.sqrt(-1);
parseInt("Hello");
```

⬆ Back to Top

84. What is the purpose of isFinite function

The isFinite() function is used to determine whether a number is a finite, legal number. It returns false if the value is +infinity, -infinity, or NaN (Not-a-Number), otherwise it returns true.

```
isFinite(Infinity); // false
isFinite(NaN); // false
isFinite(-Infinity); // false

isFinite(100); // true
```

⬆ Back to Top

85. What is an event flow

Event flow is the order in which event is received on the web page. When you click an element that is nested in various other elements, before your click actually reaches its destination, or target element, it must trigger the click event for each of its parent elements first, starting at the top with the global window object. There are two ways of event flow

i. Top to Bottom(Event Capturing)
ii. Bottom to Top (Event Bubbling)

⬆ Back to Top

86. What is event bubbling

Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element.

⬆ Back to Top

87. What is event capturing

Event capturing is a type of event propagation where the event is first captured by the outermost element, and then successively triggers on the descendants (children) of the target element in the same nesting hierarchy till it reaches the innermost DOM element.

⬆ Back to Top

88. How do you submit a form using JavaScript

You can submit a form using `document.forms[0].submit()` . All the form input's information is submitted using onsubmit event handler

```
function submit() {
  document.forms[0].submit();
}
```

⬆ Back to Top

89. How do you find operating system details

The window.navigator object contains information about the visitor's browser OS details. Some of the OS properties are available under platform property,

```
console.log(navigator.platform);
```

⬆ Back to Top

90. What is the difference between document load and DOMContentLoaded events

The `DOMContentLoaded` event is fired when the initial HTML document has been completely loaded and parsed, without waiting for assets(stylesheets, images, and subframes) to finish loading. Whereas The load event is fired when the whole page has loaded, including all dependent resources(stylesheets, images).

⬆ Back to Top

91. What is the difference between native, host and user objects

`Native objects` are objects that are part of the JavaScript language defined by the ECMAScript specification. For example, String, Math, RegExp, Object, Function etc core objects defined in the ECMAScript spec. `Host objects` are objects provided by the browser or runtime environment (Node). For example, window, XmlHttpRequest, DOM nodes etc are considered as host objects. `User objects` are objects defined in the javascript code. For example, User objects created for profile information.

⬆ Back to Top

92. What are the tools or techniques used for debugging JavaScript code

You can use below tools or techniques for debugging javascript

i. Chrome Devtools
ii. debugger statement
iii. Good old console.log statement

⬆ Back to Top

93. What are the pros and cons of promises over callbacks

Below are the list of pros and cons of promises over callbacks,

Pros:

i. It avoids callback hell which is unreadable
ii. Easy to write sequential asynchronous code with .then()
iii. Easy to write parallel asynchronous code with Promise.all()
iv. Solves some of the common problems of callbacks(call the callback too late, too early, many times and swallow errors/exceptions)

Cons:

i. It makes little complex code
ii. You need to load a polyfill if ES6 is not supported

⬆ Back to Top

94. What is the difference between an attribute and a property

Attributes are defined on the HTML markup whereas properties are defined on the DOM. For example, the below HTML element has 2 attributes type and value,

```
<input type="text" value="Name:">
```

You can retrieve the attribute value as below,

```
const input = document.querySelector("input");
console.log(input.getAttribute("value")); // Good morning
console.log(input.value); // Good morning
```

And after you change the value of the text field to "Good evening", it becomes like

```
console.log(input.getAttribute("value")); // Good evening
console.log(input.value); // Good evening
```

⬆ Back to Top

95. What is same-origin policy

The same-origin policy is a policy that prevents JavaScript from making requests across domain boundaries. An origin is defined as a combination of URI scheme, hostname, and port number. If you enable this policy then it prevents a malicious script on one page from obtaining access to sensitive data on another web page using Document Object Model(DOM).

⬆ Back to Top

96. What is the purpose of void 0

Void(0) is used to prevent the page from refreshing. This will be helpful to eliminate the unwanted side-effect, because it will return the undefined primitive value. It is commonly used for HTML documents that use href="JavaScript:Void(0);" within an `<a>` element. i.e, when you click a link, the browser loads a new page or refreshes the same page. But this behavior will be prevented using this expression. For example, the below link notify the message without reloading the page

```
<a href="JavaScript:void(0);" onclick="alert('Well done!')">
  Click Me!
</a>
```

⬆ Back to Top

97. Is JavaScript a compiled or interpreted language

JavaScript is an interpreted language, not a compiled language. An interpreter in the browser reads over the JavaScript code, interprets each line, and runs it. Nowadays modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run.

⬆ Back to Top

98. Is JavaScript a case-sensitive language

Yes, JavaScript is a case sensitive language. The language keywords, variables, function & object names, and any other identifiers must always be typed with a consistent capitalization of letters.

⬆ Back to Top

99. Is there any relation between Java and JavaScript

No, they are entirely two different programming languages and have nothing to do with each other. But both of them are Object Oriented Programming languages and like many other languages, they follow similar syntax for basic features(if, else, for, switch, break, continue etc).

⬆ Back to Top

100. What are events

Events are "things" that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can `react` on these events. Some of the examples of HTML events are,

i. Web page has finished loading
ii. Input field was changed
iii. Button was clicked

Let's describe the behavior of click event for button element,

```
<!doctype html>
<html>
 <head>
   <script>
     function greeting() {
```

```
        alert('Hello! Good morning');
      }
    </script>
  </head>
  <body>
    <button type="button" onclick="greeting()">Click me</button>
  </body>
</html>
```

⬆ Back to Top

101. Who created javascript

JavaScript was created by Brendan Eich in 1995 during his time at Netscape Communications. Initially it was developed under the name `Mocha`, but later the language was officially called `LiveScript` when it first shipped in beta releases of Netscape.

⬆ Back to Top

102. What is the use of preventDefault method

The preventDefault() method cancels the event if it is cancelable, meaning that the default action or behaviour that belongs to the event will not occur. For example, prevent form submission when clicking on submit button and prevent opening the page URL when clicking on hyperlink are some common use cases.

```
document
  .getElementById("link")
  .addEventListener("click", function (event) {
    event.preventDefault();
  });
```

Note: Remember that not all events are cancelable.

⬆ Back to Top

103. What is the use of stopPropagation method

The stopPropagation method is used to stop the event from bubbling up the event chain. For example, the below nested divs with stopPropagation method prevents default event propagation when clicking on nested div(Div1)

```
<p>Click DIV1 Element</p>
<div onclick="secondFunc()">DIV 2
  <div onclick="firstFunc(event)">DIV 1</div>
</div>

<script>
function firstFunc(event) {
  alert("DIV 1");
  event.stopPropagation();
}

function secondFunc() {
  alert("DIV 2");
}
</script>
```

↑ Back to Top

104. What are the steps involved in return false usage

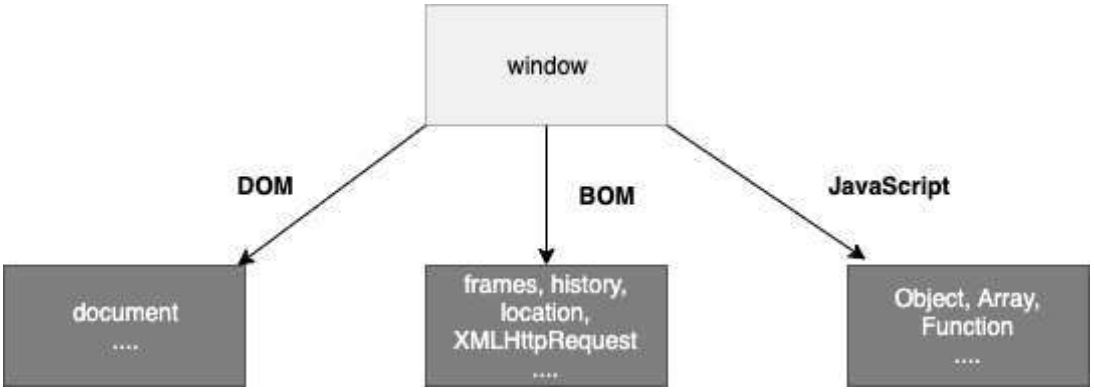
The return false statement in event handlers performs the below steps,

i. First it stops the browser's default action or behaviour.
ii. It prevents the event from propagating the DOM
iii. Stops callback execution and returns immediately when called.

↑ Back to Top

105. What is BOM

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser. It consists of the objects navigator, history, screen, location and document which are children of the window. The Browser Object Model is not standardized and can change based on different browsers.



⬆ Back to Top

106. What is the use of setTimeout

The setTimeout() method is used to call a function or evaluate an expression after a specified number of milliseconds. For example, let's log a message after 2 seconds using setTimeout method,

```
setTimeout(function () {
  console.log("Good morning");
}, 2000);
```

⬆ Back to Top

107. What is the use of setInterval

The setInterval() method is used to call a function or evaluate an expression at specified intervals (in milliseconds). For example, let's log a message after 2 seconds using setInterval method,

```
setInterval(function () {
  console.log("Good morning");
}, 2000);
```

⬆ Back to Top

108. Why is JavaScript treated as Single threaded

JavaScript is a single-threaded language. Because the language specification does not allow the programmer to write code so that the interpreter can run parts of it in parallel in multiple threads or processes. Whereas languages like java, go, C++ can make multi-threaded and multi-process programs.

⬆ Back to Top

109. What is an event delegation

Event delegation is a technique for listening to events where you delegate a parent element as the listener for all of the events that happen inside it.

For example, if you wanted to detect field changes in inside a specific form, you can use event delegation technique,

```
var form = document.querySelector("#registration-form");

// Listen for changes to fields inside the form
form.addEventListener(
  "input",
  function (event) {
    // Log the field that was changed
    console.log(event.target);
  },
  false
);
```

⬆ Back to Top

110. What is ECMAScript

ECMAScript is the scripting language that forms the basis of JavaScript. ECMAScript standardized by the ECMA International standards organization in the ECMA-262 and ECMA-402 specifications. The first edition of ECMAScript was released in 1997.

⬆ Back to Top

111. What is JSON

JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. It is based on a subset of JavaScript language in the way objects are built in JavaScript.

⬆ Back to Top

112. What are the syntax rules of JSON

Below are the list of syntax rules of JSON

i. The data is in name/value pairs
ii. The data is separated by commas
iii. Curly braces hold objects
iv. Square brackets hold arrays

⬆ Back to Top

113. What is the purpose JSON stringify

When sending data to a web server, the data has to be in a string format. You can achieve this by converting JSON object into a string using stringify() method.

```
var userJSON = { name: "John", age: 31 };
var userString = JSON.stringify(userJSON);
console.log(userString); //"{"name":"John","age":31}"
```

⬆ Back to Top

114. How do you parse JSON string

When receiving the data from a web server, the data is always in a string format. But you can convert this string value to a javascript object using parse() method.

```
var userString = '{"name":"John","age":31}';
var userJSON = JSON.parse(userString);
console.log(userJSON); // {name: "John", age: 31}
```

⬆ Back to Top

115. Why do you need JSON

When exchanging data between a browser and a server, the data can only be text. Since JSON is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

⬆ Back to Top

116. What are PWAs

Progressive web applications (PWAs) are a type of mobile app delivered through the web, built using common web technologies including HTML, CSS and JavaScript. These PWAs are deployed to servers, accessible through URLs, and indexed by search engines.

⬆ Back to Top

117. What is the purpose of clearTimeout method

The clearTimeout() function is used in javascript to clear the timeout which has been set by setTimeout()function before that. i.e, The return value of setTimeout() function is stored in a variable and it's passed into the clearTimeout() function to clear the timer.

For example, the below setTimeout method is used to display the message after 3 seconds. This timeout can be cleared by the clearTimeout() method.

```
<script>
var msg;
function greeting() {
   alert('Good morning');
}
function start() {
  msg =setTimeout(greeting, 3000);

}

function stop() {
    clearTimeout(msg);
}
</script>
```

↑ Back to Top

118. What is the purpose of clearInterval method

The clearInterval() function is used in javascript to clear the interval which has been set by setInterval() function. i.e, The return value returned by setInterval() function is stored in a variable and it's passed into the clearInterval() function to clear the interval.

For example, the below setInterval method is used to display the message for every 3 seconds. This interval can be cleared by the clearInterval() method.

```
<script>
var msg;
function greeting() {
   alert('Good morning');
}
function start() {
  msg = setInterval(greeting, 3000);

}

function stop() {
```

```
    clearInterval(msg);
}
</script>
```

⬆ Back to Top

119. How do you redirect new page in javascript

In vanilla javascript, you can redirect to a new page using the `location` property of window object. The syntax would be as follows,

```
function redirect() {
  window.location.href = "newPage.html";
}
```

⬆ Back to Top

120. How do you check whether a string contains a substring

There are 3 possible ways to check whether a string contains a substring or not,

i. **Using includes:** ES6 provided `String.prototype.includes` method to test a string contains a substring

```
var mainString = "hello",
  subString = "hell";
mainString.includes(subString);
```

i. **Using indexOf:** In an ES5 or older environment, you can use `String.prototype.indexOf` which returns the index of a substring. If the index value is not equal to -1 then it means the substring exists in the main string.

```
var mainString = "hello",
  subString = "hell";
mainString.indexOf(subString) !== -1;
```

i. **Using RegEx:** The advanced solution is using Regular expression's test method(`RegExp.test`), which allows for testing for against regular expressions

```
var mainString = "hello",
  regex = /hell/;
regex.test(mainString);
```

⬆ Back to Top

121. How do you validate an email in javascript

You can validate an email in javascript using regular expressions. It is recommended to do validations on the server side instead of the client side. Because the javascript can be disabled on the client side.

```
function validateEmail(email) {
  var re =
    /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3
  return re.test(String(email).toLowerCase());
}
```

⬆ Back to Top

The above regular expression accepts unicode characters.

122. How do you get the current url with javascript

You can use `window.location.href` expression to get the current url path and you can use the same expression for updating the URL too. You can also use `document.URL` for read-only purposes but this solution has issues in FF.

```
console.log("location.href", window.location.href); // Returns full URL
```

⬆ Back to Top

123. What are the various url properties of location object

The below `Location` object properties can be used to access URL components of the page,

i. href - The entire URL
ii. protocol - The protocol of the URL
iii. host - The hostname and port of the URL
iv. hostname - The hostname of the URL
v. port - The port number in the URL
vi. pathname - The path name of the URL

vii. search - The query portion of the URL

viii. hash - The anchor portion of the URL

⬆ Back to Top

124. How do get query string values in javascript

You can use URLSearchParams to get query string values in javascript. Let's see an example to get the client code value from URL query string,

```
const urlParams = new URLSearchParams(window.location.search);
const clientCode = urlParams.get("clientCode");
```

⬆ Back to Top

125. How do you check if a key exists in an object

You can check whether a key exists in an object or not using three approaches,

i. **Using in operator:** You can use the in operator whether a key exists in an object or not

```
"key" in obj;
```

and If you want to check if a key doesn't exist, remember to use parenthesis,

```
!("key" in obj);
```

i. **Using hasOwnProperty method:** You can use `hasOwnProperty` to particularly test for properties of the object instance (and not inherited properties)

```
obj.hasOwnProperty("key"); // true
```

i. **Using undefined comparison:** If you access a non-existing property from an object, the result is undefined. Let's compare the properties against undefined to determine the existence of the property.

```
const user = {
  name: "John",
};
```

```
console.log(user.name !== undefined); // true
console.log(user.nickName !== undefined); // false
```

⬆ Back to Top

126. How do you loop through or enumerate javascript object

You can use the `for-in` loop to loop through javascript object. You can also make sure that the key you get is an actual property of an object, and doesn't come from the prototype using `hasOwnProperty` method.

```
var object = {
  k1: "value1",
  k2: "value2",
  k3: "value3",
};

for (var key in object) {
  if (object.hasOwnProperty(key)) {
    console.log(key + " -> " + object[key]); // k1 -> value1 ...
  }
}
```

⬆ Back to Top

127. How do you test for an empty object

There are different solutions based on ECMAScript versions

i. **Using Object entries(ECMA 7+):** You can use object entries length along with constructor type.

```
Object.entries(obj).length === 0 && obj.constructor === Object; // Since date ob
```

i. **Using Object keys(ECMA 5+):** You can use object keys length along with constructor type.

```
Object.keys(obj).length === 0 && obj.constructor === Object; // Since date objec
```

i. **Using for-in with hasOwnProperty(Pre-ECMA 5):** You can use a for-in loop along with hasOwnProperty.

```
function isEmpty(obj) {
  for (var prop in obj) {
    if (obj.hasOwnProperty(prop)) {
      return false;
    }
  }

  return JSON.stringify(obj) === JSON.stringify({});
}
```

↑ Back to Top

128. What is an arguments object

The arguments object is an Array-like object accessible inside functions that contains the values of the arguments passed to that function. For example, let's see how to use arguments object inside sum function,

```
function sum() {
  var total = 0;
  for (var i = 0, len = arguments.length; i < len; ++i) {
    total += arguments[i];
  }
  return total;
}

sum(1, 2, 3); // returns 6
```

Note: You can't apply array methods on arguments object. But you can convert into a regular array as below.

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

↑ Back to Top

129. How do you make first letter of the string in an uppercase

You can create a function which uses a chain of string methods such as charAt, toUpperCase and slice methods to generate a string with the first letter in uppercase.

```
function capitalizeFirstLetter(string) {
  return string.charAt(0).toUpperCase() + string.slice(1);
```

```
}
```

⬆ Back to Top

130. What are the pros and cons of for loop

The for-loop is a commonly used iteration syntax in javascript. It has both pros and cons

Pros

i. Works on every environment
ii. You can use break and continue flow control statements

Cons

i. Too verbose
ii. Imperative
iii. You might face one-by-off errors

⬆ Back to Top

131. How do you display the current date in javascript

You can use `new Date()` to generate a new Date object containing the current date and time. For example, let's display the current date in mm/dd/yyyy

```
var today = new Date();
var dd = String(today.getDate()).padStart(2, "0");
var mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
var yyyy = today.getFullYear();

today = mm + "/" + dd + "/" + yyyy;
document.write(today);
```

⬆ Back to Top

132. How do you compare two date objects

You need to use date.getTime() method to compare date values instead of comparison operators (==, !=, ===, and !== operators)

```
var d1 = new Date();
var d2 = new Date(d1);
console.log(d1.getTime() === d2.getTime()); //True
console.log(d1 === d2); // False
```

⬆ Back to Top

133. How do you check if a string starts with another string

You can use ECMAScript 6's `String.prototype.startsWith()` method to check if a string starts with another string or not. But it is not yet supported in all browsers. Let's see an example to see this usage,

```
"Good morning".startsWith("Good"); // true
"Good morning".startsWith("morning"); // false
```

⬆ Back to Top

134. How do you trim a string in javascript

JavaScript provided a trim method on string types to trim any whitespaces present at the beginning or ending of the string.

```
"  Hello World   ".trim(); //Hello World
```

If your browser(<IE9) doesn't support this method then you can use below polyfill.

```
if (!String.prototype.trim) {
  (function () {
    // Make sure we trim BOM and NBSP
    var rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g;
    String.prototype.trim = function () {
      return this.replace(rtrim, "");
    };
  })();
}
```

⬆ Back to Top

135. How do you add a key value pair in javascript

There are two possible solutions to add new properties to an object. Let's take a simple object to explain these solutions.

```
var object = {
  key1: value1,
  key2: value2,
};
```

i. **Using dot notation:** This solution is useful when you know the name of the property

```
object.key3 = "value3";
```

i. **Using square bracket notation:** This solution is useful when the name of the property is dynamically determined.

```
obj["key3"] = "value3";
```

↑ Back to Top

136. Is the !-- notation represents a special operator

No,that's not a special operator. But it is a combination of 2 standard operators one after the other,

i. A logical not (!)
ii. A prefix decrement (--)

At first, the value decremented by one and then tested to see if it is equal to zero or not for determining the truthy/falsy value.

↑ Back to Top

137. How do you assign default values to variables

You can use the logical or operator `||` in an assignment expression to provide a default value. The syntax looks like as below,

```
var a = b || c;
```

As per the above expression, variable 'a 'will get the value of 'c' only if 'b' is falsy (if is null, false, undefined, 0, empty string, or NaN), otherwise 'a' will get the value of 'b'.

⬆ Back to Top

138. How do you define multiline strings

You can define multiline string literals using the '\' character followed by line terminator.

```
var str =
  "This is a \
very lengthy \
sentence!";
```

But if you have a space after the '\' character, the code will look exactly the same, but it will raise a SyntaxError.

⬆ Back to Top

139. What is an app shell model

An application shell (or app shell) architecture is one way to build a Progressive Web App that reliably and instantly loads on your users' screens, similar to what you see in native applications. It is useful for getting some initial HTML to the screen fast without a network.

⬆ Back to Top

140. Can we define properties for functions

Yes, We can define properties for functions because functions are also objects.

```
fn = function (x) {
  //Function code goes here
};

fn.name = "John";

fn.profile = function (y) {
  //Profile code goes here
};
```

⬆ Back to Top

141. What is the way to find the number of parameters expected by a function

You can use `function.length` syntax to find the number of parameters expected by a function. Let's take an example of `sum` function to calculate the sum of numbers,

```
function sum(num1, num2, num3, num4) {
  return num1 + num2 + num3 + num4;
}
sum.length; // 4 is the number of parameters expected.
```

⬆ Back to Top

142. What is a polyfill

A polyfill is a piece of JS code used to provide modern functionality on older browsers that do not natively support it. For example, Silverlight plugin polyfill can be used to mimic the functionality of an HTML Canvas element on Microsoft Internet Explorer 7.

⬆ Back to Top

143. What are break and continue statements

The break statement is used to "jump out" of a loop. i.e, It breaks the loop and continues executing the code after the loop.

```
for (i = 0; i < 10; i++) {
  if (i === 5) {
    break;
  }
  text += "Number: " + i + "<br>";
}
```

The continue statement is used to "jump over" one iteration in the loop. i.e, It breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
for (i = 0; i < 10; i++) {
  if (i === 5) {
    continue;
  }
```

```
  text += "Number: " + i + "<br>";
}
```

⬆ Back to Top

144. What are js labels

The label statement allows us to name loops and blocks in JavaScript. We can then use these labels to refer back to the code later. For example, the below code with labels avoids printing the numbers when they are same,

```
var i, j;

loop1: for (i = 0; i < 3; i++) {
  loop2: for (j = 0; j < 3; j++) {
    if (i === j) {
      continue loop1;
    }
    console.log("i = " + i + ", j = " + j);
  }
}

// Output is:
//   "i = 1, j = 0"
//   "i = 2, j = 0"
//   "i = 2, j = 1"
```

⬆ Back to Top

145. What are the benefits of keeping declarations at the top

It is recommended to keep all declarations at the top of each script or function. The benefits of doing this are,

i. Gives cleaner code
ii. It provides a single place to look for local variables
iii. Easy to avoid unwanted global variables
iv. It reduces the possibility of unwanted re-declarations

⬆ Back to Top

146. What are the benefits of initializing variables

It is recommended to initialize variables because of the below benefits,

i. It gives cleaner code
ii. It provides a single place to initialize variables
iii. Avoid undefined values in the code

↑ Back to Top

147. What are the recommendations to create new object

It is recommended to avoid creating new objects using `new Object()`. Instead you can initialize values based on it's type to create the objects.

i. Assign {} instead of new Object()
ii. Assign "" instead of new String()
iii. Assign 0 instead of new Number()
iv. Assign false instead of new Boolean()
v. Assign [] instead of new Array()
vi. Assign /()/ instead of new RegExp()
vii. Assign function (){} instead of new Function()

You can define them as an example,

```
var v1 = {};
var v2 = "";
var v3 = 0;
var v4 = false;
var v5 = [];
var v6 = /()/;
var v7 = function () {};
```

↑ Back to Top

148. How do you define JSON arrays

JSON arrays are written inside square brackets and arrays contain javascript objects. For example, the JSON array of users would be as below,

```
"users":[
  {"firstName":"John", "lastName":"Abrahm"},
  {"firstName":"Anna", "lastName":"Smith"},
```

```
  {"firstName":"Shane", "lastName":"Warn"}
]
```

↑ Back to Top

149. How do you generate random integers

You can use Math.random() with Math.floor() to return random integers. For example, if you want generate random integers between 1 to 10, the multiplication factor should be 10,

```
Math.floor(Math.random() * 10) + 1; // returns a random integer from 1 to 10
Math.floor(Math.random() * 100) + 1; // returns a random integer from 1 to 100
```

Note: Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)

↑ Back to Top

150. Can you write a random integers function to print integers with in a range

Yes, you can create a proper random function to return a random number between min and max (both included)

```
function randomInteger(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
randomInteger(1, 100); // returns a random integer from 1 to 100
randomInteger(1, 1000); // returns a random integer from 1 to 1000
```

↑ Back to Top

151. What is tree shaking

Tree shaking is a form of dead code elimination. It means that unused modules will not be included in the bundle during the build process and for that it relies on the static structure of ES2015 module syntax,(i.e. import and export). Initially this has been popularized by the ES2015 module bundler `rollup` .

↑ Back to Top

152. What is the need of tree shaking

Tree Shaking can significantly reduce the code size in any application. i.e, The less code we send over the wire the more performant the application will be. For example, if we just want to create a "Hello World" Application using SPA frameworks then it will take around a few MBs, but by tree shaking it can bring down the size to just a few hundred KBs. Tree shaking is implemented in Rollup and Webpack bundlers.

⬆ Back to Top

153. Is it recommended to use eval

No, it allows arbitrary code to be run which causes a security problem. As we know that the eval() function is used to run text as code. In most of the cases, it should not be necessary to use it.

⬆ Back to Top

154. What is a Regular Expression

A regular expression is a sequence of characters that forms a search pattern. You can use this search pattern for searching data in a text. These can be used to perform all types of text search and text replace operations. Let's see the syntax format now,

```
/pattern/modifiers;
```

For example, the regular expression or search pattern with case-insensitive username would be,

```
/John/i;
```

⬆ Back to Top

155. What are the string methods available in Regular expression

Regular Expressions has two string methods: search() and replace(). The search() method uses an expression to search for a match, and returns the position of the match.

```
var msg = "Hello John";
var n = msg.search(/John/i); // 6
```

The replace() method is used to return a modified string where the pattern is replaced.

```
var msg = "Hello John";
var n = msg.replace(/John/i, "Buttler"); // Hello Buttler
```

⬆ Back to Top

156. What are modifiers in regular expression

Modifiers can be used to perform case-insensitive and global searches. Let's list down some of the modifiers,

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match rather than stops at first match
m	Perform multiline matching

Let's take an example of global modifier,

```
var text = "Learn JS one by one";
var pattern = /one/g;
var result = text.match(pattern); // one,one
```

⬆ Back to Top

157. What are regular expression patterns

Regular Expressions provide a group of patterns in order to match characters. Basically they are categorized into 3 types,

i. **Brackets:** These are used to find a range of characters. For example, below are some use cases,
   a. [abc]: Used to find any of the characters between the brackets(a,b,c)
   b. [0-9]: Used to find any of the digits between the brackets
   c. (a|b): Used to find any of the alternatives separated with |

ii. **Metacharacters:** These are characters with a special meaning For example, below are some use cases,
   a. \d: Used to find a digit

b. \s: Used to find a whitespace character

c. \b: Used to find a match at the beginning or ending of a word

iii. **Quantifiers:** These are useful to define quantities For example, below are some use cases,

a. n+: Used to find matches for any string that contains at least one n

b. n*: Used to find matches for any string that contains zero or more occurrences of n

c. n?: Used to find matches for any string that contains zero or one occurrences of n

⬆ Back to Top

158. What is a RegExp object

RegExp object is a regular expression object with predefined properties and methods. Let's see the simple usage of RegExp object,

```
var regexp = new RegExp("\\w+");
console.log(regexp);
// expected output: /\w+/
```

⬆ Back to Top

159. How do you search a string for a pattern

You can use the test() method of regular expression in order to search a string for a pattern, and return true or false depending on the result.

```
var pattern = /you/;
console.log(pattern.test("How are you?")); //true
```

⬆ Back to Top

160. What is the purpose of exec method

The purpose of exec method is similar to test method but it executes a search for a match in a specified string and returns a result array, or null instead of returning true/false.

```
var pattern = /you/;
console.log(pattern.exec("How are you?")); //["you", index: 8, input: "How are y
```

Back to Top

161. How do you change the style of a HTML element

You can change inline style or classname of a HTML element using javascript

i. **Using style property:** You can modify inline style using style property

```
document.getElementById("title").style.fontSize = "30px";
```

i. **Using ClassName property:** It is easy to modify element class using className property

```
document.getElementById("title").className = "custom-title";
```

Back to Top

162. What would be the result of 1+2+'3'

The output is going to be `33`. Since `1` and `2` are numeric values, the result of the first two digits is going to be a numeric value `3`. The next digit is a string type value because of that the addition of numeric value `3` and string type value `3` is just going to be a concatenation value `33`.

Back to Top

163. What is a debugger statement

The debugger statement invokes any available debugging functionality, such as setting a breakpoint. If no debugging functionality is available, this statement has no effect. For example, in the below function a debugger statement has been inserted. So execution is paused at the debugger statement just like a breakpoint in the script source.

```
function getProfile() {
  // code goes here
  debugger;
```

```
  // code goes here
}
```

⬆ Back to Top

164. What is the purpose of breakpoints in debugging

You can set breakpoints in the javascript code once the debugger statement is executed and the debugger window pops up. At each breakpoint, javascript will stop executing, and let you examine the JavaScript values. After examining values, you can resume the execution of code using the play button.

⬆ Back to Top

165. Can I use reserved words as identifiers

No, you cannot use the reserved words as variables, labels, object or function names. Let's see one simple example,

```
var else = "hello"; // Uncaught SyntaxError: Unexpected token else
```

⬆ Back to Top

166. How do you detect a mobile browser

You can use regex which returns a true or false value depending on whether or not the user is browsing with a mobile.

```
window.mobilecheck = function () {
  var mobileCheck = false;
  (function (a) {
    if (
      /(android|bb\d+|meego).+mobile|avantgo|bada\/|blackberry|blazer|compal|ela
        a
      ) ||
      /1207|6310|6590|3gso|4thp|50[1-6]i|770s|802s|a wa|abac|ac(er|oo|s\-)|ai(ko
        a.substr(0, 4)
      )
    )
      mobileCheck = true;
  })(navigator.userAgent || navigator.vendor || window.opera);
  return mobileCheck;
};
```

Back to Top

167. How do you detect a mobile browser without regexp

You can detect mobile browsers by simply running through a list of devices and checking if the useragent matches anything. This is an alternative solution for RegExp usage,

```
function detectmob() {
  if (
    navigator.userAgent.match(/Android/i) ||
    navigator.userAgent.match(/webOS/i) ||
    navigator.userAgent.match(/iPhone/i) ||
    navigator.userAgent.match(/iPad/i) ||
    navigator.userAgent.match(/iPod/i) ||
    navigator.userAgent.match(/BlackBerry/i) ||
    navigator.userAgent.match(/Windows Phone/i)
  ) {
    return true;
  } else {
    return false;
  }
}
```

Back to Top

168. How do you get the image width and height using JS

You can programmatically get the image and check the dimensions(width and height) using Javascript.

```
var img = new Image();
img.onload = function () {
  console.log(this.width + "x" + this.height);
};
img.src = "http://www.google.com/intl/en_ALL/images/logo.gif";
```

Back to Top

169. How do you make synchronous HTTP request

Browsers provide an XMLHttpRequest object which can be used to make synchronous HTTP requests from JavaScript

```
function httpGet(theUrl) {
  var xmlHttpReq = new XMLHttpRequest();
  xmlHttpReq.open("GET", theUrl, false); // false for synchronous request
  xmlHttpReq.send(null);
  return xmlHttpReq.responseText;
}
```

⬆ Back to Top

170. How do you make asynchronous HTTP request

Browsers provide an XMLHttpRequest object which can be used to make asynchronous HTTP requests from JavaScript by passing the 3rd parameter as true.

```
function httpGetAsync(theUrl, callback) {
  var xmlHttpReq = new XMLHttpRequest();
  xmlHttpReq.onreadystatechange = function () {
    if (xmlHttpReq.readyState == 4 && xmlHttpReq.status == 200)
      callback(xmlHttpReq.responseText);
  };
  xmlHttp.open("GET", theUrl, true); // true for asynchronous
  xmlHttp.send(null);
}
```

⬆ Back to Top

171. How do you convert date to another timezone in javascript

You can use the toLocaleString() method to convert dates in one timezone to another. For example, let's convert current date to British English timezone as below,

```
console.log(event.toLocaleString("en-GB", { timeZone: "UTC" })); //29/06/2019, 0
```

⬆ Back to Top

172. What are the properties used to get size of window

You can use innerWidth, innerHeight, clientWidth, clientHeight properties of windows, document element and document body objects to find the size of a window. Let's use them combination of these properties to calculate the size of a window or document,

```
var width =
  window.innerWidth ||
  document.documentElement.clientWidth ||
  document.body.clientWidth;

var height =
  window.innerHeight ||
  document.documentElement.clientHeight ||
  document.body.clientHeight;
```

⬆ Back to Top

173. What is a conditional operator in javascript

The conditional (ternary) operator is the only JavaScript operator that takes three operands which acts as a shortcut for if statements.

```
var isAuthenticated = false;
console.log(
  isAuthenticated ? "Hello, welcome" : "Sorry, you are not authenticated"
); //Sorry, you are not authenticated
```

⬆ Back to Top

174. Can you apply chaining on conditional operator

Yes, you can apply chaining on conditional operators similar to if ... else if ... else if ... else chain. The syntax is going to be as below,

```
function traceValue(someParam) {
  return condition1
    ? value1
    : condition2
    ? value2
    : condition3
    ? value3
    : value4;
}
```

```
// The above conditional operator is equivalent to:

function traceValue(someParam) {
  if (condition1) {
    return value1;
  } else if (condition2) {
    return value2;
  } else if (condition3) {
    return value3;
  } else {
    return value4;
  }
}
```

⬆ Back to Top

175. What are the ways to execute javascript after page load

You can execute javascript after page load in many different ways,

i. **window.onload:**

```
window.onload = function ...
```

i. **document.onload:**

```
document.onload = function ...
```

i. **body onload:**

```
<body onload="script();">
```

⬆ Back to Top

176. What is the difference between proto and prototype

The `__proto__` object is the actual object that is used in the lookup chain to resolve methods, etc. Whereas `prototype` is the object that is used to build `__proto__` when you create an object with new

```
new Employee().__proto__ === Employee.prototype;
new Employee().prototype === undefined;
```

⬆ Back to Top

177. Give an example where do you really need semicolon

It is recommended to use semicolons after every statement in JavaScript. For example, in the below case it throws an error ".. is not a function" at runtime due to missing semicolon.

```
// define a function
var fn = (function () {
  //...
})(
  // semicolon missing at this line

  // then execute some code inside a closure
  function () {
    //...
  }
)();
```

and it will be interpreted as

```
var fn = (function () {
  //...
})(function () {
  //...
})();
```

In this case, we are passing the second function as an argument to the first function and then trying to call the result of the first function call as a function. Hence, the second function will fail with a "... is not a function" error at runtime.

⬆ Back to Top

178. What is a freeze method

The **freeze()** method is used to freeze an object. Freezing an object does not allow adding new properties to an object,prevents from removing and prevents changing the enumerability, configurability, or writability of existing properties. i.e, It returns the passed object and does not create a frozen copy.

```
const obj = {
  prop: 100,
```

```
};

Object.freeze(obj);
obj.prop = 200; // Throws an error in strict mode

console.log(obj.prop); //100
```

Note: It causes a TypeError if the argument passed is not an object.

⬆ Back to Top

179. What is the purpose of freeze method

Below are the main benefits of using freeze method,

i. It is used for freezing objects and arrays.
ii. It is used to make an object immutable.

⬆ Back to Top

180. Why do I need to use freeze method

In the Object-oriented paradigm, an existing API contains certain elements that are not intended to be extended, modified, or re-used outside of their current context. Hence it works as the `final` keyword which is used in various languages.

⬆ Back to Top

181. How do you detect a browser language preference

You can use navigator object to detect a browser language preference as below,

```
var language =
  (navigator.languages && navigator.languages[0]) || // Chrome / Firefox
  navigator.language || // All browsers
  navigator.userLanguage; // IE <= 10

console.log(language);
```

⬆ Back to Top

182. How to convert string to title case with javascript

Title case means that the first letter of each word is capitalized. You can convert a string to title case using the below function,

```
function toTitleCase(str) {
  return str.replace(/\w\S*/g, function (txt) {
    return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();
  });
}
toTitleCase("good morning john"); // Good Morning John
```

Back to Top

183. How do you detect javascript disabled in the page

You can use the `<noscript>` tag to detect javascript disabled or not. The code block inside `<noscript>` gets executed when JavaScript is disabled, and is typically used to display alternative content when the page generated in JavaScript.

```
<script type="javascript">
    // JS related code goes here
</script>
<noscript>
    <a href="next_page.html?noJS=true">JavaScript is disabled in the page. Pleas
</noscript>
```

Back to Top

184. What are various operators supported by javascript

An operator is capable of manipulating(mathematical and logical computations) a certain value or operand. There are various operators supported by JavaScript as below,

i. **Arithmetic Operators:** Includes + (Addition),– (Subtraction), * (Multiplication), / (Division), % (Modulus), + + (Increment) and – – (Decrement)
ii. **Comparison Operators:** Includes = =(Equal),!= (Not Equal), ===(Equal with type), > (Greater than),> = (Greater than or Equal to),< (Less than),<= (Less than or Equal to)
iii. **Logical Operators:** Includes &&(Logical AND),||(Logical OR),!(Logical NOT)
iv. **Assignment Operators:** Includes = (Assignment Operator), += (Add and Assignment Operator), – = (Subtract and Assignment Operator), *= (Multiply and

Assignment), /= (Divide and Assignment), %= (Modules and Assignment)

v. **Ternary Operators:** It includes conditional(: ?) Operator

vi. **typeof Operator:** It uses to find type of variable. The syntax looks like `typeof variable`

⬆ Back to Top

185. What is a rest parameter

Rest parameter is an improved way to handle function parameters which allows us to represent an indefinite number of arguments as an array. The syntax would be as below,

```
function f(a, b, ...theArgs) {
  // ...
}
```

For example, let's take a sum example to calculate on dynamic number of parameters,

```
function total(…args){
let sum = 0;
for(let i of args){
sum+=i;
}
return sum;
}
console.log(fun(1,2)); //3
console.log(fun(1,2,3)); //6
console.log(fun(1,2,3,4)); //13
console.log(fun(1,2,3,4,5)); //15
```

Note: Rest parameter is added in ES2015 or ES6

⬆ Back to Top

186. What happens if you do not use rest parameter as a last argument

The rest parameter should be the last argument, as its job is to collect all the remaining arguments into an array. For example, if you define a function like below it doesn't make any sense and will throw an error.

```
function someFunc(a,…b,c){
//You code goes here
return;
}
```

⬆ Back to Top

187. What are the bitwise operators available in javascript

Below are the list of bitwise logical operators used in JavaScript

i. Bitwise AND (&)
ii. Bitwise OR (|)
iii. Bitwise XOR (^)
iv. Bitwise NOT (~)
v. Left Shift (<<)
vi. Sign Propagating Right Shift (>>)
vii. Zero fill Right Shift (>>>)

⬆ Back to Top

188. What is a spread operator

Spread operator allows iterables(arrays / objects / strings) to be expanded into single arguments/elements. Let's take an example to see this behavior,

```
function calculateSum(x, y, z) {
  return x + y + z;
}

const numbers = [1, 2, 3];

console.log(calculateSum(...numbers)); // 6
```

⬆ Back to Top

189. How do you determine whether object is frozen or not

Object.isFrozen() method is used to determine if an object is frozen or not.An object is frozen if all of the below conditions hold true,

i. If it is not extensible.
ii. If all of its properties are non-configurable.
iii. If all its data properties are non-writable. The usage is going to be as follows,

```
const object = {
  property: "Welcome JS world",
};
Object.freeze(object);
console.log(Object.isFrozen(object));
```

⬆ Back to Top

190. How do you determine two values same or not using object

The Object.is() method determines whether two values are the same value. For example, the usage with different types of values would be,

```
Object.is("hello", "hello"); // true
Object.is(window, window); // true
Object.is([], []); // false
```

Two values are the same if one of the following holds:

i. both undefined
ii. both null
iii. both true or both false
iv. both strings of the same length with the same characters in the same order
v. both the same object (means both object have same reference)
vi. both numbers and both +0 both -0 both NaN both non-zero and both not NaN and both have the same value.

⬆ Back to Top

191. What is the purpose of using object is method

Some of the applications of Object's `is` method are follows,

i. It is used for comparison of two strings.
ii. It is used for comparison of two numbers.
iii. It is used for comparing the polarity of two numbers.
iv. It is used for comparison of two objects.

⬆ Back to Top

192. How do you copy properties from one object to other

You can use the Object.assign() method which is used to copy the values and properties from one or more source objects to a target object. It returns the target object which has properties and values copied from the target object. The syntax would be as below,

```
Object.assign(target, ...sources);
```

Let's take example with one source and one target object,

```
const target = { a: 1, b: 2 };
const source = { b: 3, c: 4 };

const returnedTarget = Object.assign(target, source);

console.log(target); // { a: 1, b: 3, c: 4 }

console.log(returnedTarget); // { a: 1, b: 3, c: 4 }
```

As observed in the above code, there is a common property(`b`) from source to target so it's value has been overwritten.

⬆ Back to Top

193. What are the applications of assign method

Below are the some of main applications of Object.assign() method,

i. It is used for cloning an object.
ii. It is used to merge objects with the same properties.

⬆ Back to Top

194. What is a proxy object

The Proxy object is used to define custom behavior for fundamental operations such as property lookup, assignment, enumeration, function invocation, etc. The syntax would be as follows,

```
var p = new Proxy(target, handler);
```

Let's take an example of proxy object,

```
var handler = {
  get: function (obj, prop) {
    return prop in obj ? obj[prop] : 100;
  },
};

var p = new Proxy({}, handler);
p.a = 10;
p.b = null;

console.log(p.a, p.b); // 10, null
console.log("c" in p, p.c); // false, 100
```

In the above code, it uses `get` handler which define the behavior of the proxy when an operation is performed on it

⬆ Back to Top

195. What is the purpose of seal method

The **Object.seal()** method is used to seal an object, by preventing new properties from being added to it and marking all existing properties as non-configurable. But values of present properties can still be changed as long as they are writable. Let's see the below example to understand more about seal() method

```
const object = {
  property: "Welcome JS world",
};
Object.seal(object);
object.property = "Welcome to object world";
console.log(Object.isSealed(object)); // true
delete object.property; // You cannot delete when sealed
console.log(object.property); //Welcome to object world
```

⬆ Back to Top

196. What are the applications of seal method

Below are the main applications of Object.seal() method,

i. It is used for sealing objects and arrays.
ii. It is used to make an object immutable.

⬆ Back to Top

197. What are the differences between freeze and seal methods

If an object is frozen using the Object.freeze() method then its properties become immutable and no changes can be made in them whereas if an object is sealed using the Object.seal() method then the changes can be made in the existing properties of the object.

⬆ Back to Top

198. How do you determine if an object is sealed or not

The Object.isSealed() method is used to determine if an object is sealed or not. An object is sealed if all of the below conditions hold true

i. If it is not extensible.
ii. If all of its properties are non-configurable.
iii. If it is not removable (but not necessarily non-writable). Let's see it in the action

```
const object = {
  property: "Hello, Good morning",
};

Object.seal(object); // Using seal() method to seal the object

console.log(Object.isSealed(object)); // checking whether the object is sealed o
```

⬆ Back to Top

199. How do you get enumerable key and value pairs

The Object.entries() method is used to return an array of a given object's own enumerable string-keyed property [key, value] pairs, in the same order as that provided by a for...in loop. Let's see the functionality of object.entries() method in an example,

```
const object = {
  a: "Good morning",
```

```
  b: 100,
};

for (let [key, value] of Object.entries(object)) {
  console.log(`${key}: ${value}`); // a: 'Good morning'
  // b: 100
}
```

Note: The order is not guaranteed as object defined.

⬆ Back to Top

200. What is the main difference between Object.values and Object.entries method

The Object.values() method's behavior is similar to Object.entries() method but it returns an array of values instead [key,value] pairs.

```
const object = {
  a: "Good morning",
  b: 100,
};

for (let value of Object.values(object)) {
  console.log(`${value}`); // 'Good morning'
  100;
}
```

⬆ Back to Top

201. How can you get the list of keys of any object

You can use the `Object.keys()` method which is used to return an array of a given object's own property names, in the same order as we get with a normal loop. For example, you can get the keys of a user object,

```
const user = {
  name: "John",
  gender: "male",
  age: 40,
};

console.log(Object.keys(user)); //['name', 'gender', 'age']
```

⬆ Back to Top

202. How do you create an object with prototype

The Object.create() method is used to create a new object with the specified prototype object and properties. i.e, It uses an existing object as the prototype of the newly created object. It returns a new object with the specified prototype object and properties.

```
const user = {
  name: "John",
  printInfo: function () {
    console.log(`My name is ${this.name}.`);
  },
};

const admin = Object.create(user);

admin.name = "Nick"; // Remember that "name" is a property set on "admin" but nc

admin.printInfo(); // My name is Nick
```

⬆ Back to Top

203. What is a WeakSet

WeakSet is used to store a collection of weakly(weak references) held objects. The syntax would be as follows,

```
new WeakSet([iterable]);
```

Let's see the below example to explain it's behavior,

```
var ws = new WeakSet();
var user = {};
ws.add(user);
ws.has(user); // true
ws.delete(user); // removes user from the set
ws.has(user); // false, user has been removed
```

⬆ Back to Top

204. What are the differences between WeakSet and Set

The main difference is that references to objects in Set are strong while references to objects in WeakSet are weak. i.e, An object in WeakSet can be garbage collected if there is no other reference to it. Other differences are,

i. Sets can store any value Whereas WeakSets can store only collections of objects
ii. WeakSet does not have size property unlike Set
iii. WeakSet does not have methods such as clear, keys, values, entries, forEach.
iv. WeakSet is not iterable.

↑ Back to Top

205. List down the collection of methods available on WeakSet

Below are the list of methods available on WeakSet,

i. add(value): A new object is appended with the given value to the weakset
ii. delete(value): Deletes the value from the WeakSet collection.
iii. has(value): It returns true if the value is present in the WeakSet Collection, otherwise it returns false.

Let's see the functionality of all the above methods in an example,

```
var weakSetObject = new WeakSet();
var firstObject = {};
var secondObject = {};
// add(value)
weakSetObject.add(firstObject);
weakSetObject.add(secondObject);
console.log(weakSetObject.has(firstObject)); //true
weakSetObject.delete(secondObject);
```

↑ Back to Top

206. What is a WeakMap

The WeakMap object is a collection of key/value pairs in which the keys are weakly referenced. In this case, keys must be objects and the values can be arbitrary values. The syntax is looking like as below,

```
new WeakMap([iterable]);
```

Let's see the below example to explain it's behavior,

```
var ws = new WeakMap();
var user = {};
ws.set(user);
ws.has(user); // true
ws.delete(user); // removes user from the map
ws.has(user); // false, user has been removed
```

⬆ Back to Top

207. What are the differences between WeakMap and Map

The main difference is that references to key objects in Map are strong while references to key objects in WeakMap are weak. i.e, A key object in WeakMap can be garbage collected if there is no other reference to it. Other differences are,

i. Maps can store any key type Whereas WeakMaps can store only collections of key objects
ii. WeakMap does not have size property unlike Map
iii. WeakMap does not have methods such as clear, keys, values, entries, forEach.
iv. WeakMap is not iterable.

⬆ Back to Top

208. List down the collection of methods available on WeakMap

Below are the list of methods available on WeakMap,

i. set(key, value): Sets the value for the key in the WeakMap object. Returns the WeakMap object.
ii. delete(key): Removes any value associated to the key.
iii. has(key): Returns a Boolean asserting whether a value has been associated to the key in the WeakMap object or not.
iv. get(key): Returns the value associated to the key, or undefined if there is none. Let's see the functionality of all the above methods in an example,

```
var weakMapObject = new WeakMap();
var firstObject = {};
var secondObject = {};
// set(key, value)
weakMapObject.set(firstObject, "John");
```