For More Details Call +91 8368 979712, 63804 86914
EmailID–a1projecttraining@gmail.com

# JavaScript Interview Questions & Answers

| 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 array slice method? |
| 6 | What is the purpose of 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? |
| 15 | What is currying function? |
| 16 | What is a pure function? |
| 17 | What is the purpose of let keyword? |
| 18 | What is the difference between let and var? |

| No. | Questions |
|---|---|
| 19 | What is the reason to choose the name let as keyword? |
| 20 | How do you redeclare variables in switch block without an error? |
| 21 | What is Temporal Dead Zone? |
| 22 | What is IIFE(Immediately Invoked Function Expression)? |
| 23 | What is the benefit of using modules? |
| 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 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? |

| No. | Questions |
|---|---|
| 38 | Why do you need a Cookie? |
| 39 | What are the options in a cookie? |
| 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? |

| No. | Questions |
|---|---|
| 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 |
| 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? |

| No. | Questions |
|---|---|
| 76 | What is the difference between window and document? |
| 77 | How do you access history in javascript? |
| 78 | What are the javascript data types? |
| 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? |
| 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? |

| No. | Questions |
|---|---|
| 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? |

| No. | Questions |
|---|---|
| 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? |
| 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? |

| No. | Questions |
|---|---|
| 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? |
| 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? |

| No. | Questions |
|---|---|
| 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? |
| 189 | How do you determine whether object is frozen or not? |

| No. | Questions |
|---|---|
| 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? |

| No. | Questions |
|---|---|
| 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? |
| 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? |

| No. | Questions |
|---|---|
| 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 | What is an event loop? |
| 236 | What is call stack? |
| 237 | What is an event queue? |
| 238 | What is a decorator? |
| 239 | What are the properties of Intl object? |
| 240 | What is an Unary operator? |
| 241 | How do you sort elements in an array? |
| 242 | What is the purpose of compareFunction while sorting arrays? |
| 243 | How do you reversing an array? |
| 244 | How do you find min and max value in an array? |
| 245 | How do you find min and max values without Math functions? |
| 246 | What is an empty statement and purpose of it? |

| No. | Questions |
|---|---|
| 247 | How do you get meta data of a module? |
| 248 | What is a comma operator? |
| 249 | What is the advantage of a comma operator? |
| 250 | What is typescript? |
| 251 | What are the differences between javascript and typescript? |
| 252 | What are the advantages of typescript over javascript? |
| 253 | What is an object initializer? |
| 254 | What is a constructor method? |
| 255 | What happens if you write constructor more than once in a class? |
| 256 | How do you call the constructor of a parent class? |
| 257 | How do you get the prototype of an object? |
| 258 | What happens If I pass string type for getPrototype method? |
| 259 | How do you set prototype of one object to another? |
| 260 | How do you check whether an object can be extendable or not? |
| 261 | How do you prevent an object to extend? |
| 262 | What are the different ways to make an object non-extensible? |
| 263 | How do you define multiple properties on an object? |
| 264 | What is MEAN in javascript? |
| 265 | What Is Obfuscation in javascript? |

| No. | Questions |
|---|---|
| 266 | Why do you need Obfuscation? |
| 267 | What is Minification? |
| 268 | What are the advantages of minification? |
| 269 | What are the differences between Obfuscation and Encryption? |
| 270 | What are the common tools used for minification? |
| 271 | How do you perform form validation using javascript? |
| 272 | How do you perform form validation without javascript? |
| 273 | What are the DOM methods available for constraint validation? |
| 274 | What are the available constraint validation DOM properties? |
| 275 | What are the list of validity properties? |
| 276 | Give an example usage of rangeOverflow property? |
| 277 | Is enums feature available in javascript? |
| 278 | What is an enum? |
| 279 | How do you list all properties of an object? |
| 280 | How do you get property descriptors of an object? |
| 281 | What are the attributes provided by a property descriptor? |
| 282 | How do you extend classes? |
| 283 | How do I modify the url without reloading the page? |
| 284 | How do you check whether an array includes a particular value or not? |

| No. | Questions |
|---|---|
| 285 | How do you compare scalar arrays? |
| 286 | How to get the value from get parameters? |
| 287 | How do you print numbers with commas as thousand separators? |
| 288 | What is the difference between java and javascript? |
| 289 | Is javascript supports namespace? |
| 290 | How do you declare namespace? |
| 291 | How do you invoke javascript code in an iframe from parent page? |
| 292 | How do get the timezone offset from date? |
| 293 | How do you load CSS and JS files dynamically? |
| 294 | What are the different methods to find HTML elements in DOM? |
| 295 | What is jQuery? |
| 296 | What is V8 JavaScript engine? |
| 297 | Why do we call javascript as dynamic language? |
| 298 | What is a void operator? |
| 299 | How to set the cursor to wait? |
| 300 | How do you create an infinite loop? |
| 301 | Why do you need to avoid with statement? |
| 302 | What is the output of below for loops? |
| 303 | List down some of the features of ES6? |

| No. | Questions |
|---|---|
| 304 | What is ES6? |
| 305 | Can I redeclare let and const variables? |
| 306 | Is const variable makes the value immutable? |
| 307 | What are default parameters? |
| 308 | What are template literals? |
| 309 | How do you write multi-line strings in template literals? |
| 310 | What are nesting templates? |
| 311 | What are tagged templates? |
| 312 | What are raw strings? |
| 313 | What is destructuring assignment? |
| 314 | What are default values in destructuring assignment? |
| 315 | How do you swap variables in destructuring assignment? |
| 316 | What are enhanced object literals? |
| 317 | What are dynamic imports? |
| 318 | What are the use cases for dynamic imports? |
| 319 | What are typed arrays? |
| 320 | What are the advantages of module loaders? |
| 321 | What is collation? |
| 322 | What is for...of statement? |

| No. | Questions |
|---|---|
| 323 | What is the output of below spread operator array? |
| 324 | Is PostMessage secure? |
| 325 | What are the problems with postmessage target origin as wildcard? |
| 326 | How do you avoid receiving postMessages from attackers? |
| 327 | Can I avoid using postMessages completely? |
| 328 | Is postMessages synchronous? |
| 329 | What paradigm is Javascript? |
| 330 | What is the difference between internal and external javascript? |
| 331 | Is JavaScript faster than server side script? |
| 332 | How do you get the status of a checkbox? |
| 333 | What is the purpose of double tilde operator? |
| 334 | How do you convert character to ASCII code? |
| 335 | What is ArrayBuffer? |
| 336 | What is the output of below string expression? |
| 337 | What is the purpose of Error object? |
| 338 | What is the purpose of EvalError object? |
| 339 | What are the list of cases error thrown from non-strict mode to strict mode? |
| 340 | Is all objects have prototypes? |
| 341 | What is the difference between a parameter and an argument? |

| No. | Questions |
|---|---|
| 342 | What is the purpose of some method in arrays? |
| 343 | How do you combine two or more arrays? |

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

There are many ways to create objects in javascript as below,

1. **Object constructor:**

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

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

2. **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);
```

3. **Object literal syntax:** The object literal syntax is equivalent to create method when it passes null as parameter

```
var object = {};
```

4. **Function constructor:** Create any function and apply the new operator to create object instances,

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

5. **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 new instance.
console.log(result && typeof result === 'object' ? result : newInstance);
```

6. **ES6 Class syntax:** ES6 introduces class feature to create the objects

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

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

7. **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";
}
```

## 2. What is 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.

## 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 you?
invite.call(employee2, 'Hello', 'How are you?'); // Hello Jimmy Baily, How are you?
```

**Apply:** Invokes the function 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 are you?
invite.apply(employee2, ['Hello', 'How are you?']); // Hello Jimmy Baily, How are you?
```

**bind:** returns a new function, allowing you to pass in an array and 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().

## 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)
```

## 5. What is the purpose of 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 new array.

## 6. What is the purpose of 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 option 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]; original
array: [3, 4, 5]
let arrayIntegers2 = arrayIntegersOriginal2.splice(3); // returns [4, 5]; original array:
[1, 2, 3]
let arrayIntegers3 = arrayIntegersOriginal3.splice(3, 1, "a", "b", "c"); //returns [4];
original array: [1, 2, 3, "a", "b", "c", 5]
```

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

## 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 ar |

## 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.

1. The keys of an Object are Strings and Symbols, whereas they can be any value for a Map, including functions, objects, and any primitive.
2. 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.
3. 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.
4. 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.

5. 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.

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

## 7. What is the difference between == and === operators?

JavaScript provides both strict(===, !==) and type-converting(==, !=) equality comparison. The strict operators takes 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,

1. Two strings are strictly equal when they have the same sequence of characters, same length, and same characters in corresponding positions.
2. Two numbers are strictly equal when they are numerically equal. i.e, Having the same number value. There are two special cases in this,
   i. NaN is not equal to anything, including NaN.
   ii. Positive and negative zeros are equal to one another.
3. Two Boolean operands are strictly equal if both are true or both are false.
4. Two objects are strictly equal if they refer to the same Object.
5. 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
```

## 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 function are best suited for non-method functions, and they cannot be used as constructors.

## 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);
```

## 12. What is a first order function?

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

```
const firstOrder = () => console.log ('Iam a first order functionn!');
```

## 13. What is a higher order function?

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

```
const firstOrderFunc = () => console.log ('Hello I'am a First order function');
const higherOrder = ReturnFirstOrderFunc => ReturnFirstOrderFunc ();
higherOrder (firstOrderFunc);
```

## 14. What is a unary function?

Unary function (i.e. monadic) is a function that accepts exactly one argument. Let us take an example of unary function. It stands for single argument accepted by a function.

```
const unaryFunction = a => console.log (a + 10); //Add 10 to the given argument and
display the value
```

## 15. What is 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;
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 re-usability and functional composition.

## 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 6
console.log (numberArray); // returns [6]
console.log (pureAddNumber (7) (numberArray)); // returns [6, 7]
console.log (numberArray); // returns [6]
```

As per above code snippets, Push function is impure itself by altering the array and returning an push number index which is independent of 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 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 makes 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.

## 17. What is the purpose of 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 if block variable won't exist here)
```

## 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); // error: age is not defined
     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)
}
```

## 19. What is the reason to choose the name let as 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.

## 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 will 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;
      }
    }
```

## 21. What is 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;
}
```

## 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
```

## 23. What is the benefit of using modules?

There are a lot of benefits to using modules in favour of a sprawling. Some of the benefits are,

i. Maintainablity
ii. Reusability
iii. Namespacing

## 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 starts
with the number which is not valid JavaScript  identifier. Hence, can only be
accessed using the square bracket notation.
  }
  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
```

## 25. What is Hoisting?

Hoisting is a JavaScript mechanism where variables and function declarations 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';
```

## 26. What are classes in ES6?

In ES6, Javascript classes are primarily syntactical 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;
  }
}
```

## 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(greetingInfo) has access to the variables in the outer function scope(Welcome) even after outer function has returned.

## 28. What are modules?

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

## 29. Why do you need modules?

Below are the list of benefits using modules in javascript ecosystem

i. Maintainablity
ii. Reusability
iii. Namespacing

## 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.

## 31. What is a service worker?

A Service worker is basically a script (JavaScript file) that runs in background, separate from a web page and provide 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.

## 32. How do you manipulate DOM using 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.

## 33. How do you reuse information across service worker restarts?

The problem with service worker is that it get 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.

## 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.

## 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.

## 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).

## 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";
```

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

## 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";
```

ii. 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";
```

## 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.

## 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 |

## 42. What is the main difference between localStorage and sessionStorage?

LocalStorage is 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.

## 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');
```

## 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();
```

## 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 + '.');
};
```

## 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 recommended approach than Cookies.

## 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..
}
```

## 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..
}
```

## 49. Give an example of web worker?

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

ii. 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 2. 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;
};
```

iii. 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 terminate() method to terminate listening the messages.

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

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

```
w = undefined;
```

## 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

iv.Window object

v. Document object

vi. Parent object

## 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 would be as below

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

## 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.

## 53. What are the three states of promise?

Promises have three states:

.**Pending:** This is an initial state of the Promise before an operation begins

i. **Fulfilled:** This state indicates that specified operation was completed.

ii. **Rejected:** This state indicates that the operation did not complete. In this case an error value will be thrown.

## 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);
```

## 55. Why do we need callbacks?

The callbacks are needed because javascript is a 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 first function invoking an API call(simulated by setTimeout) and 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 first function and remaining code block get executed. So callbacks used in a way to make sure that certain code doesn't execute until other code finished execution.

## 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(){
                ....
            });
        });
    });
});
```

## 57. What is 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 is been used in Facebook/Twitter updates, stock price updates, news feeds etc.

## 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>";
  };
}
```

## 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
}
```

## 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|

## 61. What are the main rules of promise?

A promise must follow a specific set of rules,

.A promise is an object that supplies a standard-compliant `.then()` method

i. A pending promise may transition into either fulfilled or rejected state
ii. A fulfilled or rejected promise is settled and it must not transition into any other state.
iii. Once a promise is settled, the value must not change.

## 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
    });

  })

});
```

## 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,

.The initial promise resolves in 1 second,

i. After that `.then` handler is called by logging the result(1) and then return a promise with the value of result * 2.
ii. After that the value passed to the next `.then` handler by logging the result(2) and return a promise with result * 3.
iii. Finally the value passed to the last `.then` handler by logging the result(6) and return a promise with result * 4.

## 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(result) }) .catch(error => console.log(`Error in promises ${error}`))
```

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

## 65. What is the purpose of 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
faster
});
```

## 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.

## 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.

## 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 declare 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
}
```

## 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 or
null
```

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

## 70. What is the purpose of 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"}
```

## 71. What is 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"
```

## 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
```

## 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
```

## 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 |

| Null | Undefined |
|---|---|
| Converted to zero (0) while performing primitive operations | Converted to NaN while performing primitive operations |

## 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
```

## 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, getElementByTagName, createElement etc |

## 77. How do you access history in javascript?

The window.history object contains the browsers 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.

## 78. What are the javascript data types?

Below are the list of javascript data types available

.Number

i. String
ii. Boolean
iii. Object
iv. Undefined

## 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
```

## 80. What are the differences between undeclared and undefined variables?

Below are the major differences between undeclared 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. |

## 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
```

## 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.

## 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")
```

## 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
```

## 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 each of its parent elements first, starting at the top with the global window object. There are two ways of event flow

.Top to Bottom(Event Capturing)

i. Bottom to Top (Event Bubbling)

## 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.

## 87. What is event capturing?

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

## 88. How do you submit a form using JavaScript?

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

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

## 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 avaialble under platform property,

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

## 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).

## 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 considered as host objects. `User objects` are objects defined in the javascript code. For example, User object created for profile information.

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

You can use below tools or techniques for debugging javascript

.Chrome Devtools

i. debugger statement
ii. Good old console.log statement

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

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

.It avoids callback hell which is unreadable

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

**Cons:**

iv.It makes little complex code

v. You need to load a polyfill if ES6 is not supported

## 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 morning
console.log(input.value); // Good evening
```

## 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).

## 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 document that uses href="JavaScript:Void(0);" within an 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>
```

## 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.

## 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.

## 99. Is there any relation between Java and JavaScript?

No, they are entirely two different programming languages and has 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).

## 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,

.Web page has finished loading

i. Input field was changed
ii. 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>
```

## 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.

## 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 hyper link are some common usecases.

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

**Note:** Remember that not all events are cancelable.

## 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>
```

## 104. What are the steps involved in return false usage?

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

.First it stops the browser's default action or behaviour.

i. It prevents the event from propagating the DOM
ii. Stops callback execution and returns immediately when called.

## 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 window. The Browser Object Model is not standardized and can change based on different browsers.

## 106. What is the use of setTimeout?

The setTimeout() method is used to call a function or evaluates 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);
```

## 107. What is the use of setInterval?

The setInterval() method is used to call a function or evaluates 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);
```

## 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.

## 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);
```

## 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.

## 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.

## 112. What are the syntax rules of JSON?

Below are the list of syntax rules of JSON

.The data is in name/value pairs

i. The data is separated by commas
ii. Curly braces hold objects
iii. Square brackets hold arrays

## 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(user);
console.log(userString); //"{"name":"John","age":31}"
```

## 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 javascript object using parse() method.

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

## 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.

## 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.

## 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 clearTimeout() method.

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

```
}

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

## 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 clearInterval() method.

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

}

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

## 119. How do you redirect new page in javascript?

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

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

## 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,

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

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

ii. **Using indexOf:** In an ES5 or older environments, 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 exist in the main string.

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

iii. **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)
```

## 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 client side. Because the javascript can be disabled on the client side.

```
function validateEmail(email) {
    var re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    return re.test(String(email).toLowerCase());
}
```

The above regular expression regular 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 purpose but this solution has issues in FF.

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

## 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,

iii.href - The entire URL

iv. protocol - The protocol of the URL
v. host - The hostname and port of the URL
vi. hostname - The hostname of the URL
vii. port - The port number in the URL
viii. pathname - The path name of the URL
ix. search - The query portion of the URL
x. hash - The anchor portion of the URL

## 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');
```

## 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 two approaches,

.** 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)
```

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

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

## 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 ...
    }
}
```

## 127. How do you test for an empty object?

There are different solutions based on ECMAScript versions

ii.**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 object
length is 0, you need to check constructor check as well
```

ii. **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 object
length is 0, you need to check constructor check as well
```

iii. **Using for-in with hasOwnProperty(Pre-ECMA 5):** You can use 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({});
}
```

## 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
```

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

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

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

## 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**

iii. Works on every environment

iv. You can use break and continue flow control statements **Cons**

v. Too verbose

vi. Imperative

vii. You might face one-by-off errors

## 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);
```

## 132. How do you compare two date objects?

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

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

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

You can use ECMAScript 6's `String.prototype.startsWith()` method to check 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
```

## 134. How do you trim a string in javascript?

JavaScript provided a trim method on string types to trim any whitespaces present at the begining 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, '');
        };
    })();
}
```

## 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
};
```

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

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

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

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

## 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,

ii.A logical not (!)

iii. 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.

## 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'.

## 138. How do you define multiline strings?

You can define multiline string literals using '' 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.

## 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.

## 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
}
```

## 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.
```

## 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.

## 143. What are break and continue statements?

The break statement is used to "jumps 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 "jumps 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>";
}
```

## 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"
```

## 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,

.Gives cleaner code

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

## 146. What are the benefits of initializing variables?

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

.It gives cleaner code

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

## 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.

.Assign {} instead of new Object()

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

## 148. How do you define JSON arrays?

JSON arrays are written inside square brackets and array 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"}
]
```

## 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)

## 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
```

## 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`.

## 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 few MBs, but by tree shaking it can bring down the size to just few hundred KBs. Tree shaking is been implemented in Rollup and Webpack bundlers.

## 153. Is it recommended to use eval?

No, it allows arbitrary code to be run which casues 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.

## 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
```

## 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 return a modified string where the pattern is replaced.

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

## 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 |

| Modifier | Description |
|---|---|
| m | Perform multiline matching |

````
Let's take an example of global modifier,
```javascript
  var text = "Learn JS one by one";
  var pattern = /one/g;
  var result = text.match(pattern); // one,one
```
````

## 157. What are regular expression patterns?

Regular Expressions provided 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

## 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+/
```

## 159. How do you search a string for a pattern?

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

```
var pattern = /you/;
```

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

## 160. What is the purpose of exec method?

The purpose of exec method is similar to test method but it returns a founded text as an object instead of returning true/false.

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

## 161. How do you change 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";
```

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

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

## 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 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.

## 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
}
```

## 164. What is the purpose of breakpoints in debugging?

You can set breakpoints in the javascript code once the debugger statement is executed and 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 play button.

## 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
```

## 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|com
pal|elaine|fennec|hiptop|iemobile|ip(hone|od)|iris|kindle|lge
|maemo|midp|mmp|mobile.+firefox|netfront|opera m(ob|in)i|palm(
os)?|phone|p(ixi|re)\/|plucker|pocket|psp|series(4|6)0|symbian|treo|up\.(browser|li
nk)|vodafone|wap|windows ce|xda|xiino/i.test(a)||/1207|6310|6590|3gso|4thp|50[1-
6]i|770s|802s|a wa|abac|ac(er|oo|s\-
)|ai(ko|rn)|al(av|ca|co)|amoi|an(ex|ny|yw)|aptu|ar(ch|go)|as(te|us)|attw|au(di|\-
m|r |s )|avan|be(ck|ll|nq)|bi(lb|rd)|bl(ac|az)|br(e|v)w|bumb|bw\-
(n|u)|c55\/|capi|ccwa|cdm\-|cell|chtm|cldc|cmd\-
|co(mp|nd)|craw|da(it|ll|ng)|dbte|dc\-s|devi|dica|dmob|do(c|p)o|ds(12|\-
d)|el(49|ai)|em(l2|ul)|er(ic|k0)|esl8|ez([4-7]0|os|wa|ze)|fetc|fly(\-|_)|g1
u|g560|gene|gf\-5|g\-mo|go(\.w|od)|gr(ad|un)|haie|hcit|hd\-(m|p|t)|hei\-
|hi(pt|ta)|hp( i|ip)|hs\-c|ht(c(\-| |_|a|g|p|s|t)|tp)|hu(aw|tc)|i\-
(20|go|ma)|i230|iac( |\-
|\/)|ibro|idea|ig01|ikom|im1k|inno|ipaq|iris|ja(t|v)a|jbro|jemu|jigs|kddi|keji|kgt(
|\/)|klon|kpt |kwc\-|kyo(c|k)|le(no|xi)|lg( g|\/(k|l|u)|50|54|\-[a-
w])|libw|lynx|m1\-w|m3ga|m50\/|ma(te|ui|xo)|mc(01|21|ca)|m\-
cr|me(rc|ri)|mi(o8|oa|ts)|mmef|mo(01|02|bi|de|do|t(\-| |o|v)|zz)|mt(50|p1|v
)|mwbp|mywa|n10[0-2]|n20[2-3]|n30(0|2)|n50(0|2|5)|n7(0(0|1)|10)|ne((c|m)\-
|on|tf|wf|wg|wt)|nok(6|i)|nzph|o2im|op(ti|wv)|oran|owg1|p800|pan(a|d|t)|pdxg|pg(13|
\-([1-8]|c))|phil|pire|pl(ay|uc)|pn\-2|po(ck|rt|se)|prox|psio|pt\-g|qa\-
a|qc(07|12|21|32|60|\-[2-7]|i\-
)|qtek|r380|r600|raks|rim9|ro(ve|zo)|s55\/|sa(ge|ma|mm|ms|ny|va)|sc(01|h\-|oo|p\-
)|sdk\/|se(c(\-|0|1)|47|mc|nd|ri)|sgh\-|shar|sie(\-|m)|sk\-
0|sl(45|id)|sm(al|ar|b3|it|t5)|so(ft|ny)|sp(01|h\-|v\-|v
)|sy(01|mb)|t2(18|50)|t6(00|10|18)|ta(gt|lk)|tcl\-|tdg\-|tel(i|m)|tim\-|t\-
mo|to(pl|sh)|ts(70|m\-|m3|m5)|tx\-
9|up(\.b|g1|si)|utst|v400|v750|veri|vi(rg|te)|vk(40|5[0-3]|\-
v)|vm40|voda|vulc|vx(52|53|60|61|70|80|81|83|85|98)|w3c(\-| )|webc|whit|wi(g
|nc|nw)|wmlb|wonu|x700|yas\-|your|zeto|zte\-/i.test(a.substr(0,4))) mobileCheck =
true;})(navigator.userAgent||navigator.vendor||window.opera);
  return mobileCheck;
};
```

## 167. How do you detect a mobile browser without regexp?

You can detect mobile browser 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;
  }
}
```

## 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';
```

## 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;
}
```

## 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 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);
}
```

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

You can use toLocaleString() method to convert date 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,
09:56:00
```

## 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;
```

## 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 statement.

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

## 174. Can you apply chaining on conditional operator?

Yes, you can apply chaining on conditional operator 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; }
}
```

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

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

ii.** window.onload:**

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

ii. **document.onload:**

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

iii. ** body onload:**

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

## 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;
```

## 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 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.

## 178. What is a freeze method?

The freeze() method is used to freeze an object. Freezing an object does'nt 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.

## 179. What is the purpose of freeze method?

Below are the main benefits of using freeze method,

iii.It is used for freezing objects and arrays.

iv. It is used to make an object immutable.

## 180. Why do I need to use freeze method?

In 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 `final` keyword which is used in various languages.

## 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);
```

## 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
```

## 183. How do you detect javascript disabled in the page?

You can use `<noscript>` tag to detect javascript disabled or not. The code block inside `<noscript>` get executed when JavaScript is disabled, and are 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 apge. Please
click Next Page</a>
</noscript>
```

## 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,

.**Arithmetic Operators:** Includes + (Addition),– (Subtraction), * (Multiplication), / (Division), % (Modulus), + + (Increment) and – – (Decrement)

i. **Comparison Operators:** Includes = =(Equal),!= (Not Equal), ===(Equal with type), > (Greater than),> = (Greater than or Equal to),< (Less than),<= (Less than or Equal to)
ii. **Logical Operators:** Includes &&(Logical AND),||(Logical OR),!(Logical NOT)
iii. **Assignment Operators:** Includes = (Assignment Operator), += (Add and Assignment Operator), – = (Subtract and Assignment Operator), *= (Multiply and Assignment), /= (Divide and Assignment), %= (Modules and Assignment)
iv. **Ternary Operators:** It includes conditional(: ?) Operator
v. **typeof Operator:** It uses to find type of variable. The syntax looks like `typeof variable`

## 185. What is a rest parameter?

Rest parameter is an improved way to handle function parameter 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

## 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;
}
```

## 187. What are the bitwise operators available in javascript?

Below are the list of bit-wise logical operators used in JavaScript

.Bit-wise AND ( & )

i. Bit-Wise OR ( | )
ii. Bit-Wise XOR ( ^ )
iii. Bit-Wise NOT ( ~ )
iv. Left Shift ( << )
v. Sign Propagating Right Shift ( >> )
vi. Zero fill Right Shift ( >>> )

## 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
```

## 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,

.If it is not extensible.

i. If all of its properties are non-configurable.
ii. 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));
```

## 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:

.both undefined

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

## 191. What is the purpose of using object is method?

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

.It is used for comparison of two strings.

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

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

You can use 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 exampple 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: 5 }
```

```
console.log(returnedTarget); // { a: 1, b: 3, c: 5 }
```

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

## 193. What are the applications of assign method?

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

.It is used for cloning an object.

i. It is used to merge object with same properties.

## 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); // 1, 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

## 195. What is the purpose of seal method?

The Object.seal() method is used 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)); // Welcome to object world
delete object.property; // You cannot delete when sealed
console.log(object.property); //Welcome to object world
```

## 196. What are the applications of seal method?

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

.It is used for sealing objects and arrays.

i. It is used to make an object immutable.

## 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.

## 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

.If it is not extensible.

i. If all of its properties are non-configurable.

ii. 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
or not
```

## 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.

## 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
}
```

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

You can use `Object.keys()` method which is used 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']
```

## 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 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(person);

admin.name = "Nick"; // Remember that "name" is a property set on "admin" but not
on "user" object

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

## 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
```

## 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,

.Sets can store any value Whereas WeakSets can store only collections of objects

i. WeakSet does not have size property unlike Set
ii. WeakSet does not have methods such as clear, keys, values, entries, forEach.
iii. WeakSet is not iterable.

## 205. List down the collection of methods available on WeakSet?

Below are the list of methods available on WeakSet,

.add(value): A new object is appended with the given value to the weakset

i. delete(value): Deletes the value from the WeakSet collection.
ii. has(value): It returns true if the value is present in the WeakSet Collection, otherwise it returns false.
iii. length(): It returns the length of weakSetObject 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
console.log(weakSetObject.length()); //2
weakSetObject.delete(secondObject);
```

## 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
```

## 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,

.Maps can store any key type Whereas WeakMaps can store only collections of key objects

i. WeakMap does not have size property unlike Map
ii. WeakMap does not have methods such as clear, keys, values, entries, forEach.
iii. WeakMap is not iterable.

## 208. List down the collection of methods available on WeakMap?

Below are the list of methods available on WeakMap,

.set(key, value): Sets the value for the key in the WeakMap object. Returns the WeakMap object.

i. delete(key): Removes any value associated to the key.
ii. has(key): Returns a Boolean asserting whether a value has been associated to the key in the WeakMap object or not.
iii. 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');
weakMapObject.set(secondObject, 100);
console.log(weakMapObject.has(firstObject)); //true
console.log(weakMapObject.get(firstObject)); // John
weakMapObject.delete(secondObject);
```

## 209. What is the purpose of uneval?

The uneval() is an inbuilt function which is used to create a string representation of the source code of an Object. It is a top-level function and is not associated with any object. Let's see the below example to know more about it's functionality,

```
var a = 1;
uneval(a); // returns a String containing 1
uneval(function user() {}); // returns "(function user(){})"
```

## 210. How do you encode an URL?

The encodeURI() function is used to encode complete URI which has special characters except (, / ? : @ & = + $ #) characters.

```
var uri = 'https://mozilla.org/?x=шеллы';
var encoded = encodeURI(uri);
console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
```

## 211. How do you decode an URL?

The decodeURI() function is used to decode a Uniform Resource Identifier (URI) previously created by encodeURI().

```
 var uri = 'https://mozilla.org/?x=шеллы';
 var encoded = encodeURI(uri);
 console.log(encoded); // https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B
try {
  console.log(decodeURI(encoded)); // "https://mozilla.org/?x=шеллы"
} catch(e) { // catches a malformed URI
  console.error(e);
}
```

## 212. How do you print the contents of web page?

The window object provided print() method which is used to prints the contents of the current window. It opens Print dialog box which lets you choose between various printing options. Let's see the usage of print method in an example,

```
   <input type="button" value="Print" onclick="window.print()" />
```

**Note:** In most browsers, it will block while the print dialog is open.

## 213. What is the difference between uneval and eval?

The `uneval` function returns the source of a given object; whereas the `eval` function does the opposite, by evaluating that source code in a different memory area. Let's see an example to clarify the difference,

```
var msg = uneval(function greeting() { return 'Hello, Good morning'; });
var greeting = eval(msg);
greeting(); // returns "Hello, Good morning"
```

## 214. What is an anonymous function?

An anonymous function is a function without a name! Anonymous functions are commonly assigned to a variable name or used as a callback function. The syntax would be as below,

```
function (optionalParameters) {
  //do something
}

const myFunction = function(){ //Anonymous function assigned to a variable
  //do something
};
```

```
[1, 2, 3].map(function(element){ //Anonymous function used as a callback function
  //do something
});
```

Let's see the above anonymous function in an example,

```
var x = function (a, b) {return a * b};
var z = x(5, 10);
console.log(z); // 50
```

## 215. What is the precedence order between local and global variables?

A local variable takes precedence over a global variable with the same name. Let's see this behavior in an example.

```
var msg = "Good morning";
function greeting() {
   msg = "Good Evening";
   console.log(msg);
}
greeting();
```

## 216. What are javascript accessors?

ECMAScript 5 introduced javascript object accessors or computed properties through getters and setters. Getters uses `get` keyword whereas Setters uses `set` keyword.

```
var user = {
  firstName: "John",
  lastName : "Abraham",
  language : "en",
  get lang() {
    return this.language;
  }
  set lang(lang) {
  this.language = lang;
  }
};
console.log(user.lang); // getter access lang as en
user.lang = 'fr';
console.log(user.lang); // setter used to set lang as fr
```

## 217. How do you define property on Object constructor?

The Object.defineProperty() static method is used to define a new property directly on an object, or modifies an existing property on an object, and returns the object. Let's see an example to know how to define property,

```
const newObject = {};

Object.defineProperty(newObject, 'newProperty', {
  value: 100,
  writable: false
});
```

```
console.log(newObject.newProperty); // 100

newObject.newProperty = 200; // It throws an error in strict mode due to writable
setting
```

## 218. What is the difference between get and defineProperty?

Both has similar results until unless you use classes. If you use `get` the property will be defined on the prototype of the object whereas using `Object.defineProperty()` the property will be defined on the instance it is applied to.

## 219. What are the advantages of Getters and Setters?

Below are the list of benefits of Getters and Setters,

.They provide simpler syntax

i. They are used for defining computed properties, or accessors in JS.
ii. Useful to provide equivalence relation between properties and methods
iii. They can provide better data quality
iv. Useful for doing things behind the scenes with the encapsulated logic.

## 220. Can I add getters and setters using defineProperty method?

Yes, You can use `Object.defineProperty()` method to add Getters and Setters. For example, the below counter object uses increment, decrement, add and substract properties,

```
var counterObj = {counter : 0};

// Define getters
Object.defineProperty(obj, "increment", {
  get : function () {this.counter++;}
});
Object.defineProperty(obj, "decrement", {
  get : function () {this.counter--;}
});

// Define setters
Object.defineProperty(obj, "add", {
  set : function (value) {this.counter += value;}
});
Object.defineProperty(obj, "subtract", {
  set : function (value) {this.counter -= value;}
});

obj.add = 10;
obj.subtract = 5;
console.log(obj.increment); //6
console.log(obj.decrement); //5
```

## 221. What is the purpose of switch-case?

The switch case statement in JavaScript is used for decision making purposes. In few cases, using the switch case statement is going to be more convenient than if-else statements. The syntax would be as below,

```
switch (expression)
{
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    .
    .
    case valueN:
        statementN;
        break;
    default:
        statementDefault;
}
```

The above multi-way branch statement provides an easy way to dispatch execution to different parts of code based on the value of the expression.

## 222. What are the conventions to be followed for the usage of swtich case?

Below are the list of conventions should be taken care,

.The expression can be of type either number or string.

i. Duplicate values are not allowed for the expression.
ii. The default statement is optional. If the expression passed to switch does not matches with any case value then the statement within default case will be executed.
iii. The break statement is used inside the switch to terminate a statement sequence.
iv. The break statement is optional. But if it omitted, the execution will continue on into the next case.

## 223. What are primitive data types?

A primitive data type is data that has a primitive value (which has no properties or methods). There are 5 types of primitive data types.

.string

i. number
ii. boolean
iii. null
iv. undefined

## 224. What are the different ways to access object properties?

There are 3 possible ways for accessing the property of an object.

.**Dot notation:** It uses dot for accessing the properties

```
objectName.property
```

ii. **Square brackets notation:** It uses square brackets for property access

```
objectName["property"]
```

iii. **Expression notation:** It uses expression in the square brackets

```
objectName[expression]
```

## 225. What are the function parameter rules?

JavaScript functions follow below rules for parameters,

iii.The function definitions do not specify data types for parameters.

iv. Do not perform type checking on the passed arguments.

v. Do not check the number of arguments received. i.e, The below function follows the above rules,

```
function functionName(parameter1, parameter2, parameter3) {
  console.log(parameter1); // 1
}
functionName(1);
```

## 226. What is an error object?

An error object is a built in error object that provides error information when an error occurs. It has two properties: name and message. For example, the below function logs error details,

```
try {
  greeting("Welcome");
}
catch(err) {
  console.log(err.name + "<br>" + err.message);
}
```

## 227. When you get a syntax error?

A SyntaxError is thrown if you try to evaluate code with a syntax error. For example, the below missing quote for the function parameter throws a syntax error

```
try {
  eval("greeting('welcome)");   // Missing ' will produce an error
}
catch(err) {
  console.log(err.name);
}
```

## 228. What are the different error names from error object?

There are 6 different types of error names returned from error object, | Error Name | Description | |---- | --------- | EvalError | An error has occurred in the eval() function | | RangeError | An error has occurred with a number "out of range" | | ReferenceError | An error due to an illegal reference| | SyntaxError | An error due to a syntax error| | TypeError | An error due to a type error | | URIError | An error due to encodeURI() |

## 229. What are the various statements in error handling?

Below are the list of statements used in an error handling,

.**try:** This statement is used to test a block of code for errors

i. **catch:** This statement is used to handle the error
ii. **throw:** This statement is used to create custom errors.
iii. **finally:** This statement is used to execute code after try and catch regardless of the result.

## 230. What are the two types of loops in javascript?

.**Entry Controlled loops:** In this kind of loop type, the test condition is tested before entering the loop body. For example, For Loop and While Loop comes under this category.

i. **Exit Controlled Loops:** In this kind of loop typpe, the test condition is tested or evaluated at the end of loop body. i.e, the loop body will execute atleast once irrespective of test condition true or false. For example, do-while loop comes under this category.

## 231. What is nodejs?

Node.js is a server-side platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. It is an event-based, non-blocking, asynchronous I/O runtime that uses Google's V8 JavaScript engine and libuv library.

## 232. What is an Intl object?

The Intl object is the namespace for the ECMAScript Internationalization API, which provides language sensitive string comparison, number formatting, and date and time formatting. It provides an access to several constructors and language sensitive functions.

## 233. How do you perform language specific date and time formatting?

You can use `Intl.DateTimeFormat` object which is constructor for objects that enable language-sensitive date and time formatting. Let's see this behavior with an example,

```
var date = new Date(Date.UTC(2019, 07, 07, 3, 0, 0));
```

```
console.log(new Intl.DateTimeFormat('en-GB').format(date)); // 07/08/2019
console.log(new Intl.DateTimeFormat('en-AU').format(date)); // 07/08/2019
```

## 234. What is an Iterator?

an iterator is an object which defines a sequence and a return value upon its termination. It implements the Iterator protocol with a next() method which returns an object with two properties: value (the next value in the sequence) and done (which is true if the last value in the sequence has been consumed).

## 235. What is an event loop?

The Event Loop is a queue of callback functions. When an async function executes, the callback function is pushed into the queue. The JavaScript engine doesn't start processing the event loop until async function has finished executing the code. **Note:** It allows Node.js to perform non-blocking I/O operations eventhough JavaScript is single-threaded.

## 236. What is call stack?

## 237. What is an event queue?

## 238. What is a decorator?

A decorator is an expression that evaluates to a function and that takes the target, name, and decorator descriptor as arguments. Also, it optionally returns a decorator descriptor to install on the target object. Let's define admin decorator for user class at design time,

```
function admin(isAdmin) {
   return function(target) {
       target.isAdmin = isAdmin;
   }
}

@admin(true)
class User() {
}
console.log(User.isAdmin); //true

 @admin(false)
 class User() {
 }
 console.log(User.isAdmin); //false
```

## 239. What are the properties of Intl object?

Below are the list of properties available on Intl object,

.**Collator:** These are the objects that enable language-sensitive string comparison.

i. **DateTimeFormat:** These are the objects that enable language-sensitive date and time formatting.
ii. **ListFormat:** These are the objects that enable language-sensitive list formatting.
iii. **NumberFormat:** Objects that enable language-sensitive number formatting.
iv. **PluralRules:** Objects that enable plural-sensitive formatting and language-specific rules for plurals.
v. **RelativeTimeFormat:** Objects that enable language-sensitive relative time formatting.

## 240. What is an Unary operator?

The unary(+) operator is used to convert a variable to a number.If the variable cannot be converted, it will still become a number but with the value NaN. Let's see this behavior in an action.

```
var x = "100";
var y = + x;
console.log(typeof x, typeof y); // string, number

var a = "Hello";
var b = + a;
console.log(typeof a, typeof b, b); // string, number, NaN
```

## 241. How do you sort elements in an array?

The sort() method is used to sort the elements of an array in place and returns the sorted array. The example usage would be as below,

```
var months = ["Aug", "Sep", "Jan", "June"];
months.sort();
console.log(months); //  ["Aug", "Jan", "June", "Sep"]
```

## 242. What is the purpose of compareFunction while sorting arrays?

The compareFunction is used to define the sort order. If omitted, the array elements are converted to strings, then sorted according to each character's Unicode code point value. Let's take an example to see the usage of compareFunction,

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
console.log(numbers); // [5, 4, 3, 2, 1]
```

## 243. How do you reversing an array?

You can use reverse() method is used reverse the elements in an array. This method is useful to sort an array in descending order. Let's see the usage of reverse() method in an example,

```
let numbers = [1, 2, 5, 3, 4];
numbers.sort((a, b) => b - a);
numbers.reverse();
console.log(numbers); // [1, 2, 3, 4 ,5]
```

## 244. How do you find min and max value in an array?

You can use `Math.min` and `Math.max` methods on array variable to find the minimum and maximum elements with in an array. Let's create two functions to find the min and max value with in an array,

```
var marks = [50, 20, 70, 60, 45, 30];
function findMin(arr) {
  return Math.min.apply(null, arr);
}
function findMax(arr) {
  return Math.max.apply(null, arr);
}

console.log(findMin(marks));
console.log(findMax(marks));
```

## 245. How do you find min and max values without Math functions?

You can write functions which loops through an array comparing each value with the lowest value or highest value to find the min and max values. Let's create those functions to find min an max values,

```
var marks = [50, 20, 70, 60, 45, 30];
function findMin(arr) {
  var length = arr.length
  var min = Infinity;
  while (length--) {
    if (arr[length] < min) {
      min = arr[len];
    }
  }
  return min;
}

function findMax(arr) {
  var length = arr.length
  var max = -Infinity;
  while (len--) {
    if (arr[length] > max) {
      max = arr[length];
    }
  }
  return max;
}

console.log(findMin(marks));
console.log(findMax(marks));
```

## 246. What is an empty statement and purpose of it?

The empty statement is a semicolon (;) indicating that no statement will be executed, even if JavaScript syntax requires one. Since there is no action with an empty statement you might think that it's usage is quite less, but the empty statement is occasionally useful when you want to create a loop that has an empty body. For example, you can initialize an array with zero values as below,

```
// Initialize an array a
for(int i=0; i < a.length; a[i++] = 0) ;
```

## 247. How do you get meta data of a module?

You can use `import.meta` object which is a meta-property exposing context-specific meta data to a JavaScript module. It contains information about the current module, such as module's URL. In browser, you might get different meta data than NodeJS.

```
<script type="module" src="welcome-module.js"></script>
console.log(import.meta); // { url: "file:///home/user/welcome-module.js" }
```

## 248. What is a comma operator?

The comma operator is used to evaluate each of its operands from left to right and returns the value of the last operand. This is totally different from comma usage within arrays, objects, and function arguments and parameters. For example, the usage for numeric expressions would be as below,

```
var x = 1;
x = (x++, x);

console.log(x); // 2
```

## 249. What is the advantage of a comma operator?

It is normally used to include multiple expressions in a location that requires a single expression. One of the common usage of this comma operator is to supply multiple parameters in a `for` loop. For example, the below for loop uses multiple expressions in a single location using comma operator,

```
for (var a = 0, b =10; a <= 10; a++, b--)
```

You can also use the comma operator in a return statement where it process before returning.

```
function myFunction() {
   var a = 1;
   return (a += 10, a); // 11
}
```

## 250. What is typescript?

```
TypeScript is a typed superset of JavaScript created by Microsoft that adds optional
types, classes, async/await, and many other features, and compiles to plain JavaScript.
Angular built entirely in TypeScript and used as a primary language.
You can install it globally as
```
npm install -g typescript
```
Let's see a simple example of TypeScript usage,
```typescript
function greeting(person: string) {
    return "Hello, " + person;
}
```
```

```
let user = "Sudheer";

document.body.innerHTML = greeting(user);
```
The greeting method allows only string type as argument.

## 251. What are the differences between javascript and typescript?

Below are the list of differences between javascript and typescript,

| feature | typescript | javascript |
|---|---|---|
| Language paradigm | Object oriented programming language | Scripting language |
| Typing support | Supports static typing | It has dynamic typing |
| Modules | Supported | Not supported |
| Interface | It has interfaces concept | Doesn't support interfaces |
| Optional parameters | Functions support optional parameters | No support of optional parameters for functions |

## 252. What are the advantages of typescript over javascript?

Below are some of the advantages of typescript over javascript,

i. TypeScript is able to find compile time errors at the development time only and it make sures less runtime errors. Whereas javascript is interpreted language.
ii. TypeScript is is strongly-typed or supports static typing which allows for checking type correctness at compile time. This is not available in javascript.
iii. TypeScript compiler can compile the .ts files into ES3,ES4 and ES5 unlike ES6 features of javascript which may not be supported in some browsers.

## 253. What is an object initializer?

An object initializer is an expression that describes the initialization of an Object. The syntax for this expression is represented as a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}). This is also known as literal notation. It is one of the ways to create an object.

```
var initObject = {a: 'John', b: 50, c: {}};

console.log(initObject.a); // John
```

## 254. What is a constructor method?

The constructor method is a special method for creating and initializing an object created within a class. If you do not specify a constructor method, a default constructor is used. The example usage of constructor would be as below,

```
class Employee {
  constructor() {
    this.name = "John";
  }
}

var employeeObject = new Employee();

console.log(employeeObject.name); // John
```

## 255. What happens if you write constructor more than once in a class?

The "constructor" in a class is a special method and it should be defined only once in a class. i.e, If you write a constructor method more than once in a class it will throw a `SyntaxError` error.

```
 class Employee {
   constructor() {
     this.name = "John";
   }
   constructor() {   //  Uncaught SyntaxError: A class may only have one
constructor
     this.age = 30;
   }
 }

 var employeeObject = new Employee();

 console.log(employeeObject.name);
```

## 256. How do you call the constructor of a parent class?

You can use `super` keyword to call the constructor of a parent class. Remember that `super()` must be called before using 'this' reference. Otherwise it will cause a reference error. Let's the usage of it,

```
class Square extends Rectangle {
  constructor(length) {
    super(length, length);
    this.name = 'Square';
  }

  get area() {
    return this.width * this.height;
  }

  set area(value) {
    this.area = value;
  }
}
```

## 257. How do you get the prototype of an object?

You can use `Object.getPrototypeOf(obj)` method is used to return the prototype of the specified object. i.e. The value of the internal `prototype` property. If there are no inherited properties then `null` value is returned.

```
const newPrototype = {};
const newObject = Object.create(newPrototype);

console.log(Object.getPrototypeOf(newObject) === newPrototype); // true
```

## 258. What happens If I pass string type for getPrototype method?

In ES5, it will throw a TypeError exception if the obj parameter isn't an object. Whereas in ES2015, the parameter will be coerced to an `Object`.

```
// ES5
Object.getPrototypeOf('James'); // TypeError: "James" is not an object
// ES2015
Object.getPrototypeOf('James'); // String.prototype
```

## 259. How do you set prototype of one object to another?

You can use `Object.setPrototypeOf()` method that sets the prototype (i.e., the internal `Prototype` property) of a specified object to another object or null. For example, if you want to set prototype of a square object to rectangle object would be as follows,

```
Object.setPrototypeOf(Square.prototype, Rectangle.prototype);
Object.setPrototypeOf({}, null);
```

## 260. How do you check whether an object can be extendable or not?

The `Object.isExtensible()` method is used to determine if an object is extensible or not. i.e, Whether it can have new properties added to it or not.

```
const newObject = {};
console.log(Object.isExtensible(newObject)); //true
```

**Note:** By default, all the objects are extendable. i.e, The new properties can added or modified.

## 261. How do you prevent an object to extend?

The `Object.preventExtensions()` method is used to prevent new properties from ever being added to an object. In other words, it prevents future extensions to the object. Let's see the usage of this property,

```
const newObject = {};
Object.preventExtensions(newObject); // NOT extendable

try {
  Object.defineProperty(newObject, 'newProperty', { // Adding new property
    value: 100
  });
} catch (e) {
```

```
  console.log(e); // TypeError: Cannot define property newProperty, object is not
extensible
}
```

## 262. What are the different ways to make an object non-extensible?

You can mark an object non-extensible in 3 ways,

i. Object.preventExtensions
ii. Object.seal
iii. Object.freeze

```
var newObject = {};

Object.preventExtensions(newObject); // Prevent objects are non-extensible
Object.isExtensible(newObject); // false

var sealedObject = Object.seal({}); // Sealed objects are non-extensible
Object.isExtensible(sealedObject); // false

var frozenObject = Object.freeze({}); // Frozen objects are non-extensible
Object.isExtensible(frozenObject); // false
```

## 263. How do you define multiple properties on an object?

The `Object.defineProperties()` method is used to define new or modifies existing properties directly on an object and returning the object. Let's define multiple properties on an empty object,

```
const newObject = {};

Object.defineProperties(newObject, {
  newProperty1: {
    value: 'John',
    writable: true
  },
  newProperty2: {}
});
```

## 264. What is MEAN in javascript?

The MEAN (MongoDB, Express, AngularJS, and Node.js) stack is the most popular open-source JavaScript software tech stack available for building dynamic web apps where you can write both the server-side and client-side halves of the web project entirely in JavaScript.

## 265. What Is Obfuscation in javascript?

Obfuscation is the deliberate act of creating obfuscated javascript code(i.e, source or machine code) that is difficult for humans to understand. It is something similar to encryption, but a machine can understand the code and execute it. Let's see the below function before Obfuscation,

```
function greeeting() {
   console.log('Hello, welcome to JS world');
}
```

And after the code Obfuscation, it would be appeared as below,

```
eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return
d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('2 1(){0.3(\'4, 7 6 5
8\')}',9,9,'console|greeeting|function|log|Hello|JS|to|welcome|world'.split('|'),0,
{}))
```

## 266. Why do you need Obfuscation?

Below are the few reasons for Obfuscation,

i. The Code size will be reduced. So data transfers between server and client will be fast.
ii. It hides the business logic from outside world and protects the code from others
iii. Reverse engineering is highly difficult
iv. The download time will be reduced

## 267. What is Minification?

Minification is the process of removing all unnecessary characters(empty spaces are removed) and variables will be renamed without changing it's functionality. It is also a type of obfuscation .

## 268. What are the advantages of minification?

Normally it is recommend to use minification for heavy traffic and intensive requirements of resources. It reduces file sizes with below benefits,

i. Decreases loading times of a web page
ii. Saves bandwidth usages

## 269. What are the differences between Obfuscation and Encryption?

Below are the main differences between Obfuscation and Encryption,

| Feature | Obfuscation | Encryption |
|---|---|---|
| Definition | Changing the form of any data in any other form | Changing the form of information to an unreadable format by using a key |
| A key to decode | It can be decoded without any key | It is required |

| Feature | Obfuscation | Encryption |
|---|---|---|
| Target data format | It will be converted to a complex form | Converted into an unreadable format |

## 270. What are the common tools used for minification?

There are many online/offline tools to minify the javascript files,

i. Google's Closure Compiler
ii. UglifyJS2
iii. jsmin
iv. javascript-minifier.com/
v. prettydiff.com

## 271. How do you perform form validation using javascript?

JavaScript can be used to perform HTML form validation. For example, if form field is empty, the function needs to notify, and return false, to prevent the form being submitted. Lets' perform user login in an html form,

```
<form name="myForm" onsubmit="return validateForm()" method="post">
User name: <input type="text" name="uname">
<input type="submit" value="Submit">
</form>
```

And the validation on user login is below,

```
function validateForm() {
  var x = document.forms["myForm"]["uname"].value;
  if (x == "") {
    alert("The username shouldn't be empty");
    return false;
  }
}
```

## 272. How do you perform form validation without javascript?

You can perform HTML form validation automatically without using javascript. The validation enabled by applying `required` attribute to prevent form submission when the input is empty.

```
<form method="post">
  <input type="text" name="uname" required>
  <input type="submit" value="Submit">
</form>
```

**Note:** Automatic form validation does not work in Internet Explorer 9 or earlier.

## 273. What are the DOM methods available for constraint validation?

The below DOM methods are available for constraint validation on an invalid input,

i. checkValidity(): It returns true if an input element contains valid data.
ii. setCustomValidity(): It is used to set the validationMessage property of an input element. Let's take an user login form with DOM validations

```
function myFunction() {
  var userName = document.getElementById("uname");
  if (!userName.checkValidity()) {
    document.getElementById("message").innerHTML = userName.validationMessage;
  } else {
    document.getElementById("message").innerHTML = "Entered a valid username";
  }
}
```

## 274. What are the available constraint validation DOM properties?

Below are the list of some of the constraint validation DOM properties available,

i. validity: It provides list of boolean properties related to the validity of an input element.
ii. validationMessage: It displays the message when the validity is false.
iii. willValidate: It indicates if an input element will be validated or not.

## 275. What are the list of validity properties?

The validity property of an input element provides a set of properties related to the validity of data.

i. customError: It returns true, if a custom validity message is set.
ii. patternMismatch: It returns true, if an element's value does not match its pattern attribute.
iii. rangeOverflow: It returns true, if an element's value is greater than its max attribute.
iv. rangeUnderflow: It returns true, if an element's value is less than its min attribute.
v. stepMismatch: It returns true, if an element's value is invalid according to step attribute.
vi. tooLong: It returns true, if an element's value exceeds its maxLength attribute.
vii. typeMismatch: It returns true, if an element's value is invalid according to type attribute.
viii. valueMissing: It returns true, if an element with a required attribute has no value.
ix. valid: It returns true, if an element's value is valid.

## 276. Give an example usage of rangeOverflow property?

If an element's value is greater than its max attribute then rangeOverflow property returns true. For example, the below form submission throws an error if the value is more than 100,

```
<input id="age" type="number" max="100">
<button onclick="myOverflowFunction()">OK</button>
```

```
function myOverflowFunction() {
  if (document.getElementById("age").validity.rangeOverflow) {
    alert("The mentioned age is not allowed");
  }
}
```

## 277. Is enums feature available in javascript?

No, javascript does not natively support enums. But there are different kind of solutions to simulate them even though they may not provide exact equivalent. For example, you can use freeze or seal on object,

```
var DaysEnum = Object.freeze({"monday":1, "tuesday":2, "wednesday":3, ...})
```

## 278. What is an enum?

An enum is a type restricting variables to one value from a predefined set of constants. JavaScript has no enums but typescript provides built-in enum support.

```
enum Color {
  RED, GREEN, BLUE
}
```

## 279. How do you list all properties of an object?

You can use `Object.getOwnPropertyNames()` method which returns an array of all properties found directly in a given object. Let's the usage of it in an example,

```
const newObject = {
  a: 1,
  b: 2,
  c: 3
};

console.log(Object.getOwnPropertyNames(newObject));  ["a", "b", "c"]
```

## 280. How do you get property descriptors of an object?

You can use `Object.getOwnPropertyDescriptors()` method which returns all own property descriptors of a given object. The example usage of this method is below,

```
 const newObject = {
   a: 1,
   b: 2,
   c: 3
 };
const descriptorsObject = Object.getOwnPropertyDescriptors(newObject);
console.log(descriptorsObject.a.writable); //true
console.log(descriptorsObject.a.configurable); //true
console.log(descriptorsObject.a.enumerable); //true
console.log(descriptorsObject.a.value); // 1
```

## 281. What are the attributes provided by a property descriptor?

A property descriptor is a record which has the following attributes

i. value: The value associated with the property
ii. writable: Determines whether the value associated with the property can be changed or not
iii. configurable: Returns true if the type of this property descriptor can be changed and if the property can be deleted from the corresponding object.
iv. enumerable: Determines whether the property appears during enumeration of the properties on the corresponding object or not.
v. set: A function which serves as a setter for the property
vi. get: A function which serves as a getter for the property

## 282. How do you extend classes?

The `extends` keyword is used in class declarations/expressions to create a class which is a child of another class. It can be used to subclass custom classes as well as built-in objects. The syntax would be as below,

```
class ChildClass extends ParentClass { ... }
```

Let's take an example of Square subclass from Polygon parent class,

```
class Square extends Rectangle {
  constructor(length) {
    super(length, length);
    this.name = 'Square';
  }

  get area() {
    return this.width * this.height;
  }

  set area(value) {
    this.area = value;
  }
}
```

## 283. How do I modify the url without reloading the page?

The `window.localtion.url` property will be helpful to modify the url but it reloads the page. HTML5 introduced the `history.pushState()` and `history.replaceState()` methods, which allow you to add and modify history entries, respectively. For example, you can use pushState as below,

```
window.history.pushState('page2', 'Title', '/page2.html');
```

## 284. How do you check whether an array includes a particular value or not?

The `Array#includes()` method is used to determine whether an array includes a particular value among its entries by returning either true or false. Let's see an example to find an element(numeric and string) with in array.

```
var numericArray = [1, 2, 3, 4];
console.log(numericArray.includes(3)); // true
```

```
var stringArray = ['green', 'yellow', 'blue'];
console.log(stringArray.includes('blue')); //true
```

## 285. How do you compare scalar arrays?

You can use length and every methods of arrays to compare two scalar(compared directly using ===) arrays. The combination of these expressions can give the expected result,

```
const arrayFirst = [1,2,3,4,5];
const arraySecond = [1,2,3,4,5];
console.log(arrayFirst.length === arraySecond.length && arrayFirst.every((value,
index) => value === arraySecond[index])); // true
```

If you would like to compare arrays irrespective of order then you should sort them before,

```
const arrayFirst = [2,3,1,4,5];
const arraySecond = [1,2,3,4,5];
console.log(arrayFirst.length === arraySecond.length &&
arrayFirst.sort().every((value, index) => value === arraySecond[index])); //true
```

## 286. How to get the value from get parameters?

The `new URL()` object accepts url string and `searchParams` property of this object can be used to access the get parameters. Remember that you may need to use polyfill or `window.location` to access the URL in older browsers(including IE).

```
let urlString = "http://www.some-domain.com/about.html?x=1&y=2&z=3";
//window.location.href
let url = new URL(urlString);
let parameterZ = url.searchParams.get("z");
console.log(parameterZ); // 3
```

## 287. How do you print numbers with commas as thousand separators?

You can use `Number.prototype.toLocaleString()` method which returns a string with a language-sensitive representation such as thousand separator,currency etc of this number.

```
function convertToThousandFormat(x){
  return x.toLocaleString(); // 12,345.679
}

console.log(convertToThousandFormat(12345.6789));
```

## 288. What is the difference between java and javascript?

Both are totally unrelated programming languages and no relation between them. Java is statically typed, compiled, runs on its own VM. Whereas Javascript is dynamically typed, interpreted, and runs in a browser and nodejs environments. Let's see the major differences in a tabular format, | Feature | Java | JavaScript | |---- | -------- | Typed | It's a strongly typed language | It's a dynamic typed language | | Paradigm |

Object oriented programming | Prototype based programming | | Scoping | Block scoped | Function-scoped | | Concurrency | Thread based | event based | | Memory | Uses more memory | Uses less memory. Hence it will be used for web pages |

## 289. Is javascript supports namespace?

JavaScript doesn't support namespace by default. So if you create any element(function, method, object, variable) then it becomes global and pollute the global namespace. Let's take an example of defining two functions without any namespace,

```
function func1() {
    console.log("This is a first definition");

}
function func1() {
    console.log("This is a second definition");
}
func1(); // This is a second definition
```

It always calls the second function definition. In this case, namespace will solve the name collision problem.

## 290. How do you declare namespace?

Even though JavaScript lack namespaces, we can use Objects , IIFE to create namespaces.

i. **Using Object Literal Notation:** Let's wrap variables and function inside Object literal which act as a namespace. After that you can access them using object notation

```
var namespaceOne = {
   function func1() {
       console.log("This is a first definition");
   }
}
var namespaceTwo = {
     function func1() {
         console.log("This is a second definition");
     }
 }
namespaceOne.func1(); // This is a first definition
namespaceTwo.func1(); // This is a second definition
```

ii. **Using IIFE (Immediately invoked function expression):** The outer pair of parenthesis of IIFE creates a local scope for all the code inside of it and makes the anonymous function a function expression. Due to that, you can create same function in two different function expressions to act as namespace.

```
(function() {
 function fun1(){
   console.log("This is a first definition");
   } fun1();
}());
```

```
(function() {
    function fun1(){
       console.log("This is a second definition");
   } fun1();
 }());
```

iii. **Using a block and a let/const declaration:** In ECMAScript 6, you can simply use a block and a let declaration to restrict the scope of a variable to a block.

```
{
 let myFunction= function fun1(){
 console.log("This is a first definition");
 }
 myFunction();
}
 //myFunction(): ReferenceError: myFunction is not defined.

{
 let myFunction= function fun1(){
 console.log("This is a second definition");
 }
 myFunction();
}
 //myFunction(): ReferenceError: myFunction is not defined.
```

## 291. How do you invoke javascript code in an iframe from parent page?

Initially iFrame need to be accessed using either `document.getElementBy` or `window.frames`. After that `contentWindow` property of iFrame gives the access for targetFunction

```
document.getElementById('targetFrame').contentWindow.targetFunction();
window.frames[0].frameElement.contentWindow.targetFunction(); // Accessing iframe this way may not work in latest versions chrome and firefox
```

## 292. How do get the timezone offset from date?

You can use `getTimezoneOffset` method of date object. This method returns the time zone difference, in minutes, from current locale (host system settings) to UTC

```
var offset = new Date().getTimezoneOffset();
console.log(offset); // -480
```

## 293. How do you load CSS and JS files dynamically?

You can create both link and script elements in the DOM and append them as child to head tag. Let's create a function to add script and style resources as below,

```
function loadAssets(filename, filetype) {
  if (filetype == "css") { // External CSS file
       var fileReference = document.createElement("link")
       fileReference.setAttribute("rel", "stylesheet");
       fileReference.setAttribute("type", "text/css");
       fileReference.setAttribute("href", filename);
  } else if (filetype == "js") { // External JavaScript file
```

```
        var fileReference = document.createElement('script');
        fileReference.setAttribute("type", "text/javascript");
        fileReference.setAttribute("src", filename);
    }
    if (typeof fileReference != "undefined")
        document.getElementsByTagName("head")[0].appendChild(fileReference)
}
```

## 294. What are the different methods to find HTML elements in DOM?

If you want to access any element in an HTML page, you need to start with accessing the document object. Later you can use any of the below methods to find the HTML element,

iii. document.getElementById(id): It finds an element by Id

iv. document.getElementsByTagName(name): It finds an element by tag name

v. document.getElementsByClassName(name): It finds an element by class name

## 295. What is jQuery?

jQuery is a popular cross-browser JavaScript library that provides Document Object Model (DOM) traversal, event handling, animations and AJAX interactions by minimizing the discrepancies across browsers. It is widely famous with its philosophy of "Write less, do more". For example, you can display welcome message on the page load using jQuery as below,

```
$(document).ready(function(){ // It selects the document and apply the function on page load
    alert('Welcome to jQuery world');
});
```

**Note:** You can download it from jquery official site or install it from CDNs, like google.

## 296. What is V8 JavaScript engine?

V8 is an open source high-performance JavaScript engine used by the Google Chrome browser, written in C++. It is also being used in the node.js project. It implements ECMAScript and WebAssembly, and runs on Windows 7 or later, macOS 10.12+, and Linux systems that use x64, IA-32, ARM, or MIPS processors. **Note:** It can run standalone, or can be embedded into any C++ application.

## 297. Why do we call javascript as dynamic language?

JavaScript is a loosely typed or a dynamic language because variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned/re-assigned with values of all types.

```
let age = 50;    // age is a number now
age  = 'old'; // age is a string now
```

```
age  = true;  // age is a boolean
```

## 298. What is a void operator?

The `void` operator evaluates the given expression and then returns undefined(i.e, without returning value). The syntax would be as below,

```
void (expression)
void expression
```

Let's display a message without any redirections or reloads

```
<a href="javascript:void(alert('Welcome to JS world'))">Click here to see a
message</a>
```

**Note:** This operator is often used to obtain the undefined primitive value, using "void(0)".

## 299. How to set the cursor to wait?

The cursor can be set to wait in JavaScript by using the property "cursor". Let's perform this behavior on page load using the below function.

```
function myFunction() {
window.document.body.style.cursor = "wait";
}
```

and this function invoked on page load

```
<body onload="myFunction()">
```

## 300. How do you create an infinite loop?

You can create infinite loop using for and while loops without using any expressions. The for loop construct or syntax is better approach in terms of ESLint and code optimizer tools,

```
for (;;) {}
while(true) {
}
```

## 301. Why do you need to avoid with statement?

JavaScript's with statement was intended to provide a shorthand for writing recurring accesses to objects. So it can help reduce file size by reducing the need to repeat a lengthy object reference without performance penalty. Let's take an example where it is used to avoid redundancy when accessing an object several times.

```
a.b.c.greeting   = 'welcome';
a.b.c.age = 32;
```

Using `with` it turns this into:

```
with(a.b.c) {
        greeting   = "welcome";
```

```
        age = 32;
}
```

But this `with` statement creates performance problems since one cannot predict whether argument will refer to a real variable or to a property inside the with argument.

## 302. What is the output of below for loops?

```
for (var i = 0; i < 4; i++) { // global scope
  setTimeout(() => console.log(i));
}

for (let i = 0; i < 4; i++) { // block scope
  setTimeout(() => console.log(i));
}
```

The output of the above for loops is 4 4 4 4 and 0 1 2 3 **Explanation:** Due to event loop of javascript, the `setTimeout` callback function is called after the loop has been executed. Since the variable i is declared with `var` keyword it became a global variable and the value was equal to 4 using iteration when the time setTimeout function is invoked. Hence, the output of the first loop is `4 4 4 4`. Whereas in the second loop, the variable i is declared as `let` keyword it became a block scoped variable and it holds a new value(0, 1 ,2 3) for each iteration. Hence, the output of the first loop is `0 1 2 3`.

## 309. List down some of the features of ES6?

Below are the list of some new features of ES6,

.Support for constants or immutable variables

i. Block-scope support for variables, constants and functions
ii. Arrow functions
iii. Default parameters
iv. Rest and Spread Parameters
v. Template Literals
vi. Multi-line Strings
vii. Destructuring Assignment
viii. Enhanced Object Literals
ix. Promises
x. Classes
xi. Modules

## 310. What is ES6?

ES6 is the sixth edition of the javascript language and it was released on June 2015. It was initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015. Almost all the modern browsers support ES6 but for the old browsers there are many transpilers, like Babel.js etc.

## 311. Can I redeclare let and const variables?

No, you cannot redeclare let and const variables. If you do, it throws below error

```
Uncaught SyntaxError: Identifier 'someVariable' has already been declared
```

**Explanation:** The variable declaration with `var` keyword refers to a function scope and the variable is treated as if it were declared at the top of the enclosing scope due to hoisting feature. So all the multiple declarations contributing to the same hoisted variable without any error. Let's take an example of re-declaring variables in the same scope for both var and let/const variables.

```
var name = 'John';
function myFunc() {
    var name = 'Nick';
    var name = 'Abraham'; // Re-assigned in the same function block
    alert(name); // Abraham
}
myFunc();
alert(name); // John
```

The block-scoped multi-declaration throws syntax error,

```
let name = 'John';
function myFunc() {
    let name = 'Nick';
    let name = 'Abraham'; // Uncaught SyntaxError: Identifier 'name' has already
been declared
    alert(name);
}

myFunc();
alert(name);
```

## 312. Is const variable makes the value immutable?

No, the const variable doesn't make the value immutable. But it disallows subsequent assignments(i.e, You can declare with assignment but can't assign another value later)

```
const userList = [];
userList.push('John'); // Can mutate even though it can't re-assign
console.log(userList); // ['John']
```

## 313. What are default parameters?

In E5, we need to depends on logical OR operator to handle default values of function parameters. Whereas in ES6, Default function parameters feature allows parameters to be initialized with default values if no value or undefined is passed. Let's compare the behavior with an examples,

```
//ES5
var calculateArea = function(height, width) {
   height =  height || 50;
   width = width || 60;

   return width * height;
}
```

```
console.log(calculateArea()); //300
```

The default parameters makes the initialization more simpler,

```
//ES6
var calculateArea = function(height = 50, width = 60) {
   return width * height;
}

console.log(calculateArea()); //300
```

## 314. What are template literals?

Template literals or template strings are string literals allowing embedded expressions. These are enclosed by the back-tick ( ) character instead of double or single quotes. In E6, this feature enables using dynamic expressions as below,

```
var greeting = `Welcome to JS World, Mr. ${firstName} ${lastName}.`
```

In ES5, you need break string like below,

```
var greeting = 'Welcome to JS World, Mr. ' + firstName + ' ' + lastName.`
```

**Note:** You can use multi-line strings and string interpolation features with template literals.

## 315. How do you write multi-line strings in template literals?

In ES5, you would have to use newline escape character('\n') and concatenation symbol(+) in order to get multi-line strings.

```
console.log('This is string sentence 1\n' +
'This is string sentence 2');
```

Whereas in ES6, You don't need to mention any newline sequence character,

```
console.log(`This is string sentence
'This is string sentence 2`);
```

## 316. What are nesting templates?

The nesting templates is a feature supported with in template literals syntax to allow inner backticks inside a placeholder ${ } within the template. For example, the below nesting template is used to display the icons based on user permissions whereas outer template checks for platform type,

```
const iconStyles = `icon ${ isMobilePlatform() ? '' :
 `icon-${user.isAuthorized ? 'submit' : 'disabled'}` }`;
```

You can write the above usecase without nesting template feature as well. However, nesting template feature is more compact and readable.

```
//Without nesting templates
 const iconStyles = `icon ${ isMobilePlatform() ? '' :
```

```
  (user.isAuthorized ? 'icon-submit' : 'icon-disabled'}`;
```

## 317. What are tagged templates?

Tagged templates are the advanced form of templates in which tags allow you to parse template literals with a function. The tag function accepts first parameter as array of strings and remaining parameters as expressions. This function can also return manipulated string based on parameters. Let's see the usage of this tagged template behavior of an IT professional skill set in an organization,

```
var user1 = 'John';
var skill1 = 'JavaScript';
var experience1 = 15;

var user2 = 'Kane';
var skill2 = 'JavaScript';
var experience2 = 5;

function myInfoTag(strings, userExp, experienceExp) {
  var str0 = strings[0]; // "Mr/Ms. "
  var str1 = strings[1]; // " is a/an "
  var str2 = strings[2]; // "in"

  var expertiseStr;
  if (experienceExp > 10){
    expertiseStr = 'expert developer';
  } else if(skillExp > 5 && skillExp <= 10) {
    expertiseStr = 'senior developer';
  } else {
    expertiseStr = 'junior developer';
  }

  return `${str0}${userExp}${str1}${experienceExp}{str3}`;
}

var output1 = myInfoTag`Mr/Ms. ${ user1 } is a/an ${ experience1 } in ${skill1}`;
var output2 = myInfoTag`Mr/Ms. ${ user2 } is a/an ${ experience2 } in ${skill2}`;

console.log(output);// Mr/Ms. John is a/an expert developer in JavaScript
console.log(output);// Mr/Ms. Kane is a/an junior developer in JavaScript
```

## 318. What are raw strings?

ES6 provides raw strings feature using `String.raw()` method which is used to get the raw string form of template strings. This feature allows you to access the raw strings as they were entered, without processing escape sequences. For example, the usage would be as below,

```
var calculationString = String.raw `The sum of numbers is \n${1+2+3+4}!`;
console.log(calculationString); // The sum of numbers is 10
```

If you don't use raw strings, the newline character sequence will be processed by displaying the output in multiple lines

```
var calculationString = `The sum of numbers is \n${1+2+3+4}!`;
console.log(calculationString);
// The sum of numbers is
// 10
```

Also, the raw property is available on the first argument to the tag function

```
function tag(strings) {
  console.log(strings.raw[0]);
}
```

## 319. What is destructuring assignment?

The destructuring assignment is a JavaScript expression that makes it possible to unpack values from arrays or properties from objects into distinct variables. Let's get the month values from an array using destructuring assignment

```
var [one, two, three] = ['JAN', 'FEB', 'MARCH'];

console.log(one); // "JAN"
console.log(two); // "FEB"
console.log(three); // "MARCH"
```

and you can get user properties of an object using destructuring assignment,

```
var {name, age} = {name: 'John', age: 32};

console.log(name); // John
console.log(age); // 32
```

## 320. What are default values in destructuring assignment?

A variable can be assigned a default value when the value unpacked from the array or object is undefined during destructuring assignment. It helps to avoid setting default values separately for each assignment. Let's take an example for both arrays and object usecases, **Arrays destructuring:**

```
var x, y, z;

[x=2, y=4, z=6] = [10];
console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

**Objects destructuring:**

```
var {x=2, y=4, z=6} = {x: 10};

console.log(x); // 10
console.log(y); // 4
console.log(z); // 6
```

## 321. How do you swap variables in destructuring assignment?

If you don't use destructuring assignment, swapping two values requires a temporary variable. Whereas using destructuring feature, two variables values can be swapped in one destructuring expression. Let's swap two number variables in array destructuring assignment,

```
var x = 10, y = 20;

[x, y] = [y, z];
console.log(x); // 20
console.log(y); // 10
```

## 322. What are enhanced object literals?

Object literals make it easy to quickly create objects with properties inside the curly braces. For example, it provides shorter syntax for common object property definition as below.

```
//ES6
var x = 10, y = 20
obj = { x, y }
console.log(obj); // {x: 10, y:20}
//ES5
var x = 10, y = 20
obj = { x : x, y : y}
console.log(obj); // {x: 10, y:20}
```

## 323. What are dynamic imports?

The dynamic imports using `import()` function syntax allows us to load modules on demand by using promises or the async/await syntax. Currently this features is in stage4 proposal(https://github.com/tc39/proposal-dynamic-import). The main advantage of dynamic imports is reduction of our bundle's sizes, the size/payload response of our requests and overall improvements in the user experience. The syntax of dynamic imports would be as below,

```
import('./Module').then(Module => Module.method());
```

## 324. What are the use cases for dynamic imports?

Below are some of the use cases of using dynamic imports over static imports,

.Import a module on-demand or conditionally. For example, if you want to load a polyfill on legacy browser

```
if (isLegacyBrowser()) {
    import(···)
    .then(···);
}
```

ii. Compute the module specifier at runtime. For example, you can use it for internationalization.

```
import(`messages_${getLocale()}.js`).then(···);
```

iii. Import a module from within a regular script instead a module.

## 325. What are typed arrays?

Typed arrays are array-like objects from ECMAScript 6 API for handling binary data. JavaScript provides 8 Typed array types,

iii. Int8Array: An array of 8-bit signed integers
iv. Int16Array: An array of 16-bit signed integers
v. Int32Array: An array of 32-bit signed integers
vi. Uint8Array: An array of 8-bit unsigned integers
vii. Uint16Array: An array of 16-bit unsigned integers
viii. Uint32Array: An array of 32-bit unsigned integers
ix. Float32Array: An array of 32-bit floating point numbers
x. Float64Array: An array of 64-bit floating point numbers

For example, you can create an array of 8-bit signed integers as below

```
const a = new Int8Array();
// You can pre-allocate n bytes
const bytes = 1024
const a = new Int8Array(bytes)
```

## 326. What are the advantages of module loaders?

The module loaders provides the below features,

. Dynamic loading
i. State isolation
ii. Global namespace isolation
iii. Compilation hooks
iv. Nested virtualization

## 327. What is collation?

Collation is used for sorting a set of strings and searching within a set of strings. It is parameterized by locale and aware of Unicode. Let's take comparision and sorting features,

. **Comparison:**

```
var list = [ "ä", "a", "z" ]; // In German,  "ä" sorts with "a" Whereas in Swedish,
"ä" sorts after "z"
var l10nDE = new Intl.Collator("de");
var l10nSV = new Intl.Collator("sv");
console.log(l10nDE.compare("ä", "z") === -1); // true
console.log(l10nSV.compare("ä", "z") === +1); // true
```

ii. **Sorting:**

```
var list = [ "ä", "a", "z" ]; // In German,  "ä" sorts with "a" Whereas in Swedish,
"ä" sorts after "z"
var l10nDE = new Intl.Collator("de");
var l10nSV = new Intl.Collator("sv");
```

```
console.log(list.sort(l10nDE.compare)) // [ "a", "ä", "z" ]
console.log(list.sort(l10nSV.compare)) // [ "a", "z", "ä" ]
```

## 328. What is for...of statement?

The for...of statement creates a loop iterating over iterable objects or elements such as built-in String, Array, Array-like objects (like arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. The basic usage of for...of statement on arrays would be as below,

```
let arrayIterable = [10, 20, 30, 40, 50];

for (let value of arrayIterable) {
  value ++;
  console.log(value); // 11 21 31 41 51
}
```

## 329. What is the output of below spread operator array?

```
[...'John Resig']
```

The output of the array is ['J', 'o', 'h', 'n', '', 'R', 'e', 's', 'i', 'g'] **Explanation:** The string is an iterable type and the spread operator with in an array maps every character of an iterable to one element. Hence, each character of a string becomes an element within an Array.

## 330. Is PostMessage secure?

Yes, postMessages can be considered very secure as long as the programmer/developer is careful about checking the origin and source of an arriving message. But if you try to send/receive a message without verifying its source will create cross-site scripting attacks.

## 331. What are the problems with postmessage target origin as wildcard?

The second argument of postMessage method specifies which origin is allowed to receive the message. If you use the wildcard "*" as an argument then any origin is allowed to receive the message. In this case, there is no way for the sender window to know if the target window is at the target origin when sending the message. If the target window has been navigated to another origin, the other origin would receive the data. Hence, this may lead to XSS vulnerabilities.

```
targetWindow.postMessage(message, '*');
```

## 332. How do you avoid receiving postMessages from attackers?

Since the listener listens for any message, an attacker can trick the application by sending a message from the attacker's origin, which gives an impression that the

receiver received the message from the actual sender's window. You can avoid this issue by validating the origin of the message on the receiver's end using "message.origin" attribute. For examples, let's check the sender's origin(http://www.some-sender.com) on receiver side(www.some-receiver.com),

```
//Listener on http://www.some-receiver.com/
window.addEventListener("message", function(message){
    if(/^http://www\.some-sender\.com$/.test(message.origin)){
         console.log('You recieved the data from valid sender', message.data);
   }
});
```

## 333. Can I avoid using postMessages completely?

You cannot avoid using postMessages completely(or 100%). Even though your application doesn't use postMessage considering the risks, a lot of third party scripts use postMessage to communicate with the third party service. So your application might be using postMessage without your knowledge.

## 334. Is postMessages synchronous?

The postMessages are synchronous in IE8 browser but they are asynchronous in IE9 and all other modern browsers (i.e, IE9+, Firefox, Chrome, Safari).Due to this asynchronous behaviour, we use a callback mechanism when the postMessage is returned.

## 335. What paradigm is Javascript?

JavaScript is a multi-paradigm language, supporting imperative/procedural programming, Object-Oriented Programming and functional programming. JavaScript supports Object-Oriented Programming with prototypical inheritance.

## 336. What is the difference between internal and external javascript?

**Internal JavaScript:** It is the source code with in the script tag. **External JavaScript:** The source code is stored in an external file(stored with .js extension) and referred with in the tag.

## 337. Is JavaScript faster than server side script?

Yes, JavaScript is than server side script. Because JavaScript is a client-side script it does require any web server's help for its computation or calculation. So JavaScript is always faster than any server-side script like ASP, PHP, etc.

## 338. How do you get the status of a checkbox?

You can apply `checked` property on selected checkbox in the DOM. If the value is `True` means the checkbox is checked otherwise it is unchecked. For example, the below HTML checkbox element can be access using javascript as below,

```
  <input type="checkbox" name="checkboxname" value="Agree"> Agree the
conditions<br>
console.log(document.getElementById('checkboxname').checked); // true or false
```

## 339. What is the purpose of double tilde operator?

The double tilde operator(~~) is known as double NOT bitwise operator. This operator is going to be a quicker substitute for Math.floor().

## 340. How do you convert character to ASCII code?

You can use `String.prototype.charCodeAt()` method to convert string characters to ASCII numbers. For example, let's find ASCII code for the first letter of 'ABC' string,

```
"ABC".charCodeAt(0) // returns 65
```

Whereas `String.fromCharCode()` method to convert numbers to equal ASCII character.

```
String.fromCharCode(65,66,67); // returns 'ABC'
```

## 341. What is ArrayBuffer?

An ArrayBuffer object is used to represent a generic, fixed-length raw binary data buffer. You can create it as below,

```
let buffer = new ArrayBuffer(16); // create a buffer of length 16
alert(buffer.byteLength); // 16
```

To manipulate an ArrayBuffer, we need to use a "view" object.

```
//Create a DataView referring to the buffer
 let view = new DataView(buffer);
```

## 342. What is the output of below string expression?

```
console.log("Welcome to JS world"[0])
```

The output of the above expression is "W". **Explanation:** The bracket notation with specific index on a string returns the character at a specific location. Hence, it returns character "W" of the string. Since this is not supported in IE7 and below versions, you may need to use .charAt() method to get the desired result.

## 343. What is the purpose of Error object?

The Error constructor creates an error object and the instances of error objects are thrown when runtime errors occur. The Error object can also be used as a base object for user-defined exceptions. The syntax of error object would be as below,

```
new Error([message[, fileName[, lineNumber]]])
```

You can throw user defined exceptions or errors using Error object in try...catch block as below,

```
try {
  if(withdraw > balance)
  throw new Error('Oops! You don't have enough balance');
} catch (e) {
  console.log(e.name + ': ' + e.message);
}
```

## 344. What is the purpose of EvalError object?

The EvalError object indicates an error regarding the global `eval()` function. Even though this exception is not thrown by JavaScript anymore, the EvalError object remains for compatibility. The syntax of this expression would be as below,

```
new EvalError([message[, fileName[, lineNumber]]])
```

You can throw EvalError with in try...catch block as below,

```
try {
  throw new EvalError('Eval function error', 'someFile.js', 100);
} catch (e) {
  console.log(e.message, e.name, e.fileName);              // "Eval function
error", "EvalError", "someFile.js"
```

## 345. What are the list of cases error thrown from non-strict mode to strict mode?

When you apply 'use strict'; syntax, some of the below cases will throw a SyntaxError before executing the script

ii.When you use Octal syntax

```
var n = 022;
```

ii. Using `with` statement
iii. When you use delete operator on a variable name
iv. Using eval or arguments as variable or function argument name
v. When you use newly reserved keywords
vi. When you declare a function in a block

```
if (someCondition) { function f() {} }
```

Hence, the errors from above cases helpful to avoid errors in development/production environments.

## 346. Is all objects have prototypes?

No. All objects have prototypes except for the base object which is created by the user, or an object that is created using the new keyword.

## 347. What is the difference between a parameter and an argument?

Parameter is the variable name of a function definition whereas an argument represent the value given to a function when it is invoked. Let's explain this with a simple function

```
function myFunction(parameter1, parameter2, parameter3) {
  console.log(arguments[0]) // "argument1"
  console.log(arguments[1]) // "argument2"
  console.log(arguments[2]) // "argument3"
}
myFunction("argument1", "argument2", "argument3")
```

## 348. What is the purpose of some method in arrays?

The some() method is used to test whether at least one element in the array passes the test implemented by the provided function. The method returns a boolean value. Let's take an example to test for any odd elements,

```
var array = [1, 2, 3, 4, 5, 6 ,7, 8, 9, 10];

var odd = element ==> element % 2 !== 0;

console.log(array.some(odd)); // true (the odd element exists)
```

## 349. How do you combine two or more arrays?

The concat() method is used to join two or more arrays by returning a new array containing all the elements. The syntax would be as below,

```
array1.concat(array2, array3, ..., arrayX)
```

Let's take an example of array's concatenation with veggies and fruits arrays,

```
  var veggies = ["Tomato", "Carrot", "Cabbage"];
  var fruits = ["Apple", "Orange", "Pears"];
  var veggiesAndFruits = veggies.concat(fruits);
  console.log(veggiesAndFruits); // Tomato, Carrot, Cabbage, Apple, Orange, Pears
```