50+ Advanced JavaScript Interview Questions (ANSWERED)

Kela Casey(https://codersera.com/blog/author/je_45473/)
March 5, 2020(https://codersera.com/blog/2020/03/05/)

Share This Post

Hungry for more advanced and tricky JavaScript Interview questions and answers to transform yourself from a Junior JavaScript Developer into a Senior JavaScript Guru before your next tech interview? Search no more! Come along and explore top hardest JavaScript Interview Questions and Answers including ES6, ES2015 for experienced web developer and get your next six-figure job offer.

Q1: Explain equality in JavaScript

> Topic: **JavaScript**
> Difficulty: ⭐

JavaScript has both strict and type–converting comparisons:

- **Strict comparison (e.g., ===)** checks for value equality without allowing *coercion*
- **Abstract comparison (e.g. ==)** checks for value equality with *coercion* allowed

```
var a = "42";
var b = 42;

a == b;            // true
a === b;        // false
```

Some simple equalityrules:

- If either value (aka side) in a comparison could be the `true` or `false` value, avoid `==` and use `===`.
- If either value in a comparison could be of these specific values (`0`, `""`, or `[]` — empty array), avoid `==` and use `===`.
- In all other cases, you're safe to use `==`. Not only is it safe, but in many cases it simplifies your code in a way that improves readability.

? **Source: FullStack.Cafe (https://www.fullstack.cafe)**

Q2: Provide some examples of non-bulean value coercion to a boolean one

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

The question is when a non-boolean value is coerced to a boolean, does it become `true` or `false`, respectively?

```
"" (empty string)
0, -0, NaN (invalid number)
```

```
0, -0, NaN (invalid number)
null, undefined
false
```

The specific list of “falsy” values in JavaScript is as follows:

Any value that’s not on this “falsy” list is “truthy.” Here are some examples of those:

```
"hello"
42
true
[ ], [ 1, "2", 3 ] (arrays)
{ }, { a: 42 } (objects)
function foo() { .. } (functions)
```

? **Source: FullStack.Cafe (https://www.fullstack.cafe)**

? Javascript Frameworks you should definitely learn in 2020 (https://codersera.com/blog/javascript-frameworks-you-should-definitely-learn-in-2020/)

Q3: What is IIFEs (Immediately Invoked Function Expressions)?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

It’s an Immediately-Invoked Function Expression, or IIFE for short. It executes immediately after it’s created:

```
(function IIFE(){
    console.log( "Hello!" );
})();
// "Hello!"
```

This pattern is often used when trying to avoid polluting the global namespace, because all the variables used inside the IIFE (like in any other normal function) are not visible outside its scope.

? **Source: stackoverflow.com (https://codeburst.io/javascript-generator-yield-next-async-await-e428b0cb52e4?gi=3772a6db7886)**

Q4: When should I use Arrow functions in ES6?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

I'm now using the following rule of thumb for functions in ES6 and beyond:

- Use `function` in the global scope and for Object.prototype properties.
- Use `class` for object constructors.
- Use `=>` everywhere else.

Why use arrow functions almost everywhere?

- **Scope safety**: When arrow functions are used consistently, everything is guaranteed to use the same thisObject as the root. If even a single standard function callback is mixed in with a bunch of arrow functions there's a chance the scope will become messed up.
- **Compactness**: Arrow functions are easier to read and write. (This may seem opinionated so I will give a few examples further on).
- **Clarity**: When almost everything is an arrow function, any regular function immediately sticks out for defining the scope. A developer can always look up the next-higher function statement to see what the thisObject is.

? **Source: stackoverflow.com (https://stackoverflow.com/questions/22939130/when-should-i-use-arrow-functions-in-ecmascript-6)**

Q5: What are the differences between ES6 class and ES5 function constructors?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

Let's first look at example of each:

```
// ES5 Function Constructor
function Person(name) {
  this.name = name;
}

// ES6 Class
class Person {
  constructor(name) {
    this.name = name;
  }
}
```

For simple constructors, they look pretty similar.

The main difference in the constructor comes when using inheritance. If we want to create a `Student` class that subclasses `Person` and add a `studentId` field, this is what we have to do in addition to the above.

```
// ES5 Function Constructor
function Student(name, studentId) {
  // Call constructor of superclass to initialize superclass-derived members.
  Person.call(this, name);

  // Initialize subclass's own members.
  this.studentId = studentId;
}

Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

// ES6 Class
class Student extends Person {
  constructor(name, studentId) {
    super(name);
    this.studentId = studentId;
  }
}
```

It's much more verbose to use inheritance in ES5 and the ES6 version is easier to understand and remember.

? **Source: github.com/yangshun (https://github.com/yangshun/front-end-interview-handbook/blob/master/questions/javascript-questions.md)**

Q6: Explain `Function.prototype.bind`.

> Topic: **JavaScript**
> Difficulty: ★★★

Taken word-for-word from **MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_objects/Function/bind)**:

> The `bind()` method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

In my experience, it is most useful for binding the value of `this` in methods of classes that you want to pass into other functions. This is frequently done in React

components.

? **Source: github.com/yangshun (https://github.com/yangshun/front-end-interview-handbook/blob/master/questions/javascript-questions.md)**

Q7: What's a typical use case for anonymous functions?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

They can be used in IIFEs to encapsulate some code within a local scope so that variables declared in it do not leak to the global scope.

```
(function() {
  // Some code here.
})();
```

As a callback that is used once and does not need to be used anywhere else. The code will seem more self-contained and readable when handlers are defined right inside the code calling them, rather than having to search elsewhere to find the function body.

```
setTimeout(function() {
  console.log('Hello world!');
}, 1000);
```

Arguments to functional programming constructs or Lodash (similar to callbacks).

```
const arr = [1, 2, 3];
const double = arr.map(function(el) {
  return el * 2;
});
console.log(double); // [2, 4, 6]
```

? **Source: github.com/yangshun (https://github.com/yangshun/front-end-interview-handbook/blob/master/questions/javascript-questions.md)**

Q8: Explain the difference between Object.freeze() vs const

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

`const` and `Object.freeze` are two completely different things.

- `const` applies to **bindings** (“variables”). It creates an immutable binding, i.e. you cannot assign a new value to the binding.

```
const person = {
    name: "Leonardo"
};
let animal = {
    species: "snake"
};
person = animal; // ERROR "person" is read-only
```

- `Object.freeze` works on **values**, and more specifically, *object values*. It makes an object immutable, i.e. you cannot change its properties.

```
let person = {
    name: "Leonardo"
};
let animal = {
    species: "snake"
};
Object.freeze(person);
person.name = "Lima"; //TypeError: Cannot assign to read only property 'name' of object
console.log(person);
```

? **Source: stackoverflow.com (https://stackoverflow.com/questions/33124058/object-freeze-vs-const)**

Q9: What is generator in JS?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

Generators are functions which can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances. Generator functions are written using the `function*` syntax. When called initially, generator functions do not execute any of their code, instead returning a type of iterator called a Generator. When a value is consumed by calling the generator’s `next` method, the Generator function executes until it encounters the `yield` keyword.

The function can be called as many times as desired and returns a new Generator each time, however each Generator may only be iterated once.

```
function* makeRangeIterator(start = 0, end = Infinity, step = 1) {
    let iterationCount = 0;
    for (let i = start; i < end; i += step) {
        iterationCount++;
        yield i;
    }
    return iterationCount;
}
```

```
}
```

? **Source: stackoverflow.com (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators_and_Generators)**

Q10: When should we use generators in ES6?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐

To put it simple, generator has two features:

- one can choose to jump out of a function and let outer code to determine when to jump back into the function.
- the control of asynchronous call can be done outside of your code

The most important feature in generators—we can get the next value in only when we really need it, not all the values at once. And in some situations it can be very convenient.

? **Source: stackoverflow.com (https://stackoverflow.com/questions/20768922/javascript-generators-understanding-them)**

Q11: Explain what is hoisting in Javascript

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐⭐

Hoisting is the concept in which Javascript, by default, moves all declarations to the top of the current scope. As such, a variable can be used before it has been declared.

Note that Javascript only hoists declarations and not initializations.

? **Source: https://github.com/kennymkchan (https://github.com/kennymkchan/interview-questions-in-javascript)**

Q12: What will be the output of the following code?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐⭐

```
var output = (function(x) {
  delete x;
  return x;
})(0);
```

```
console.log(output);
```

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

? **Source: github.com/ganqqwerty (https://github.com/ganqqwerty/123-Essential-JavaScript-Interview-Questions/blob/master/README.md)**

Q13: What will be the output of the following code?

> Topic: **JavaScript**
> Difficulty: ★★★★

```
var Employee = {
  company: 'xyz'
}
var emp1 = Object.create(Employee);
delete emp1.company
console.log(emp1.company);
```

Above code will output `xyz` as output.
Here `emp1` object got company as **prototype** property.
`delete` operator doesn't delete prototype property.

`emp1` object doesn't have **company** as its own property. You can test it like:

```
console.log(emp1.hasOwnProperty('company')); //output : false
```

However, we can delete company property directly from `Employee` object using `delete Employee.company` or we can also delete from `emp1` object using `__proto__` property `delete emp1.__proto__.company`.

? **Source: github.com/ganqqwerty (https://github.com/ganqqwerty/123-Essential-JavaScript-Interview-Questions/blob/master/README.md)**

Q14: Explain the Prototype Design Pattern

> Topic: **JavaScript**
> Difficulty: ★★★★

The Prototype Pattern creates new objects, but rather than creating non-initialized objects it returns objects that are initialized with values it *copied* from a prototype - or

objects it returns objects that are initialized with values it *copied* from a prototype – or sample – object. The Prototype pattern is also referred to as the Properties pattern.

An example of where the Prototype pattern is useful is the initialization of business objects with values that match the default values in the database. The prototype object holds the default values that are copied over into a newly created business object.

Classical languages rarely use the Prototype pattern, but JavaScript being a prototypal language uses this pattern in the construction of new objects and their prototypes.

Q15: What is the Temporal Dead Zone in ES6?

> Topic: **JavaScript**
> Difficulty: ★★★★

In ES6 `let` and `const` are hoisted (like `var`, `class` and `function`), but there is a period between entering scope and being declared where they cannot be accessed. **This period is the temporal dead zone (TDZ)**.

Consider:

```
//console.log(aLet)  // would throw ReferenceError

let aLet;
console.log(aLet); // undefined
aLet = 10;
console.log(aLet); // 10
```

In this example the **TDZ** ends when `aLet` is declared, rather than assigned.

? **Source: github.com/ajzawawi (https://github.com/ajzawawi/js-interview-prep/blob/master/answers/es6/temporal-dead-zone.md)**

Q16: Can you describe the main difference between a `.forEach` loop and a `.map()` loop and why you would pick one versus the other?

> Topic: **JavaScript**
> Difficulty: ★★★★

To understand the differences between the two, let's look at what each function does.

`forEach`

- Iterates through the elements in an array.
- Executes a callback for each element.
- Does not return a value.

```
const a = [1, 2, 3];
const doubled = a.forEach((num, index) => {
  // Do something with num and/or index.
});

// doubled = undefined
```

`map`

- Iterates through the elements in an array.
- “Maps” each element to a new element by calling the function on each element, creating a new array as a result.

```
const a = [1, 2, 3];
const doubled = a.map(num => {
  return num * 2;
});

// doubled = [2, 4, 6]
```

The main difference between `.forEach` and `.map()` is that `.map()` returns a new array. If you need the result, but do not wish to mutate the original array, `.map()` is the clear choice. If you simply need to iterate over an array, `forEach` is a fine choice.

? **Source: github.com/yangshun (https://github.com/yangshun/front-end-interview-handbook/blob/master/questions/javascript-questions.md)**

Q17: What’s the difference between a variable that is: `null`, `undefined` or undeclared? How would you go about checking for any of these states?

> Topic: **JavaScript**
> Difficulty: ⭐⭐⭐⭐

Undeclared variables are created when you assign a value to an identifier that is not previously created using `var`, `let` or `const`. Undeclared variables will be defined globally, outside of the current scope. In strict mode, a `ReferenceError` will be thrown when you try to assign to an undeclared variable. Undeclared variables are bad just like how global variables are bad. Avoid them at all cost! To check for them, wrap its usage in a `try`/`catch` block.

```
function foo() {
  x = 1; // Throws a ReferenceError in strict mode
}

foo();
console.log(x); // 1
```

A variable that is `undefined` is a variable that has been declared, but not assigned a value. It is of type `undefined`. If a function does not return any value as the result of executing it is assigned to a variable, the variable also has the value of `undefined`. To check for it, compare using the strict equality (`===`) operator or `typeof` which will give the `'undefined'` string. Note that you should not be using the abstract equality operator to check, as it will also return `true` if the value is `null`.

```
var foo;
console.log(foo); // undefined
console.log(foo === undefined); // true
console.log(typeof foo === 'undefined'); // true

console.log(foo == null); // true. Wrong, don't use this to check!

function bar() {}
var baz = bar();
console.log(baz); // undefined
```

A variable that is `null` will have been explicitly assigned to the `null` value. It represents no value and is different from `undefined` in the sense that it has been explicitly assigned. To check for `null`, simply compare using the strict equality operator. Note that like the above, you should not be using the abstract equality operator (`==`) to check, as it will also return `true` if the value is `undefined`.

```
var foo = null;
console.log(foo === null); // true
console.log(typeof foo === 'object'); // true

console.log(foo == undefined); // true. Wrong, don't use this to check!
```

As a personal habit, I never leave my variables undeclared or unassigned. I will explicitly assign `null` to them after declaring if I don't intend to use it yet. If you use a linter in your workflow, it will usually also be able to check that you are not referencing undeclared variables.

? **Source: github.com/yangshun (https://github.com/yangshun/front-end-interview-handbook/blob/master/questions/javascript-questions.md)**

Q18: Describe the Revealing Module Pattern design pattern

Topic: **JavaScript**
Difficulty: ★★★★★

A variation of the **module pattern** is called the **Revealing Module Pattern**. The purpose is to maintain encapsulation and reveal certain variables and methods returned in an object literal. The direct implementation looks like this:

```
var Exposer = (function() {
  var privateVariable = 10;

  var privateMethod = function() {
    console.log('Inside a private method!');
    privateVariable++;
  }

  var methodToExpose = function() {
    console.log('This is a method I want to expose!');
  }

  var otherMethodIWantToExpose = function() {
    privateMethod();
  }

  return {
      first: methodToExpose,
      second: otherMethodIWantToExpose
  };
})();

Exposer.first();        // Output: This is a method I want to expose!
Exposer.second();       // Output: Inside a private method!
Exposer.methodToExpose; // undefined
```

An obvious disadvantage of it is unable to reference the private methods

An obvious disadvantage of it is unable to reference the private methods

? **Source: scotch.io (https://scotch.io/bar-talk/4-javascript-design-patterns-you-should-know)**

Q19: What's the difference between ES6 Map and WeakMap?

> Topic: **JavaScript**
> Difficulty: ★★★★★

They both behave differently when a object referenced by their keys/values gets deleted. Lets take the below example code:

var map = new Map();
var weakmap = new WeakMap();

```
(function() {
    var a = {
        x: 12
    };
    var b = {
        y: 12
    };

    map.set(a, 1);
    weakmap.set(b, 2);
})()
```

The above IIFE is executed there is no way we can reference `{x: 12}` and `{y: 12}` anymore. Garbage collector goes ahead and deletes the key b pointer from "WeakMap" and also removes `{y: 12}` from memory. But in case of "Map", the garbage collector doesn't remove a pointer from "Map" and also doesn't remove `{x: 12}` from memory.

WeakMap allows garbage collector to do its task but not Map. With manually written maps, the array of keys would keep references to key objects, preventing them from being garbage collected. In native WeakMaps, references to key objects are held "*weakly*", which means that they do not prevent garbage collection in case there would be no other reference to the object.

? **Source:** stackoverflow.com (https://stackoverflow.com/questions/15604168/whats-the-difference-between-es6-map-and-weakmap)

Q20: Is JavaScript a pass-by-reference or pass-by-value language?

> Topic: **JavaScript**
> Difficulty: ★★★★★

It's always pass by value, but for objects the value of the variable is a reference. Because of this, when you pass an object and change its *members*, those changes persist outside of the function. This makes it *look* like pass by reference. But if you actually change the value of the object variable you will see that the change does not persist, proving it's really pass by value.

Example:

```
function changeStuff(a, b, c)
{
  a = a * 10;
  b.item = "changed";
  c = {item: "changed"};
}

var num = 10;
var obj1 = {item: "unchanged"};
var obj2 = {item: "unchanged"};

changeStuff(num, obj1, obj2);

console.log(num);
console.log(obj1.item);
console.log(obj2.item);
```

Output:

```
10
changed
unchanged
```

? **Source:** stackoverflow.com (https://stackoverflow.com/questions/518000/is-javascript-a-pass-by-reference-or-pass-by-value-language)

Q21: How to “deep-freeze” object in JavaScript?

> Topic: **JavaScript**
> Difficulty: ★★★★★

If you want make sure the object is deep frozen you have to create a recursive function to freeze each property which is of type object:

Without deep freeze:

```
let person = {
    name: "Leonardo",
    profession: {
        name: "developer"
    }
};
Object.freeze(person); // make object immutable
person.profession.name = "doctor";
console.log(person); //output { name: 'Leonardo', profession: { name: 'doctor' } }
```

With deep freeze:

```
function deepFreeze(object) {
    let propNames = Object.getOwnPropertyNames(object);
    for (let name of propNames) {
        let value = object[name];
        object[name] = value && typeof value === "object" ?
            deepFreeze(value) : value;
    }
    return Object.freeze(object);
}
let person = {
    name: "Leonardo",
    profession: {
        name: "developer"
    }
};
deepFreeze(person);
person.profession.name = "doctor"; // TypeError: Cannot assign to read only property 'name'
of object
```

? **Source: medium.com (https://medium.com/@leonardobrunolima/javascript-tips-object-freeze-vs-const-4cc0fd30449a)**

Q22: In JavaScript, why is the “this” operator inconsistent?

Topic: **JavaScript**
Difficulty: ★★★★★

The most important thing to understand is that a function object **does not have a fixed `this` value** — the value of `this` changes depending on how the function is called. We say that a function is *invoked with some a particular `this` value* — the `this` value is determined at invocation time, not definition time.

- If the function is called as a "raw" function (e.g., just do `someFunc()`), `this` will be the global object (`window` in a browser) (or `undefined` if the function runs in strict mode).
- If it is called as a method on an object, `this` will be the calling object.
- If you call a function with `call` or `apply`, `this` is specified as the first argument to `call` or `apply`.
- If it is called as an event listener, `this` will be the element that is the target of the event.
- If it is called as a constructor with `new`, `this` will be a newly-created object whose prototype is set to the `prototype` property of the constructor function.
- If the function is the result of a `bind` operation, the function will always and forever have `this` set to the first argument of the `bind` call that produced it.

  (This is the single exception to the "functions don't have a fixed `this`" rule — functions produced by `bind` actually *do* have an immutable `this`.)

? **Source: stackoverflow.com (https://stackoverflow.com/questions/80084/in-javascript-why-is-the-this-operator-inconsistent)**

Q23: Compare Async/Await and Generators usage to achive same functionality

Topic: **JavaScript**
Difficulty: ★★★★★

- Generator function are executed yield by yield i.e one yield-expression at a time by its iterator (the next method) where as Async-await, they are executed sequential await by await.
- Async/await makes it easier to implement a particular use case of Generators.
- The return value of Generator is always {value: X, done: Boolean} where as for Async function it will always be a promise that will either resolve to the value X or throw an error.
- Async function can be **decomposed into Generator and promise** implementation like:

implementation like:

? **Source: stackoverflow.com (https://codeburst.io/javascript-generator-yield-next-async-await-e428b0cb52e4)**

? Top Programming Languages 2020 For Android App Development (https://codersera.com/blog/top-programming-languages-2020-for-android-app-development/)

Q24. In what ways we can empty a JavaScript Array?

Topic: **JavaScript**
Difficulty: ★★★★★

Below techniques can be used to get an empty array:

- You can modify the older array as an empty array.

```
emptyArray = [ ];
```

- You can fix the length according to you of a predefined array to zero.

```
definedArray.length = 0;
```

- You can update the array to contain zero elements with the help of the JavaScript splice() method.

```
definedArray.splice(0, definedArray.length);
```

- You can pluck out all the elements of the array with the help of array methods: pop() or shift().

```
while(definedArray.length){
        definedArray.pop();
} //pop() method
```

Or

```
while(definedArray.length){        definedArray.shift(); } //shift() method
```

Q25. What is the role of closures in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ★★★★★

The script builds a closure in JavaScript at the time of making of a function. It is a local variable that is present in the memory even after the function finishes its execution. Besides, it has a great access to the variable in three scopes: variable in its scope, global variables, and variables in the enclosing function's scope. However, closures are tough in JavaScript to make sure the variable privacy, that is, to build private variables. Since JavaScript has no modifiers for access, closures permit the programmer to make variables that are not directly accessible.

Q26. **Write the JavaScript code to print the following output on your console window.**

Array: ["DataFlair", 2019, 1.0, true]

> Topic: **JavaScript**
>
> Difficulty: ★★★★

The code to print the below output in the console window is:

```
<html>
  <body>
```

```
    <script>
        var array = ["DataFlair", 2019, 1.0, true];
        var msg = "Array: [";
        for(var i = 0; i < array.length-1; i++){
            msg += array[i] + ", ";
        }
        msg += array[array.length-1] + "]";
            console.log(msg);
    </script>

  </body>
</html>
```

Q27. Explain the procedure of document loading.

> Topic: **JavaScript**
>
> Difficulty: ★★★★★

Loading a document tells preparing it for further execution on the system. The document loads instantly on the browser when there is a running document on the system. The application (browser) permits the JavaScript engine to do dual tasks:

- Search for all the properties, provided to the object.

- Involve all the property values, used in the content that is being served for the page about to load.

To load the document instantly, it is a great practice to add the <script> tag inside the <body> tag. The below program loads the document automatically and returns the OS details of the user:

```
<html>
  <body>
  <H1>JavaScript used</H1>

    <script>
        <!-- Comments to hide js in old browsers
        document.write("Give the version: " + navigator.appVersion)
        document.write(" of <b>" + navigator.appName + "</b>.")
        // end of the comment section -->
    </script>

  </body>
</html>
```

Q28. How can you eliminate duplicate values from a JavaScript array?

> Topic: **JavaScript**
>
> Difficulty: ★★★★

There are several ways to eliminate duplicate values from an array. You will the most popular one's below:

- **By using set**

This is the simplest way you can eliminate duplicate values from a JavaScript array. Set is a JavaScript object that built-in and stores the creative values in a different array. The below program will build an array with innovative values:

```
function uniqueArray(array) {
        var newSet= new Set(array);
            var newArray = Array.from(newSet);
        return newArray;
}
var arr = [1,5,2,4,1,6]
console.log(uniqueArray(arr));
```

- **By using filter**

This valuable approach eliminates all the duplicate values from the JavaScript array by implementing a filter on it. The filter() method needs three arguments: element (we wish to check for duplicity), index (of the element) and array (containing duplicates). The below program returns an innovative array:

```
function uniqueArray(arr){
        var newSet = arr.filter(function(elem, index, self) {
            return index == self.indexOf(elem);
        })
        return newSet;
}
var arr = [1,5,2,4,1,6]
console.log(uniqueArray(arr));
```

- **By using for loop**

This helpful method is a somewhat tiresome and not always suggested. However, you can simply build a unique array by making an empty array and only putting fresh values that are not already there in the array. The code implementation looks like this:

```
function uniqueArray(dups_names) {
    var unique = [];
    arr.forEach(function(i) {
        if (!unique[i]) {
```

```
                unique[i] = true;
        }
    });
    return Object.keys(unique);
}
var arr = [1,5,2,4,1,6]
console.log(uniqueArray(arr));
```

Q29. **Is it possible to write a multi-line string in JavaScript?**

> Topic: **JavaScript**
>
> Difficulty: ★★★★

Yes, JavaScript offers the valuable facility to write a String in various lines using the below three approaches:

- **Using backticks**

```
var string = `line1
line2
line3`;
```

- **Using + operator**

```
var string = “line1” +
“line2” +
“line3”;
```

- **Using \ (backslash)**

```
var string = “line1 \
line2 \
line3”;
```

Q30. **Explain the output of the following code.**

> Topic: **JavaScript**
>
> Difficulty: ★★★★

```
var courses = ["JavaScript","Java","C","C++","Python"];
delete courses[2];
console.log(courses.length);
```

The output of the above-mentioned code is 5. It’s because the delete operator does not really affect the entire length of the array as the operates removes only the value

which is there at the position. At the deleted index, we receive the value undefinedx1 (in Chrome) or just undefined/ empty. Therefore, if we have wished to try the statement **console.log(courses),**we will get **(5) [“JavaScript”, “Java”, empty, “C++”, “Python”].**

Q31. Can you remove the content from a website? If yes, then how?

> Topic: **JavaScript**
>
> Difficulty: ★★★★

Yes, you will see some ways in which the content of a website can be easily removed at will. The simplest way is to hide the entire content from the browser you wish to remove. There is a display property of the JavaScript Style Attribute that assists us finish the job. Also, whenever we thing of adding any content to the webpage, its default display fixes to block (display: block). Moreover, we can easily modify the value to none (display: none). It will protect the content from being visible in the browser window. This whole approach only hide the details, you will still see it in the source code, and simply access it from the Elements window.

There is one more way to do this by removing the element node corresponding the website content. This valuable technique entirely removes the data from your webpage along with the code. Also, we can do this by **.removeChild()** method. Or if we have to deal with an array (involving a NodeList), then we can simply empty that array, and receive the similar result. The final decision of choosing the correct method for the situation completely depends on the programmer and his/her requirements.

Q32. Write a program to calculate associative array’s length.

> Topic: **JavaScript**
>
> Difficulty: ★★★★

Following is the JavaScript code to calculate the length of an associative array:

```
<html>
  <body>
    <script>
      //an associative array
      var associativeArray = {one: "DataFlair", two:"JavaScript", three: 435, four: true};
      var count = 0;
      //calculating the length of the array
```

```
      //calculating the length of the array
      for (arrayValue in associativeArray){
        count = count + 1;
      }
      console.log(count);
    </script>
  </body>
</html>
```

The code creates the output 4 in the browser console.

Q33. What would the given code return?

```
console.log(typeof typeof 1);
```

> Topic: **JavaScript**
>
> Difficulty: ★★★★

The output of the above-mentioned code in the console will be 'string'. So, the value that is returned by typeof 1 is 'number'; typeof 'number' will return a string.

Q34. Write an easy function mentioning whether or not a string is a palindrome.

> Topic: **JavaScript**
>
> Difficulty: ★★★★

The following code checks whether the two strings are palindromes or not.

```
<html>
  <body>

    <script>
        function isPalindrome(str) {
                str = str.replace(/\W/g, '').toLowerCase(); //gets a lowercase string
                var string = str.split('').reverse().join(''); //reverses the string
                //checks if the string and the reversed strings are the same
                if(string == str)
                        return "A palindrome"
                else
                        return "Not a palindrome"
        }
        console.log(isPalindrome("level"));
        console.log(isPalindrome("levels"));
    </script>

  </body>
</html>
```

The output in the browser console demonstrates the below output:

A palindrome
Not a palindrome

Q35. What is the function of close () in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ★★★

The function of close () is mainly used to close the latest window. You have to write window.close() to make sure that this command is clearly associated with a window object and not the other JavaScript object.

Q36. What is the JavaScript Event Delegation Model?

> Topic: **JavaScript**
>
> Difficulty: ★★★

There is a lot of cool stuff available in JavaScript and one of them is Delegation model. When you are bubbling and capturing, it permit functions to apply a single handler to several elements at a specific time then it's called Event Delegation. It basically permits you to put event listeners to single parent instead of particular nodes. That specific listener analyzes the bubbled events to get a match on the child elements. Several individuals think it to be daunting job but in reality, it seems easy if one begins to understand it.

Q37. What do you mean by Imports and Exports?

> Topic: **JavaScript**
>
> Difficulty: ★★★★

Exports and Imports specially assist us in writing modular JavaScript code. Using these, we can easily split our code into various files. Imports enables us taking only particular variables or procedures of a file. We can simply import variables or methods that are exported by a module. Following is the example for more details.

```
//index.js

 import name, age from './person';
```

```
import name,age from "./person";

console.log(name);
console.log(age);

//person.js

let name ='Sharad', occupation='developer', age =26;

export { name, age};
```

Q38. Explain the function “use strict”.

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

“Use Strict” is a JavaScript directive that was launched in Es5. The main purpose of using this directive is to enforce the code is performed in strict mode. In this mode, we cannot use a variable without proclaiming it. This directive is ignored by the previous versions of JavaScript.

Q39. What do you mean by JavaScript BOM?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

BOM refers to “Browser Object Modal” that permits JavaScript to “talk” to the browser, modern browsers, no standards apply same BOMS – screen, history, window, location, timing, navigator, and cookies.

Q40. What is Typecasting in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

In JavaScript, whenever we require to transform a variable from one data to another, then Typecasting is used. Besides, we can do this through library functions. There are mainly 3 typecasts that you will see in JavaScript Programming:

- Boolean(value): Casts the inputted value to a Boolean
- Number(value): Casts the inputted value to a Floating point Number or an Integer.

- String(value) : Casts the inputted value value a string

Q41. Tell the way to import all the exports of a file as an object.

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

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

```
Example:
objectname.member1;
objectname.member2;
objectname.memberfunc();
```

Q42. What according to you are the primitive data types in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

A primitive is a simple data type that is not made out of some other data types. It can only show you a single value. Every primitive is considered built-in data type by requirement, (the compiler has to be aware about them), yet not all built-in data types are called primitives. There are 5 types of primitive data that are available in JavaScript. They are boolean, undefined, null, number, and string are available. Everything else apart from this is an object only.

Q43. What do you know about the different types of Popup boxes present in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

You will find 3 types of Popup boxes available in JavaScript:

- Confirm

- Alert
- Prompt

Q44. What is the way to add/delete properties to object in JavaScript dynamically?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

You can simply add a property to an object using object.property_name=value, delete object.property_name is mainly used to remove a property.

Example:

```
let user = new Object();
// adding a property
user.name='Anil';
user.age  =25;
console.log(user);
delete user.age;
console.log(user);
```

Q45. In what way can you decode or encode a URL in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

encodeURL() function is mainly used to encode an URL in JavaScript. It requires a URL string as a parameter and return that encoded string. Point to note: encodeURL() did not encode characters such as **/ ? : @ & = + $ #**, if you need to encode these characters too, then use encodeURLComponent().

Usage:

```
var uri = "my profile.php?name=sammer&occupation=pāntiNG";
var encoded_uri = encodeURI(uri);
```

decodeURI() function is basically used to deocde an URL in JavaScript. It requires an encoded URL string as parameter and return that decoded string.

Usage:

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

```
var encoded_uri = encodeURI(uri);
decodeURI(encoded_uri);
```

Q46. In what way we can change the title of the page in JavaScript?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

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

Example usage

```
document.title="My New Title";
```

Q47. What is the difference between null, undefined, or undeclared JavaScript variable?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

Null is a value that you can assign to a particular object or variable. Undefined refers to a variable that has been proclaimed but no value is yet assigned to it. This specific type of variable is declared itself to be undefined. Undeclared refers to the variable that has been declared without any datatype. These all are primitive data types whereas Undeclared is not considered a primitive data type.

Q48. What do you mean by Escape Characters?

> Topic: **JavaScript**
>
> Difficulty: ⭐⭐⭐

Escape characters also known as backlash is mainly used when you are working with some special characters like double quotes, single quotes, ampersands, and apostrophes. Just place backslash before the characters to make it show.

Example:

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

Q49. Explain JavaScript Cookies

Q49. Explain JavaScript Cookies.

Topic: **JavaScript**

Difficulty: ⭐⭐⭐

Cookies are the tiny test files that are properly stored in a computer, and they get build when the user goes to the websites to store some information that they require. Examples are User name details and information about the shopping cart from earlier visits.

Q50. Explain the role of deferred scripts in JavaScripts.

Topic: **JavaScript**

Difficulty: ⭐⭐⭐

The HTML code's parsing while the page is loading is stopped by default until the script has not paused executing. If your server is a little slow or the script is specifically heavy, then your webpage will be delayed. While you are using Deferred, scripts

delays the execution for sometime of the script till the time the HTML parser is running. This lessens the loading time of web pages and get showed up faster.

Wrapping Up

These all were tricky and advanced JavaScript interview questions and answers. We have tried our best to cover every topic which is considered while asking interview questions. We hope you are now clear with all of it. If you have any queries regarding programming languages and interview questions, connect with our **Codersera (https://codersera.com/contact-us)** team.

Enter Your Email

Send →

← (https://codersera.com/blog/react-native-bridg →(https://codersera.com/blog/top-30-questions-

More To Explore

What Are CFML Vulnerabilities & Security Issues? (Https://Codersera.Com/Blog/Cfml-Vulnerabilities-And-Security-Issues/)

ColdFusion Markup Language, more commonly known as CFML, is a scripting language for web development that runs on JVM, .Net framework, and Google App engine. Multiple

Kela Casey • July 6, 2022

Cognizance Of Blockchain Security Implications (Https://Codersera.Com/Blog/Blockchain-Security-Implications/)

Blockchain technology is still in its relative infancy and thus, suffers from a lack of case studies and real-world applications. As a result, many people

Kela Casey • May 4, 2022

(https://codersera.com/blog)

+1 415-630-4868 info@codersera.com 251 Little Falls Drive, Wilmington, New Castle