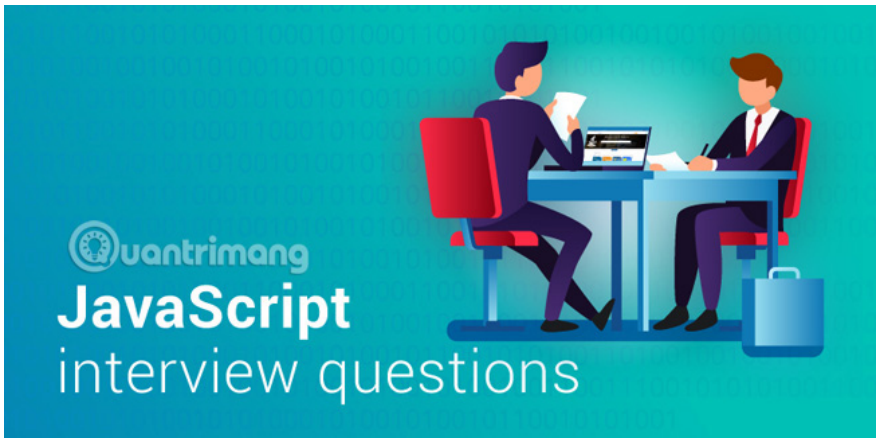
9 popular JavaScript interview questions

If you've missed 'fell in love' and intend to find a job related to this language, you will definitely need to review some of the issues to be able to confidently show your abilities and knowledge. before employers.

JavaScript is considered a great language for the 'newbie'. Partly because the use of the internet is expanding and developing, the web-capable programmers are always welcomed with an attractive salary, and of course, if you have done the web, you can't help mentioning the language. JavaScript language. Another part is that JavaScript does not "make it difficult" for amateurs or newcomers because of strict rules, for example, you can miss a semicolon after the command, the program still runs completely. Good, very different from other famous languages, right?



If you've missed 'fell in love' and intend to find a job that is related to this language, you will definitely need to review the following issues to be confident in showing your abilities and knowledge. I am in front of employers. The more you master the knowledge, overcome the interview questions easily, the higher your chances of getting and good wages will be correspondingly high. Let's take a look at the common issues in JavaScript interviews via TipsMake.com's general article below.

Part 1: The difficult question

Your interview will be 'difficult to chew' if one of the following questions appears unexpectedly

1. Why Math.max () is smaller than Math.min ()

In fact, when running `Math.max() > Math.min()` , the return value is `False` , which sounds unreasonable. However, if no parameters are passed, *Math.min ()* returns *Infinity* and *Math.max ()* returns -*Infinity*. So `Math.max() Math.min()` .

Follow the following code:

```
Math.min(1) // 1 Math.min(1, infinity) // 1 Math.min(1, -infinity) // -infinity
```

If `-infinity` appears as the parameter of `Math.min()` , the result will definitely be `-infinity` matter how many parameters. Meanwhile, if the parameter appears to be `infinity` and some other, the result returned will be that value.

2. Why do the calculations return false results? 0.1 + 0.2 is not equal to 0.3.

This issue involves Javascript storing float data in binary form to exactly one digit after the comma. If you enter the following program into the control panel, you will see the following result:

```
0.1 + 0.2 // 0.30000000000000004 0.1 + 0.2 - 0.2 // 0.10000000000000003 0.1 + 0.
```

Of course this will not cause any big problems if you only implement simple equations that do not need high precision. However, it will cause headaches if you want to check for equality between objects.

There are several solutions to this problem:

Fixed Point - Fixed point

Use when you know the maximum accuracy you need, the number of digits after the comma is fixed.

For example, you are trading with a currency, enter an integer to store the value, instead of entering *$ 4.99*, enter *499* and perform calculations on this number. After finishing the session with it, you can display the results to the end user using an expression like this:

```
result = (value / 100).toFixed(2)
```

to return the result string as desired.

Binary Coded Decimal - Binary Coded Decimal Numbers

If the accuracy in your equation is very important, use this **Binary Coded Decimals (BCD)** format by accessing the BCD library right in JavaScript. Each decimal value is stored separately in a single byte (8 bits). This obviously wastes a lot of bits, but it makes translating into a string that can be read much easier than the traditional binary to decimal conversion. It may also not be the best, because 1 byte can store up to 16 separate values ??while the system uses only 0-9 values. So as initially stated, if your application prioritizes accuracy, use this solution, it's worth a little bit of a trade.

3. Why 018 subtracts 017 by 3?

`018 - 017` returns `3` as the result of silent type conversion. In this case we are referring to octal number.

Quick introduction to octal numbers

You can listen to and use a lot of binary number systems (base 2) and hexadecimal (base 16) in computing, but you may hear less about octal systems (base 8) even though This system also has a prominent position in computer history. In the 1950s and 1960s, the octal system was used for binary abbreviations, helping to cut material costs in expensive systems for manufacturing.

Octal system today

The octal system has an advantage over hexadecimal in some cases because it does not require any letters to represent any number (use 0-7 instead of 0-F).

Returning to the question, in JavaScript, prefix 0 will convert any number to octal numbers. However, 8 does not exist in the octal and the number containing the digit 8 will be converted implicitly into a normal decimal.

Therefore, `018 - 017` actually equivalent to the decimal expression `017` , since `017` is an octal, `018` is a decimal, so `017` also converted to the corresponding decimal number 15. So the answer is This is `3` .

Part 2: Frequently asked questions

In this section, Quantrimang will mention more popular interview questions. This is usually the knowledge you easily overlook when you get used to JavaScript from the beginning, but they are all very useful and can help you write better code later.

4. Difference between Function Declaration and Function Expression

Function declaration uses function keyword and function name. And **Function expression** starts with `var` , `let` or `const` , followed by the name of the function and the operator `=` .

Here are some examples:

```
// Function Declaration function sum(x, y) { return x + y; }; // Function Expres
```

How to use, Function declaration, you can call before declaring or after declaring it all, while the function expression must have a sequence.

The function declaration is moved to the beginning of scope by the JavaScript interpreter and can be called before the declaration or after the declaration is made, call it anywhere in your code. Conversely, calling Function expression must have a sequence: it must be declared before it can be called.

The function expression today is a bit more popular by developers because of several reasons like this:

1. The function expression implements structured codebase, easy to follow and predict.
2. It is possible to use a more concise syntax of ES6 for Function expressions, `let` and `const` provide more control over whether a variable can be reassigned or not.

5. What is the difference between var, let and const?

This is a quite popular interview question since ES6 was released because of the high possibility that recruitment agencies have also used the more modern syntax in this new version.

`var` is the variable declaration keyword that appears right from the first release of JavaScript version. However, it has several disadvantages that lead to the development of two new keywords in ES6 later `let` and `const` .

These three keywords have different approaches related to *assignment, hoisting* and *scope.* Let's go to each part separately.

a. Assignment

The most fundamental difference is that `let` and `var` can be accessed again while `const` is not possible. This makes `const` the best choice for constant variables, preventing errors if randomly reassigned.

b. Hoisting

Similar to the difference between declarations and expressions function (discussed above), variables using `var` to declare are always placed at the top of their respective scope, that is, regardless of where you declare the variable. In a function, it automatically takes it to the top of the function to declare (javascript automatically implements this concept implicitly). Meanwhile, variables use `const` and `let` to declare a scope in a block code, that is, it will not be affected by the concept of hoisting, but if trying to access these variables before they are declared You will receive a "temporal dead zone" error. Take the following example:

```
var x = "global scope"; function foo() { var x = "functional scope"; console.log
```

Here, the results of *foo ()* and *console.log (x)* are as expected. But what if we leave the second `var` ?

```
var x = "global scope"; function foo() { x = "functional scope"; console.log(x);
```

Although defined in a function, `x = "functional scope"` overrides global variables. We need to repeat `var` to determine that the second variable *x is* only within the scope of *foo ().*

c) Scope

While `var` operates in function-scoped, `let` and `const` are block-scoped, a block is all code within curly brackets `{}` , including functions, command statements event and loop. To illustrate the difference, see the following code:

```
var a = 0; let b = 0; const c = 0; if (true) { var a = 1; let b = 1; const c = 1
```

It can be seen that, within the conditional code block, the variable `var a` access range is globally re-accessed, but `let b` and `const c` are not

6. What happens if you specify a variable without keywords?

What happens if you define a variable without using any keywords? Technically, if `x` not defined, then `x = 1` is an abbreviation for `window.x = 1`, but this is a common cause of memory leak.

To prevent this problem, using strict mode, this mode requires you to explicitly declare everything to avoid potential errors. If declaring a variable missing the keyword, the program will display an error: `Uncaught SyntaxError: Unexpected indentifier`.

7. What is the difference between Object Oriented Programming (OOP) and Functional Programming (FP)?

JavaScript is a multi-model language, which means it supports many different programming styles, including event-oriented, functional, and object-oriented.

There are many different programming models, but in contemporary computing, two of the most popular styles are **Functional Programming (FP)** and **Object Oriented Programming (OOP)** - JavaScript can do both.

a. Object-oriented programming

OOP is based on the concept of "object". These are data structures that contain known data fields in JavaScript as "attributes" - and procedures called "methods".

Some objects built into JavaScript can be included as `Math` (using methods like `random`, `max`, `sin`.), `JSON` (used to analyze `JSON` data) or data types such as `String`, `Array`, `Number` and `Boolean`.

Basically, whenever you use built-in methods, prototypes or classes, you are using object-oriented programming.

b. Functional programming

There are many functional programming models built into JavaScript like the `higher-order` function. This is a function that can take another function as an argument.

In addition, JavaScript also has some functions that work with arrays such as `map()`, `reduce()`, `filter()`. all are `higher-order` functions. This means you can string them together to quickly implement the methods in an array.

Although JavaScript initially had some problems with transformability, newer versions of the ECMAScript standard provided fixes. Instead of using the `var` keyword to define variables, you can use `const` and `let`. The first keyword allows you to define constants as implied names. The second keyword `let`, limits the scope of a variable in the function it declares.

8. The difference between Imperative programming and Declarative programming

You can rely on the difference between OOP and FP to answer this question about imperative programming and declarative programming.

These are terms that describe common characteristics between many different programming models. FP is an example of declarative programming, while OOP is an example of imperative programming.

It can be envisioned that with Imperative Programming you are interested in how to solve the problem, go step by step in the most essential way, characterized by `for` , `while` loops, `if` , `switch` .

```
const sumArray = array => { let result = 0; for (let i = 0; i  array.length; i++
```

In contrast, Declarative Programming is concerned with the input and output of the problem. This often leads to more concise code, but only at a certain scale, but it can become more difficult to debug because it is not explicit.

The declaration example for sumArray () function above.

```
const sumArray = array => { return array.reduce((x, y) => x + y) };
```

9. What is prototype-based inheritance (Prototype-Based Inheritance)?

There are a number of different object-oriented programming types and JavaScript uses prototype-based mechanism inheritance. The system that allows inherited behavior through the use of existing objects serves as the prototype.

Note, even if you have not heard about prototypes, you certainly have encountered a prototype system using the in-built methods.

For example, functions used to manipulate arrays such as `map` , `less` , `splice` . are methods of `Array.prototype` object. In fact, every instance object of the array (defined by square brackets `[]` or using `Array()`) inherits from Array.prototype, which is why methods like `map` , `reduce` and `splice` Default available.

This is similar to almost any other built-in object, such as string and boolean (except `Infinity` , `NaN` , `null` and `undefined` no properties or methods).

At the end of the prototype string, we will find `Object.prototype` and almost all objects in JavaScript are an instance of `Object.prototype` . For example, `Array.prototype` and `String.prototype` both inherit properties and methods from `Object.prototype` .

To add properties and methods to an object with prototype syntax, you simply initialize the object as a function and use the prototype keyword to add properties and methods:

```
function Person() {}; Person.prototype.forename = "John"; Person.prototype.surnar
```

In this article, TipsMake.com introduced you to some of the issues that employers often ask to check out a JavaScript developer. The actual interview questions may be different but basically the issues will be similar and revolve around these contents. If you can't answer all the questions, don't worry, try to read a lot, do a lot, find out, Quantrimang is sure you can do well.

If you have any interesting questions related to article content: JavaScript developer interviews, share them in the comments section, this will be helpful for many others.

Thank you for following the article. Good luck!

You finished reading the article "**9 popular JavaScript interview questions**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.
