search

10 JavaScript concepts you need to know for interviews

arnav-aggarwal Aug 21 '17 · 4 min read

#javascript #webdev **#interviews** **#programming**

Self-Learning

There are thousands of people learning JavaScript and web development in the hopes of getting a job. Often, self-learning leaves gaps in people's understanding of the JavaScript language itself.

It's actually surprising how little of the language is needed to make complex web pages. People making entire sites on their own often don't have a good grasp of the fundamentals of

features using basic skills. It's also easy to create a website by relying on Stack Overflow without understanding the code being copied.

> If you're looking to master JavaScript interviews, check out Step Up Your JS: A Comprehensive Guide to Intermediate JavaScript

Interviews

The problem is that questions testing your understanding of JS are exactly what many tech companies ask in their interviews. It becomes clear very quickly when an applicant knows just enough to have scraped by, but doesn't have a solid understanding the language.

Here are concepts that are frequently asked about in web development interviews. This is assuming you already know the basics such as loops, functions, and callbacks.

Concepts

1. Value vs. Reference — Understand how objects, arrays, and functions are copied and passed into functions. Know that the reference is what's being copied. Understand that primitives are copied and passed by copying the value.
2. Scope — Understand the difference between global scope

engine performs a variable lookup.

3. Hoisting — Understand that variable and function declarations are hoisted to the top of their available scope. Understand that function expressions are not hoisted.
4. Closures — Know that a function retains access to the scope that it was created in. Know what this lets us do, such as data hiding, memoization, and dynamic function generation.
5. `this` — Know the rules of `this` binding. Know how it works, know how to figure out what it will be equal to in a function, and know why it's useful.
6. `new` — Know how it relates to object oriented programming. Know what happens to a function called with `new`. Understand how the object generated by using `new` *inherits* from the function's `prototype` property.
7. `apply`, `call`, `bind` — Know how each of these functions work. Know how to use them. Know what they do to `this`.
8. Prototypes & Inheritance — Understand that inheritance in JavaScript works through the `[[Prototype]]` chain. Understand how to set up inheritance through functions and objects and how `new` helps us implement it. Know what the `__proto__` and `prototype` properties are and what they do.
9. Asynchronous JS — Understand the event loop. Understand how the browser deals with user input, web

how JavaScript is both asynchronous and single-threaded.

10. Higher Order Functions — Understand that functions are first-class objects in JavaScript and what that means. Know that returning a function from another function is perfectly legal. Understand the techniques that closures and higher order functions allow us to use.

More Resources

If the links included aren't enough, there are countless resources out there to help you learn these concepts.

I personally created Step Up Your JS: A Comprehensive Guide to Intermediate JavaScript to help developers advance their knowledge. It covers all of these concepts and many more.

Here are resources which I've read or watched at least some of and can recommend.

- You Don't Know JS
- JavaScript is Sexy
- javascript.com
- Frontend Masters
- Eloquent JavaScript

Good luck on your interviews.

search

Step Up Your JS: A Comprehensive Guide to Intermediate JavaScript

What I learned from attending a coding bootcamp and teaching another one

React Ecosystem Setup — Step-By-Step Walkthrough

arnav-aggarwal + FOLLOW
Full-stack developer
@arnavaggarwal farm_fresh_js arnav-aggarwal

Add to the discussion

PREVIEW SUBMIT

152 18 144

search

(Note: the original author has clarified the wording on this point since I wrote this. My note here applies to the *original* wording.)

Great stuff, especially the note about hoisting and higher-order functions. *But*, your note about pass-by-reference is incorrect. It's important to differentiate *what* is passed from *how* it's passed. Yes, JavaScript passes *references*, but it passes them *by value*.

For example, if JavaScript passed object references *by reference*, this would log `{ x: 'bar' }`:

```
var o = { p: 'foo' };
(function(n) {
    n = { x: 'bar' };  // Sets the value of n. Value of o unaffected.
})(o);
console.log(o); // Emits { p: 'foo' }
```

However, because JavaScript passes all arguments *by value*, it emits something like `{ p: 'foo' }`. Changing the value of `n` inside the function doesn't change the value of `o` outside the function.

It's a common point of confusion in object-oriented languages, since this code appears at first glance to be very similar to the code above:

```
var o = { p: 'foo' };
(function(o) {
    o.p = 'bar';  // Sets the value of o.p.
})(o);
console.log(o); // Emits { p: 'bar' }
```

Note that in this example, we're changing a *property* of `o`, rather than `o` itself. Since `o` is a reference to an object, changes to its properties *do* affect those properties outside the scope of the function.

This is a really solid list-- nearly every point here has come up on every JavaScript-related interview I've done, and if you've got these nailed, you're well on your way to getting the gig.

Cheers!

REPLY

152 18 144

search

Interesting distinction, and thanks for clearing that up!

I was curious what would happen if you did something like this

```
var o = { p: 'foo' };
(function(o) {
    o = { x: 'bar' };
})(o)
console.log(o);
```

This is weird at first glance to me because while, yes, you're passing in `o` to the closure, when you set `o` to `{x: 'bar'}`, which `o` are you talking about; is it the `o` outside the function?

This becomes a bit more of a scoping issue. But the solution is that the `o` inside the function is still a separate variable from the `o` outside the function, because it was "declared" separately as an argument (`function(o)`). So the final output is still `{ p: 'foo' }`

If you change it to this:

```
var o = { p: 'foo' };
(function(z) { //note now we've declared 'z', and there's no 'o' in the inner function
    o = { x: 'bar' }; //because there's no 'o' inside here, we use the 'o' defined in line 1
})(o)
console.log(o);
```

`o` gets changed! Because it was never redeclared inside the function. So the final output here is: `{ x: 'bar' }`

3 REPLY

pildit

Aug 22 '17

This has nothing to do with pass-by-reference issue! it's about scoping

4 REPLY

152 18 144

search

actually passing a reference, because you can do:
var o = { p: 'foo' };
(function(n) {
n.p: 'bar'; // Sets the value of n.p, Value of o affected.
})(o);
console.log(o); // Emits { p: 'bar' }

2 REPLY

Johan Fagerberg Aug 24 '17

You are passing *a reference*, but you are not passing *by reference*.

If you were passing by reference, the variable inside the function would refer to the variable outside of the function.

See this PHP example.

```php
<?php
$o = "foo";
function modifyVariable(&$n) {
  $n = "bar";
}
modifyVariable($o);
echo $o; // "bar"
?>
```

This is not the case in JavaScript - the variable inside the function is completely separate from the variable outside the function, but it *refers to the same value*.

You are given a reference to the value - you are not given the variable by reference.

3 REPLY

arnav-aggarwal Aug 25 '17

I don't think most JS developers that are reading this article don't know the concept of true pass by reference as it's used in other programming languages. That's why I think the wording I use is sufficient to get the point across that the reference is being copied instead of the object being copied, even though it is technically by value. Thanks for the suggestion!

152 18 144

I see you updated the wording, and yes, it's now *much* clearer. Awesome! Thanks!

3 REPLY

Gedion Dessie

Aug 24 '17

JavaScript passes functions by reference! So when you pass 'o', you are not passing the value { p: 'foo' }... you are passing its memory location which is named 'o' for readability. 'n' is a local variable inside your function that points to whatever value/reference you are passing to your function. n will refer to the same object o refers to. When you say n = { p: 'bar'}, n is now pointing to a "newly" created object that happens to have the same property 'p' as that of the one you passed to the function.

2 REPLY

David James

Aug 22 '17

For someone wanting to move away from just using JS as a way to add functionality to a basic webpage, and start using it for the complex ecosystem that it has become, this article helps immensely. Thank you!

2 REPLY

edA-qa mort-ora-y

Aug 22 '17

With the advent of the `class` keyword in ES6, is it still necessary to understand `prototype` and friends?

1 REPLY

arnav-aggarwal

Aug 22 '17

Absolutely. The class keyword uses prototypes in its implementation. It's just "syntactic sugar", or a simpler way of using prototypes, but we're still using prototypes nonetheless. It's key to the core of OOP in JS.

search

I think it is, because, in the background, class still uses prototype chain etc. You might understand some things better (or debug faster) if you understand how it really works.

3 REPLY

codypatnaude

Aug 25 '17

The class keyword is just different syntax for creating objects. Once the object is created it still uses the same prototypal inheritance objects always have.

Using the 'class' keyword without understanding prototypal inheritance is like building a house with no understanding of carpentry. Eventually something's going to break, and you won't know how to fix it.

1 REPLY

Martin Häusler

Mar 21 '18

Unfortunately yes, even with the nicer syntax it's still plain old prototype chain. They missed the historical opportunity to clean up this mess.

1 REPLY

codypatnaude

Aug 25 '17

"Understand that objects, arrays, and functions are copied and passed by reference."

If something is passed by reference it is NOT copied. That's the whole point of passing by reference. May want to change the wording there.

1 REPLY

arnav-aggarwal

Aug 25 '17

That's the point. Arrays & objects are not copied when they're assigned, either using `=` or passed into a function. Only the reference is copied/passed in.

152 18 144

search

Thanks for this piece! I've added this to me "Learn Next" list and plan to go through a lot of the links you've shared. I already checked the "hoisting" one and it cleared up a lot of things that have confused me in the past.

1 REPLY

rhymes

Aug 22 '17

Thank you! This deserves a unicorn!

2 REPLY

Rob Waller

Aug 28 '17

Great Post, will read all the links.

2 REPLY

Paul

Aug 23 '17

Great article. Thank you for this!

2 REPLY

Žane Suhadolnik

Aug 24 '17

YDKJS books are essential for anyone that is trying to get into front-end basics.

1 REPLY

Varan Biten

Sep 1 '17

152 18 144

search

1 REPLY

varunvikramsingh Aug 30 '17

How about Promises?

1 REPLY

code of conduct - report abuse

Classic DEV Post from Jun 21 '18

I'm visiting dev.to more & more every day 😍

Benjamin Faught

I've noticed that I'm beginning to visit dev.to more than social-media or almos...

103 16

Another Post You Might Like

How to Build Your Own Serverless Contact Form

Sai gowtham

Build Your Own Serverless Contact Form

101

152 18 144

search

A dev's guide to meditation - part 1

Daragh Byrne

Why meditation might help you code

122 3

How to understand Angular using the Documentation

Hassan Sani - Apr 8

Learn To Fold Your JS Arrays

Neil Syiemlieh - Apr 8

JavaScript: Armadilhas do Async/Await em loops

Eduardo Rabelo - Apr 8

Number Truncation in JavaScript

Samantha Ming - Apr 8