Pass-By Value vs. Pass-By Reference in JavaScript

Reina Mitchell Follow
Jul 25 · 3 min read ★

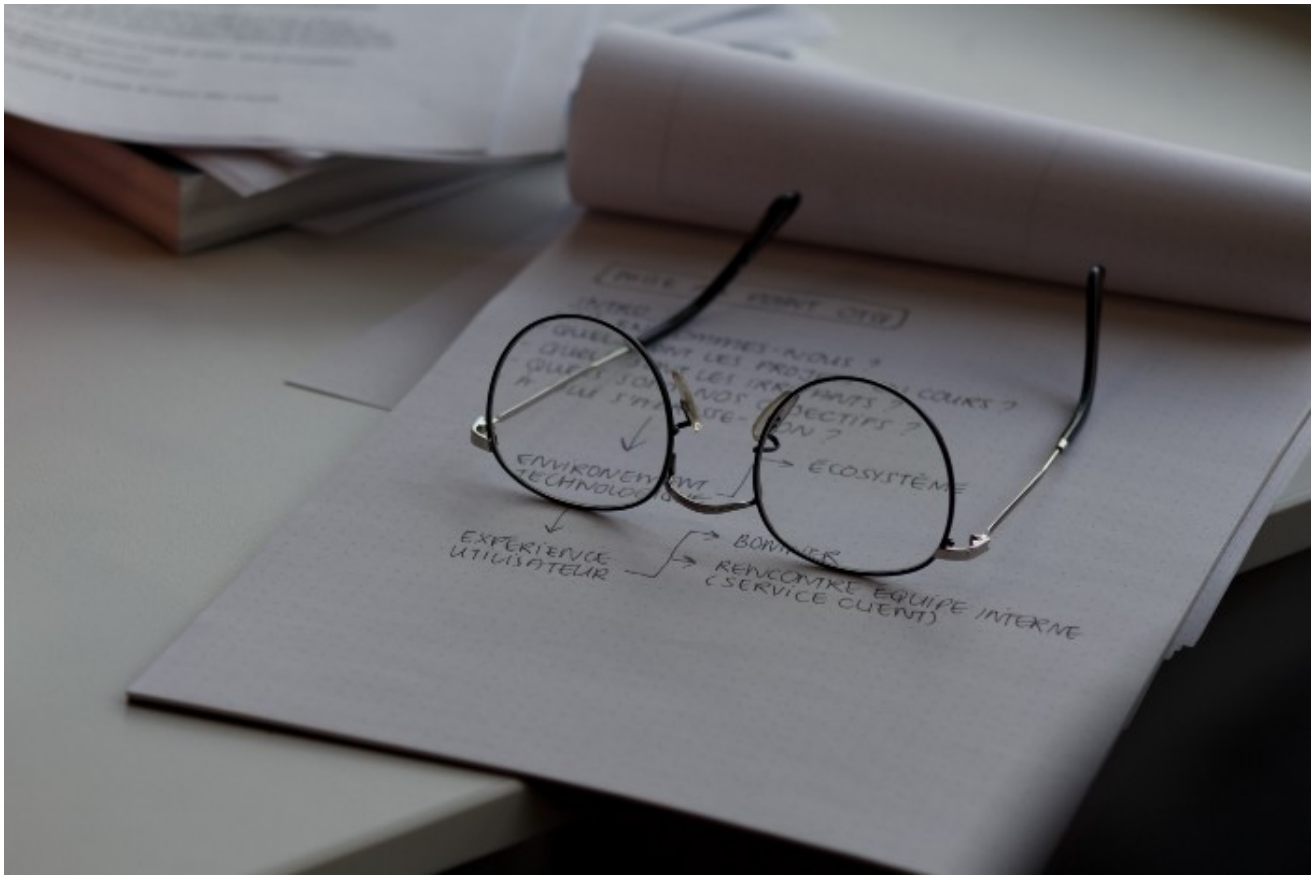


Photo by Sigmund on Unsplash

Introduction

One of my favorite topics in programming is understanding pass-by-value and pass-by-reference. It is a concept that came up during one of my technical interviews and is one that you should know if you don't already.

The difference between pass-by-reference and pass-by-value is, pass-by-value creates a new space in memory and makes a copy of a value, whereas pass-by-reference does not. Instead of making a copy, pass-by-reference does exactly what it sounds like; a value stored in memory gets referenced.

Short Way To Remember: ***Complex*** *values are pass-by-reference and* ***Primitive*** *values are pass-by-value.*

Pass-By Reference	Pass-by Value
• Arrays • Objects	• Strings • Numbers • Booleans • Undefined • Null • Everything else!

Pass-By-Reference:

Objects

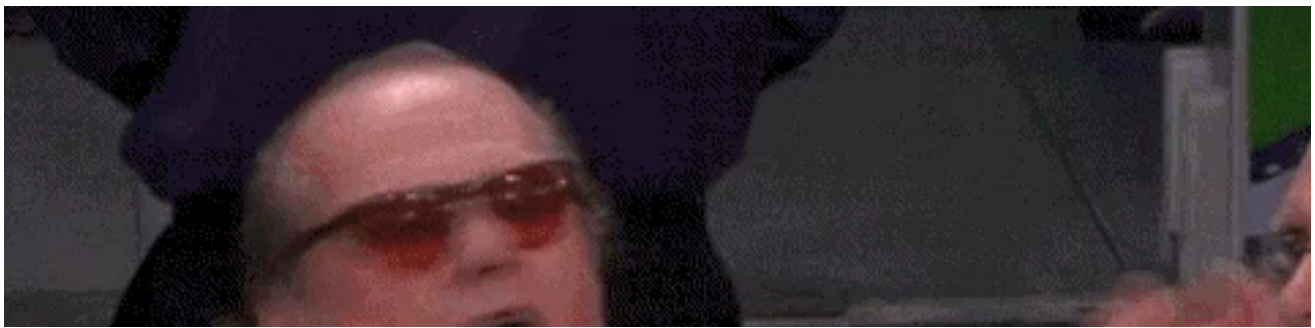
To get started, I am going to start by making an object literal and assign it to the variable `john`. The object will contain simple properties such as name, age, and gender. Let's create another variable called `newJohn` and set it equal to the `john` object.

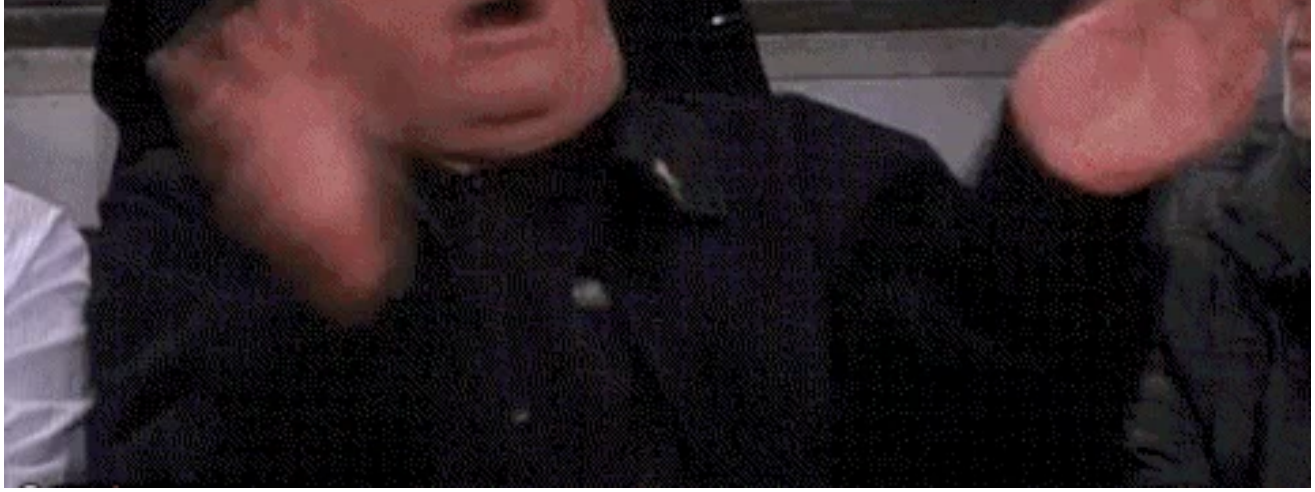
```
let john = {
        name: "John Smith",
        age: 45,
        gender: "male"
}

let newJohn = john
//{name: "John Smith", age: 45, gender: "male"}
```

PBRexampleOne.js hosted with ❤ by **GitHub** **view raw**

At this point, you are probably thinking that these are now two separate objects for which you can now modify in any way you want.





NOOOO!!!!!

Remember when I said that instead of making a copy, a value in memory will be referenced?

```
1  let john = {
2          name: "John Smith",
3          age: 45,
4          gender: "male"
5  }
6
7  let newJohn = john
8  //{name: "John Smith", age: 45, gender: "male"}
9
10 // Lets edit the name of newJohn variable
11
12 newJohn.name = "Tom Doe" // "Tom Doe"
13
14 console.log(newJohn)
15 //{name: "Tom Doe", age: 45, gender: "male"}
16
17 console.log(john)
18 // {name: "Tom Doe", age: 45, gender: "male"}
```

PBRexampleOne_2.js hosted with ❤ by **GitHub** **view raw**

The `newJohn` object is NOT a copy of the object stored inside of the `john` variable. Instead, the variable `newJohn` is referencing the original `john` object.

That means if you decide to edit the `newJohn` object (e.g changing the `name` property of `newJohn` on line 3 to `"Tom Doe"`), you are in fact modifying the `john` object!

As you can see in lines 16 and 19, both objects' `name` property has a value of `"Tom Doe"`.

Arrays

Arrays work the same way as objects.

We will get the same result as we did with the object example.

We will create a variable called `john` and initialize it to an array. We will then create a new variable called `newJohn` and assign it the value of `john`.

However, when we try to add a new item to the array stored in the `newJohn` variable, it will also affect the `john` array. Again, this is because we are only dealing with ***one array and NOT two separate copies of an array***. The variable `newJohn` is referencing the same array that was initialized in the variable `john`.

This is not the case when dealing with values that are pass-by-value.

Pass-by Value

Unlike objects and arrays, primitive values such as numbers or strings will actually create a copy.

Here, we create a variable `const num` and assign it the value of `25`. We will also create a function called `passByValue` that takes an argument and adds 2 to that argument. The parameter that we will pass in is the `num` variable.

As we saw with pass-by-reference, when we tried to mutate the object in `newJohn`, we were essentially referencing the object stored in the variable `john` and changing the object. With pass-by-value, when we pass in the variable `num` to the function and return the value of `27`, *WE ARE NOT CHANGING* the original value of the variable `num`.

This is because when we pass `num` into the function, it creates a copy and it is given its own space in memory.

When we `console.log(num)` on line 14, we still have our original `25`.

The value `const num` is not changed!

Conclusion

Understanding PBV vs PBR is helpful because if you are writing code and you want to set one variable equal to another variable, or in the example above, pass a variable into a function as an argument, you need to know whether the value will be pass-by-value or pass-by-reference. This will help you avoid errors when working with your data.

Programming JavaScript Coding