Understanding “this” Keyword

4 min read · Oct 26, 2024

Omkar Bhavare Follow

The `this` keyword in JavaScript is context-dependent, meaning that its value changes based on how and where it is used. It typically refers to the object that is currently executing the code.

👉🏻 **Global Context:** In the global scope, `this` refers to the global object. In browsers, this is the `window` object.

```
console.log(this); // In browser: window
```

👉 **Inside an Object’s Method:** When `this` is used inside an object's method, it refers to the object itself.

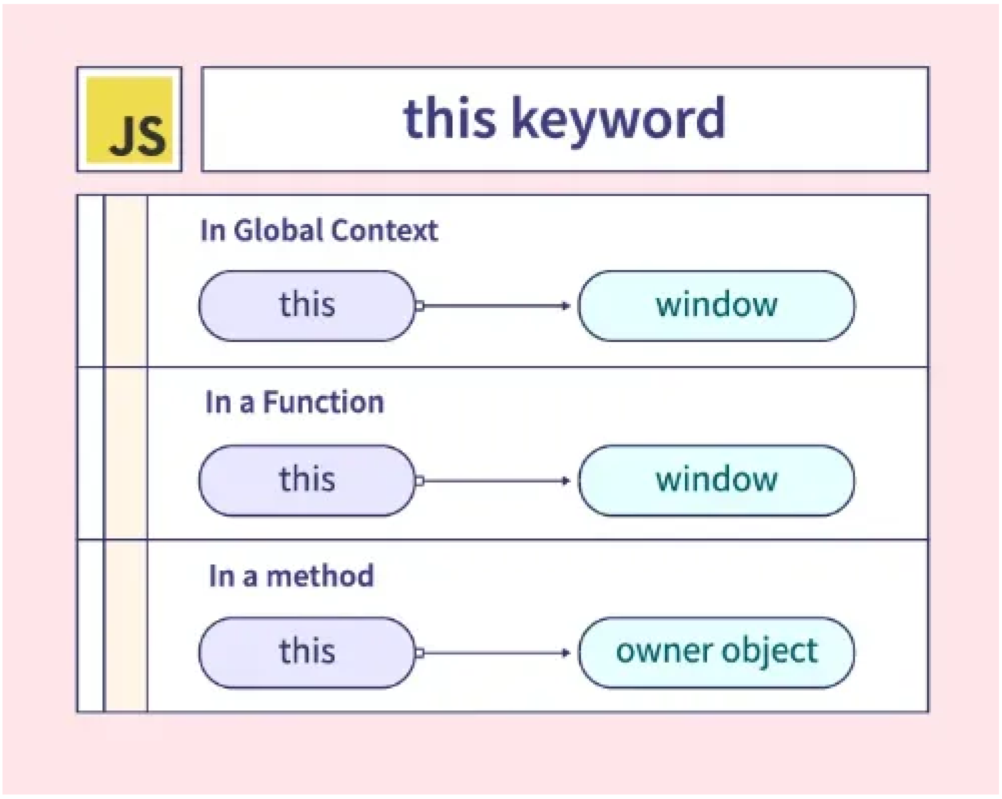
```
const obj = {
  name: 'Omkar',
  greet() {
    console.log(this.name);
  }
};
obj.greet(); // Outputs: 'Omkar'
```

👉 **In a Function (Non-Strict Mode):** When a function is called without any object context, `this` refers to the global object (`window` in browsers).

```
function showThis() {
  console.log(this);
}
showThis(); // In browser: window
```

👉 **In Strict Mode:** In strict mode (`'use strict'`), `this` inside a function will be `undefined` if it is not bound to any object.

```
'use strict';
function showThis() {
  console.log(this); // Outputs: undefined
}
showThis();
```



⭐ `"this"` in Arrow Functions :

Arrow functions behave differently from regular functions with respect to `this`. Arrow functions **do not bind their own `this`**, instead, they inherit `this` from the surrounding lexical context.

Lexical Scoping of `this` : The value of `this` inside an arrow function is determined by the scope in which the arrow function is created, not where it is called. [Arrow function inherit “this” from their surrounding context]

```
age = 26;
this.rollNo =54;
let school= "NMV";
const obj = {
  name: 'Omkar',
  greet: () => {
    console.log(age); // 26  (age is logged as 26 because it is a variable in t
    console.log(this.rollNo); // 54 ( WIndow Object )
    console.log(school); // NMV (present in outer scope)
    console.log(this); // { rollNo: 54 } ( In browser: window this logs the glc
    console.log(this.name); // < empty string >(because `this` is inherited frc
  }
};
obj.greet();
```

Contrast with Regular Functions : [In regular function “this” binds to the object calling the function]

```
age = 26;

const obj = {
  name: 'Omkar',
  greet() {
    console.log(this.name); // 'Omkar'
    console.log(age); // 26
  }
};
obj.greet();
```

⭐Object and `this`:

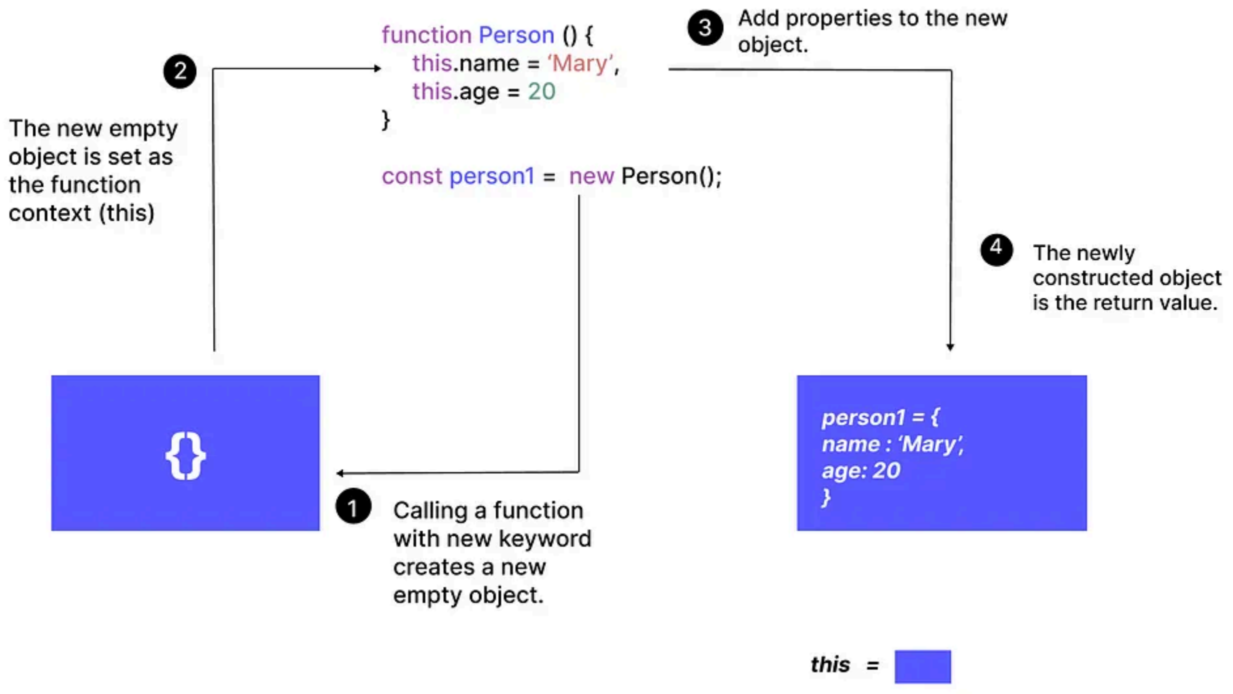
When a function is called as a method on an object, `this` refers to the object from which the method was called.

```
this.age=54;
const school="NMV"
const person = {
  name: 'Omkar',
  sayName: function() {
    console.log(this); // { name: 'Omkar', sayName: [Function: sayName] }
    console.log(this.name); // Omkar
    console.log(this.age);  // undefined
    console.log(this.school);  // undefined
  }
};
person.sayName();
```

```
this.age = 26;
const school = "NMV";
const person = {
  name: 'Omkar',
  sayName: () => {
    console.log(this); // { age : 26 }
    console.log(this.name); // undefined
    console.log(this.age);  // 26
    console.log(this.school);  // undefined
  }
};
person.sayName();
```

⭐ Constructor Function and "`this`":

A constructor function in JavaScript is used to create objects, and when used with the `new` keyword, `this` refers to the new object being created.


function Person () {
this.name = 'Mary',
this.age = 20
}
const person1 = new Person();
2 The new empty object is set as the function context (this)
3 Add properties to the new object.
4 The newly constructed object is the return value.
1 Calling a function with new keyword creates a new empty object.
{}
person1 = {
name : 'Mary',
age: 20
}
this =

```
function Person(name) {
  this.name = name;
  this.age = 14;
  const rollNo = 33;
  school = "NMV";
}
const person1 = new Person('Omkar');

console.log(person1); // Person { name: 'Omkar', age: 14 }

// rollNo , school is not assigned to 'this' and Hence not accessible outside t
console.log(person1.rollNo); // Outputs: undefined
```

⭐ Result of accessing it’s ref

```
name = "omkar"
function makeUser(){
    const name ="vijay";
    const age=33;
    this.school = "NMV"

    return {
        name: 'om',
        ref: this
    }

}

let user = new makeUser();

console.log(user); // { name: 'om', ref: makeUser { school: 'NMV' } }

console.log(user.ref); // makeUser { school: 'NMV' }

console.log(user.ref.school); // NMV
```

```
const name = "omkar"
function makeUser(){
     this.name ="vijay";
    const age=33;
    this.school = "NMV"
    return {
        name: 'om',
        ref(){
          return this;
        }
    }
}



const newUser = new makeUser();

console.log(newUser); // { name: 'om', ref: [Function: ref] }

console.log(newUser.ref()); // { name: 'om', ref: [Function: ref] }

console.log(newUser.ref().name) // om
```

Overall Summary:

- The `this` keyword in JavaScript is context-dependent, referring to different objects based on how and where the function is invoked.
- Arrow functions inherit `this` from their surrounding context, while regular functions bind `this` to the object calling the function.
- Constructor functions use `this` to reference the new object they are creating.
- Methods inside objects use `this` to refer to the object.

Reference Videos: Javascript Interview Questions (‘this’ Keyword)

📚 **Earlier Post:**

👉 React Js Fundamentals
👉 Javascript Fundamentals

🤝 Let’s connect on LinkedIn: https://www.linkedin.com/in/omkarbhavare/

JavaScript · Web Development · Software Development · Development · Javascript Tips

Follow

Written by Omkar Bhavare

46 followers · 6 following

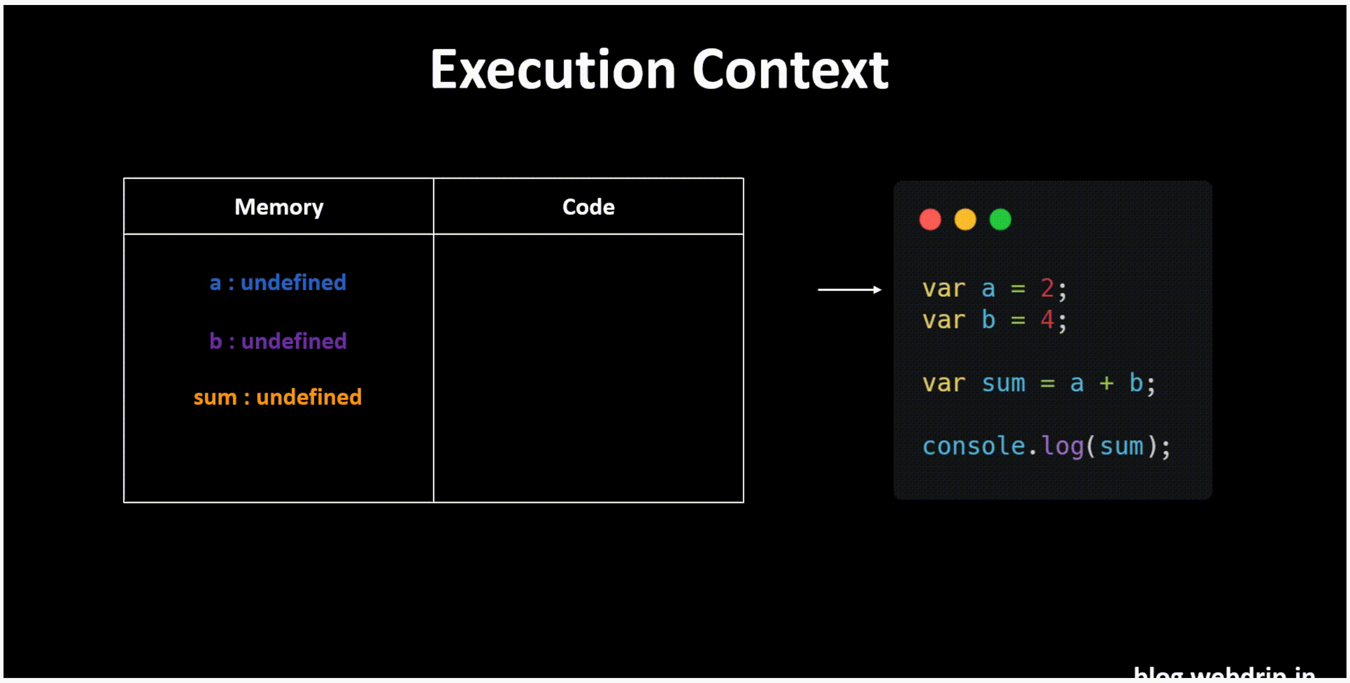
Passionate developer, blogger, and problem solver. Crafting code, sharing insights, and building innovative projects. 🚀

No responses yet

sahil siddiqui

What are your thoughts?

More from Omkar Bhavare



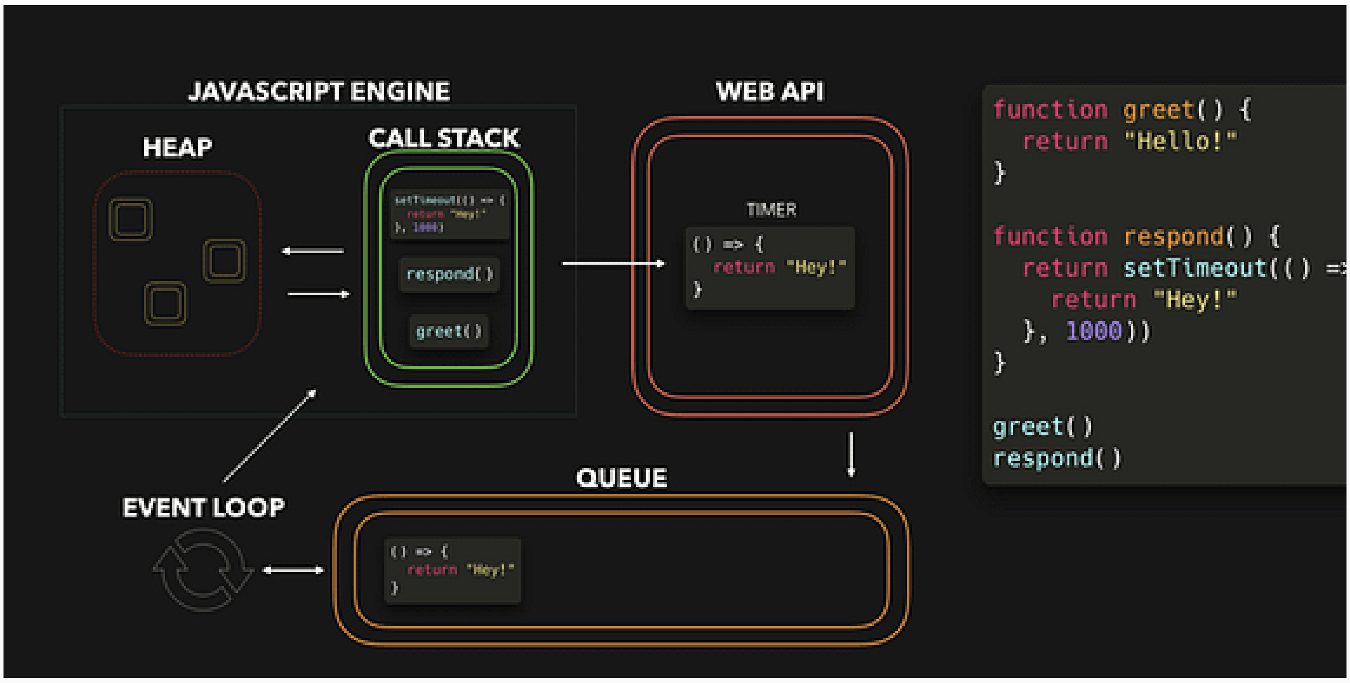
```
var a = 2;
var b = 4;

var sum = a + b;

console.log(sum);
```

Omkar Bhavare

Understanding the Execution Context and Call Stack in JavaScript

When we are working with JavaScript, there’s a lot happening under the hood. One important thing to get the hang of is the Execution…

Jan 2, 2024 43 1



```
setTimeout(() => {
  return "Hey!"
}, 1000)
```

```
respond()
greet()
```

```
() => {
  return "Hey!"
}
```

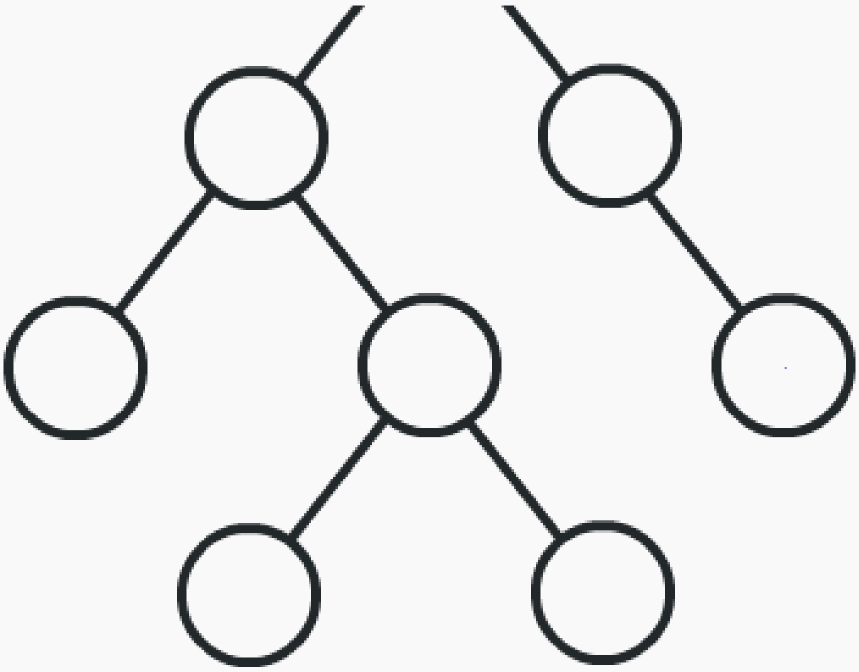
```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() =
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

Omkar Bhavare

JavaScript Event Loop, Web APIs, and Task Queues Explained

JavaScript, being single-threaded, processes one operation at a time, yet we see it handle various asynchronous operations efficiently...

Oct 26, 2024 16 1

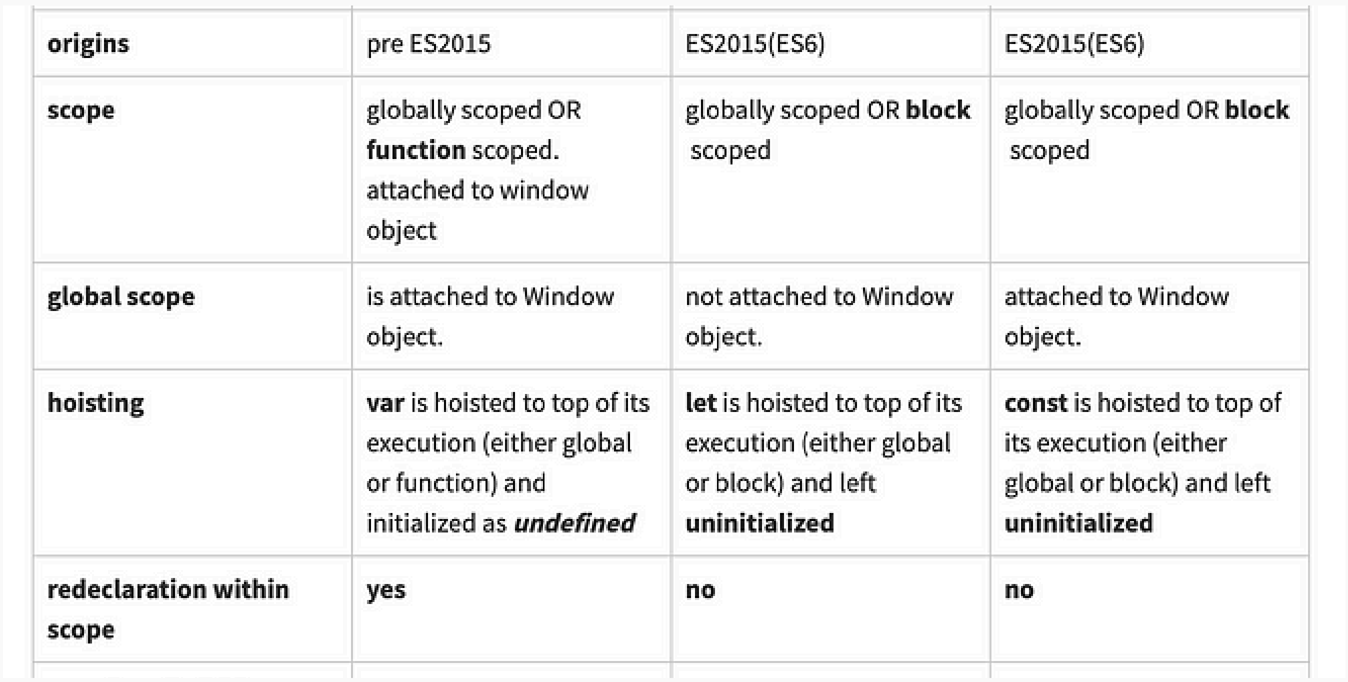


Omkar Bhavare

Prop Drilling in React

You know React, that well-liked JavaScript toolbox for designing interfaces? It uses a system of parts called “components”. This lets...

Dec 23, 2023 8 1



origins	pre ES2015	ES2015(ES6)	ES2015(ES6)
scope	globally scoped OR **function** scoped. attached to window object	globally scoped OR **block** scoped	globally scoped OR **block** scoped
global scope	is attached to Window object.	not attached to Window object.	attached to Window object.
hoisting	**var** is hoisted to top of its execution (either global or function) and initialized as ***undefined***	**let** is hoisted to top of its execution (either global or block) and left **uninitialized**	**const** is hoisted to top of its execution (either global or block) and left **uninitialized**
redeclaration within scope	**yes**	**no**	**no**

Omkar Bhavare

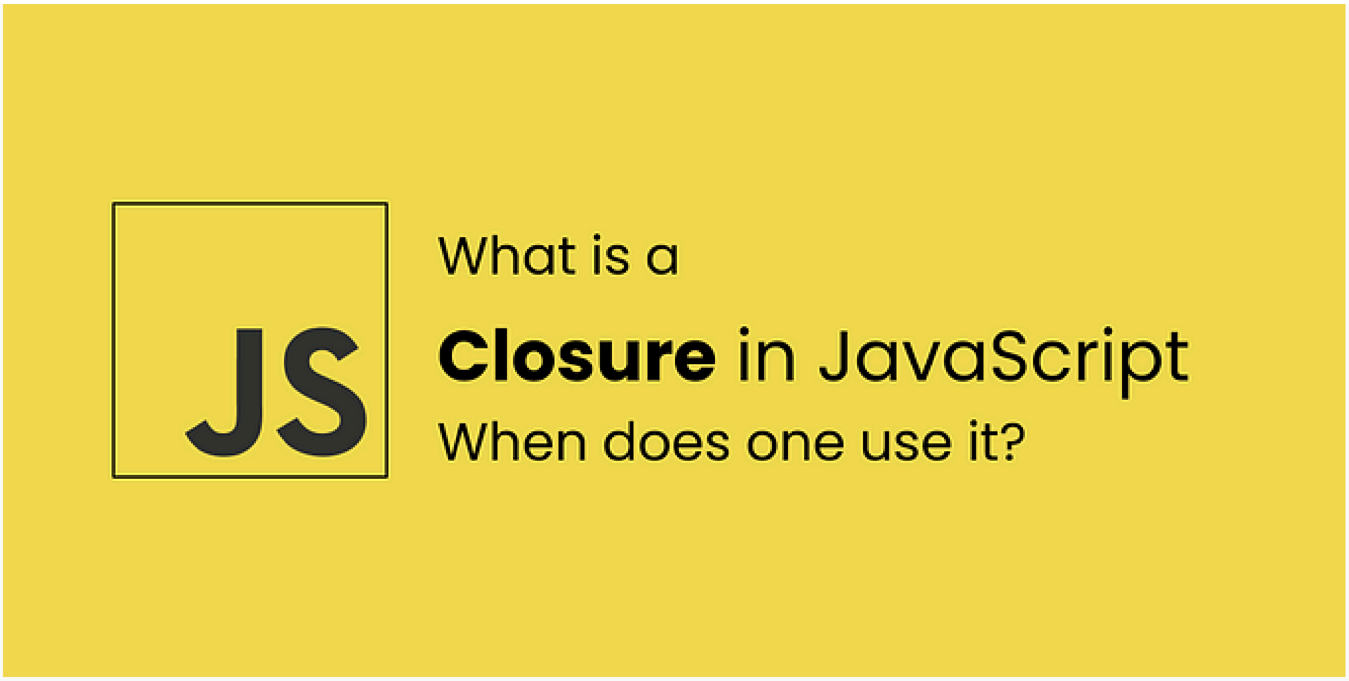
Javascript Fundamentals

JavaScript, as a versatile and dynamic programming language, is crucial for both beginners and experienced developers. In this article…

Jan 1, 2024 31 1

See all from Omkar Bhavare

Recommended from Medium

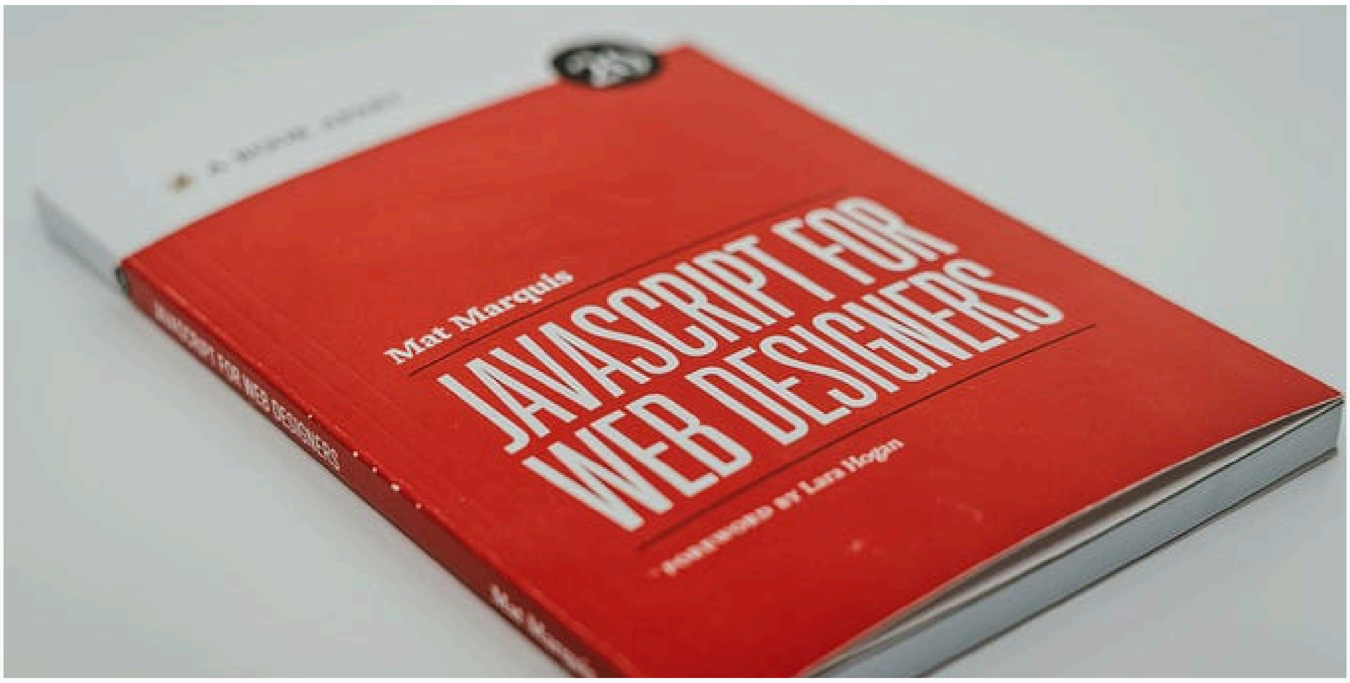


Apoorv Bedmutha

What is a Closure in JavaScript? When does one use it?

a closure gives a function access to its outer scope.—MDN Docs

Feb 19

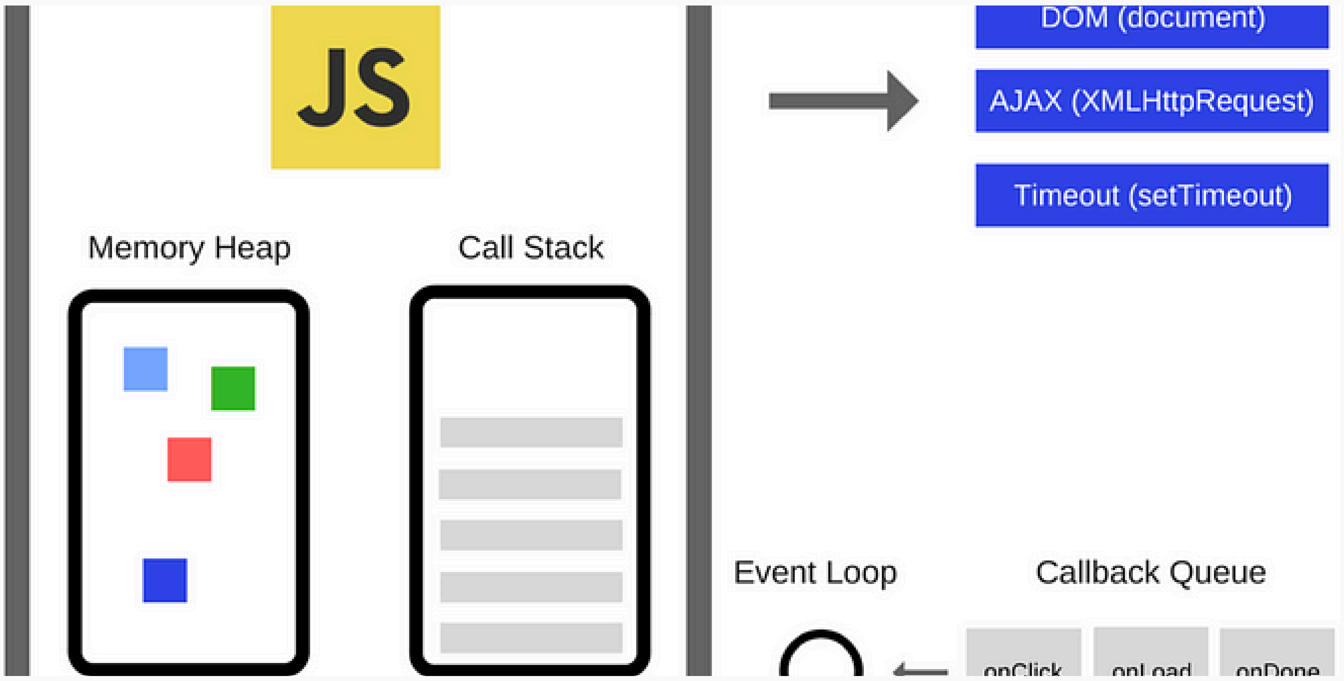


Swetha Kattukota

Javascript tricky coding interview questions 2025

1.What will be the output of the following code?

Apr 24 101 4



Dzmitry Ihnatovich

The JavaScript Event Loop Explained with Examples

The event loop is a core concept in JavaScript that enables non-blocking, asynchronous behavior. Understanding how the event loop works is...

Dec 16, 2024 88 5

Abhi Kshirsagar

JavaScript Array Methods: Understanding map, filter, and reduce

The `map`, `filter`, and `reduce` methods in JavaScript are array functions that help manipulate and process data. Each serves a distinct…

Nov 21, 2024 103 1

Aayushpatniya

Tricky Advanced JavaScript Interview Questions—Episode 5

Welcome to Episode 5 of my "Tricky Advanced JavaScript Interview Questions" series. Let's discuss and learn more advanced problems today.

Feb 19 2

habtesoft

Global Variables in JavaScript: Understanding and Best Practices

In JavaScript, global variables are variables that are accessible from any part of your code, regardless of where they are declared...

Nov 13, 2024 38 1

See more recommendations