## **this Keyword & Binding in JavaScript**

The this keyword refers to the object that is **executing the current function**. However, its behavior **depends on how the function is called**, meaning it can point to different objects in different contexts.  
In global execution, it typically refers to the global object (window in browsers, global in Node.js), while inside an object method, it refers to that specific object. In strict mode, this remains undefined in functions that are called without an explicit owner.   
  
**Strict Mode ("use strict")**

Strict Mode ("use strict") is a way to enforce stricter rules in JavaScript, preventing common mistakes, like accidental **global variable declarations** anddisallows using **reserved keywords** for variables.

## **🔹 How this Behaves in Different Contexts**

### **1️⃣ Global Context (window or undefined in strict mode)**

console.log(this); // In browsers: `window` (global object)

"use strict";

console.log(this); // `undefined` in strict mode

✅ **Explanation:**

* Outside any function, this refers to the global object (window in browsers, global in Node.js).
* In **strict mode**, this is undefined.

### **2️⃣ Function Context (this depends on how the function is called)**

function showThis() {

console.log(this);

}

showThis(); // ❓ (window in non-strict, undefined in strict mode)

✅ **Explanation:**

* In non-strict mode, this refers to window.
* In **strict mode**, this is undefined.

### **3️⃣ Object Methods (this refers to the object)**

const obj = {

name: "Alice",

greet() {

console.log(`Hello, ${this.name}`);

},

};

obj.greet(); // ❓ ("Hello, Alice")

✅ **Explanation:**

* this inside a method refers to the object before the dot (obj).

## **🔹 Implicit vs. Explicit Binding**

### **4️⃣ Implicit Binding (this refers to the calling object)**

const person = {

name: "Bob",

sayHi() {

console.log(this.name);

},

};

person.sayHi(); // ❓ ("Bob")

✅ **Explanation:**

* this refers to person, because it's the object before the dot.

### **5️⃣ Explicit Binding (call, apply, bind)**

#### **call() and apply()**

function introduce(age) {

console.log(`My name is ${this.name} and I am ${age} years old.`);

}

const user = { name: "Charlie" };

introduce.call(user, 25); // ❓ ("My name is Charlie and I am 25 years old.")

introduce.apply(user, [30]); // ❓ ("My name is Charlie and I am 30 years old.")

✅ **Explanation:**

* call() invokes a function with this explicitly set to user, passing arguments individually.
* apply() works the same, but accepts arguments as an array.

#### **bind() (Returns a New Function)**

const boundFunc = introduce.bind(user, 28);

boundFunc(); // ❓ ("My name is Charlie and I am 28 years old.")

✅ **Explanation:**

* bind() **doesn't call the function immediately**, it **returns a new function** with this permanently set.

## **🔹 Arrow Functions & this**

const obj2 = {

name: "David",

regularFunc: function () {

console.log(this.name); // ❓ ("David")

},

arrowFunc: () => {

console.log(this.name); // ❓ undefined/window (inherits from global)

},

};

obj2.regularFunc();

obj2.arrowFunc();

✅ **Explanation:**

* **Regular functions** take this from the calling object.
* **Arrow functions** **do not have their own this**; they inherit this from the surrounding scope i.e, the global scope that could be undefined in strict mode or window.name otherwise