

Below is a **simplified, exam-ready, and concept-clear** explanation of the `this` keyword in JavaScript, written in the **same depth + format** you've been following, but **cleaner and easier to remember**.

## 🔑 What is the `this` Keyword in JavaScript?

In JavaScript, `this` refers to the object that is currently executing the code.

👉 In simple words:

`this` = "who is calling the function?"

It allows you to:

- Access **properties**
- Call **methods**
- Refer to the **current execution context**

## 🧠 Core Rule (MOST IMPORTANT)

The value of `this` is decided at runtime, not at declaration time.

It depends on **how a function is called**, not where it is written.

## 📌 Syntax

```
js
```

```
this.property
```

```
this.method()
```

## 📌 Where Does `this` Point To?

The value of `this` changes based on **context**.

### 1 `this` in Global Scope

#### 👉 Non-Strict Mode

In the global scope, `this` refers to the **global object**.

- Browser → `window`
- Node.js → `global`

## Example

```
html
```

```
<script>
```

```
var num = 10;
```

```
console.log(this.num); // 10
</script>
```

✓ `this === window`

## 2 `this` Inside a Normal Function

### 👉 Non-Strict Mode

Inside a regular function, `this` still refers to the **global object**.

### Example

```
html

<script>
  var message = "Hello World";

  function show() {
    console.log(this.message);
  }

  show(); // Hello World
</script>
```

✓ `this === window`

## 3 `this` in Strict Mode

In **strict mode**, `this` becomes **undefined** inside normal functions.

### Example

```
html

<script>
  "use strict";

  function test() {
    console.log(this);
  }

  test(); // undefined
</script>
```

📌 This avoids accidental use of the global object.

## 4 `this` Inside an Object Method

When a function is called as a **method of an object**,

`this` refers to **that object**.

### Example

```
html

<script>
const fruit = {
  name: "Apple",
  color: "Red",
  show() {
    console.log(this.name, this.color);
  }
};

fruit.show(); // Apple Red
</script>
```

✓ `this === fruit`

## 5 `this` in Constructor Function

When a function is used with the `new` keyword:

👉 `this` refers to the **newly created object**

### Example

```
html

<script>
function Animal(name, age) {
  this.name = name;
  this.age = age;
}

const dog = new Animal("Dog", 5);
console.log(dog.name); // Dog
</script>
```

✓ `this === newly created object`

## 6 `this` in Arrow Functions ⚠️ (Very Important)

Arrow functions **do NOT** have their own `this`.

👉 They inherit `this` from their parent scope (lexical binding).

## Example: Arrow Function Inside Object

```
html

<script>
const wall = {
  color: "Blue",
  getDetails() {
    const arrowFn = () => {
      console.log(this.color);
    };
    arrowFn();
  }
};

wall.getDetails(); // Blue
</script>
```

✓ Arrow function borrows `this` from `getDetails()`

✓ `this === wall`

## ✗ Arrow Function as Object Method (Common Mistake)

```
js

const obj = {
  name: "JS",
  show: () => {
    console.log(this.name);
  }
};

obj.show(); // undefined
```

✗ Arrow functions should **not** be used as object methods.

## 7 `this` in Nested Functions (Problem Case)

A normal function inside a method loses object context.

## Example

html

```
<script>
const person = {
  name: "Salman",
  show() {
    function inner() {
      console.log(this);
    }
    inner();
  }
};

person.show(); // window
</script>
```

📌 Reason: `inner()` is a **normal function call**

### ✓ Solution

- Use arrow function
- Or store `this` in a variable
- Or use `bind()`

## 8 `this` in Event Handlers

In **HTML event handlers**, `this` refers to the **HTML element**.

### Example

html

```
<div onclick="this.style.display='none'"
  style="width:200px;height:100px;background:red;">
</div>
```

✓ `this === clicked element`

## 9 Explicit Binding ( `call`, `apply`, `bind` )

JavaScript allows **manual control** of `this`.

♦ `call()`

Calls function immediately, arguments separated by commas.

js

```
func.call(thisArg, a, b)
```

## Example

js

```
lion.printDetails.call(tiger);
```

### ◆ apply()

Same as `call()`, but arguments in array.

js

```
func.apply(thisArg, [a, b])
```

### ◆ bind()

Returns a **new function** with fixed `this`.

js

```
const newFn = func.bind(thisArg);  
newFn();
```

## Example

html

```
<script>  
const lion = {  
  color: "Yellow",  
  age: 10,  
  show() {  
    console.log(this.color, this.age);  
  }  
};  
  
const tiger = {  
  color: "Orange",  
  age: 15  
};  
  
const tigerShow = lion.show.bind(tiger);
```

```
tigerShow(); // Orange 15
```

```
</script>
```

## `this` Precedence Order (VERY IMPORTANT FOR EXAMS)

When multiple rules apply, JavaScript follows this order:

1 `bind()`

2 `call()` / `apply()`

3 Object method

4 Global scope

## One-Line Memory Trick

“`this` depends on who calls the function, not who writes it.”

## ✓ Final Exam Summary

- `this` refers to **execution context**
- Determined **at runtime**
- Arrow functions **do not have their own** `this`
- Constructors bind `this` to new object
- `call`, `apply`, `bind` explicitly set `this`
- Strict mode makes `this` → `undefined` in functions

If you want next:

- 🔥 `this` interview traps
- 🖋️ Output-based questions
- 🔄 `this` vs arrow function deep comparison
- 🧠 Real-world use cases

Just say **next** 🚀