`this` Behavior in Different Contexts

1. `this` in Global Scope

```
console.log(this); // globalObject - window (browser) or global (Node.js)
```

2. `this` Inside a Function

```
function a() {
    console.log(this); // Window (non-strict mode)
}

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

- In non-strict mode, `this` defaults to `window`.
- In strict mode, `this` is `undefined`.

3. `this` Inside an Object's Method

```
let student = {
    name: "shaik",
    PrintName: function() {
        console.log(this); // student object
        console.log(this.name); // "shaik"
    }
};
student.PrintName();
```

- Inside an object method, `this` refers to the object itself.

4. `this` Inside an Object's Arrow Function

```
let student = {
    name: "shaik",
    PrintName: () => {
        console.log(this); // window (or global)
    }
};
student.PrintName();
```

- Arrow functions don't have their own `this`, so they inherit from the enclosing lexical scope (global in this case).

5. `this` Inside a Nested Arrow Function

```
let student = {
    name: "shaik",
    PrintName: function () {
        let student2 = () => {
            console.log(this); // student
        };
        student2();
    }
};
student.PrintName();
```

- `this` inside a nested arrow function refers to the parent function's `this` (lexical scoping).

6. Call, Apply, and Bind (Sharing Methods)

```
function z() {
    console.log(this); // window
}

let student1 = {
    name: "MD",
};

let student = {
    name: "shaik",
    PrintName: function() {
        console.log(this);  // `this` will be set by call/apply/bind
    }
};

student.PrintName.call(student1);

let student2 = {
    name: "MD",
};

function x() {
    console.log(this); // student2
}
x.call(student2);
```

- `call` , `apply` , and `bind` explicitly set `this` .

Summary

- In the **global scope**, `this` is the global object (`window` in browsers, `global` in Node.js).
- In a **regular function**, `this` is `undefined` in strict mode and `window` otherwise.
- Inside an **object's method**, `this` refers to the object.
- In an **arrow function**, `this` is inherited from the lexical scope.
- `call` , `apply` , and `bind` are used to explicitly set `this` .

Execute Output

```
function a() {
    console.log(this); // Window
}
function b() {
    'use strict';
    console.log(this); // undefined
}
a();
b();

let student = {
    name: "shaik",
    PrintName: function() {
        console.log(this); // student object
    }
};
student.PrintName();

let student1 = {
    name: "shaik",
    PrintName: () => {
        console.log(this);  // window
    }
};
student1.PrintName();

let student2 = {
    name: "shaik",
    PrintName: function () {
        let student2 = () => {
            console.log(this); // student
        };
        student2();
    }
};
student2.PrintName();
```