1. New Keyword Binding

✅ When a function is invoked using `new`, it creates a **new object**, and `this` refers to that new object.

```
function Person(name) {
  this.name = name;
}

const p1 = new Person("Alice");
console.log(p1.name); // Output: Alice
```

Explanation: Using `new` creates a new object and binds `this` to that object (`p1` here).

---

2. Implicit Binding

✅ When a function is called as a method on an object, `this` refers to the object **left of the dot**.

```
const user = {
  name: "Bob",
  greet() {
    console.log("Hi, I'm " + this.name);
  }
};

user.greet(); // Output: Hi, I'm Bob
```

Explanation: `this` refers to the object `user` because `greet()` was called with `user.`

---

3. Explicit Binding (with `bind`, `call`, or `apply`)

✅ You **manually set** the value of `this` using `.bind()`, `.call()`, or `.apply()`.

```
function sayHello() {
  console.log("Hello, " + this.name);
}

const person = { name: "Carol" };

const boundHello = sayHello.bind(person);
boundHello(); // Output: Hello, Carol
```

Explanation: `bind(person)` sets `this` inside `sayHello()` to refer to `person`.

---

4. Arrow Functions as Methods

⚠ Arrow functions **don't have their own `this`**. They inherit it from their **lexical scope** (the surrounding scope).

```
const obj = {
  name: "Dave",
  greet: function () {
    const arrowFunc = () => {
      console.log("Hello, " + this.name);
    };
    arrowFunc();
  }
};

obj.greet(); // Output: Hello, Dave
```

Explanation: Arrow function doesn't get its own `this`, it **inherits** `this` from `greet()`, which is bound to `obj`.

⚠ Problem with arrow function inside a regular function:

If we go deeper and `this` is lost (e.g., nested functions), we can fix it with arrow functions.

```
const person = {
  name: "Eva",
  greet: function () {
    function innerFunc() {
      console.log(this.name); // undefined (because `this` is now window/global)
    }
    innerFunc();
  }
};

person.greet(); // Output: undefined
```

☑ Fix using arrow function:

```
const person = {
  name: "Eva",
  greet: function () {
    const innerFunc = () => {
      console.log(this.name); // this is lexically inherited from `greet`
    };
    innerFunc();
  }
};
```

```
person.greet(); // Output: Eva
```
