Core JavaScript Mastery Guide

Perfect 👍 You've already gathered the **theory of JavaScript Object Methods**.
Now let's **deep dive step by step** — from **basics → shorthand → adding/updating → built-in methods → advanced use cases & interview insights**.

📘 JavaScript Object Methods

1. What is a Method?

- An **object method** = a property whose **value is a function**.
- Example:

```js
const user = {
  name: "Anoop",
  greet: function() {
    return `Hello, my name is ${this.name}`;
  }
};

console.log(user.greet()); // Hello, my name is Anoop
```

✅ Here, `greet` is a **method** because it's a function inside an object.
✅ It uses `this` → refers to the object itself (`user`).

---

2. Shorthand Syntax (ES6)

Instead of writing `greet: function() {}`, you can use shorthand:

```js
const user = {
  name: "Anoop",
  greet() {
    return `Hello, I am ${this.name}`;
  }
};
```

```
console.log(user.greet());
```

✔️ Cleaner, shorter, widely used in modern JavaScript.

---

3. Adding/Updating Methods Dynamically

You can add or modify methods after the object is created:

```
js

const calculator = {
  add(a, b) {
    return a + b;
  }
};

// Adding new method
calculator.multiply = function(a, b) {
  return a * b;
};

console.log(calculator.add(2, 3));      // 5
console.log(calculator.multiply(2, 3)); // 6
```

---

4. Using Parameters in Methods

Methods can accept arguments and return values.

```
js

const math = {
  square(n) {
    return n * n;
  },
  cube(n) {
    return n * n * n;
  }
};
```

```
console.log(math.square(4)); // 16
console.log(math.cube(3));   // 27
```

---

5. `this` Keyword in Methods

Inside an object method, `this` refers to the **object**.

```
js

const person = {
  name: "Ravi",
  age: 25,
  info() {
    return `${this.name} is ${this.age} years old.`;
  }
};

console.log(person.info()); // Ravi is 25 years old.
```

⚠ BUT be careful: if you assign the method to a variable, `this` can lose context:

```
js

let f = person.info;
console.log(f()); // ❌ "undefined is undefined"
```

👉 Solution → use `bind()` to fix `this`:

```
js

let g = person.info.bind(person);
console.log(g()); // Ravi is 25 years old.
```

---

6. Built-in Object Methods

All JavaScript objects come with **built-in methods**. Examples:

```
js

// Working with Number
const num = 123;
```

```
console.log(num.toString());  // "123"

// Working with String
const str = "JavaScript";
console.log(str.toUpperCase()); // "JAVASCRIPT"

// Working with Arrays
const arr = [1, 2, 3];
console.log(arr.includes(2)); // true

// Working with Objects
const obj = { a: 1, b: 2 };
console.log(Object.keys(obj));   // ["a", "b"]
console.log(Object.values(obj)); // [1, 2]
```

---

7. Special Case: Arrow Functions as Methods

⚠ Avoid using arrow functions as object methods when you need `this`.

```
js

const person = {
  name: "Anoop",
  greet: () => {
    return `Hello, I am ${this.name}`;
  }
};

console.log(person.greet()); // "Hello, I am undefined"
```

👉 Reason: Arrow functions **don't bind their own** `this`; they inherit it from the surrounding scope.

---

8. Interview Questions on Object Methods

1. **What's the difference between a function and a method?**
   - Function → standalone
   - Method → function stored as an object property
2. **What does** `this` **refer to inside an object method?**

- It refers to the object itself.

3. **Why shouldn't you use arrow functions as methods?**
   - Because arrow functions don't have their own `this`.
4. **How do you add a new method to an existing object?**
   - Using `obj.newMethod = function() {...}`
5. **What's the difference between method shorthand and regular function property?**
   - Shorthand is cleaner (`greet() {}` instead of `greet: function() {}`).

✅ Now you know **everything from basics to advanced** about **Object Methods**. They are extremely important because **JavaScript OOP, classes, prototypes, and inheritance all rely on methods**.

👉 Do you want me to continue into **Object Built-in Utility Methods** (like `Object.assign`, `Object.create`, `Object.freeze`, etc.) or go next into **Object Prototypes & Inheritance**?