Core JavaScript Mastery Guide

Perfect 👍 Let's break **JavaScript Static Methods** down into a structured explanation (with examples + reasoning + use cases), so you fully understand them from **basic to advanced**.

🔹 Static Methods in JavaScript

1. Definition

- A **static method** is a method that belongs to the **class itself**, not to instances (objects) of the class.
- It is defined using the `static` keyword inside a class.
- You call it using the **class name**, not by creating an object.

👉 Think of static methods as **utility/helper functions** grouped under a class.

---

2. Basic Syntax

```js
class MyClass {
  static myMethod() {
    return "Hello, I am static!";
  }
}

// ✅ Correct way: call with class name
console.log(MyClass.myMethod());

// ❌ Wrong way: object cannot access static method
const obj = new MyClass();
// console.log(obj.myMethod()); // Error
```

---

3. Static vs Instance Method

- **Static method** → called on **class**
- **Instance (normal) method** → called on **object**

Example:

```js
class Table {
  static printSize() {
    return "Size: 20x20";
  }

  getColor() {
    return "Black";
  }
}

console.log(Table.printSize());   // ✅ Static
const t = new Table();
console.log(t.getColor());      // ✅ Instance
// console.log(t.printSize());    // ❌ Error
```

---

4. Multiple Static Methods

```js
class Table {
  static getSize() {
    return "20x20";
  }

  static getColor() {
    return "Brown";
  }
}

console.log(Table.getSize());   // 20x20
console.log(Table.getColor());  // Brown
```

⚠ If you declare **two static methods with the same name**, the **last one overrides** the earlier one.

---

5. Static Methods in Constructor

You can use a static method inside the constructor with:

```js
this.constructor.methodName()
```

Example:

```js
class Num {
  constructor(a) {
    this.square = this.constructor.getSquare(a); // call static inside constructor
  }

  static getSquare(a) {
    return a * a;
  }
}

const n1 = new Num(5);
console.log(n1.square); // 25
```

---

6. Calling Static Methods Inside Normal Methods

Use the **class name** to access:

```js
class Calculator {
  static multiply(a, b) {
    return a * b;
  }

  printMultiply(a, b) {
    return `Result: ${Calculator.multiply(a, b)}`;
  }
}
```

```js
const calc = new Calculator();
console.log(calc.printMultiply(4, 5)); // Result: 20
```

---

7. Real-Life Example (Utility Class)

Static methods are commonly used for **utility/helper functions** (like `Math` object):

```js
class MathHelper {
  static add(a, b) {
    return a + b;
  }

  static subtract(a, b) {
    return a - b;
  }

  static factorial(n) {
    if (n === 0) return 1;
    return n * MathHelper.factorial(n - 1);
  }
}

console.log(MathHelper.add(10, 5));       // 15
console.log(MathHelper.subtract(10, 5));  // 5
console.log(MathHelper.factorial(5));     // 120
```

---

8. Built-in Static Methods in JavaScript

- `Math.random()`
- `Math.floor()`
- `Object.keys(obj)`
- `Array.isArray(arr)`
- `Date.now()`

👉 Notice: We don't need to create an object. We directly call on the **class/object name**.

Example:

```js
console.log(Math.random());       // e.g., 0.382
console.log(Array.isArray([1,2])); // true
console.log(Date.now());          // timestamp
```

🔹 Advantages of Static Methods

✅ No need to create an object (memory-efficient).
✅ Used for utility/helper functions.
✅ Provide better performance (shared at class level).
✅ Keep code organized under a **namespace**.

🔹 Disadvantages

❌ Cannot access instance properties (`this.name`, etc.) directly.
❌ Can only be called using the **class name**.
❌ Too many static methods may make the class less OOP-oriented.

✨ In short:

- Use **instance methods** when behavior depends on object's data.
- Use **static methods** when behavior is independent of objects (general utility).

👉 Do you want me to prepare a **set of interview-style questions with answers on static methods** (like I did for closures)? That way you can test your understanding.