Core JavaScript Mastery Guide

Here's a structured summary of **Encapsulation in JavaScript** based on your detailed explanation, including examples and benefits:

🔹 Encapsulation in JavaScript

Definition:
Encapsulation is the practice of **bundling related properties and methods** together and **restricting direct access to some of an object's components**. This helps in **data protection, integrity, and security**.

In JavaScript, encapsulation can be implemented using:

1. **Function closures**
2. **ES6 classes with private fields**
3. **Getters and setters**

1️⃣ Need for Encapsulation

Without encapsulation, object properties are accessible and modifiable from anywhere:

```
js

const car = { Brand: "Honda City", model: "SX", year: 2016 };
car.Brand;       // Access allowed
car.Brand = true; // Modification allowed → Data integrity lost
```

Encapsulation **hides the data** and provides controlled access.

2️⃣ Ways to Achieve Encapsulation

A) Using Function Closures

Closures allow **private variables** that cannot be accessed from outside the function scope.

```js
function shoppingCart() {
  const cartItems = []; // private
  function add(item) {
    cartItems.push(item);
    console.log(`${item.name} added to cart.`);
  }
  function remove(itemName) {
    const index = cartItems.findIndex(item => item.name === itemName);
    if (index !== -1) cartItems.splice(index, 1);
  }
  return { add, remove }; // expose methods only
}

const cart = shoppingCart();
cart.add({ name: "Car" });
cart.add({ name: "Bike" });
cart.remove("Bike"); // private cartItems cannot be accessed directly
```

✅ **Benefit:** `cartItems` cannot be directly modified from outside.

B) Using ES6 Classes and Private Fields

ES6 classes allow **private fields** using `#`.

```js
class Car {
  #brand = "TATA";
  #name = "Nexon";
  #milage = 16;

  getMilage() { return this.#milage; }
  setMilage(milage) { this.#milage = milage; }
}

const carObj = new Car();
console.log(carObj.getMilage()); // 16
carObj.setMilage(20);
console.log(carObj.getMilage()); // 20
// carObj.#milage → Error: private field
```

✅ **Benefit:** Private data cannot be accessed or modified directly.

C) Using Getters and Setters

Getters allow controlled reading, and **setters** allow controlled modification of private data.

```
js

class User {
  #username = "Bob";

  get username() { return this.#username; }
  set username(name) { this.#username = name; }
}

const user = new User();
console.log(user.username); // Bob
user.username = "Alice";
console.log(user.username); // Alice
```

✅ **Benefit:** Allows **validation or logic** while accessing or updating private fields.

3️⃣ Benefits of Encapsulation

Benefit	Description
Data Protection	Restricts direct access to sensitive data and ensures integrity.
Code Reusability	Classes can be reused to create multiple objects with different data.
Code Maintenance	Easier to maintain since objects are independent and changes don't affect others.

✅ Key Takeaways

- Encapsulation **hides internal state** of objects and exposes only necessary methods.
- In JavaScript, it is achieved via **closures, ES6 private fields, and getters/setters**.
- Protects your code from **unintended modifications**, improving **security and maintainability**.

If you want, I can create a **visual diagram showing all three methods of encapsulation** in JavaScript along with **example flow**, which is really useful for exams or interviews.

Do you want me to create that diagram?