# Core JavaScript Mastery Guide

Perfect explanation 👍

Let me break this down into **exam-level clarity** for you, since this is a very important JavaScript topic.

---

## 🔹 JavaScript Object Accessor Properties (Getters & Setters)

In JavaScript, objects can have **two types of properties**:

1. **Data Properties** → Directly hold a value.

```
js

const obj = { key: "value" };
```

2. **Accessor Properties** → Defined using `get` and `set` methods to control access to values.

---

### 1. Getter (`get`)

- Used to **retrieve (read)** a property value.
- Defined using the `get` keyword.
- Accessed like a property (no parentheses needed).

✅ **Syntax:**

```
js

const obj = {
  property: "value",
  get propertyName() {
    return this.property;
  }
};
console.log(obj.propertyName); // calls getter
```

✅ **Example:**

```html
<script>
const wall = {
  color: "brown",
  get getColor() {
    return this.color;
  }
};
document.write("Wall color: " + wall.getColor);
</script>
```

📌 Output: `Wall color: brown`

---

## 2. Setter (`set`)

- Used to **update (write)** a property value.
- Defined using the `set` keyword.
- Always takes **exactly one parameter**.

✅ **Syntax:**

```js
const obj = {
  property: "value",
  set propertyName(newValue) {
    this.property = newValue;
  }
};
obj.propertyName = "new value"; // calls setter
```

✅ **Example:**

```html
<script>
const wall = {
  color: "brown",
  set setColor(newColor) {
    this.color = newColor;
  }
};
```

```
document.write("Before update: " + wall.color + "<br>");
wall.setColor = "red"; // setter called
document.write("After update: " + wall.color);
</script>
```

📌 Output:

```
pgsql

Before update: brown
After update: red
```

---

## 3. Methods vs. Getters/Setters

You can achieve the same with methods, but getters/setters give **cleaner syntax**.

```
js

const wall = {
  color: "brown",
  get getColor() { return this.color; },
  colorMethod() { return this.color; }
};

console.log(wall.getColor);     // clean syntax
console.log(wall.colorMethod()); // needs parentheses
```

---

## 4. Encapsulation & Data Quality

Getters/Setters allow:

- **Data validation** before setting a value.
- **Data hiding (abstraction)** → Users don't directly access properties.

✅ Example (uppercase getter):

```
js

const wall = {
  color: "Brown",
  get getColor() {
```

```
    return this.color.toUpperCase();
  }
};
console.log(wall.getColor); // "BROWN"
```

---

## 5. Define Getters/Setters with `Object.defineProperty()`

Another way to add accessors dynamically.

```
html

<script>
const door = { size: 20 };

Object.defineProperty(door, "getSize", {
  get: function() { return this.size; }
});

Object.defineProperty(door, "setSize", {
  set: function(value) { this.size = value; }
});

document.write("Before: " + door.getSize + "<br>");
door.setSize = 30;
document.write("After: " + door.getSize);
</script>
```

📌 Output:

```
makefile

Before: 20
After: 30
```

---

## 6. Why use Getters & Setters?

✔️ Cleaner syntax than methods
✔️ Improve **data quality** (validation possible)

✔️ Provide **encapsulation** (data hiding & abstraction)
✔️ Allow **extra logic** when getting/setting values

---

⚡ Quick Example with Validation:

```js
const student = {
  name: "Anoop",
  age: 20,
  set setAge(value) {
    if(value < 0) {
      console.log("Age cannot be negative!");
    } else {
      this.age = value;
    }
  },
  get getAge() {
    return this.age;
  }
};

student.setAge = -5;   // Invalid
student.setAge = 22;   // Valid
console.log(student.getAge); // 22
```

---

👉 Would you like me to also create a **table comparison (Methods vs Getters/Setters vs Data Properties)** so you can remember it easily for exams/interviews?