# Property descriptors and getters/setters

Let's explore **Property Descriptors** and **Getters/Setters** in JavaScript — tools that give you advanced control over object properties.

---

## 🔍 What Are Property Descriptors?

Every object property has hidden metadata, known as a **property descriptor**, which controls how that property behaves.

### Property descriptor attributes:

1. **value**: the actual data (for data properties)
2. **writable**: can the value be changed?
3. **enumerable**: will it show up in loops like `for...in`?
4. **configurable**: can the property be deleted or reconfigured?
5. **get/set**: functions for getter/setter behavior (for accessor properties)

---

## 🧪 Example: Viewing a Property Descriptor

```
const user = { name: "Abhi" };

console.log(Object.getOwnPropertyDescriptor(user, "name"));
```

### Output:

```
{
  value: 'Abhi',
  writable: true,
  enumerable: true,
```

```
  configurable: true
}
```

## ✍️ Defining Custom Property Descriptors

Use `Object.defineProperty()` to create or modify a property with custom settings:

```
const user = {};

Object.defineProperty(user, "name", {
  value: "Abhi",
  writable: false,      // cannot change value
  enumerable: true,
  configurable: false   // cannot delete or redefine
});

console.log(user.name); // Abhi
user.name = "John";     // ❌ No effect because writable is false
console.log(user.name); // Still "Abhi"
```

## 🧠 Accessor Properties: Getters and Setters

Instead of storing a value, you can use **functions** to get or set it dynamically.

### ✅ Getters

```
const user = {
  firstName: "Abhi",
  lastName: "Gurjar",
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  }
};
```

```
console.log(user.fullName); // Abhi Gurjar
```

> Note: You access user.fullName without parentheses — it looks like a property.

---

## ✅ Setters

```
const user = {
  firstName: "Abhi",
  lastName: "Gurjar",
  set fullName(name) {
    [this.firstName, this.lastName] = name.split(" ");
  }
};

user.fullName = "John Doe";
console.log(user.firstName); // John
console.log(user.lastName);  // Doe
```

---

## ✅ Defining Getters/Setters with `Object.defineProperty()`

```
const person = {
  firstName: "Abhi",
  lastName: "Gurjar"
};

Object.defineProperty(person, "fullName", {
  get() {
    return `${this.firstName} ${this.lastName}`;
  },
  set(value) {
    [this.firstName, this.lastName] = value.split(" ");
  }
});
```

```
console.log(person.fullName); // Abhi Gurjar
person.fullName = "John Smith";
console.log(person.firstName); // John
```

## 🔐 When to Use These?

- **Property descriptors**: when you need fine control over object behavior (like read-only or non-enumerable fields).
- **Getters/Setters**: when the property's value depends on other data, or you want to run logic during access or assignment.

## 🧾 Summary Table

| Feature | Description |
|---|---|
| `writable` | Can the value be changed? |
| `enumerable` | Shows up in loops? |
| `configurable` | Can it be deleted or changed? |
| `get()` | Runs code when property is **read** |
| `set()` | Runs code when property is **written** |

Let me know if you'd like a breakdown of **real-world use cases** (e.g., data validation, computed properties) or a mini project using getters/setters.