

Object creation methods (literal, constructor, Object.create)

Let's explore **Object Creation Methods** in JavaScript in detail:

1. Object Literal Syntax

This is the **simplest and most common** way to create an object.

```
const person = {  
  name: "Abhi",  
  age: 22,  
  greet: function () {  
    console.log(`Hello, my name is ${this.name}`);  
  }  
};  
  
person.greet(); // Hello, my name is Abhi
```

✓ Features:

- Quick and clean
 - Easy to read and write
 - Best for creating **single** objects
-

2. Object Constructor (new Object())

Creates an object using the built-in `Object` constructor.

```
const person = new Object();  
person.name = "Abhi";  
person.age = 22;  
person.greet = function () {
```

```
console.log(`Hello, I'm ${this.name}`);  
};
```

```
person.greet(); // Hello, I'm Abhi
```

✓ Features:

- More verbose than literal syntax
- Useful when working with dynamic object creation in certain cases

3. Constructor Functions

Allows you to create **multiple similar objects** with a shared structure.

```
function Person(name, age) {  
  this.name = name;  
  this.age = age;  
  this.greet = function () {  
    console.log(`Hi, I'm ${this.name}`);  
  };  
}
```

```
const p1 = new Person("Abhi", 22);  
const p2 = new Person("John", 25);
```

```
p1.greet(); // Hi, I'm Abhi  
p2.greet(); // Hi, I'm John
```

| 🔍 Functions used with `new` act like object blueprints (like classes).

4. Using `Object.create()`

Creates a new object with the specified **prototype object** and optional **properties**.

```
const proto = {
  greet() {
    console.log(`Hello, I'm ${this.name}`);
  }
};

const person = Object.create(proto);
person.name = "Abhi";

person.greet(); // Hello, I'm Abhi
```

✓ Features:

- Allows **fine-grained control** over inheritance
- Useful for **prototypal inheritance**
- You can also define **property descriptors**

5. ES6 Classes (Bonus)

Syntactic sugar over constructor functions.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello from ${this.name}`);
  }
}
```

```
const p = new Person("Abhi", 22);  
p.greet(); // Hello from Abhi
```

Comparison Table

Method	Inheritance	Suitable For	Simplicity
Object Literal	No	Single objects	✓ Easy
new Object()	No	Rarely used today	✗ Verbose
Constructor Function	No	Object templates	✓ Common
Object.create()	Yes	Prototypal inheritance	✓ Flexible
Class (ES6)	Yes	Modern, class-style coding	✓ Modern

Summary

- Use **object literals** for most simple object needs.
- Use **constructor functions or classes** for creating reusable object templates.
- Use **Object.create()** when you want full control over inheritance/prototypes.

Let me know if you'd like code examples comparing **Object.create vs constructor functions** or want to dive into **property descriptors** next.