JavaScript Inheritance

1. Prototypal Inheritance (Core Mechanism)

Every JavaScript object has an internal link to another object—its prototype—forming a prototype chain. Constructors define a .prototype object. Instances created with new inherit from this .prototype, enabling them to share methods and properties.

Example:

```
function Animal(name) { this.name = name; } Animal.prototype.speak = function () { console.log(this.name + ' makes a sound.'); }; function Dog(name) { Animal.call(this, name); } Dog.prototype = Object.create(Animal.prototype); Dog.prototype.constructor = Dog; Dog.prototype.bark = function () { console.log(this.name + ' barks.'); };
```

2. ES6 Class-Based Syntax (Syntactic Sugar)

Introduced in ES6, the class syntax abstracts the prototypal mechanics but still uses them under the hood. The extends keyword sets up the prototype chain, and super() ensures the parent class's constructor runs correctly.

Example:

```
class Car { constructor(brand) { this.carname = brand; } present() { return 'I have a ' + this.carname; } } class Model extends Car { constructor(brand, mod) { super(brand); this.model = mod; } show() { return this.present() + ', it is a ' + this.model; } } const myCar = new Model("Ford", "Mustang"); console.log(myCar.show()); // "I have a Ford, it is a Mustang"
```

3. Prototype Chain Visualization

Think of inheritance like DNA: child objects inherit traits (properties/methods) from parent prototypes, and so on up the chain. Modifying prototypes (like adding methods to builtin objects) is possible but typically discouraged.

Summary Table

Inheritance Style	Mechanism
Prototypal (ES5-style)	Manual prototype chain manipulation via constructor functions + Object.create()
Class Syntax (ES6+)	Cleaner syntax (class, extends, super) built on the same prototype model