# **Understanding the Prototype Chain in JavaScript**

The prototype chain is a core JavaScript concept enabling the inheritance of properties and methods between objects. It facilitates code reuse, efficient property lookup, and object hierarchy creation.

- Every JavaScript object has an internal link to another object, called its prototype.
- The prototype chain forms when objects inherit properties and methods from their prototypes.
- Property or method access starts from the object itself and traverses up the chain if not found.
- The chain ends at null, the prototype of Object.prototype.

```
const parent = { greet: () => "Hello!" }; ‡ x ▷ □
const child = Object.create(parent);
```

#### In this example

- parent is the prototype of child.
- child inherits the greet method from parent.
- The chain consists of child → parent → Object.prototype → null.

# **Syntax**

```
function ConstructorName(params) {
   // Initialization code
}

ConstructorName.prototype.methodName = function () {
```

```
// Method code
};
```

### Real-World Use Cases

# **Extending Built-in Objects**

### Output

```
6
```

### **Custom Object Hierarchies**

```
const vehicle = {
1
                                                      X D G
         start() {
2
             console.log("Engine started.");
3
         },
4
    };
5
    const car = Object.create(vehicle);
6
    car.drive = function () {
         console.log("Car is driving.");
8
    };
9
10
    car.start();
11
    car.drive();
12
```

# Output

```
Engine started.
Car is driving.
```

# **Checking Property Existence**

```
const car = {
    wheels: 4
};
const myCar = Object.create(car);
myCar.color = "red";
```

#### Output

```
true
true
false
```

### **Prototype Chain Traversal**

```
function fun(obj) {
                                                     × ▷ ⑤
         let current = obj;
2
        while (current) {
3
             console.log(current);
             current = Object.getPrototypeOf(current);
5
        }
6
7
    }
    const animal = { eats: true };
    const mammal = Object.create(animal);
9
    mammal.hasFur = true;
10
    const dog = Object.create(mammal);
11
    dog.barks = true;
12
    fun(dog);
13
```

### Output

```
{ barks: true }
{ hasFur: true }
{ eats: true }
[Object: null prototype] {}
```

# **Prototype Method Overriding**

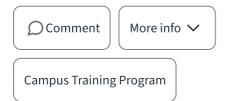
```
function Vehicle() { }
                                                             0
    Vehicle.prototype.start = function () {
2
         return "Vehicle is starting";
3
    };
4
    function Car() { }
    Car.prototype = Object.create(Vehicle.prototype);
6
    Car.prototype.constructor = Car;
7
    Car.prototype.start = function () {
         return "Car is starting";
9
    };
10
    const myCar = new Car();
11
    console.log(myCar.start());
12
```

#### Output

```
Car is starting
```

### Advantages of the Prototype Chain

- Code Reusability: Shared properties and methods reduce redundancy.
- Efficient Memory Usage: Shared prototypes lower memory consumption.
- Dynamic Behavior: Prototypes can be extended at runtime.
- Scalable Object Hierarchies: Simplifies creation and management of relationships between objects.



Next Article >

Nesting For Loops in JavaScript

# Similar Reads

# **JavaScript Prototype**