🔄 **Frontend Interview Series 9/600: Prototypal Inheritance in JavaScript**

Question: How does prototypal inheritance work in JavaScript and how is it different from classical inheritance?

Explanation: JavaScript uses prototype-based inheritance, which is different from the class-based inheritance found in languages like Java or C++. In prototypal inheritance:

- Every JavaScript object has a hidden property called `[[Prototype]]` (accessible via `__proto__`)
- This property points to another object called its prototype
- When accessing a property/method, if it doesn't exist on the object itself, JavaScript looks for it in the object's prototype, then that prototype's prototype, and so on (forming the "prototype chain")
- The chain continues until reaching `Object.prototype`, which points to `null`

This is different from classical inheritance where classes inherit from other classes. In JavaScript, objects inherit directly from other objects.

Code Example:

Creating a prototype chain using object literals

```
const animal = {
  isAlive: true,
  eat() {
    console.log('Nom nom nom');
  }
};
```

Create dog object with animal as its prototype

```
const dog = Object.create(animal);
dog.bark = function() {
  console.log('Woof!');
};
```

Create a specific dog with dog as its prototype

```
const rover = Object.create(dog);
rover.name = 'Rover';
```

Using the prototype chain

```
console.log(rover.name); // "Rover" - own property
console.log(rover.isAlive); // true - inherited from animal
rover.eat(); // "Nom nom nom" - method from animal
rover.bark(); // "Woof!" - method from dog
```

Verifying the prototype chain

```
console.log(rover.__proto__ === dog); // true
console.log(dog.__proto__ === animal); // true
console.log(animal.__proto__ === Object.prototype); // true
console.log(Object.prototype.__proto__ === null); // true
```

Modern approach: constructor functions with prototypes

```
function Animal(name) {
  this.name = name;
}

Animal.prototype.eat = function() {
  console.log(`${this.name} is eating`);
};

function Dog(name, breed) {
  // Call parent constructor
  Animal.call(this, name);
  this.breed = breed;
}
```

Set up inheritance

```
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog; // Fix constructor property
```

Add method to Dog prototype

```
Dog.prototype.bark = function() {
  console.log(`${this.name} says woof!`);
};
```

```
const rex = new Dog('Rex', 'German Shepherd');
rex.eat(); // "Rex is eating" - inherited from Animal
rex.bark(); // "Rex says woof!" - from Dog
```

ES6 Classes (syntactic sugar over prototypal inheritance)

```
class AnimalClass {
  constructor(name) {
    this.name = name;
  }

  eat() {
    console.log(`${this.name} is eating`);
  }
}

class DogClass extends AnimalClass {
  constructor(name, breed) {
    super(name);
    this.breed = breed;
  }

  bark() {
    console.log(`${this.name} says woof!`);
  }
}

const max = new DogClass('Max', 'Beagle');
max.eat(); // "Max is eating"
max.bark(); // "Max says woof!"
```

Understanding prototypal inheritance is essential for working with JavaScript objects, creating efficient inheritance patterns, and understanding how modern features like classes work under the hood.

Follow for more frontend interview preparation content!

Sumedh Patil