Custom Prototype

INTERVIEW QUESTIONS-126

Follow on

@Duvvuru Kishore

Creating your own prototype in JavaScript is a fundamental concept that allows you to define shared properties and methods for objects of a particular type

Four steps to define your own prototype

- Define a Constructor Function
- Add Methods to the Prototype
- Create Instances of the Constructor Function
- Access Prototype Methods

sample and real time example at last 

Step-by-step guide

- **Define a Constructor Function:**

A constructor function is used to create objects of a specific type. It initializes the object's properties.

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}
```

- **Add Methods to the Prototype:**

To add methods that should be shared among all instances of the constructor function, you attach them to the prototype of the constructor function.

```
Person.prototype.getFullName = function() {
    return `${this.firstName} ${this.lastName}`;
};
```

- **Create Instances of the Constructor Function:**

Use the new keyword to create instances of the object, which will inherit properties and methods from the prototype.

```
const person1 = new Person('John', 'Doe');
const person2 = new Person('Jane', 'Smith');
```

- **Access Prototype Methods:**

The instances can now access the methods defined on the prototype.

```
console.log(person1.getFullName()); // Output: John Doe
console.log(person2.getFullName()); // Output: Jane Smith
```

Sample example

```
// Step 1: Define the constructor function
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

// Step 2: Add methods to the prototype
Person.prototype.getFullName = function() {
    return `${this.firstName} ${this.lastName}`;
};

// Step 3: Create instances of the constructor function
const person1 = new Person('John', 'Doe');
const person2 = new Person('Jane', 'Smith');

// Step 4: Access prototype methods
console.log(person1.getFullName()); // Output: John Doe
console.log(person2.getFullName()); // Output: Jane Smith
```

Real-Time Example: E-commerce Product Prototype

```javascript
// Step 1: Define the constructor function
function Product(name, price, category) {
    this.name = name;
    this.price = price;
    this.category = category;
}

// Step 2: Add methods to the prototype
Product.prototype.getDiscountedPrice = function(discountPercentage) {
    return this.price - (this.price * (discountPercentage / 100));
};

Product.prototype.displayProductInfo = function() {
    return `Product: ${this.name}, Category: ${this.category}, Price: $${this.price.toFixed(2)}`;
};

// Step 3: Create instances of the constructor function
const laptop = new Product('Laptop', 1200, 'Electronics');
const book = new Product('Book', 20, 'Books');

// Step 4: Access prototype methods
console.log(laptop.displayProductInfo());
// Output: Product: Laptop, Category: Electronics, Price: $1200.00

console.log(`Discounted Price: $${laptop.getDiscountedPrice(10).toFixed(2)}`);
// Output: Discounted Price: $1080.00
```

- Creating custom prototypes in JavaScript is a powerful technique that allows you to define shared properties and methods for objects of a particular type.
- By leveraging prototypes, you can achieve more efficient memory usage, promote code reusability, and establish a clear structure for your objects.

Follow on

@Duvvuru Kishore