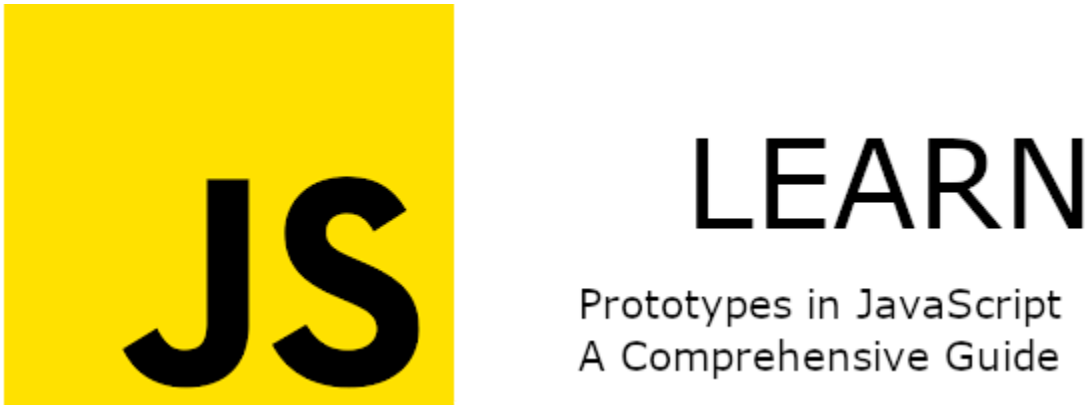
Prototypes in JavaScript: A Comprehensive Guide



Overview:

In JavaScript, prototypes play a crucial role in the inheritance model. Understanding prototypes is essential for mastering JavaScript and building scalable, efficient code.

Key Concepts:

Prototype Chain:

Every JavaScript object has a prototype, and this forms a chain. Objects inherit properties and methods from their prototype.

Prototype Property:

The prototype property is inherent to all JavaScript functions.
It allows the creation of shared properties and methods among instances.

Constructor Function:

Objects are created using constructor functions.
The prototype property of the constructor becomes the prototype of its instances.

Coding Examples:

1. Creating a Prototype:

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

// Adding a method to the prototype
Person.prototype.greet = function() {
  console.log(`Hello, my name is ${this.name}!`);
```

```
};

// Creating instances
const person1 = new Person("Alice", 25);
const person2 = new Person("Bob", 30);

// Using the prototype method
person1.greet(); // Outputs: Hello, my name is Alice!
person2.greet(); // Outputs: Hello, my name is Bob!
```

2. Inheriting from Prototypes:

```
function Student(name, age, grade) {
  // Inheriting properties from the Person prototype
  Person.call(this, name, age);
  this.grade = grade;
}

// Inheriting methods from the Person prototype
Student.prototype = Object.create(Person.prototype);

// Adding a method specific to Student
Student.prototype.study = function() {
  console.log(`${this.name} is studying hard!`);
};
```

```
// Creating a Student instance
const student1 = new Student("Charlie", 22, "A");

// Using inherited and specific methods
student1.greet(); // Outputs: Hello, my name is Charlie!
student1.study(); // Outputs: Charlie is studying hard!
```

3. Built-in Object Prototypes:

```
// Extending the Array prototype
Array.prototype.doubleValues = function() {
  return this.map(item => item * 2);
};

const numbers = [1, 2, 3, 4];

// Using the extended method
const doubledNumbers = numbers.doubleValues();
console.log(doubledNumbers); // Outputs: [2, 4, 6, 8]
```

Summary:

Prototypes are at the core of JavaScript's object-oriented nature. They enable the creation of efficient, reusable code through inheritance.

Coding Exercises

10 coding exercises focused on prototypes in JavaScript, along with detailed steps, descriptions, and solutions.

Exercise 1: Creating a Basic Prototype

Description: Create a prototype named Car with properties make and model. Create an instance and log the properties.

Solution:

```
function Car(make, model) {
  this.make = make;
  this.model = model;
}

var myCar = new Car("Toyota", "Camry");
```

```
console.log(myCar.make); // Outputs: Toyota
console.log(myCar.model); // Outputs: Camry
```

Exercise 2: Adding a Method to the Prototype

Description: Extend the Car prototype with a method startEngine that logs "Engine started!".

Solution:

```
Car.prototype.startEngine = function() {
  console.log("Engine started!");
};

myCar.startEngine(); // Outputs: Engine started!
```

Exercise 3: Inheriting Properties

Description: Create a prototype SportsCar that inherits from Car and adds a property topSpeed.

Solution:

```
function SportsCar(make, model, topSpeed) {
  // Inheriting properties from Car
```

```
  Car.call(this, make, model);
  this.topSpeed = topSpeed;
}

var mySportsCar = new SportsCar("Ferrari", "458 Italia", 200);
console.log(mySportsCar.make); // Outputs: Ferrari
console.log(mySportsCar.topSpeed); // Outputs: 200
```

Exercise 4: Inheriting Methods

Description: Inherit the startEngine method from Car in the SportsCar prototype.

Solution:

```
SportsCar.prototype = Object.create(Car.prototype);

mySportsCar.startEngine(); // Outputs: Engine started!
```

Exercise 5: Adding a Specific Method

Description: Add a method revEngine to the SportsCar prototype that logs "Vroom Vroom!".

Solution:

```
SportsCar.prototype.revEngine = function() {
  console.log("Vroom Vroom!");
};

mySportsCar.revEngine(); // Outputs: Vroom Vroom!
```

Exercise 6: Extending Built-in Prototypes

Description: Extend the Array prototype with a method sum that calculates the sum of all elements.

Solution:

```
Array.prototype.sum = function() {
  return this.reduce((acc, num) => acc + num, 0);
};

var numbers = [1, 2, 3, 4, 5];
console.log(numbers.sum()); // Outputs: 15
```

Exercise 7: Adding a Static Method

Description: Add a static method getTotalCars to the Car prototype that logs the total number of cars created.

Solution:

```
Car.totalCars = 0;

Car.prototype.getTotalCars = function() {
  console.log(`Total cars created: ${Car.totalCars}`);
};

var newCar1 = new Car("Honda", "Civic");
Car.totalCars++;
var newCar2 = new Car("Ford", "Mustang");
Car.totalCars++;

newCar2.getTotalCars(); // Outputs: Total cars created: 2
```

Exercise 8: Using Prototypes for Efficiency

Description: Create a function calculateSquare that calculates the square of a number. Use a prototype to reuse the function across instances.

Solution:

```
function Calculator(base) {
```

```
  this.base = base;
}

Calculator.prototype.calculateSquare = function() {
  return this.base * this.base;
};

var calc1 = new Calculator(5);
var calc2 = new Calculator(8);

console.log(calc1.calculateSquare()); // Outputs: 25
console.log(calc2.calculateSquare()); // Outputs: 64
```

Exercise 9: Dynamic Prototype Property

Description: Create a function Person with properties name and age. Add a dynamic prototype property isAdult based on age.

Solution:

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

```
Person.prototype.isAdult = function() {
  return this.age >= 18;
};

var adultPerson = new Person("Alice", 25);
console.log(adultPerson.isAdult()); // Outputs: true
```

Exercise 10: Prototype Chain Exploration

Description: Explore the prototype chain of an object. Create instances of Person, Student (inheriting from Person), and Graduate (inheriting from Student).

Solution:

```
function Student(name, age, grade) {
  Person.call(this, name, age);
  this.grade = grade;
}

Student.prototype = Object.create(Person.prototype);

function Graduate(name, age, grade, specialization) {
  Student.call(this, name, age, grade);
  this.specialization = specialization;
```

```
}

Graduate.prototype = Object.create(Student.prototype);

var person = new Person("John", 25);
var student = new Student("Jane", 20, "A");
var graduate = new Graduate("Jack", 22, "B", "Computer
Science");

console.log(graduate.name); // Outputs: Jack
console.log(graduate.grade); // Outputs: B
console.log(graduate.specialization); // Outputs: Computer
Science
```

These exercises cover a range of scenarios involving prototypes in JavaScript. Practice them to enhance your understanding of prototype-based inheritance!

Quiz questions and answers

Questions:

Q1: What is the purpose of prototypes in JavaScript?

a) To create static properties

b) To create shared properties and methods among objects

c) To define constants

d) To handle errors in code

Q2: How do you create a prototype in JavaScript?

a) Using the prototype keyword

b) By declaring a new function

c) Automatically for every object

d) Only for built-in objects

Q3: What does the prototype property of a function contain?

a) The function's source code

b) Shared properties and methods for instances created by the function

c) The function's parameters

d) The function's return value

Q4: How do you add a method to a prototype?

a) Using the addMethod function

b) By modifying the prototype directly

c) Only within the constructor function

d) By creating a new instance method

Q5: Inheriting properties in JavaScript is achieved through:

a) Static properties

b) Prototype chain

c) Object literals

d) Constructor properties

Q6: What is the purpose of Object.create() in prototype-based inheritance?

a) To create a new object with the same properties

b) To create a new object with the same prototype

c) To add a method to an existing object

d) To check if an object has a prototype

Q7: How do you call a method from a parent prototype when using inheritance?

a) callMethod()

b) parentMethod()

c) Using the super keyword

d) parent.method()

Q8: What does the Array.prototype property contain?

a) Array methods and properties

b) Static methods for arrays

c) Shared properties for array instances

d) The source code of the array constructor

Q9: What is a static method in a prototype?

a) A method specific to an instance

b) A method shared among instances

c) A method added to a constructor

d) A method for handling errors

Q10: How can you create a static property for a prototype?

a) By modifying the prototype directly

b) Using the static keyword

c) Only within the constructor function

d) By creating a new instance property

Answers:

Answer: b) To create shared properties and methods among objects

Answer: b) By declaring a new function

Answer: b) Shared properties and methods for instances created by the function

Answer: b) By modifying the prototype directly

Answer: b) Prototype chain

Answer: b) To create a new object with the same prototype

Answer: c) Using the super keyword

Answer: a) Array methods and properties

Answer: c) A method added to a constructor

Answer: a) By modifying the prototype directly