Gmail Bikash Adhikari <virtual.bikash@gmail.com>

Task 3.1: Create Inheritance Using Prototypes

1 message

Bikash <bikash.caim@gmail.com> Sat, Mar 8, 2025 at 7:00 PM
To: virtual.bikash@gmail.com

JavaScript

Task 1: Create Inheritance Using Prototypes

Create a constructor **Animal** with a method `makeSound()`. Then create a constructor Dog that inherits from **Animal** and adds a method `bark()`

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

Animal.prototype.makeSound = function() {
  console.log("Some generic animal sound");
};

function Dog(name, breed) {
  Animal.call(this, name); // Call the Animal constructor to set the 'name' property
  this.breed = breed;
}

// Inherit from Animal by setting Dog's prototype to an instance of Animal
Dog.prototype = Object.create(Animal.prototype);

// Reset Dog's constructor property to Dog
Dog.prototype.constructor = Dog;

Dog.prototype.bark = function() {
  console.log("Woof! Woof!");
};

// Example usage:
const animal = new Animal("Generic Animal");
animal.makeSound(); // Output: Some generic animal sound

const dog = new Dog("Buddy", "Golden Retriever");
dog.makeSound(); // Output: Some generic animal sound (inherited)
dog.bark(); // Output: Woof! Woof!

console.log(dog instanceof Animal); //true
console.log(dog instanceof Dog); //true
console.log(animal instanceof Dog); //false
```

Explanation:

1. `Animal` **Constructor:**

- The `Animal` constructor takes a `name` parameter and sets it as a property of the created object.
- `Animal.prototype.makeSound` defines a method that will be available to all `Animal` instances.

2. `Dog` **Constructor:**
   - The `Dog` constructor takes `name` and `breed` parameters.
   - `Animal.call(this, name);` is crucial. It calls the `Animal` constructor in the context of the `Dog` instance, ensuring that the `name` property is set correctly.
   - `this.breed = breed;` sets the `breed` property specific to `Dog` objects.
3. **Inheritance:**
   - `Dog.prototype = Object.create(Animal.prototype);` establishes the inheritance relationship. It sets the prototype of `Dog` to a new object whose prototype is `Animal.prototype`. This means that `Dog` instances will inherit properties and methods from `Animal.prototype`.
   - `Dog.prototype.constructor = Dog;` resets the constructor property of `Dog.prototype` to point back to the `Dog` constructor. Without this, the constructor would incorrectly point to `Animal`.
4. `Dog.prototype.bark`**:**
   - `Dog.prototype.bark` adds a method specific to `Dog` instances.
5. **Example Usage:**
   - We create an `animal` instance of `Animal` and a `dog` instance of `Dog`.
   - We demonstrate that `dog` can access both `makeSound` (inherited from `Animal`) and `bark` (defined in `Dog`).
   - The `instanceof` operator is used to show the inheritance chain.