# Super

(https://github.com/learn-co-curriculum/phase-1-super) (https://github.com/learn-co-curriculum/phase-1-super/issues/new)

## Learning Goals

- Recognize how to use the `super` method
- Recognize how to use the `super` object

## Introduction

In addition to simply extending classes, JavaScript provides an additional keyword, `super`, for directly working with a parent class constructor and inherited methods.

## Recognize How to Use the `super` Method

In the code below, we have two JavaScript classes: `Pet` and `Dog`. The `Dog` class is a *child* class of `Pet` and it uses the `extends` keyword to inherit methods from the parent class:

```
class Pet {
  constructor(name) {
    this.name = name;
    this.owner = null;
  }

  speak() {
    return `${this.name} speaks.`;
  }
}

// Inherits from Pet
```

```
class Dog extends Pet {
  constructor(name, breed) {
    super(name); /* new */
    this.breed = breed;
  }
}

const creature = new Pet("The Thing");
const dog = new Dog("Spot", "Foxhound");

dog;
// => Dog { name: 'Spot', owner: null, breed: 'Foxhound' }
```

Above, there is something new. The `Pet` class takes in a `name` parameter, assigns it to the `name` property, and *also* creates an `owner` property, setting it to `null`. The `Dog` class takes in `name` and `breed` properties, calls `super`, passing in the name, then sets `this.breed` to the provided breed.

What is happening? In our `Dog` constructor, we are able to use `super` to call the `Pet` constructor. Doing this will set up the `name` and `owner` properties. Then, once complete, the `Dog` constructor continues to execute, setting `breed`.

In a child class constructor, `super` is used as a **method** and calls the parent class constructor before continuing with the child. This lets us extend a parent's constructor inside a child. If we need to define custom behavior in a child constructor, we can do so without having to override or ignore the parent.

# Recognize How to Use the `super` Object

Outside of the constructor, the `super` keyword is also used, but this time, as an `object`. When used, it refers to parent class' properties or methods.

We could, for instance, use `super.speak()` from the `info` method in our `Dog` class to call the `speak` method in the parent `Pet` class:

```
// Inherits from Pet
class Dog extends Pet {
```

```
    constructor(name, breed) {
      super(name); /* new */
      this.breed = breed;
    }

    get info() {
      return `${this.name} is a ${this.breed}. ${super.speak()}`;
    }
  }

  const charlie = new Dog("Charlie B. Barkin", "Mutt");

  charlie.info;
  // => 'Charlie B. Barkin is a Mutt. Charlie B. Barkin speaks.'

  const lady = new Dog("Lady", "Cocker Spaniel");

  lady.info;
  // => 'Lady is a Cocker Spaniel. Lady speaks.'
```

In the above code, we've added an `info` getter that uses `super.speak()` to call the `speak` method in the parent class.

**However**, since instance methods and properties are *already* inherited, this *will be the same as using* `this.speak()` .

Using `super` as an object is useful in situations where a parent class contains a static method that we want to expand on in a child class:

```
  class Pet {
    constructor(name) {
      this.name = name;
      this.owner = null;
    }

    static definition() {
```

```
      return `A pet is an animal kept primarily for a person's company.`;
    }
  }

  // Inherits from Pet
  class Dog extends Pet {
    constructor(name, breed) {
      super(name);
      this.breed = breed;
    }

    static definition() {
      return (
        super.definition() + " Dogs are one of the most common types of pets."
      );
    }
  }

  const creature = new Pet("The Thing");
  const dog = new Dog("Spot", "foxhound");

  Pet.definition();
  // => "A pet is an animal kept primarily for a person's company."
  Dog.definition();
  // => "A pet is an animal kept primarily for a person's company. Dogs are one of the most common types of pets."
```

In the `Pet` class above, we've included a static method, `definition()`, for what a pet is. In `Dog`, we are able to use `super.definition()` to access that static method, then *add* to it, in this case, extending the definition to specifically reference dogs.

# Conclusion

In this lesson, we dove deeper into class extensions and inheritance in JavaScript. In combination with `extends`, `super` allows a child class to access a parent's constructor from within a child's constructor. It also allows a child class to access methods and properties from a parent class, but as most of these are already inherited, this is only useful when modifying static methods from the parent class.

# Resources

- **Inheritance in JavaScript (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance)**
- **"Super" and "Extends" In JavaScript ES6 - Understanding The Tough Parts (https://medium.com/beginners-guide-to-mobile-web-development/super-and-extends-in-javascript-es6-understanding-the-tough-parts-6120372d3420)**
- **Class inheritance, super (https://javascript.info/class-inheritance)**