Class Extensions and Inheritance

(https://github.com/learn-co-curriculum/phase-1-class-extension-and-inheritance) (https://github.com/learn-co-curriculum/phase-1-class-extension-and-inheritance/issues/new)

Learning Goals

- Use the `extends` keyword

Introduction

In JavaScript, as in other Object Oriented languages, we've learned that we can create classes and build methods that can perform actions on instance data, or specific to the class. What if you have classes that exhibit many of the same behaviors, such as `Cat`, `Dog`, and `Bird`, which all have a method for `speak`?

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

  speak() {
    return `${this.name} says woof!`
  }
}

class Cat {
  constructor(name) {
    this.name = name;
  }

  speak() {
```

```
    return `${this.name} says meow!`
  }
}

class Bird {
  constructor(name) {
    this.name = name;
  }

  speak() {
     return `${this.name} says squawk!`
  }
}
```

In this code snippet, `Dog`, `Cat`, and `Bird` all accept `name` and have a method called `speak()`, thus repeating code. In JavaScript, we can create "child" object classes that inherit methods and properties from their "parent" classes, allowing us to reuse some class methods while building in additional functionality.

In this lesson, we'll discuss 1 way of *extending* functionality to other classes.

Use the `extends` Keyword

To get started with inheriting class functionality, we utilize the `extends` keyword. `extends` is used in class declarations to create a class which is a *child* of another class.

```
class Pet {
  constructor(name, sound) {
    this.name = name;
    this.sound = sound;
  }

  speak() {
    return `${this.name} says ${this.sound}!`
```

```
    }
}

class Dog extends Pet {
  // inherits constructor from Pet
}

class Cat extends Pet {
  // inherits constructor from Pet
}

class Bird extends Pet  {
  // inherits constructor from Pet
  fly() {
    return `${this.name} flies away!`
  }
}

let dog = new Dog("Shadow", "woof");
let cat = new Cat("Missy", "meow");
let bird = new Bird("Tiki", "squawk");

dog.speak(); // Shadow says woof!
cat.speak(); // Missy says meow!
bird.speak(); // Tiki says squawk!
bird.fly(); // Tiki flies away!
```

In addition to *inheriting* the functionality of the `Pet` class, each "child" class extending the functionality of the parent. For example, `Bird` has an additional method called `fly` that is unique to it, and not present on `Pet`. `Bird` can still call `speak()`.

Conclusion

In this lesson, we learned about more functionality in JavaScript that allows us to leverage Object Orientation concepts: class extensions and inheritance. With `extends` we can create new classes that are capable utilizing of all the same methods as its parent. Leveraging inheritance and `extends` is vital in Object Oriented programming. It keep code bases maintainable by sharing and reusing code in a beneficial manner.

Resources

- **Inheritance in JavaScript (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance)**
- **Extends (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends)**
- **“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)**