Overriding Methods & Properties

This lesson teaches us how to override methods and properties in both the ES5 and ES6 versions of JavaScript.

WE'LL COVER THE FOLLOWING

Overriding in the ES5 Version

The properties and methods defined on the prototype of a *constructor* function can be **overridden** when *inherited* by another *constructor* function.

Example

Let’s take a look at an example implementing this:

```
//constructor function Shape
function Shape(shapeName,shapeSides){
  this.name = shapeName
  this.sides = shapeSides
}
Shape.prototype.displayInfo = function(){
  console.log(`Shape is ${this.name}`)
}

Shape.prototype.equalSides = 'no'

//constructor function Rectangle
function Rectangle(shapeName,shapeSides,shapeLength,shapeWidth){
  Shape.call(this,shapeName,shapeSides)
  this.length = shapeLength
  this.width = shapeWidth
}

Rectangle.prototype = Object.create(Shape.prototype)
Rectangle.prototype.constructor = Rectangle
```

```
//overriding the value of "equalsides" property
Rectangle.prototype.equalSides = 'yes'
console.log(Rectangle.prototype.equalSides)

//overriding the displayInfo method
Rectangle.prototype.displayInfo = function(){
  return this.sides
}
var rec = new Rectangle('Rectangle',4,3,5)
//shows sides instead of name
console.log(rec.displayInfo())
```

In the example above:

- In **line 6**, the `displayInfo` function is defined on the prototype of `Shape` constructor function.
- In **line 10**, the `equalSides` property is defined on the prototype of `Shape` constructor function.
- When `Shape` object becomes the prototype for `Rectangle`, `Rectangle` can then inherit `Shape`'s prototype properties.
- In **line 24**, `Rectangle` overrides the inherited property, `equalSides`, by accessing it and setting its value equal to `yes`.
- Similarly, in **line 29**, `Rectangle` overrides the inherited `displayInfo` function by modifying it and returning `sides` instead of displaying the `name`.

Overriding in the ES6 Version

As discussed in the previous lesson, using `super()` invokes the *parent* class *constructor*. Similarly, `super.method()` is used to invoke a parent class method.

Example

Let's see how `super.method()` can be used to override a method in the child class in the example below:

```
class Shape{
  constructor(shapeName,shapeSides){
    this.name = shapeName
    this.sides = shapeSides
  }
  getArea(){

    return 0
  }
}

class Rectangle extends Shape{
  constructor(shapeName,shapeSides,shapeLength,shapeWidth){
    super(shapeName,shapeSides)
    this.length = shapeLength
    this.width = shapeWidth
  }
  //method calculating the area of rectangle
  calculateArea(){
    console.log("Area:",this.length*this.width)
  }
  //overriding the getArea() method from parent class
  getArea(){
    //parent class's getArea called first
    super.getArea()
    //calculateArea called
    this.calculateArea()
  }
}

var rec = new Rectangle('Rectangle',4,3,5)
console.log("Name:",rec.name)
console.log("Sides:",rec.sides)
console.log("Length",rec.length)
console.log("Width",rec.width)
rec.getArea()
```

As seen in the code above, in order to modify the functionality of the original `getArea` method in the `Rectangle,` i.e., the child class:

- A method `calculateArea` is defined that displays the area of a rectangle by computing the product of its `length` and `width` properties.

- A method with the same name, `getArea`, is defined which calls `Shape`'s `getArea` method using the `super` *keyword* followed by a call to its `calculateArea` function.

Hence, `Rectangle` gets a `getArea` function with a modified functionality.

Let's discuss the interesting concept of *mixins* in the next lesson.