Classes

WE'LL COVER THE FOLLOWING

Classes are now here in JavaScript, up until now we have had many ways to ways to emulate this concept. For awhile now we have been emulating classes with the use of functions. The introduction of classes to the languages was met with a lot of opinions, some positive, most negative. Classes have been an important part of many languages, as they allow you a means to create blueprints for common elements in your program.

Class emulation in JavaScript

In JavaScript we can use functions and objects to emulate classes. Functions can be used to create constructors that, when used with the `new` keyword, can instantiate new objects.

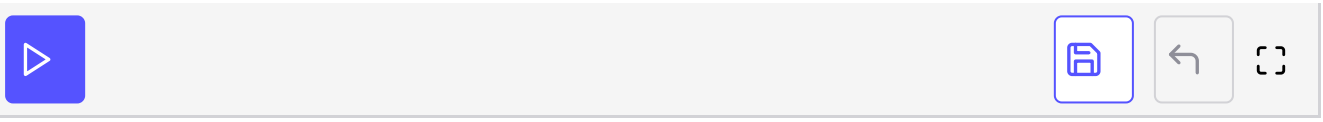
```
function Plane() {
        this.wings = 2;
        this.speed = 100;
        this.altitude = 0;
}
```

Convention has it that constructors are named with a capital first letter. To create a new `Plane` we use `new`.

```
const myPlane = new Plane();
```

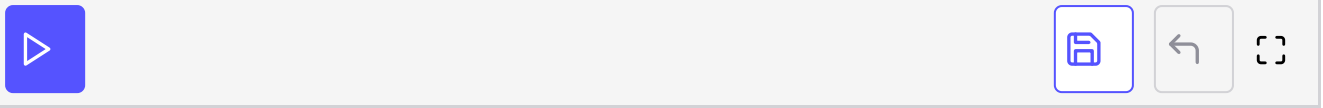
If we want to add methods to the `Plane` we can use the `prototype` property to add them on.

```
Plane.prototype.fly = function() {
        this.altitude = 30000;
};
```



This will allow us to use the `fly` method on our new `myPlane`.

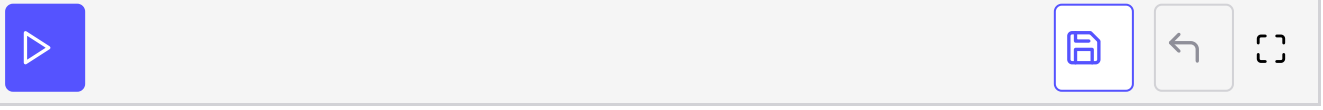
```
function Plane() {
        this.wings = 2;
        this.speed = 100;
        this.altitude = 0;
}

Plane.prototype.fly = function() {
        this.altitude = 30000;
  console.log('Flying. Altitude:', this.altitude, 'Speed:', this.speed);
};

const myPlane = new Plane();
myPlane.fly();
```



Inheritance with functions

It we want to create inheritance with functions, we can also use the `prototype` property to create an inheritance chain.

```
function JetFighter () {
        this.speed = 1000;
}

JetFighter.prototype = new Plane();

const jet = new JetFighter();
jet.fly();
```



We won't go much further into object-oriented JavaScript. There is an excellent book by Nicholas Zakas called *Principles of Object-Oriented JavaScript* that I recommend checking out. I should point out that the `class` keyword is actually just syntactic sugar for the method shown above.

Class

Let's now look at how the new `class` syntax works in ES6. If you come from a language like Java this might look very familiar.

To define a class we use the `class` keyword and give it a name; this is a class declaration.

```
class Human {}
```

We can also create a class on a variable; this is called a class expression.

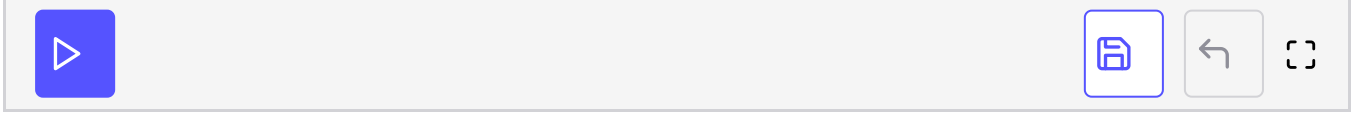
```
const Human = class {}
```

By convention we use a capital letter for the class name. There is a a special method that needs to be included if we would like to apply initial properties to our class. This method is the `constructor()` method. You will notice we use method shorthand we saw earlier in the Objects chapter.

```
class Human {
        constructor() {
                this.height = 185;
        }
}
```

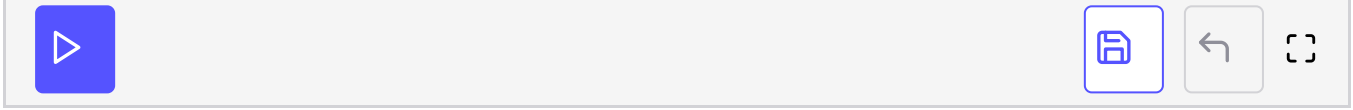
The `constructor()` method is called whenever we create a new instance of our class using the `new` keyword. Take a look at the example below.

```
const ryan = new Human();
console.log(ryan.height); //185
```



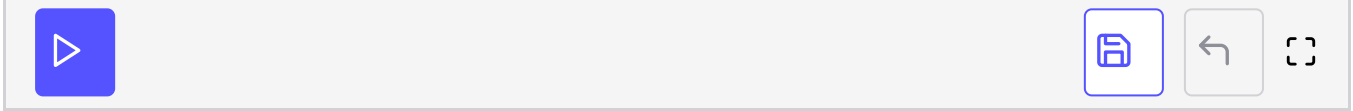
We can pass arguments into the `constructor` method.

```
class Human {
        constructor(height) {
                this.height = height;
        }
}

const ryan = new Human(190);
console.log(ryan.height);
```



Let's expand on our `Human` class and add some more properties. Maybe we are creating a video game, so we add `speed`, `strength`, and `location` properties. We also add a `walk` method. Something to note about the syntax for adding new methods to a class is that we do not add a `,` for each new new method.

```
class Human {
        constructor() {
                this.height = 185;
                this.weight = 220;
                this.strength = 10;
                this.speed = 10;
                this.location = {
                        x: 0,
                        y: 0
                }
        }
        walk(x,y) {
                this.location.x += x;
                this.location.y += y;
    console.log('Walked to new location:', this.location.x, this.location.y);
        }
}

const human = new Human();
human.walk(10, 10);
```



This will probably trip you up the first few times you write a class, since we are so used to the object literal syntax. With the introduction of classes, we also get the ability to extend other classes. We can do this with the `extends` keyword. Consider the game we are making, perhaps we want to use the `Human` class we defined earlier as a base for a `Warrior` class.

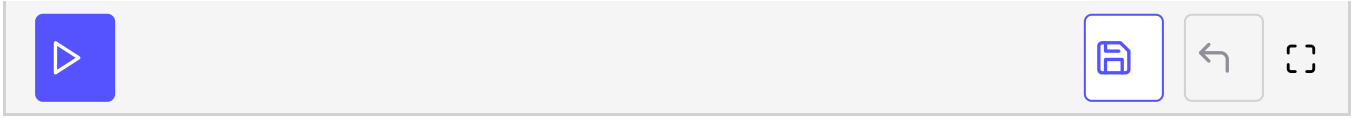
```
class Warrior extends Human {}
```

This will create a new class base on `Human`. It will have the same `walk` method and properties. However we might want to change the `strength` of the `Warrior`, and to do this we need to adjust the constructor.

super()

If we tried to just call our constructor with some new properties we would see an error.

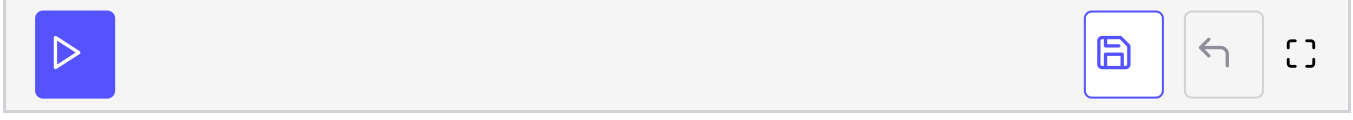
```
class Warrior extends Human {
        constructor() {
                this.strength = 15;
        }
}

const conan = new Warrior(); //"ReferenceError: this is not defined
```



In order for us to add to our `Warrior` class we need to call `super()` before any definitions with the `this` keyword.

> more explanation for introduction of super()

```
class Warrior extends Human {
        constructor() {
                super();
                this.strength = 15;
        }
}

const conan = new Warrior();
conan.walk(100, 100);
console.log('Strength:', conan.strength);
```

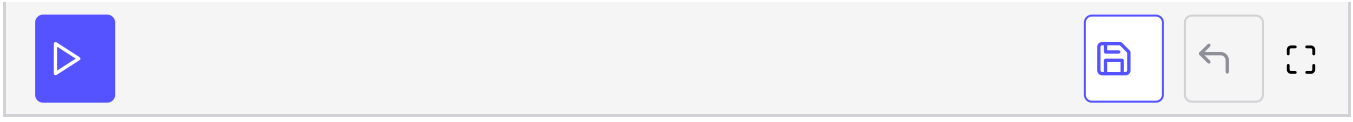


Calling `super()` will call the base class' `constructor()` method and allow us to override properties. This can also be done from other methods in the code.

Static Methods

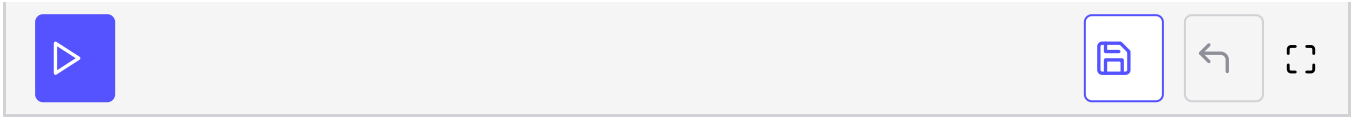
In order to access methods from a class we have to instantiate it in order to create a new version of it. However we can also create static methods for our classes that allow us to call them without having to instantiate a new object. To do this we can use the `static` keyword in front of the method name.

```
class Human {
        static sayHello() {
                return 'Hi there!';
        }
}

console.log(Human.sayHello());
```



You can mix this `static` method with the `get` syntax that binds a function or property. If you are familiar with this syntax from Objects, it works the same way.

```
class Human {
        static get sayHello() {
                return 'Hi there!';
        }
}

console.log(Human.sayHello);
```



But I read I should never use a class...

There are a lot of options on the web, not sure if you were aware of that! A lot of people state that adding classes to ES6 was a mistake. I wanted to add it to this book and my video series because I wanted to show you things without opinions around them. Now that you have an understanding of how they work, I would like you to form your own opinions about it. It is always best to have your own opinion rather than that of someone you read about.