Learning Objectives

- **Create a class in JavaScript**
- **Differentiate constructors between Python and JavaScript**
- **Use `this` instead of `self` in JavaScript classes**
- **Inherit from a superclass**
- **Reference the superclass with the `super` keyword**
- **Explain the benefit of using an arrow function inside a class**

Clone Blango Repo

Clone Blango Repo

Before we continue, you need to clone the `blango` repo so you have all of your code. You will need the SSH information for your repo.

In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/blango.git
```

- You should see a `blango` directory appear in the file tree.

You are now ready for the next assignment.

JavaScript Classes

Intro

JavaScript classes have many similarities to Python classes. They can have methods, attributes and a constructor. Unlike Python classes though, they can only inherit from one super class. The constructor method is called `constructor`, and is the equivalent of Python's `__init__`. When instantiating a class, you must use the `new` keyword, you can't just "call" the class as you do in Python.

To access the current object from within the class, the keyword `this` is used. It's the equivalent of `self` in Python, but is implicit, i.e. `this` is not passed to every method like `self` is.

Let's look at a really basic class, called `Greeter`. It will accept one (optional) parameter in its constructor, `name`. It will then set this to the `name` attribute on itself. We'll also write some other methods that generate greetings and print them out. The main "entrypoint" to the class will be the `greet()` method.

```
class Greeter {
  constructor (name) {
    this.name = name
  }

  getGreeting () {
    if (this.name === undefined) {
      return 'Hello, no name'
    }

    return 'Hello, ' + this.name
  }

  showGreeting (greetingMessage) {
    console.log(greetingMessage)
  }

  greet () {
    this.showGreeting(this.getGreeting())
  }
}
```

The other methods are: `getGreeting()`, which will build the greeting string based on the name that was passed to the constructor; and `showGreeting()`, which is just a wrapper around `console.log()`. It could be overridden in a child class to output the message in another way (maybe in an `alert()`). Note that the methods don't need to be preceded with `function` or use the `=>` operator, the interpreter can deduce that they're methods because they have parentheses at the end of their identifier.

Here's how the class is instantiated and used:

```
const g = new Greeter('Patchy')
g.greet()
```

This will output `Hello, Patchy` to the console, by calling `greet()` which calls `showGreeting()` with the result of `getGreeting()`.

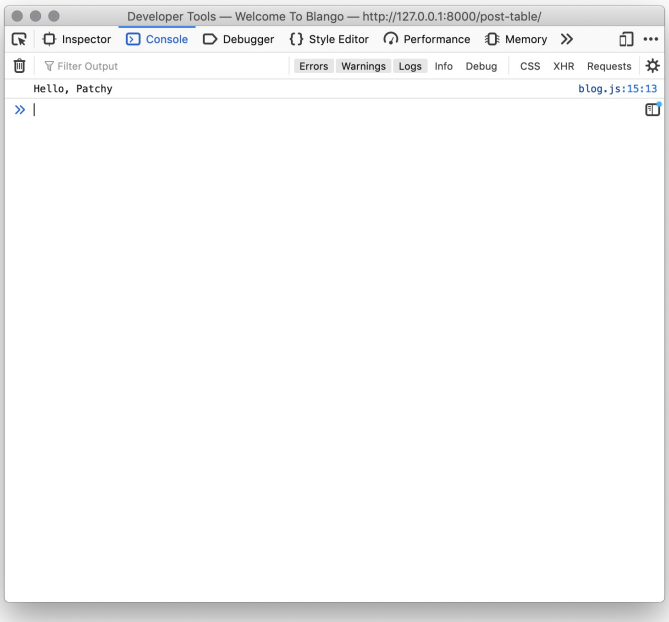
Try It Out:

You can set up the `Greeter` class in your Blango project. Once again start by emptying the contents of `blog.js` and put in this code instead:

```
class Greeter {
  constructor (name) {
    this.name = name
  }

  getGreeting () {
    if (this.name === undefined) {
      return 'Hello, no name'
    }

    return 'Hello, ' + this.name
  }

  showGreeting (greetingMessage) {
    console.log(greetingMessage)
  }

  greet () {
    this.showGreeting(this.getGreeting())
  }
}

const g = new Greeter('Patchy')  // Put your name here if you
        like
g.greet()
```

Load the `/post-table/` page and look at the browser console. You probably won't be surprised to find your greeting there.

View Blog



greeter output

Now let's look at inheritance, attributes, and arrow functions.

Inheritance, Attributes, and Arrow Functions

Inheritance, Attributes, and Arrow Functions

It may seem a bit strange to lump these together but these are the last three things that we're going to look at in relation to classes. We're assuming that you already know about these things in relation to Python classes, and you just need to see how it's done in JavaScript.

First, inheritance is denoted with the `extends` keyword. We're going to create another class called `DelayedGreeter`, which will greet someone, but after a specified delay. Here's how we start off defining the class:

```
class DelayedGreeter extends Greeter {

}
```

Right now, if we used `DelayedGreeter`, it would work *exactly* like `Greeter`, as none of the methods have been overridden.

Let's override the constructor, so as well as providing a `name`, we can also provide an optional `delay`. We'll make use of `super()`, a special function that calls the method in which it's used on the superclass. It's similar to the `super()` method in Python, but not as explicit in its behavior:

```
class DelayedGreeter extends Greeter {
  constructor (name, delay) {
    super(name)
    if (delay !== undefined) {
      this.delay = delay
    }
  }
}
```

Here, `super(name)` calls the `constructor()` method on `Greeter`, with the `name` argument. We then set `this.delay` to the `delay` argument, if it's passed in.

If it's not passed in, then what will `delay` be? Currently, it's undefined, but we can fix that by setting an attribute. Like in Python, attributes are just set in the class body. Here, we'll set `delay` to a default of `2000`, or 2 seconds:

```
class DelayedGreeter extends Greeter {
  delay = 2000

  constructor (name, delay) {
    super(name)
    if (delay !== undefined) {
      this.delay = delay
    }
  }
```

Finally we'll override the `greet()` method to use `setTimeout()` and greet after a delay.

```
class DelayedGreeter extends Greeter {
  // other methods/attribute omitted

  greet () {
    setTimeout(
      () => {
        this.showGreeting(this.getGreeting())
      }, this.delay
    )
  }
}
```

Here we've used an arrow function, instead of a function defined with the `function` keyword, like this:

```
setTimeout(
  function() {
    this.showGreeting(this.getGreeting())
  }, this.delay
)
```

We mentioned the execution context in the previous section introducing JavaScript functions, and now we have a concrete example of the difference between these two methods of defining functions.

When using "traditional" `functions` (arrow functions are relatively new to JavaScript), the context in which the function executes changes. The upshot is that `this`, inside a `function`, no longer refers to the class instance. Instead it refers to the function itself. If we call `this.showGreeting()` it will fail as the function doesn't have a `showGreeting()` method.

Using an arrow function avoids this problem, and the function executes in the context you expect with `this` referring to the class instance. All browsers released in the last few years support arrow function so you should prefer to use them unless you have a good reason not to.

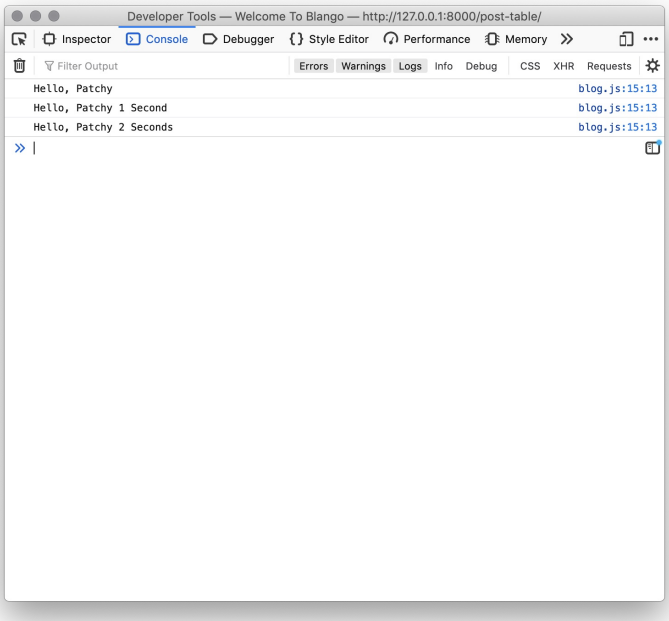
Try It Out

Now you can add the `DelayedGreeter` class, and verify for yourself that the timeout works as we expect. Open `blog.js` and add this code underneath the existing code you have (you can add your own name if you like):

```
class DelayedGreeter extends Greeter {
  delay = 2000

  constructor (name, delay) {
    super(name)
    if (delay !== undefined) {
      this.delay = delay
    }
  }

  greet () {
    setTimeout(
      () => {
        this.showGreeting(this.getGreeting())
      }, this.delay
    )
  }
}

const dg2 = new DelayedGreeter('Patchy 2 Seconds')
dg2.greet()

const dg1 = new DelayedGreeter('Patchy 1 Second', 1000)
dg1.greet()
```

Now reload `/post-table/`, and check the browser console. You should see it display the first name, then the second name after a one-second delay (even though it was called last) and then the name with the two-second delay (which is the default delay for the class).

View Blog



delayed greeter output

That's all you need to know about JavaScript classes (at least for now). In the next section we're going to look at promises.

Pushing to GitHub

Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Finish JavaScript classes"
```

- Push to GitHub:

```
git push
```