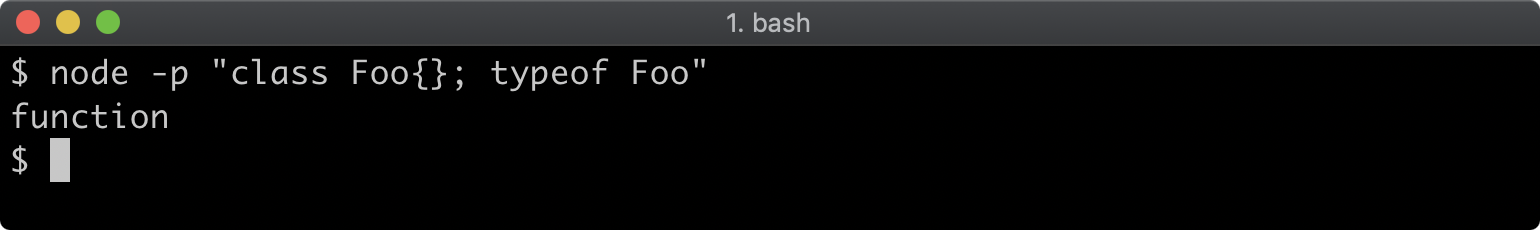
**Prototypal Inheritance (Class-Syntax Constructors)**

Modern JavaScript (EcmaScript 2015) has a **class** keyword. It's important that this isn't confused with the **class** keyword in other Classical OOP languages.

The **class** keyword is syntactic sugar that actually creates a function. Specifically it creates a function that should be called with **new**. It creates a Constructor Function, the very same Constructor Function discussed in the previous section.



This is why it's deliberately referred to here as "Class-syntax Constructors", because the EcmaScript 2015 (ES6) **class** syntax does not in fact facilitate the creation classes as they are traditionally understood in most other languages. It actually creates prototype chains to provide Prototypal Inheritance as opposed to Classical Inheritance.

The **class** syntax sugar does reduce boilerplate when creating a prototype chain:

**class Wolf {  
  constructor (name) {  
    this.name = name  
  }  
  howl () { console.log(this.name + ': awoooooooo') }  
}**

**class Dog extends Wolf {  
  constructor(name) {  
    super(name + ' the dog')  
  }  
  woof () { console.log(this.name + ': woof') }  
}**

**const rufus = new Dog('Rufus')**

**rufus.woof() // prints "Rufus the dog: woof"  
rufus.howl() // prints "Rufus the dog: awoooooooo"**

This will setup the same prototype chain as in the Functional Prototypal Inheritance and the Function Constructors Prototypal Inheritance examples:

**console.log(Object.getPrototypeOf(rufus) === Dog.prototype) //true  
console.log(Object.getPrototypeOf(Dog.prototype) === Wolf.prototype) //true**

To describe the full prototype chain:

* the prototype of **rufus** is **Dog.prototype**
* the prototype of **Dog.prototype** is **Wolf.prototype**
* the prototype of **Wolf.prototype** is **Object.prototype**.

The **extends** keyword makes prototypal inheritance a lot simpler. In the example code, **class Dog extends Wolf** will ensure that the prototype of **Dog.prototype** will be **Wolf.prototype**.

The **constructor** method in each **class** is the equivalent to the function body of a Constructor Function. So for instance **function Wolf (name) { this.name = name }** is the same as **class Wolf { constructor (name) { this.name = name } }**.

The **super** keyword in the **Dog** class **constructor** method is a generic way to call the parent class constructor while setting the **this** keyword to the current **instance**. In the Constructor Function example **Wolf.call(this, name + ' the dog')** is equivalent to **super(name + ' the dog')** here.

Any methods other than **constructor** that are defined in the **class** are added to the **prototype** object of the function that the **class** syntax creates.

Let's take a look at the **Wolf** class again:

**class Wolf {  
  constructor (name) {  
    this.name = name  
  }  
  howl () { console.log(this.name + ': awoooooooo') }  
}**

This is desugared to:

**function Wolf (name) {  
  this.name = name  
}**

**Wolf.prototype.howl = function () {  
 console.log(this.name + ': awoooooooo')  
}**

The class syntax based approach is the most recent addition to JavaScript when it comes to creating prototype chains, but is already widely used.