Prototypal Inheritance

[(https://github.com/learn-co-curriculum/phase-1-intro-to-prototypal-inheritance)](https://github.com/learn-co-curriculum/phase-1-intro-to-prototypal-inheritance) [(https://github.com/learn-co-curriculum/phase-1-intro-to-prototypal-inheritance/issues/new)](https://github.com/learn-co-curriculum/phase-1-intro-to-prototypal-inheritance/issues/new)

Learning Goals

- Define "syntactic sugar"
- Recognize that the inheritance model of JavaScript is prototypal

Introduction

When you learned object-oriented JavaScript, you were exposed to the syntax that came into existence around 2014. This syntax uses the `class` keyword, and methods called `constructor()` s to initialize new instances of the class. We'll call this the "ES2015 standard." ES2015's syntax is similar to Ruby, Python, and Java and feels to many developers learning JavaScript as a second language to be "the way OO should be done."

Define "syntactic sugar"

However, from JavaScript's perspective, ES2015 syntax is "syntactic sugar." This is a term that developers use to communicate "we made the thing easier to type and to read, but underneath, it's still doing something a lot more complicated." The object-oriented model that ES2015 simplifies is the "prototypal object oriented" model that JavaScript *actually* works in.

Recognize that the Inheritance Model of JavaScript is Prototypal

> **IMPORTANT**: This is a common interview question used to weed-out front end developers. When asked what kind of inheritance model JavaScript has, boldly and proudly say "Prototypal!" To demonstrate mastery of this vocabulary word, you'll need to do the rest of the work in this section, though. :)

The difference between Prototypal and Class-based OO goes all the way back to Plato and Aristotle. No kidding! As we think about how humans categorize and try to teach it to computers, we often find ourselves bumping up against those philosophers who sought to explain those *same* ideas *to humans* millennia ago.

In Plato's Theory of Forms he suggests that when we say "tree", our minds build a foggy rough idea of what all trees are like. Philosophy students call this "the form of a Tree" or "Tree-ness." It's the thing that makes all trees kinda the same. My conception of "Tree-ness" has something to do with bark, and leaves, and green, and pine trees, and oak trees. Yours is probably similar. There are also forms for Dogs and Cats. Dogs all share "bark" and Cats all share "meow" and Politicians all share "be shady." When we find or create a specific *instance* that "participates in" or "references" a form it stops being a cloudy murky idea and becomes a real thing. Not just Dog-ness, but my Poodle, Byron.

This should remind you of the object-oriented world of Ruby, Java, Python, and ES2015 standard JavaScript. Forms are `class` es and instances are the `new` embodiments of those `class` es with some unique, specific, real data bonded to them (via a constructor).

> **ASIDE**: Programming, for many, is an exercise in applied philosophy.

Aristotle, the biologically-minded student of Plato's, said that his teacher was talking about make-believe nonsense like dragons and unicorns when he talked about "forms."

No one has ever seen a Tree-ness or a Dog-ness.

Everything that *is*, Aristotle argued, is like a something else with more specification added. Everything new is based off of a pre-existing pattern, a first of its kind, the `proto` (first) - `typos` (kind), or *prototype*.

- A Poodle is based off the Water-Dog breed...
- ...and Water-Dogs are breeds based off of Dogs...
- ...and Dogs are domesticated Wolves...
- ...and a wolf is a carnivorous mammal...
- ...which means it's a chordate and all chordates are animals (not plants).

This probably reminds you of basic biology and the **KPCOFGS (https://www.acronymfinder.com/King-Philip-Came-Over-For-Good-Spaghetti-(mnemonic-for-taxonomy-order%3A-Kingdom%2C-Phylum%2C-Class%2C-Order%2C-Family%2C-Genus%2C-Species)-(KPCOFGS).html)** way of naming the things living on this planet.

```
Kingdom > Phylum > Class > Order > Family > Genus > Species

Animalia > Chordata > Mammalia > Carnivora > Canidae > Canis > lupus
```

Given this view of the world, it's no surprise Aristotle was a huge fan of making trees (another computer science idea that started in philosophy) of these relationships (or, "taxonomies," another computer science idea that started in philosophy). Languages that allow you to "extend what's already there"; that is, Prototypal object systems are languages like C, Lisp, Self, and JavaScript (natively).

Ultimately, both class-based (or, "form-based") and prototype-based models are ways of describing how to create instances from a common ancestor. Neither is "better" than the other. They're just different ways of seeing objects (no pun intended) in the world.

Conclusion

In this section we'll put aside the syntactic sugar of the ES2015 standard. We'll go back to writing classes and objects the way we did from 1997-2014. The ES2015 standard is fast becoming *the* standard, but it's common to see Prototypal patterns in legacy code and in job interview questions.

In this lesson you were introduced to the inheritance model of JavaScript, prototypal inheritance. You were also shown a common pattern in later versions of JavaScript: wrapping ugly or clumsy syntax in cleaner syntax ("syntactic sugar"). In the remainder of this section we're going to show you how to create prototypal classes and instances in JavaScript.