Slide 3

These bullets come from Wikipedia, but if I could sum up the concept of OOP, I would say that it is “using objects as an interface to represent real life and act as the building blocks of an application”

Slide 4

One reason why OOP is so valuable is the concept of inheritance. This is powerful because it provides code reuse, and makes your code more modular as you can associate properties/methods with certain objects, like “animal” or “dog”, and objects lower on the chain can simply inherit those properties, so we don’t have to explicitly define all of the properties on of these object.

Slide 5

I want to quickly talk about the benefits of OO design and why it is so valuable. Who hear has ever heard of pokemon? I want us to quickly think about how we would build out all of Charizard’s properties. Think of Charizard as an object, he has properties, like his name, health, etc. He also has methods, like fire spin or flamethrower.

Slide 6

Charizard inherits a lot of properties/methods from other groups. You can call these groups classes, but in JavaScript we will call them prototype.

Slide 7

Can anyone tell me what objects/classes/prototypes they would create in order to create our Charizard object?

Slide 8

I chose the word prototype because JavaScript has prototypes, JavaScript does not have Classes, and we will get to that in second. What is valuable to understand is that certain properties or characteristics can be declared at a specific level, and other objects can simply inherit that functionality. This makes code more modular and quickly allows me to add things to our application.

I don’t have to write that Charmander, Charmeleon and Charizard are super-effective on leaf pokemon, I only have to define it once, when I create a “fire” prototype. I don’t have to program in that a Charizard has levels, and cannot go past level 100. All pokemon have this characteristic

Slide 9

One way you can think about classes is imagine you are building a box. The class represents the blueprints for our box. The builder builds the box by copying the plans. Once the box is created, the blueprints no longer matter. There is no connection between the box and the blueprints, the blueprints were merely a plan that the builder used to help him build it

Slide 10

We have a class called Foo, when we create an a1 instance, behavior is copied from Foo into a1. When we make a subclass called Bar and create a b1 instance, behavior is copied from Bar into b1. Once the instances are created, THERE IS NO MORE RELATIONSHIP BETWEEN THE CLASS AND THE OBJECTS

Slide 11

JavaScript, in contrast, provides a construct for on-going linkage. A1 is inked to Foo.prototype, it is not a copy of Foo.prototype. This is the fundamental difference that makes JavaScript unique, and truly is a class-less language. (note how the arrows go bottom to top, different that the image I showed below that copied behavior top down)

Slide 12

If you remember nothing else, remember that JavaScript HAS NO CLASSES, it links objects to one another… We are going to go deeper into what that exactly means, but please remember these key principles

Slide 13

As you’ll see in a moment, the reason prototypical inheritance in JavaScript is so powerful is because of the “this” keyword. I don’t want to go too deep into this subject, but it is basically a reference to an object that is determined at runtime, not at initial declaration. The next JavaScript jumpstart section goes deeper “this”

Slide 16

On the window object, you will find these constructors… which are FUNCTIONS. Inside each constructor, you will find a .prototype object that contains all the methods you’ve seen (toString, split, slice, reverse, etc.). We are really only going to focus on the Object constructor for this discussion… feel free to look at this in your browser.

Specifically, look at window > Object > prototype and you will see a list of all the properties/methods that every Objects gets on initial creation.

Slide 17

It is generally bad practice to extend the Object.prototype, but I want to make you aware that it is possible

Slide 18

There is a lot to handle here, and if you can understand what is going on here, you can grasp almost all there is no know about prototypical inheritance, so let’s break it down line by line on the white board.

Slide 19

See coding examples for more practice