Let’s get started. So today’s lecture and Wednesday’s lecture, we’re going to talk about this thing called object oriented programming. And if you haven’t programmed before, I think this is a fairly tough concept to grasp. But hopefully with many, many examples and just by looking at the code available from lectures, you’ll hopefully get the hang of it quickly.

So let’s talk a little bit about objects. And we’ve seen objects in Python so far. Objects are basically data in Python. So every object that we’ve seen has a certain type. Ok, that we know.

Behind the scenes, though, every object has these two additional things. One is some data representation. So how Python represents the object just behind the scenes and what are different ways that you can interact with the object.

So for example, every one of these is a different object.

For example, this is the number 1234. It’s a specific object that is of type integer.

Lists and dictionaries are more complicated objects types. But every object has a type, some sort of way that it’s represented in Python and some ways that we can interact with them.

So the idea behind object oriented programming is, first of all, everything in Python is an object. We’ve said that before and in this lecture I think we’ll really get at what that means.

So we’ve seen strings, integers, dictionaries, lists. Those are all objects. When we did functions, we saw that we could pass as a parameter another function. So functions were also objects in Python. So literally everything in Python is an object. So what are the kinds of things we can do with objects?

Well, once you have a type, you can create a new object that is of some type .

And you can create as many objects as you’d like of that particular type, right? An integer 5 and integer 7. Those all work in a program. Once you’ve created these new objects, you can manipulate them. So for a list, for example, you can append an item to the end of the list, you can delete an item, remove it, concatenate two lists together. So that’s ways that you can interact with objects. And the last thing you can do is you can destroy them. So and with lists, we saw explicitly that you can delete elements from a list, or you can just forget about them by reassigning a variable to another value, and then at some point, Python will collect all of these dead objects and reclaim the memory. So let’s continue exploring what objects are. So let’s say I have these two separate objects.

One is a blue car

One is a pink car

So objects are really data abstractions . so these two cars can be created by the same blueprint.

This is a blueprint for a car and if an object is a data abstraction, there’s two things that this abstraction is going to capture. The first is some sort of representation. What is going to represent the car, what data represents a car object? And the second is what are ways that we can interact with the object? So if we think about a car blueprint, some general representation for a car could be the number of wheels it has, the number of doors it has, maybe its length, maybe its height, so this is all part of what data represents the car. The interface for the car is what are ways that you can interact with it. So for example, you could paint a car, right? , so you could change its color. You could have the car make a noise and different cars might make different noises. Or you can drive the car, right? So these are all ways that you can interact with the car. Whereas the representation are what makes up the car. What data abstractions make up the car. Let’s bring it a little closer to home by looking at a list. So we have this data type of list, right? We’ve worked with lists before.

The list with elements 1, 2, 3 and 4 is a very specific object that is of type list. Again, we think about it in terms of two things.

One is what is the data representation of the list?

So behind the scenes how does Python see lists?

And the second is, how do you interact with lists? So what are ways that you can manipulate a list object once it’s created?

So behind the scenes, you have a list, L, which is going to be made up of essentially two things. One is going to be the value at specific index. So at index 0, it has the value 1, right, because it’s the first element in the list. And the second thing that represents a list is going to be this second part, which is a pointer. And internally this pointer is going to tell Python where is the memory location in the computer where you can access the element index 1. So it’s just essentially going to be a chain, going from one index to the other. And at the next memory location you have the value at index 1 and then you have another pointer that takes you to the location in memory where the index 2 is located. And in index 2 you have the value and then the next pointer, and so on and so on. So this is how Python internally represents a list. How you manipulate lists, we’ve done this a lot, right? You can index into a list, you can add two lists together, you can get the length, you can append to the end of a list, you can sort a list, reverse a list, and so many other things , right? So these are all ways that you can interact with the list object as soon as you’ve created it. So notice both of these, the internal representation and how you manipulate lists, you don’t actually know internally how these are represented, right? How did whoever wrote the list class decide to implement a sort. We don’t know. You also weren’t aware of how these lists were represented internally, and you didn’t need to know that. That’s the beauty of object oriented programming and having these data abstractions. The representations are private of these objects and they are only known by what you can find out how it’s done but they only should be known by whoever implemented them. You, as someone who uses this class, doesn’t really need to know how a list is represented internally in order to be able to use it and to write cool programs with them.

So just find a motivation here before we start writing our own types of objects is the advantages of object oriented programming is really that you’re able to bundle this data, bundle some internal representation, and some ways to interact with a program into these packages.

And with these packages, you can create objects and all of these objects are going to behave the exact same way. They’re going to have the same internal representation and the same way that you can interact with them.