6.00 Introduction to Computer Science and Programming, Fall 2008
Transcript – Lecture 15

PROFESSOR: Last time, Professor Guttag introduced the idea of objects and classes and this wonderful phrase called object-oriented programming. And it's a topic I want to pick up on today, we're going to do for the next few lectures, and it's a topic I want to spend some time on because this idea of capturing data and methods, the term we're going to use for it, but data and functions that belong to that data, things that can be used to manipulate them, is a really powerful one. What we're really getting at is the idea of saying I want to have a way of grouping together information into units that make sense. So I can go back to one of those topics we had at the beginning, which is the idea of abstraction, that I can create one of those units as a simple entity, bury away the details and write really modular code. And so we're going to talk about that a lot as we go along. What we're really doing, or I shouldn't say what we're really doing, a basic piece of what we're doing, when we talk about classes or objects, is we're doing something that Professor Guttag mentioned, we're defining an abstract data type.

Now what in the world does that mean? Well basically what we're doing is we're giving ourselves the ability to create data types the same way that we have some built-ins, so we have things like int, float, string, these are built-in data types. And if you think about it, associated with each one of those data types is a set of functions it's intended to apply to. Sometimes the functions -- sometimes a function can be used on multiple data types, plus, for example, we saw could add strings, or could add ints, but each one of those data types has associated with it a set of functions that are geared to handling them. We want to do the same thing, but with our data types. We want to create data types and functions, or we're going to call them methods, that are specifically aimed at manipulating those kinds of objects. And our goal is then to basically see how we can build systems that take advantage of that modularity.

Right, so the fundamental idea then, is, I want to glue together information, I want to take pieces of data that naturally belong together, glue them together, and attach some methods to it. And that's, you know, saying a lot of words, let's do an example because it's probably easiest to see this by looking at a specific example. So here's the example I'm going to start with. Suppose I want to do little piece of code that's going to do planar geometry, points in the plane. All right, so I want to have some way of gluing those things together. Well you know what a point is, it's got an x- and a y- coordinate, it's natural to think about those two things as belonging as a single entity. So an easy way to do this would be to say, let's just represent them as a list. Just as a 2-list, or a list of 2 elements. It's easy to think of a point as just a list of an x- and a y- coordinate. OK, for example, I might say point p1 is that list, x is 1, y is

2. in fact, if I draw a little simple -- it's basically pointing to that point in the plane, right, x is 1, y is 2.

OK, fine, there's another way to represent points on the plane now, and that's in polar form, so this, if you like, is Cartesian. Another way to represent a point in a plane is I've got a radius and I've got an angle from the x-axis, right, and that's a standard thing you might do. So I might define, for example, in polar form p 2, and let me see, which example did I do here, we'll make this the point of radius 2 and at angle pi by 2, I'm going to make it easy because pi by 2 is up along this axis, and that's basically that point. Ok, just fine, it's no big deal. But here now becomes the problem. I've glued things together but just using a list. Suppose I hand you one of these lists. How do you know which kind it is? How do you know whether it's in Cartesian form or in polar form? You have nothing that identifies that there, you have no way of saying what this grouping actually means. Right, and just to get a sense of this, let's look at a simple little example, so on your hand-out, you'll see I've got a little piece of code that says assuming I've got one of these points, I want to do things with it, for example I might want to add them together.

So this first little piece of code right here says, ok you give me 2 points, I'll create another 1 of these lists and I'll simply take the x, sorry I shouldn't say x, I'm going to assume it's the x, the x-values are the two points, add them together, just right there, the y-values, add them together and return that list. And if I actually run this, which I'm going to do -- excuse me, do it again -- OK, you can see that I've added together and I've printed out the value of r, and I'll just show you that in fact that's what I've got. This looks fine, right, I'm doing the right thing. Another way of saying it is, I've actually said, what did I use there, (1,2) and (3,1), It's basically saying there is the first point, there's the second point, add them together and I get that point. OK, that sounds fine.

Now, suppose in fact these weren't x and y glued together, these were radius and angle glued together. In that case point p 1 doesn't correspond to this point, it actually corresponds to the point of radius 2 and angle 1, which is about here. I think I wrote this down carefully so I would make sure I did it right. Sorry, said that wrong, radius 1 and angle 2, 2 radians is a little bit more than pi half. And the second point is of radius 3 and angle 1, which is up about there. So what point, sorry, bad pun, what point am I trying to make here? Different understandings of what that piece means gives you different values, and that's a bit of a problem. The second problem is, suppose actually I had p 1 and p 2 were in polar form, and I ran add points on them. This little piece of code here that I did. Does that even make any sense? Course not, right? You know when you add 2 polar forms, you add the radii together, you don't add the angles together, you need to do it in Cartesian form.

So what I'm leading up to here is that we've got a problem. And the problem is, that we want to build this abstract data type, but we'd like to basically know what kind of object is it, and what functions actually belong to it, how do we use them? And so I'm going to go back to this idea of a class, and let's build the first of these, and that is shown right here on this piece of your handout. I'm going to define a class, and in particular, what I'm going to do, is walk through what that says. So I'm going to now build an object, it's going to represent a point. So what does that thing say up there? It's got this funky looking form, right, it says, I've got something that I'm going to call a class, got that key word class right here. And I'm going to give it a name, and right now I'm just building a simple piece of it -- but first of all, what does a class do? Think of this as, this is a template for creating instances of an object. At the moment, it's a really dumb template. I'm going to add to it in a second, but I want to build up to this. Right now it's got that second key word there called pass, which is just Python's way of saying there's an empty body in here. Right,, we're going to add to it in a second, but the idea is class is going to be a template for creating instances.

How do I use it? Well, I call class just like a function, and you can see that below. Having created this thing called Cartesian point, I'm going to create two instances of it. c p 1 and c p 2. Notice the form of it, it's just the name of the class followed by open paren, close paren, treating it like a function. What that does, is that it creates, c p 1 and c p 2 are both instances of this type, specific versions of this type. For now the way to think about this is, when I call that class definition, it goes off and allocates a specific spot in memory that corresponds to that instance. Right now it's empty, actually it's not quite empty, it has a pointer back to the class. And I can give a name to that, so c p 1 and c p 2 are both going to point to that. Once I've got that, I can now start giving some variable names, sorry not, rephrase that, I can give some attributes, I can give some characteristics to these classes. So each instance has some internal, or will have some internal attributes. Notice how I did that up there. Having created c p 1 and c p 2, I had this weird looking form here. Not so weird, you've actually seen it before. In which I said c p 1 dot x equals 1.0.

What's this doing? c p 1 points to an instance, it points to a particular version of this class. And I have now given an internal variable name x and a value associated with that. So I've just given it an x variable. All right, c p 1 dot y, I've said assign that to the value 2, 2,0. So now c p 1 has inside of it an x and y value. Did the same thing with c p 2, give it a different x and y value. Again, remind you, c p 2 is a different instance of this data type. All right, when I call the class definition it goes off and finds another spot in memory, says that the spot I'm going to give you a pointer back to that, give it the name c p 2, and then by running these 2 little assignments statements here, I've given it an x and a y value for c p 2.

So you see why I say it's a template, right? Right now it's a simple template, but it's a template for creating what a class looks like, and I now have an x- and y- value associated with each instance of this. OK, and if I wanted to look at it, we can come back over here, and we can see what does c p 1 look like, interesting. It says some funky stuff, and says it's a kind of Cartesian point. And that's going to be valuable to me when I want to get back to using these things, right? You see that little thing says dot Cartesian point in there. If I want to get out right now the versions of these things, I can ask what's the value of c p 1 x, and it returns it back out. I could say c p 2 dot x, that was a bad one to use because they use the same valuable in both places, didn't I? So let's do c p 1 dot y, c p 2 dot y.

OK, so I've just created local versions of variables with each one of these objects. I can get at them just like I would before, I can assign them in as I might have done before. OK, now that I've got that, we could think about what would I want to do with these points? Well one thing I might want to do is say, is this the same point or not? So the next little piece of code I've written here, just move down to it slightly. I've got a little piece of code called same point. And you can look at it. What does it say to do? It says, if you give me two of these data objects, I'm going to call them p 1 and p 2. I'm going to say, gee, is the x value the same in both of them, and if it is, and the y value's the same, then this is the same point, I'm going to return true. Notice the form. This is saying, that's a class, or sorry, an instance of a class, and

I'm going to get the x value associated with it. I going to come back in a second to how it actually does that, but it basically says, get me x value for p 1, get me the x value for p 2, compare them, just as you would normally. I've got another little thing here that I'm going to use a little later on that just prints out values of things.

OK, let's see what happens if I do this. Let me show you simple little example. I'm going to go over here, and let me define a couple of these things. I'm going to say p 1, try it again, p 1 is a Cartesian, it would help if I could type, Cartesian point, and I'm going to say p 1 of x is 3, p 1 of y is 4, and I'm going to make p 2 another Cartesian point. And I'll give it an x value of 3 and a y value of 4. OK, now I want to say, are these the same? Thing I've got a little procedure that could do that, but you know the simplest thing I could do is to say well, gee, wait a minute, why don't I just check to see if these are the same thing? So I can say is p 1 the same as p 2, using Scheme's built-in is comparator. Say -- sorry?

PROFESSOR 2: Part of Python?

PROFESSOR: Part of Scheme, whoa, there's a Freudian slip, thank you, John. I'm showing my age, and my history here, is p 1 and p 2 the same thing? Hey, there's a bad English sentence even worse, now I'm really thrown off. I'm using Python's is comparator to say is it the same thing? It says no. But if I say, are p 1 and p 2 the same point, it says yes. And this is a point I want to stress here. So what's going on in this case is, I want to distinguish between shallow equality and deep equality. The first thing is testing shallow equality. What it is doing, that's another bad English sentence, but what it is doing? Is is essentially saying, given 2 things, do they point to exactly the same referent? Or another way of thinking about it, is remember I said when I call that class definition it creates an instance, that's a pointer to some spot in memory that's got some local information around it. Is is saying, do these things point to exactly the same spot in memory, the same instance. Deep equality, we get to define, that's what I did by writing same point. OK, as I said, I want equality in the case of points to be, are the x- and y- coordinates the same? And I'm actually going to change it. just to show you this point. If I do the following, and I say, I'm going to assign p 1 to be p 2. What's that doing? It's taking the name p 1 and it's changing its value to point to exactly what p 2 points to. And then I say, are they the same thing? Answer's yes, because now they are pointing to exactly the same spot in memory. The same instance.

OK, the reason I'm saying this is, we have one class definition, is a cookie cutter, it's a template that's going to let us build versions of these things. Every time I use it, I'm creating a new instance, that's a different thing inside of memory. And I want to have that because I want to have lots of versions of points. OK, now, let's go back to where I was. I said one of the things I want to do is, I want to have different versions of points. So I've got now things that are Cartesian points. I could do the same thing, I could build polar point. I wanted to show it to you here. I've got a class called polar point, which is right there, and same kind of thing, I can create instances of it, and then assign to them things like a radius and an angle, make instances of those. OK, John?

PROFESSOR 2: I just want to maybe mention that in some of the reading, you'll see terms like object equality and value equality, instead of shallow equality and deep equality.

PROFESSOR: Right, so, this object, this is, right, value quality. Right. And you will see both terms used. Some people like to use shallow and deep, object and value, but they're talking about the same thing, which is it the same object or is it the same, in this case, set of values, depending on what you want to define as you use it. OK, so as I said, now I can go off and I could create a different class. I've got Cartesian points, I could create a polar points. And I'm going to run it in a sec, but you can see, the same kind of idea. I define a class call polar point, I create a couple of them, and I give them a radius and an angle. And then I could do things like again, say, okay having done, that let me just run it here, run that, so I've now got polar point 1, and polar point 2. I can say is polar point 1 the same as polar point 2, and the answer should be no. And then I could say well, gee, are they the same point? Oops. What happened? Well it bombed out. Because, what was I expecting to do? I was expecting to compare x- and y- values, not radius and angle. And so this doesn't know how to do it, it doesn't have a method to deal with it, so it complains.

So what's my problem here, and this is what I want to now lead up to. I could imagine writing another function for same point, and I have to give it a name like same point polar, and same point Cartesian. A different function to compare polar versions of these points. But that's starting to get to be a nuisance. What I'd really like to do is to have 1 representation for a point that supports different ways of getting information out, but has gathered within it, a method or a function for dealing with things like how do I know if it's the same point or not. So I want to take this idea classes now, and I want to generalize it. Right, and that is going to lead us then to this funky looking thing. Right there, and I'd like you to look at that in your handout. OK, I'm going to go back and rebuild the class. Ok, and again, I'm going to remind you, the class is this template. But now I'm going to change it, so what is that new version of class say. I'm going to call it c point just to make it a little shorter. You can see inside of it, it's got a set of definitions for things like functions. And that first one is this kind of interesting thing, it's two underbars, init, and two underbars. Underscores, I guess is the right way to say it, not underbars. Right that's a specific name, and what it basically says is, when I call the class instance. That's a bad mistake. When I call the class definition, that is I call c point, I'm going to call it with a specific set of arguments. And what is it going to happen is that init is going to then apply. It's going to apply to those arguments.

So let me in fact show you an example. I've got a definition of Cartesian point, I've got a definition of polar point. Let me just run these to get them in there. Now let's do the following. Let's let p be Cartesian point, and we'll give it a couple of values. OK? So what happened? Notice in the class definition here, is there, this is the first thing that's got called, and I just called with the value for x and the value for y, and it went off and did something for me. Does that look right? This is where you all hate it, I get no eye contact anywhere. Anything look odd about that? I said. When I call this class definition, it calls init, and I give it an x and a y value. How many arguments does init take? Three. How many arguments did I give it? Two. What in the world's going on? Well, this is a piece of object-oriented coding that we get to talk about a little bit. There's this weird extra variable in there called self. So what is self? And I have to admit, I did the standard thing you do every time you run across something you don't know about, you go to Wikipedia. So I went and looked up self in Wikipedia, and I have to read it out.

Wikipedia informs us that the self is the idea of a unified being, which is the source of an idiosyncratic consciousness. Moreover, this self is the agent responsible for the thoughts and actions of an individual to which they are ascribed. It is a substance which therefore endures through time, thus thoughts and actions at different moments of time may pertain to the same self. OK, how do we code that up? Sounds like an AI problem, I guess right?

But there's actually hidden in there an important element, and that is, when I create an instance, I have to be able to get access to the things that characterize that instance. I won't say that they're thoughts and emotions or things, but what characterizes an instance here, it's the internal parameters that specify what is going on. So in fact what happens inside of an object-oriented system, and particularly in Python's object-oriented system, is the following. When we call init, it's going to create the instance, all right, just as we said before. But in particular, it's going to use self to refer to that instance. Right, so let me say this a little differently.

I have a class definition. It's actually an object somewhere. It has inside of it all those internal definitions. When I call that class definition, it calls init. Init creates a pointer to the instance. And then it needs to have access to that, so it calls it, passing in self as the pointer to the instance. That is, it says it has access to that piece in memory, and now inside of that piece of memory, I can do things like, as you see here, define self dot x to be the value passed in for x.

What's that doing? It's saying where's self pointing to? Inside of that structure, create a variable name x, and a value associated with it. Notice what I also do here, I create self dot y, give it a value, and then, oh cool, I can also set up what's the radius and angle for this point, by just doing a little bit of work. OK, in fact if you look at what it does there, just put the pointer over here, it says, get the value of x that I just stored away, square it, add it to the value of y squared that I just stored away, and then take square root, pass it back out. So I just computed the radius of that particular thing. Right? Compute the angle the same way, just using the appropriate things. So the idea is that self will always point to the particular instance.

Now you might say, why? Why do it this way? Well, basically because it was a design choice when the creators of Python decided to create the language, they basically said, we're always going to have an explicit pointer to the instance. Some other object-oriented programming languages do not provide that pointer. This is kind of nice in my view, I don't know if John, you'd agree, but this is explicit. It actually lets you see how to get access to that pointer so you know what you're referring to. But it's simply design choice. So another way saying it again is, when I call the class definition, by default I'm going to look to see is there an init method there, and if there is, I'm going to use it. First argument by convention is always self, because it has to point to the instance, and then I pass, in this case, another couple of arguments in.

OK, now, if I actually do this, and I'm going to show you the example, I just, what did I type over there, I got p was a c point. If I want to get values back out, I could in fact simply send to that instance a message, in this case I could say p dot x. In fact let's do it. If I do that over here -- aha -- it gets me back the value. Now let me spend just a second to say, what was this actually doing? p is an instance. It knows, or has stored away, and in fact let's look at it, if we look at what p does, p says -- it says reading through a little bit of this stuff here, it says -- it's a kind of Cartesian point, it's an instance, there's actually the memory location that it's at, that's why I say this idea of it's an instant at a specific spot. It knows that it came from this class, c point. So when I type, I'm sorry, I shouldn't say type, when I write, although I would have typed it, p dot x, here's what basically happens. p is an instance, it's being sent a message, in this case the message x, it says I want the x-value back out. p knows that it is a kind of Cartesian point, it actually goes and gets, if you like, the class definition up here. And is able to then say, inside of that class definition, find the value of x. All right, now, that's one of the ways we could get things out, but in fact it's really not a good way.

A better way to do this would be the following. If I could type. What did I just do there? One of the things that I defined inside my class definition here was an internal method. That method has a name, obviously, and what does it do? It's going to go off and get the values of x and y attached to this thing and return them to me. And that's one of the things I want. I would like my classes to have methods. So you can access the values of the specific instance. Now, this is still a nuance, why would I like to do this? Well this is leading up to why I want to gather things together in classes to start with. It's perfectly legal in Python to type that in and get the value back out. As I said, I would prefer to do something that uses an accessor that I just wrote. So p dot Cartesian is a kind of accessor, it's getting access to the data. And here's why I'd like to have it. Right now, I still have the problem that those classes, those instances of classes, are exposed. What do I mean by that? Here's something I could do. Let's do it in fact. OK. What point in the plane does p now point to? X-axis is foobar y-axis ought to be foobass something else, right? I know it looks like a simple and silly little example, but at the moment, I still have the ability to go in and change the values of the parameters by that little definition. And this makes no sense. And this is because I don't have something I would really like to have, which is data hiding.

So you'll see lots of definitions of this. I think of data hiding as basically saying, one can only access instance values, or, we'll call them that, instance values through defined methods. And that's a wonderful thing to have because it gives you that modularity, that encapsulation that basically says, when I create a point, the only way I can get at the values, is by using one of the defined methods, in this case it could be Cartesian, and get all the pieces of that. Unfortunately, Python doesn't do this. Which is really a shame. Or another way of saying it is, please don't do that. Don't go in and change the values of things by using the direct access. Have the computational hygiene, if you like, to only go through accessors, only go through methods that are actually provided to you as you do this. I actually don't remember, John, C++ does have data hiding, I think, right?

PROFESSOR 2: And not only shouldn't you change it, you shouldn't even read it.

PROFESSOR: Exactly. What you're going to see in a second I violated in some of my code, which Professor Guttag is going to yell at me shortly because I should have done it through accessors, but, he's exactly right. A good, hygienic way of doing this is, not only do I not go in and change things except through a pre-defined method, I shouldn't read it other than through a pre-defined method. I should use Cartesian or polar to pull out those pieces of it.

Once I've got that, you notice I can now define a polar point, same way. Notice I've now solved one of my problems, which is, in each one of these cases here, I'm creating both x y and radius angle values inside of there. If it's in polar form I passed in a radius and angle and I'll compute what the x- and y- value is. If its in Cartesian form I'll pass in an x and y and compute what a radius and angle is. But it now says that in any, in no matter what kind of form I made it from, I can get out that kind of information. So for example I defined p, remember back over here, as a Cartesian point, but I can actually ask for its polar form. It's there accessible to me. OK, this is great. Just to drive home one more reason why I don't want to have changes to the values other than through pre-defined things. Notice what happens if I do the following. I could say I want to change the radius of this particular thing. OK, perfectly reasonable thing to do. And if I go look at the polar form of this, OK, good, looks right, right? It's now got a different radius, same angle, so I just changed the radius of it.

Oh, but what happened to the Cartesian form. I should have done this earlier by typing the Cartesian form earlier, so let me go back to where I was, sorry for that, let me go make this a 1 again. If I look at the Cartesian, oh, I did have the Cartesian form, don't mind me while I mutter to myself here quietly. Yeah, that's right, I did screw that up badly.

All right, we try one more time, here we go, let's try one more time. We'll make p a new point, ok? There's the Cartesian representation of it, which is right, (1,2). Here's the polar representation of it, some random set of numbers which makes sense. If I now say, I'm going to go ahead and change the radius of this, something, my polar form did it right, but what happened to the Cartesian form? Ah yes, didn't change. Which makes sense if you think of my code. I didn't have anything in there that says, if you change one of these values, other values depend on it, and I want to make that change to it. So this is one more example of stressing why I only want to come access to the instances through defined methods. Because I could've built that in, it says if you change the value of this thing, by the way you need to change recompute those other values in order to make this hold up.

OK, so what else do I have then in my little class definitions here? So, I've got an init in both cases. I don't have to put an init in, but it's again, usually a good idea to put that in originally. I've got and init that says, when you create an instance, here's what you do. Notice that that typically also defines for me what the internal variables are, what the internal characteristics of the class are going to be. Again, I could have some other functions to compute things, but this is typically the place where I'm going to put them in. So this is giving me now that template, better way of saying it, all right, a template now, for a point is x, y, radius, angle. And I can see that in those pieces there. And then I've got some things that get me back out information about them. But I got a couple of other of these strange looking things in there with underbars to them. So let's look at what some of the traditional methods for classes are in Python. I have init. This is what's actually going to create the instance, instantiate it, create what the set of variable values are for it. OK, I have another one in there, underbar, underbar, str. Anybody have a sense of what that's doing? What's s -- sorry, I heard something, sorry go ahead.

STUDENT: Display what I have.

PROFESSOR: Displaying what I have. Thank you. Yeah, I was going to say, think about what does str do, in general? It converts things into a string type. How do we typically print things, we convert them to strings. So str is basically telling us how we want to have it printed out. OK, in fact if we look at this, if I say, print of p, it prints it out in that form. Now this is actually a poor way to do it, because you might say, well, it's just the list. But remember, it wasn't a list. What does it do? It says, if I want to print out something I built in Cartesian form up here, says, again, I'm going to pass it in a pointer to the instance, that self thing, and then I'm going to return a string that I combine together with an open and close paren, a comma in the middle, and getting the x-value and the y-value and converting them into strings before I put the whole thing together. So it gives me basically my printed representation. OK. What else do I have in here? Well, I have cmp. My handout's wrong, which I discovered this morning after I printed them all out. So the version I'd like you to have uses, that, greater than rather than equals that I had in my handout. What's cmp doing as a method? Yeah?

STUDENT: Comparing values?

PROFESSOR: Yeah, comparing values, right? And again, it's similar to what cmp would do generically in Python. It's a way of doing comparisons. So this is doing comparisons. Now, I put a version up there, I have no idea if this is the right way to do comparisons or not. I said both the x- and y- coordinates are bigger, then I'm going to return something to it. And I think in the polar one I said, if, what did I do there, I said, yeah, again if the x and y are greater than the other one, I'm going to return them to it. The version in the handout, what was that actually doing? You could look at the handout. Well I think it was comparing, are they the same? So that would actually be another method I could put in. Underbar underbar eq, underbar underbar. Would be a default or generic way of doing, are these things the same? OK, in each case, what these things are doing, is they're doing, what sometimes gets referred to as operator overloading. I know you don't remember that far back, but in about the second lecture I made a joke of Professor Guttag which, you know, you didn't laugh at, he didn't laugh at, that's okay. In which I said, you know, I didn't like the fact that things like plus are overloaded, because you can use plus to add strings, you can use plus to add numbers, you can use plus to add floats. And he quite correctly, because he's more senior than I am, more experienced than I am, said it's actually a good thing. And he's right. Most of the time.

The reason I say that is, by having operator overloading I can use 1 generic interface to all of the objects that I want to use. So it makes sense to be able to say, look for many methods I do want to have a way of doing comparison, and I don't have to remember, at top level, what the name of the comparison method was. I can simply use the built-in Sc -- about to say Scheme again -- the built-in Python comparison operation. Say, are these 2 things the same? Same thing with cmp, that's just saying greater than, and greater than now can apply to strings, it can apply to floats, it could apply to points, it could add other pieces into it. So there are some downsides, in my view, to doing operator overloading, but there's some real pluses. And the main one is, I get to just decide, how do I want to use this, and call it. Yes, ma'am?

STUDENT: [INAUDIBLE]

PROFESSOR: Right, cmp other, so how would I call this? A good question. Here's the way I would call it. Let me give you, I'm going to create, a polar point, I'm going to call it q, and we'll give it some random values. OK, and now I want to know, is p greater than q? Now happens to return true here, but the question is, where's the other come from? P is a particular object type. When I try and evaluate that expression of greater than, is going to go into the class to say greater than is a comp method. So let me say it very carefully here. When I evaluate, yeah, when I evaluate this, p is an instance of a point, in this case it was actually a Cartesian point, it sends a message to the instance, which sends a message to the class, to get the cmp method from the class. And that then gets applied to itself, just p, and one other argument, which is the second piece there, so other points to the second argument that was present. OK. John?

PROFESSOR 2: -- other, it could have said who or zort or --

PROFESSOR: Yeah, sorry, that was part of the question, I could have a picked foobar could put anything in here. It's simply, notice the form of it here is, it's going to take two arguments, and you're right, self is the original instance. This says, I need a second argument to it, and that second argument better be a point so I can do the comparison. Yes ma'am?

STUDENT: [INAUDIBLE]

PROFESSOR: What do you think happens? Sorry, the question was, what happens if I said p is less than q? Got it, yes? Seems pretty obvious, right? Next time I bring the right glasses. It's still calling cmp, but it's knowing that cmp is just reversing the order of the arguments. Ok, which makes sense. If greater than takes, expects, arguments in order x y, less than simply takes greater than, but with the arguments reversed. OK, so I don't have to, it's a great question, I don't have to create a second one for cmp. Cmp is just saying, is this bigger than, and if I want to reverse it, it goes the other way. Question?

STUDENT: [INAUDIBLE]

PROFESSOR: Or equal equal? Let's try equal equal because I didn't define it here. It says they're not the same, and boy, I need help on this one, John, it's not, there's no pre-defined eq in there.

PROFESSOR 2: So, what cmp does, and maybe this isn't exactly the right way to write is, is cmp actually returns 1 of 3 values. A 0, minus a positive value, zero or a negative value, depending upon whether it's less than, equal, or greater than.

PROFESSOR: Right.

PROFESSOR2: So it's not really a Boolean-valued function. It has 3 possible values it could return.

PROFESSOR: And so in this case, it's using the same piece, but it's returning that middle value that says they're actually the same. Right, one the things you can see now is, we start building up classes, we get these methods. So you can actually say, how do I know which methods are associated with the class? For that, we can call dir. And what it does, is it gives me back a listing of all the things, all the methods, that are associated with it. Some of which I built: cmp, init, str. And there, notice, are the internal definitions and there are the internal variables. And in fact I should've said, we often call those things fields. So inside of an instance, associated with an instance, we have both methods and fields. These are both altogether called attributes of the instance. And then there were a couple of other ones in there that I hadn't actually dealt with.

The reason I want to point this out to you is, if we go back up to the kinds of data objects we started with, floats, ints, strings, they actually behave the same way. They are instances of a class, and associated with that class is a set of methods. So for example, I can say, what are all the methods associated with the number, or the integer 1? And you probably recognize some of them in there, right, absolute value, add, comp, cors, well we didn't do cors, we did a bunch of other things. It could also say, what are the methods associated with the string, 1. I'm sure you can quickly graph it, but notice they aren't the same. That makes sense. We have some set of things we want to do with strings, and different set of things we want to do with numbers. But underlying Python is the same idea. These are instances of a class, and associated with that class are a set of methods, things that I can deal with. So this is a handy way of being able to see, what are in fact the methods that are available if I don't happen to remember them, and want to go back to them.

OK, I want to spend the last few minutes just showing you a couple of other things that we can do in here. Let me see where I want to go with this. So let's add one more piece to this. OK, now that I've got points, I might want to do something with points. So an easy thing to do in planar geometry is I want to make a line segment. It's got a start point, it's got an end point. Right, if you want to think of it back over here. There's a line segment, it's got a starting point and ending point. Well, I can do the same thing. And the reason I want to use this as an example is, here's my little definition of segment. Again, it's got an initializer, or an instance creator, right there. Takes a start and an end point, just going to bind local variable names start and end to those pieces. But notice now, those aren't just simple things like numbers, those are actually points. And that's where the modularity comes in. Now I have the ability to say, I've got a new class, I can create instances of a line segment, and it's elements are themselves instances of points. OK? And then what might I want to do with the segment? I might want to get the length of the segment. And I know it's kind of, you can see it on your handout, it has the rest of the pieces over here.

Ok, what's the geometry say? The length of a line segment? Well, it's Pythagoras, right? I take the difference in the x-values, squared, the difference in the y-values, squared, add them up, take the square root of that. Notice what this says to do. It says if I want to get the length of a segment, going to pass in that instance, it says from that instance, get the start point, that's the thing I just found. And then from that start point, get the x-value. Same thing, from that instance, get the endpoint, from that end point get the x-value, square. Add the same thing to the y-values, squared, take the square root. Yes, ma'am?

STUDENT: So are you entering a tuple in for start and end?

PROFESSOR: No. I'm entering -- well, let's look at the example right down here. In fact, let me uncomment it so we can look at it. All right. I'm going to uncomment that. So notice what I'm going to do. I'm going to build, this case, a Cartesian point, I'm going to build a second Cartesian point, and my segment passes in those class instances. All right, they're not tuples, they're simply an instance with some structuring. And in fact if I go off and run this, OK, what I was printing here was s 1 dot length, and that's -- What is it doing? S 1 is a segment. It has inside of it pointers to 2 points which are instances. And when I call length on this, it takes that starting point, sends it the message saying give me your x-coordinate, takes the endpoint, says give me your x-coordinate, and add them together. Now, I prefaced this a few minutes ago about saying Professor Guttag wasn't going to like me. He doesn't like me generally, but that's between he and I. He beats me regularly at tennis, which is why I don't like him. Sorry, John. This is being taped, which is really good, isn't it? So why am I saying that? I said that if I was really hygienic, and you can now wonder about how often do I shower? If I was really hygienic. I would only ever access the values through a method. And I'm cheating here, right, because what am I doing? I'm taking advantage of the fact that start is going to be a point, and I'm just directly saying, give me your x-value. So I don't know don't, John, I would argue if I'd written this better, I would have had a method that returned the x- and the y- value, and it would be cleaner to go after it that way. This is nice shorthand, all right, but it's something that in fact I probably would want to do differently.

Why would I want to do it differently? Imagine that I've written code like this, written a bunch of code. And I originally decided I was going to have as points, it's going to have internal values of an x and a y. And then somewhere along the line, I decide to store things in a different representation. If I had had a clean interface, that I had a specific method to get those values out, I wouldn't have to change anything. Other than that interface. But here, if I decide I'm going to store things not in x and y, but with some other set of names, for example, I've gotta go back into these pieces of code that use the points, and change them. So I've lost modularity. I'd really like to have that modularity that says, I'm only going to get access to the values, not by calling their names, but by calling some specific method to get access to their names. You could argue, well, x is in some sense inherently a method, but it's not nearly as clean as what I would like.

And the last piece I want you to see here, and then I'll let you go is, notice now how that encapsulation, that binding things together has really helped me. Given the abstraction, the notion of a point as an instance with some values, I can now start building segments. And I could now extend that. I could have, you know, polygonal figures, that are a sequence of segments. And I would be able to simply bury away the details of how those other instances are created from how I want to use them by simply calling methods on the classes. We'll come back to this next time.