PROFESSOR: So in this final segment today, we're going to talk about set theory just a little bit. Because if you're going to take a math class, if you're going to be exposed to math for computer science, it's useful to have at least a glimmering of what the foundations of math looks like, how it starts and how it gets justified. And that's what set theory does. In addition, we will see that the diagonal argument that we've already made much of played a crucial role in the development and understanding of set theory.

So let's begin with an issue that plays an important role in set theory and in computer science, having to do with the idea of taking a function and applying it to itself, or having something refer to itself. And this is one of these things that's notoriously doubtful. There's all these paradoxes.

But maybe the simplest one is when I assert this statement is false. And the question is, it true or false? Well, if it's true, then it's false. And if it's false, then it's true. And we get a kind of buzzer. It's not possible to figure out whether this statement is true or false. I think we would deny that it was a proposition. So that's a hint that there's something suspicious about self-reference, self-application, and so one. On the other hand, it's worth remembering that in computer science, we take this for granted. So let's look at an example.

Here's a simple example of a list in Scheme Lisp notation, meaning it's a list of three things, 0, 1, and 2. And the black parens indicate that we're thinking of it as an ordered list.

Now the way that I would represent a list like that in memory, typically, is by using these things are called cons cells. So a cons cell has these two parts. The left hand part points to the value in that location in the list. So this first cons cell points to 0, which is the first element in the list. The second component of the cons cell points to the next element to the list. And so here you see 1 pointing to the third element of the list. And there you see 2. And that little nil marker indicates that's the end of the list. So that's a simple representation of a list of length three with three con cells.

One of the things that computer science lets you do and many languages let you do is you can manipulate these pointers. So using the language of Scheme, what I can do is I'll do an operation called setcar where I'm taking, in this case, I'm setting the second element of L, that is this cons cell, to L. And setcar is saying, let's change what the element in the left hand part

of this cell is. This is where the value of the second element is. Let's change the value of the second element to be L.

What does that mean as a pointer manipulation? Well, it's pretty simple. I just move this pointer to point to the beginning of the list L. And now I have an interesting situation, because this list now is a list it consists of 0 and L And 2. It's a list that has itself as a member.

And it makes perfect sense. And if you sort of expand that out, L is this list that begins with 0. And then its second element is a list that begins with 0. And the second element of that list is a list that begins with 0, and so on. And then the third element of L is 2, and the third element of the second element of L is 2, and so on. It's an interesting infinite nested structure that's nicely represented by this finite circular list.

Let's look at another example where, in computer science, we actually apply things to themselves. So let's define the composition operator. And again, I'm using notation from the language Scheme. I want to take two functions f and g that take one argument. I'm going to define their composition.

The way that I compose f and g is I define a new function, h of x, which is going to be the composition of h and g. The way I defined h of x is I say apply f to g applied to x and return the value h. So this is a compose is a procedure that takes two procedures f and g and returns their composition, h.

OK, let's practice. Suppose that I compose the square function that maps x to x squared, and the add1 function that maps x to x plus 1. Well, if I compose the square of adding 1, and I apply it to 3, what I'm saying is let's add 1 to 3, and then square it. And I get 3 plus 1 squared, or 16, because the add1 and then square it is the function that's the composition of square and add1.

Now I can do the following. I could compose square with itself. If I take the function, square it, and square that, I'm really taking the fourth power. So if I apply the function composed of square square to 3, I get 3 square square, or 81, or 3 to the fourth. All makes perfect sense.

Well now let's define a compose it with itself operation. I'm going to call it comp2. comp2 takes one function f. And the definition of comp2 is compose f with f.

And if I then apply comp2 to square and 3, it's saying, OK, compose square and square. We just did that. That was the fourth power. Apply it 3. I get 81.

And now we can do some weird stuff. Because suppose that I apply comp2 to comp2, and then apply that to add1, and apply that to 3. Well that one's a little hard to follow, and I'm going to let you think it through. But comp2 of comp2 is compose something four times. And when you do that with add1, what happens is that you're adding 1 four times to 3. If I comp2 of comp2 of square, I'm composing square with itself, and then composing that with itself. I'm really squaring four times. And I wind up with 2 to the fourth, or 16, is the power that I'm taking. And so compose2 of compose2 of square of 3 is this rather large number, 3 to the 16th.

It could be a little bit tricky to think through, but it all makes perfect sense. And it works just fine in recursive programming languages that allow this kind of untyped or generically typed composition.

So why is it that computer scientists are so daring, and mathematicians are so timid about self-reference? And the reason is that mathematicians have been traumatized by Bertrand Russell because of Russell's famous paradox, which we're now ready to look at.

So what Russell was proposing, and it's going to look just like a diagonal argument is, Russell said, let's let W be the collection of sets s such that s is not a member of s. Now let's think about that for a little bit.

Most sets are not members of themselves. Like the set of integers is not a member of itself because the only thing in it are integers. And the power set of integers is not a member of itself because every member of the power set of integers is a set of integers, whereas the power set of integers is a set of sets of those things. So those familiar sets are typically not members of themselves. But who knows, maybe there are these weird sets like the circular list, or a function that can compose with itself, that is a member of itself.

Now let me step back for a moment and mention where did Russell get thinking about this. And it comes from the period in the late 19th century when mathematicians and logicians were trying to think about confirming and establishing the foundations of math. What was math absolutely about? What were the fundamental objects that mathematics could be built from, and what were the rules for understanding those objects? And it was pretty well agreed that sets were it. You could build everything out of sets. And you just needed to understand sets, and then you were in business.

And there was a German mathematician named Frege who tried to demonstrate this by developing a set theory very carefully, giving careful axioms for what sets were. And he showed how you could build, out of sets, you could build the integers. And then you could build rationals, which are sort of just pairs of integers. And then you could build real numbers by taking collections of rationals that had at least upper bound. And then you keep going. You can build functions and continuous functions. And he showed how you could build up the basic structures of mathematical analysis and prove their basic theorems in his formal set theory.

The problem was that Russell came along and looked at Frege's set theory, and came up with the following paradox. He defined W to be the collection of s in sets such that s is not a member of s. Frege would certainly have said that's a well defined set, and he will acknowledge the W is a set.

And let's look at what this means. This is a diagonal argument. So let's remember, by this definition of W, what we have is that a set s is in W if and only if s is not a member of s. OK, that's fine. Then just let s be W. And we immediately get a contradiction that W is in W if and only if W is not in W.

Poor Frege. His book was a disaster. Math is broken. You can prove that you're the pope. You could prove that pigs fly, verify programs crash. Math is just broken. It's not reliable. You can prove anything in Frege's system, because it reached a contradiction. And from something false, you can prove anything.

Well Frege had to book. It was a vanity publication. And the preface of it had to be rewritten. And he said look, my system's broken. And I know that. And Russell showed that unambiguously. But I think that there's still something here that's salvageable. And so I'm going to publish a book. But I apologize for the fact that you can't rely on the conclusions. Poor Frege. That was his life work gone down the drain.

How do we resolve this? What's wrong with this apparent paradox of Russell's? Well, the assumption was that W was a set. And that turns out to be what we can doubt. So the definition of W is that for all sets W, s is in W if and only if s is not in s. And we got a contradiction by saying OK, substitute W for s. But that's allowed only if you believe that W is a set.

Now it looks like it ought to be, because it's certainly well defined by that formula. But it was well understood at the time that that was the fix to the paradox. You just can't allow W to be a

set.

The problem was that W was acknowledged by everybody to be absolutely clearly defined mathematically. It was this bunch of sets. And yet, we're going to say it's not a set. Well, it's OK. That will fix Russell's paradox. But it leaves us with a much bigger general philosophical question is, when it is a well defined mathematical object a set, and when isn't a set? And that's what you need sophisticated rules for. When is it that you're going to define some collection of elements, and you could be sure it's a set, as opposed to something else-- called a class by the way-- which is basically something that's too big to be a set, because if it was a set, it would contain itself and be circular and self-referential.

Well, there's no simple answer to this question about what things are sets and what are not sets. But over time, by the 1930s, people had pretty much settled on a very economical and persuasive set of axioms for set theory called the Zermelo-Fraenkel set theory axiom.