PROFESSOR: Good morning. I want to start today's lecture with, I guess one could call it, a confession. This isn't a real 6.00 lecture. I'm standing here on June 30th in an empty classroom dressed, for some bizarre reason, as if it's the winter. I guess it's what the video folks would call continuity. What happened is Professor Grimson gave a beautiful lecture 13, for which we have a lovely picture and no sound.

We decided that probably those of you watching this in OpenCourseWare would be unhappy just to watch Professor Grimson and not hear him, so we're now re-taping the lecture. This is, actually, one of two lectures we're going to be re-taping because we had video problems. So when I say something hilariously funny, and there is no laughter in the room, it's because the room is empty. Do me a favor and laugh out there in video-land, so maybe at least someone will respond.

OK, the last lecture closed with a slight mystery, not a great mystery, but a little mystery. We ran our simulation of the drunkard's walk and got a result that wasn't credible. How did we know it wasn't credible? Well we had worked out some details on the blackboard of what we thought would happen with a small number of steps, and the results we got didn't match that. That told us we had something wrong. We asked you to go think about it and come back today prepared to tell us what was wrong.

Well since there is no one here to ask, I'm not going to ask the class to fix it, but instead I have had the laborious task of fixing it myself. So let's look at it. The problem was in SimWalk. And what we had is we had the wrong argument here. We had numTrials Instead of numSteps. And so it didn't make any sense. I've now fixed it, and I, now, want to run it. Well clearly, what I should do to begin with, is run it on some examples for which we already know the answer.

So I'll change drunkTest to instead of running it on large numbers of steps, run it on just a few. And let's see what we get. We run a lot of trials here, since it's a short trial. And what we see is for 100 trials of 0 steps, the mean was 0, the max was 0, the min was 0. Well that's exactly what we saw when we looked at it on the board. It should happen. And the same thing with 1.

Everything worked the way it was supposed to work. It doesn't tell us the program is perfect. It does tell us it works in at least two examples, which was better than we had last time. All right now let's look at it on a larger set of examples. I think I won't run 100 trials here, because it'll take a little longer than we want. So let's run maybe 20. Let's see what we get.

Well it's running through here. Now we're getting examples that seem much more credible. When we take 10 steps, the mean is 2.85, and the max is between 6 and 1, or the max is 6, the min is 1. It's what we would hope for, that there's some dispersion there. And as we get up higher, we see that for a 100,000 steps, the mean is 248, and there's quite a spread between the max and the min. Finally we get to look at the question that had us writing this code in the first place-- how far should we expect this drunk to be, given a particular amount of time.

Well we can look at these numbers and try and think about them in our heads, but in fact, it's a lot easier to look at a picture. And this will get us to a theme that we'll be getting to shortly in the course of how do we visualize data. So here we have a visualization. What I'm plotting, here, is the mean number of the mean distance against the number of steps for this random walk. And what we can see is, as we knew, at 0, it's 0, and then it sort of goes up. And I've taken it only up to a 1,000. And at 1,000 we're somewhere, it looks like, around 25 steps. So we can learn something.

Well what can we learn from this? How fast is it growing? Well it seems to grow pretty fast, and then it flattens out. It looks to me like, roughly speaking, the distance is growing as sort of close to the square root of the number of steps. Well I could look at it more closely. We, actually, know this is not exactly the square root. Right,

the square root of a 1,000 is not going to be 25, but I'm not going to delve into the details of that. It's, actually, a little bit complicated. We could derive it, but we won't, because I want to get to what I think is a more important message.

How much should we infer from this or from the numbers I displayed before, and I want to say not too much. Because what we saw, if we go look at what we had before, is quite a dispersion between the max and the min. And furthermore, if we run it again, the 20 trials, we're getting different answers. So what you can see, here, is for 10,000 steps, here, my mean was 78, and here, for 10,000 steps, my mean is 90. Here, the mean was 279. Here, it was 248. The maxs and the mins are different. So I don't want to read too much into that graph.

Now one of the issues we should have asked is, well it's the mean of how many trials. I didn't tell you. And to be honest, I don't quite remember. I think it was 20. But I don't have enough information to interpret it. I need a lot more and that's going to be what this whole next unit of the course is about is how do we think about the results of programs when the programs themselves are stochastic. And this is important because, as you will see, not only in this course, but as you progress in your careers, if you're involved in engineering or science, is that almost everything in the real world is stochastic.

And in order to think about it, we have to really think pretty hard about what those things mean. So I want to, now, pull back and talk a little bit, sort of, philosophically about that, about the role of randomness in computation-- something you probably haven't seen a lot, if at all in the other courses you've taken if you're a freshman or sophomore, and that's because there's something really comforting about Newtonian mechanics.

When I first learned physics, it was really comforting. I learn the physics of Isaac Newton. You push down on one end of the lever, the other end of the lever goes up. You throw a ball up in the air, it travels a parabolic path and lands. F equals MA, that wonderful rule of physics. Everything happened for a reason. It was predictable. It was a great comfort to some of us about it. And for centuries, that's the way the

world thought, from almost the beginning of the times of science to, really, if you look at history of science, to today, people believed the world was deterministic. And they liked it.

Then along came the so-called Copenhagen Doctrine and quantum physics in the 20th century, and the comforting world of Newtonian physics disappeared. The Doctrine, the Copenhagen Doctrine, led by the physicists Bohr and Heisenberg, argued that at its most predictable and most fundamental level, the behavior of the physical world cannot be predicted. One can make probabilistic statements of the form x is highly likely to occur, but not statements of the form x is certain to occur-- period.

I can make the probabilistic statement that if I take this laser pointer and put it on the table, it won't fall through. The molecules won't separate in such a nice way that it just drops through, but I can't promise it won't happen. Truth is I'm going to make that promise anyway, because I believe in probability, and it's highly, highly probable. But they tried to say that, in fact, the world is all stochastic. Everything is probabilistic.

Other distinguished physicists at the time, most notably Einstein and Schrodinger, vehemently disagreed. This debate, it's hard to believe, actually, roiled the world of physics, philosophy, and religion. The heart of the debate was the validity of something called causal non-determinism. The idea here-- causal means caused by previous events. So causal non-determinism was the belief that not every event is caused by previous events. Einstein and Schrodinger found this view philosophically unacceptable, as exemplified by Einstein's often quoted comment, "God does not play dice."

What they argued is for something called predictive non-determinism. The concept here was that our inability to make accurate measurements about the physical world makes it impossible to make precise predictions about the future. So this distinction was nicely summed up again by Einstein, who said, and I quote, "the essentially statistical character of contemporary theory is solely to be ascribed to the fact that

this theory operates with an incomplete description of physical systems," i.e., things are not unpredictable, they just look unpredictable because we don't know enough about the initial states.

This question is still unsettled in science. The good news is it probably doesn't matter at all what the truth is. However you want to look at it, we have to assume that the world is non-deterministic, because we can't, actually, predict it. So there's a little experiment I sometimes do. I'll pretend to do it, even though there are no students here. I take three coins-- see if there are students in the class, I ask the students to give me coins, and then I try to steal them. But since there are no students, I'm going to have to use my own coins, and I'm going to ask, is at least one of them heads.

Well the truth is, it's completely predictable. I know the answer. But you don't know the answer, because you can't see these coins. And so you might as well assume it's probabilistic, and guess well, he put 3 coins down at random, the odds of each one being a head is 1/2, so probably 1 of the 3 is a head. And you're relying on probability, because you don't know what's going on-- predictive non-determinism.

OK, and that's the way we're going to deal with a lot of things going on for the rest of the semester. We'll look at a lot of examples where we have to act as if things are non-deterministic. And that gets us to this notion of what mathematicians call stochastic processes. A process is stochastic if it's next state depends on both the previous states and some random element.

So now what I'm going to do is pick up one of these coins, flip it in the air, put it down, and ask, again, about the state of these three coins. It depends upon the previous state, because these two coins have the same value they had in the previous state, plus a stochastic element, a probabilistic element, a random element-- the value of that coin, which I just flipped.

OK, most programming languages, including Python, include simple ways to write programs that use randomness. As we'll see, as we've already seen in Python with our drunkard's walk, we use the function random.choice, which, given a set of

values at random, chose one of those values. That function and almost all of the other functions in Python that involve randomness are implemented using something called random.random.

This function generates a random float that's greater than 0 and no greater than 1.0, So you get one of the infinite or seemingly infinite number of floating point values that are greater than 0 and no greater than 1. So let's go look at another example of the stochastic process. We're going to look at throwing dice. So I've got something called rollDie, which chooses a value between 1 and 6.

For those of you who have never gambled with dice, it's a cube. You roll it, and it has a value between 1 and 6 that shows up at random. And then I've got this little program called testRoll that rolls a bunch of dice and comes up with an answer. All right, so let's see what happens. Actually, before we do that, let me ask you-- we can look at a question. Imagine I roll it, and I run it some large number of times, 10. Would you expect to see the value-- more likely see that value or might more likely see a value that looks like this.

Which of these values is more likely to come up from my random rolls of the die? Well when I take a vote-- if I take a vote-- historically in this class, this strikes people as more likely to happen than this. But it's a trick question, because as it happens, they're equally likely. And the reason they're equally likely is each roll is independent of the previous rolls. And as we'll see in our excursion in probability and randomness, independence is a very important assumption. In a stochastic process, two events are independent if the outcome of one event has no influence on the outcome of the other.

The events are independent if the outcome of one event has no influence on the outcome of the other. So it's a bit easier to think about this, maybe, if we simplify the situation for the moment to think about flipping coins, which have either heads or tails, and I'll look at the value 0 or 1 as the examples there. That means I can use-- I have a binary die for some reason. So as we've seen before, if I flip a coin ten times, how many different possibilities, sequences of 0 and 1 are there? Well we've

seen this kind of thing a lot of times already this semester. There are 2 to the 10 binary numbers of 10 digits, so we know that there are 2 to the 10 possibilities.

Each of these 2 to the 10 possibilities are equally likely. So the number in which I have all 0's is no more likely than the number of all 1's, is no more likely than some seemingly random combinations of 0 and 1. So it's a very small probability. So what's the probability of getting all 1's? It's 1 out of 2 to the 10. What's the probability of getting all 0's? 1 out of 2 to the 10. What's the probability of any combination you would happen to pick of 0's and 1's? 1 over 2 to the 10.

I know I'm belaboring this point, but the point I want to make is that when we talk about some result having a particular probability, we are asking, essentially, the question, what fraction of the possible results have the property we're testing for. So I ask what property are all 1's. I'm saying what fraction are all 1's. If I say well, the properties are exactly four 1's. What fraction of these numbers have exactly four 1's in them? Whatever I want.

So probabilities will always be fractions. That's important because it means that when we talk about the probability of some event occurring, we know it has to be somewhere between 0 and 1. Probabilities are never less than 0. They're never more than 1. Cannot happen, guaranteed to happen, usually somewhere in between. All right, that's the key thing to remember when thinking about probabilities.

Suppose I want to ask, what's the probability of getting some sequence of coin flips other than all 1's. Well I know the probability of getting all 1's is 1 out of 2 to the 10th. I know the probability of getting some sequence of flips is 1. It's certain I'll get one of those numbers. So the answer is the probability of not getting all 1's is 1 minus 1 over 2 to the 10th.

This is an important trick to remember. Typically we have two ways of computing probabilities. We can either compute it directly, as I did when I computed the probability of getting all 1's, or we can compute the probability of something not happening by subtracting one probability from another, this 1 minus trick. And so

you'll see me using this formulation a lot of times, And we'll talk as we go forward about when you do it which way.

All right so let's go back, finally, to our six-sided die. How many sequences are there of length 10 for that? 2 to the 10th? No. 6 to the 10th. Because unlike the coin where we only had 2 possibilities, we now have 6 possibilities. So there are 6 to the 10th different sequences of rolls I could get, quite a few. So the probability of getting 10 consecutive 1's is 1 over 6 to the 10th. And of course, the probability of getting this sequence, here, is also 1 over 6 to the 10th, so we see that they are equal.

OK, we're going to spend a lot more time on probability and randomized and stochastic algorithms. But before I do that, I want to take a brief digression and return to the topic we looked at a little earlier this morning, which was data visualization, plotting. I want to do this for two reasons. One, it's really important. It's something that all of us do a lot of in the course of our work, but it also will just make it a lot easier for me to talk about probability and stochastics when I can draw some pretty pictures to illustrate what's going on.

Now many people, most of us, probably, when we're writing code to do something, focus on writing programs that perform some complicated analysis of the data, and then print something. We don't spend enough time, I think, worrying about how the results of our analyses are presented so that somebody else can make sense of them, or in fact, we can understand them better ourselves. Sometimes text is the best way, but sometimes there's a lot of truth to the Chinese proverb that a picture's meaning can express 10,000 words.

Now most of us, sort of, believe this. Why don't we do it? Well because in most programming languages, it's hard to draw pretty pictures. One of the reasons we use Python in this class is because in Python, it's easy to draw pretty pictures or, at least, to make plots. Why is that? It's because somebody-- not me-- went to the trouble of building something called PyLab. PyLab is a Python library that provides many of the facilities of something called MATLAB.

If you're an MIT student, the probability of your graduating without using MATLAB is

very low. It is something people use a lot. It's not my favorite programming language. It has its utility. I like Python, because it brings-- a lot of the features of MATLAB are easy to use in a programming language that I find much more convivial than MATLAB. All right I'm not going to give you a complete tutorial for PyLab here. It would take a long time, and it would be boring.

Instead I'm going to give you a few examples, and in fact, focus primarily on the plotting capabilities. And the good news is the plotting capabilities in PyLab are almost identical to those in MATLAB, so if you learn how to do it here, you'll already know how to do it here. For details you should take a look at-- let me make sure I write this correctly-- this website, matplotlib.sourceforge.net, and it's a very nicely put together website that will give you all of the capabilities of plotting.

Also you'll find in the class website, I've written a little chapter of a book about how to do this sort of thing, and you may also find that helpful. I should point out that PyLab is not part of the standard Python distribution. It has to be installed on your computer, and again, there are instructions about how to do this posted on the class website. All right let's start with something very simple. We'll look at that, here.

I'm beginning by importing. I'll import PyLab. You have to import it to use it. And then I'm going to plot two things. So what I want to observe, here, is I'm going to plot two vectors-- the vector 1, 2, 3, 4 and the vector 1, 2, 3, 4. These are the x-coordinates. These are the y-coordinates. So we'll get a two-dimensional plot, x versus y. And it's very important that these two vectors be of the same length. Doubtless when you're using this in your problem sets, you will screw up and you'll get an error message, which you will have a hard time interpreting, but what it probably is going to boil down to is you've done something wrong, and you're plotting things that are not the same length.

After I plot these two things, I'm going to type "pyLab.show," which will put the plots up for us to look at. So let's do that now. So the first plot in our straight line-- 1 versus 1, 2 versus 2, 3 versus 3, 4 versus 4-- got that. Then it plotted this rather funny looking zigzag we saw, kind of just randomly chosen in the other one. And the

plots will always look something like this. I should mention, if we go look at the code, pyLab.show-- if I haven't said that, the plot would not have appeared on my screen. PyLab would have produced it, but not displayed it.

You may think that's silly, but in fact it's useful, because most of the time when I'm writing programs that do plotting, I don't want to see them on my screen. I'm producing a whole bunch of plots, and I'm going to write them to files that I will then look at later or include in a paper or a lecture or something, so it makes me say that I want to see it. I should point out, depending upon your operating system, it can be pretty annoying, because if you try and do this twice in your code, something bad can happen. The code can hang.

Therefore, you should only execute pyLab.show once per program. And it should always be the last thing you executed, because once you execute pyLab.show, the program will stop running, essentially. It's annoying. I wish it weren't that way, but it is. So live with it. All right let's go look back at our graph, our plot. At the top is a title. This is the default title. It says figure 1. Later we'll see that I could have given it a much better title than that. Then it's got the values of the x and y-axes and a bunch of things down at the bottom that we could point out.

You can zoom in on the plots, using that or zoom out. You can use this funny icon. It happens to be a floppy disk icon, something that probably most of you have never seen. Congratulations if you've never seen a floppy disk. Your life is better than it would be had you had to deal with them. But that's used for saving them to a file. You can move around in it. You can get back to what the original figure was, a whole bunch of useful things. I suggest you just bring this up and play with it.

One of the things I should mention here is you'll note that when I produced this, I only use four points-- 1, 2, 3, and 4, say, for this one. Yet it looks as if I have a continuous plot. You know, it claims that 1.5 and 1.5 match. This can be very deceptive, as we'll see later on, and probably it might have been better for me to plot not lines here, but points, indicating there are only four points in this graph. It's more apparent here, where we see this funny looking zigzag, implying some

complicated relationship amongst the points, which, actually, probably doesn't exist. And again later on we'll see ways to avoid that.

OK. It is, of course, possible to produce more than one figure. So let's look at this. We'll comment this piece out for now. And we'll run this code. So this says I want to plot something on a figure I'm calling figure 1, as before, And then I'm going to go to figure 2. So now instead of plotting both of these on the same figure, I'm going to put them on separate figures. And then pyLab.savefigure will, actually, create a file in the directory in which I'm running the program and save it-- called firstsaved And then this will be secondsaved. And now I can run it.

And now I have two figures-- figure 1, as before-- well not quite as before, a different figure-- and figure 2. Again, nothing very magical there. And if we look in my directory, we should see, I hope, firstsaved and secondsaved, so if we look at secondsaved, we'll see, there it is. And just to show that I'm not cheating, you can notice that the time stamp is today at 10:34 AM, which happens to be when I'm giving this particular lecture.

All right you can put that away now and continue. Now what I want you to notice is the last one. I just gave it one argument-- 5, 6, 7 and 10. And if we look at what it plotted-- actually, I think I put that in figure 1, didn't I? You'll see, here it is. And it's made up some values for the x-axis. If you only give it one set of values, it assumes it's the y-axis, and it finds values.

All right now what values is it going to choose for the x-axis? Well this is Python, so surprise, surprise, the first value is 0, 1, 2, and 3. It's how I get the four values. Now we could look at another example, a slightly more interesting one. Comment this out, so we don't look at the boring stuff over and over again. I've written a little program to calculate interest. So I'm going to start with an initial principal, here, of 1,000, an interest rate of 5%, 20 years, and just do compound interest.

This, you all would know how to write this. And I'm going to plot it and see what we get. All right so what we have-- something here, which, sort of, shows the kind of beautiful growth you get with compound interest, what the finance people call the

magic of compounding, which will make us all rich in principle, until we see what the markets really do. But at any rate, for now we can look at it, and it looks very pretty. But I don't know what it means.

I look and say, "oh, its figure 1." Well that's not very informative, and x goes from 0 to 20, but if I haven't told you, you wouldn't know what that meant. And y from 10,000 up to 28,000, but again, you wouldn't know what that means. We see this all the time. It's not a good thing. It's a bad thing, in fact. All plots should have informative titles, and all axes should be labeled. I can't tell you the number of times I've had a graduate student show up in my office, having worked for weeks producing some data, put a plot on my desk, and say, look at this, isn't it great. And I say, I have no idea what it means.

And sometimes, I'll say, well what is the y-axis, and they end up scratching their head. They're not quite sure. You've got to label your axes. You've got to put a title. You've got to give the person a break. Well how do we do that? Well it's pretty simple. So here's this code. So I just want to get rid of the old graphs, because sometimes if you look at them, it causes you problems. Because it's now hung. It won't continue until I get rid of it. Now my shell is back.

So pyLab.title just says, OK, I'm going to call it 5% growth, compounded annually. Notice that I've put in that it's 5% interest rate and that I'm compounding it annually, not semi-annually or quarterly. The x-axis is going to be the years of compounding and the y-axis, the value of the principle in dollars. So now it's the same curve, but a far more informative picture. All right here's where I want to stop today.

We're going to come back to this topic, in probably more detail than you want, and spend quite a lot of time talking about how do we produce beautiful plots, and more importantly, how do we produce plots that are, actually, meaningful to those reading them. Thanks a lot. I'll see you in the next lecture.