PROFESSOR: You'll recall, at least some of you will recall, that last time we ended up looking at a simulation of a drunken university student wandering randomly in a field. I'm going to return to that. Before I get to the interesting part, I do want to call your attention to something that some people seemed a little bit confused by last time. So you remember that we had this thing called perform trial. And the way we basically tested the drunk, what would happen, is we would call this, passing in a time, the number of steps, the amount of time that the drunk was wondering, and the field. And there was just 1 line, that I blazed past, probably too quickly. So what that line is saying, is we, from the field, we're going to call the method get drunk, which is going to return us a drunk. And then we're going to call the move method for that drunk, passing the field as an argument. Now the question is, what would have happened if I had left off these parentheses? There. A volunteer? What would, we will do the experiment in a minute. And by the way, I want to emphasize the importance of experimentation. I was kind of surprised, for example, to get email from people saying, well, what was the correct answer to question 3 on the quiz? And my sort of response was, why don't you type it and find out?

When in doubt, run the experiment. So, what's going to happen, what would have happened, had I failed to type that pair of parentheses? Anybody? Well, what is, get drunk? It is a method. And remember, in Python, methods, like classes, are themselves objects. So get drunk would have a returned an object which was a method, and then, well, let's try it. And the error message will tell us everything we need to know. Function has no attribute move. Sure enough, the method does not have an attribute move. The instance of the class has the attribute. And so what those parentheses did, was tell me to invoke the method get drunk, so now instead of that being the method itself, it's the result returned by the method. Which is an instance of class drunk, and that instance has an attribute move, which is itself a method. This is a very common programming paradigm, and it's important to sort of lock into your heads, the distinction between the method and an invocation of the method. Because you can write either, and sometimes you won't get an error message, you'll just get the wrong answer. And that's kind of bad. That make sense to everybody? Does it not make sense to anybody, maybe is the question I should ask?

All right, we'll go on, and we'll see more examples of this kind of thing. Now last week, I ran this several times to see what would happen, and we saw that in fact contrary to everyone's, most everyone's, expectation, the longer we ran the simulation, the further the drunk was from the starting point. And we saw that by plotting how far the drunk was after each time unit. We ran it several times, we got different answers. And at that point we should ask ourselves, is that really the way to go about answering the original question? Which was, how far should we expect the drunk to be? The answer is no. I don't want to sit there at the keyboard typing in 400 examples and then, in my head, trying to figure out what's quote, typical, unquote. Instead, I need to organize my simulation so that it runs a number of trials for me and summarizes the results. All the simulations we're going to look at it, in fact almost all the simulations anyone will ever write, sort of, have the same kind of structure. We start with an inner loop that simulates one trial.

That's what we have here, right, we have happen to have a function. So when I say inner loop, what I mean is, not that I'll write a program a bunch of nested loops, but that I'll write a program with some function calls, and down at sort of the bottom of the stack will be perform trial. Which stimulates one trial of some number of seconds, in this case. And then I'll quote, enclose, unquote the inner loop in another loop that conducts an appropriate number of trials. Now a little bit later in the term, we'll get to the question of, how do we know what an appropriate number is? For today, we'll just say, a lot. And we'll talk about that a little bit more. And then finally, what we want to do, is calculate and present some relevant statistics about the trials. And we'll talk about what's relevant. I want to emphasize today the presentation of them. Last time we looked at a graph, which I think you'll all agree, was a lot prettier to look at an array of, say, 1000 numbers.

All right. So now, on your handout and on the screen, you'll see the way we've done this. So perform trial, we've already seen, so we know what the inner loop looks like. We can ignore first test, that was just my taking the code that I had in line last time and putting it in a function. And now what you'll see in the handout is perform sim and answer question. So, let's look it those. So perform sim, sim for simulation, takes the amount of time, the number of steps for each trial, plus the number of trials. It starts with an empty list, dist short for distances here, saying so far we haven't run any trials, we don't know what any of the distances are. And then for trial in range number of trials, it creates a drunk, and I'm putting the trial as part of the drunk's name, just so we can make sure that the drunks are all different. Then I'm going to create a field with that drunk in it at location 0,0. And then I'm going to call perform trial with the time in that field to get the distances.

What does perform trial return? We looked at it earlier. What is perform trial returning? Somebody? Surely someone can figure this one out. I have a whole new bag of candy here, post-Halloween. What kind of thing is it returning? Is it returning a number? If you think it's returning a number, oh, ok, you gonna tell me what kind of thing it's returning, please? A list, thank you. And it's a list of what? I'm giving you the candy on spec, assuming you'll give me the right answer. List of distances, exactly. So how far way it is after each time step? This is exactly the list that we graphed last time.

So perform sim will get that list and append it to the list. So dist list will be a list of lists, each element will be a list of distances, right? OK, so that's good. And that's now step 2. In some sense, all of this is irrelevant to the actual question we started with, in some sense. This is just the structure. Then I'm going to write a function called answer question, or ans quest, designed to actually address the original question. So it, too, is going to create a list. This is a list called means. Then it's going to call perform sim, to get this list of lists, then it will go through that and calculate the means and create a list of means. And then it will plot it, and we'll get to, before this lecture's over, how the plotting works. So I'm following exactly that structure here. It calls this function, which runs an appropriate number of trials by calling that function, and then we'll calculate and present some statistics.

So now let's run it. All right, so what have I done? I typed an inadvertent chara -- ah, yes, I typed an s which I didn't intend to type. It's going to take a little while, it's loading Pylab. Now it's running the simulation. All right, and here's a picture. So, when we ran it, let's look at the code for a minute here, what we can see is at the bottom, I called ans quest, saying each trial should be 500 steps, and run 100 trials. And then we'll plot this graph. Graph is lurking somewhere in there, it is. And one of the nice things we'll see is, it's kind of smooth, And we'll come back to this, but the fact that it's sort of smooth makes me feel that running 100 trials might actually be enough to give me a consistent answer. You know, if it had been bouncing up and down as we went, then we'd say, jeez, no trend here. Seeing a relatively smooth trend makes me feel somewhat comfortable that we're actually getting an appropriate answer. And that if I were to run 500 trials, the line would be smoother, but it would look kind of the same. Because it doesn't look like it's moving here, in arbitrary directions large amounts. It's not like the stock market.

Should I be happy? I've sort of done what I wanted, I kind of I think I have an answer now, which is 500 steps, it should be four and a half units away from the origin. What do you think? Who think this is the right answer? So who thinks it's a wrong answer, raise your hand? All right, TAs, what do you guys think? Putting you on the spot. Right answer or wrong answer? They think it's right. Well, shame on them. Let's remember, rack our brains to a week ago, when we ran a bunch of individual tests. And let's see what we get if we do that. And the point here is, it's always good to check. My recollection, when I looked at this, was that something was amiss. Because I kind of remember, when I ran the test last time, we were more like 40 away than four away. Well all right, let's try it. We'll, sometimes happens, all right, I'm going to have to restart Idol here, just, as you all, at least all who use Macintoshes know, this happens sometimes, it's not catastrophic. Sigh. So this reminds me of the old joke. That a computer scientist, a mechanical engineer, were riding in a car and the car stalled, stopped running. And the mechanical engineer said I know what to do, let's go out and check the carburetor, and look at the engine. The computer scientist said, no that's the wrong thing to do. What you ought to do is, let's turn off the key, get out of the car, shut the doors, open the doors, get back in and restart it. And sure enough, it worked. So when in doubt, reboot.

So, we'll come down, we'll do that, and we're going to call first test here, and see what that gives us. And we'll, for the moment, ignore that. Well, look at this. We ran a bunch of Homer's random walks, and maybe it isn't 40, but not even one of them was four. So now we see is, we've run two tests, and we've gotten inconsistent answers. Well, we don't know which one is wrong. We know that one of them is wrong. We don't even know that, maybe we just got unlucky with these five tests. But odds are, something is wrong, that there's a bug here. And, we have to figure out which one. So how would we go about doing that? Well, I'm going to do what I always recommend people do. Which was, find a really simple example, One for which I actually know the answer. So what would be a good example for which I might know the answer? Give me the simplest example of a simulation of the random walk I could run, where you're confident you know what the answer is. Yeah? one step, exactly, and what's the answer after one step? One. She can't catch and talk at the same time. Exactly. So we know if we simulate it, one, the drunk has moved in some direction, and is going to be exactly one step from the origin. So now we can go and see what we get. So let's do that. And we'll change this to be one, and we'll change this to be one. We'll see what the answer is.

Well. 50? Well, kind of makes me worry. 1. All right, so we see that the simple test of Homer gives me the right answer. We don't know it's always the right answer, we know it's, at least for this 1. But we know the other 1 was just way off. Interestingly, unlike the previous time, instead of being way too low, it's way too high. So that gives us some pause for thought. And now we need to go in and debug it. So let's go and debug it. And seems to me the right thing to do is to go here. Oh, boy, I'm going to have to restart it again, not good. And we'll put an intermediate value. Actually, maybe we'll do it. What would be a nice thing to do here? We're going to come here. Well, let's, you know, we want to go somewhere halfway through the program, print some intermediate value that will give us some information. So, this might be a good place. And, what should we print? Well, what values do you think you should get here? Pardon?

STUDENT: The total distance so far.

PROFESSOR: The total distance so far. So that would be a good thing to print. And what do we think it should be? We'll comment this 1 out since we think that works, and just to be safe, let's not even run 100 trials let's, run one trial, or two trials maybe. See we get. 0 and then 2. W, 0 was sort of what we expected the first time around, but 2? How did you get to be 2? Anyone want to see what's going on here? So we see, right here we have the wrong answer. Well, maybe we should see what things looked like before this? Is it the lists are wrong? What am I doing wrong here? I'll bet someone can figure this out. Pardon?

STUDENT: [INAUDIBLE]

PROFESSOR: Well, I'm adding them up, fair enough. But so tot looks OK. So, all right, maybe we should take a look at means. Right? Let's take a look at what that looks like. Not bad. All right, so maybe my example's too simple. Let's try a little bit bigger. Hmmm -- 2.5? All right, so now I know what's going wrong is, somehow not that

I'm messing up tot, but that I'm computing the mean incorrectly. Where am I computing the mean? They're only two expressions here. There's tot, we've checked that. So there must be a problem with the divisor, that's the only thing that's left. Yeah?

STUDENT: [INAUDIBLE]

PROFESSOR: Exactly right. I should be dividing by the length of the list. The number of things I'm adding to tot. So I just, inadvertently, divided by, I have a list of lists. And what I really wanted to do is, divide by the number of lists. Because I'm computing the mean for each list, adding it to total, and then at the end I need to divide by the number of lists who's means I computed, not by the length of 1 of the lists, right? So now, let's see what happens if we run it. Now, we get some output printed, which I really didn't want, but it happens. Well this looks a lot better. Right? Sure enough, it's 1. All right, so now I'm feeling better. I'm going to get rid of this print statement, if we're gonna run a more extensive test. And now we can go back to our original question. And run it.

Well, this looks a lot more consistent with what we saw before. It says that on average, you should be around 20. So we feel pretty good about that. Now, just to feel even better, I'm going to double the number of trials and see what that tells us. And it's still around 20. Line a little smoother. And if I where do 1000 trials will get a little smoother, and it would still be around 20. Maybe slightly different each time, but consistent with what we saw before, when we ran the other program. We can feel that we're actually doing something useful. And so now we can conclude, and would actually be the correct conclusion, that we know about how far this random drunk is going to move in 500 steps. And if you want to know how far he would move in 1000 steps, we could try that, too.

All right. What are the lessons here? One lesson is to look at the labels on the axes. Because if we just looked at it without noticing these numbers, it looks the same. Right? This doesn't look any different, in some sense, than when the numbers were four. So you can't just look at the shape of the curve, you have to look at the values. So what does that tell me? It tells me that a responsible person will always label the axes. As I have done here, not only giving you the numbers, but telling you it's the distance. I hate it when I look at graphs and I have to guess what the x- and y- axes are. Here it says time versus distance, and you also notice I put a title on it. So there's a point there. And look, when you're doing it.

Ask if the answer make sense. One of the things we'll see as we go on, is you can get all your statistics right, and still get the wrong answer because of a consistent bug. And so always just say, do I believe it, or is this so counterintuitive that I'm suspicious? And as part of that ask, is it consistent with other evidence? In this case we had the evidence of watching an individual walk. Now those two things were not consistent, don't know which is wrong, but it must be one of them. And then the final point I wanted to make, is that you can be pretty systematic about debugging, And in particular, debug with a simple example. Right, instead of trying to debug 500 steps and

100 trials, I said, all right, let's look at one step and four trials, five trials. OK, where in my head I knew what it should look like, and then I could check it. All right.

Jumping up a level or three of abstraction now. What we've done, is we've introduced the notion of a random walk in the context of a pretty contrived example. But in fact, it's worth knowing that random walks are used all over the place to solve real problems, deal with real phenomena. So for example, if you look at something like Brownian motion, which can be used to model the path traced by a molecule as it travels in a liquid or a gas. Typically, people who do that model it using a random walk. And, depending upon, say the density of the gas or the liquid, the size of the molecules, they change parameters in the simulation, how far it, say, goes in each unit time and things like that. But they use a random walk to try and model what will really happened. People have attempted, for several hundred years now, to use, well, maybe a 150 years, to use random walks to model the stock market. There was a very famous book called A Random Walk Down Wall Street, that argued that things happened as a, random walk was a good way to model things. There's a lot of evidence that says that's wrong, but people continue to attempt to do it.

They use it a lot in biology to do things like model kinetics. So, the kinetics of a protein, DNA strand exchange, things of that nature. A separation of macro-molecules, the movement of microorganisms all of those things are done in biology. And do that. People use it to model evolution. They look at mutations as kind of a random event. So, we'll come back to this, but random walks are used over and over and over again in the sciences, the social sciences and therefore a very useful thing to notice about. All right, we're going to come back to that. We're going to even come back to our drunken student and look at other kinds of random walks other than the kind we just looked at.

Before I do that, though, I wanted back up and take the magic out of plotting. So we've gone from the sublime, of what random walks are good for, to in some sense the ridiculous, the actual syntax for plotting things. And maybe it's not ridiculous, but it's boring. But you need it, so let's look at it. So we're doing this using a package called Pylab, which is in itself built on a package called Pylab, either pronounced num p or num pi, you can choose your pronunciation as you prefer. This basically gives you a lot of operations on numbers, numbered things, and on top of that, someone bill Pylab which is designed to provide a Python interface to a lot of the functionality you get in Matlab. And in particular, we're going to be using today the plotting functionality that comes with Matlab, or the version of it. So we're going to say, from Pylab import star, that's just so I don't have to type Pylab dot plot every time. And I'm going import random which we're going to use later. So let's look at it now.

First thing we're going to do is plot 1, 2, 3, 4, and then 1, 2, 3, and then 5, 6, 7, 8. And then at the very bottom, you'll see this line show. That's going to annoy the heck out of you throughout the semester, the rest of the semester. Because what happens is, Pylab produces all these beautiful plots, and then does not display them until you type show. So remember, at the end of every program, kind of, the last thing you should execute should be show. You don't want to execute it in the middle, because what happens in the middle is it, in an interactive mode at least, it just stops. And displays the graphs, and until you make the plots go away, it won't execute the next line. Which is why I've tucked the show at the very bottom of my script here. Inevitably, you will forget to type show. You will ask a TA, how come my graphs aren't appearing in the screen, and the TA will say, did you do show? And you'll go -- but it happens to all of us.

All right, so let's try it. See what we get. So sure enough, it's plotted the values 1, 2, 3, 4, and 5, 6, 7, 8, on the x- and y- axis. Two things I want you to notice here. One Is, that both plots showed up on the same figure. Which is sometimes what you want, and sometimes not what you want. You'll notice that also happened with the random walk we looked at, where when I plotted five different walks for Homer they all showed up superimposed on top of one another. The other thing I want you to notice, is the x-axis runs from 0 to 3. So you might have kind of thought, that what we would see is a 45 degree angle on these things. But of course, Python, when not instructed otherwise, always starts at zero. Since when we called plot, I gave it only the y-values, it used default values for x. It was smart enough to say, since we have four y-values, we should need four x-values, and I'll choose the integers 0, 1, 2, 3 as those values. Now you don't have to do that. We could do this instead. Let's try this one. What did I just do? Let's comment these two out, if we could only get there. This is highly annoying. Let's hope it doesn't tell me that I have to -- All right, so let's go here. We'll get rid of those guys, and we'll try this one. We'll plot 1, 2, 3, 4 against 1, 4, 9, 16. OK? So now, it's using 1, 2, 3, 4 as the x-axis, and the y-axis I gave it. First x then y.

Now it looks a little funny, right, you might have not expected it to look like this. You'll notice they're these little inflection points here. Well, because what it's really doing is, I gave it a small number of points, only four. It's found those four points, and it's connected them, each point by a straight line. And since the points are kind of spread out, the line has little bumps in it. That makes sense to everyone? Now, it's often deceptive to plot things this way, where you think you have a continuous function when in fact you just have a few miscellaneous points. So let's look at another example. Here, what I'm going to do, is I've called figure, and remember, this is Pylab dot figure, which says, create a new figure. So instead of putting this new curve on the same figure as the old curve, start a new one. And furthermore, I've got this obscure little thing at the end of it. After you give it the x- and y- values, you can give it some instructions about how you want to plot points, or anything else. In this case, what this little string says is, each point should be represented as a red o. r for red, o for o.

I'm not asking you to remember this, what you will discover, the good news is there's very good documentation on this. And so you'll find in the reading of pointer to plots, and it will tell you everything you need to know, all of the wizardry and the magic you can put in these strings that tell you how to do things. These are basically the same strings borrowed from Matlab. And now if we run it. Figure one is the same figure we saw before. But figure two has not connected the dots, not drawn a line, it's actually planted each, or plotted, excuse me, each point as a red circle. Now when I look at this, there's something that's not very pleasing about this. That in particular, I know I plotted four points, but it a quick glance it looks like they're only three. And that's because it's taking this fourth point and stuck it way up there in the corner where I missed it. It's there. But it's so close to the edge of the graph that it's kind of hard to see. So I can fix that by executing the command axis, which tells it how far I want it to be. And this says, I want 1 axis to go from 0 to 6, and the other 0 to 20. We'll do that, and also to avoid boring you, we'll do more at the same time.

We'll put some labels on these things. I'm going to put that the title of the graph is going to be earnings, and that the x-axis will be labelled days, and the y-axis will be labelled dollars. So earnings dollars against days. OK, now let's see what happens when we do this. Well, we get the same ugly figure one as before, and now you can see figure two I've moved the axes so that my graph will show up in the middle rather than at the edges, and therefore easier to read. I put a title in the top, and I put labels on the axes. Every graph that I ask you to do this course, I want you to put a title on it and to label your axes so we know what we're reading. Again, nothing very deep here, this is really just syntax, just to give you an idea of the sorts of things you can do.

All right. now we get to something a little bit more interesting. Let's look at this code here. So far, what I've been passing to the plot function for the x- and y- values are lists. In fact, what Pylab uses is something it gets from NumPy which are not lists really, but what it calls arrays. Now, truth be told, most programming languages use array to mean something quite different. But, in NumPy an array is basically a matrix. On which we can do some interesting things. So for example, when I say x-axes equals array 1, 2, 3, 4. Array is a type, so array applied to the list is just like applying float to an int. If I apply float to an int, it turns it into a floating point number. If I apply array to a list, it turns it into an array. Once it's an array, as we'll see, we can do some very interesting things with it. Now, in addition to getting an array by coercing a the list, which is probably the most common way to get it, by the way. Because you build up a list in simulations of the sort we looked at, and then you might want to change it to an array to perform some operations on it. You can get an array directly with aRange. This is just like the range function we've been using all term, but whereas the range function gives you a list of ints, this gives you an array of ints. But the nice thing about an array is, I can perform operations on it like this. So if I say y-axis equals x-axis raised to the third power, that's something I can't do with a list. I get an error message if I try that with a list. What that will do is, will point-wise, take each element in the array and cube it.

So the nice thing about arrays is, you can use them to do the kinds of things you, if you ever took a linear algebra course, you learned about doing. You can multiply an array times an array, You can multiply an array times an integer. And sometimes that's a very convenient thing to do. It's not what this course is about, I don't want to emphasize it. I just want you to know it's there so if in some subsequent life you want to do more complicated manipulations, you'll know that that's possible. So let's run this and see what we get. So the first thing to look at, is we'll ignore the figure for the moment. And we've seen that when I printed test and I printed x-axis, they look the same, they are the same. And in fact, I can do this interesting thing now. Print test double equals x-axis. You might have thought that would return a single value, true. Instead it returns a list, where it's done a point-wise comparison of each element. So when we deal with arrays, they don't behave like lists.

And you can imagine that it might be very convenient to be able to do this. Answers the question, are all the elements the same? Or which ones are the same? So you can imagine doing some very clever things with these. And certainly, if you can convert problems to vectors of this sort, you can really perform what's almost magical. And then when we look at the figure, which should be tucked away somewhere here, what did I do with the figure? Did I make it go away? Well, I think I did one of those ugly things and made it go away again. Oh, no, there it is. All right. And sure enough here, we're plotting a cubic. All right. Nothing very important to observe about any of that, other than that arrays are really quite interesting and can be very valuable.

Finally, the thing I want to show you is that there are a lot of things we can do that are more interesting than what we've done. So now I'm going to use that random, which I brought in before, and show you that we can plot things other than simply curves. In this case, I'm going to plot a histogram. And what this histogram is going to do is, I'm going to throw a pair of dice a large number of times, and add up the sum, and see what I get. So, for die values equals 1 through 6, for i in range 10,000, a lot of dice. I'm just going to append random choice, we've seen this before, of the two dice, and their sum. And then I'm going to plot a histogram, Pylab dot hist instead of plot, and we'll get something quite different. A nice little histogram showing me the values I get, and we will come back to this later and talk about why this is called a normal distribution.