JOHN GUTTAG: So today, we're going to move on to a fairly different world than the world we've been living in. And this will be a world we'll be living in for quite a few lectures. But before I do that, I want to get back to just finish up something that Professor Grimson started.

You may recall he talked about family trees and raised the question, was it actually possible to represent all ancestral relationships as a tree? Well, as a counterexample, I'm sure some of you are familiar with *Oedipus Rex.* For those of you who are not, I'm happy give you a plot summary at the end of the lecture. It's a rather bizarre plot.

But it was captured in a wonderful song by Tom Lehrer. The short story is Oedipus ended up marrying his mother and having four children. And Tom Lehrer, if you've never heard of Tom Lehrer, you're missing one of the world's funniest songwriters. And he had a wonderful song called "Oedipus Rex," and I recommend this YouTube as a way to go and listen to it.

And you can gather from the quote what the story is about. I also recommend the play, by the way. It's really kind of appalling what goes on, but it's beautiful.

Back to the main topic, here's the relevant reading-- a small bit from later in the book and then chapter 14. You may notice that we're not actually going through the book in order. And the reason we're not doing that is because we're trying to get you information you need in time to do problem sets.

So the topic of today is really uncertainty and the fact that the world is really annoyingly hard to understand. This is a signpost related to 6.0002, but we won't go into too much detail about it. We'd rather things were certain. But in fact, they usually are not.

And this is a place where 6.0002 diverges from the typical introductory computer science course, which focuses on things that are functional-- given an input, you always get the same output. It's predictable. And we like to do that, because that's easier to teach. But in fact, for reasons we'll be talking about, it's not nearly as useful if you're trying to actually write

computations that help you understand the world. You have to face uncertainty head on.

An analogy is for many years people, believed in Newtonian mechanics-- I guess they still do in 8.01 maybe-- that every effect has a cause. An apple falls from the tree because of gravity, and you know where it's going to land. And the world can be understood causally. And people believed this really for quite a long time, most of history, until the early part of the 20th century, when the so-called Copenhagen doctrine was put forth.

The doctrine there from Bohr and Heisenberg, two very famous physicists, was one of what they called causal nondeterminism. And their assertion was that the world at its very most fundamental level behaves in a way that you cannot predict.

It's OK to make a statement that x is highly likely to occur, almost certain to occur, but for no case can you make a statement x will occur. Nothing has a probability of one. This was hard for us to imagine today, when we all know quantum mechanics. But at the turn of the century, this was a shocking statement.

And two other very well-known physicists, Albert Einstein and Schrodinger, basically said, no, this is wrong. Bohr, Heisenberg, you guys are idiots. It's just not true. They probably didn't call them idiots. And this is most exemplified by Einstein's famous quote that "God does not play dice," which is indicative of the fact that this was actually a discussion that permeated not just the world of physics, but society in general people really turned it into literally a religious issue, as did Einstein.

Well, so now we should ask the question, does it really matter? And to illustrate that, I need two coins. I forgot to bring any coins with me. Does anyone got a coin they can lend me?

AUDIENCE: I have some coins.

JOHN GUTTAG: All right. Now, this is where I see how much the students trust me. Do I get a penny? Do I get a silver dollar? So what do we got here? This is someone who's entrusting me with quarters, not so bad.

So we'll take these quarters, and we'll shake them up, and we'll put them down on the table. And now, we'll ask a question-- do we have two heads, two tails, or one head and one tail? So who thinks we have two heads? Who thinks we have two tails? Who thinks we have one of each?

Well, clearly, everyone except a few people-- for example, the Indians fan, who clearly believe in the counterfactual-- made the most probabilistic decision. But in fact, there is no nondeterminism here. I know the answer. And so in some sense, it doesn't matter whether it's deterministic, because in fact, it's not causally nondeterministic. The answer is quite clear, but you don't know the answer.

And so whether or not the world is inherently unpredictable, the fact that we never have complete knowledge of the world suggests that we might as well treat it as inherently unpredictable. And so this is called predictive nondeterminism. And this really is what's going to underline pretty much everything else we're going to be doing here.

No comments about that? I wouldn't do that to you. Thank you. I know you are wishing to get interest on the money, but you don't get any.

AUDIENCE: Was it heads or tails?

JOHN GUTTAG: What was that? So when we think about nondeterminism in computation, we use the word stochastic process. And that's any process that's ongoing in which the next state depends upon the previous states in some random element.

So typically up till now when we've written code, one line of code did depended only on what the previous lines of code did. There was no randomness. Here, we're going to have randomness.

And we can see the difference if we look at these two specifications of rolling a die. The first one, returns an int between 1 and 6, is what I'll call underdetermined. By that I mean you can't tell what it's going to return. Maybe it will return a different number each time you call it, but it's not required to. Maybe it will return three every time you call it.

The second specification requires randomness. It says, it returns are randomly chosen int. So it requires a stochastic implementation.

Let's look at how we implement a random process in Python. We start by importing the library random. This is not to say you can import any random library you want. It's to say you import the library called random. Let me get my pen out of here. And we'll use that a lot.

And then we're going to use the function in random called random.choice. It takes as an argument a sequence, in this case a list, and randomly chooses one member of the list. And it

chooses it uniformly. It's a uniform distribution.

And what that means is that it's equally probable that it will choose any number in that list each time you call it. We'll later look at distributions that are not uniform, not equally probable, where things are weighted. But here, it's quite simple, it's just uniform. And then we can test it using testRoll-- take some number of n and rolls the die that many times and creates a string telling us what we got.

So let's consider running this on, say, testRoll of five. And we'll ask the question, if we run it, how probable is it that it's going to return a string of five 1's? How do we do that?

Now, how many people here are either in 6.041 or would have taken 6.041? Raise your hand. Oh, good. So very few of you know probability. That helps.

So how do we think about that question? Well, probability, to me at least, is all about counting, especially discrete probability, which is what we're looking at here. What you do is you start by counting the number of events that have the property of interest and the number of possible events and divide one by the other.

So if we think about rolling a die five times, we can enumerate all of the possible outcomes of five rolls. So if we look at that, what are the outcomes? Well, I could get five 1's. I could get four 1's and a 2 or four 1's and 3, skip a few. The next one would be three 1's, a 2 and a 1, then a 2 and 2, and finally, at the end, all 6's.

So remember, we looked before at when we're looking at optimization problems about binary numbers. And we said we can look at all the possible choices of items in the knapsack by a vector of 0's and 1's. We said, how many possible choices are there?

Well, it depended on how many binary numbers you could get in that number of digits. Well, here we're doing the same thing, but instead of base 2, it's base 6. And so the number of possible outcomes of five rolls is quite high.

How many of those are five 1's? Only one of them, right? So in order to get the probability of a five 1's, I divide 1 by 6 to the fifth. Does that makes sense to everybody? So in fact, we see it's highly unlikely. The probability of a five 1's is quite small.

Now, suppose we were to ask about the probability of something else-- instead of five 1's, say 53421. It kind of looks more likely than that than five 1's in a row, but of course, it isn't, right?

Any specific combination is equally probable. And there are a lot of them.

So this is all the probability we're going to think about we could think about this way, as simply a matter of counting-- the number of possible events, the number of events that have the property of interest-- in this case being all 1's-- and then simple division. Given that framework, there were three basic facts about probability we're going to be using a lot of.

So one, probabilities always range from 0 to 1. How do we know that? Well, we've got a fraction, right? And the denominator is all possible events. The numerator is the subset of that that's of interest.

So it has to range from 0 to the denominator. And that tells us that the fraction has to range from 0 to 1. So 1 says it's always going to happen, 0 never.

So if the probability of an event occurring is p, what's the probability of it not occurring? This follows from the first bullet. It's simply going to be 1 minus p. This is a trick that we'll find we'll use a lot.

Because it's often the case when you want to compute the probability of something happening, it's easier to compute the probability of it not happening and subtract it from 1. And we'll see an example of that later today.

Now, here's the biggie. When events are independent of each other, the probability of all of the events occurring is equal to the product of the probabilities of each of the events occurring. So if the probability of A is 0.5 and the probability of B is 0.4, the probability of A and B is what? 0.5 times 0.4.

You guys can figure that out. I think that's 0.2. So you'd expect that, that it should be much smaller than either of the first two probabilities.

This is the most common rule, it's something we use all the time in probabilities, the so-called multiplicative law. We have to be careful about it, however, in that it only holds if the events are actually independent. Two events are independent if the outcome of one has no influence on the outcome of the other.

So when we roll the die, we assume that the first roll, the outcome, was independent of the-- or the second roll was independent of the first roll, independent of the fourth roll. When we looked at the two coins, we assume that heads and tails of each coin was independent of the

other coin. I didn't, for example, look at one coin and make sure that the other one was different.

The danger here is that people often compute probabilities assuming independence when you don't actually have independence. So let's look at an example. For those of you familiar with American football, the New England Patriots and the Denver Broncos are two prominent teams. And let's look at computing the probability of whether one of them will lose on a given Sunday.

So the Patriots have a winning percentage of 7 of 8-- they've won 7 of their 8 games so far-- and the Broncos 6 of 8. The probability of both winning next Sunday, assuming that this is indicative of how good they are, we can get with the multiplicative rule. So it's 7/8 times 6/8, or 42/64.

We could simplify that fraction, I suppose. Does that makes sense? So this is probably a pretty good estimate of both of them winning next Sunday.

So the probability of at least one of them losing is 1 minus that. So here's an example of why we often use the 1 minus rule, because we could compute the probability of both of them winning by simply multiplying. And we subtract that from 1.

However, what about Sunday, December 18? What's the probability? Well, as it happens, that day the Patriots are playing the Broncos.

So now suddenly, the outcomes are not independent. The probability of one of them losing is influenced by the probability of the other winning. So you would expect the probability of one of them losing is much closer to 1 than 22/64, which is about 1/3.

So in this case, it's easy. But as we'll see, as we get through the term, there are lots of cases where you have to work pretty hard to understand whether or not two events really are independent. And if you get it wrong, you get a totally bogus answer. 1/3 versus 1 is a pretty big difference. By the way, as it happens, the probability of the Broncos losing is about 1.

Let's go look at some code. And we'll go back to our dice, because it's much easier to simulate dice games than it is to simulate football games. So here it is. And we're going to talk a lot about simulations.

So here, rather than rolling the die, I've written a program to do it. We've already seen the

code for rolling a die. And so to run this simulation, typically what we're doing here is I'm giving you the goal-- for example, are we going to get five 1's-- the number of trials-- each trial, in this case, will be say of length 5-- so I'm going to roll the same die five times say 1,000 different times, and then just some text as to what I'm going to print.

Almost all the simulations we look at are going to start with lines that look a lot like that. We're going to initialize some variable. And then we're going to run some number of trials. So in this case, we're going to get from the length of the goal-- so if the goal is five 1's, then we're going to roll the dice five times; if it's 10 runs, we'll roll it 10 times. So this is essentially one trial, one attempt.

And then we'll check the result. And if it has the property we want-- in this case, it's equal to the goal-- then we're going to increment the total, which we initialized up here by 1. So we'll keep track with just the counting-- the number of trials that actually meet the goal. And then when we're done, what we're going to do is divide the number that met the goal by the number of trials-- exactly the counting argument we just looked at. And then we'll print the result.

Almost every simulation we look at is going to have this structure. There'll be an outer loop, which is the number of trials. And then inside-- maybe it'll have a loop, or maybe it won't-- will be a single trial. We'll sum up the results. And then we'll divide by the number of trials.

Let's run it. So a couple of things are going to go on here. If you look at the code as we've looked at it before, what you're seeing is I'm computing the estimated probability by the simulation.

And I'm comparing it to the actual probability, which we've already seen how to compute. So if you look at it, there are a couple of things to look at. The estimated probability is pretty close to the actual probability but not the same.

So let's go back to the PowerPoint. Here are the results. And there are at least two questions raised by this result. First of all, how did I know that this is what would get printed? Remember, this is random.

How did I know that the estimate-- well, there's nothing random about the actual probability. But how did I know that the estimated probability would be 0? And why did it print it twice? Because I messed up the PowerPoint.

Any rate, so how do I know what would get printed? Well a confession-- random.choice is not

actually random. In fact, nothing we can do in a computer is actually random. You can prove that it's impossible to build a computer that actually generates truly random numbers.

What they do instead is generate numbers that called pseudorandom. How do they do that? They have an algorithm that given one number generates the next number in a sequence. And they start that algorithm with a seed.

Now, typically, they get that seed by reading the clock of the computer. So most computers have a clock that, say, keeps track of the number of microseconds since January 1, 1978. I don't know if that's still true. That's what Unix used to do.

So the notion is, you start your program, there's no way of knowing how many microseconds have elapsed. And so you're getting a random number to start the process. Since you don't know where it starts, you don't know what the second number is, you don't know what the third number is, you don't know what the fourth number is. And so it's predictably nondeterministic, because you don't know what the seed is going to be.

Now, you can imagine that this makes programs really hard to debug. Every time you run it, something different could happen. Now, we'll see often you want them to be unpredictable. But for now, we want them to be predictable, makes it easier prepare PowerPoint.

So what you have is a command. You can call random.seed and give it a value and say, I don't want you to just choose some random seed, I want you to use 0 as the seed. For the same seed, you always get the same sequence of random values. And so what I've done is I set the seed to be, I think, 0 in this case, not because there's anything magic about 0, it's just sort of habit. But it made it predictable.

As you write programs with randomness in and when you're debugging it, you will almost surely want to start by setting random.seed to a value so you get the same answer. But make sure you debug it with more than one value of this, so you didn't just get lucky with your seed. So that's how I knew what would get printed.

The next question is, why did the simulation give me the wrong answer? The actual probability is three 0's and 1286. But it's estimated a probability of 0. Why is it wrong?

Well, let's think about this. I ran 1,000 trials. What does it mean to say the probability is zero? It means that I tried it 1,000 times and didn't ever get a sequence of five 1's. So the numerator

of the division at the bottom was 0. Hence, the answer is 0.

Is this surprising? Well, no. Because if that's the actual probability of getting five 1's, it's not very shocking that in 1,000 trials it never happened. It's not a surprising result. And so we have to be careful when we run these things to understand the difference between what's in this case an actual probability and what statisticians call a sample probability.

So what we got with the sample was 0. So what's the obvious thing to do? If you're doing a simulation of an event and the event is pretty rare, you want to try it on a very large number of trials.

So let's go back to our code. And we'll change it to instead of 1,000, 1,000,000. You can see up here, by the way, where I set the seed. And now, let's run it.

We did a lot better. If we look at here our estimated probability, it's three 0's 128, still not quite the actual probability but darn close. And maybe if I had done 10 million, it would have been even closer.

So if you're writing a simulation to compute the probability of an event and the event is moderately rare, then you better run a lot of trials before you believe your estimated probability. In a week or so, we'll actually look at that more mathematically and say, what is a lot, how do we know what is enough.

What are the morals here? Moral one, I've just told you-- takes a lot of trials to get a good estimate of the frequency of a rare event. Moral two, we should always, if we're getting an estimated probability, know that, and probably say that, and not confuse it with the actual probability.

The third moral here is, it was kind of stupid to do a simulation. Since it was a very simple closed-form answer that we could compute that would really tell us what the actual probability is, why even bother with the simulation? Well, we're going to see why now, because simulations can be very useful.

Let's look at another problem. This is the famous birthday problem. Some of you have seen it. What's the probability of at least two people in a group having the same birthday?

There's a URL at the bottom. That's pointing to a Google form. I'd like please all of you who have a computing device to go to it and fill out your birthday. It's anonymous, so we won't

know how old you are, don't worry. Actually, it's only the date. It's not the year.

So suppose there were 367 people in the group, roughly the number of people who took the 6.0001 600 midterm. If they are 367 people, what's the probability of at least two of them sharing a birthday? One, by something called the pigeonhole principle. You got some number of holes. And if you have more pigeons than holes, two pigeons have to share a whole.

What about smaller numbers? Well, if we make a simplifying assumption that each birthdate is equally likely, then there's actually a nice closed-form solution for it. Again, this is a question where it's easier to compute the opposite of what you're trying to do and subtract it from 1. And so this fraction is giving the probability of two people not sharing a birthday.

The proof that this is right, it's a little bit elaborate. But you can trust me, it's accurate. But it's a formula, and it's not that complicated a formula. So numbers like 366 factorial are big.

So let's approximate a solution. We'll right a simulation and see if we get the same answer that that formula gave us. So here's the code for that-- two arguments-- the number of people in the group and the number that we asking do they have the same birthday. So since I'm assuming for now that every birthday is equally likely, the possible dates range from 1 to 366, because some years have a February 29.

I'll keep track of the number of people born in each date by starting with none. And then for p in the range of number of people, I'll make a random choice of the possible dates and increment that element of the list by 1. And then at the end, we can say, look at the maximum number of birthdays and see if it's greater than or equal to the number of same. So that tells us that.

And then we can actually look at the birthday problem-- number of people, the number of same, and, as usual, the number of trials. So the number of hits is 0 for t in range number of trials. If sameDate is true, then we'll increment the number of hits by 1 and then as usual divide by the number of trials.

And we'll try it for 10, 20, 40, and 100 people. And then just, we'll print the estimated probability and the actual probability computed using that formula I showed you. I have not shown you, but I've imported a library called math, because it is a factorial implementation. It's way faster than the recursive one that we've seen before. Let's run it.

And we'll see what we get. So for 10, the estimated probability is 0.11 now. So you can see,

the estimates are really pretty good. Once again, we have this business that for 100, we're estimating 1, when the real answer is point many, many 9's. But again, this is sample probability.

It just means in the number of trials we did, every 1 for 100 people, there was a shared birthday. This is a number that usually surprises people, as to why with 100 people the probability is so high. But we could work out the formula and see it. And as you can see, the estimates are pretty good from my simulation.

Now, we're going to see why we did a simulation in the first place. Suppose we want the probability of three people sharing a birthday instead of two. It's pretty easy to see how we changed the simulation. I even made a parameter. I just changed the number 2 to number 3.

The math, on the other hand, is ugly. Why is the math so much uglier for 3 than for 2? Because for 2, the complementary problem-- the number we're subtracting from 1-- is simply the question of, are all birthdays different? So did two people share a birthday is 1 minus or all does everybody have a different birthday.

On the other hand, for 3 people, the complementary problem is a complicated disjunct-- a bunch of ors-- either all birthdays are distinct, or two people share a birthday and the rest are distinct, or there are two groups of two people sharing a birthday and everything is distinct. So you can see here, there's a lot of possibilities. And so it's 1 minus now a very complicated formula.

And in fact, if you try and look how to do this, most people will tell you don't bother. Here's kind of a good approximation. But the math gets very hairy.

In contrast, changing the simulation is dead easy. We can do that. Whoops. So if we come over here for the code, all I have to do is change this to 2 or 3. And I'm going to leave in this code, which is the wrong code, computing the actual probability now for 2 people sharing rather than 3, because I want to make it easy for you to see the difference between what happens when we look at 3 shared rather than 2 shared.

And I get invalid syntax. That's not good. That's what happens when I type in real time. Why do I have invalid syntax?

AUDIENCE: Line 56.

JOHN GUTTAG: Pardon.

AUDIENCE: Line 56.

JOHN GUTTAG: One person, Anna.

AUDIENCE: Line 56, there's a comma.

JOHN GUTTAG: Oh. That's not a good line. So now, we see that if we get, say, to n equals 100, for 2, you'll remember, it was 0.99. But for 3, it's only 0.63. So we see going from two sharing to three sharing gets us a radically different answer, not surprisingly.

But we also-- and the real thing I wanted you to see-- is how easy it was to answer this question with the simulation. And that's a primary reason we use simulations to get probabilistic questions rather than sitting down and the pencil and paper and doing fancy probability calculations, because it's often way easier to do a simulation.

We can see that in spades if we look at the next question. Let's think about this assumption that all birthdays are equally likely. Well, as you can see, this is a chart of how common birthdates are in the US, a heat map. And you'll see, for example, that February 29 is quite an uncommon birthday. So we should probably treat that differently.

Somewhat surprisingly, you'll see that July 4 is a very uncommon birthday as well. It's easy to understand why February 29. The only thing I can figure out for July 4 is obstetricians don't like working on holidays. And so they induce labor sometime around the 2nd or the 3rd, so they don't have to come to work on the 4th or the 5th.

Sounds a horrible thought. But I can't think of any other explanation for this anomaly. You'll probably, if you look at it, see Christmas day is not so common either.

So now, the question, which we can answer, since you've all fill out this form, is how exceptional are MIT students? We like to think that you're different in every respect. So are your birthdays distributed differently than other dates? Have we got that data? So now we'll go look at that.

We should have a heat map for you guys. This one?

AUDIENCE: Yep. I removed all the February 31. Thank you for those submissions.

[LAUGHTER]

JOHN GUTTAG: So here it is. And we can see that, well, they don't seem to be banded quite as much in the summer months, probably says more about your parents than it does about you. But you can see that, indeed, we do have-- wow, we have a day where there are five birthdays, that look like? Or no?

AUDIENCE: February 12.

JOHN GUTTAG: Wow. You want to raise your hand if you're born on February 12?

[LAUGHTER]

So you are exceptional in that you lie about when you're born. But if you hadn't lied, I think we would have still seen the probabilities would hold. How many people were there, do we know?

AUDIENCE: 146 with 112 unique birthdays.

JOHN GUTTAG: 146 people, 112 unique birthdays. So indeed, the probability does work.

So we know you're exceptional in a funny way. Well, you can imagine how hard it would be to adjust the analytic model to account for a weird distribution of birthdates. But again, adjusting the simulation model is easy.

I could have gone back to that heat map I showed you of birthdays in the US and gotten a separate probability for each day, but I was too lazy. And instead, what I observed was that we had a few days, like February 29, highly unlikely, and this band in the middle of people who were conceived in the late fall and early winter.

So what I did is I duplicated some dates. So the 58th day of the year, February 29, occurs only once. The dates before that, I said, let's pretend they occur four times. What only matters here is not how often they occur but the relative frequency. And then the dates after that occur four times except for the dates in that band, which is going to have occur yet more often.

So now-- and don't worry about the exact details here-- but what I'm doing is simply adjusting the simulation to change the probability of each date getting chosen by same date. And then I can run the simulation model. And, again, with a very small change to code, I've modeled something that's mathematically enormously complex. I have no idea how to actually do this

probability mathematically. But the code is, as you can see, quite straightforward.

So let's go to that here. So what I'm going to do is comment this one out and uncomment this more complicated set of dates and see what we get. And again, it changes quite dramatically.

You might remember, before it was around I think 0.6-something for 100, and now, it's 0.75. So getting away from the notion that birthdays are uniformly distributed to saying some birthdays are more common than others, again, dramatically changes the answer. And we can easily look at that.

So that gets us to the big topic of simulation models. It's a program that describes a computation that provides information about the possible behaviors of a system. I say possible behaviors, because I'm particularly interested in stochastic systems.

They're descriptive not prescriptive in the sense that they describe the possible outcomes. They don't tell you how to achieve possible outcomes. This is different from what we've looked at earlier in the course, where we looked at optimization models. So an optimization model is prescriptive. It tells you how to achieve an effect, how to get the most value out of your knapsack, how to find the shortest path from A to B in a graph.

In contrast, a simulation model says, if I do this, here's what happens. It doesn't tell you how to make something happened. So it's very different, and it's why we need both, why we need optimization models and we need simulation models.

We have to remember that a simulation model is only an approximation to reality. I put in an approximation to the distribution of birthdates, but it wasn't quite right. And as the very famous statistician George Box said, "all models are wrong, but some are actually very useful."

In the next lecture, we'll look at a useful class of models. When do we use simulations? Typically, as we've just shown, to model systems that are mathematically intractable, like the birthday problem we just looked at. In other situations, to extract intermediate results--something happens along the way to the answer.

And as I hope you've seen that simulations are used because we can play what if games by successively refining it. We started with a simple simulation that assumed that we only asked the question of, do two people share a birthday. We showed how we could change it to ask do three people share a birthday.

We then saw that we could change it to assume a different distribution of birthdates in the group. And so we can start with something simple. And we get it ever more complexed to answer questions what if.

We're going to start in the next lecture by producing a simulation of a random walk. And with that, I'll stop. And see you guys soon.