The following content is provided under a Creative Commons license. Your support will help MIT OpenCourseWare continue to offer high quality educational resources for free. To make a donation or view additional materials from hundreds of MIT courses, visit MIT OpenCourseWare at ocw.mit.edu.

PROFESSOR: So we'll start off by going over the Quiz Two practice questions. So, (1.1). In Python a subclass could override methods of its superclass. What say you all? True? All right. There are some repeat people from last night, so they know these already.

Standard deviation, coefficient of variation are different names for the same thing. True or false?

AUDIENCE: False.

PROFESSOR: False. All right. (1.3.)

AUDIENCE: Can you go over what the coefficient of variation is?

PROFESSOR: So, coefficient of variation. It is-- I have to write big for the camera. Is that right? People who haven't asked a question. It's the other way around, right? Coefficient of variation is the standard deviation divided by the mean. What does this tell you? Like, what sort of information does the coefficient of variation tell you? Anyone?

So, it's another way of telling if your distribution is very widely spread. So, what it's getting at is-- it's trying to help you compare your standard deviation against your mean. Because, let's say that I have a distribution and its mean is all the way out at 1.

Let's pick a very large number. So it's got a really big mean, but say its standard deviation is-- I don't know-- 100. Would you say that that's widely spread out? It's pretty tight. If you're taking-- if this were like, I don't know, an observatory taking measurements of deep space and you've got a fluctuation of 100 for each standard deviation, you'd probably say that was pretty good. And you wouldn't be too concerned about it, because you've got a planet out there that's billions of light years away or something.

So on the other hand, let's say that I have another measurement, and its mean is somewhere around, say, 50. And its standard deviation is 100. So it's a lot fatter. You would say that this has a much wider spread, right? So it's a way of relating the standard deviation to the mean.

And, rule of thumb is if this is less than or equal to 1, we're going to say that it's not too widespread, there's not a lot of variation. That's a standard that Professor Guttag has. Depending on your purposes maybe that's too large or whatever, so but that's what coefficient of variation is. Everyone clear on it?

AUDIENCE: What's the difference between variance and standard deviation?

PROFESSOR: The difference between variance and standard deviation? Standard deviation is the square root of variance. All right.

Unit testing is useful for debugging. What does the audience say?

AUDIENCE: True.

AUDIENCE: True.

PROFESSOR: OK. You know what unit testing is, can you define it?

AUDIENCE: Well, unit testing is when you use other code to test your code.

PROFESSOR: Right. So say you've broken your code up into nice little functions like a good programmer. And you want to make sure that those functions are giving you the right values. What you're doing with unit testing is, you're writing a bunch of other code. Or a little bit of code. Sometimes a lot, it depends on how thorough you want to be in your testing.

And what you have is known input and known output. And you're going to feed your input to the functions you're testing, and you're going to make sure that they give you the output that you're expecting. And if they don't give you the output that you're expecting, that indicates that there's probably a bug. Which also kind of supports the idea that unit testing is good for debugging.

And if you are very conscientious about your code and doing unit testing, you're going to run your unit tests every time you either run your code or make a change. Because you might have inadvertently introduced a bug into the code that you're modifying. So it's a way of verifying, it's a way of detecting bugs, and also verifying that any changes you make in the process of writing more functions for your program or modifying functionality in your program don't introduce other bugs.

So in Python, functions cannot be used as actual parameters. False. Why?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Well, you could just rephrase the question. But a better answer is that functions are first class citizens. So they are objects in and of themselves, and so you can pass them around to a function and functions can make use of them. So we might see an example of that later on if you want.

All right. 1.5. Increasing the size of a hash table typically increases the amount of time needed to locate a variable or value in the table. False. Why?

AUDIENCE: It is Constant type. PROFESSOR: Right. The hashing function is generally constant. OK.

So, what does this code print?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Six?

AUDIENCE: 11.

PROFESSOR: It's 11.

AUDIENCE: [INAUDIBLE]

PROFESSOR: So what is the string x actually?

AUDIENCE: Binary?

PROFESSOR: It's the binary representation of an integer. And all this code is doing is, it's just converting it to a decimal. And because I came prepared tonight-- sorta kind of-- I have the code here. And we can verify it. Any burning questions on that problem? Moving on.

What is the expected value of g1? Anyone?

Well, look at what the code's doing. Did someone--

AUDIENCE: What's "gauze," that function? We haven't used that one.

PROFESSOR: Gauss is--

AUDIENCE: Oh, Gauss.

PROFESSOR: --a method on the random object that draws a random value from a Gaussian distribution that has a mean at-- well, in this case, mean-- and a standard deviation of s-t-d-dev, whatever you put in there.

So maybe for this example, if I wanted to draw a random value from this distribution, which I said had a mean of 50 and a standard deviation of 100, I would say random.Gauss 50, 100.

So, in answering the question (3.1), what do people think? What about (3.2)? 100,000. So they're both the same. But they both have a different standard deviation. So what this is getting at is, it's getting you to look at this.

So this code is straight from the question. And all I'm going to do is, I'm going to plot a histogram of what this function produces. And also I'm going to run it a number of times and verify that I am getting 100,000 in general. Python. There we go.

So it's not exactly 100,000, but it's close enough. I mean, we're not going to be too upset. So this picture here shows the actual values that were drawn when I repeated the experiment a number of times.

So I don't know if you can see on the screen but there's this big, really thin, vertical line here. Those are the results from the Gaussian with a 0 standard deviation. So everything was 100,000 exactly. And then these red blocks here, this is with the standard deviation of 20. Any questions on this? Yeah?

AUDIENCE: What's the difference between a normal distribution and a Gaussian distribution?

PROFESSOR: The name. The question was, what's the difference between a normal distribution and a Gaussian distribution. And it's just, it's a different name.

AUDIENCE: [INAUDIBLE]

PROFESSOR: How I came across it?

AUDIENCE: Yeah. How are we supposed to see that in this problem, [UNINTELLIGIBLE]?

PROFESSOR: Well, if you look at the plots, which I apparently killed-- so let me pull them up again-- All right. So the thing that you want to know about Gaussian distributions is that, at the mean, is the value that's the most probable. So its peak is at the mean. And so if you're taking a 1000 numbers from this distribution, then if you know which number you're expecting to see with the highest probability-- I'm not sure if this sentence is grammatical any more--

AUDIENCE: Yeah. So really standard deviation doesn't really have anything to do with it.

PROFESSOR: What it's going to affect is, it's going to tell you that-- like, if you have a standard deviation of 10 versus 20, it's just going to tell you what the spread of the curve is. But it's not going to change the expected value. So, why don't I actually demonstrate that? Because I'm a fan of visuals and not necessarily good at English tonight.

All right. So all I did in the code was, I just changed the standard deviation of the first distribution to be 10 versus 20. And what you can see from the plots are that the value with the highest probability is still centered at 100,000. But the first plot, as opposed to the first time we ran this with a 0 standard deviation, is a little bit more spread out. And the plot with 20, for the standard deviation, is even more spread out, than it. But they both have the highest probability at 100,000 or centered at 100,000.

AUDIENCE: So it's like, if you have a mean for g1 of 100, or g2 of 200, then you would expect 200,000 for g2?

PROFESSOR: Yeah. So if instead you had a mean of, say, 200 for the second distribution, versus 100, you'd expect them in two completely separate places. And you'd get an expected value that's 100,000 versus 200,000. So here close to 100,000, close to 200,000 and then see what my distribution is. All right. We good on this? So moving on.

Question (4). What is the probability of the final value of num6 being 0?

AUDIENCE: 9/10 to the 10.

PROFESSOR: 9 over 10, the whole thing to the 10th power, right? OK. So does anyone not see why that's the case? So I'm going to explain that anyway.

So what this code is doing is, for 10 times its drawing an integer from 0 to 9. A random integer. It's a uniform distribution. And it's counting the number of times that the integer drawn is 6. And it's incrementing a counter. So the question can be rephrased as, what's the probability that no 6's were drawn 10 times out of a pool of 10 integers?

And the way you can figure it out is on one pool where, say I had a 10 sided dice. On one roll, what's the probability of not getting a 6? It's going to be 9 over 10, right? And if you do that 10 times. 9 over 10 times 9 over 10 times 9 over 10, that gives you 9 over 10 to the 10th power. And because this is a code class, we have a demonstration.

So here's the answer that we hand-jammed or that we just discussed. And now here is a function count of 6. And it's the code that you see in problem (4). And all I'm going to do is, I'm going to run a number of trials. And I'm going to count the number of trials that come up 0, and increment a count numzero's. And then I'm going to say, this is what I think the probability of getting a 0 is. And then we'll compare the two numbers.

OK so that's the probability that we count 9/10 to the 10th power. And then this is the estimated probability that we got from running this function 100,000 times. So we good on this problem?

AUDIENCE: Instead of solving it the way that you guys solved it, can you solve it by saying that the problem's also-- that it's 1 minus the probability that 6 will always be picked?

PROFESSOR: That you will see at least one 6? 1 minus the probability that you'll see at least one 6?

AUDIENCE: No, never mind.

PROFESSOR: You can solve it in different ways. It's just this way is-- I think, probably, this way of thinking about it is probably more intuitive.

AUDIENCE: When would you want to solve this kind of problem with [INAUDIBLE], instead of when it says it's not something, to use 1 minus the probability? That it is [UNINTELLIGIBLE]

PROFESSOR: Well, my answer would be when it makes sense to you. Some problems are easier viewed in the negative light, which is what you're saying as 1 minus whatever. And some problems are better solved in a straightforward fashion. So it's whatever tool gets the job done. I don't have a general rule of thumb for you. Maybe there's a course 18 person who does.

AUDIENCE: So this problem, would we possibly be asked, what is the probability that it'll end up being 1? Or does it necessarily matter which of those 10 came up 6? And how would you show that, if we were going to be asked that question?

PROFESSOR: Well. One, I haven't seen the quiz. So I can't give you a definite. But you could be asked, instead of using 6, maybe say, 1. Is that going to really change anything?

AUDIENCE: No. I guess I meant that one of those 10 outcomes ends up being a 6.

PROFESSOR: Right.

AUDIENCE: I can do it if, they say the fifth trial is a 6. But just 1 of those 10 trials being a 6, I don't know how to write properly?

PROFESSOR: So what's the probability that you get 1 of the trials as exactly 6? Or exactly 1 of the trials comes up as 6?

AUDIENCE: Yeah.

PROFESSOR: Well, one way of thinking of it is-- so we calculated 9/10. That's the probability that 6 doesn't come up, right? So what's the probability that 6 will come up? 1/10. So instead of doing 9/10 to the 10th power, it would be 9/10 to the 9th power times 1/10, right?

AUDIENCE: OK, that'll account for. Thanks

PROFESSOR: And because I'm never very comfortable with this stuff, why don't we do a simulation?

So this is what we're saying. Exactly one 6 comes up. And I'm going to set this to 1.

AUDIENCE: Isn't that saying that it's less likely that one 6 comes up than zero 6s come up? Is it? That's true?

PROFESSOR: Yeah. Because you're looking for exactly-- you're looking for a sequence of 10 events where exactly 1 of those of events is a 6. Which seems more unlikely than, say, getting everything that's not a 6. So I want to verify this.

AUDIENCE: Why is it 9/10 to the 9th [INAUDIBLE]

PROFESSOR: Because we're asking, if I roll this 10-sided dice 10 times, what's the probability that exactly one of those rolls is going to come up with 6? And so what we're asking is, what is the probability that I roll the dice 9 times and it doesn't come up with 6? And then what's the probability that on the remaining roll, that it comes up as 6? Which is 1/10. So let's see if I'm in the ballpark. Did I make a mistake?

AUDIENCE: [INAUDIBLE] where if there's ten different ways-- so it would be like--

PROFESSOR: Oh, yeah. So yeah, you're right. That's why I don't like that stuff. So yeah, it's a little bit more complicated than that.

AUDIENCE: You only counted one case of getting--

PROFESSOR: That's right. So in this, I'm only counting one case. There's multiple ways that exactly one dice could come up 6. So the answer is more complicated.

AUDIENCE: So would you just multiply by 10? Because there's 10 different places?

PROFESSOR: Yeah. So actually, that's what we got, so-- we'll just do this. Well, of course that's going to come up. But the simulation gives us the same answer as what we've just come up with discussing. OK, problem solved. Did that work?

AUDIENCE: Yeah.

PROFESSOR: Probability can be tricky.

AUDIENCE: So essentially, you could just take the probability that 9 of the rolls aren't 6, and leave it at that? Because you multiply it by 10 [INAUDIBLE].

PROFESSOR: Yeah. You can also do it that way. So again, there's multiple ways to think about the problem. So she was saying, instead of multiplying by 10, we just say that-- we take the probability that we have 9 rolls that aren't 6. And that works too. That's what she just pointed out. So these parts cancel each other out. It's basically-- it is the same thing as this. Yay. All right. Are we good on this question?

All right. Problem (5). It was matching plots. So you have three functions here. You have a polynomial, or two polynomials. One's a cubic, one's a quintic. And one's an exponential. There are a couple ways that you could go about solving this one. One is to plug values in and kind of match them up with the plots.

Another is to look at the axis of the plots. So this exponential function here, if we were to have it on two linear axes, the x-axis and the y-axis for both linear, you'd see the expected curve upward sharply. But if you notice, a couple of the plots are in a semi-log y-axis.

So if you put an exponential with a linear x-axis and a logarithmic y-axis, what would you expect the plot to look like? It would be a straight line, right. So what that tells us is that if we look at the bottom plot, we have something with a straight line.

So, bottom plot is the third function. So, exponential. And then for the remaining two functions, there are a couple methods that you would use, or you could use. One is to plug values in and see where the points wind up and compare them with the plots. Another way to think about it is that one of the plots is on a semi-log and another plot is on just both linear axes. And if you plot a polynomial that's not exponential on a logarithmic y-axis, what would you expect the curve to look like? It should look somewhat similar to a logarithm. So with that in mind, you could then start plugging points in. And what you would come up with is that the second function is this top plot here. And the middle plot here is the first function. Is there anyone who's lost on that? Uh-oh.

AUDIENCE: Wait, why? So you have two quadratic functions, right? One and two are quadratic?

PROFESSOR: They're polynomials. They're not exponential.

AUDIENCE: OK. And then the third one's exponential. And the reason why the third one's linear is because when you plot an exponential--

PROFESSOR: If you plot an exponential with a linear x-axis and a logarithmic y-axis, it should be a straight line. Right? Or a straightish line.

AUDIENCE: Why?

PROFESSOR: It's the way the math works out.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Any other questions? All right, we'll move onto the next one. What does the following code print?

AUDIENCE: 16. And then square root of [INAUDIBLE]

PROFESSOR: Close. So I don't actually have the answer. Well, not written down. But I have the code, so I'm just going to run the code. So you're close.

Shall we spend some time stepping through this code, understanding it? All right. So, what question (6) does is, it creates a class hierarchy. And at the top of it, it has the shape class. Which has just two methods, double underbar EQ and double underbar GE. That was just defined as equals operator and the greater than or equal operator for shapes. In terms of their area.

So what this is saying is that two shapes are equal if their areas are equal, and a shape S1 is going to be larger than another shape if its area is larger than the other shape. So if I had-- well let's go down and we'll get to that. Square, subclass of shape. It is what it says it is. It's a square. And it has a side. And its area is what you'd expect it to be, the side squared. And circle is just defined-- it just has a radius, and its area is what you would expect it to be.

There is a function here, f. And it takes a parameter, l. And l -- if you look at the code here and you kind of walk through it -- l would have to be either a list or a tuple of elements. Doesn't have to necessarily be shapes. But this first line is just going to return none if the list is empty. And then these next lines are going to, one it's going to set the first element equal to x, and then iterate through each element in l. And then if the element it's looking at is larger than x it's going to set x to the element.

So what this function is doing is finding the largest element in the list. Everyone follow that? So, this should be pretty straightforward. As should this. This part my trip some people up. So, this obviously creates an empty list. What this does is, it creates a dictionary of class names. And what it's saying is that for a key of 0, the class is circle. And for a key of one the class is square. Everyone see that?

This loop here iterates for 10 times. And it appends 10 elements. And then each of those elements are defined by this shapes dictionary. On each iteration, it's going to take the modulus of i, the index, or the number. And if it's a 0, it's going to construct whatever class the dictionary of shapes returns. So if it's an even number, it's going to be circle. And if it's an odd number is going to be a square. Everyone follow that? Does no-one follow that? OK. Or does anyone not follow that? Got my logic messed up.

All right. So now what we have here is a list of shapes. And then all this does is it's going to print which shape. Is it going to be the fourth or the fifth shape? Fifth, right? Because we've started indexing it at 0.

So this number is obviously even. So this should be a circle shape, or a circle class. And what should its radius be?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Right. And then this line here is just going call the function f, which is going to find the larger shape. Which is determined by its area. And in this particular instance, it's going to be a circle with radius eight. Once you figure out what f does, this is actually not such a hard problem. So the key to understanding this one is figuring out what f is actually doing. And you'll probably see a problem or two on the exam that's written like this, where the function has been obfuscated. So it doesn't have nice little variable names. If we were writing this for real, it would be something like that.

AUDIENCE: Wait, what was the radius on L? It was--

PROFESSOR: Radius on L?

AUDIENCE: Or not radius. Like, L4 was circled. And was that all it was, it was just a circle?

PROFESSOR: With radius 4.

AUDIENCE: With radius 4? OK, that's what I thought.

PROFESSOR: All right.

AUDIENCE: How many of those special underbar functions have we seen? Have we seen anything besides edit string and [INAUDIBLE]?

PROFESSOR: So, how many of the double underbar functions have you seen? That's a good question. You've seen a lot, right? The ones that I would be familiar with definitely INIT definitely EQ, GE and definitely STR. You might want to be familiar with this, LE. So what is LE? What do you think that does? Less than or equal to. Less than. So it's this the opposite of GE, right? I think there's also an LT, so less than. And GT. It's fairly logical. And historically you don't need to have these memorized. I've never seen a question where you've actually had to produce code that uses double underbar on the quiz.

You just have to understand what it's doing if you can see it. Like, in a code reading exercise. And the conventions are fairly straightforward to understand. Can I move on?

AUDIENCE: Can you scroll down a little bit? Where it says print L4, or I guess print FL. Does it automatically just print the string function?

PROFESSOR: Yes. So you're asking why I don't have this.

AUDIENCE: Right.

PROFESSOR: So in this flavor of Python, and if you don't have an explicit conversion to string, Python will look for the underbar STR function if it exists. If it doesn't exist then you get that weird object at blah, blah, blah.

AUDIENCE: Right.

PROFESSOR: Any other questions?

AUDIENCE: What list are you reading up on top?

PROFESSOR: Are you confused because it's the same name?

AUDIENCE: Maybe. I don't know. So the third thing that [INAUDIBLE]. I understand the second thing, circle with radius 4. But how do you get that circle of radius 8?

PROFESSOR: Well, the way we get that is, f of l-- f is the function, right?

AUDIENCE: Right, and it produces the largest element of the list. And the list, L, contains 0 to 9?

PROFESSOR: Yes. So we could actually print out what L contains.

AUDIENCE: Why isn't it square? Isn't 9 [INAUDIBLE]?

PROFESSOR: Why don't we do it?

All right. So all it's going to do is, it's going to print out all the shapes in the list and their areas. I've got stuff at the top. So this is the first shape. And then, as we move down the list, we have square with side 1, circle with radius 2, 3, et cetera, et cetera, et cetera. And then, this is the area of the circle with radius 8 versus the area of a square with side 9. So, and it depends on-- the reason why it works this way is that this is greater than or equal to operator is defined in terms of the area method of the shapes. Does that--

AUDIENCE: So wait. What is this doing then? L is a list of areas?

PROFESSOR: No, L is a list of shapes. It's a list of subclasses of shapes.

AUDIENCE: So you're feeding f this list of shapes. Oh, and to compare if S and L [? into that list ?] is greater. And for equal to x, do you have to use the greater than or equal to--

PROFESSOR: Operator, right.

AUDIENCE: --method, the class of shapes which compares areas?

PROFESSOR: Mm-hm. Now, I might regret this. I think this'll work. Yep. It works.

So, you see this weirdness I have here? It's showing pretty explicitly that I'm calling this a GE operator. It's just so-- this is really just shorthand for that. Syntactic sugar.

It's easier to look at. Easier to understand. And to even drive that point further home-- whenever that GE, whenever that comparison is done, it's going to print out I'm here. So it is actually getting called and not just blowing smoke. Have we beaten this problem to death?

Last problem. So, I'm actually not a huge fan of this problem. But the way I think of it is, it's the number guessing game, right? So I say to you, I'm thinking of a number between 0 and 100. Or in this case, maxVal. And your job is to guess, as quickly as possible, what number I'm thinking of. The only thing I can tell you is that you give me a guess and I'll tell you if it's higher or lower, or if it is the answer. So if I'm thinking of a number between 0 and 100, what's the strategy?

AUDIENCE: [UNINTELLIGIBLE]

PROFESSOR: Binary search.

AUDIENCE: Binary search, yeah.

PROFESSOR: Binary search, right? So you're going to start in the middle. You're going to start at 50, and you're going to say-- and I'm going to say, well, no, the number I'm thinking of is lower. So now you're going to guess between 0 and 50, 25. And I'm going to say it's higher. Now you're going to guess between 25 and 50. I don't know, I didn't actually think of a number. But you get the idea, right? So this problem is asking you to implement a binary search. And comp.guess here is functioning as me in this problem. It knows a number and you write a program that's going to guess it, which is find.number.

So, there are multiple ways to do this problem. Here is my solution. This is just an implementation of comp guess. At the beginning, set a maximum value of 100,000. And I'm going to guess the magic number, or I'm going to choose a random number from between-- yeah?

AUDIENCE: When you say binary search, is that the same thing as bisection search?

PROFESSOR: It's the same idea. But the way that I differentiate binary search from bisection search is that you do a binary search on a finite list of elements, of sorted elements. So I might have a list of names in alphabetical order, or a list of numbers in ascending order. And I would do a binary search on that list.

With bisection search, it's not necessarily a finite list of numbers or elements that I'm looking at. It's, I know that the answer exists within a lower and an upper bound, and I'm going to divide that search space in half successively or iteratively to find that actual value. But it's not necessarily a finite list of elements between those bounds. That's good?

AUDIENCE: Yes.

PROFESSOR: So this is one implementation. You can do it any number of ways. This numSteps thing, you wouldn't need to have. It's just for me to illustrate some stuff. So I'm going to start at 0, and I'm going to start at maxVal. And then I'm going to keep iterating. While I don't have, or while I haven't reached the answer, basically.

So, I'm going to make my guess in the middle. And then if comp.guess tells me that it's higher, or that my guess is too low, basically, then I'm going to set my upper-- sorry, if my guess is too high, then I'm going to set my upper bound to my guess. And then I'm going to search in the middle from there. And then if it tells me that my guess is too high, then I'm going to search in the upper region. Otherwise, that means I've made the correct guess and I'm done. So, very simple.

The key part of this problem that tells you that you should use binary search, though, is that it stipulates you need to have a log maxVal algorithm. And binary search is a log N algorithm. I mean, you could do find number in a linear search, but it would be fairly inefficient. All right?

AUDIENCE: [INAUDIBLE] just guess one number, then go--

PROFESSOR: Yeah, start at 0--

AUDIENCE: [UNINTELLIGIBLE] guess 1, you guess 2, then guess 3.

PROFESSOR: Yep. So you just keep incrementing by one.

AUDIENCE: If it didn't specify, oh, it should run log [UNINTELLIGIBLE], would it be OK if we had used the word [UNINTELLIGIBLE]?

PROFESSOR: If it didn't specify the complexity, then yeah, you could use whatever search you deemed appropriate. But because it said it needs to be log N --

AUDIENCE: Yeah, yeah. OK.

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that?

AUDIENCE: Are you going to [UNINTELLIGIBLE]?

PROFESSOR: Not in this particular case. Because we're dealing with integers anyway. So all these operations are going to result in integer results.

AUDIENCE: Are you going to ever be able to get to the highest number?

PROFESSOR: So if it were, like, 99,000? Is that what you're asking?

AUDIENCE: If the number were maxVal.

PROFESSOR: Well, we'll see. I might have it wrong in my implementation, always possible. If I do, I'd probably take one point off me. Yeah, so I have a bug. So I am actually--

AUDIENCE: When it divides, it rounds it down [INAUDIBLE].

PROFESSOR: Yeah. So I have a bug. And I'm not going to try and fix it right now, but yeah. If I were grading this and this were my solution, I'd dock me a point or two.

AUDIENCE: What's your error?

PROFESSOR: My error is that because of this the way that I get my guess here, I'm never going to get to max val. If the computer chose maxVal as its guess, then I'd never get to that. So yeah, you caught me.

AUDIENCE: A friend of mine looked at this and was like, oh, you should define another function to solve it. Is that kind of thing actually allowed on the exam at all, or no?

PROFESSOR: Well, why did your friend think you need to find another function?

AUDIENCE: I'm not exactly sure. It was supposed to be [UNINTELLIGIBLE]

PROFESSOR: Did they say define a recursive function?

AUDIENCE: I think that's what I was trying to do.

PROFESSOR: So I'm solving it iteratively. You really can write a recursive function. And in this case, because we've given you the specification for find number, you'd probably write something like find number helper, and have whatever additional parameters you needed in order to support the recursion.

AUDIENCE: So you can do that or no?

PROFESSOR: Yeah. We'd be fine with that. The point is that you write a function that runs in log maxVal time.

AUDIENCE: Because at the top it says that the magic number is in range maxVal, doesn't that mean that the number would not ever be maxVal?

PROFESSOR: Let me see. Actually, yeah. Because it would be 0 to 999-- or 999,999. Well, I'm not sure if this actually constitutes a bug or not. So I'd have to do unit testing. I don't have any test cases. Which is, again, back to unit testing. You have a question or--?

AUDIENCE: Yeah. Could you do this by saying your guess equals choose a random number between high and low?

PROFESSOR: You could. But it would have-- the analyzing complexity would be problematic, I would think. Doing it this way, by guessing right in the middle, you know it's going to be log maxVal complexity. If you were to choose a random number between low and high, I'm not sure what that would do to the complexity. It would make the analysis a little bit more intricate. And if this were the actual quiz and you put that as your answer we'd probably take a fair number of points off for it. There are people who do study algorithms that do that, though. They use random numbers in order to [UNINTELLIGIBLE] the values.

So we're done with the quiz. And now we've got a list of topics to go over. So, let's start at the top. The first subject that was listed-- so all you got the list of quiz topics for the quiz, I'm assuming? Did all of you look at it?

The first topic is algorithms. And I have big-O notation, exhaustive enumeration, guess and check, successive approximation, divide and conquer, binary search, merge sort, hashing, orders of growth, and amortized analysis. So, first come, first serve. What do people want to see?

AUDIENCE: Hashing.

AUDIENCE: Yeah. Hashing and amortized analysis.

PROFESSOR: Hashing and amortized analysis. OK. So why don't I start with hashing, because that's pretty easy. The idea behind hashing is that you have a function, and it's going to take some input. And it's going to compute an address of some sort.

And let's imagine that I have a data structure that has a number of buckets arranged one after the other. What hashing does is, it says I have this input s, or a key, if you will. And I'm going to compute an address like a number of some sort. And it's going to, say, wind up here. And this relates to Python because this is how dictionaries are implemented. So this s here is going to be the key that you give the dictionary. And then it'll compute a location for this bucket where it's going to put the value.

So, the idea-- what makes these hashing functions valuable, especially with respect to dictionaries, is that you can check for the presence of a key or a value in constant time. Because the hashing function is more or less constant. And once you have an address into the data structure, it's very easy to compute the location where you're going to put something. So, I'm not sure-- does that kind of help you?

You don't need to know hashing in great detail. You're not going to need to know the hashing function for like a string, or the hashing function for a tuple or something like that. The main points that you need to get from hashing is that it computes an address into this data structure. It's a constant time thing. And that's about it, actually. If you have those two concepts down-- well, also, three that that's how dictionaries are implemented-- if you have those in your mind for hashing, you'll be good. Anyone have any other questions for hashing?

AUDIENCE: So we never have to like write a hashing function. It's more like [INAUDIBLE]

PROFESSOR: This is so that you know what it is, and when you see it in the future, if you see it in the future, you're not totally lost. You have an idea of what it's doing. You can define your own hash functions, but we're not going to require that of you in this course. So that's hashing, basically. And all you need to know for the quiz.

So for amortized analysis, the idea behind amortized analysis is that you have-- maybe you have a known sequence of operations. And you're trying to find what the worst case runtime for that sequence of operations is. And you might have a bunch of operations that are really, really low cost, or low complexity. And one really expensive one. And for our purposes, we're trying to find this-- find which of these operations is the dominating factor for the complexity.

So the example I would use is, let's say that I have the task of searching for an element in a big list of elements. So we know two basic search algorithms for this, right? We know a linear search, where we go one at a time through each of the elements. And what's the complexity of that?

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that?

AUDIENCE: Just the length of [INAUDIBLE].

PROFESSOR: It's going to be linear in terms of the number of elements in the list. We also know binary search. What's the complexity of binary search?

AUDIENCE: Log N.

PROFESSOR: Log N. Now, there's a problem with binary search. It has a requirement. Will binary search work on a list that is not sorted? No. You require it to be sorted. So, we've talked about sorting in class. He covered it during the lecture. For basic sorting that we've seen, the lower bound, the complexity of the sorting algorithms is N log N. So merge sort will be an N log N algorithm. So, when-- let's say that we just need to do one search, right? Do I really want to use binary search? Why?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Because I have to sort it first. And if I just do-- if I do I have to sort and then search, and this is going to be-- whereas if I do a linear search, I just have search.

And for just a single search, for binary search, this is going to dominate my complexity. But, let's say that I'm going to do a million searches. Now, it might behoove me to actually sort this first. Because I only have to sort the list once. And then I can do as many searches as I want. So, in the grand scheme of things, if I'm doing a million searches, then linear search is a really poor choice. And binary search would be my best option.

And this is about as deep as we want to go with amortized analysis. It's this idea that there's a certain point at which one algorithm becomes more important than another algorithm, because the composition of its operations kind of determines, it changes as you increase the number of elements you're working with the number of times you're going to do the operation. That's kind of what you need to understand for amortized analysis. And it can be used in other types of analyses, but we're not going to get into that.

All right. What other topics do we want to go over for algorithms? We've got big-O notation, exhaustive enumeration, divide and conquer. Successive approximation, merge sort, and orders of growth.

AUDIENCE: Can we go over merge sort?

PROFESSOR: You want to go over merge sort? OK. So-- and today actually, I have my code.

So merge sort is a divide and conquer algorithm. And the idea is that I am going to take a list of elements. And I'm going to divide the list in half. So I'm going to get a left list and a right list, and they're unsorted.

And then I'm going to call merge.sort on them. And I'm going to keep recursively calling merge sort until I have one element or no elements in the list. And then I'm going to return the list. After I do a merge sort on the left and right halves, I'm going to merge them. And the merge operation is just going to start at the beginning of the left and right halves of the list, and it's going to append the smaller element at the beginning of each of the lists to result until it gets to the end of the lists.

So I'll try a blackboard demonstration in a second. Or explanation in a second. So, this before parameter, all this is doing is it's determining the order of the elements. So it's saying that left should come before right. It'll return true if left should come before right, and false if right should come before left.

So, intuitively, because I don't think that was a very good explanation, let's say that I have a list of elements. And they're unsorted.

All right. So it's an unsorted list of elements. When merge.sort first starts, it's going to divide this list into left and right halves. So basically it's going to take the middle here. And then it's going to call itself, on this left half and then on this right half. And it's going to keep doing this. It's going to divide this half into say -- 5, 2, 1.

Now, when it gets to these little single elements, it's going to do something interesting. It's going to merge these two. So when it gets down to these single elements, it's just going to return the list as is, it's not going to do anything to them. And then the code after this, division, is going to merge them. And the way it merges them is, it's going to start at the beginning of these two lists. And it's going to say, is this element larger or smaller than this element? Or is it going to come before this element? And it's going to construct a merged list. And then return that as its result. So this is going to come down.

So now at this point it's popping back up the stack. It's going to return this. It's going to merge these two lists. Starts at the beginning of both lists. One comes before two. And then this list is done, so now it only has this list to do.

After each merge step, these lists are going to be sorted. So now where are we at? So now let's do this guy. So this guy gets merged into 6 and 9. And then this guy, I didn't divide up. So, bad on me.

So we're going to merge those two single element lists. Now these get merged. So these guys are going to get merged. So, this is going to become, I think my tree got messed up a little bit. Now we're going to merge these lists. So we're going to start at the beginning. 1 comes before 3. First element is 1. 2 comes before 3. Now we're here in this list. We're still at the beginning of this list. 3 comes before 5. 5 comes before 7. And we're here. 6 comes before 7. Then we have 7. And then 9 comes before 10. Now we're done with this list. So now we have 10, 13. And at the end of the last merge the list is sorted.

So, I'm not sure how clear that was.

AUDIENCE: What happens if there's [UNINTELLIGIBLE] to begin with? Like all of your split equally into [UNINTELLIGIBLE]? What if there's one more element at the top, [UNINTELLIGIBLE]? Do you know what I mean, like, there'd be one left over?

PROFESSOR: What do you mean? So there's an odd number of elements in the list? So you would just get an uneven split. So your left sub-list might have 6 elements. Say you have 11 elements in the list. Left element would have 6, or left list would have 6 elements and then the right list would have 5.

AUDIENCE: [UNINTELLIGIBLE] then that would split with 3 and 2. And then the 2 would split to 1 and 1, the 3 would split into 1 and 2, and then the 2 would split again?

PROFESSOR: And then the 1 would actually make a call, but it would be an empty list. It would split, but one of them would be an empty list.

AUDIENCE: Oh. So and then it would just get--

PROFESSOR: And that's kind of what happened with my tree. I didn't go all the way down on some of the branches of my tree. I mean, the main idea behind merge.sort is that you split the list in half, whatever size it this. And then you merge sort those separate halves. And merge sort is going to do the same thing with those separate halves. It's going to split them in half. And sort those halves until it gets to the smallest case, which is an empty list or just a single element. And then it's just going to start merging these lists.

AUDIENCE: Right, I was just wondering what happened [UNINTELLIGIBLE]

PROFESSOR: Yeah, you'd split it into 2 and 1, and then that merge sort would be like 1 and 1. And then this 1 would split it into a list of 1 and nothing. That's all. And then the merge between that's obvious, right?

AUDIENCE: So in the merge sort lecture notes, at one point, lambda is used. Can you explain a bit how this works?

PROFESSOR: Does he actually use a lambda?

AUDIENCE: Yeah. Why is it useful? I guess [UNINTELLIGIBLE].

PROFESSOR: Ah. He used lambda. OK, Python has these functions called lambda functions. And what they are essentially are like one-liner bits of code that return a value.

So let's see. I can't believe he used lambda. I'm trying to think of where I would actually use them. Like, an example. And I don't have one off the top of my head. You're not going to see a lambda on the quiz. The syntax is not too difficult. It's basically, so, like lambda x. And then it's going to return, like, x-squared. This is going to create a function. And it can be something like, I assign this lambda function to a variable square. And I'm going to pull up a separate--

So I'm creating a function square. But instead of writing something like this, I write it as this. And ...

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that?

AUDIENCE: Do you have to save it before it lets you run it?

PROFESSOR: Yeah. That's, again, what I'm doing. I'm just trying to think of a witty name. I'm actually stunned that he whipped lambda out in lecture.

So really it's a way of defining these one-liner functions. Python's lambda functions are kind of broken, at least in the 2.5, 2.6, 2.7, 2.x versions. 3.0, 3.x versions of Python, I think, actually do away with lambda because they were broken. You find lambda functions in a lot of other programming languages like Lisp or Scheme. And you're never going to see this on a quiz. But if you want to learn more about them you can look it up online.

AUDIENCE: While you're running the function, what is the [INAUDIBLE] The same thing? Oh, OK.

PROFESSOR: Yeah. I had it there as an illustration that the functions are-- they're functionally equivalent. And it actually serves to illustrate another point, that functions are objects and you can sign them and pass them around and stuff. But lambda functions on their own, you're not going to need to know that piece of linguistic Python, Pythonese. Bad sentence. You're not going to need to know it. Does that work? OK.

I'm going to have to ask him about that, because I can't believe he used lambda in lecture.

AUDIENCE: In one of the lecture notes, he used something called x-range.

PROFESSOR: x-range.

AUDIENCE: What does that do?

PROFESSOR: Did he talk about generator objects or iterate? Or yield? You haven't seen those?

OK. So what you're asking is, something like that. He used something like that. The difference between range and x-range is that range returns a list of integers, and x-range returns in generator object. And I know that really meant a whole lot to you.

So let's say that I assign ...

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. that's what I'm going to show. So if I do print list of integers, or list of numbers, it's actually going to print the list of numbers. Someone fall back there? So it's actually going to print that list.

Now, if I do something like-- pretty sure this is not going to print the... The reason is that if you look at the type of the object that's returned, it's-- well, x-range doesn't really help much either. Why does he do this to me?

It's a way of-- so let's say that you have a list of a billion elements, or numbers, or things that you need to iterate over in a for loop. It's probably not a good idea to bring all of those into memory at once. So, or to necessarily keep them all around. You want-- that's where things like x-range come in. So, let's do this.

AUDIENCE: [INAUDIBLE] for i in x-range 10 [INAUDIBLE]

PROFESSOR: Yeah. So if I do this, it's going to be the same as if I were to do this. It's going to have the same functional effect, but the difference comes in-- let's say that I were to do something like this. Lot of numbers, right? I'm hoping Python crashes on this, because this is my whole point. Yeah, so there's too many items.

But let's see if I do it with x-range. I should probably comment this out. I chose a really too large number, didn't I? I'm going to find a number that has too many list elements, but Python can still feel comfortable constructing an x-range out of...

The point is that there's a certain point, a certain number. I don't know if it's going to do that comfortably. So you see this little delay that's occurring? Actually, this kind of a long delay? It's actually hanging up on this range command. Because it's actually constructing this list object with all these integers in it. So, let's say I do this. Are you going to need to know the exact difference? No. Is there a difference? Yes. Am I prepared to explain it exactly, right now? Not really. Previous iterations of this class have covered the yield statement in Python, which allows you to create functions that create generator objects that... You did yield?

AUDIENCE: [INAUDIBLE]

PROFESSOR: All right, then you should know about this. So range is going to give you a list of integers. x-range is going to give you a generator object that uses yields to produce these integers. So let me see if can rescue my machine. All right. So, he covered something like that with yield? So if I were to, say, do i in my x-range-- back to where I was. My x-range does the exact same thing that x-range does.

Now, as I said before, if I do a list, or if I do range, that's going to be a list, right? If I assign this to ... see that? That's what the yield statement does for you. It produces that generator and that's how x-range functions.

So, basically it allows you to do some interesting things. Like, in this case, it's not too interesting because it's just returning one integer after the other. But let's say that I wanted to do -- def my_squares. All that's going to do is return these squares of all the numbers between 0 and max N. So if I were to do this, it should print out all the squares-- or the square root of all the numbers between 0 and 9. Make sense?

So that's what yield gets you. Those are generator objects. And that's how x-range works. That's how it's different than range. Does that make sense to everyone? OK. I don't need to beat that to death any longer? I was not expecting that question.

OK. What should we go over next? Is everyone comfortable with orders of growth?

All right. The next major topic area I have is linguistic issues. So, this covers stuff like exceptions, polymorphism, classes and objects. So people want to go over classes? Objects? Polymorphism? OK.

AUDIENCE: What's polymorphism?

PROFESSOR: Yeah, we'll get to that. So before my computer crashed-- all right.

So here are some class definitions. So I have a shape object, and I've defined two methods, area and perimeter. They currently do nothing except for raise not implemented errors. And now I have to find a class hierarchy. I have a rectangle class that inherits from shape. So shape is a superclass of rectangle. Rectangle is a subclass of shape. This also says-- well, let me ask the question, is shape a rectangle? It's not rhetorical.

AUDIENCE: [INAUDIBLE]

PROFESSOR: Not necessarily. Is rectangle a shape?

AUDIENCE: Yes.

PROFESSOR: OK. It's an easy question. It's not a trick question. In the rectangle class, I have an INIT method. It just takes the length and the width. So computing the area is really easy. And then computing the perimeter is also very easy. And now I have a string representation which is just rectangle with the length and width. Ellipse inherits from shape and it has one radius-- or one, the long axis and the short axis. And computing the area for that is pretty easy. Computing the perimeter, on the other hand, I had to look it up online. So if you're interested in the formula I used go take a look at this website.

Again, I have a string method. And then now we have a square which inherits from rectangle. Because square is just a special case of a rectangle. And so I'm going to reuse it. I'm just going to make both lengths the same. That make sense to everyone? And I don't need to override the area and perimeter because they're pretty easy to understand. Or they're the same.

And then circle. It just inherits from ellipse, and same thing. I'm just going to reuse the stuff that I have in ellipse.

All right. I've defined a bag of shapes here. Circles, squares, rectangles. Rectangles. Lots of stuff. Now, to answer your question, what polymorphism allows me to do is, it allows me to treat subclasses that share a particular superclass, or a parent class, the same. So I've defined an area method on the shape class. And all of my subclasses, rectangle, ellipse, circle and square, inherit from this superclass. And some of them override area, some of them don't. It doesn't really matter to me.

What matters to me is that if I have a bag of these shapes, I know that I have an area method. And I don't need to know the particular or the type of the object that I'm calling the area method on. Right? Python is going to take care of that for me. I can just iterate through this bag of shapes, call the area method, and I can get a sum of the areas. That's what polymorphism gives to me, is it allows me to treat objects uniformly. It allows the objects to change their behavior based on the particular type of object that I'm using. But I don't need to know the particular details. So it's pretty powerful.

So it's like, if I were to create a racing game and I wanted to drive a car, I could have different models of car. But my game engine wouldn't need to know to drive method to call for. Like, a Ford car versus a Chevrolet, all that stuff. That kind of answer your question?

AUDIENCE: Yeah.

AUDIENCE: So if there's not an area function that overrides it, what is it going to sum if it just raises a non-implemented error? Yeah, if you haven't redefined.

PROFESSOR: In this example, I'm using all what are known as concrete classes. So they're not abstract. They all have implementations of area. But if I were to say, for some reason I decided that I wanted to make something amorphous. This should raise an error, right? Unless I have invalid syntax. So when it gets to there, yeah. Any other questions?

AUDIENCE: So what's the difference between inheritance and [UNINTELLIGIBLE]?

PROFESSOR: They're related. But what inheritance means is that if I say that rectangle inherits from shape, that means it gets all of its methods and attributes. But it's not necessarily polymorphic in that case. It's just, it's like saying a rectangle is a shape so it has all these characteristics of a shape and then some. The polymorphism comes in when I override the area method and then I can treat all shapes, whether they're rectangles, squares, circles, uniformly with the area method. You might-- I'm trying to think of an example where it would be.

So let's say that I didn't do this in a class or entered a way or using classes. And so I had a really naive implementation that said, let's represent them as dictionaries.

So I have two really stupid functions here. And they say they make a circle and they make a square. And so really all it's doing is representing these as dictionaries. And the key is side and radius.

Now, let's say that I want to have my list of shapes. I make a square. I'm going to make a circle. Another square. OK, so now I've got a bag or shapes or a list of shapes. What's that?

AUDIENCE: Can you [INAUDIBLE]?

PROFESSOR: Spending too much time on that one. All right. So if I were to take a look at this. It's going to print out a bunch of dictionaries. Now, the question is, what if I wanted to compute the areas of these data types? Using polymorphism, I can just call dot area on each shape. But if I wanted to do the same thing with this code, I would have to have square area s, return, d, side. And then def circle, area, c, return, 3.14. 9 times c-- I'm already getting tripped up.

So even this is not so bad, because if I do something like this, I can still get the areas, right? And I can do this because I know what's in those positions in the list. So.

AUDIENCE: You forgot to close your parentheses.

PROFESSOR: Oh. OK. So it's going to work. But now, if I want to print out all the areas of all the shapes in this loop here, how am I going to do that? Because I can't just call square area of s, because what if s is a circle It's going to give me an error, right? I could get around this by saying, if radius in s, then I know it's a circle, right? Or, side in s. But you see how the code is getting kind of ugly now?

I have to know details about how these different objects are implemented, how square and circle are implemented in the dictionary. I have to know what type they are before I compute their area. And I can do it, but it's kind of ugly. Whereas if I invest a little bit of time in a class hierarchy, I can do this much more easily. Much more cleanly. It's a way of abstracting from the problem. You don't have to worry about the details of the individual shapes. Did I kill that? And beat it into the ground.

AUDIENCE: Just for clarification. So the point of polymorphism is just like inheritance, essentially, kind of boiled down?

PROFESSOR: The point of polymorphism is that if you have a class hierarchy of objects, you don't need to know the details of the objects, of what specific type an object is. As long as you know that in the case of--

AUDIENCE: So long as there exists a method that you want to use.

PROFESSOR: Right.

AUDIENCE: Where it inherits from that class, you could use it.

PROFESSOR: Yeah. So in this particular example, shape has an area method. And I have a bunch of concrete subclasses of shape that have all overridden the area method. And they do it in different ways. You compute the area differently for a rectangle versus an ellipse. I don't have to worry about which particular shape it is. I just call the area method, and I don't have to do all the checking that I do here. Where I look at, is this a circle, is this a square. I don't know, so I'm not going to compute that.

AUDIENCE: How do you avoid the checking? [INAUDIBLE]

PROFESSOR: So--

AUDIENCE: [INAUDIBLE]

PROFESSOR: See?

AUDIENCE: Oh, OK, Great. Yeah.

PROFESSOR: [INAUDIBLE] Anyone lost? Any other questions? So that actually touched on a couple of topics. So we saw exceptions here. Is there anyone who wants to see anything else on exceptions?

AUDIENCE: What's the difference between pass and continue?

PROFESSOR: Pass and continue? OK. So the question is what's the difference between pass and continue. So when you have pass, this is basically a null command. It's a do nothing command. I can actually run this code right now. And it will do absolutely nothing useful. But it also won't complain. It's a placeholder. Sometimes-- so a lot of times you'll see in your code something like this. Let's x equals 150. If x mod 2 equals 0, pass else, do something useful.

So we see this a lot in your p-sets. It's not a slam on you or anything. But you could simplify this by saying-- but the way that you conceptualize a problem forced you to have this if statement. And then you didn't have anything useful to put in here, so you had to put a pass in order to make Python not complain. So, let's be a little less flip and more concrete. It's getting late, I'm getting punchy.

All right. So if I were to run this now, it's not going to print anything because it hit the pass command. Now let's make this on. So if we've verified that this code works and that I'm not blowing smoke. Now, let's take out the pass. Python's going to complain. Why? Because after this if line, here it's expecting a block of code. But there's nothing there. The pass keyword give you a do nothing command. It says, I don't really have anything to say here, but you want me to put something here so I'm going to put something here. But it doesn't do anything. So that's pass. so It's a placeholder, is the basic answer.

Now, continue is different. Because continue actually has a function that is useful. Let's say that I'm not too familiar with the range command, and I don't know that I can specify a step. And I want to print out all the odd integers. And I also am going to restrict myself and say that I can't use an else or something of that nature. So I'm going to handicap myself. This x-range -- is x-range cool?

So. I only want to print out odd numbers. Or we can do it like this. Either/or, it doesn't matter. So, comment this. What continue does is it, if you're within a loop, as we are here with this for loop, tells Python if it sees a continue statement, do not continue beyond this point within the block of the loop. Go instead and perform the next iteration of this loop. Go to the top of this loop. So, what this code should do, I'm hoping, in case I don't-- I might have a bug-- it should print out the odd integers.

So what's happening is that when the integer is even, it's hitting this continue statement. And this is telling Python, go back up to the top of this for loop and do the next iteration.

AUDIENCE: If you were to have [UNINTELLIGIBLE]?

PROFESSOR: So if I just had pass? It'll print all the integers. That work? Anyone confused?

AUDIENCE: You could also have your code if not that [UNINTELLIGIBLE].

PROFESSOR: Yeah, but [INAUDIBLE]. There are some schools of thought that say that continue is something that disturbs the flow of code and should be avoided. I don't subscribe to that. I think that if it makes your code clearer, then use it. Which is to say that, yeah, you can rewrite a lot of your code to not use continue.

OK, do we need to go over anything with linguistic issues? Are we good on classes?

AUDIENCE: Can we go over exceptions, please?

PROFESSOR: So there is a question on exceptions. So exceptions are another way of flow control. And if I can pull up my source file, I have an example of handling exceptions. There we go.

Exceptions give you a way of handling certain errors that occur. So, the easiest one to come up with is division by 0. So normally, if you would do a division by 0 -- so I'm going trying to divide 10 by 0. The universe won't end, but Python will be very unhappy. In programs that are more complex that the ones that we've been writing, division by 0 happens but you don't necessarily want the program to crash, right? You want to be able to handle that error in a semi-graceful way. And in this particular example, with a division by 0, we can do that with exceptions.

So let's say that instead of the program crashing I want to print out a nicer error message rather than everything in big red. I say, well, you divided by 0 so maybe you should change your numbers. The way I capture, or catch this exception, is I have a tribe keyword, a colon, and then I have this block of code. This is the code that could potentially raise an exception where the error might occur. This code could be-- in this case, it's just straight up assignment and division. It could be a function that raises the exception. It doesn't matter. The point is that there's an exception that might occur.

If an exception occurs, then Python is going to look past this block of code here at these except statements. And it's going to look at the type of exception it has. In this case it's a by 0 division error, and it's going to match the exception against whatever type this is. And you can put inside here whatever code you want in order to handle this exception.

So in this case I'm just printing divided by 0. I could be rude and I could say-- see, can't even spell. So I can handle this exception however I want. But the program's not going to crash. And this program's actually not going to really convince you of that, but you see that-- the first thing you see here is that there's no big red blaring message that says you have an exception. It's all code that I've written. And I could even be silent about it. I could say, I'm not going to say anything. I'm just going to fail silently. Tell me it's all good. It's not going to tell me I divided by 0.

It can come in handy if I like have an input loop. So let's say I'm going to do a little bit more, be a little bit more creative and do-- oh.

AUDIENCE: [INAUDIBLE] How does it [INAUDIBLE]?

PROFESSOR: That's an internal Python thing. So if you want to learn about all the wonderful exceptions that Python provides you in its class library, you can go to the Python documentation. Or you can even do a Google search. Of course, Chrome is being very slow, and the network's probably down.

So built-in exceptions. So these are all the exceptions that Python has. And you can use these too. Let's say that, let me get back to my contrived example. I want to create a loop that says-- let's say that I want to catch, I don't want to divide even numbers for some reason. What do we have here? What can we find? I think there's a value error. There we go.

So let's say that I'm going to raise a value error. And I am actually going to do this this way. So I've created a function that's not going to allow me to input even numbers. It's going to raise a value error if I enter in a 2 or a 4 or whatever. But, if I do enter in an odd number it is just going to return that number to me. So I'm going to use that function now.

And I'm just going to do the silly thing and divide, do a division, and print it out. So if I comment out this code up here and I comment this out, and I run this, so-- let's do 3. That's interesting. Apparently Python is a little constipated right now. We edit this during the summer, so we can take that out, I think.

All right. So it does what we expected it to. It's going to keep on prompting us for numbers and it's going to divide them for us and give us the answers. So we've got a really dumb calculator here. Except when I try to do even numbers, because my calculator is really, really dumb.

So it's going to raise a value error. And it raises the value error in this function here. I can handle this, though, because I know how. So what should I type here? OK.

AUDIENCE: If you wanted to specifically catch that error. But you could just leave it as except, and it would catch any error?

PROFESSOR: Yes. [INAUDIBLE].

AUDIENCE: So if any error were to be thrown, then [INAUDIBLE]. But if you wanted specifically to say, OK, the value error [UNINTELLIGIBLE].

PROFESSOR: Right. So what she was saying is that I could do this. I don't have to specify what error I'm expecting. I could say-- But it's not going to give me any detail about what happened. And what I'm going to do is, I'm going to say that if I have an exception then I'm just going to repeat my getting of the numbers. So, let's say that I do this. Something happened, not really specific. Nothing happened? OK, I divided all right.

AUDIENCE: What if you put in [UNINTELLIGIBLE]?

PROFESSOR: What's that?

AUDIENCE: [INAUDIBLE] Would it also come back in case something happened?

PROFESSOR: Yeah. That's a good question. Something happened, but we don't know what. We know we have a bug, because an exception was thrwn. But we don't know what it is.

AUDIENCE: Because it's never going to crash.

PROFESSOR: Yeah. It's never going to crash.

AUDIENCE: So a useful example of when to use this, would that be like [INAUDIBLE], and you have the list to choose from the letters [INAUDIBLE] it would remove each of the guesses? If you accidentally guess something that you've already guessed?

PROFESSOR: Yeah, you can raise an exception to say, already guessed this.

AUDIENCE: As opposed to the program, that first one I ran, I accidentally did the same letter again, and the whole program just crashed.

PROFESSOR: Yep.

AUDIENCE: So you can raise an exception for whoever's [INAUDIBLE].

PROFESSOR: Yeah. You can catch any errors that were raised. So we've got a couple of exceptions that are raised here. And actually this program can crash. Oh, wait. No, it can't. Because [INAUDIBLE]. It thinks [INAUDIBLE]. Silly Python. So it's not going to crash at all. We did a really good job, right?

So now we want to get a little bit more specific so I'm going to catch a value error. So I'm only going to handle this one type of exception. Because-- so it still works. A, something happened. But now we know that it's because I did something silly. And, now I can break the program because this keyboard interrupt, that's actually an exception. When you hit Control-C, it's going to break you out of the program. But because we're catching that exception it just kept going.

AUDIENCE: [INAUDIBLE] make an exception [INAUDIBLE].

PROFESSOR: Yes.

AUDIENCE: [INAUDIBLE]

AUDIENCE: [INAUDIBLE]

AUDIENCE: Wait, what?

[INTERPOSING VOICES]

PROFESSOR: I'm quoting the girls over there, so.

AUDIENCE: Are you?

PROFESSOR: Uh-oh. So something happened. Oh, wow. How did that happen? Oh, I know how that happened. What am I missing here? So I caught the keyboard interrupt, right? I didn't go to the top of the loop. So what happened is, it tried to execute this, but num2 hadn't been defined. So we're going to kick me back up to the top of the loop here.

AUDIENCE: Oh. It's a clingy program.

[LAUGHTER]

PROFESSOR: Yes. It is a clingy program. Anyway. So, does everyone get the idea of what the exceptions are? So I don't have to go any more?

AUDIENCE: In this except up there, there's always [INAUDIBLE]?

PROFESSOR: Yeah.

AUDIENCE: But what does this [INAUDIBLE]?

PROFESSOR: So When you get an exception, when an exception is raised, it's actually an object. So there's an object that's associated with it. And in a lot of cases you really don't have to be concerned about it, but sometimes you want some additional information. So these are not very interesting exceptions, but some exceptions might have some additional diagnostic information. So if you were, say, logging the run of your program. And you wanted to diagnose any exceptions that you caught? Like, say, you got a generic exception or you got something that was kind of odd that weren't expecting, you could print out additional information. And so E is the name of that object. So, I can do this. Well. This is because -- if you give it a primer like that, it's going to put that message into the exception. So now I can retrieve it when the exception is raised.

So if I'm raising the exception, I can provide additional information. So that's the message that I've put into the exception when I raised it. Does that make sense?

AUDIENCE: Well, how is that different-- so is it the same thing as saying print, [INAUDIBLE]?

PROFESSOR: Yeah. So this is a really [INAUDIBLE] example. So this is just going to print out whatever message I passed in. So when I say this and point to my screen, I really meant this and highlighted this. So, this line is going to print out this message that I passed in when I raised the exception. Because what I'm actually doing is, I'm creating a value error object that has this message associated with it. And that's what gets passed up this exception chain until I finally handle it. here. And E holds the reference to that value error object.

AUDIENCE: So E is an actual value?

PROFESSOR: Yeah. So if I do something like this-- and I print out the type-- you can see that? It's pretty neat. But did that answer your question?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Could it be anything? So let's think of something like this. Like a list.

AUDIENCE: No. I meant in your accept value error dash E. Could you put any variable there, or does it have to be E?

PROFESSOR: Oh, any name?

AUDIENCE: Yeah.

PROFESSOR: I've always used E. That's kind of a convention. But I think you'd be fine doing that. E is the convention that I use, but some people don't like that. They're like, you should name it something a little bit better. But honestly, why bother? Ok. Are we good with exceptions? That answer your question? Any other questions on exceptions?

AUDIENCE: Is there a built-in amount of errors you can have? Like value errors, [UNINTELLIGIBLE], can you make up your own?

PROFESSOR: Can you make up your own exceptions? Yes. You can do that. You can say-- but-- and-- I think this is going to work. It's been while since I've defined my own exceptions. So and again, I can handle it. So if you ever work on big systems, a lot of times they'll have their own exception hierarchies that define all sorts of exceptions for the specific application you're working on. Python's built in exceptions are pretty rich, so-- there we go.

AUDIENCE: [INAUDIBLE]

PROFESSOR: I don't think you do. It's convention. I could probably do this, I think. Oh, wait. The only difference-- or the only problem is, is that now-- see I expect my exception to have a message attribute. And I don't have anything on this. So I have to do this. Or something similar, right? I have to define message on this. And I think this'll work. Well, actually, it does guard against it. So yes, you do have to inherit from exception. It's one of the few things that Python seems to guard against. So, does that answer your question?

So we're getting kind of late. The only other topics that we have are the simulations and the distributions and that's about it. So, are there any questions on Monte Carlo methods? We've kind of done those to death, right? Real quick, someone humor me, what is a Monte Carlo method?

AUDIENCE: An algorithm that uses random sampling for trials to [INAUDIBLE].

PROFESSOR: Right. So, it's-- what's that?

AUDIENCE: [INAUDIBLE]

PROFESSOR: It computes some value or solution to a problem using a random process and repeated trials. So--

AUDIENCE: Is it almost brute force?

PROFESSOR: No, it's not what we call brute force. I mean, it's like we computed pi by throwing a dart at a chalkboard or something like that. And it landed in a square, and we used some sort of analysis to figure out what pi was based on that throwing a dart at a board a million times. So that's what a Monte Carlo method is. Anything that uses random stuff to compute a value. A random simulation. Where'd my chalk go?

And we actually a couple of Monte Carlo methods when I showed the solutions for problems (3) and (4) on the quiz. I used a random number generator in order to figure out the probabilities of not rolling a 6 and all that stuff. And I used the Monte Carlo method to show me that I was wrong on my initial answer for rolling exactly one 6.

So, is everyone comfortable with coefficient of variation? Because I did that first, initially.

Confidence intervals and levels? Does anyone have any questions on that?

AUDIENCE: What is a confidence interval?

PROFESSOR: What is a confidence interval and a confidence level? OK. So, let's say I do a political poll. Let's say I do a political poll and I sample 1,000 people in the population. And I say 46% of them will vote for Candidate X. My margin of error is plus or minus 4, and my confidence is 95.

What this is saying is that if I conduct this poll, the same poll, sampling 1,000 random people at a time, over 100 trials-- so I do 100 polls-- what I'm saying is that 95 out of 100 of those times, my level of support for Candidate X will be between 42% and 50%, right? Basically what you need to know for confidence levels and intervals.

The other thing that you need to know for confidence levels and intervals is that if you have a normal distribution, with a standard deviation sigma, then 1 standard deviation from the mean is going to have-- is that 66 or 67? Is it 68? I wrote it down, 68. Thank you.

And then 2 standard deviations is going to be 95. And then 3 standard deviations is going to be what, 97? Or is it 99.7? 99.7. So, but you need to just know that for normal distributions. So if this is one standard deviation, it's saying that 68 of the sample points are going to be within 1 standard deviation of this mean. And then if you have 2 sigma, then that means 95 are going to be within 2 sigma of the mean. And then if you have 3 sigma, then 99.7 are going to be within 3 standard deviations of the mean.

So I think that's all you really need to know for confidence intervals and levels, and also for normal distributions.

So we're actually kind of at the end of the review session. So does anyone have any questions that I haven't answered? No? All right. I'm done.