[SQUEAKING]

[RUSTLING]

[CLICKING]

ANA BELL: OK, let's get started with today's lecture. It's going to be more of a chill lecture than what we've done in the past, even though we've got quite a few things to cover, as you can see from this title slide. I'm not going to go super duper fast, so please feel free to ask lots of questions. And then the second half of the lecture will be really chill, because we're going to be talking about testing and debugging strategies, SO super high level topic.

But first we're going to tie up some loose ends related to lists and relating to functions. So we're not going to introduce a lot of new syntax. These ideas are more optional in your day to day coding. But they're just really, really nice to know.

So let's first start talking about this idea of a list comprehension. So you've been writing functions that deal with lists. And one really common pattern that I hope you've seen so far is the following. So this code right here shows something that we've definitely coded together and you've definitely coded in the finger exercises and the quizzes. And so it is a really common pattern.

So the idea here is you have a function that creates a new list where the elements of this new list are a function of the input list. So the pattern here is we create a new empty list inside the function. We have a loop over every element in the input. And to each one of these elements in the input, we apply the same function.

So in this particular case, we're taking that element and squaring it. And each one of these elements we're appending to this new list, originally empty, until we've done this function to every element in L. And then we return this newly created list.

So now since this is a really common thing that programmers do, Python allows you to do this exact functionality with one line of code. And the way we do this is using something called a list comprehension. So the way that we do a list comprehension, essentially taking these four lines of code from this function, we are going to write them in this one line of code that looks something like this.

So the idea here is with this one line of code, we're going to create a new list. We're going to have an iterator that goes through some sort of sequence of values. And we're going to apply the same function to every one of those elements. And the other optional piece that we can add inside this list comprehension is to only apply that function if some condition holds. So let's look at this example and see how we can convert these four lines of code to one line of list comprehension code.

So we've got creating a new empty list. This is going to tell Python to create a new empty list for us, so just open and closed square brackets. And within these open and closed square brackets, we're going to write a one liner expression. And this one liner is going to encapsulate these two lines of code here.

So the expression-- sorry, the function we're going to apply to every element in L is going to be taking that element and squaring it. So on the right-hand side here in the list comprehension, we've got some e squared. Well, what is e? Well, it's going to be every element e in L. So if we read this in English, we basically say L is going to contain elements e squared for e in L. So it sounds weird, but it kind of makes sense even if we read it in English. And behind the scenes, Python will take one by one each element in L, square it. And that's the sequence of elements it will populate this Lnew with.

Now, what if we add a condition to that? So let's say we want to create this new list of elements only for even elements. So we only want to square the even elements within my original list, L. Well, if we were to write a function that does that, we have to add this extra condition here. So everything else is the same except for this if e percent 2 equals 0. This tells Python to only grab elements that are even, divisible by 2.

So how do we write this in list comprehension form? So here's a new list. And this is the function to apply only if the test is true. In list comprehension, this is my new list. I've got the for loop is over here. And then the test to apply is at the end here, if e % 2 equal 0. And then what is the function we're applying? It's just e squared like before. So the test just gets appended to the end of this list comprehension expression here. Yeah?

AUDIENCE: [INAUDIBLE] running faster, is there a reason to do that?

PROFESSOR: Does it run faster? I'm not sure, actually. It might run marginally faster but probably not significantly. The reason to do this is because as you get more practice with it, this will be easier to read in code. And often if you see a large chunk like this, your eyes will glaze over. You're not going to want to read a chunk like that. But if you see it all in one line, you're going to think, well, how bad can it be? [LAUGHS]

And so you can come up with really complicated list comprehension expressions. But usually we reserve them for really simple, really quick ways to create these lists that you just populate with some values right off the bat. So it just makes the code a lot easier to read.

OK, so list comprehensions are pretty useful. If you get a little bit of practice with them, you'll find yourself kind of using them all over the place. And they basically replace code that looks like this. So these lines of code is a very generic way of writing this one liner list comprehension.

So here I've got a function f that I would like to apply. This expr expression is the function I would like to apply to each element. This is the list I would like to apply that function to. And the test is going to be the conditional. In this particular case, this test means I apply it to every single element. But you can imagine having a function, which in the previous case, we would say lambda x, x % 2 equals 0 as our condition.

And then the function that we're essentially replacing is this with list comprehensions. We create this new list. Again, this is the pattern that we saw in the previous slide. We loop through every element in the list. If that condition holds, append that function applied to each element. And then at the end, return the list. So this is just a very generic way to write a list comprehension.

So let's look at some concrete examples. So here I'm not applying the function e squared to a particular set of elements from a list. I'm applying it to the sequence of values given by range. Remember when we were talking about for loops iterating through things, they can iterate through integers following some pattern, like range 6, range 1, 9, 2, something like that. As long as you have a sequence of values you can iterate over, you can plop that into this list comprehension.

So you can iterate over lists. You could iterate over tuples. You could iterate over these direct ranges. You could iterate over a range of the length of a list. Whatever creates an iterable for you, you can put that in the list comprehension.

So in this particular case, the way I read this is I've got something that I'm squaring. And what's the thing that I'm squaring? It's going to be each value in range 6. So I think about it like, what is this sequence of values that I'm going to operate on? Well, it's going to be the numbers 0, 1, 2, 3, 4, 5. And the thing that I'm going to do to them is square each one of those values. So the end list that I get out of this one liner here is a list containing 0 squared, 1 squared, 2 squared, 3 squared, 4 squared, and 5 squared.

We can add a condition to that. So here I've got each element squared for e in range 8, only if e is even. So then the way I think about it is let's start off with what every element in the range is. Well, it's 0, 1, 2, 3, 4, 5, 6, 7.

The condition I'm applying to that is that it's even. So the numbers I'm going to end up with, I'm filtering all those to only contain 0, 2, 4, and 6, because we go up to but not including 8. And then I'm going to square every one of those. So the end result from this list comprehension is a list containing the elements 0 squared, 2 squared, 4 squared, and 6 squared.

And lastly, we've been doing just single integers in the resulting list. But as I mentioned, we can do more complicated things. So as long as we can write a little expression here for the thing that we'd like to calculate or add to the list, we can put it in the list comprehension.

So in this particular case, the element that I'm adding to my list comprehension, my resulting list from the list comprehension is a list itself. So each element in my resulting list is another list. And that inner list is going to contain two elements every time, the thing I'm actually iterating over and its square.

And I've got a condition here. So I've got the elements 0, 1, 2, and 3. That's the range. But I'm only grabbing the odd ones in this particular case. So the resulting set of numbers that I'm going to apply this to is going to be the number-- is the numbers 1 and 3, because those are the two odd numbers in range 4.

And so the resulting list is going to contain two elements. So this outer square bracket is the list that I've created. And its elements will be the element that I have actually iterated over and its square as a list, so 1 and 1 squared for e and e squared when e is 1, and then 3 and 9, 3 squared when e is 3. Questions about that?

OK, so pretty cool. It's a really nice way to create lists really quickly. Like if you wanted to create a list full of 0's, full of a hundred 0's, no need to do a loop. You basically do a list comprehension that says square brackets 0 for e in range 101 or 100. And then you've got yourself a nice little list full of a hundred 0's.

All right, so think about this and then tell me what the answer is. So the idea here is we have this list comprehension. And just go through it step by step. It looks a little bit intimidating. But the first step is to look at for loop and ask yourself, what are the values I'm iterating over? Then look at the condition, if there is one. There is one. In this case, it's at the end here.

So now, what subsets of those original things you're iterating over are you actually keeping? And then from those things that you're keeping, what function are you applying? It's the one right at the beginning. So think about it. And then I'll ask you to tell me.

So step one, what are the values I'm iterating over, the full values, not including the condition? Someone yell it out. Yeah, that list in the middle. Awesome. OK, so xy, abcd, and then 7, and then what's the last thing? Is it the number 4.0 or a string? Yeah, exactly, 4.0. OK, string, string.

Step 2, from this--

[NO AUDIO]