PROFESSOR: By way of review, we're going to start off with loops. So, what are two loops that we've covered in lecture or two types of loops?

AUDIENCE: FOR and WHILE.

PROFESSOR: OK. Yeah. FOR and WHILE. I'm going to start with WHILE loops because that's what I have first. Can someone walk me through the syntax of a WHILE loop?

AUDIENCE: WHILE condition.

PROFESSOR: WHILE condition and what type is this condition?

AUDIENCE: [INAUDIBLE]

PROFESSOR: OK. And then I have--

AUDIENCE: [INAUDIBLE]

PROFESSOR: And then a block of code, indicated by indentation. OK. Next question is, what does this code do?

AUDIENCE: Executes the loop and returns Boolean values.

PROFESSOR: Right. That's a little bit too high level. Specifically, what does this loop do?

AUDIENCE: Prints the even numbers from 2 to 10.

PROFESSOR: Perfect. That's exactly the kind of comment we're looking for. When you're doing your problem sets or you're just coding in general, you want to have a comment that describes in an abstract way what's going on in the chunks of code.

You don't want to have a comment that says assigns 2 to a. Continues looping until a is 10. Comments like that aren't really helpful because you can get that by reading the code.

So that's exactly right. It's perfect. I don't have candy to pass out, though.

Next question is what's the decrementing function? The way to think of a decrementing function is that it's a statement that moves the loop closer to termination. In this case--

AUDIENCE: 10 minus a.

PROFESSOR: Yeah, 10 minus a or a plus equal 2. As we're incrementing a, we're getting closer and closer to 10 and that's going to cause us to kick out of the loop.

Next thing. Is a a good variable name? The answer is no, otherwise, I wouldn't have asked it. But a better variable name might be even_number. Because, instead of just a being an integer, now we know what it's supposed to represent in the code. And it becomes clearer when we go to execute it or run it.

For loops, same thing. Could someone walk me through the syntax for a FOR loop? And first question, does this loop do the same thing as a WHILE loop? If this is our WHILE loop, does this loop do the same thing?

AUDIENCE: Yes.

PROFESSOR: OK. Let's see. The WHILE loop printed out 4, 6,8,10 and 12; this is going to print out, 8 and 10. The reason is where we have the increment, right? If we had done this, they would print out both the same. A little aside. That was a bug I just caught.

Someone walk me through the syntax for a FOR loop.

AUDIENCE: [INAUDIBLE]

PROFESSOR: A FOR keyword and then what do we have here? A variable of some sort. Then we have in keyword and then we have this thing here. We're going to get to tuples in a

second, but this is a tuple literal. And for FOR loops, a FOR loop requires something that is enumerable, which means that we can take one element after the other and assign it to the variable i. FOR loops iterate over enumerable items.

Thank you. That's exactly what I was talking about. If I'm speaking too low, just jack your thumbs up.

This construction is inconvenient though. If we wanted to list all the numbers that we wanted to print-- 2 to 10 is not too hard. But let's say we wanted all the even numbers between 2 and 100. I don't want to have to write out all 50 of those.

So we get to the range function. You've seen this before. We just haven't ever explained it.

All the range function does is it takes between one and three parameters and returns a list of integers. So let's say that I pass it one parameter. I'm going to give it the integer 100.

This is going to give me nothing because I didn't print it out. This is going to give me integers 0 to 99. If I give it two parameters, what it's going to do is give me? The integers between 1 and 100.

In this case, it starts off at 1 and it gets to 99. One important point is that when you give it two parameters, the first one is inclusive and the second parameter is exclusive. Does everyone follow that?

Finally, the third form is with three parameters. I have a start, an end, start inclusive, end exclusive, and then a step. This will give me all the odd numbers between 1 and 100.

We can also go in reverse. If I want to go count down from 100 to 1, I can give it a negative step. If you want help on using the range function, there's a handy command called help, which you can type in the interactive prompt. This gives you the syntax for using range.

You can do this for any function in Python. It's going to tell you, here's the name.

Here's a parameter start. Here's a parameter stop. Here's a parameter step.

This little arrow here means this is what it returns, a list of integers. When you type help and you see one of these square brackets that means the perimeter is optional. A start is optional. Step is optional.

Stop, we always have a stop. We always know when we're stopping a range. That's why we can pass one, two, or three parameters. Is anyone confused on that?

AUDIENCE: If you only put stop, it will start with 0?

PROFESSOR: Right. If you only put one parameter, then it will start from 0. So the start is implicit. Are we good on this?

So, we can code our FOR loop like that, which for five numbers is not too inconvenient, or is convenient. But if we have 500 numbers, it makes it a lot easier to just change one end point than it is to type in 500 numbers, right? OK.

AUDIENCE: Can you create a range for floats? So let's say you want to do it by float size?

PROFESSOR: That's a good question. We should try it. Why don't we give it a shot?

The question was can we make a range of floats? Let's try 1.02 to 10.0. It'll give us an answer, right? But it also gives us a warning. It says that an integer argument is expected.

Let's say I do this. Will this work? What's it's going to do is it's going to truncate the floats.

It's going to truncate the start to 1 and it's going to truncate the end to 10. Then it'll just return the integers as required. So you can, but it doesn't work the way you think it would.

Moving on, you hit tuples in lecture this week. So we're done with the old review stuff. Can anyone tell me what a tuple is? It's a non-scalar data type that can hold many items. What does non-scalar mean?

AUDIENCE: It's multiple elements that you can search individually.

AUDIENCE: It's a field that you can hold only one value at a time.

PROFESSOR: You're both right. A scalar can hold only one element at a time. A non-scalar can hold more than one element at a time.

Tuples are actually the second scalar data type that we've seen. A string is the first. A string can have multiple characters.

Tuples are flexible. We can have tuples of numbers. This syntax, the parentheses with a set of elements separated by commas, this is the literal syntax for a tuple.

All we're saying is that my tuple of numbers has the approximation of pi, 2, 1, -100. We can have tuples of strings. We can have tuples of anything, but you'll get to that in a second.

AUDIENCE: Can you mix the data types inside of that?

PROFESSOR: Yes, we're going to get to that. The question was, can you mix data types? The answer is yes. We'll get to that in a second.

To access individual elements of a tuple, we do something called indexing. We specify an index by-- if I have tuple_of_numbers, my variable name, I have a left square bracket and a right square bracket, and I have an integer in between. This would give me the item in the tuple that exists at index 0. Tuples are indexed starting at 0 and in increments of 1. When I say tuple_of_numbers[0], what should this print out?

AUDIENCE: 3.14159.

PROFESSOR: Right. It will print out 3.14159. Now if I change this to 1, what will this print out?

AUDIENCE: 2.

PROFESSOR: Exactly. It doesn't matter what data type is contained in tuple. If it's string, it'll just print out whatever's here, which is 'what'.

You can also use negative indices. Negative indices tell Python that I want to go to one past the end of the tuple. In this case, tuple_of_ strings, I'm out here somewhere, and then walk back 1, or however many the integer is. I have here tuple_of_strings minus 1. Python's going to go to somewhere around here and then walk back one and give me 'name'.

Now what if I do minus 3? What's that going to print out? Is. Let's see who is paying attention. What's this going to do?

AUDIENCE: [INAUDIBLE]

AUDIENCE: My name, or name, rather?

PROFESSOR: Remember we index tuples at 0, right?

AUDIENCE: That's going to give an error.

PROFESSOR: Exactly. It's going to tell us index out of range. We have four elements in tuple strings 0, 1, 2, 3, oops, we're off. You can get outside of a tuple and get an error. To avoid that-- sorry?

AUDIENCE: The thing is when you said for tuple_of_numbers, you said 1 and 3.141, number, which would technically be 0 in this case, wouldn't it?

PROFESSOR: No, I switched it.

AUDIENCE: You switched it.

PROFESSOR: Yeah. Just to make you feel better--

AUDIENCE: I see it now.

PROFESSOR: You got it? OK. It's always possible that I made an error.

In your code, in order to avoid that, you can check yourself by getting the length of tuple_of_numbers. So there's a function len. It's going to tell us that

tuple_of_numbers has 6 elements. We can count them: 1, 2, 3, 4, 5, 6. What's the last index of this tuple? 5, right?

Back to your question, tuples can hold different data types. They can hold data types that are different from each other. I'm wording that improperly.

So, here we have a float and three strings. This is a heterogeneous data structure. A homogeneous data structure would be one that you would say only hold ints only holds floats.

But tuples are very flexible. So, we have a float here and some strings. Then you could also have tuples that contain other tuples. In this case, how many elements does tuple_of_tuples have?

AUDIENCE: 3.

PROFESSOR: 3. The first element is a tuple, the second element is 'got', and the third element is 'real'. What should this print out?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. I had to sanitize that quote.

AUDIENCE: Why does it print out the strings?

PROFESSOR: Python's behavior, when it sees a tuple data type and you call it with the print statement, is it represents as a string a tuple the way you would literally write out a tuple. 'Stuff, just' is a tuple with the two strings 'stuff' and 'just'. Python is going to print out the literal representation of that tuple.

Another way to try and explain it, let's say that I print out the entire tuple. What should this look like? Well, It's going to be just the literal representation of the tuple. What does it mean for the data type to be immutable?

AUDIENCE: You can't change it. You can't add another element to it?

PROFESSOR: You can't change it. If I try to change the first element of tuple_of_numbers, it's

going to tell me it doesn't support-- or Python's going to tell me it doesn't support item assignment. You'll see this error.

Tuples support something called slicing, which means that if I have a tuple_of_numbers and I give it a-- where I normally put just a single integer for an index, if I give it a start index and an end index, separated by this colon, Python's going to get the item at index 1 to whatever this end element is minus 1. It helps if I just print it out. To make it a little bit easier to follow. I've said I've told Python that I want a slice out of this tuple from 1 to 3.

What Python's going to go do is look into tuple_of_numbers. This is element 0. This is index 1; it's going to pull in 2.

This is index 2; i it's going to pull in 1. Then 3-- in Python, we go 1 past the end of the range that we want. It's going to return a tuple of 2, 1. This is slicing a tuple.

Anyone confused by that? No? Wow.

There are many different ways that we can slice a tuple. We can have an implicit start. If you see this, where there's no number before the colon, that tells Python start at index 0.

Then, in complimentary fashion, if you see this, where you have an integer on the left side and nothing on the right side, it tells Python go from index 1 all the way to the end of the tuple. What happens is I do this? What's that telling Python to do?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah. It looks redundant but it will become important to us when we get to lists and something called aliasing. What this does is it tells Python to take a slice that is the entire tuple. Then when we do these slices, we can also use negative indices.

This is going to tell Python to go from 0 to the last element minus 1. It'll look like that. Everyone following? We good?

AUDIENCE: It was minuses from the end--

PROFESSOR: Yeah.

AUDIENCE: --of the tuple?

PROFESSOR: Okay. I wonder if this would work? I don't know.

No, it has unexpected behavior. I don't know why. I'll have to look that up.

Tuples are also-- we already said earlier that they were enumerable items, right? So we can use a FOR loop with them. If I want to print out all the numbers in tuple_of_numbers, I have my FOR loop, my variable, in, and then my enumerable object. OK?

I have a question now. Who thinks this will work? We know that tuples are immutable.

AUDIENCE: Yeah. But you have to change the inside.

PROFESSOR: Right. It's going to work. I still have print up here.

What I'm doing is I'm taking tuple_of_numbers and I'm printing before. Then what I'm doing here is I'm telling Python to take this tuple and this tuple and add them together and reassign it back to tuple_of_numbers. So, It looks like I'm modifying the tuple. But in reality, what's happening is I'm creating a new tuple.

Let's say I have ton here. Ton is short for tuple_of_numers. Originally, it's telling Python that some chunk of memory has a tuple of 1, 2, 3, 4.

When I say a statement like ton equal ton plus another tuple, what it's telling Python to do is create another chunk of memory that includes ton, whatever's in ton, and the other tuple. Then this assignment statement tells Python that ton now points to this new object. You had a question?

AUDIENCE: Does Python have a garbage collector?

PROFESSOR: It's an advanced question. The question was, does Python have a garbage collector

to discard this memory that's no longer being used. The answer is yes.

If you don't know what a garbage collector is, don't worry about it. You don't need to. But to answer your question, yes, it does. I don't want to get too far into it.

Does everyone follow that? That also is going to be important when we get to lists and aliasing, that type of object creation modification. We won't get to that for a while. Everyone's good with this? I can move on? All right.

Python has what some might consider a wart, when it comes to tuples. That is when you want to create a tuple with a single element in it. People just starting out with Python would sometimes mistake the tuple literal of a single element to be this: So, parenthesis with a single integer.

The problem is that parentheses are the grouping operatorS in Python, but they're also used for making tuples. They serve a dual purpose. What Python will say is, oh, I've got a number between two parentheses. Well, this person really wants this integer to have high precedence.

We're going to make an integer 50 and assign it to oopsie. But it's not what we want. We want a tuple with 50 as one element.

The way you do that is you have a lone comma after the first element. If we run this and we look at what it prints out, you see that oopsie has the integer 50, which is not what we want. We want what's in onesie. Anyone confused by that? We're zooming along here.

Strings, they're actually a lot like tuples. They're immutable. You can't change them. They're non-scalar. They have multiple characters within them.

If I printed out-- everyone's seen this-- I can get to individual characters. If I want to get to the first character in name, I use 0. If I want to get to the second character, I can use 1. Then of course they're immutable, so that's going to tell me that I can't do that.

They also support iteration. So, if I want to print out all the letters in name, one on

each line, I can do this. Not too useful, but it works. You looked confused.

AUDIENCE: How did that happen?

PROFESSOR: How did that happen? So, name is a string. Follow that? String is just a bunch of characters. It is enumerable, meaning that we can go one character at a time through the string. That's what the FOR loop does.

AUDIENCE: Do spaces count as characters?

PROFESSOR: Good. The question was do spaces count as characters. The answer is yes, they do. If I write spaces there and I run this again, as I iterate through the string, I get a space where I'm supposed to.

Is everyone good with this? OK. Like tuples, you can take slices. That's going to give me 'it'.

Strings also have many functions. This is an incomplete list of functions that you can use on string objects. I can make everything uppercase, lowercase.

I can also find characters. What find does is it finds the index of the left-most character. If I want to find i, it returns 1, because i is at index 1 in the string.

I could also do something like this. I can find an entire string. 'tch', the substring, starts at index 2.

AUDIENCE: What does it return if it can't find it?

PROFESSOR: If it doesn't find it-- let's put in garbage-- you get negative 1.

We can also call a replace function. If I want to replace m with p-- it doesn't make sense anymore-- I can do that.

The question is, how would I use this to change the string? Think back to how we modified the tuple. We created a new object and then we assigned it to a variable of the same name. If I know that replace is going to return a string with m replaced, I can do that. Does everyone follow that?

AUDIENCE: Is the version that that captures your string function?

PROFESSOR: That's a good question. That also gives me a perfect opportunity to demonstrate another command that you'll find helpful. If you're working with an object like string, you can use a command in the interactive editor called dir. If you type dir str, it's going to return all of the symbols that exist within the str object. You can also type help str, and it will give you a nicer version of this.

AUDIENCE: In this case, the first character indicates what you're trying to replace and the second is what you're replacing it with?

PROFESSOR: Right. We can do help str replace, and it'll tell us.

AUDIENCE: OK. So it did one. And then the count in this case would be if you had multiple instances?

PROFESSOR: Yes.

AUDIENCE: OK. So like if you had your name a couple times, you could replace it a few times.

PROFESSOR: Right. If I were particularly narcissistic that day, I could replace it multiple times. So let's demonstrate that because some people might be confused.

Let's say, I want to replace t. I'm going to replace it with r. Whoops. Well, why didn't it replace these to t's here?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Right. In strings, you differentiate between lowercase and uppercase. If I wanted to get the behavior that I was hoping for, I could do it this way.

Now I have something that's not even remotely looking like English. Or, I could do something like this. I'm going to make everything lowercase, and then replace the t.

AUDIENCE: So that's pretty exact that you have to be, lower--

PROFESSOR: Remember what I said first recitation that computers will do exactly what you tell them to do and nothing more or less. There we go.

AUDIENCE: It's good though. It figured out [INAUDIBLE]

PROFESSOR: Python is a very nice language. It's one of my favorite languages right now. Because, it's not as fussy as some other languages, like MATLAB or C or C++ or TCL. Anyone of those.

Is everyone good on strings? I can move on? Just remember, if you need help on any of these commands, just remember you have the help command at your disposal.

AUDIENCE: It'll automatically default to do everything, unless you tell it something to do like two of the letters.

PROFESSOR: Yes. We didn't demonstrate that. Demonstrations are worth a thousand words. If I wanted to, for whatever reason, only replace the first two t's, I could tell it to only replace two t's within the string. It leaves the other t alone.

AUDIENCE: How would you pick out the last field?

PROFESSOR: That's a good question. I don't have a ready answer for you. You see these functions with r in front of them? The string object often has functions that will start from the right side as opposed to the left side.

The find command I showed you has an rfind command as well. This guy right here. I'm pointing on if there is the analog to replace. Can I get back to you? I don't want to spend all our time searching for this and then fail spectacularly.

I'll get back to you on that. Is it possible? Yeah. Is there a one-liner function?

I can't tell you at the moment. But again, that's why you have dir and help. You can find this out on your own.

Next thing, BREAK. We're done with strings. We're done with tuples. We're still

working on loops. Everyone's seen this function before, right?

This is our find-the-cube-root-of-a-perfect-cube function. If I give it 27, and I tell Python to stop printing my name, that's what this does. As a toy example to illustrate the BREAK statement-- first of all, can someone tell me in English what the BREAK statement does?

AUDIENCE: Would it just stop the program.

PROFESSOR: Well, It doesn't stop the program, but it kicks you out of the loop. We can rewrite the cube root program to work like this. Instead of our stopping criteria being answer cubed less than the absolute value of the number, we're just going to tell the FOR loop to go from a range 0 to the absolute value of the input plus 1. Why do we have the plus 1 there?

AUDIENCE: So it goes to the absolute value of x.

PROFESSOR: Yeah. Because, otherwise, it would go to 1 before the absolute value of x. Then we break out of this loop, if answer cubed is equal to absolute value of x. As soon as we see that it's equal to x, we're going to call BREAK. That's going to immediately kick us out to here, without executing any more of the FOR loop.

Do people understand that? Are people good with that?

AUDIENCE: Does it break you of the innermost loop? or [INAUDIBLE]

PROFESSOR: Yes. Good question. Does it break you out of the innermost loop or all the loops? The answer is the innermost loop. When he says innermost, what he's talking about are nested loops.

Let's say I have something like this. I'm creating on the fly now, so excuse my inability to type. What is this going to do?

This is an example of a nested loop. We have an outer loop here, then we have an inner loop here. All the outer loop is doing is it's going from the integers 0, 1, 2, 3 to 9.

And then we have an inner loop, which looks like it should go from the integers 10 to 100. But we have the statement in here if j mod 2-- that's what the percent sign is, modulus-- is equal to 0. What we're saying is if j is evenly divisible by 2, we're going to break.

Now, the question is-- this is obviously going to break when j is 10. This loop is going to execute once. The question is does it print out only one set of i,j values, or does it print out 10?

I'm getting to the answer to this question. If BREAK statement breaks out of all the loops, then we would only see one printout of i and j. But if it only breaks out of this inner loop, then we should see 10, followed by 'here'. So it breaks out of the inner loop. Long answer, but demonstration.

Is anyone confused by that? Is so, anyone too shy to admit it? There's office hours. Or, you can talk to me afterwards.

PROFESSOR: Now we're going to get to functions, which are triple underlined and circled over here because they're extraordinarily important for you to understand. Can someone wing it and tell me what they think a function is?

AUDIENCE: It's a snippet of code that takes some input, does something to it, and returns some output.

PROFESSOR: Perfect. It's a bit of code. It's named, so you can refer to it. It takes input, does something with it, and returns something, some value.

The way that we define a function in Python is like so. Let's say I have a function cube. It consists of a few parts.

We've got the DEF keyword. We've got a name, cube. The name should be meaningful for functions.

Like variables, the name should mean something So, this would be a bad name for this function. You want it to be meaningful.

It has a set of parameters. In this case, it only has one parameter. This is what we pass to the function when we call it. We'll talk about that in another second.

It has this string. This is called a doc string. It's the specification for this function.

When you write functions, it's good to have this string here. What it allows you to do is describe what the function does, what it expects for input, and what it gives you as output. In this case, it's very simple.

This is the body of the function. Again we denote the block by indenting. All it does is it takes a number, which is passed into it and raises it to the third power. This RETURN statement tells Python to send that back to whoever called the function.

As an example if this working, let's look at this line of code. What it's doing is-- we've seen the print statement, so we know what it does. It's going to name cube and it's going to call it.

Python knows it's calling it because it's got the name of the function with the input parameters. So, 3 is being passed to cube to be, well, cubed. What's going on here is that Python is breaking out of its normal flow of execution, sending 3, calling it number in the function, and then raising it to the third power.

If we run this, we see 27. We can pass it any number that we want. Is anyone confused by this?

AUDIENCE: The things you're running between quotation marks, doesn't that kind of--

PROFESSOR: They don't do anything. The question was, this string here between the quotation marks, this doesn't do anything. No, this is a comment. This is so that you can tell yourself, six months down the road, what you were thinking. Or so that you could tell another programmer what this function does. It's a way of documenting your code.

AUDIENCE: So cube in this case is something that is built to go around.

PROFESSOR: Yeah. This is just a toy example. I wanted to keep it simple. I'm illustrating concepts. It wouldn't be too hard just to do this.

I'm going to move on. Everyone's good with functions?

AUDIENCE: [INAUDIBLE] Can you say x equals number-- can you keep the number, and just keep the number, and just make it into a variable without the RETURN function. Do you have to use the RETURN function on that?

PROFESSOR: I was just getting to that. The question is, do you have to use a RETURN function. The answer is well, it depends on what you want to do.

So, let's take a look at a new function. Can someone tell me what this does? First, what does the function do? Or what is it supposed to do?

AUDIENCE: Takes the number and doubles it.

PROFESSOR: You got that by reading the writing on the wall, right? Really, it's too easy. In the body of the function, all it does is it creates a new variable, answer, and it assigns number times 2 to it.

What's going to print out here? Why don't we run it and see? That's not doing what we wanted.

When you don't have a RETURN statement, Python returns implicitly none to whoever calls the function. Ok. In this case, this function obviously doesn't have a RETURN statement Python says, OK. I'm just going to return none. Whatever work it did in the function is lost in this case.

To get the right functionality, we have to add a RETURN statement. And it works. Was that what you were getting at? OK. Anyone confused? Can I move on?

Functions have something called variable scope. I apologize for the punning. I was getting tired when I wrote this.

In this chunk of code, I define a global variable, all_hope. I also define a function that takes a parameter, variables. It says it steals all the variables, but I don't know how it does that in a computer. It doesn't return anything.

In the body of the function, I create another variable, called my_variable, and I assign it a string. I'm not actually returning anything right because I've said I'm not returning anything. All this is doing is it's just printing stuff out.

It's printing out what the parameter passed into it was. It's printing out variables. It's printing out all_hope, which is a global variable that we define up here. It's printing out my_variable, which is a local variable in the function.

Down here I'm defining a variable, old_meme_is_old. I'm calling the function. It does what we expect it to do.

What I want to illustrate, though is what happens if I do this? As you might expect, it's going to give me an error, right? The reason is that my_variable has local scope to this function, all _your_vars_are_belong_to_us.

Is anyone confused by this? No one's confused by this? OK.

Let's try something else. Let's not do that. If I run this code, we can tell from the function that it's taking one parameter. It's incrementing that parameter.

It's incrementing this global int that we've defined up here. It's returning the parameter that it's just incremented. Erase my corny humor.

The question now is, let's say that I have a variable y, and I give it a value of 10. I'm being completely arbitrary. If I call the variable, inc_it, on y, first of all, what's going to print?

If I print out the value of y, what's going to print? We want to run it. Uh-oh, I'm failing. I'm stepping on myself here. This is why you don't debug code on the fly.

If I have a global variable and I need to reference it, I use a global keyword. You should never have to do this. Ink_it is going to return 11, right? Because it's taken y, which is 10, and it's called x equal x plus 1. Then it's returned this x.

If I look at what y is, after I've run this, y is still 10. This is because when we've passed in x here and we've called x equal x plus 1, it's actually shadowing itself.

It's overriding what's in the local parameter. But it's not overriding the actual variable, y. We'll get more into this later on. The important thing to understand in this case is that the changes that you make to this parameter stay within the function. We good? All right.

I need to move pretty quickly now to gotchas. I guarantee you someone's going to make this mistake. Print is not RETURN.

When you call print print_is_not_return, it's going to call this function. It will print out this string, but it's not returning the string. What it's returning is none, because there's no return statement. That's where this none comes from.

RETURN is not print. So, if I say print this return value, it will print this; it will return a string. But if I just call this, it's not going to print it again. It's going to print nothing. You'll just have to make the mistake.

One thing to be careful of in Python-- remember I said everything is an object? Functions are no different. Functions are objects.

If you just reference the function's name, cube, which is not defined now, it's going to print something out that looks like that. This is what the object looks like to Python. In order to call it, you have to have the parentheses with the parameters.

Python's not going to complain. Some programming languages will complain. Python won't. It's possible when you're running your code, if you're trying to call a function and you forget the parentheses, and Python's just not complaining, it will merrily do what you tell it to do.

AUDIENCE: [INAUDIBLE]

PROFESSOR: What's that?

AUDIENCE: [INAUDIBLE]

PROFESSOR: I'm sorry, can you speak up?

AUDIENCE: Print q open parentheses [INAUDIBLE]

PROFESSOR: OK. Is x defined?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Yeah.