PROFESSOR: So the way that I envision recitations going is I'll start off kind of reviewing some of the points, some of the high points from lectures, might go into detail on some of the more important topics. And this is also a time for you to ask questions. There's only 25-30 of you here, so you should feel a little bit more comfortable asking questions. And there are no stupid questions here. For the most part, you're all beginners at programming. Am I wrong? Has anyone programmed here before?

OK. So presumably, none of you have any experience with what we're talking about. And so if you ask a question, no matter how basic you think it is, it's not stupid. Because everyone's been there, including me and the professor when we started programing. And we've made, ourselves, really stupid mistakes too. So.

That said. We started out by talking about the purpose of the class, which is to teach you how to take problems, real world problems, break them down, abstract them, and divvy them up so that you can solve them with a computer. And when we talk about computer, we're talking about a very simple model where-- and this is the only time you'll see anything related to hardware in this class. When we talk about a computer, we're talking about something very simple with a CPU and memory and maybe some input and output. All right.

And when we talk about programs, these are sequences of instructions that are loaded into the computer's memory which can be divided up into little cells like this. All right. And what they might look like is-- This actually doesn't mean anything, but it's just, you know, for the sake of demonstration.

This is what a computer would see. As the CPU starts running a program, it looks at whatever's in this memory location and says, this is an instruction I can handle, so

I'm gonna go do this. So this might be like add two numbers together and produce a result. Something very simple, very basic. And as it moves along in a straight line fashion, it just executes this instruction, this instruction, this instruction, et cetera, et cetera.

The problem is is that while the computer is perfectly happy to look at this, this is gibberish to us. And most people, most sane people, if they're actually looking at computer code, can't decipher this. So this is where programming languages come in. We can move up a level. We can abstract away from this, and maybe say something like this stands for -- move a number into a register or something like that. You don't have to know what this means. All you have to know is that this is a little bit more readable than this. And it represents an instruction that the computer can understand.

The problem is that even at this level, this is still very atomic and very low level, and really you can't understand what's going on. So-- just so I can fill this in. We move into say, I'm gonna say x is equal to 1 and I'm gonna add x plus 5 and then multiply by 2. This is a lot easier for us to understand. And this is what we're talking about when we're talking about any sort of programming language that we'll be studying in this course. And we're studying Python.

So there are hundreds of languages that allow you to express these concepts in a form similar to that. And all it does is it allows us to talk to the computer in this language. So everyone good with that? Cool? All right.

So when we have these programming languages, they're put together in specific ways. So does everyone remember the term syntax? Can anyone tell me what it meant or means in terms of programming languages?

AUDIENCE: I could guess. It is the way that the string is structured?

PROFESSOR: Yeah. It's the way the parts of the language are put together. So let's say that I take a very simple statement. I'm taking a variable plus this variable. This is a valid piece of syntax for Python, for example, or, say, arithmetic, or a bunch of different

languages. So this is good. I can't write.

On the other hand, if we say variable variable plus, this isn't valid in Python, for example. It might be valid in another language, but the syntax of Python won't allow this. So this is bad syntax. So.

All right. So is everyone kind of understand what syntax is? All right. What about static semantics? Talked about that in class. Can anyone give me kind of a rough view of what that is? Can I choose someone? All right. When we talk about static semantics, we're talking about syntactically valid statements that mean something.

So let's say that I have variable a is equal to 5 and b is equal to 2. A statement that's syntactically correct and is also meaningful with respect to static semantics could be something like a divided by 2, right. Or a divided by b. They're valid statements syntactically. And we know that 5 divided by 2 or a divided by b, they're both numbers. They mean something.

On the other hand, let's say that I had this. I'm gonna say that this variable d is a string 'foo'. All right. This c divided by d where c is a number and d is a string, that's something that's not meaningful, right. What does a number divided by a string mean? Nothing. So that's what we're talking about with static semantics. So bad.

These two aspects of computer programs are pretty easy to check for a compiler interpreter because they're pretty explicit rules, right. The part that we spend most of our time in is the semantic part. And this is where the syntax checks out, the statements are meaningful in and of themselves, but the program as a whole, or the whole kind of recipe, doesn't work.

So as an example, so this, as we know is fine. That's correct. But let's say we have this. So a is 6, c is 0. I do that, what's 6 divided by 0? It's an error. But syntactically this is fine. And by static semantics, this is fine. So this is the type of thing we're talking about with the semantic aspects of a program. Does it work properly?

And before I jump into Python, one tip to keep in mind is that when you're writing programs and you're trying to do your problem sets, your program is very explicit.

It's not ambiguous. So a program will do what you tell it to do and no more and no less. And a statement doesn't mean two things. So like if I make the statement in English, I cannot say enough good things or recommend this person highly enough, if you write that out that can be considered a statement with two different meanings. One of them is not too complimentary. All right. So when you're writing your programs, if it's not doing what you think it's doing, one of the skills that you need to learn throughout the semester is to be able to read the code and try to follow along in your mind what's going on. It's not magic. OK.

So now we're at Python. We're past kind of the generic introductory scaffolding stuff. Now we're onto something that's actually gonna be meaningful for you for the next 12, 13, 14 weeks. All right.

So Python is a general purpose language. It's used for all sorts of things, web development, small device development, desktop programs, et cetera. It's an interpreted language, which means that if I have a program, blah, blah, blah, blah, then Python executes it for me straight away. Some languages, compiled languages, have to go through a compiler, and then you have to run them. All right. It's an extra step. The nice thing about having an interpreted language is that as you may changes to the code, you can instantly see what's going on with your program. So. See.

It's got a very simple syntax. And it's also very widely used and has been getting-- more and more people have started using it, oh, for the past, say, 15 years. So. 10 years ago when I first heard about it, it was kind of like this little cute language. Now it's kind of blossomed into this wonderful language that everyone uses and loves.

So. Programs in Python, and in all languages, are sequences of expressions. So this is getting back to syntax. And these expressions, they are composed of operands and operators and functions. So let's say, for example, this is an expression. This is a variable name. This is the assignment operator, and this is a string literal. When I say operand, what I'm referring to are things in the language. That's what a literal and a variable are. And then if I say operator, this is something

that does something to things.

So it's-- this is an example of what we call an assignment operator. What it does is it says, take what I'm referring to my right hand side here, create a name called myvar, and say that every time that I reference myvar, I get this value. We'll talk more about that later on.

Um. Let's see. Where am I going? So one important thing to know about Python is that everything-- so these things-- are objects. For now, you don't have to get too familiar with what an object is just-- if you know anything, especially for the first quiz, this is something to know.

These objects, they have types, right. So some types that we have in Python are ints. What are examples of ints or integers? What?

AUDIENCE: 7.

PROFESSOR: Yeah. Just rattle a few off. So 7, 0, negative 1, 2, et cetera. Now there's another number type, floats, right. So this is what we normally think of as real numbers. But when you're talking about real numbers on the computer, they are kind of dicey to deal with-- and we'll actually cover that later on in the semester when we talk about kind of the inexactness of these. But for now, just know that these are numbers with like decimal points. All right. OK. Where am I going next?

AUDIENCE: [INAUDIBLE]?

PROFESSOR: Actually, syntactically, yes. So one thing that you might encounter when you're dealing with your programs-- this is kind of off on a tangent-- is when you say assign a number to a variable, when you have a literal like this number here, 0, Python infers what type you're talking about. So in this case, it's gonna create a number, and it's gonna call it a type integer. All right. If it sees this, it's gonna create this variable, and it's gonna have a type of float.

Now, it might seem like something minor to you right now, but when you start doing some of the math on problem set one, if you divide by an integer, you might run into

problems, right, because in computer-land 5 integer divided by 2 integer is equal to 2. Whereas in the real world, it's equal to that.

So just be aware when you're working with numbers that you need to be aware of their type, all right, especially when you're talking about the different operations you can do on them. So. So for these number types, you have addition, subtraction, multiplication, division. For integers, exponentiation, which is represented by two asterisks. And for integers, modulo. And then for floats plus, minus, same thing except no modulo. All right. So back to the list.

AUDIENCE: Can I ask a quick question?

PROFESSOR: Yes.

AUDIENCE: Could you explain why like maybe briefly 5 divided by 2 is 2? You said in computer-land, like what? Is it doing some sort of weird rounding? Or like why, why is that different than--?

PROFESSOR: This is because the type of the variable is an integer. And when you say something is an integer, you're talking about only these numbers, right. So 2 and a 1/2 as an integer doesn't exist. It only exists as a float. But when you have an operand like this that takes two numbers on both sides, if both numbers are integers, this produces a result that is an integer. So when you have these math operators, these produce a result, right. So that result has a type as well. Because I can take that result and I say c is equal to 5 and 1/2. C is now a variable in the language. That means that it has to have a type. And Python says that because I did this operation with two integers, I'm gonna give it a type as an integer. And the closest representation to 2 and 1/2 in integer terms is 2.

AUDIENCE: I thought it would round up to 3. I assumed that it would round up to [UNINTELLIGIBLE].

PROFESSOR: Oh, yeah. You're talking about a separate issue. It's not necessarily that it's closest, but it truncates.

AUDIENCE: OK.

PROFESSOR: So when it converts-- yeah, it just truncates the decimal.

AUDIENCE: What happens if you have 5 divided by 2-- [UNINTELLIGIBLE]?

PROFESSOR: So when you get into stuff like that, Python's gonna look at, see it's an integer and a float, and-- this is something that we're gonna have to actually test out when I turn on the computer. It should convert to a float, but it might not. So.

We'll test that out, actually. That's one of the nice things about Python is that if you have a question like that, you can test it out instantaneously. So remind me ints divided by float, and we'll demonstrate it. OK.

All right. So another data type, string. Anyone know what a string is or can tell me? Shout it out. Pantomime it.

AUDIENCE: Sequence of characters.

PROFESSOR: OK. So it's something like this. All right. And you can-- does anyone mind if I erase this? So in Python, you can specify strings a couple ways. One is with single quotes on the sides. And the other is with double quotes. This is useful, for example, if you need to embed quotes inside a string. So if I need to, say, have this as a quoted string, I would have single quotes on the outside and then double quotes on the inside, or vice versa. I could do double quotes and then single quotes. The point is that Python needs to know when and where the string starts and where the string ends. We good? All right.

AUDIENCE: [UNINTELLIGIBLE].

PROFESSOR: So if you put that? If you write this, Python's gonna call that a string. It's a string that contains the character two, though. All right. I guess you're asking because of the input with the problem set?

AUDIENCE: --divide that by a string, by an integer that would [UNINTELLIGIBLE].

PROFESSOR: So in your program, there's a point where you have to enter in numbers for problem set 1. Oh, you didn't start? Well there is a place where you have to enter numbers. So raw_input, for example, is gonna return a variable, and the type of this is gonna be a string. So if you need to use it as an integer or as a float, you can convert it. So now this is an integer, and you can do math with it. Or you can also say-- Does that make sense? That's something that'll be useful in your problem set. So just FYI. All right.

Another data type, Boolean. Does anyone know what a Boolean is?

AUDIENCE: [UNINTELLIGIBLE].

PROFESSOR: What's that?

AUDIENCE: It's like an if-then statement?

PROFESSOR: It has to do with if-then statements, but it's a variable. It's a type that only has two values, true or false. OK. And we'll talk about this, all of these, when I start talking about operations on them. Last one that you need to know of right now is the none type. The way to think about none is that it's sort of like dark matter. It's there. We know it's there. It holds a place, but we can't do anything with it. It's there just so that you know that there's a variable that should refer to something but that something doesn't exactly exist. OK. You'll see more of it. But it's nothing, but it's there.

And then later on in the semester, there's other data types. I mean, there's a lot of data types, but these are the big ones that you need to know right now. Other data types that we might look at are lists, tuples, and dictionaries. These are other major data types in Python. You don't need to know them right now 'cause Professor Guttag will go over them in lecture. But just keep them in your brain.

OK. So we've already talked about the operations that we can do on numbers. We can also do operations on strings. So we can do something called concatenation. So concatenation is just a big word for sticking two things together. So if I have s1 and s2 and I want to concatenate them to together, I use the plus operator. So the

other thing, too, that I want to point out is that we have a plus operator for ints. We got a plus operator for float. We've got one for strings. I don't think they work for bools, but they work for lists and tuples.

This is what's known as an overloaded operator. So it changes shape or changes its behavior depending on the data types of its operands. So that's why when you're dividing an ints by a float, some languages will convert it to a float for you. Other languages will convert to an ints. It's something that varies. That's why I'm not sure of my answer right now. So.

AUDIENCE: [INAUDIBLE]?

PROFESSOR: No. If your computer blows up in your face, then that's an issue. That's a separate issue.

All right. So some other operations that we have, and they relate to Booleans, are comparison operations. What these mean are these take two operands and compare them. So if I want to see if-- if I say a less than b and a is 2 and b is 3, then this is gonna return true. The value of this expression becomes true. And if I say a greater than b, obviously it's gonna be false, right. And all of these operators work in basically the same way. They take two operations, and they give you a Boolean value back. All right. Any questions on that?

AUDIENCE: So are Boolean values always true or false?

PROFESSOR: Always true or false. And I'm actually gonna get to that now. So Boolean values-- and this is kind of the last major one we're gonna talk about-- have three operators, AND, OR, and NOT. Actually, these should be lowercase. These, AND and OR, take two operands, NOT takes one. And what they do is if you have something like a is true, b is true, c is false, if you say like a AND b, then this entire expression is gonna return true, right. If I say a AND c, this will be false. AND returns true, if and only if both operands are true, false otherwise.

If you say a OR c, OR returns true if both operands or one operand is true. So if any of them are true or both are true. If they're both false, then it returns false. And

NOT, all this does is if I say NOT a, it'll return false. It reverses it.

Now we can combine these together. So these are very simple expressions, but we can also combine them. So we can say like a AND b OR c, right. So if both a amd b are true, then this becomes true. And then this entire expression becomes true if this part is true or this part is true. So you can build up pretty complicated expressions.

And then the way that they relate to these logical operators is, remember, these take numbers on either side and they produce Boolean values. So if I have-- I can say d is less than e. This is gonna give me a Boolean value, right, which I can then use the AND operator on and I can say-- So what this does is it says if d is less than e and e is less than f, then return true. This, by the way, would check to see that these numbers are in order, so 3, 4, 5, as opposed to 5, 4, 3.

So is everyone good on all that? Did I lose anyone? No questions? All right.

So the last couple of things and then I'm gonna turn on the computer and actually walk through some code with you and we'll be done for the day. So this is kind of the crash course in basic syntax for our basic types for Python. So there were three-- So what we have now is a way to create programs that run in a straight line, right. So can anyone give me kind of a synopsis of what a straight line program is?

AUDIENCE: Go down line by line.

PROFESSOR: What's that?

AUDIENCE: Everything one-- go down line by line.

PROFESSOR: Go down line by line, do everything once. All right. The problem is that this doesn't allow us to do anything, right. So we have branching. This is implemented by something called an IF statement. Now, the way you use an IF statement is-- all right.

This is the full version of the IF statement. It's saying, if this condition is true, I'm

gonna execute the code in this block. If this condition is true, if this condition is false and this condition is true, then I'm gonna execute the code here. And if none of those were true, then I execute what's here. You don't need to have an ELIF, and you don't need to have an ELSE. You can just have an IF or an IF and an ELSE. So the three versions of this branching--

Now when I, just by way of explanation, when I draw like a line like that, I'm talking about a block of code. Can anyone tell me how Python represents blocks of code?

AUDIENCE: Indents?

PROFESSOR: Yeah. So indentation, right. We kind of talked about that on Thursday. So blocks of code are chunks of code that belong kind of logically together. So what this saying is that if I execute this, all this block is gonna get executed, all this block is going to get executed, et cetera. So if I were to represent this pictorially, then this is kind of the main part of the program and then this is the if statement and the block. And then it goes back to whatever code is down here, right.

If I represent this pictorially, this would be the true part. So this part of code, this would be else. And then this part would be multiple excursions. So it would look a little bit like a tree. All right. So different branches for the different bits of code. Is everyone puzzled on that? Or anyone puzzled on that? OK.

And there was a last bit of flow control that we talked about. What was it? So if we want to do something multiple times?

AUDIENCE: Iterations or loops

PROFESSOR: Iterations or loops, right. So it's called a loop because it looks like a loop in the code, right. And there's two variants. There's a WHILE loop and there's a FOR loop. A FOR loop is when you want to iterate over a finite set of elements. So what this FOR loop does is it says I'm gonna take-- this is the range function, so we'll talk about this a little bit later. But all it does is it gives me all the numbers from 1 to 9. It goes 1 past. So. But you don't need to know it just yet.

What this is telling Python, though, is that we're gonna execute this block of code 9 times, all right. And on each iteration through this block of code, we're gonna set i to 1, and then set it to 2, set it to 3, 4, 5, 6, 7, 8, 9, 10. Anyone lost by that? We'll see an example of it pretty shortly.

And then the last one is a WHILE loop. A WHILE loop executes as long as a condition is true. So it's useful-- don't want to do that. It's useful when you're not necessarily iterating over this finite set of elements or you don't necessarily know how many times you need to execute a specific loop. You just know that you need to keep executing this code while something is true. And we'll see an example of that pretty quickly.

So I basically shotgun blasted a whole bunch of stuff at you. Is there anything that people want me to touch on before I pull down the screen and we start looking at code?

All right. So has everyone been able to get Python set up? All right. So you'll know that this is the editor window, right. So what we use to edit longer scripts in? And you know that this is the interactive prompt.

Before I forget, let's see what happens if I divide the integer 5 by the float 2. So Python does what you would hope it would do. It turns it into a float. If I do this, though, it gives me 2. All right. So it's an easy way to test.

So we've got two chunks of code here that I want to go over. One of them, you've already seen. It's the program that tries to find the cube root of a perfect cube. And why don't we just walk through it and read the code? All right.

So here's that raw_input function that I told you about. Now, it's gonna take a string and it's gonna print the string out on the screen and it's gonna say enter an integer. And, comment this out. So let's enter-- what's a perfect cube?

All right, the return type of raw_input is a string, right. So that int will convert to x. Now, here's an example of a loop, the WHILE loop. And because we don't have knowledge of what the user's gonna input when the program is run, a while loop is

an appropriate kind of control loop to use, right. Because we can test a condition where our guess, which is what's represented by ans, if we cube it, is it still less than x. All right. I explained that a little bit incorrectly.

What we're doing here is we're making a guess, and we're calling it ans right now, and we're gonna set it initially to zero. And then we're gonna enter this WHILE loop and we're gonna say we're gonna take answer and we're gonna multiply it three times. We're gonna cube it. And if the value of ans cubed is less than whatever input the user gave us, then we're gonna keep looping. And on each loop, we're gonna increase the value of our guess for ans, right. And if it turns out that x is a perfect cube, eventually, by just iterating through all the integers from 1 to whatever, whatever the cube root of x is, we will find the answer. And we'll know we find the answer because answer cubed is going be less than x. Now, can anyone tell me why we have abs here?

AUDIENCE: The absolute value.

PROFESSOR: Right. So let's say that the user entered in like negative 27. It still has a cube, right. It's still negative 3. But the way that we set up our loop, if we were to take this out, the program would just continue executing forever. Well, for actually a very, very long time, but-- the universe would die before this finished. So we have this absolute value here. Is anyone puzzled by this? Do I need to belabor the point? OK.

So eventually I've entered 27 and we can actually-- a good way to kind of check ourselves is to print out diagnostic input. So if guess is 1, that's not-- 1 cubed is obviously not 27. 2 cubed is not 27. 3 cubed is 27. And when we get to that point, we leave the loop. And now we're down in this bit of code. All right.

Now, because we're asking for cube roots, we need to check the condition for when the-- sorry, I'm kind of mixed up here. Sorry. So when we exit the WHILE loop, if we've had a number that's not a perfect cube, then we know that the value the answer stopped on is not going to equal the cube of it, by way of explanation. So my last guess 3 cubed is 27.

Let's run it again. Let's say that I have 20, right. So what happens is it gets to 4. So 4 cubed is 64, right. That's obviously not 28. And that's what this condition checks. And so it knows that if it gets to that point, it's not a perfect cube. Now, this ELIF statement here says, OK, so if this turns out to be true, that means that it wasn't a perfect cube.

But let's say that it was a perfect cube, this will be true, and Python will then start to look at this and say, well, let's look at this if condition, or actually-- I'm sorry. This condition will be false if it's a perfect cube, right. And Python will say, OK, look at the next part of the IF-ELIF statement and will say, well, was x less than 0? What it's doing here, it's checking whether or not we've entered a negative number or not. So if we enter negative 27, what it would do is it would to enter this branch, and then it would negate whatever answer it got. Because we found the answer for a positive perfect cube, right.

Did I break it? We are having technical difficulties. Oh, OK. I'm not sure what happened there. So anyway.

So we go about trying to find the cube root of 27, which is the absolute value of 27. And we, of course, find it. It's 3. But then because we've entered a negative number, Python's gonna enter this branch and negate whatever answer we got. Because we know that in order for it to be negative, then it would have to have been a negative number, right. Is anyone lost? Everyone good?

AUDIENCE: Do you need an exclamation mark for the--?

PROFESSOR: Oh, this one?

AUDIENCE: Yeah.

PROFESSOR: So this is one of the comparison operators. This stands for not equal to. So if I have-- do a little Python work here-- if I have a is equal to 5 and b is equal to 6, if I say a is double equal, that's gonna check and see if they're the same value. Obviously, they're not, so it's gonna return false. On the other hand, if I say not equal-- or bang equal sometimes we call-- this will return true. OK. Any other

questions?

AUDIENCE: The one that you did 28 for, shouldn't it not print the cube root of 28 is 4?

PROFESSOR: Oh. Yeah. So yeah. Should it be indented? Because if we indented here, right, then what's gonna happen? It's only gonna print out the cube root for negative numbers. So what-- what's that?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Well, again, if we put an ELSE statement, then it won't print out the cube root for negative numbers. What we could do, and this is kind of a hackish way, is write it like this. But that's kind of ugly, right, because you're repeating yourself. And one thing about computer programmers is that we are the laziest people on the face of the earth. Like, we'll spend 20 hours writing a program to do something in five minutes that we could have originally done in 10 minutes. It's just our nature. So anyway.

So we're repeating ourselves. And while I'm speaking, this is the solution I came up with. If I were to go back and rewrite this, I'd probably make it a little bit less convoluted. So maybe, let's try this.

So we're gonna check if we were successful, and then we're gonna check if x is less than 0. Now we're not repeating ourselves. And we catch all the cases. So let's make sure that my fix works because oftentimes when you write programs, you'll introduced bugs of your own.

So we've tested when it is a cube. Let's test it when it's not. OK. So there we fixed it, I think. One thing about computer programs is that this is a very simple case, but for longer, more complicated programs, it's almost impossible to get out all of the bugs. But in this case, pretty confident that we were successful, right. So does anyone else have any other questions on this bit, right?

All right. The last thing that I'm gonna go over is something called a fizzbuzz program. This is just a silly little program. This is the English specification. And this

is kind of a first instance of where we're gonna take English and kind of break it down into code, figure out how to break it up, chunk it up, and abstract it into something that actually works. So the problem is to write a program the prints the numbers from 1 to 100, but for multiples of three, print fizz instead of the number. And for multiples of 5, print buzz. And if there are multiples are both 3 and 5, print fizzbuzz.

So if it says numbers from 1 to 100, when you see something like this when you're trying to figure out how to write your programs, the first thing that should go off in your mind is, I probably need a for loop because I'm iterating over a set of numbers. So we happen to know the numbers 1 to 100. And here's a range function again. And we'll talk about range next week most likely.

And all we're gonna do is first we're gonna get the string value of this number. So remember how earlier you asked if, or one of the students asked, if we had a number in the string, if that was a number or not? So like this. That's what that STR function does. So if I have a variable a is equal to 1, I can say s is equal to STR a, and s is now gonna be that. All right.

And then what I'm gonna do here is I'm gonna check the integer value, so i, and see if it's evenly divisible by 3 or 5. And the way that I checked that is I use a modulo operator. That's what that percent sign is. What this operator does is it takes two integer values and it returns the remainder after you've divided the left integer by the right integer. So if I have 6 modulo 3, what's that gonna be? It should be 0, right, because you can divide 6 evenly by 3. The other hand, if I say 5 modulo 3, then that's going to be 2, right. Yeah, I had to think about it for a second.

OK. So that's all this does. And these are two expressions that return a Boolean value, right. Because this modulo operation is gonna return an integer, and I'm gonna use the equality operator to compare it to a number, another integer, 0, and that's gonna give me a Boolean value. This expression also gives me a Boolean value.

And then I'm gonna combine them into-- using the or operator for Boolean values--

into a larger expression. And then if this is true, then I know that I'm gonna have to at lease print fizz, buzz, or fizzbuzz. Because I'm not gonna print the number, right, because it's a multiple of 3 or 5.

And so all this code does is just figures out if it's evenly divisible by 3, then I know I print fizz. So I'm gonna concatenate my final string onto it. And then if it's evenly divisible by 5, then I know I need to print buzz. So then I'm gonna attach buzz onto my output. And then I'm just gonna print whatever I'm left over with.

So to see this in action because we are way out of time right now. So 1, 2, fizz, 4, buzz, 6, fizz, 7, 8. Then we have fizzbuzz for 15. So it seems to work. Everyone follow that? We good?

So I'm done for this recitation.