...
In the last lecture, we added looping variants.
We had for loops, we had while loops,
we had the notion of iteration.
We used that to start generating some examples of guess
and check algorithms.
In this lecture, we're going to go back to that idea
but extend it.
And we're going to add in another kind
of data type, a string.
We talked about it a little bit earlier on,
but we're going to use it to actually
look at ways to generate algorithms
that can start to do some more interesting kinds of things.
Just to review, I want to remind you
what we did inside of loops.
Here's an example of some code that's
going to compute a square root of something.
And it does it by running through a loop.
As we've done before, it's going to set up
an initial value, which it gets here by inputting something
that enters an integer.
And then it's going to run through a little while loop
where it does guess and check.
It keeps trying different versions of answers,
starting at 0, then 1, then 2, going on
until it sees whether that answer squared
is bigger than or equal to x.
Once it gets to that point, it checks
to see if the square is equal to x, in which case
it prints out that it's found the cube root.
And if not, it tells me that it's not a perfect cube.
And just depending on whether it was a positive or negative
thing, it gives me a little bit of additional information.
You can look through the code.
I'm not going to run it.
What I want you to see is that idea of guess and check.
I generate values for answer until I get to a point
where either I'm done and I have the answer
or I've gone too far.
And again, as before, I've got something
where I'm setting up the variable outside,
and I'm changing the variable inside
with a test that depends on it.
We're going to use that idea as we generalize,
but I want to remind you of what loops do.
And as before, I don't need to do that actual version
of answer equal answer plus 1, I could use plus equal
as a variation of it.
We also talked about strings earlier,
and I just want to remind you of those,
because we're going to use them in this lecture as well.
Think of strings as a sequence of characters, in fact,
case-sensitive characters for letters.
It could be lower case or upper case.
As with numbers, I can compare them.

I can say if two strings are equal while using
the double equal sign, I can use greater than or less
than using lexicographic order to decide
if two strings are greater than or less than each other.
Length can give me the number of things in the string
and, as we saw before, we can use square brackets
to do indexing into a string.
We can get the zeroth element.
Remember, it starts at 0, the first element,
the second element.
And if I have a string three long,
and I try and get something three or bigger,
it's going to complain, telling me
that I've tried to access beyond the end of the string.
We also saw slicing in strings.
This lets us go into a string of pull out pieces of the string.
And here we saw we can specify a start point, a stop
point, and how many steps to take as we do the accessing.
And that lets us pull out different elements
out of a string.
One unusual thing we haven't seen
before is if the step is negative 1,
we actually access the string in reverse order.
There it is right there.
Otherwise, if we say for example start with the element 3,
it says 0.
0, 1, 2, 3.
So I start here, and I stop just before the sixth one.
So 3, 4, 5.
I'm going to pull that out, which gives me
the piece that I wanted there.
One last thing about strings is that they are
what we call an immutable type.
You can't change them.
They can't be modified.
And to see that, let me do a little example.
I've shown it on the slide, but let me actually
type it in over here.

I'm going to define the name as to be the string hello.
If I print out s, it gives me back.
If I say what's the type of s, it
tells me that it's a string, which is great.
I can certainly index it by saying
give me the zeroth element of s.
But if I wanted to change the zeroth element of s,
for example, saying I want to have the zeroth element of s
be the character y.
It's going to complain.
I can't go in and change a piece of s.
We're going to see other structures where we can do
that that are called mutable.
Strings are not.
I can't say I want the zeroth element of s to be y.
I can change it, but I have to redefine it.
So I could say let s be the character y plus the rest of s

starting at 1.
And I could, in fact, say until I get to the length of s,
or I can just say 1 all the way onto the end.
And now if I look at s, it's yellow, misspelled for yellow.
Point is, I've now changed the definition for s.
So I can't change it individually,
I have to redefine s.
And that's why I say strings are immutable.
One of the things I would suggest to you
is if you're not sure what a command does, look it up or try
it out yourself.
That will give you a sense of the different things you can do
with strings as you move along.
And here, just to remind you, s was initially bound to hello.
When I do the last call, I'm changing that binding
and redefining it to be a changed
new version of a string.
The original string is still stuck in memory.
It's just floating around.
I can't access it anymore.
Finally, remember for loops, the quick recap.
We have that idea of having a loop variable that
iterates over a set of values.
And we can either do that with simple things
like the range shown here, which is going to go from 0 up to 4,
but I could also do it starting at 4, going up to 8.
Expressions inside the loop are going
to be executed for each value of the variable that I use.
And so range is a nice way to iterate over numbers,
but a loop, a for loop variable, can iterate over any set
of values, not just numbers.
And so for example, I could loop over strings.
That sounds kind of neat.
So here's a simple little piece of code.
I've defined s to be a string, the first set of letters
in the alphabet.
And I can now write a for loop that
says loop for the index over the range of the length of s.
Boy, that's a lot of words.
Let's say it a little bit better.
It says, I'm going to see how long is s,
and then I'm going to use range to generate the integers from 0
up to the length of s minus 1.
I'm going to call that index, and then
I could run through a loop that says
if the element of s at that point is either an i or a u,
I'm going to print out a little piece of information.
No big deal.
That just looks like a normal loop over integers.
But I could also write it this way.
I could say loop for character in s.
And that's kind of cool. s is something we call an iterable.
It says I can actually write a loop that
lets characters start with the first element of s,
and then the second element of s,
and then the third element of s, doing the same kind of test

as I move along.
And it will do this until it runs out
of characters in s, in which case it will exit from the loop
and move on to other kinds of things.
That second piece of code is much cleaner.
It's easier to see what I'm doing.
I'm looping for all the characters
in a string as opposed to saying I'm
looping for an index over all the indices into the string.
And that's one of the nice things about both strings
and especially about loops.
I can iterate over anything where
I can successively enumerate each
of the elements of that piece.
So now I've got strings as something
which I can use as parts of loops, and that's really nice.
Let me give you one last example, sort of a fun example.
I'm going to set up a little loop that is going to run over
basically a set of words.
And what is it going to do inside of here?
I've got a little while loop here
that says as long as-- sorry, rephrase that.
I'm going to input a word based on some input I give there.
And what it's going to do is it's
going to loop over all of the things inside of the word,
basically saying if the character in the word
is in a special set of characters,
I'm going to print out something based on that.
If it's not, I'm going to print out something different
based on that.
And then I'm going to increment i, and I'm going to keep going.
So, again, it's a while loop, but I'm using the character
inside there to loop over particular elements
of a string.
Let's see what this does.
So here's the code.
And if I load it into my environment,
says I'm going to cheer for you.
Give me a word.
So we'll give it the best word I know,
which is-- let me just get it over here-- which is MIT.
And it says how enthusiastic are you about MIT?
Well, I like MIT a lot, so I'm going to give it a 10
out of 10.
And it says-- I'm not going to repeat it all,
but give me an M, M. Give me an I,
I. Give me a T, T What does that spell?
And it prints it all out.
All right.
It's a lame example, but you know what?
I'm from MIT.
I like it.
What I want you to see is how I'm
looping over, especially here, in this place right in here.
I'm using the loop over a set of strings
in order to be able to decide what I'm going to do.

And why am I doing that?
Because if it's something that is a vowel,
I don't want to say give me a A. I want to say give me
an A. If it's something like an M that we pronounce that way,
I want to say give me an M, whereas in the other cases,
I want to say just give me a C, give me a D. OK.
Cute example.
Key point, I've now got two different loops going on here.
The outer one is walking through each
of the elements of the word.
That's this one right here.
And that's just doing the standard thing
where I've set an iteration variable outside right there.
I'm changing it inside the loop, but in the loop
I've got another one, which is right here where
I'm looping over all of the elements of a string.
I can do it either way, and that works out really well.
On that cheery note, we're going to move on to another topic.