Dictionaries

...
We've seen now four different ways
to collect things together into compound data structures,
non-scalar data structures.
We've seen strings, we've seen tuples, we've seen ranges,
and we've seen lists.
And I want to take a little bit of time
to talk about the common operations that
apply to all of them.
They're all ordered collections of elements.
They all have certain properties,
and then they have a few differences.
So I'm going to let seq, sequence, be
a name-- a variable, if you like--
but a name that describes any one of these.
And what do we know about all four of these data types?
I can get the ith element out by simply saying give me
the ith element of sequence by using the square braces right
there.
I can ask for the length of them.
I will get it back.
I can concatenate them using pluses
with one exception, which is ranges.
I cannot concatenate ranges.
But I can concatenate strings, tuples, lists.
I can actually repeat a sequence n times by using the star.
We saw that earlier with strings.
I haven't shown it to you with lists,
but it works exactly the same way.
If my sequence is a list and I say 3 times a sequence,
I'll get three versions of that sequence
all concatenated together-- again, not with ranges.
It's the one exception here.
I can slice into any of these things using
exactly the same format, especially that colon there.
I can test if something is in a sequence.
By simply asking is this element in the sequence,
we'll return true.
Same common form-- I can say is it not in the sequence.
And I can iterate over the sequence
by simply using for and in and walk down the sequence.
Why am I showing you this?
This is a nice property of this programming language,
that given these collections of elements,
exactly the same operation applies-- again,
with just the two exceptions here of concatenation where
ranges cannot be concatenated.
But it's really nice, because I don't
have to think about what operation am I
doing based on what type.
I'm overloading these operations so
that I let the type of the data help
Python decide which version to use to collect them off.
More importantly, intellectually, I
can think about the operation itself
without worrying about the details of which version of it

do I want to use.
I can say I want to concatenate these things together.
It doesn't matter if they're tuples, if they're lists,
if they're strings.
I can just go ahead and do them.
And so I could pull all of those together in a very nice way.
I've got one other piece of it that I want to think about,
which is which of these are mutable and which are not.
And as we've seen, only the list is mutable.
So how do I think about this?
Strings-- it's a concatenation of characters.
A tuple-- a concatenation of any kind of object.
A range-- a concatenation or sequence,
I should be saying, of integers and list
a sequence of any kind of object.
We've seen the examples of them, and we've
seen while three of them are immutable, one of them
has this nice property of being mutable.
But all of these behave the same way.
And that idea of collecting data together into a structure
is something we really want to be able to use.
And I'm going to show you an example of that next.

dheeraj nagpal