core elements of program

...
Welcome back to 600 part 1.
In the first lecture, we started talking
about computation, computational thinking.
And we began to introduce simple expressions
and very simple programs.
In this lecture, we're going to add a new type.
We're going to go back and look at branching as a key element
in order to make decisions in algorithms.
And then we're going to start introducing
the first class of algorithms that
are going to be really valuable, things
called iteration or iterative algorithms.
To do that, I want to just quickly remind you of things
that you saw last time and that you need to keep in mind.
First of all, variables, often called
bindings-- these were names that we had to which we
could associate values.
And we saw they were created by using a name followed
by an equal sign followed by an expression that had a value.
Why did we want the name?
It could be descriptive.
It tells us what that value is.
It gives us something that helps us think about how
we're going to use that.
It let's us reread code more easily,
and it certainly can be meaningful
because it gives us a sense of how we want to use computations
in more complicated ways.
One of the things to remind you is there are certain words that
cannot be used as variables.
These are things called keywords,
things like int or float.
They describe a type that cannot be used as a name
for a variable.
There's a set of these that we'll
talk about as we go through.
And you'll notice in Python when you type one of these keywords
in, it will show up in a different color, which
is a hint that it's a keyword and not something you should
be trying to bind as a name for a value.
The value could be any legal expression
that returns a value.
And that's going to store information
that we're going to use.
And finally we saw that we could update the value associated
with a name by reassigning it using another assignment
statement.
The way we did it, just to remind you,
was we compute the right hand side of the expression.
For example here, the value associated with two
is just the int 2.
We then store it-- sometimes we refer
to that as bind it-- to the left hand side, which
is the variable name.
That creates a binding off in the machine

of x associated with 2.
But of course we can also replace it with a new value.
And here, again, if you look at the second expression,
remember we first evaluate that expression.
The value of x right now is 2, so 2 times 2 is 4.
That associates the value 4 with the variable x,
which says when I get around to then evaluating x plus 1,
this is going to be 5, which is the value associated with y.
Equals sign does the assignment.
It creates those bindings for us.
And as I just said, we're going to always compute the value
first and then override the name if that value is actually used,
or rather if that name is used as part of the expression
to get the value.
You might think this is pretty straightforward,
but we need to be careful about the order in which things
are done.
Suppose I got two variables named x and y,
and I want to swap their values.
The little sequence of expressions
here looks like it should do the right thing, right?
Wrong.
Let's think about it.
The first expression assigns x to the value 1.
The second expression assigns y to the value 2.
The third expression gets the value of x, which is 1,
and assigns it to y.
So I have x is 1.
I have y is 2.
I look up the value of x.
I now assign it to y.
So the 2 was replaced by 1.
Now when I do this expression, it says what's the value of y?
It's 1.
Assign it to x.
And I replace the 1 with a 1.
So it didn't swap them because there's
a sequence to this operation.
It doesn't do what you think it was going to do.
There's a better way to do it in fact, the right way to do it,
which is to temporarily assign the value of y
to the variable temp so that I can overwrite the value of y,
and I still got that value around to restore into x.
So we need to think carefully about not only what
do I compute as a value and what do I give as a name,
but the order in which I do it.
Because if I lose a binding for a name, I can't get it back.
But that reminds us then of how to do assignments,
how to do bindings, and how to do
the actual sequence of operations
to create bindings associated with variables.