...
Welcome to 600 part 1.
We're delighted you're going to join us
for the next several weeks as we explore
interesting issues around computational thinking
and programming.
Now, what are we going to do in this course?
What do we want you to take away?
At the end of this course, what is
it we'd like you to have in your armamentarium
of great problem-solving tools?
We're certainly going to teach you about programming.
We'll teach it in a particular language called Python.
But more importantly, we want you
to start learning how to think computationally,
to think algorithmically, to think like a computer
scientist.
And what does that mean?
It means we'd like you to think about when
given a new challenge how can I get the computer
to solve this for me?
How can I describe the stages I want
to use to get this done in such a manner
that I don't have to do it.
I can get the computer to do it.
That's the notion of computational thinking,
of algorithmic thinking, and that's
what we're going to try and teach you about in this course.
Now, that means you really want the computer
to do the work for you.
It's going to be your servant, and that
means you need to think about how do you get it to do
the things you want it to do.
To do that, we're going to cover a range of topics,
and we'll see all of these over the next several weeks.
We want the computer to compute something for us,
infer some new knowledge for us.
That means we have to think about
how do we represent that knowledge,
and we'll do that with particular things
inside the machine called data structures.
We want it to infer a new information
or define information, and we're going
to see there are standard tools for making that happen.
Things called iteration and recursion.
And we'll come back to those over the next several lectures.
A big part of what we want to do inside the computer
is to have it be able to deal with things in a manner
that we can see and understand, and that's says
we're going to use the notion of abstraction to capture elements
and then treat them as if they were primitives and reuse them.
And that leads naturally to the idea of modularization,
creating modules, tokens, elements
that we can stitch together to come up
with solutions to problems in interesting ways.
Once we started learning how to build algorithms to think

algorithmically, we're going to see
that there are standard classes of algorithms,
and we're going to use those for common parlance
like searching and sorting and we're
going to see as well that different algorithms have
different costs.
And we want to see how to use that to reason
about the expense of doing something and better
ways of finding a solution to different problems.
So here's our roadmap.
These are the things that we're going
to deal with over the next several weeks as we talk about
and get you engaged in computational thinking.

If we're going to get the computer to do this for us
though, we could start by asking so what does it really do?
Boy, that sounds like a dumb question, right?
Of course, computers do all sorts
of amazing and awesome things.
They can play Go, they can find things in the World Wide Web,
they can do all sorts of wonderful, marvelous things.
But fundamentally, a computer really only does two things.
It performs calculations.
Well, duh.
But in this case, the calculations
are actually very simple things.
Turns out they can do them amazingly fast.
But all they really do, they perform calculations,
and they remember things.
Early computers didn't have much of this power.
Modern computers have a lot.
But those are really the basis of them--
perform a lot of calculations really
quickly and remember results.
Now you could ask, how fast is it really in terms
of performing calculations?
The machine you're using can probably
do about a billion calculations a second.
And just to put that in context, if I had a lamp sitting
on my desk here-- about a foot above--
and I hit the switch, by the time light went from the bulb
to the table, your computer's performed two operations.
That's amazing.
It's really fast, and it sounds like that's
going to let a computer do almost anything.
How about remembering things?
Depends on the size of your computer.
You probably have a few gigabytes of memory in there.
A big computer or something on the cloud
might actually have hundreds of gigabytes of storage.
What does that say in terms of what it can hold?
Well, if you took the standard novel
and you put it inside a machine, a typical machine
could hold about 1 and 1/2 million books
of a standard size.
So if you're going to start reading those great classics,

now it's going to take you a while
before you get through all the things that
are stored on your machine.
So sounds like computers are amazingly good,
even though they only do simple calculations
and they remember results.
Hold that thought, because we're going to come back to it.
Because we can also ask what kinds of calculations
does the computer actually do?
Every computer comes with a set of built-in operations.
These are typically primitive arithmetic operations--
multiplication, addition, division-- and simple logic
operations, comparing true and false values in order
to make decisions with that.
If that's all we had, that's going to be a real pain.
And so what we want to do through this course
is figure out how to define new calculations,
new operations, things we create and give to the computer
so that it can abstract them, encapsulate them, and treat
them as if they're primitives.
But to start with, a computer simply
performs a lot of those calculations.
So simple primitive calculations very quickly.
Is that enough?
It might be.
If that's the case, we really don't
have to do a lot in terms of computation.
And I want to give you a couple of examples to show you
why even with the speed of modern computers,
you need to be able to think carefully, cleverly,
algorithmically.
Here are two obvious examples of things you might like to do.
You want to find a piece of information
on the web, something you do every day with a search engine.
You might want to play chess or have your computer play
chess for you.
Suppose you want to search the web.
How much could you do if you just
were using simple calculations?
Well, here's a little computation
I did before I came in to capture this lecture.
There are about 45 billion pages right now
on the World Wide Web.
On average, there are about 100 words on a page.
And for sake of argument, let's assume
if we want to find a word on a page,
it's going to take us about 10 operations
to try and find out whether that word is on that page or not.
We'll see later on how he got it down to about 10 operations.
That says if I'm going to just brute force
try and search everything on the web
to see if I can find the thing I'm looking for, it's only
going to take me about 5.2 days to find something.
You probably don't want to wait that long.
So even with a very fast machine using
these simple calculations, it's not going to be enough.

How about playing chess?
An expert will tell you there about,
on average, 35 moves for every setting on the chessboard
until you get to the endgame.
Suppose you want to look ahead six moves
in order to try to decide what you want to do in order
to beat your opponent.
That says you've got about 1.8 billion boards
that you need to check.
And if it's going to take you, for example,
100 operations for every choice, it's
going to take you about 30 minutes to decide each move.
Probably too slow.
And this is simply a way of saying
that even with fast computers, we need cleverness,
we need algorithmic thinking to take those simple computations
and turn them into something more powerful.
And that's as good algorithm design is going to be crucial
and it's one of the skills you're
going to learn throughout this course.
What about storage?
For lots of storage in the machine.
Why don't I just compute everything once, store it away,
and then just look it up.
So let's go back to chess.
Imagine I just want to look at all the possible chess games
and store them away so that when I'm in any move,
I'll just know what I want to do in order
to get to a winning position.
Well, experts would suggest that there
are something on the order of 10 to the 123 different
possible chess games.
That's a really big number.
And in fact, there are only about 10
to the 80th atoms in the observable universe.
So there's no way that we can store all of that information
away.
And again, it comes back to saying we can't just
use brute force or pre-compute.
We need to be clever about how we come up with solutions.

Even with that, we're going to ask
are there going to be limits to computation,
even if we can build clever algorithms?
And in fact, one can suggest that there are still
some limitations to what a computer can do.
Some problems are still, at least at the moment, too
complex, even with clever algorithms
to come up with solutions fast enough.
I'd love to know what the weather's going to be right
in my neighborhood every morning before I
get in my car to come to work.
I just don't have enough data and enough compute power
to be able to model at that level of scale.
Maybe eventually, but not yet.
In some cases, the fact that some things

are too hard to compute actually works in our favor.
And encryption schemes are an example of that.
Things that you want to store on a computer
encoded so nobody can break them rely on encoding or encryption
schemes that in turn, rely on the fact
that some problems are simply too complex to be solved
by a computer.
And in some cases, even if the computers get faster,
it's still not going to be possible to solve them.
Some problems are just fundamentally
impossible to compute.
And the classic one from computer sciences
called the Turing halting problem
and it simply says if I want to write a piece of code,
a program that could take as input any other program
and tell me whether it will always work,
whether it will always stop with an answer,
it turns out you simply can't compute that in all cases.
So there are going to be limits to computation.
Not to worry.
It's going to be a lot of things we can do,
and that's what we're going to do throughout this course.