PROFESSOR: Now today's lecture. First I want to talk a little bit about what computer scientists do. This whole course has been about computer science. And I hope that at least for many of you, it might have persuaded you that there's an interesting career out there for you as a computer scientist. And then I'll finish up with an overview of what I think we covered this term.

So what do computer scientists do? Well, they look a lot like that as you all know. In fact, it's amazing. There's almost nothing that computer scientists don't do. So here I've just collected a few of the different pictures related to what computer scientists in EECS are doing these days, things like working on the software that keeps airplanes from falling out of the skies. Quite a few people working in the movie industry these days doing animations in 3D and that sort of thing. These days, computer scientists don't get paid as much as the actors. But they get paid a lot to produce movies.

Robots, of course. A lot of medical work, particularly medical imaging. This is a picture where someone has written software to try and identify the tumors in the human brain, and then will draw little lines telling the surgeon where to cut so as to do the least damage and the most good. A lot of work in genetics. And of course, core things like making the internet actually not crash.

Fundamentally, what computer scientists do is they think computationally. And I hope that you've understood that that's been the theme of 6.00, is how to formulate problems and think computationally. This will be, I think, the most important skill used by certainly scientists and engineers but, in fact, everybody by the middle of the 21st century or sooner. People will just do everything computationally. And just

as once upon a time maybe it's still important everyone learn to read and write and do a little arithmetic, people will think that computation is exactly as important and that you'll have to know how to formulate problems computationally to survive in society.

The process, and this is what we've been doing all term, is not an easy one. But it's not an impossible one either. We begin by identifying and inventing useful abstractions. Everything we do is an abstraction of reality. And as we've gone through the second half of the term, almost every interesting computation we've designed starts with inventing classes that give us useful data abstraction or functions that compute useful things. And that's sort of what we start with. We start with some existing set of abstractions, and we invent new ones that will help us think about the current problem.

We then formulate the solution to the problem as a computational experiment and then design and construct a sufficiently efficient implementation. It doesn't have to be the most efficient one possible. It just has to be efficient enough that we can run it and get an answer. Of course, before we trust the answer, as with any experimental apparatus-- and it is an experimental apparatus, a program-- we need to validate the experiment, convince ourselves that when we run it, we should believe the answer. That's the process of debugging. And I'm sure it's a process that all of you have spent more time on this term then you might have liked. I hope by now you feel you are a little bit better at it than you used to be.

Once we think we've got the experiment put together properly, we run it. We then evaluate the results. And then we repeat as needed, the classic iterative style of science. And a lot of this, of course, is what you would have learned in your chemistry lab or your physics lab or the bio lab in designing any experiment. You have to go through a lot of these steps. And the difference is these are all computational experiments.

So we think of there are two ways of computational thinking. There's abstraction. You have to choose the right abstraction. Typically, we operate in terms of multiple

layers of abstraction simultaneously. And this is an important part of thinking computationally. We have to be able to not worry about, say, the details of how floating point numbers were implemented, but just assume it's a good approximation of the real numbers. That's one level of abstraction. And another level of abstraction, we think about bus stop queues. And maybe we have to implement those ourselves.

And then we have to think about the relationships among the layers. A key thing that makes computational abstractions different from so many others is we can automate them. So we often think about abstractions as we go through life and think about how to cope with things that are too complex to understand. We have these various abstractions. Newton gave us some good abstractions of the physical world that let us understand how things like levers and such almost how they work.

But here we get the big advantage that we not only can invent an abstraction, we can mechanize it. We can do this for two reasons. One, we have a precise and exacting notation for expressing the abstractions for building our computational models. This term, that has been Python and PyLab and NumPy. But it could have been Java. It could have been C++. It could have been MATLAB. Doesn't matter very much. What matters is that we have a notation that's precise enough that we can actually describe a computation. And we have some machine that can take a computation written in that notation, or a set of computations actually written in that notation, and execute them and give us results.

And that's been the big transformation that has made computation so important. We've had the notion of computation long before we had machines that could execute them. The Greeks, the Egyptians understood notions of computation, as we've seen before when we looked at some early algorithms. Algorithms have been around a long time. But nobody cared very much until we had machines and notations that would let us actually run the algorithms. That came along in the late '40s. And then the world changed dramatically.

So some examples of computational thinking. How difficult is this problem, and how

best can I solve it? And that's what theoretical computer scientists spend their time thinking about. They try to give precise meanings to these kinds of questions so that we can ask them in the context of, say, random access machines or parallel machines or, more lately, quantum machines, how do we formulate those questions precisely? And how can we answer them?

And of course, thinking recursively is very important. Fundamentally, it's the key in that what we try and do is we try and reformulate a seemingly difficult problem into one we already know how to solve. This is what we talked about before, reduction to a previously solved problem. And we've looked at different ways of doing this. There is reduction, as I mentioned. There's embedding, where, OK, our problem is complex. We can't reduce it to a different problem. But we can embed a different problem in it as at least part of the solution.

We can do some transformations, take one problem and transform it to a different problem. Usually, a simpler problem that we know how to solve. We've seen a lot of that. That's, of course, what you've seen in the bus simulation you're studying, I hope as-- I hope not as I speak, but as soon as I finish speaking, you'll be studying it. We've taken a complex problem and simplified it by transforming it into a simpler problem that we can then attack.

And we've used a lot of simulation as a mechanism for dealing with problems. So here's a little recursive picture. Here's one of my favorite illustrations of recursion. I don't know how they did that, but I think it's pretty cool.

All right. That's a very quick introduction, at a very abstract level, what computer scientists might do. I'm gonna spend some time now on what one particular computer scientist does, and that's me. Why did I choose me? Well, not because I'm the most interesting, but because I'm the one I know the best. So it's kind of easier for me to talk about my work than about somebody else's.

And the truth be told, it's not actually my work. Like all professors, I claim credit for the work that my students do. So all of the work you're going to see is actually work that my students have done. Mostly graduate students, but also some very

productive UROPs. So, I won't give them as much credit as I should, but you should know in your heart that I didn't do any of this actually myself.

All right. What do we do? The goals of our research group is the rather modest one of helping people live longer and better quality lives. Other than that, we don't intend to accomplish anything. And we do this mostly in the medical area by collaborating with clinical researchers and practicing clinicians. Along the way we have a lot of fun pushing the frontiers, the state of the art in computer science, electrical engineering, and certainly medicine.

We use a lot of the technical kind of ideas you've seen this semester and that you'll see in almost any computer science course you take. We do a lot of machine learning and data mining. I'll explain why shortly. We do a lot of algorithm design because a lot of the problems we're trying to solve are too complex to solve if you're not clever about how you solve them. We do some signal processing. I'll explain why we do that. And a lot of just plain-old software systems are needed. Do we use databases? How we build software that injects electrical signals into people's brains? So we spend a fair amount of time trying to convince ourselves it actually works, doesn't inject the wrong signals at the wrong time. But that might be fun too. And these are just the logos of some of the hospitals we work with.

So what are some of the problems we try and deal with? Probably our longest standing project has to do with medical telepresence. These hands do not belong to a basketball player. These are normal sized hands like yours or mine. That is not a normal sized baby. This is not a Photoshop photo. This is a real photo. This is a very small premature baby. And we have spent a lot of time working with groups figuring out how to provide better outcomes for these tiny little, very delicate people.

A problem is that a lot of these babies are born-- if these babies are born in the right hospital, Brigham and Women's Hospital or something close to Children's Hospital or MGH, they have very good outcomes. They tend to survive, and they tend to grow up to be relatively normal. If these babies are born in a community hospital that does not know how to take care of these kinds of babies, they have very bad

outcomes. A lot of them die. Many of them who survive end up being permanently disabled. Very sad.

One of the shocking statistics is that globally there's a neonatal death every 10 seconds. Shocking number, I think. It's just tragic. And many of these are unnecessary. And they occur because there's not adequate care at the point where the baby is born. And so we've been trying to use technology to link places where these babies might be born to places where people can advise how to take care of them.

And it turns out to be a very interesting set of computer networking communications problems. You can't just use Skype for reasons I won't have time to go into. But it's very exciting. It uses a lot of computer science, a lot of medicine, but mostly computer science, in this case. And we're actually currently running some trials in conjunction with Children's Hospital Boston where they're actually trying to look at babies born elsewhere.

Another problem we've been looking at is health-care associated infections. Approximately 1 in 20 hospital visits results in the patient contracting an infection totally unrelated to the reason they entered the hospital. And today it's among the top ten leading causes of death in the United States. Somebody enters a hospital for a reason x, picks up an infection unrelated to x, and dies. Not a very good outcome. And it's particularly discouraging because, in principle, these infections should be preventable. You shouldn't acquire a life-threatening infection while you're in the hospital. But people do.

We've been trying to understand why. We've been working on this project with Microsoft, which has provided us with data from 4.5 million different hospital visits and about 2000 pieces of information per visit, lab tests, drug tests, who the doctors were, who the nurses were, what rooms they were in. And we're trying to study-- these are pictures of some of the most obnoxious of these infections-- and we've been trying to figure out what's causing them and what can be done not to cause them.

This has been primarily a machine learning project. We've been struggling with it. We've been dealing with exactly some of the issues we talked about in class. Which features are most important? How should you weight the different features? How can we reduce the number of features so that we can actually finish our computations? We've been using different kinds of clustering, different kinds of supervised learning, many different techniques.

And we're beginning to get a handle here on, in fact, what are some of the causes of these infections and what could they be done. Some of the things are very disturbing. Some of them, a lot of them, seem to be caused by drugs that are given in the hospital. And maybe we can substitute one drug for another if we understand that. At least in some cases they seem to be related to things like what room you happen to be placed in, suggesting that maybe they're not cleaning the rooms adequately. Things like that.

All right. Another project, probably our biggest sets of projects, have to do with extracting information from physiological signals. We've been focusing on the heart, the brain, and the connected anatomy. For those of you who don't happen to be biology or course 20 majors, here's where the heart and the brain are located and what they look like.

Some of the examples we're interested in are predicting adverse cardiac events. And for that you can think about death from a heart event, heart attack. And we spent a lot of time dealing with epilepsy. So let me give a little bit more detail about these two examples because each of them relate to things we've covered in 6.00.

So epilepsy, an interesting disease. Surprisingly, it affects about 1% of the world's population. When I first heard this, I was astounded. Because gee, 1 out of 100 people that I know have epilepsy? And the truth is, yes. I just didn't know it. I've been amazed, since it's become known that I do research in this area, the number of people I've known for years who come up to me say, you know, I never mentioned this before, but I have epilepsy, or my kid has epilepsy or my parent has epilepsy, and I have some questions now that you're supposed to be an expert on

this. And I give them the usual caveats, you know, I'm not a doctor, I just play one on television. But then I tell them go talk to one of my students. They're not doctors either, but they're really smart. Probably smarter than your doctor. All right.

Why is it underestimated when we do it ourselves? Well, because they're still some stigmas. A few hundred years ago, they burned women in Salem, Massachusetts, who had epilepsy because they thought the seizures meant they were possessed by the devil. Today, if you tell the Registry of Motor Vehicles that you have epilepsy, they won't let you drive an automobile. So people tend not to announce it.

What's it characterized by is a recurrent seizure. A seizure is abnormal electrical activity that originates and persists in the brain. Number of causes. It can be acquired. It can be inherited. There's manifest different symptoms. Most of them do not look like what's portrayed on the Simpsons.

Slightly more realistically, and maybe more horrifying. Here's a picture of a young girl, a movie, you're gonna see a seizure. It's not pleasant, I'm gonna warn you. What I want you to notice is two things. (1) She's all happy, and the seizure seems to come out of nowhere. She doesn't expect it. She doesn't know she's going to have. It just hits. That's very important.

(2) She happened to be in the clinic at the time wearing this funny looking cap, with it has electrodes in it. Just sits on the scalp. And here is a record of the electrical activity at the surface of her scalp. And what you'll see is a quick slight change on it just before the seizure hits, and then enormous changes during the seizure. Turns out that the enormous changes during the seizure have nothing really to do with the seizure. They have to do with muscle activity. As you'll see once the seizure hits, there's a lot of ugly muscle activity.

All right. Let's look at it. Uh oh. All right. Let's look at it-- I thought I had put the links in here that should have done it, but who knows. I know where to find them.

[VIDEO PLAYBACK]

[SINGING IN FOREIGN LANGUAGE]

[SPEAKING FOREIGN LANGUAGE]

[END VIDEO PLAYBACK]

PROFESSOR: So kind of scary. Certainly when I've been seeing them actually happen, I've found it terrifying to watch. That kind of-- it's called a tonic-clonic seizure. They seem- I emphasize the word 'seem' -- unpredictable. The interesting thing about these seizures, or an interesting thing, is they're self-limiting. In a few minutes, that young lady or girl's seizure will be over, not because anybody intervened but because the brain resets itself. Then it will take-- in this case, probably took her about an hour to recover and get back to normal, which is not good. But after that, everything was as before the seizure.

The difficulty is because they're unpredictable, there are huge injuries. Imagine she'd been riding a bicycle when that hit. Well, something catastrophic. Or more simply, imagine she'd been going down a flight of stairs or in a bathtub. Any one of a number of things would have resulted in this potentially catastrophic collateral damage after the seizure or during the seizure. And indeed people who have epilepsy, you can see their scars. It's awful. It can result in death, about 1 per 100 patient years. It's called the sudden unexpected death in epilepsy patients.

So what we wanted to do is see whether we could detect the seizures and give a little warning. The notion being that even if you couldn't do anything about the seizure, if you could give somebody, say, even five seconds warning, they could sit down before they fell down. They could get out of the tub. They could back away from the stove. All sorts of things. Furthermore, if you could use technology to signal to someone else that there was about to be a seizure, help could arrive. Also potentially a good thing. Turns out there are now some fast-acting drugs that if you put under the tongue can ameliorate the seizure. These have not yet been approved by the FDA, but soon.

And we were particularly interested in neural stimulation. If we ejected electrical current into the brain at exactly the right moment, could we offset essentially the

effect of the seizure and do a reset? And maybe stop the seizure or abort it, prevent it, or at least reduce the long term recovery time.

The thing to keep in mind for the seizure is there were two distinct onset times. What you saw if you're looking at the girl was what's called the clinical onset, when there's some clinical event. If you were able to not look at the person and instead look at the EEG, not so easy, you would have seen that there was an electrographic onset that preceded the clinical onset. We know, in fact, since the clinical effects are caused by the electrical activity, there will always be abnormal electrical activity prior to any abnormal clinical activity. And so the hope was that we could detect the electrical activity early.

Now that's not so easy. What makes it hard is that the EEG, the electroencephalograph, differs greatly across patients. First of all, people with epilepsy have abnormal baseline EEG. Even when they're not having a seizure, bizarre things are going on in their brains electrically, all unusual things. And so they don't look like people who don't have epilepsy. And it varies tremendously across patients.

So for about 35 years, people attempted to build generic detectors that would detect seizures in everybody. And they've not worked well at all. Every time a hospital buys an EEG machine, it comes with a detector built in. And usually the first thing they do is they turn them off because the false alarm rate is so high that it's like the boy who cried wolf. They're just saying seizure, seizure, seizure, and people start ignoring it.

The good news is it's pretty consistent seizure onset for a particular individual. That suggests you should build not generic detectors, but patient specific detectors. And we've been working on using machine learning to do that. And in fact, it's been highly successful. We've done several retrospective studies indicating that it works really well. And we're now doing a substantial prospective study -- in progress at MGH. And as part of that, we're actually working at turning on a neuro-stimulator that we hope will attenuate the effect of the seizure. And I emphasize hope because we don't have enough data to know if it works. It's part of what goes on in this

business. But it's certainly been an interesting project.

All right. Heart attacks. Let's look at that. And I'll go to the easy way to show them. I should warn you, you're gonna see something that's not very pleasant. You're actually gonna see somebody die. This was actually a person who was playing in a soccer game and died while the game was being televised.

All right. What I want to show you is this didn't have to happen. We're now gonna see another televised soccer game where the player had exactly the same event but with a rather different outcome.

So look at the upper center of field and you'll see the person has collapsed, much the same way it happened before. Now watch the body convulse. You'll see the knees kick up there. And now comes the miracle. He sits up. He leaves the field and, in fact, he later asked if he can reenter the game. The coach, to the coach's everlasting credit, said no. Clearly the right decision.

So it's amazing really that this happens. So what was the difference between the two? well, we see this here. There are things you can do. So in acute coronary syndrome, think of it as some sort of a heart attack. They're very common, about one and a quarter million per year in the US. 15% to 20% of these people will suffer a cardiac-related death within the next four years. If you could figure out who were the people at highest risk of different events and choose the treatment properly, for example, implant an implantable cardiac defibrillator, you could save these lives.

And that was what we saw in our movies. That the first patient, first person, the one who died, did not have a defibrillator. The second person, the one who survived, did. So he collapsed on the field. The defibrillator sensed that his heart had stopped, gave it an electric shock.

You've all seen this in television where they put the paddles on and they say, clear. And then there's this moment of drama where everyone stares at the EKG machine, and it goes from a flat line to suddenly up and down, and everyone goes, ah. Well that's exactly what happened here. And that convulsion was this huge electrical

shock getting sent through the person's body which restarted the heart, saved his life. Probably, if the other player had had an ICD, he would not have died either.

So great technology. Well, yes and no. This was a study in the New England Journal of Medicine not so long ago, a very good control study, where they matched patients with ICDs and patients without ICDs and over 72 months tracked which ones lived and which ones died. Well, the disturbing news is there isn't much difference between the red line and the blue line. So these people had what was essentially a, well, which was about a $50,000 implant and various kinds of risks and inconvenience and discomfort, et cetera, and on average it didn't save lives. Oh, dear. In fact, 90% of the people who got them-- and this is not just in this study. This is in every study- receives zero, actually less than zero medical benefit from the ICD.

How do we know that? Well, remember, it only turns on when it senses the heart is in trouble. For 90% of the people who get it, it never energizes because the heart doesn't get in trouble, or detectable trouble. So what we see, if we look at a little more detail in this study, is that for sudden cardiac death, the kind of death we saw in those videos, or non-death, the ICD reduces that. Unfortunately, it increases other causes of death, for example, infections related to the surgery, et cetera, those unfortunate hospital acquired infections, for example, we talked about earlier.

So what we see here is we have a technology that if we knew whose heart was likely to stop or go into serious fibrillation, beating uncontrollably, and we only gave those people ICDs, we could save an enormous number of lives. But currently we don't know who's in that population, and therefore we don't know who should get them. So we use other mechanisms deciding who should get them. And we're wrong most of the time, doing more harm than good on those patients for whom we're wrong.

OK. Well, how do the people predict it today? The usual things. Are you male or female? Do you have high blood pressure? Diabetes? What's your cholesterol level? BMP? Various other kinds of things. Electrocardiograms, which look at-- it's

an echo cardiogram that looks at the activity of the heart. Is the blood flowing through it OK? EKGs. Many different methods for analyzing the electrical activity, et cetera.

We played with many of these techniques. The one I want to tell you about today, because it's the most closely related to 6.00 material, is what we think of the Tolstoy approach to risk stratification. So Tolstoy is famous for saying that happy families are all alike, but every unhappy family is unhappy in its own way. So we generalize that, or specialized it maybe, to say that happy hearts are all alike, but every unhappy heart is different, and then did some fairly simple work to quantify the difference between electrical activities in different people's hearts, converting it to symbols. You don't have to worry about that. We used dynamic programming as an important part because we were looking at roughly a billion heart beats and so we needed to make it run fast. And then we used clustering to identify patients whose hearts we thought were similar to one another.

Here are some results. These are people who had an acute coronary syndrome. So the dominant group, the biggest group, the quote "people who we thought were most likely to be fine"-- because remember, most people are fine after a heart attack-- 457 patients in this particular data set. And you can see that they did pretty well. A very small fraction of them died. Can't see it from this angle, but come out here. Yeah, less than 1%.

But then we looked at people who were in these other clusters. And in fact, we used a glomerate of hierarchical clustering to do this. And in cluster A, which had 53 patients, well, 3.77% died. If you happened to fall in cluster C, you were at very high risk. And we can just see that there's a big difference here. And so the notion is could we have used this to predict in advance who was most likely to benefit from the various kinds of treatments?

All right. Again, very quick just to give you the details-- not the details, the overview that the kinds of things that we cover in this course are actually quite useful in solving real practical problems.

All right. Let me wrap up the term. So I hope most of you feel you've come a long way-- maybe you prefer this picture. That if you think about the kinds of problems and struggles you had three months ago in getting tiny little programs to work and think about how easy those would be for you today, you should have some appreciation that you've really taught yourself a lot. And you really taught it to yourself, by doing the problem sets, in many ways, right. We've tried to help, but learning is very much up to the individual who's doing the learning. But I'm certainly impressed in looking at the kinds of really pretty complicated problems you guys can now solve.

We looked at six major topics. Clearly, you learned a notation for expressing computations, and that was Python. And I hope you don't think that's the only notation you can use and that if you take a course that uses MATLAB, you'll say, oh, this is just the same. It's easy. Or Java or anything else. Learning programming languages is easy. Once you've learned one, the next one is much easier.

Harder was learning about the process of writing and debugging programs. You've learned that, I think, largely through experience. You've learned about the process of moving from a problem statement, something as vague as should shuttle buses be bigger or faster to improve service, to a computational formulation of the problem and a method for solving a problem. And we've looked at lots of different methods.

You've learned basic recipes, algorithms, things like dynamic programming, things like depth-first search, things like decision trees, that you'll be able to use again and again. The good news is there are only a really small number of important recipes. And once you've mastered those, you just use them over and over again to solve new problems.

Spent a lot of time on using simulations to shed light on problems that don't easily succumb to closed form solutions. And I think that's really the place where computation is so important. There are a lot of problems where you can turn them into a set of differential equations, maybe solve the equations, and you're done. Oh,

we'll write programs to do those because we're lazy. But in principle, you could solve those without a computer.

But in fact, the thing you can't do without a computer is these messy problems that are most of what goes on in the real world, where there's randomness, and things like that, and there is no simple formula that gives you the answer. And what you have to do is write a simulation and run it and see what goes on. And that's why we spent so much time on simulation because it really is increasingly the thing people use.

And we learned about how to use computational tools to help model and understand data, how to do [?] plots, a small amount of statistics, machine learning, just dealing with data.

Why Python? It's pretty easy to learn to use. Compared to most other programming languages like, say, C++ or Java, Python is easy. The syntax is simple. It's interpretive, which makes debugging easier. You don't have to worry about managing memory as you do in, say, C, right. You get a big list or dictionary, and magically it exists. And when you don't need it any more, it's gone. It's modern. It supports the currently stylish mode of object-oriented programming in a nice way with its classes and things like that. So indeed what you would hear about in a Java class, we can cover almost all the interesting things with Python.

And it's increasingly popular. It's used in an increasing number of subjects at MIT. A lot of course 6 subjects, but also a lot of course 20 subjects, course 9 subjects, over and over again. It's becoming, probably, the most popular language at MIT and at other universities. Increasingly it's used in industry. And as you've seen with PyLab, the libraries are amazing. And so PyLab, Random, et cetera, there's just a lot of stuff you can get for free if you're living in the world of Python.

The main thing in writing, testing, and bugging programs, I hope you've learned, is to take it a step at a time. Understand the problem. Think about the overall structure and algorithms separately of how you express them in the programming language, right. We can talk about dynamic programming conceptually without worrying about

how to code it up.

Always break the problem into small parts. Identify useful abstractions. Code and test each unit independently. Worry about functionality first. Get it to work. Get it to give you the right answer. Then worry about making it do that more quickly. But separate those things. Start with some pseudo code.

Then above all, be systematic. When debugging, think about the scientific method of forming hypotheses, forming experiments that can test the hypotheses, running the experiment, checking the results. Don't try and do it too quickly. Just be slow and careful. Slow and steady will win the race in debugging.

And when your program behaves badly-- and the sad news is no matter how many years you spend at it, you will write programs that don't work the way you think they should the first time-- ask yourself why it did what it did, not why didn't it do what you wanted to do. It's a lot easier to debug from the how come it's behaving the way it is then why isn't it behaving the way I want it to behave.

In going from problem statement to computation, break the problem into smaller problems. Try and relate your problem to a problem that somebody else, ideally, has already solved. For example, is it a knapsack problem? Is it a shortest path problem? If so, good, I know how to solve those.

Think about what kind of output you might want to see. What should the plots be like? I usually design the output before I design the program. Often you can formulate things as an optimization problem. Pull back. Say, well, can I formulate this as finding the minimum or maximum values satisfying an objective function and some set of constraints? If I can, it's an optimization problem, and therefore I know how to attack it.

And don't be afraid to think about approximate solutions. Sometimes you're just gonna not actually be able to solve the problem you want to solve. And so you'll find a solution to a simpler problem, and that will be good enough. Sometimes you can actually solve the problem by finding a series of solutions that approaches, but may

never reach, a perfect answer. Probably the third week in the course, we looked at Newton's method as an example of that kind of problem.

And then there are algorithms. We looked at big O notation, various kinds of algorithms, a lot of kinds. You've already - I sent a list of specific algorithms. And particularly, we looked at optimization problems.

We spent a lot of time on modeling the world, keeping in mind that the models are always wrong. But nevertheless, they're often useful. They provide abstractions of reality. So we looked at two kinds of simulation models, Monte Carlo and queuing networks. We looked at statistical models, for example, linear regression. And we looked at some graph theoretic models. Those are not the only techniques, but they're very useful techniques.

We looked at making sense of data. We talked about statistical techniques. How to use them well. How to use them badly. We looked at plotting. And we spent some time on machine learning, supervised and unsupervised and feature vectors, and spent a lot of time talking about how do you choose the features, because fundamentally that's typically the difference between success and failure in the world of machine learning. Throughout it all, the pervasive themes were the power of abstraction and systematic problem solving.

So what's next? Many of you have worked really hard this semester. We know that. And the TAs and LAs and I all appreciate that. And I thank you sincerely for the effort you put into the course. Only you know your return on the investment. I hope it's good. Remember as you go forward in your careers that you can now write programs to solve problems you need to solve. Don't be afraid to do it.

If you like this, there are plenty of other CS courses that you now could take. You're actually equipped, probably, to take any one of these four courses based upon what you've learned in 6.00. You could major in course 6 or think about the new major in computer science and molecular biology. And you're certainly qualified to go off and to look for UROPs that involve serious programming.

All right. Wrapping up with some famous last words, these were words that some famous people said as they were about to die. An actor, Douglas Fairbanks senior, was asked by a family member, how are you feeling? He said, never felt better. And then that was the last thing he ever said. In contrast, the last thing Luther Burbank was reported to have said, a famous scientist, was, I don't feel so good. He was a scientist rather than actor. He had a better appreciation of the state of the world.

Conrad Hilton, who you probably think of as Paris Hilton's grandfather, but actually is more well known in some circles for running the Hilton hotels, his last words when his family asked him-- they knew he was dying-- is there anything you want us to know about running the business? And he said, yes, leave the shower curtain on the inside of the tub. And that's advice, I have to confess, my wife has given me on several occasions.

Archimedes, basically before he was taken away and executed, asked, could he please finish solving the problem he was working on. You know, I could imagine some poor 6.00 student who's about to be carted away by the police saying, can I finish my problem set first, please? Actually, I can't imagine that.

And finally, this US Civil War general, John Sedgwick, was reputedly-- and this is, I believe, a true story-- said, talking about the snipers from the other side who were quite far away, that telling his people, don't be afraid, they couldn't get an elephant at this distance. And if you go to the site of the battle, you'll find this plaque describing the death of John Sedgwick who was shot and killed immediately upon saying this by one of said snipers. It says something about generals that probably is still true today.

All right, quick reminders and then we're done. There's a final exam. If you haven't done the underground guide, go do it. These are things I said at the beginning of the lecture. I'm just repeating them.

Thanks a lot. Good luck in the final 6.00, all your finals. And then more importantly, have a great summer.