Donald Knuth: Programming is like nothing else. Become friends with geeks

17. října 2019

Telling the computer what to do is unlike any other task in the world. Algorithms are now everywhere. Not everyone will learn how to write computer code, but everyone should understand a bit about how programming works. Professor Donald Knuth, one of the original computer geeks, introduced us to his Art of Computer Programming.

Donald Knuth on how to start programming | (0:45) | video: Pavel Kasík, Technet.cz

Česká verze rozhovoru

“Testing. Zero one, zero one.” Even his microphone check combined his life-long love for computer science and his unique sense of humor. The lecture hall was full of students – mostly studying computer science – who knew professor Knuth as the legendary author of the seminal opus The Art of Computer Programming, which Knuth started writing in 1962 and is still working on diligently.

Talking to Donald Knuth, now 81 years old, is in some ways akin to reading a book. He is thoughtful, systematic and careful to give a precise answer to your question. It is clear he is thinking about the intended audience when finding the right explanations and metaphors. Despite his age, he is still actively programming, creating and solving puzzles and finding ways to explain computers to yet another generation of students.

Donald Knuth giving a public lecture at Masaryk University in Brno (2019)

After his lecture at Faculty of Informatics at Masarykova University in Brno, we sat down with Professor Knuth in a quiet corner of a bland hotel corridor, for what was supposed to be a short interview. Instead, we talked for a full 90 minutes. If this text feels too long to our readers, it means we did a poor job conveying how the conversation felt at that moment. Within minutes, we were transported back in time and space, and then into the world of ones and zero, Biblical symbolism, precise definitions and artificial intelligence.

Computers are changing everything

How would you define an infinite loop?
That is a remarkable question. We will get to that later.

Donald Knuth, PhD.

Computer scientist, mathematician and author, born 1938 in Milwaukee, Wisconsin. He earned his PhD in mathematics from the California Institute of Technology in 1963.

He is the author of the famous comprehensive multi-volume guide The Art of Computer Programming, which he started in 1968 and is still working on. Knuth is known as the founder of the analysis of the computational complexity of algorithms. His typesetting system T_EX is still widely used around the world.

In 1974 Knuth, then 34, won the prestigious Turing Award. He was also awarded the National Medal of Science or John von Neumann Medal.

When you are introduced on stage, it often takes the host several minutes to get through all your accomplishments and awards. How would you introduce yourself to our readers?
Well, I could, of course, list where I studied, what books I wrote and such general things. But one specific thing you can say is that I'm a star. Or, rather, a minor planet. Czech astronomers discovered an object 21656 and named it Knuth after me. It is cool. Of course I hope it doesn't collide with Earth one day or anything like that.

You are sometimes called the father of computer science. How did computers shape human civilization in the past seventy years?
What is civilization? Let's say it's things that were not present in nature but

were added, invented by humans. Just sitting in this hotel lobby, we can notice one hundred different aspects of civilization. The lights, the glass, the fact we are sitting and we wear clothes, the building itself. Civilization is incredibly complex; and was complex already when computers came along.

When I was your age, computers were slow. The speed comparison I use is that in those years the speed and storage space increased from a snail to a jet plane. When it comes to storage space… I was programming for 20 years without having even a megabyte of storage. Now you can put a terabyte in your pocket.

Donald Knuth, Professor Emeritus of The Art of Programming at the Stanford University

So many things that were previously inconceivable – so much that you wouldn't even ask for them – are now feasible and commonplace. Computers are universal. We can make a digital simulation of absolutely anything. Sure, we don't eat digital foods, yet, but it is hard to find any aspect of civilization that hasn't been affected by a computer.

Because computers are everywhere, we have a problem we didn't have just 20 years ago. For a long time, almost nobody was listening to us computer scientists. Now I worry maybe too many people are listening to us and they think we know more than we do. We are supposed to save everything.

Almost every field now uses computers. Even music and poetry uses computers. That gives us computer scientists too much responsibility. Of course, we can always do better. And also worse.

There is a tremendous potential danger of autonomous weapons (as seen for example in the movie Slaughterbots).

Slaughterbots (fiction, part of the push to ban autonomous weapons):

Thousands of people from my field, myself included, have signed a letter warning against this danger.

Computers can do things faster than people. One program can be used again and again. Is the difference of the computer era mainly a difference in scale?
No, it's more than that. As we express our design, we are explaining something to the computer. And to do that, we need to really understand what we are explaining. If I explain something to you, you will nod your head and I know you got it. When explaining things to a computer, you won't get such feedback. The computer doesn't nod its head. You have to explain things absolutely perfectly.

When someone makes software to design buildings, they need to understand buildings much better. It is a great educational boost to anybody involved. Everybody who is designing computer programs needs to understand in great detail how things work.

And, of course, when someone makes a mistake, the computer will repeat the mistake infinite times without question.
That is precisely the point.There are billions of things that are part of our civilization. We are not going to get everything right. At the same time, when you think about how many things could have gone wrong, it is not so bad. Computers help us understand scale a little better than before.

Computers force us to understand the world

Did early programmers understand this effect computers would have?
In early 60s, George Forsythe (*the founder of Stanford Computer Science department*) told me that the most important aspect of computers is the education: what computers teach us when we are using them. Not what they do for us, but what they teach us.


Donald Knuth at California Institute of Technology (1965)

I subsequently learned that this was true from the beginning. When Remington made their first computer UNIVAC in the 50s, it was pretty much the first computer available outside of the military and scientific institutions.

The first UNIVAC had four customers: an insurance company, a US Census Bureau, the Army, and the fourth one was Nielsen Corporation. They later became famous for TV ratings. But in the fifties, Nielsen had a company that tried to analyze various aspects of business. Nielsen was a foresighted person and thought, hmm, maybe I could use a computer.

A pioneer of big data, perhaps.
Yeah. And people from UNIVAC studied Nielson's company to figure out how they are using the computer for the operations of his firm. As a result of the study, Nielson realized how he can do things more efficiently, even without a computer. But just having to think how he would use the computer forced him to learn more about the workings of his company.

Computer forces you to express in a way that has only one interpretation.
Yes, complete and unambiguous explanation. I say this over and over again. Computers change people as people use them.

Most people don't understand programming. Yet almost everything is – in some way or another – dependent on computer code. Is this a recipe for disaster? What price are we paying for people not understanding computer science?
I don't want to imply that other science fields are not important. If anything, I think biology is much more important. Anyway, we cannot expect 100 percent of people to understand the computers. *Vive la différence*. Diversity is good. Different people understand different aspects of the world more than others. Knowledge is multi-dimensional.

And science is now taught at school more than before. People ask, how should we teach programming? Well, it's hard, because most teachers don't know how to program themselves. But even if we have a better understanding of what programming is, most people won't be in tune with it. I say maybe one person in fifty will find that they groove with programming.

Only some will get the underlying workings…
It will sound natural to them. And these are the people, let's call them computer geeks. I seem to be born as one of these. My goal is not to change everyone in the world into a geek. I try to write books that geeks will learn from.

The Art of Computer Programming is Knuth's main project since 1968. The volumes have over three thousand pages so far, and more are scheduled for publication.

Appreciating the mystery

What should the non-geeks do?

Non-geeks, become friends with geeks! (laughing) And then they will understand more about what geeks do, and geeks will understand more of what non-geeks do. I worked on a project with graphic designers and programmers. Everyone appreciated the other, we were in the same room and we learned from each other.

Think of companies like Pixar. Walking the corridors you'll find all kinds of different skills. Nobody dominating the other but each contributing.

You wrote a book called Questions Computer Scientists Don't Get Asked. In that book you explain your personal Christian belief. For some people, it can be surprising to find that an expert in a scientific field believes in God.

On one hand, I realize I became religious also because of where I was born. Had I been born in India or Japan, it would change me. On the other hand, I personally believe that God exists. But, thank God, there is no way to prove this.


Donald Knuth at the opening event of Computer History Museum exhibition (2011)

I don’t have a magical proof of God’s existence. Many people claim that they had such proof, but I don’t. If there was proof, there would be no searching. There would be no mystery; it would be trivial.

I appreciate the fact that there are things beyond my understanding and always will be. If everything in my life was cut and dry and certain, I would feel incomplete. If everything in my life was a mystery, I would also feel very lost. I am grateful to know 1+1 = 2 and other mathematical things that I can combine to prove different things. Mystery teaches me to be humble and not to expect I can know everything.

In my lectures, I emphasize to people they shouldn’t believe in something just because “this other smart person told them so.” But I think these are important questions people should be thinking about. They shouldn’t be apathetic.


Donald Knuth in Brno

The musical piece you composed and came to present in Brno is religious in nature as well.
It’s based on one of the books of the Bible. In fact, I tried to do something new and I might have succeeded. I tried to take the Greek text of the book of Revelation and I made an almost literal translation into music.

The book of Revelation is the most mystical part of the Bible. I concluded the book is intended to be mystical. Not to give answers but to inspire questions. There are a lot of numbers in it, a lot of symbols. Three is often used to refer to God, four to the world, seven references the relationship between God and the world. I studied the book and found over one hundred symbols. My piece is based on the same sequence of the symbols, but I also tried to make good music, music that you could hum. It lasts 90 minutes and as the musical symbols play out, we put the words and symbols on the screen.

„Fantasia Apocalyptica“ by Donald Knuth (2018):

Is writing music similar to writing code?
Oh, it’s amazingly similar. You just don’t use the same kind of notation, but there is a lot of commonality. You deal with patterns and you try to

understand what the listener is expecting and not expecting. You have to be organized. Well, you could be disorganized, some composers are.

On my hand-written score, I write down what I was thinking about. Because I knew I wouldn't remember later.

Questions programmers ask other programmers

We couldn't resist. Who better to bring some unity into these divisive geeky questions than one of the original geeks?

Do you use tabs or spaces when indenting your code?
Yesterday I heard for the first time that people have correlated this preference to salaries, which is a funny thing. Anyway, I use spaces. I indent with two spaces, not four. One didn't look like enough. When I read code from other people, I immediately remove tabs using the untabify command on emacs.

Do you use code syntax highlighter?
My Emacs uses minimal markup. It shows me matching parenthesis and that is it. No colors. When I type in TeX, formatting comes up red and sequences come up green and that helps me understand the code.

Should indexes start with one or zero?
One or the other, depending on what seems more natural in that context. Sometimes you can use zero for something else that will make your code simpler. When I'm trying to explain something, it's easier to say "go from one to n" than "go from zero to n minus one". Other times, the "start with zero" convention is easier and cleaner.

What computer do you use?
The computer I use for all my programming is a standalone PC running Linux. It is not connected to the internet. I use a Mac for internet browsing. All my manuscripts are on the PC. Everything is customized to be efficient for me and my process. I move files using a flash drive.

Do you back up your data?
Once a month I back up my files at Stanford University. At that point I will check all the changes. I look at the differences (diff files) and I rethink everything. If it was all on one computer I wouldn't have this opportunity. This way I am more aware of what I am doing on a monthly basis. This separation works for me.

What error message irritates you the most?
It changes over the years. Recently I had an error on my Mac saying the "Bluetooth device lost connection." My hearing aid can connect to the computer as wireless headphones. When I walked out of the room and came back in, my screen had a hundred error messages saying "Bluetooth device lost connection." There was no way to dismiss all of them, so I had to confirm all them, one by one. I stopped using that feature.

Do people ask you to fix their computer problems since you are the computer scientist?

I have a secretary who protects me against requests like that. I have defenses put up.

Don't try things at random

When you were a teenager, you won a prize in a contest. Participants were asked to rearrange the letter of the candy brand "Ziegler's Giant Bar". The task was to only guess how many words could be made using those letters. You instead found and listed all 4500 words, almost double of what the judges thought possible. How did you go about it?

This is described in one of my books (*Selected Papers on Fun and Games*), I remember it well. I actually pretended that I had a stomach ache so that I could stay home from school for two weeks. At the library, I had seen these big dictionaries, and my father had a copy of an unabridged dictionary. I went systematically over all the words. I had a card with the alphabetized list of the available letters and I could scan through pages pretty fast. Skipping a lot of pages that were not fitting the letters.

What a computer could have done within seconds, you did within two weeks. But you still approached the task with an algorithm of sorts. You thought about how to make the solution complete.
I had to make sure I am not missing anything. At every state I knew exactly how much space I had not yet considered.

Not all programmers have this discipline and grasp. What do you think about how some programmers basically try different approaches, trial and error, copy and paste, until something works?
If you carry this to an extreme it is a bad approach. It's like a monkey trying to write Shakespeare by mashing the keyboard randomly. On the other hand, when you try to do something, it is wrong to try to get it perfect at the first try. You better start with something that you can tear up if it doesn't work.

But don't try at random. Try to forget that you can tear it up. Try to think systematically. If you always approach things at random, you will always have to do things from scratch. You should try to do a pretty good job, even when you know you won't do a really good job at first.

In general, the idea of iterations, trial and error, is good. But you have to always give it your best shot.

Writing computer code for other people

You wrote extensively about literate programming – the idea to write computer code that is not only functional but also easy to understand for other humans reading the code.
That also means that you will be able to understand your own code a month after you wrote it. Don't think of yourself as speaking to a compiler. Imagine you are speaking to a human being. This will also improve your own understanding of what you are doing.

As a professor, I know what is like to talk to a whole class of students. So when I do literate programming, I try to imagine that I am explaining this program to a whole class. I include even comments why I didn't do something that I thought of. I include my reasoning.

Most programmers don't include comments about the tricky parts. That makes it so much harder to understand. This goes even beyond programming. It's useful to say things twice, once informally and once more precisely.

This is how we talk to people as well. We emphasize and repeat when we know we are describing something that is harder to understand. Informally you say "have I gotten to the end of the table" whereas formally you'd say "if (i>n)". When I'm writing code, I will say both things. Those two reinforce each other. Literate programming mixes natural language and the programming language.

If someone never coded, if someone doesn't know any programming language, what are they missing about the world?
They are missing what it means to say something carefully and unambiguously. There was a TV show in the 60s where the secretary was a robot. It was amazing to some people that a computer would do exactly as told. Because people don't work that way. This character – a very good-looking young lady – was a robot and people had to learn that they need to tell her everything exactly, not approximately.

First episode of the TV Series „My Living Doll" (1964):

What is true about programming that is not true of any other endeavor. How is programming unique?
Programming means you need to spell things out without any ambiguity. Sure, you could invent loose programming, but the essence of programming is unique. You need to be completely specific and unambiguous. That forces you to think about what is happening and what could happen in any imaginable case. Computer uses no judgement.

Even if we don't realize it, a lot of processes around us are algorithms. Loan assessment, police fining speeding cars, finding connecting flights… Is there something everyone – not just programmers – should understand about algorithms?
I know what algorithms are. It's part of my life's work. So I can make use of it. Now when I see any process whatsoever, I think about what the algorithm for that process looks like. How is my keyboard auto-correcting my spelling? I can try to learn more and I can ask, how would I explain this process to a computer? I can write an algorithm that analyzes music chords. People could always analyze chords, but being forced to formally explain chords to a computer in the form of algorithm helps me understand it better than before.

The main value for me is the educational value, the new insight you get thanks to applying something explicitly and systematically. It's more important what you learn rather than how you will use your computer.

What would you tell someone who is thinking about starting programming?
If you have a chance to work with some kind of fun system that will carry out simple instructions, that's always a good start. It will teach you to be explicit. To say how to do something, all the way to the finest details. Saying exactly, exactly what to do. That's what programming is.

You are famous for analyzing the efficiency of algorithms. Yet with computers as fast as today, programmers often forgo efficiency since computers are fast enough for them not to care if they are running the task twice as long, it's still just milliseconds.
If you have to do something a trillion times, then every nanosecond counts.

You love puzzles and you keep publishing puzzles for your readers on your website. Do you have a puzzle for our readers?
OK, let me think. This is a simple one:

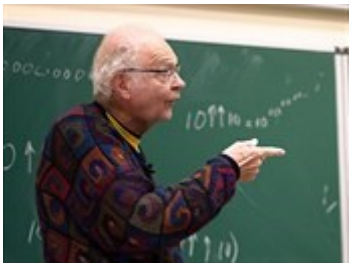
- **Puzzle**
- Solution

In what city is the t-shirt size XL actually smaller than L?

Artificial intelligence is not explaining itself

There's a lot of talk about artificial intelligence and machine learning. Do you remember when you first heard these terms in your career?
Machine intelligence came much later. Artificial Intelligence was one of the main topics from the time I was in high school. I first heard about it in college. Understanding how the human brain works was one of the fields that inspired people to push computers to their limits.

I remember losing many games of chess to a computer. There was a computer convention in Santa Monica. The computer beat me in chess and then sang a song. That was pretty memorable.



Q&A session by Donald Knuth: Boundless Interests – Computer programming is an art form (Masaryk University in Brno)

Artificial Intelligence went through many different phases and approaches. The goal was trying to figure out how the brain works. But then some people decided we don't need to solve problems the way humans do. We don't need to mimic the brain. Machine learning creates a new kind of brain and explores its possibilities.

Now we have neural networks that work great but we don't know if they will work tomorrow, and that's because we often have little idea about how exactly they work. Yet they do seem to work so we place more and more trust in them. Without really understanding their limitations, though, we are getting further and further from trustworthiness.

Smarter people than I – Stuart Russel, for example – have speculated on this. Let's say a machine learning algorithm solved an important mystery like the famous P = NP problem. We might construct a big neural network that somehow solves this problem. But we wouldn't know how the problem was solved, it wouldn't be able to tell us.

Given that AI is going to play a bigger role in the upcoming years, what should parents teach their children about AI?
People should understand its limitations and what these limitations are based on. You can start with examples that maybe don't affect your life but it will help you understand how it works. You can create a small neural network that will learn how to do one little thing. Perhaps something silly, like distinguish between a horse and a hyena.

Teaching children is in some ways like teaching neural network, isn't it?
We didn't just have babies in our house. We also had cats. And cats were a lot smarter than the babies. Up until a point. Then they weren't the smartest any longer. It was very interesting.

And finally, how would you define an infinite loop?

Oh, we talked about that already.

Donald Knuth on how to start programming | (0:45) | video: Pavel Kasík, Technet.cz

Autor: Pavel Kasík

Zdroj: https://www.idnes.cz/technet/technika/donald-knuth-interview-computer-science-brno-czech-republic.A191016_112708_tec_technika_pka