



Alan Turing: Crash Course Computer Science #15

Crash Course: Computer Science

<https://youtube.com/watch?v=7TycxwFmdB0>

<https://nerdfighteria.info/v/7TycxwFmdB0>

===== (00:00) to (02:00) =====

Hi. I'm Kerry Ann, and welcome to Crash Course Computer Science. Over the past few episodes, we've been building up our understanding of computer science fundamentals such as functions, algorithms, and data structures. Today, we're going to take a step back, and look at the person who formulated many of the theoretical concepts that underline modern computation, the father of computer science, and not quite Benedict Cumberbatch look-alike, Alan Turing.

Alan Mathison Turing was born in London in 1912, and showed an incredible aptitude for maths and science throughout his early education. His first brush of what we now call computer science came in 1935 while he was a masters student at King's College in Cambridge. He set out to solve a problem posed by German mathematician David Hilbert known as the Entscheidungs Problem or Decision Problem, which asked the following, "Is there an algorithm that takes, as input, a statement written in formal logic, and produces a 'yes' or 'no' answer that's always accurate?" If such an algorithm existed, we could use it to answer questions like, "Is there a number bigger than all numbers?" No, there's not. We know the answer to that one, but there are many other questions in mathematics that we'd like to know the answer to. So if this algorithm existed, we'd want to know it.

The American mathematician, Alonzo Church, first presented a solution to this problem in 1935. He developed a system of mathematical expressions called Lambda Calculus, and demonstrated that no such universal algorithm could exist. Although Lambda Calculus was capable of representing any computation, the mathematical technique was difficult to apply and understand. At pretty much the same time on the other side of the Atlantic, Alan Turing came up with his own approach to solve the Decision Problem. He proposed a hypothetical computing machine, which we now call a Turing Machine. Turing Machines provided a simple, yet powerful mathematical model of computation. Although using totally different mathematics, they were functionally equivalent to lambda calculus in terms of their computational power. However, their relative simplicity made them much more popular in the burgeoning field of computer science.

===== (02:00) to (04:00) =====

In fact, they're simple enough that I'm going to explain it right now. A Turing Machine is a theoretical computing device equipped with an infinitely long memory tape which stores symbols, and a device called a read-write head which can read and write or modify symbols on that tape. There's also a state variable, in which we can hold a piece of information about the current state of the machine, and a set of rules that describes what the machine does given a state and the current symbol the head is reading. The rule can be to write a symbol on the tape, change the state of the machine, move the read-write head to the left or right by one spot, or any combination of these actions.

To make this concrete, let's work through a simple example. A Turing Machine that reads a string of ones ending in a zero, and computes whether there is an even number of ones. If that's true, the machine will write a 1 to the tape, and if it's false, it will write a 0. First, we need to define our Turing Machine rules. If the state is "even," and the current symbol on the tape is 1, then we update the machine state to "odd," and move the head to the right. On the other hand, if the state is "even," and the current symbol is 0 (which means we've reached the end of the string of ones), then we write 1 to the tape, and change the state to "Halt" (as in we're finished, and the Turing Machine has completed the computation). We also need rules for when the Turing Machine is in an "odd" state, one rule for when the symbol on the tape is a 0, and another for when it is 1.

Lastly, we need to define a starting state, which we'll set to be "even."

Now that we've defined the rules and starting state of our Turing Machine, which is comparable to a computer program, we can run it on some example input. Let's say we store "1 1 0" onto tape. That's two ones, which means there is an even number of ones, and if that's news to you, we should probably get working on Crash Course: Math. Notice that our rules only ever move the head to the right, so the rest of the tape is irrelevant. We'll leave it blank for simplicity. Our Turing Machine is all ready to go, so let's start it. Our state is "even," and the first number we see is a 1. That matches our topmost rule, and so we execute the effect, which is to update the state to "odd," and move the read-write head to the right by one spot. Okay. Now we see another 1 on the tape, but this time our state is "odd," and so we execute our third rule, which sets the state back to "even," and moves the head to the right.

===== (04:00) to (06:00) =====

Now we see a 0, and our current state is "even," so we execute our second rule, which is to write a 1 to the tape signifying that, yes, it's true. There is an even number of ones. And finally, the machine halts.

That's how Turing Machines work. Pretty simple, right? So you might be wondering why that's such a big deal. Well, Turing showed that this simple, hypothetical machine can perform any computation if given enough time and memory. It's a general-purpose computer. Our program was a simple example, but with enough rules, states, and tape, you could build anything. A web browser, World of Warcraft, whatever. Of course, it would be ridiculously inefficient, but it is theoretically possible, and that's why, as a model of computing, it's such a powerful idea. In fact, in terms of what it can and cannot compute, there is no computer more powerful than a Turing Machine. A computer that is as powerful is called Turing Complete. Every modern computing system, your laptop, your smartphone, and even the little computer inside your microwave and thermostat, are all Turing Complete.

To answer Hilbert's Decision Problem, Turing applied his new Turing Machines to an intriguing computational puzzle, the Halting Problem. Put simply, this asks, "Is there an algorithm that can determine given a description of a Turing Machine and the input from its tape, whether the machine will run forever or halt?" For example, we know our Turing Machine will halt when given the input "1 1 0," because we literally walked through the example until it halted. But what about a more complex problem? Is there a way to figure out if a program will halt without executing it? Some programs might take years to run, so it would be useful to know before we run it, and wait, and wait, and wait, and then start getting worried, and wonder, and decades later when you're old and gray 'CTRL+ALT+DEL' with so much sadness. Unfortunately, Turing came up with a proof that shows the Halting Problem was, in fact, unsolvable.

For a clever, logical contradiction, let's follow his reasoning. Imagine we have a hypothetical Turing Machine that takes a description of a program and some input for its tape, and always outputs either "Yes, it halts," or "No, it doesn't." And I'm going to give this machine a fun name. H for halt. Don't worry how it works, let's just assume such a machine exists. We're talking theory here.

===== (06:00) to (08:00) =====

Turing reasoned that if there existed a program whose halting behavior was not decidable by H, it would mean the Halting Problem is unsolvable. To find one, Turing designed another Turing Machine that built on top of H. If H says the program halts, then



Alan Turing: Crash Course Computer Science #15

Crash Course: Computer Science

<https://youtube.com/watch?v=7TycxwFmdB0>

<https://nerdfighteria.info/v/7TycxwFmdB0>

we'll make our new machine loop forever. If the answer is no, it doesn't halt, we'll have the new machine output a "no" and halt. In essence, we're building a machine that does the opposite of what H says. Halt if the program doesn't halt, and run forever if the program halts.

For this argument, we'll also need to add a splitter to the front of our new machine so that it accepts only one input, and passes that as both the program and input into H. Let's call this new machine Bizarro. So far, this seems like a plausible machine, right? Now it's going to get pretty complicated, but bear with me for a second. Look what happens when you pass Bizarro a description of itself as the input. This means we're asking H what Bizarro will do when asked to evaluate itself. But, if H says Bizarro halts, then Bizarro enters its infinite loop and thus doesn't halt. And if H says Bizarro doesn't halt, then Bizarro outputs a "no," and halts. So H can't possibly decide the halting problem correctly because there is no answer. It's a paradox, and this paradox means that the halting problem cannot be solved with Turing Machines.

Remember, Turing proved that Turing Machines could implement any computation, so this solution to the Halting Problem proves that not all problems can be solved by computation. Wow, that's some heavy stuff! I might have to watch that again myself. Long story short, Church and Turing showed that there were limits to the ability of computers. No matter how much time or memory you have, there are just some problems that cannot be solved. Ever. The concurrent efforts by Church and Turing to determine the limits of computation, and in general, formalize computability, are now called the Church-Turing Thesis.

At this point in 1936, Turing was only 24-years-old, and really, only just beginning his career. From 1936 through 1938, he completed a PhD at Princeton University under the guidance of Church, then after graduating he returned to Cambridge. Shortly after in 1939, Britain became embroiled in World War II. Turing's genius was quickly applied to the war effort.

===== (08:00) to (10:00) =====

In fact, a year before the war started, he was already working part-time at the UK's Government Code and Cypher School, which was the British code-breaking group based out of Bletchley Park. One of his main efforts was figuring out how to decrypt German communications, especially those that used the Enigma Machine. In short, these machines scramble text, like you type the letters, "H E L L O," and the letters, "X W D V J," would come out. This process is called encryption. The scrambling wasn't random. The behavior was defined by a series of reorderable rotors on the top of the Enigma Machine, each with 26 possible rotational positions. There was also a plug-board at the front of the machine that allowed pairs of letters to be swapped. In total, there were billions of possible settings. If you had your own Enigma Machine, and you knew the correct rotor and plug-board settings, you could type in "XWDVJ," and, "HELLO," would come out. In other words, you decrypted the message. Of course, the German military wasn't sharing their Enigma settings on social media, so the Allies had to break the code. With billions of rotor and plug-board combinations, there was no way to check them all by hand.

Fortunately for Turing, Enigma Machines and the people who operated them were not perfect. Like, one key flaw was that a letter would never be encoded as itself. As in, an H was never encrypted as an H. Turing, building on earlier work by Polish code-breakers designed a special-purpose electromechanical computer called The Bombe, that took advantage of this flaw. It tried lots and lots of combinations of Enigma settings for a given encrypted message. If The Bombe found a setting that lead to a letter being encoded as itself, which we know the real Enigma Machine couldn't do, that

combination was discarded. Then, the machine moved on to try another combination.

So, Bombes were used to greatly narrow the number of possible Enigma settings. This allowed human code-breakers to hone their efforts on the most probable solutions, looking for things like common German words in fragments of decoded text. Periodically, the Germans would suspect someone was decoding their communications, and upgrade the Enigma Machine. Like, they'd add another rotor creating many more combinations. They even built entirely new encryption machines. Throughout the war, Turing and his colleagues at Bletchley Park worked tirelessly to defeat these mechanisms, and overall the intelligence gained from decrypted German communications gave the Allies an edge in many theaters, with some historians arguing it shortened the war by years.

===== (10:00) to (12:00) =====

After the war, Turing returned to academia, and contributed to many early electronic computing efforts like the Manchester Mark 1, which was an early and influential stored program computer. But his most famous post-war contribution was to artificial intelligence, a field so new that it didn't even get that name until 1956. It's a huge topic, so we'll get to it again in future episodes. In 1950, Turing could envision a future where computers were powerful enough to exhibit intelligence equivalent to, or at least indistinguishable from that of a human. Turing postulated that a computer would deserve to be called intelligent if it could deceive a human into believing that it was human. This became the basis of a simple test, now called the Turing Test.

Imagine that you are having a conversation with two different people, not by voice, or in person, but by sending typed notes back-and-forth. You can ask any questions you want, and you get replies, but one of those two people is actually a computer. If you can't tell which one is human and which one is a computer, then the computer passes the test. There's a modern version of this test called a Completely Automated Public Turing Test to tell Computers and Humans Apart, or CAPTCHA, for short. These are frequently used on the internet to prevent automated systems from doing things like posting spam on websites. I'll admit sometimes I can't read what those squiggly things say. Does that mean I'm a computer?

Normally in this series, we don't delve into the personal lives of these historical figures, but in Turing's case, his name has been inextricably tied to tragedy, so his story is worth mentioning. Turing was gay in a time when homosexuality was illegal in the United Kingdom and much of the world. An investigation into a 1952 burglary at his home revealed his sexual orientation to the authorities, who charged him with gross indecency. Turing was convicted and given a choice between imprisonment, or probation with hormonal treatment to suppress his sexuality. He chose the latter, in-part to continue his academic work, but it altered his mood and personality. Although the exact circumstances will never be known, it's most widely-accepted that Alan Turing took his own life by poison in 1954.

===== (12:00) to (13:05) =====

He was only 41. Many things have been named in recognition of Turing's contributions to theoretical computer science, but perhaps the most prestigious among them is the Turing Award, the highest distinction in the field of computer science, equivalent to a Noble Prize in Physics, Chemistry, or other sciences. Despite a life cut-short, Alan inspired the first generation of computer scientists, and laid key ground-work that enabled the digital era we get to enjoy today. I'll see you next week.



Alan Turing: Crash Course Computer Science #15

Crash Course: Computer Science

<https://youtube.com/watch?v=7TycxwFmdB0>

<https://nerdfighteria.info/v/7TycxwFmdB0>

Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like Gross Science, ACS Reactions, and The Art Assignment. This episode was filmed at The Chad & Stacey Emigolz Studio in Indianapolis, Indiana, and it was made with the help of all these nice people, and our wonderful graphics team, Thought Cafe. Thanks for watching, and try turning it off and then back on again.