LOGAN: Alright, so first what was the importance of the Entscheidungsproblem in Turing's work and how was Turing's approach different than that of his contemporaries (for example Alonzo Church)?

MAIN: Well, I think Turing is classed in the 1930s more as a mathematician because there was no computer science. And his work stems from the work of Bertrand Russell and Kurt Gödel both of which were looking at mathematics at systems that had reflexive qualities. So when you have a system with reflexive qualities within the system itself you're asking questions about that system. And so Bertrand Russell applied this to logic and Kurt Gödel also applied it to logic and arithmetic. And Turing applied it to algorithms. How are you going to write an algorithm that asks questions about other algorithms?

LOGAN: Alright, thank you. So why would you say the Turing Machine was such an important creation? Clearly it was the first example of this this theoretical computer, but also how did it change the world of computation? Because what I understand, and correct me if I am wrong, this was the first computer; the first theoretical idea for a computer. I mean, of course Babbage had his analytical and difference engine but…

MAIN: So, up until Turing there was a notion of a computation. In fact, there was even a word: a computer. But did you know what a computer was in the 1800s?

LOGAN: From what I can remember it's a person.

MAIN: Exactly! Exactly! Yeah, so a computer in the 1800s was a person. And in some sense programmers would write sequences of steps for those people to follow to solve various mathematical problems; maybe something about the orbit of planets. What Turing did, though, was put that in a formal foundation. In other words he laid down rules that said if you have a computer program (he didn't call it that, but…), then it has to follow this form. And prior to that there was no way really to study the power of computing. So there was no way to say well, computer programs can do this thing, they can solve this problem, and they are unable to solve this other problem because there was no formal notion of what is a computer program.

LOGAN: So he basically made it so that it can be mathematical and logic-based.

MAIN: Exactly.

LOGAN: So before it was just this more ambiguous system, would you say?

MAIN: I would say more informal. And so he turned it into a formal system so that once it was formalized you could then…he turned computer programs into something that could be formally studied. Earlier you mentioned Alonzo Church. Twenty years after Turing's work, Church proposed that, in fact, any computer program, no matter what model it was, was equivalent to what Turing had posed in the 1930s. And that's the Church-Turing Thesis or Hypothesis. It's not something that can be proved, but something that we don't have any counterexamples to.

LOGAN: And so Church and Turing were both working on competing ways to solve the Entscheidungsproblem. Well, not necessarily competing, but they were different approaches; they were taking different approaches. But then you mention this Church-Turing Thesis, is that correct?

MAIN: Yeah, so again the Church-Turing Thesis is saying that we are just going to study one model of computation; say the Turing Machines, for example. And we are going to prove things about this model: about what it can do, about what it can't do. This isn't very interesting unless the things you prove are widely applicable. And so that's what Church was saying. Church said 'Hey, I don't care whether you're programming in C++ or python or some future language, but the results that we prove about Turing Machines are going to be applicable.'

LOGAN: Yeah, so basically Turing's ideas are going to be these universal ideas.

MAIN: Universal, exactly.

LOGAN: And would you agree that this is this very important idea that this sort of allows us to do computer science, because Turing sort of built the foundation for this?

MAIN: Exactly, it turned something informal…Turing's work turned an informal notion of what a program is into a science; into a mathematical science that you can prove things now about the power of programs.

LOGAN: So do you know much about Turing's work on the Enigma and …his work on computation enabled him to crack the Enigma code?

MAIN: Unfortunately I haven't studied Turing's work on the Enigma, I mean I know the popular stories on what he did, but I can't really help you there. I could talk a little bit about his later work, after WWII, on artificial intelligence, as he's seen as one of the founders of that as well. We talked about that briefly. Is that something you would like to hear?

LOGAN: If you want to, go ahead. I mean, we can use all of this, so...

MAIN: Well Turing was very creative and very insightful in boiling things down to their essential points. And you see that in his work in the 1930s when he was able to formalize computer science. But you also see it in his work after WWII, where the beginnings of artificial intelligence were being studied by computer scientists. The computer scientists wanted to know what it would take to build something that was intelligent. And then again Turing tried again to in some sense formalize that; and tried to say 'what do you mean by intelligent?' And so what we now call the Turing test is a very to the core way of answering that question. And so, in those days in those days you didn't have computer screens, you had teletypes. So we proposed: well were are going to put a person in a room with two teletype machines and the person will type

questions on either machine and an answer will be typed back. At the end of one machine is another person is another person, and at the end of the other machine is a computer. If a typical person can't tell which is which, then that computer program is acting intelligently. So he was trying to boil down the notion of what is intelligent to a test that could be run, rather that something more ephemeral that might be reproducible or that might not be.

LOGAN: Do you know anything about his life?

MAIN: I know about his work ideas, I know that he was homosexual, and that in those decades that was not an easy thing for anybody. But I can't really tell you those details; I'm sure in your own study you know more about that than I do.

LOGAN: Would you say that any part of his life that you are familiar with affected how he went about his work?

MAIN: I can't really help you there.

LOGAN: No worries. Some of these questions were…we interviewed a computer science professor for the specific reason that you know a lot about computation that we could not get from people of other disciplines. How would you say the modern computer, on that I am actually sitting in front of…how would you say that is influenced by Turing?

MAIN: …The hardware was not influenced by Turing. To a large extent …the hardware had a much bigger influence probably by Turing's contemporary Von Neumann. Turing, though, was able to lay the foundations for the first programming languages. Every procedure that you see run on your computer today is written in a programming language. There are only about less than half a dozen control structures that appear in programming languages. You can do things in sequence: do one thing, do another thing, do another thing. You can make a choice; you can chose between doing two or more things based on the state of the program. And you can repeat things until a certain condition occurs. Maybe just those three things are what underlie every program, and those three things were in the Turing Machine that he proposed as the formal model of computation.

LOGAN: And that's what you say you can still get down to? When he formalized it, he was able to successfully distill it down to the most basic steps, right?

MAIN: That's right, so Turing was able to look at all the different formulations of computations and distill it down to these control structures that were then put into the Turing Machine. By the way, you wouldn't want to program a Turing Machine; it's done at too low a level. And so on top of that today we have much higher level structures, but anything they do could be done by a Turing Machine.

LOGAN: And that's the difference between high-level and low-level languages.

MAIN: So a Turing machine, in its action, is much closer to very low-level, assembly-level, programming than to higher-level programming of a structured language like C++ for example. But, the mathematical results of the Turing machine also apply to the higher level language.

LOGAN: Turing's work in general, where do we see it today?

MAIN: I see Turing's work today as a piece of the first 30-40 years of the 20th century that essentially destroyed mathematics, physics, and even art. If you look at the first decades of the 20th century, we came into the decade with a very certain knowledge that physics was deterministic, that physics was Newtonian. We came into the decade with a hope that mathematics; that logic in particular, could be built into a system that didn't have any holes in it. In computer science, of course, we came into the decade with it out being formalized, but you might say that we came into it thinking algorithms are capable of solving anything. And one-by-one those things were destroyed. So Bertrand Russell destroyed the hope of logic being complete, and that was destroyed, in fact formalized, but Gödel in the 1930s. Quantum mechanics destroyed classical physics. In art you had people like Ionesco and Escher going completely away from realism. Turing destroyed that hope for computer science. Essentially what all this is saying is that within a given system, the system itself is going to have limitations.

LOGAN: Would you agree that his work with the Halting Problem destroyed that; the fact that a machine can't figure out when to stop?

MAIN: What is it about Turing's work that shook the foundations of computation? The answer is that Turing, like Gödel, looked at computer programs that were analyzing themselves: computer programs can ask questions about computer programs. And what he was able to show is that no computer program can be given as its input another computer program and always determine correctly whether that other computer program is going to halt or go into an infinite loop. But that was only the key starting point. If you go beyond that, over the next 20 years that was expanded, and there was a theorem called Rice's Theorem 1950s. It essentially says this: anything you want to ask about the functional behavior of a computer program cannot be answered by a computer program.

LOGAN: The long-standing implication of this is that we will never be able to have computers that program themselves?

MAIN: No, we can have computers that can program themselves. The long-standing implication is that within any system, that system is going to be incomplete if it tries to ask questions about itself. The exact same thing applies to people; you can take Turing's result and apply it to a larger system: what happens if we want people to answer certain questions about themselves? Gödel, Escher, Bach…It talks about how all these things were so certain in 1900 that they were destroyed over the next few decades and why that's the case.

LOGAN: Do you want to go any further and to talk about how it changed the mathematics and the logic and the creel of computer science? And would you say that his work introduced the idea of using mathematical thinking to solve problems in everyday life? Not necessarily people using it in their homes, but all of the sudden we have computers that can do this for us because of his work. Would you agree?

MAIN: Without Turing there wouldn't be a way to analyze what a computer is capable of doing and, once you have written a program, there wouldn't be ways of analyzing properties of that program: how long is it going to take, is it correct, what does it even need to be correct?

LOGAN: How do you think Turing's work laid the foundations for software engineering?

MAIN: The answer is that, suppose, as an analogy, you want to prove things about nutrition. You want to start proving this about how different nutrients affect the performance of the body. Well you can do, perhaps, experimental work, but as in any science you can do theoretical work. Theoretical work means you build a model of what you're studying and based on the reasoning of the model, you make conclusions about the physical events you are studying. Turing was the first theoretical computer scientist, in that he built that model and then he could reason about it. Now are your conclusions about your model valid in the real world? Well it depends on a lot of things: how well does the model actually fit the real world. In computer science, Turing's model fits computer programs perfectly.

LOGAN: Is this how we can run entire computer programs?

MAIN: I say, Turing's work didn't directly talk about analyzing the run time of computer programs. Initially it only talked about is a certain problem solvable by a program or not. So a kind of yes/no: you can solve it or you can't. But it laid the foundation for the work in the 1960s and the 1970s in which you want to analyze how much memory is needed or how much time is needed to solve a particular problem.

LOGAN: Going off of what you just said, the whole idea of it is solvable or not that's key, and that's what allowed the wide-scale adoption of computers into more than just this little theoretical realm. If you do not know if something is solvable, you can't use it for anything else.

MAIN: So the question of whether or not something is solvable is always going to be the first step. If my boss comes to me, and I am a computer scientist, and he says, 'Oh I've got this problem and I want you to write a program to solve it.' For example he might say to me, 'I've noticed that our computers are spending a lot of CPU cycles spinning their wheels; they seem to be into infinite loops. I want you to solve this problem; I want you to detect those infinite loops before they happen.' Well I can turn to Turing's work and say, 'Hey, I would like to do that, it's not that I am being contrary. It's just that we prove that solving that particular problem is not possible.' And that's a very strong position to be in. Now, once you have concluded that a problem is solvable, well you still have the question of: well is it solvable in a reasonable amount

of time with a feasible amount of memory? And so that then becomes a secondary question. Ok now my boss comes to me and says, 'You know we are spending a lot of money on gasoline with our salesmen, and each one has a different route he/she has to travel. What we want to be able to do it to make sure each salesman is travelling the minimum distance to visit their clients.' You're not in quite such a strong position there, you can't say, 'Hey, I can't solve that problem.' Because that's a solvable problem, but it's a problem that we don't know how to solve efficiently; it's called the travelling salesman problem. Nobody knows how to solve that problem efficiently. Now there might be an efficient solution, but before that computer scientists and mathematicians had been looking at that problem and equivalent problems literally for centuries and hadn't come up with efficient solutions. So you can say to your boss, 'All the know algorithms for solving that problem are not efficient.' Turing didn't work on that kind of problem directly, but he laid the foundation for that.

LOGAN: This is a very broad question; take it however you want: how do you think our lives would be different without Turing's work?

MAIN: You can if you want, say let's say Turing hadn't done that and nobody had laid the theoretical foundations for computers, we wouldn't have computer programs; we wouldn't have computer systems, we wouldn't be able to write complex computer programs, we wouldn't have software engineering. Our day-to-day lives would be different in how you learn things; in how information is held and retrieved. I was reading a story last week that was written in 1877 by a Spaniard whose name was Enrique Gaspar. And it was the first example of a story with a time machine before H.G. Wells. And in the first chapter he's telling a funny story, but he keeps throwing in these historical facts. And I keep having to ask myself, 'If I was writing that story today, I would continually be retrieving information from my computer from the whole World Wide Web where we've got that information at our fingertips. He had to have it in his brain, or in a book on his shelf. So I think that the biggest… We are now in the age of information and that would not be there without a foundation for computer science.

LOGAN: Turing has been hailed as the father of modern computer science; could you give us your definition of computer science and go on to talk about Turing's place in that?

MAIN: My definition of computer science: it's the study of algorithms; it's the study of procedures that can solve problems. The study involves both theoretical studies, what can and can't be solved by and algorithm, and practical studies, how one goes about writing a program that you have confidence will give you the right answer. So Turing laid the foundations the foundations for the first part, the theoretical computer science: what can be solved by computer programs. When you have a computer program, how do you go about solving it? How do you study the class of computer programs? That's what he laid down. Then after that, particularly in the late 60s and throughout the 70s, the science of software engineering came into place.

LOGAN: But without Turing, there wouldn't have been computer science as a field?

MAIN: There wouldn’t be anything to study without Turing. You do not lie down and say, ‘Here’s what a computer program is.’ But you don’t know what you’re studying. It’s like a biologist who doesn’t know what life is.

MAIN: What do you get when you throw a Turing Machine down an icy hill?

LOGAN: You are just going to have to tell me this one.

MAIN: A Halting Problem. Haha.