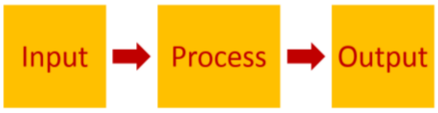
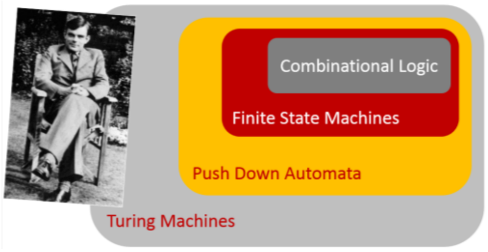
A paired or group activity that demonstrates the behaviour of a simple Turing machine, in a child friendly fashion.

Preparation required:
Chocolate buttons (or counters). Minimum 7 dark, 15 white and 6 coloured lollies (or similar) per group
Chocaholic Turing Machine (cs4fn article) per child
Chocaholic Subtraction Machine Rules per group

Models of Computation



What is a computer, and what does it means to compute? Our starting point is a general model of how something is computed: taking some input, doing something to it before the results of that process are output. We have seen how finite-state automata, are mechanisms for expressing that computational process, and can describe simple programs as state transition diagrams. In computer science, we can think of finite-state automata as abstract models that helps convey a computational process.

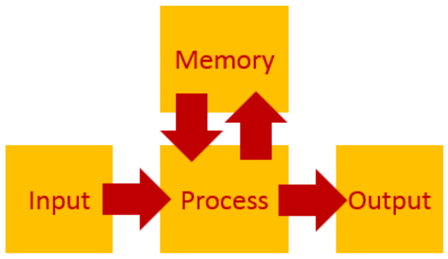


Wikipedia has a good article on Automata Theory, with a diagram that puts combinational logic and finite-state machines at the heart of the theory (goo.gl/RAe5yz). We can demonstrate many simple processes in terms of collections of logical constructs (AND, OR and NOT). We see that when looking at the behaviour of specific circuits, for example an adding circuit, within a computer. We call that combinational logic.

Finite-state machines are a more powerful abstraction – the next layer up in our models of computation. But there are things they cannot do. You can't devise a finite-state machine (FSM) to check if a phrase using brackets always has a closing bracket to match every opening bracket. Similarly no FSM can check if a word is a palindrome (spelt the same backwards). These sort of problems rely on pairing up. We can write FSMs that pair up specific numbers of items, but we can't write a general machine for any number of symbols. The presentation considers a FSM that tries to pair up cups and saucers. Notes for the animation are provided with the slide. Can you see the problem?

The name gives it away really. A FINITE state machine cannot have an INFINITE number of states to handle an indeterminate number of inputs. It can't 'keep count'. It has no memory of what has gone before, it only knows what state it is in. For this reason, more flexible models, known as Push Down Automata were conceived. In essence these are finite-state machines connected to a memory stack. Now we can just have two states for our Cups and Saucers machine, recognising a surplus of either. Each time a cup comes in, we simply remove a saucer from the top of the stack. When the stack is empty, we return to the Ready state.

Think back to the original model of a computer – the Input, Process, Output diagram. A Push Down Automaton can be thought of as this, with some form of memory attached to allow it to 'keep count'. As we've just seen, that memory is a 'stack' which limits what it can do. It can only retrieve things from the top of the stack.



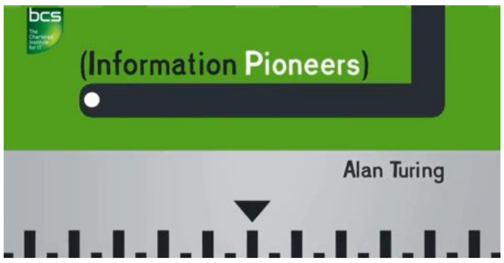
A more general model, which allows access to any element of memory is known as a Turing Machine … and it has a remarkable history. In computer science, the different levels of the diagram above are known as the 'Chomsky hierarchy'. There's more to it than we have considered here, but if you'd like to delve deeper, Computerphile is the place to start (youtu.be/224plb3bCog). If you are likely to be teaching A level at any point all the related Computerphile videos are worth watching.

Turing Machines

The Turing Machine was conceived in the 1930's by Alan Turing, a mathematical genius best known for his role in breaking the Enigma codes in the Second World War. Whilst grappling with whether it was possible to determine in advance if a mathematical problem was solvable, various mathematicians tried to develop a formal definition of an algorithm. Several ideas were developed independently

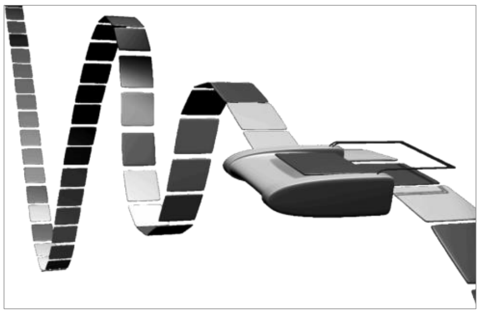
Despite the differences, there was nothing to choose between them – they were all shown to be equivalent to each other – an incredible coming together of ideas. Alan Turing conceived the idea of a computing machine, which was the most intuitive. He based it on the idea of a human 'computer' – someone who did calculations in a series of steps, using a scratch pad to jot down parts of the working out.

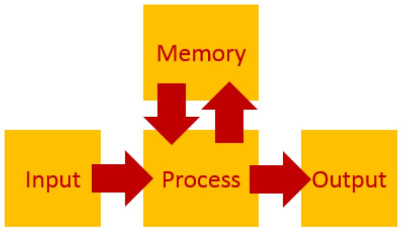
It is remarkable for two reasons. First, he had the idea long before any electronic computers had ever been built. Secondly, his model has been shown to be as powerful as any computer built since! By powerful, we don't mean fast, we mean it can compute anything any modern computer can compute. This is the heart of Computer Science. Turing machines can describe general computations. Having a model of computation allows us to reason about what sort of problems are computable, and what are not. Put another way, if something can be computed, a Turing machine can be designed to do it.



So what did Turing envisage? Actually, it is pretty simple. It consists of just two things: a very long tape that can move in either direction, and a mechanism in the middle that can look at the bit of tape beneath it, read what is on it, erase and write other symbols on if needed. A short video, ideal for students, explains the place of the Turing machine in the development of Alan Turing's ideas (youtu.be/gROuKl82BTk).

We can simplify a Turing Machine to a schematic: a tape and a read/write head. Superficially, it seems very different from our model of input-process-output, but consider the parts. The tape can have symbols written on it, either before the machine starts, or when it is running. So we have some means of inputting information. The contents of the tape, whilst it is running, are a store, just like memory in a modern computer. When the machine stops what is left on the tape can be considered the output.





'What it does whilst it is running is determined by the behaviour of the Read / Write head. The behaviour can be described by a finite-state machine. It acts like the computer processor. In fact we can think of modern computer processors as finite-state machines, albeit very complicated ones. They contain billions of logic gates which combine to put a machine in a particular state on each clock cycle.

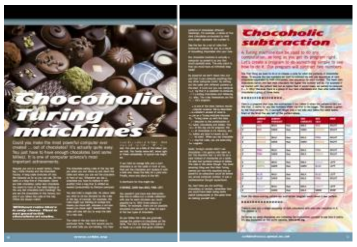
It may seem incredible to children that such a simple machine can do anything a powerful modern computer can, so it is worth stressing two important caveats:

- It has as much memory as it needs (the tape goes on for ever)
- It has a much running time as it needs

The second of these caveats is important to emphasise. Turing machines are slow – very slow!

At KS3, the implications of a Turing Machine for investigating the limits of computation aren't really important. What is, is to make the point that in order to understand how computers compute, we can abstract away the specific details of modern computers to focus on the essential process. We can make this point by demonstrating how a machine like this can do some simple computation. To do that in a fun way, we'll fall back on our traditional prop of bribing children with sweets.

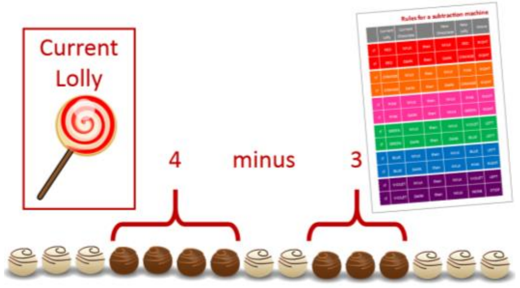
A Chocaholic Turing Machine



This activity comes from cs4fn, and was first outlined in their special issue to celebrate the centenary of Alan Turing's birth. The whole magazine (issue 14) is well worth reading (goo.gl/wYMZAM) if you wish to know more about his life and work. A copy is included in the resources.

Split the class into small groups or pairs. You need a large supply of chocolate buttons, both white and dark. Alternatively, use counters (two colours), but it isn't as much fun. For the example, a minimum of 7 dark and 15 white are needed for each group, more if other numbers are tried. You also need six different coloured lollies per group. These do not have to be edible and could be made of card. Our example uses Red, Orange, Violet, Blue, Green and Pink. The activity, with an introductory article can be found on pages 10 & 11 of cs4fn (above). Encourage students to read the article first.

The chocolates represent the starting tape of the Turing Machine. Arrange the opening chocolates in a line on the table. The presentation provides a walk-through of the opening moves. We are going to implement a subtracting machine. The dark chocolates represent the two numbers we have input, so we are going to perform the calculation 4 minus 3. Representing numbers in this way is known as the unary number system.

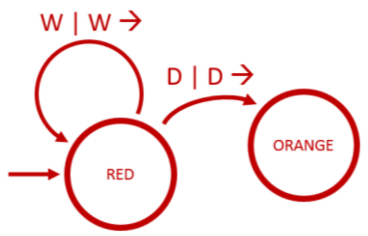


Our tape distinguishes between the two numbers by having some white chocolates between them. Each group also needs the rules for the Subtraction Machine. One person should be assigned the job of holding the Current Lolly. They start by holding the Red Lolly. Starting from the leftmost chocolate, students implement the rules outlined on the handout. The presentation includes prompts in the slide notes so the animation can proceed in response to answers from the children.

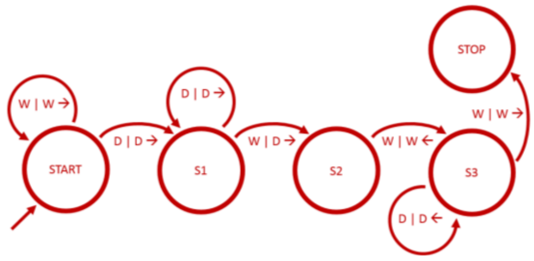
For the first few cycles the Current Lolly is RED, and the Current Chocolate WHITE so the Head moves right one place on each cycle. Eventually the Current Chocolate is DARK. It will remain DARK but the Lolly changes to ORANGE and the Head moves right. Because the lolly has changed, we now need to look at the rules for an ORANGE lolly. You might wonder if anything is ever going to change, but it will.

When a chocolate changes colour, remove the one from the tape and EAT IT! Replace it with the correct colour from the supplies. Children can take turns to move the head so they get an equal chance of eating a chocolate. Challenge them to work out the pattern of chocolates when the whole process is finished.

A further animation allows demonstration of the final two steps. The final output is displayed on the tape. One dark chocolate represents the result of the calculation (4 – 3). Ask students to set up their own subtraction using different numbers. It will work so long as the first number is larger than the second.

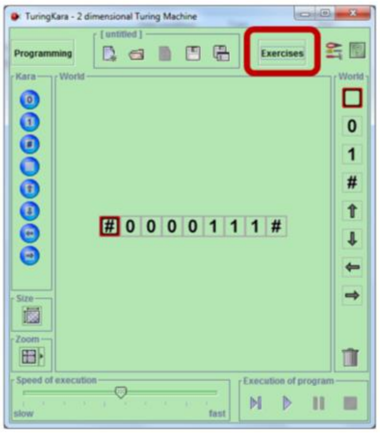


Ask students to express the rules as a finite-state machine. The presentation develops the first two states and notation for the transitions. The transition has to represent the initial read (input). It also indicates the tape contents after the action and the direction of travel of the head (output). Having modelled the first rule, written in the correct way, see if students can add the second rule. The subsequent slide gives the complete FSM.



A final challenge, for able students might be to devise a set of rules (or FSM) to increment a unary number by one. They can use the same conventions, dark chocolates bounded by white chocolates for the starting number. The solution (shown) can be worked through as a class example. To check understanding, one more transition is needed to trap a faulty layout – can anyone spot it? If students are stuck consider the transitions from S2.

Taking It Further



There is a version of Kara designed to implement Turing machines. Turing Kara is rather more challenging than Kara, the programmable ladybird, explored in a previous session. However, for very able students, it may provide good extension challenges. Like Kara, it comes with a set of exercises and the first two are accessible to able students at this age. Each challenge comes with its own worlds. They restrict the 'tape' to a short sequence which avoids a lot of potential confusion (in later challenges the tape can be represented as a grid). It also introduces 'bounding symbols (#), and uses 3 symbols on the tape, a 1 and 0, but also a blank square.

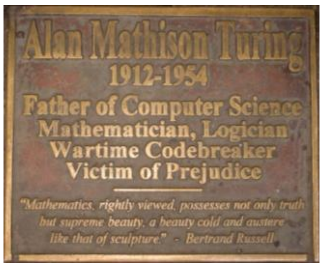
Finally, two more pointers, specifically for teachers. First, Rob Mullins from the Cambridge Computer Lab provides detailed instructions to use a Raspberry Pi to build a Turing machine that uses LED's and an interface developed in Python (goo.gl/ouz9Du). Second, on the centenary of Alan Turing's birth, Google had a wonderful doodle (goo.gl/bvFpbO) with 12 brain teasing puzzles. If you can't figure out what to do, goo.gl/DRCEM8 is an excellent explanatory article.

In Conclusion

Alan Turing demonstrated that anything that could be computed, could be computed by a Turing machine. Through this computer scientists could explore what was computable, and what wasn't. Unlike the common sense idea that computers will, one day, be able to do anything, computer science can demonstrate there are things computers will never be able to do. This is the significance of 'the halting problem', something you may have heard about, which is introduced at A Level. If a program is running and seems to be stuck, how do we know whether it is just taking a long time to perform a calculation or is stuck in an infinite loop? One will eventually halt, having performed the calculation, the other will go on for ever. Turing proved computers can't inspect every program and say, conclusively if they will run to completion. It is an example of an uncomputable problem, proving there are things that cannot be computed.

If this seems a little obscure, Turing also offered a more fundamental insight. In a Turing machine, the data is put on the tape. The purpose of the machine is captured in the finite-state machine that processes that data, like our subtracting machine. But Turing went on to show how the instructions for the finite-state machine could also be encoded as symbols on the tape. By running up and down the tape, the machine could read an instruction, then find the data and do something to it, before going to read the next instruction, and so on. He called this the Universal Machine.

We can conclude where we started. Turing envisaged a general purpose machine that could be programmed by putting different instructions in memory. He envisaged software before any software had been written. But since any Turing machine could have the instructions of any computational sequence put on its tape, all computers were equivalent in what they could, and could not do. Any machine could 'emulate' any other. In a sense, it is the ultimate 'computational abstraction'. A theoretical model of a computer that describes all modern computers … developed before any computer had ever been built.


Alan Mathison Turing
1912-1954
Father of Computer Science
Mathematician, Logician
Wartime Codebreaker
Victim of Prejudice
"Mathematics, rightly viewed, possesses not only truth but supreme beauty, a beauty cold and austere, like that of sculpture" - Bertrand Russell

Following his code breaking exploits in the war Turing went on to play a key role in designing the early computers. He laid the foundations for the field of Artificial Intelligence. At the time of his death he was exploring computational patterns occurring in nature, a field that is really just developing now. Quite simply, he was ahead of his time. He died young, a result of being hounded because he was gay. Turing was the father of Computer Science, but his life and death provide many other lessons for children today.