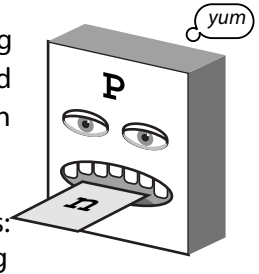
## The Halting Problem

Alan Turing simultaneously demonstrated limits of mathematical truth, *and* invented the idea of a general purpose computer, *and* invented programming, *and* invented computer science, all at the age of 23.!

One part of his idea is that there can be mathematical problems that can *never* be solved by a mechanical process. It's not that people are too dumb to design a mechanical process to solve them. To the contrary: **no process can ever exist!** And he showed how to *prove* this!!




The starting point is his Halting Problem: A given program P, fed some given data n, might run forever, or it might halt.

Simple enough. So our question is: knowing a program, and knowing what data it's being fed, how can we tell whether or not the program runs or ever or eventually halts?

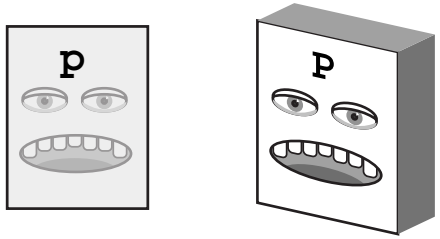
An obvious thing to try: Run the program and find out. If the program halts, all we have to do is wait to find out.

*But what the program isn't ever going to halt? We'll never learn that this way! Even if we've waited one million years, maybe it'll halt if we'd just wait another two minutes. Or maybe two minutes after that.... or maybe...*

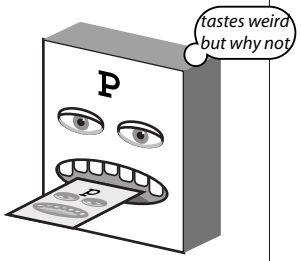
In many cases, we can work it out just by examining the program. In practice, this is what programmers actually do. But we are asking here for a fully *general*, *mechanical* procedure for deciding this question. A program!

This hypothetical "Halting Program" H would take in as data a program P and its input n, and answer:

Will P halt on input n?




(P itself is data in this context; let's make this distinction by writing P when it is a working program and p when it is data.)



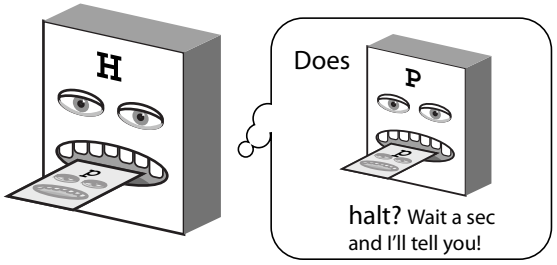

Let's simplify this just a bit. It'd be a lot easier if we just fed P into H by itself, and ask Turing's

**"Halting Problem":**
Will P halt on input p?

Fair enough. But incredibly, Turing proved *No Such* H *Can Exist*. No general method of deciding whether or not a given program halts can ever exist! He proved the Halting Problem is **"undecidable"**.

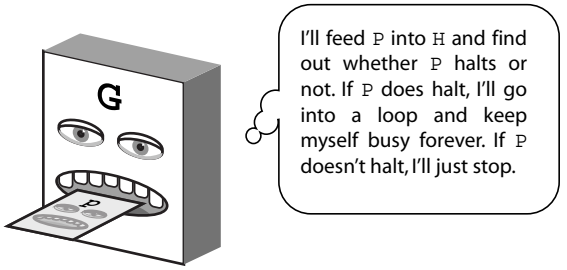
That there are problems that will remain forever beyond mathematical analysis is truly staggering. Even more so, that *mathematics can actually prove, mathematically, that such problems exist!*

How does it work? It's surprisingly easy! Suppose some crank comes in claiming to have invented a halt-testing program H. This H would take as input p, and work out whether or not P halts on p.




Then all we have to do is find a way this is nonsensical. We can dismiss all cranks if we can always make such an H into nonsense. (This style of proof is called "Proof by contradiction": assume the opposite of what you want to prove and show this reduces to nonsense.)

Here comes a crank, claiming to have discovered H. We create a short-circuit like this. We make a new program G that uses (and abuses) H.




G takes in p, and feeds it into H. H will chew on this for a while, and then spit out an answer: either

P *will halt on input* p or
P *will not halt on input* p

G then does something with this information. In the first case, if P does halt on input p, our G starts cycling around in an infinite loop, running forever. In the second case, if P doesn't halt, G just stops.

Now for the Kicker.
Feed G into itself!! What does G do on input g?

Either G will halt on input g or it will not halt on input g.

Let's see what happens:

We feed g into G, which asks H whether or not G halts on input g. If H says "yes, G halts" then G goes into an infinite loop and doesn't halt! If H says "no, G does not halt" then G stops.

**ZAP.** No such G can exist! No such H can exist! Turing's Halting Problem is Undecidable!