Created by logician Bertrand Russell.

Village with just 1 barber, all men clean-shaven. Barber announces:
“*I shave all and only those men who do not shave themselves.*”

Who shaves the barber?

Case 1: It’s the barber.
Case 2: Somebody else.

Cannot answer that question in either case! Paradox!!!
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \ \forall x \ (x \in y \iff P(x)) \]  \hspace{1cm} (1)

\( y \) is the set of elements that satisfies the proposition \( P(x) \).

\( P(x) = x \not\in x \).

There exists a \( y \) that satisfies statement 1 for \( P(\cdot) \).

Take \( x = y \).

\[ y \in y \iff y \not\in y. \]

Oops!
Is this stuff actually useful?

Verify that my program is correct!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

\[ \text{HALT}(P, I) \]

- \( P \) - program
- \( I \) - input.

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.

Text string can be an input to a program.
Program can be an input to a program.
Implementing HALT.

$HALT(P, I)$

$P$ - program
$I$ - input.

Determines if $P(I)$ ($P$ run on $I$) halts or loops forever.
Run $P$ on $I$ and check!
How long do you wait?
Halt does not exist.

\[ \text{HALT}(P, I) \]
\[ P \text{- program} \]
\[ I \text{- input.} \]

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.

**Theorem:** There is no program HALT.

**Proof Idea:** Proof by contradiction, use self-reference.
Halt and Turing.

**Proof:** Assume there is a program $HALT(\cdot, \cdot)$.

Turing($P$)
1. If $HALT(P, P) =$ “halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing.
Can run Turing on Turing!

Does Turing(Turing) halt?

Turing(Turing) halts
$\implies$ then $HALT(Turing, Turing) =$ halts
$\implies$ Turing(Turing) loops forever.

Turing(Turing) loops forever
$\implies$ then $HALT(Turing, Turing) \neq$ halts
$\implies$ Turing(Turing) halts.

Contradiction. Program HALT does not exist!
Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>...</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Halt($P$,$P$) - diagonal.
Turing - is not Halt.
and is different from every $P_i$ on the diagonal.
Turing is not on list. Turing is not a program.
Turing can be constructed from Halt.

Halt does not exist!
A Turing machine.
- an (infinite) tape with characters
- be in a state, and read a character
- move left, right, and/or write a character.

Universal Turing machine
- an interpreter program for a Turing machine
- where the tape could be a description of a ...

Now that’s a computer!
Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... a programming language!!!
Just like Python, C, Javascript, ....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
  Inconsistent: A false sentence can be proven.
  Incomplete: There is no proof for some sentence in the system.

Along the way: “built” computers out of arithmetic.
  Showed that every mathematical statement corresponds to an ....natural number!!!!
Summary: computability.

Computer Programs are interesting objects.
  Mathematical objects.
  Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Example: no computer program can tell if any other computer program HALTS.

Proof Idea: Diagonalization.
  Program: Turing (or DIAGONAL) takes $P$.
  Assume there is HALT.
  DIAGONAL flips answer.
    Loops if $P$ halts, halts if $P$ loops.
  What does Turing do on turing? Doesn’t loop or HALT.
    HALT does not exist!

More on this topic in CS 172.

Computation is a lens for other action in the world.