Next Topic: Undecidability.

- Undecidability.
Barber paradox.

Barber announces:

"The barber shaves every person who does not shave themselves."

Who shaves the barber?

Get around paradox?

The barber lies.
Barber paradox.

Barber announces:
“*The barber shaves every person who does not shave themselves.*”
Barber paradox.

Barber announces:
“The barber shaves every person who does not shave themselves.”

Who shaves the barber?
Barber paradox.

Barber announces:
“The barber shaves every person who does not shave themselves.”

Who shaves the barber?

Get around paradox?
Barber paradox.

Barber announces:
“The barber shaves every person who does not shave themselves.”

Who shaves the barber?

Get around paradox?
The barber lies.
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x(x \in y \iff P(x)) \quad (1) \]
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

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

\( y \) is the set of elements that satifies the proposition \( P(x) \).
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

$$\exists y \forall x (x \in y \iff P(x)) \quad (1)$$

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

$P(x) = x \notin x$.

Oops! What type of object is a set that contain sets?

Axioms changed.
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 $P(x)$.

$P(x) = x \notin x$.

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

Oops!

What type of object is a set that contain sets?

Axioms changed.
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x (x \in y \iff P(x)) \quad (1) \]

\( y \) 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 \( x = y \).
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x (x \in y \iff P(x)) \quad (1) \]

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

\( P(x) = x \notin x. \)

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

Take \( x = y \).

\[ y \in y \iff y \notin y. \]
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x (x \in y \iff P(x)) \quad (1) \]

\( y \) 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!
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

\[ \exists y \forall x (x \in y \iff P(x)) \] (1)

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

\( P(x) = x \notin x. \)

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

Take \( x = y \).

\[ y \in y \iff y \notin y. \]

Oops!

What type of object is a set that contain sets?
Russell’s Paradox.

Naive Set Theory: Any definable collection is a set.

$$\exists y \forall x (x \in y \iff P(x))$$ (1)

$y$ is the set of elements that satisfy the proposition $P(x)$.

$P(x) = x \notin x$.

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

Take $x = y$.

$$y \in y \iff y \notin y.$$

Oops!

What type of object is a set that contain sets?

Axioms changed.
Changing Axioms?

Goedel:

Any set of axioms is either

Concrete example:

Continuum hypothesis: "no cardinality between reals and naturals."

Continuum hypothesis not disprovable in ZFC (Goedel 1940.)

Continuum hypothesis not provable. (Cohen 1963: only Fields medal in logic)

BTW:

Cantor .. bipolar disorder..

Goedel .. starved himself out of fear of being poisoned..

Russell .. was fine...

..but for ... two schizophrenic children..

Dangerous work?

See Logicomix by Doxiadis, Papadimitriou (professor here), Papadatos, Di Donna.
Changing Axioms?

Goedel:
Any set of axioms is either
inconsistent (can prove false statements) or
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:

BTW:
Cantor: bipolar disorder.
Goedel: starved himself out of fear of being poisoned.
Russell: was fine... but for two schizophrenic children.

Dangerous work?
See Logicomix by Doxiaidis, Papadimitriou (professor here), Papadatos, Di Donna.
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinaty between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Changing Axioms?

Goedel:
Any set of axioms is either
inconsistent (can prove false statements) or
incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinatity between reals and
naturals.”
Continuum hypothesis not disprovable in ZFC
(Goedel 1940.)

Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:

BTW:

Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell .. was fine... but for ... two schizophrenic children..

Dangerous work?
See Logicomix by Doxiaidis, Papadimitriou (professor here), Papadatos, Di Donna.
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinatity between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Changing Axioms?

Goedel:
Any set of axioms is either
inconsistent (can prove false statements) or
incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and
naturals.”
Continuum hypothesis not disprovable in ZFC
(Goedel 1940.)

Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable. (Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable. (Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable. (Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell .. was fine...
Changing Axioms?

Goedel:
Any set of axioms is either
inconsistent (can prove false statements) or
incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and
naturals.”
Continuum hypothesis not disprovable in ZFC
(Goedel 1940.)

Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell .. was fine.....but for ...
Changing Axioms?

Goedel:
Any set of axioms is either
inconsistent (can prove false statements) or
incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and
naturals.”
Continuum hypothesis not disprovable in ZFC
(Goedel 1940.)

Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor .. bipolar disorder..
Goedel .. starved himself out of fear of being poisoned..
Russell .. was fine.....but for ... two schizophrenic children..
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable. (Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell .. was fine.....but for ...two schizophrenic children..
Dangerous work?
Changing Axioms?

Goedel:
Any set of axioms is either inconsistent (can prove false statements) or incomplete (true statements cannot be proven.)

Concrete example:
Continuum hypothesis: “no cardinality between reals and naturals.”
Continuum hypothesis not disprovable in ZFC (Goedel 1940.)
Continuum hypothesis not provable.
(Cohen 1963: only Fields medal in logic)

BTW:
Cantor ..bipolar disorder..
Goedel ..starved himself out of fear of being poisoned..
Russell .. was fine.....but for ...two schizophrenic children..
Dangerous work?

See Logicomix by Doxiaidis, Papadimitriou (professor here),
Is it actually useful?

Write me a program checker!
Is it actually useful?

Write me a program checker!
Check that the compiler works!
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input. 

$HALT(P, I)$
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

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

\( P \) - program
Is it actually useful?

Write me a program checker!

Check that the compiler works!

How about.. Check that the compiler terminates on a certain input.

\textit{HALT}(P, I)

\begin{itemize}
  \item \textit{P} - program
  \item \textit{I} - input.
\end{itemize}
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

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

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

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

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

Notice:
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

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

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

Notice:
Need a computer
Write me a program checker!

Check that the compiler works!

How about.. Check that the compiler terminates on a certain input.

\[ \textsc{Halt}(P, I) \]
\[
\begin{align*}
P & \quad \text{program} \\
I & \quad \text{input.}
\end{align*}
\]

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

Notice:

Need a computer

...with the notion of a stored program!!!!
Is it actually useful?

Write me a program checker!
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!)
Write me a program checker!
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.)
Write me a program checker!

Check that the compiler works!

How about.. Check that the compiler terminates on a certain input.

$HALT(P, I)$

$P$ - 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.)
Is it actually useful?

Write me a program checker!
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.
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

\( HALT(P, I) \)
\( P \) - 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.
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

\[ HALT(P, I) \]

- \( P \) - 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.
Is it actually useful?

Write me a program checker!
Check that the compiler works!
How about.. Check that the compiler terminates on a certain input.

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

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 \( (P, I) \)

- Program
- Input

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

Run \( P \) on \( I \) and check!

How long do you wait?

Something about infinity here, maybe?
Implementing HALT.

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

\[
HALT(P, I)
\]

\( P \) - program
Implementing HALT.

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

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

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

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

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

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

Run \( P \) on \( I \) and check!
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?
Implementing HALT.

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

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

Run \( P \) on \( I \) and check!

How long do you wait?

Something about infinity here, maybe?
Halt does not exist.
Halt does not exist.

\[ \text{HALT}(P, I) \]
Halt does not exist.

\[ HALT(P, I) \]
\[ P - \text{ program} \]
Halt does not exist.

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

- \( P \) - program
- \( I \) - input.
Halt does not exist.

\[ \text{HALT}(P, I) \]
\begin{align*}
P & \ - \ \text{program} \\
I & \ - \ \text{input.}
\end{align*}

Determines if \( P(I) \) (\( P \) run on \( I \)) halts or loops forever.
Halt does not exist.

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

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

**Theorem:** There is no program HALT.
Halt does not exist.

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

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

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

**Proof:** Yes!
Halt does not exist.

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

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

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

**Proof:** Yes! No!
Halt does not exist.

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

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

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

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

**Proof:** Yes! No! Yes!
Halt does not exist.

HALT\( (P, I) \)

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

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

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

**Proof:** Yes! No! Yes! No!
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:** Yes! No! Yes! No! No!
Halt does not exist.

$HALT(P, I)$

$P$ - program

$I$ - input.

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

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

**Proof:** Yes! No! Yes! No! No! Yes!
Halt does not exist.

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

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

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

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

**Proof:** Yes! No! Yes! No! No! Yes! No!
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:** Yes! No! Yes! No! No! Yes! No! Yes!
Halt does not exist.

$$HALT(P, I)$$

- $P$ - program
- $I$ - input.

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

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

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..
Halt does not exist.

\[\text{HALT}(P, I)\]
\begin{align*}
P & \text{ - program} \\
I & \text{ - input.}
\end{align*}

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

**Theorem:** There is no program \(\text{HALT}\).

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..
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:** Yes! No! Yes! No! No! Yes! No! Yes! ..

What is he talking about?
Halt does not exist.

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

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

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

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

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..  

What is he talking about?

(A) He is confused.
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 \( \text{HALT} \).

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..

What is he talking about?
(A) He is confused.
(B) Fermat’s Theorem.
Halt does not exist.

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

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

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

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..

What is he talking about?
- (A) He is confused.
- (B) Fermat’s Theorem.
- (C) Diagonalization.
Halt does not exist.

\[ HALT(P, I) \]

\[ P \] - program
\[ I \] - input.

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

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

**Proof:** Yes! No! Yes! No! No! Yes! No! Yes! ..

What is he talking about?
(A) He is confused.
(B) Fermat’s Theorem.
(C) Diagonalization.

(C).
Halt and Turing.

Proof:

Assume there is a program $\text{HALT}(\cdot, \cdot)$. Turing(P)
1. If $\text{HALT}(P, P) = \text{halts}$, 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 $\Rightarrow$ then $\text{HALTS(Turing, Turing)} = \text{halts} = \Rightarrow$ Turing(Turing) loops forever.

Turing(Turing) loops forever $\Rightarrow$ then $\text{HALTS(Turing, Turing)} \neq \text{halts} = \Rightarrow$ Turing(Turing) halts.

Contradiction.

Program HALT does not exist!

Questions?
Halt and Turing.

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

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 $\Rightarrow$ then $HALTS(Turing, Turing) =$ halts $\Rightarrow$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever $\Rightarrow$ then $HALTS(Turing, Turing) \neq$ halts $\Rightarrow$ $Turing(Turing)$ halts.

Contradiction.

Program $HALT$ does not exist!

Questions?
Halt and Turing.

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

Turing(P)
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.
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.
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 $HALT$. 

Questions?
Halt and Turing.

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

**Turing(P)**
1. If $HALT(P,P) = \text{"halts"}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program $HALT$.
There is text that “is” the program $HALT$. 
Halt and Turing.

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

Turing($P$)
1. If $HALT(P,P) = \text{“halts”}$, 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.
Halt and Turing.

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

**Turing(P)**
1. If $HALT(P,P) = \text{"halts"}$, 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!
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?
Halt and Turing.

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

Turing(P)
1. If $HALT(P, P) = \text{"halts"}$, 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 $\text{Turing(Turing)}$ halt?

Turing(Turing) halts
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 $HALTS(Turing, Turing) =$ halts
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 $HALTS(Turing, Turing) = \text{halts}$

$\implies$ Turing(Turing) loops forever.
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 $HALTS(Turing, Turing) = \text{halts}$

$\implies$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever
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 $HALTS(Turing, Turing) = \text{halts}$

$\implies$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever

$\implies$ then $HALTS(Turing, Turing) \neq \text{halts}$
Halt and Turing.

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

Turing(P)
1. If $HALT(P, P) = \text{"halts"}$, 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
\[ \implies \text{then } HALTS(Turing, Turing) = \text{halts} \]
\[ \implies \text{Turing(Turing) loops forever.} \]

Turing(Turing) loops forever
\[ \implies \text{then } HALTS(Turing, Turing) \neq \text{halts} \]
\[ \implies \text{Turing(Turing) halts.} \]
Halt and Turing.

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

\begin{align*}
\text{Turing}(P) \\
1. & \text{ If } HALT(P, P) = \text{“halts”}, \text{ then go into an infinite loop.} \\
2. & \text{ Otherwise, halt immediately.}
\end{align*}

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 \( \text{Turing(Turing)} \) halt?

\begin{align*}
\text{Turing(Turing) halts} & \implies \text{ then } \text{HALTS(Turing, Turing)} = \text{halts} \\
& \implies \text{Turing(Turing) loops forever.}
\end{align*}

\begin{align*}
\text{Turing(Turing) loops forever} & \implies \text{ then } \text{HALTS(Turing, Turing)} \neq \text{halts} \\
& \implies \text{Turing(Turing) halts.}
\end{align*}

Contradiction.
Halt and Turing.

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

$Turing(P)$
1. If $HALT(P,P) = \text{"halts"}$, 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 $HALTS(Turing, Turing) = \text{halts}$
$\implies$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever
$\implies$ then $HALTS(Turing, Turing) \neq \text{halts}$
$\implies$ $Turing(Turing)$ halts.

Contradiction. Program $HALT$ does not exist!
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 $HALTS(Turing, Turing) = halts$

$\implies$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever

$\implies$ then $HALTS(Turing, Turing) \neq halts$

$\implies$ $Turing(Turing)$ halts.

Contradiction. Program $HALT$ does not exist!
Halt and Turing.

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

Turing($P$)
1. If $HALT(P,P) = \text{"halts"}$, 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 $HALTS(Turing, Turing) = \text{halts}$
$\implies$ $Turing(Turing)$ loops forever.

$Turing(Turing)$ loops forever
$\implies$ then $HALTS(Turing, Turing) \neq \text{halts}$
$\implies$ $Turing(Turing)$ halts.

Contradiction. Program $HALT$ does not exist!

Questions?
Another view of proof: diagonalization.

Any program is a fixed length string.
Another view of proof: diagonalization.

Any program is a fixed length string. Fixed length strings are enumerable.
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.
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 - 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. 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>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

Halt - diagonal.
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>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

Halt - diagonal.  
Turing - is not Halt.
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>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
</tr>
</tbody>
</table>

Halt - diagonal.
Turing - is **not** Halt.
and is different from every $P_i$ on the diagonal.
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 - diagonal.
Turing - is not Halt. and is different from every $P_i$ on the diagonal.
Turing is not on list.
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.

\[
\begin{array}{cccc}
  & P_1 & P_2 & P_3 & \cdots \\
\hline
P_1 & H & H & L & \cdots \\
P_2 & L & L & H & \cdots \\
P_3 & L & H & H & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
\]

Halt - 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.
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>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>…</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>…</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>…</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Halt - 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.
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>$\cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>$\cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

Halt - 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!
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 - 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!
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$?
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
Proof play by play.

Assumed \( \text{HALT}(P, I) \) existed.

What is \( P? \) Text.

What is \( I? \)
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.
Proof play by play.

Assumed $\text{HALT}(P,I)$ existed.

What is $P$? Text.
What is $I$? Text.
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.
Proof play by play.

Assumed HALT($P, I$) existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program HALT($P, I$).
You have Text that is the program HALT($P, I$).
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.
You have *Text* that is the program $\text{HALT}(P, I)$.
Proof play by play.

Assumed \(\text{HALT}(P, I)\) existed.

What is \(P\)? Text.
What is \(I\)? Text.

What does it mean to have a program \(\text{HALT}(P, I)\).

You have Text that is the program \(\text{HALT}(P, I)\).

Have ___ that is the program TURING.
Proof play by play.

Assumed \( \text{HALT}(P, I) \) existed.

What is \( P \)? Text.
What is \( I \)? Text.

What does it mean to have a program \( \text{HALT}(P, I) \).

You have \textit{Text} that is the program \( \text{HALT}(P, I) \).

Have \textit{Text} that is the program \textit{TURING}. 

Here it is!!

\begin{align*}
\text{Turing}(P) \\
1. & \text{If } \text{HALT}(P, P) = \text{"halts"}, \text{ then go into an infinite loop.} \\
2. & \text{Otherwise, halt immediately.}
\end{align*}

\text{Turing "diagonalizes" on list of program.}

\( \Rightarrow \) \text{HALT is not a program.}
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.
   You have $Text$ that is the program $\text{HALT}(P, I)$.

Have $Text$ that is the program TURING.
Here it is!!
Proof play by play.

Assumed \( \text{HALT}(P, I) \) existed.

What is \( P \)? Text.
What is \( I \)? Text.

What does it mean to have a program \( \text{HALT}(P, I) \)?
You have Text that is the program \( \text{HALT}(P, I) \).

Have Text that is the program TURING.
Here it is!!
Turing(P)
Proof play by play.

Assumed HALT($P, I$) existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program HALT($P, I$).
   You have Text that is the program HALT($P, I$).

Have Text that is the program TURING.
Here it is!!

Turing($P$)
1. If HALT($P, P$) = “halts”, then go into an infinite loop.
Proof play by play.

Assumed HALT\((P, I)\) existed.

What is \(P\)? Text.
What is \(I\)? Text.

What does it mean to have a program HALT\((P, I)\).
You have Text that is the program HALT\((P, I)\).

Have Text that is the program TURING.
Here it is!!

\text{Turing}(P)

1. If HALT\((P,P)\) = “halts”, then go into an infinite loop.
2. Otherwise, halt immediately.
Proof play by play.

Assumed $\text{HALT}(P,I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P,I)$.

You have Text that is the program $\text{HALT}(P,I)$.

Have Text that is the program TURING.
Here it is!!

$\text{Turing}(P)$

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

Turing “diagonalizes” on list of program.
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.
You have Text that is the program $\text{HALT}(P, I)$.

Have Text that is the program $\text{TURING}$.
Here it is!!

$\text{Turing}(P)$

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

$\text{Turing}$ “diagonalizes” on list of program.
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.

You have Text that is the program $\text{HALT}(P, I)$.

Have Text that is the program $\text{TURING}$.
Here it is!!

$\text{Turing}(P)$

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

$\text{Turing}$ “diagonalizes” on list of program.
It is not a program!!!!
Proof play by play.

Assumed \( \text{HALT}(P, I) \) existed.

What is \( P \)? Text.
What is \( I \)? Text.

What does it mean to have a program \( \text{HALT}(P, I) \).
You have Text that is the program \( \text{HALT}(P, I) \).

Have Text that is the program TURING. Here it is!!

\[ \text{Turing}(P) \]
\[ \begin{align*}
1. & \quad \text{If } \text{HALT}(P,P) = \text{“halts”}, \text{ then go into an infinite loop.} \\
2. & \quad \text{Otherwise, halt immediately.}
\end{align*} \]

Turing “diagonalizes” on list of program. It is not a program!!!!

\[ \Rightarrow \text{HALT is not a program.} \]
Proof play by play.

Assumed $\text{HALT}(P, I)$ existed.

What is $P$? Text.
What is $I$? Text.

What does it mean to have a program $\text{HALT}(P, I)$.
You have $Text$ that is the program $\text{HALT}(P, I)$.

Have $Text$ that is the program TURING.
Here it is!!

$\text{Turing}(P)$
1. If $\text{HALT}(P,P) = \text{halts}$, then go into an infinite loop.
2. Otherwise, halt immediately.

Turing “diagonalizes” on list of program.
It is not a program!!!!

$\implies$ $\text{HALT}$ is not a program.

Questions?
Wow, that was easy!
Wow, that was easy!
We should be famous!
No computers for Turing!

In Turing's time.
No computers for Turing!

In Turing’s time.
No computers.
No computers for Turing!

In Turing’s time.
No computers.
Adding machines.
No computers for Turing!

In Turing’s time.
No computers.

Adding machines.
e.g., Babbage (from table of logarithms) 1812.
No computers for Turing!

In Turing’s time.
No computers.
Adding machines. 
  e.g., Babbage (from table of logarithms) 1812.
Concept of program as data wasn’t really there.
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 ...

Turing machine!

Now that's a computer!

Turing: AI,
self modifying code,
learning...
Turing machine.

A Turing machine.
– an (infinite) tape with characters
A Turing machine.

– an (infinite) tape with characters
– be in a state, and read a character
A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.
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
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
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...
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 ... Turing machine!
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 ... Turing machine!

Now that’s a computer!
Turing machine.

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!

Turing: AI,
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 ... Turing machine!

Now that’s a computer!

Turing: AI, self modifying code,
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!

Turing: AI, self modifying code, learning...
Turing and computing.

Just a mathematician?
Turing and computing.

Just a mathematician?

“Wrote” a chess program.
Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Simulated the program by hand to play chess.
Turing and computing.

Just a mathematician?
“Wrote” a chess program.
Simulated the program by hand to play chess.
It won!
Turing and computing.

Just a mathematician?
“Wrote” a chess program.
Simulated the program by hand to play chess.
It won! Once anyway.
Turing and computing.

Just a mathematician?
“Wrote” a chess program.
Simulated the program by hand to play chess.
It won! Once anyway.
Involved with computing labs through the 40s.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used \( \lambda \) calculus....which is...

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.... a natural number!

Today:

Programs can be written in ascii.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!
Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages!
Church proved an equivalent theorem. (Previously.) Used $\lambda$ calculus....which is... Lisp (Scheme)!!! .. functional part. Scheme’s lambda is calculus’s $\lambda$! Programming languages! javascript, ruby, python....
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!

.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!! .. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

    Any formal system either is inconsistent or incomplete.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!
Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.
   Any formal system either is inconsistent or incomplete.
   Inconsistent: A false sentence can be proven.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!

.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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 .... a natural number!
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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  
    .... a natural number!  !
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.) Used $\lambda$ calculus....which is... Lisp (Scheme)!!! .. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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 .... a natural number! !!
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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
.... a natural number! ! ! !
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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
   .... a natural number! ! ! ! Same cardinality as...
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used \( \lambda \) calculus....which is... Lisp (Scheme)!!!

.. functional part. Scheme’s lambda is calculus’s \( \lambda \)!

Programming languages! javascript, ruby, python....

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

.... a natural number! ! ! ! Same cardinality as...Text.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s λ!

Programming languages! javascript, ruby, python....

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
   .... a natural number! ! ! ! Same cardinality as...Text.

Today:
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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 .... a natural number! ! ! ! Same cardinality as...Text.

Today: Programs can be written in
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)
Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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
.... a natural number! ! ! ! Same cardinality as...Text.
Today: Programs can be written in ascii.
Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used $\lambda$ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme’s lambda is calculus’s $\lambda$!

Programming languages! javascript, ruby, python....

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
.... a natural number! ! ! ! Same cardinality as...Text.
Today: Programs can be written in ascii.
Computing on top of computing...

Computer, assembly code, programming language, browser, html, javascript..
Computing on top of computing...

Computer, assembly code, programming language, browser, html, javascript..

We can’t get enough of building more Turing machines.
Undecidable problems.

Does a program, $P$, print “Hello World”? 

Undecidable problems.

Does a program, $P$, print “Hello World”?
How?
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$?
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!
Undecidable problems.

Does a program, $P$, print “Hello World”?
How? What is $P$? Text!!!!!!
Undecidable problems.

Does a program, $P$, print "Hello World"?
How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

---

Proof: simulate a computer. Halts if finite.

Can a set of notched tiles tile the infinite plane?

Does a set of integer equations have a solution?
Example: $x^n + y^n = 1$?

Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no.

This "problem" is not undecidable.

Undecidability for Diophantine set of equations

$\Rightarrow$ no program can take any set of integer equations and always correctly output whether it has an integer solution.
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: Print “Hello World.”

Can a set of notched tiles tile the infinite plane?
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.
Undecidable problems.

Does a program, $P$, print “Hello World”?
How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “ $x^n + y^n = 1$?”
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: Print “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.

Be careful!
**Undecidable problems.**

Does a program, $P$, print “Hello World”?  
How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?  
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?  
Example: “$x^n + y^n = 1$?”  
Problem is undecidable.

Be careful!
Undecidable problems.

Does a program, \( P \), print “Hello World”? 
How? What is \( P \)? Text!!!!!!

Find exit points and add statement: \textbf{Print} “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: “\( x^n + y^n = 1? \)”
Problem is undecidable.

Be careful!

Is there an integer solution to \( x^n + y^n = 1 \)?
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$? (Diophantine equation.)
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$? (Diophantine equation.)

The answer is yes or no.
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$? (Diophantine equation.)

The answer is yes or no. This “problem” is not undecidable.
Undecidable problems.

Does a program, $P$, print "Hello World"?
How? What is $P$? Text!!!!!!

Find exit points and add statement: \textbf{Print} "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no. This "problem" is not undecidable.

Undecidability for Diophantine set of equations
Undecidable problems.

Does a program, $P$, print “Hello World”? How? What is $P$? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane? Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? Example: “$x^n + y^n = 1$?” Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$? (Diophantine equation.)

The answer is yes or no. This “problem” is not undecidable.

Undecidability for Diophantine set of equations $\implies$ no program can take any set of integer equations and
Undecidable problems.

Does a program, \( P \), print “Hello World”? 
How? What is \( P \)? Text!!!!!!

Find exit points and add statement: \textbf{Print} “Hello World.”

Can a set of notched tiles tile the infinite plane? 
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution? 
Example: \( x^n + y^n = 1? \)
Problem is undecidable.

Be careful!

Is there an integer solution to \( x^n + y^n = 1? \) 
(Diophantine equation.)

The answer is yes or no. This “problem” is not undecidable.

Undecidability for Diophantine set of equations
\( \implies \) no program can take any set of integer equations and 
always correctly output whether it has an integer solution.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis:
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person:
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob. Legs,
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob. Legs, arms,
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob. Legs, arms, head....
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob. Legs, arms, head.... How?
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
- Seminal paper in mathematical biology. Person: embryo is blob. Legs, arms, head.... How? Fly:
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn’t really work. Almost dependent matrices.
Turing: personal.

Tragic ending...
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;

(A bite from the apple....) accident?
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;
- took injections.

(A bite from the apple....) accident?
Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;
- took injections.
- lost security clearance...
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- Given choice of prison or (quackish) injections to eliminate sex drive;
- Took injections.
- Lost security clearance...
- Suffered from depression;
Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;
- took injections.
- lost security clearance...
- suffered from depression;
- suicided with cyanide at age 42.
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;
- took injections.
- lost security clearance...
- suffered from depression;
- suicided with cyanide at age 42.
  (A bite from the apple....)
Turing: personal.

Tragic ending...

- Arrested as a homosexual, (not particularly closeted)
- given choice of prison or (quackish) injections to eliminate sex drive;
- took injections.
- lost security clearance...
- suffered from depression;
- suicided with cyanide at age 42.
  (A bite from the apple....) accident?
British Apology.

Gordon Brown. 2009. “Alan and the many thousands of other gay men who were convicted as he was convicted under homophobic laws were treated terribly.
Gordon Brown. 2009. “Alan and the many thousands of other gay men who were convicted as he was convicted under homophobic laws were treated terribly. Over the years millions more lived in fear of conviction.
Gordon Brown. 2009. “Alan and the many thousands of other gay men who were convicted as he was convicted under homophobic laws were treated terribly. Over the years millions more lived in fear of conviction. ........
British Apology.

Gordon Brown. 2009. “Alan and the many thousands of other gay men who were convicted as he was convicted under homophobic laws were treated terribly. Over the years millions more lived in fear of conviction. .......... So on behalf of the British government, and all those who live freely thanks to Alan’s work I am very proud to say: we’re sorry, you deserved so much better.”
British Apology.

Gordon Brown. 2009. “Alan and the many thousands of other gay men who were convicted as he was convicted under homophobic laws were treated terribly. Over the years millions more lived in fear of conviction. .......... So on behalf of the British government, and all those who live freely thanks to Alan’s work I am very proud to say: we’re sorry, you deserved so much better.”

2013. Granted Royal pardon.
This statement is a lie.
This statement is a lie. **Neither true nor false!**
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.
Back to technical.

This statement is a lie. **Neither true nor false!**

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?
Back to technical..

This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

   **Who shaves the barber?**

def Turing(P):

This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

*Who shaves the barber?*

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return
This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

Who(shaves the barber?)

```python
def Turing(P):
    if Halts(P, P): while(true): pass
    else:
        return

...Text of Halt...
```
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program.
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\iff$ Turing Program. ($P \iff Q$)
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program. $(P \implies Q)$
Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P, P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\rightarrow$ Turing Program. ($P \rightarrow Q$)

Turing(“Turing”)?
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program \(\Rightarrow\) Turing Program. \((P \Rightarrow Q)\)

Turing(“Turing”)? Neither halts nor loops!
This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program. ($P \implies Q$)

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Progam \implies Turing Program. (P \implies Q)

Turing(“Turing”)? Neither halts nor loops! \implies No Turing program.
This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

**Who shaves the barber?**

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program. $(P \implies Q)$

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program
Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program \implies Turing Program. (P \implies Q)

Turing(“Turing”)? Neither halts nor loops! \implies No Turing program.

No Turing Program \implies
Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program. ($P \implies Q$)

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program.
This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):
    if Halts(P, P): while(true): pass
    else:
        return

...Text of Halt...

Halt Progam $\implies$ Turing Program. $(P \implies Q)$

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program. $(\neg Q \implies \neg P)$
This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

**Who shaves the barber?**

```python
def Turing(P):
    if Halts(P, P): while(true): pass
    else:
        return

...Text of Halt...

Halt Program $\implies$ Turing Program. ($P \implies Q$)

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program. ($\neg Q \implies \neg P$)

Program is text, so we can pass it to itself,
This statement is a lie. *Neither true nor false!*

Every person who doesn’t shave themselves is shaved by the barber.

*Who shaves the barber?*

```python
def Turing(P):
    if Halts(P,P): while(true): pass
    else:
        return

...Text of Halt...
```

Halt Program $\Rightarrow$ Turing Program. $(P \Rightarrow Q)$

Turing(“Turing”)? Neither halts nor loops! $\Rightarrow$ No Turing program.

No Turing Program $\Rightarrow$ No halt program. $(\neg Q \Rightarrow \neg P)$

Program is text, so we can pass it to itself, or refer to self.
Summary: decidability.

Computer Programs are an interesting thing.
Computer Programs are an interesting thing. Like Math.
Computer Programs are an interesting thing. 
Like Math. 
Formal Systems.
Computer Programs are an interesting thing. Like Math. Formal Systems.
Computer Programs are an interesting thing. Like Math. Formal Systems.

Computer Programs cannot completely “understand” computer programs.
Computer Programs are an interesting thing.
   Like Math.
   Formal Systems.

Computer Programs cannot completely “understand” computer programs.
Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Computation is a lens for other action in the world.
Probability

What’s to come?
What’s to come? Probability.
What’s to come? Probability.

A bag contains:
Probability

What’s to come? Probability.

A bag contains:

- 3 red balls
- 2 blue balls
- 1 yellow ball
- 2 blue balls
- 3 red balls
- 1 blue ball
- 1 red ball
- 1 blue ball
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue.
Probability

What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue. Count total.
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?
For now:
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?


For now: Counting!
What’s to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?
For now: Counting!
Later: Probability.