Compute: follow a fixed procedure and produce an answer (halt), aka, algorithm.

What can be computed? What cannot? What can be efficiently computed (and how)?

If a single question really is answerable "yes" or "no", then one of the machines, M_yes or M_no, computes the answer. We just don't know which one is correct.

Any finite set of examples can be computed: just make a table and look up the answer. Just because you don't know how doesn't mean it can't be done.

Are all programs algorithms? No.

```for (i = 1; i > 0; i = 1){
    j = j + 1;
}
```

Q. Are all TMs algorithms?

We can decode any finite set of questions using a fixed branching tree. For each leaf, we simply print the answer: A look-up table.

```
M is an algorithm solving Primality test for the numbers { 0, 1, ..., 15 }.
```
Fermat's Last Theorem

There are no solutions to,

\[ x^n + y^n = z^n \]

where \( n, x, y, \) and \( z \) are positive integers and \( n > 2 \).

(Proved in 1995: Frey, Ribet, Wiles, and Taylor.)

Given some positive integer \( n > 2 \), is there a solution to,

\[ x^n + y^n = z^n \]

where \( x, y, \) and \( z \) are positive integers?

If Fermat's Last Theorem is true, then

\( M_{\text{no}} \) will work w/o modification. \( \Rightarrow \) is computable.

Suppose it weren't true? That is, there are solns for some \( n \), but not all \( n \).

How would we go about computing the answer?

This works when there is a soln for a particular \( n \). \( \Rightarrow \) Will it halt for every \( n \)?
How many questions are there? How many TMs?

In our encoding, we used a string of 0s and 1s to represent a TM. Symbol set is \{0, 1\}.

--- Each TM can be identified with an integer. (There are infinitely many machines that do the same thing.)

--- Each input tape configuration can be identified with an integer.

--- Each output tape configuration can be identified with an integer.

--- A TM can be looked at as an integer function: given input, \( x \), machine \( M \) produces integer \( M(x) \).

--- \( M \) might loop forever on some inputs; if so, then \( M \) is a "partial" function.

**Q.** Can you encode an arbitrary input tape, in an arbitrary symbol set, using only \{0, 1\}? Hint, use unary encoding.

(Recall, only a finite portion of tape is non-blank.) That binary string is an integer.

**Computable (real) numbers:**
Given \( \epsilon \), output finite number of digits of \( x \) so that the output is within \( \epsilon \) of \( x \). \( \pi \) is such a number.
How many integer functions are there?

Consider a function \( g() \), which we describe by saying that \( g() \) is different from all the functions in the list above. How? Because \( g() \) is,

not the same as \( M_0 \) for the first output, \( g(0) \),
and it is,
not the same as \( M_1 \) for the second output, \( g(1) \),
and it is,

... 

--- Diagonalization:

\[
\begin{array}{ccc}
  & 0 & 1 & 2 & \cdots \\
M_0 & \neq & M_0(0) & M_0(1) & \cdots \\
M_1 & M_1(0) & \neq & M_1(1) & \cdots \\
M_2 & M_2(0) & M_2(1) & \neq & \cdots \\
\end{array}
\]

--- \( g() \) is different from every function in the list; so, \( g() \) is not in the list!

--- How many different ways are there to pick \( g() \)?

\( g() \) is any element from \( N - \{ M_0(0) \} \)
\( g() \) is any element from \( N - \{ M_1(1) \} \)
\( g() \) is any element from \( N - \{ M_2(2) \} \)

... There are so many different functions, \( g() \), proportionally, that the probability of randomly picking a function from a bag of integer functions and having that function correspond to some TM is 0.

[What the heck does that really mean?] 

That is,

There are a lot of functions (more than all the positive integers).

Nearly all are incomputable

Maybe it only means we don't know how to arrange an infinite list of TMs? We are limited in our own computing power?

How "numerous" is "infinity to the infinity"?

How can we know we are able to produce \( g() \) this way?
As long as we are building TMs, let's see how to simplify our work. How about combining two TMs to make a new one?

M3 starts in M1's start state.

- Every M1 state transition that goes to M1's "HALT" state is instead connected to M2's START state.
- M3's halting state is M2's "HALT" state.

"Spin-left 0s"

Then "parity fix"

"Spin-left 0s"
Lemma: All TM's with \( x \) as input, either (1) HALT or (2) LOOP FOREVER. (exercise: prove the lemma.)

A very special integer function: The **Halting function**: 

<table>
<thead>
<tr>
<th>input: integer ( xM )</th>
<th>(( xM ) == an encoding of input ( x ) followed by an encoding of ( M ).)</th>
</tr>
</thead>
<tbody>
<tr>
<td>output: &quot;1&quot; if ( xM ) HALTS; &quot;0&quot; otherwise.</td>
<td>(( xM ) == ( M ) reading ( x ) as its input.)</td>
</tr>
</tbody>
</table>

**Q. Can there be such an \( H \) function. Is it possible?**

**Assumption**: Either (\( H \) exists) IS TRUE, or (\( H \) does not exist) IS TRUE.

Suppose (\( H \) exists) IS TRUE.

Then we can build another machine, \( H^+ \), using \( H \) and a "Copy" TM.

\( H^+ \)

1. copies its input.
2. acts as \( H \) would, except:
   WHEN \( H^+ \) reaches \( xM \) halts, \( H^+ \) LOOPS.
Consider putting $\text{desc}(H^+)$ on $H^+$’s input tape. What must happen?

$H^+$ first does exactly what $\text{Copy}$ would do, copy its input. Next, $H^+$ acts exactly as $H$ would.

The tape is now thought of as, an input, $x = \text{desc}(H^+)$, followed by a machine description, $\text{desc}(M) = \text{desc}(H^+)$. 

$H^+$ WILL either (because $H$ always halts in HALTS or LOOPS) (reach HALTS and then loop) OR (reach LOOPS and then halt).

SUPPOSE $H^+$ loops.

1. $H^+$ reached HALTS.
2. Then $H$ with input $xM \equiv \text{desc}(H^+) \text{ desc}(H^+)$, \text{ would have halted} in HALTS.
3. BUT $H^+$ reading $\text{desc}(H^+)$ loops (our assumption).
4. Since $H$ is correct, it would not go to HALTS.
5. $H^+$ cannot reach HALTS, and does not loop.
6. This contradicts our assumption that $H^+$ loops.

We assumed $H$ exists, i.e., it works correctly. Assuming also that $H^+$ loops leads to a contradiction. At least one of these assumptions must be false.
SUPPOSE $H^+$ halts.

1. $H^+$ reached $\text{LOOPS}$.
2. $H$ reading $\text{desc}(H^+)$ desc($H^+$) must reach $\text{LOOPS}$.
3. BUT desc($H^+$) $H^+$ halts.
4. $H$ is correct; so, $H$ cannot reach $\text{LOOPS}$.
5. desc($H^+$) $H^+$ cannot reach $\text{LOOPS}$.

We assumed $H$ is correct. Assuming also that $H^+$ halts leads to a contradiction.

If $H$ exists, $H^+$ exists, is a TM, and either halts or loops. (Building $H^+$ from $H$ was easy and resulted in a TM.)

But both cases ($H^+$ either halts or loops) lead to contradictions.

The assumption that $H$ exists must be false.

This is better than diagonalization: we have a real, uncomputable function. The function exists because every TM $M$ either halts or loops forever, give an input $x$.

There is a function $H()$ mapping

$H: \{M \} \mapsto \{0, 1\}$

from positive integers to $\{0, 1\}$, but no TM can compute it.

Are we doomed?

Build something $H^+$ that partially computes the Halting Problem?

Works for some inputs, but not others?

Works for some fixed number of inputs?

Has a lookup table?

How many machines act exactly like any given description?

How many descriptions are there?

How many other things are not Turing computable? What does this say about cognition? ...???
Another Method? Does it work? Why?

\[ \text{Hnew}(x, M) \]

print "loops forever"

1. Simulate \( xM \) for one step.
2. If \( xM \) halted
   print "halts"
else
   go to 1.

\[ \Rightarrow \text{Is HP computable?} \]

\[ \text{Bottom Line} \]

Suppose we try to write a program \( H(x, M) \).

We succeed for some special cases \( \{M_1, M_{25}, M_{300}, \ldots \} \).

But, we always find a new \( M_i \) and have to rewrite \( H(x, M) \). Also, we get it to work for \( \{x_1, x_2, \ldots \} \), but find a new \( x_i \) for which \( x_iM_j \) loops (or halts) (can we figure that out?), and our \( H(x_i, M_j) \) says it will loop. Back to re-writing our \( H(x, M) \).

\[ H \Rightarrow \text{We will never be bored!} \]
Formal Proof

Notation: "[halts]" means "H+ halts when reading its own description"; "[loops]" is to be read similarly; "===>" means, "implies", in the logical sense of material implication; ".-" means logical NOT.

1. (H exists) ==> (H+ exists (is a TM))  (by properties of TM)
2. (H+ exists) ==> [halts] OR [loops]  (by properties of TM)
3. (H+ exists) ==> -[loops] AND -[halts]  (demonstrated above)
4. (H exists) ==> ([halts] OR [loops]) AND (-[loops] AND -[halts])  (by 1. and 2.)
5. (H exists) ==> ([halts] AND -[halts]) OR ([loops] AND -[loops])  (by AND/OR properties)
6. p ==> q  EQUALS  -q ==> -p  (by properties of "===>")
7. -( ([halts] AND -[halts]) OR ([loops] AND -[loops]) ) ==> -(H exists)  (by 5. and 6.)
8. -( ([halts] AND -[halts]) OR ([loops] AND -[loops]) )  (true by AND/OR properties)
9. -(H exists)  (syllogism applied to 7. and 8.)