This green function seems alien!!!

**Inputs**

\[
\begin{align*}
5 & \rightarrow 17 \\
42 & \rightarrow 18 \\
47 & \rightarrow 18 \\
100 & \rightarrow 19 \\
1000 & \rightarrow 20 \\
10000 & \rightarrow 21 \\
100000 & \rightarrow 21 \\
1000005 & \rightarrow 23 \\
100042 & \rightarrow 24 \\
1000000 & \rightarrow 21 \\
10000005 & \rightarrow 23 \\
100000...0000 & \rightarrow 23 \\
100000...0000 & \rightarrow 27
\end{align*}
\]

**Outputs**

\[
\begin{align*}
\text{cmp} & \text{ cmp}(5) == 17 \\
\text{hc} & \text{ hc}(CS5) == \text{True}
\end{align*}
\]

Two *uncomputable* functions

These would be useful - *if only they were possible*

**PLAN!**

- **Friday 5/3**: final project due (by 8pm) come to labs for f.p. help!
- **Thursday 5/9**: final exam *review* @ class time (here)
- **Tues. 5/14 @ 2 or 7**: final exam

---

**All known and imagined computers** can compute

\[\text{only}\] what Turing Machines can...

though some are faster than others

- Quantum computation
- Molecular computation
  [http://www.astrobiology.com/reviews/2q00/dna/dna-1.html](http://www.astrobiology.com/reviews/2q00/dna/dna-1.html)
- Parallel computers
- Integrated circuits
- Electromechanical computation
- Water-based computation
- Tinkertoy computation

**Turing Machines!**

TM's can do everything... *though not fast*

but FMs + TMs *can* get unruly pretty fast!

---

Extra #1!
TM's can do everything...
... but only everything that *can* be computed!

There are – *surprisingly* – many mathematical functions that no program, computer, or Turing Machine\textsuperscript{TM} can compute!

You can certainly *write* programs for them, but those programs – *provably* – have at least one bug! And, fixing the bug *necessarily* creates more!

Cantor and "friends"

"Charlatan"
"Renegade"
"*Corrupter of Youth*"

"Laughable and wrong"
"Utter nonsense"

No one shall expel us from the paradise that Cantor has created.

(Many) more *functions* than *programs*!

\[
\begin{align*}
\text{Uncountably infinite} & \quad > \quad \text{Countably infinite} \\
\text{This is how many int-bool functions there are.} & \quad \text{This is how many programs there are.}
\end{align*}
\]

(Many) more *functions* than *programs*!

\[
\begin{align*}
\text{Uncountably infinite} & \quad > \quad \text{Countably infinite} \\
\text{This is how many int-bool functions there are.} & \quad \text{This is how many programs there are.}
\end{align*}
\]

Leopold Kronecker
1823–1891

Lugwig Wittgenstein
1889–1951

David Hilbert
1862–1943

No one shall expel us from the paradise that Cantor has created.

"Laughable and wrong"
"Utter nonsense"

And this was before 1900!
Let's get **real**

**the reals**
(from 0 to 1)

\{ 0.0 \leq r \leq 1.0 \}

**the integers**
(positive)

\{ 1, 2, 3, 4, 5, ... \}

---

**Can’t get real**

Are these sets
equal-sized?

\[ \begin{align*}
.1 \\
.2 \\
.3 \\
... \\
.11 \\
.12 \\
...
\end{align*} \]

Why is this **NOT a valid matching**?

\[ \begin{align*}
1 \\
2 \\
3 \\
... \\
11 \\
12 \\
...
\end{align*} \]

---

**Cantor Diagonalization**

From any list, there are *always* real #s missing!

**the Reals**
(from 0 to 1)

**Positive integers**

\[ \begin{align*}
\text{real #s are always missing} & \quad r_0 = 0 \\
\text{regardless of the matching...} & \quad r_1 = \ldots .33333333333333\ldots \rightarrow 1 \\
& \quad r_2 = \ldots .42424242424242\ldots \rightarrow 2 \\
& \quad r_3 = \ldots .314159426535897\ldots \rightarrow 3 \\
& \quad r_4 = \ldots .420000000000000\ldots \rightarrow 4 \\
& \quad r_5 = \ldots \rightarrow 5
\end{align*} \]

---

**More functions than programs**

**Well-defined mathematical functions** (uncountable)

**uncountably many functions**

**countably many programs**

What’s swimming around out here?

**Human-describable things...**

Programs (countable)

Human-describable things...

Wow – I can’t even describe the stuff out here!

**Next...**

**But what’s a specific function**
that can’t be computed!?  

**an example!?**

**anything expressed w/math...**

it's computable!

**any describable pattern...**

it's programmable!

**the complexity of an integer**
What is the **complexity** of an integer?

100 zeros total

100000000...00000000000000000000000

each of these integers has the same # of digits

1568424042...2603635422093849215049

same number of total digits as above

Which "feels like" the more complex - or more compressible - number... *Why?*

The complexity, \( \text{cmp} \), of an integer \( x \) is the length of the shortest zero-input function that outputs \( x \). "description"

\[ \text{cmp}(42) = 18 \]

So, 42 has a complexity of 18. *Why 18?*
Although \texttt{cmp(x)} is a well-defined \textit{mathematical} function, with an int output for each int input \texttt{x},

\begin{align*}
\text{cmp}(42) &= 16 + 2 = 18 \\
\text{cmp}(9001) &= 16 + \square = 20 \\
\text{cmp}(1000000) &= 16 + \square \\
\text{cmp}(10000042) &= 16 + \square \\
\text{cmp}(100000000) &= 16 + \square \\
\text{cmp}(1000000000) &= 16 + \square \\
\text{cmp}(10000000000) &= 16 + \square \\
\text{cmp}(31415926...) &= 16 + \square \\
\text{cmp}(86753098...) &= 16 + \square
\end{align*}

\textit{Extra:} What's the largest \texttt{x} with \texttt{cmp(x)} == 20?  
\textit{Extra Extra:} Are there any integers \texttt{x} with \texttt{cmp(x)} > 1,000,000?

\textit{Extra Extra Extra:} How big is the SMALLEST such integer?

\texttt{cmp(x)} \textit{is not} a \textit{computable} function.

\begin{align*}
\text{cmp}(5) &= 17 \\
42 &= 18 \\
47 &= 18 \\
100 &= 19 \\
1000 &= 20 \\
10000 &= 21 \\
100000 &= 21 \\
1000000 &= 23
\end{align*}

We know \texttt{cmp(5) ==17}, so we propose this:

\begin{verbatim}
def cmp( x )
    return x+12
\end{verbatim}

This is a function that we will prove \textit{can't be programmed}...

Although \texttt{cmp(x)} is a well-defined \textit{mathematical} function, with an int output for each int input \texttt{x},

\texttt{cmp(x)} \textit{is not} a \textit{computable} function.

You'll show \textit{every possible} program for \texttt{cmp} has a bug! that we will prove \textit{can't be computed}...

How?!?
We know \texttt{cmp}(x) is 16+\_\_\_, so, let’s try this:

\begin{verbatim}
def cmp( x ):
    return 16 + len(str(x))
\end{verbatim}

It works for 5, 42, and 47!

\textbf{bug(s)?}

We need to prove that this \texttt{cmp}(x) function contains a bug.

\begin{verbatim}
def cmp( x ):
    do stuff and then...
    return answer
\end{verbatim}

\textit{This version of \texttt{cmp} is 900,000 chars long and claims to return the complexity of x}

\textbf{We’re going to write a function to find and show that bug!}

\textbf{Bug-Finding Function!}
**def BFF():**

```python
def cmp( x ):  
    do stuff and then...  
    return answer
```

This version of `cmp` is **900,000** chars long and claims to return the complexity of `x`.

```python
x = 0  
while cmp(x) < 1000000:  
    x += 1  
return x
```

Note that **BFF** is a zero-input function that is **900,058** characters long...

... and it returns an integer, **bug**!

**bug = BFF()**

yet...

The complexity of `x` is the length of the shortest zero-input function that outputs `x`.

**def BFF():**

```python
def cmp( x ):  
    do stuff and then...  
    return answer
```

This version of `cmp` is **900,000** chars long and claims to return the complexity of `x`.

```python
x = 0  
while cmp(x) < 1000000:  
    x += 1  
return x
```

Note: this loop checks `cmp(x)`, not `x` itself.

**bug = BFF()**

The complexity of `x` is the length of the shortest zero-input function that outputs `x`.

**Proof**

(1) When does this **while** loop stop?  
[A] when `cmp(x)<1,000,000`  
[B] when `cmp(x)== 1,000,000`  
[C] it **never** stops

(2) The **while** ensures the `x.BFF` returns is  
[T] an int `x` with complexity < 1,000,000  
[U] an int `x` with complexity >= 1,000,000  
[V] **nothing** – `x` is never returned at all

(3) Why does this mean `cmp` has a bug?  
[F] because `x != bug`  
[G] `BFF` shows `bug`'s complexity is **900,058** or less!  
[H] there were no three-eyed aliens involved :-(

Although **cmp** is a **well-defined** mathematical function, with an int output for each int input `x`,

**cmp** is **not** a computable function.

**cmp** can't be debugged!

Proven!
Compression connection?

The best-compressed version of any data \( D \) is the **shortest program that generates** \( D \).

**Minimum description length**

From Wikipedia, the free encyclopedia

The minimum description length (MDL) principle is a formalization of Occam’s razor in which the best hypothesis for a given set of data is the one that leads to the best compression of the data.

> [The MDL Principle] is based on the following insight: any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally.

- Kolmogorov complexity is uncomputable: there exists no algorithm that, when input an arbitrary sequence of data, outputs the shortest program that produces the data.

**Kolmogorov Directions**

| < | < Prev | Random | Next | > |

**HOW DO I GET TO YOUR PLACE FROM LEXINGTON?**

**HMM...**

**OK, STARTING FROM YOUR DRIVEWAY, TAKE EVERY LEFT THAT DOESN'T PUT YOU ON A PRIME-NUMBERED HIGHWAY OR STREET NAMED FOR A PRESIDENT.**

**When people ask for step-by-step directions, I worry that there will be too many steps to remember, so I try to put them in minimal form.**

Xkcd

Kolmogorov Complexity?

- Some infinite patterns/functions have finite descriptions.
  - They're all programmable/computable

More infinite patterns/functions do **not** have finite descriptions.

- They're **not** programmable/computable
Two **useful** functions that *provably can't be computed*...

\[
\text{\texttt{cmp( x )}}
\]

~ the *complexity* of an integer \( x \)

\[
\text{\texttt{hc( f )}}
\]

~ does \( f() \) halt or loop forever?

---

**Haltchecking** is uncomputable.

\[
\text{\texttt{hc( f )}}
\]

~ returns whether \( f() \) halts or not

\( \text{hc always has a bug!} \)

---

**Haltchecking** is uncomputable.

\[\text{def \texttt{hc( f ):}}\]

any Python function

\[\text{def \texttt{hc( f ):}}\]

\[\text{def \texttt{hc( f ):}}\]

It is **impossible** to write a (bug-free) function \( \text{hc( f )} \) that determines if a function \( f \) halts when run:

- \( \text{hc( f )} \) returns \texttt{True} if \( f() \) halts and
- \( \text{hc( f )} \) returns \texttt{False} if \( f() \) loops infinitely

---

Suppose \( \text{hc( f )} \) worked for all \( f \)

Create this **BFF**:

\[\text{def \texttt{BFF():}}\]

\[\text{if \texttt{hc(BFF) == True:}}\]

\[\text{while 1+1==2: print 'Ha!' \]}

\[\text{else:}}\]

\[\text{return} \quad \#\ \text{halt!} \]

\[\text{Is \texttt{hc(BFF) == True} ?} \]

\[\text{Is \texttt{hc(BFF) == False} ?} \]

\[\text{hc always has a bug} \quad \text{Proven!} \]

---
∞ loops are **undetectable**

some are detectable, but some are not — and there's no way to know!

bugs are **inevitable**

infinite loops are just one type of bug... In general, they're all undetectable — all behavioral, not syntactic, bugs

programming is **not automatable**...

at least, not *bug-free* programming

---

And this is important because...

---

Good luck with final projects

They will **halt** all too soon!

---

xkcd's take is... ...all **too True**, in fact.
\[
\text{cmp}(42) = 16 + 2 = 18
\]
\[
\text{cmp}(9001) = 16 + \_\_ = 20
\]
\[
\text{cmp}(1000000) = \quad 16 + \_\_
\]
\[
\text{cmp}(1000042) = \quad 16 + \_\_
\]
\[
\begin{align*}
\text{cmp}(1000000000) &= 16 + \_\_ \\
\text{cmp}(9001) &= 16 + \_\_ \\
\text{cmp}(10000042) &= 16 + \_\_ \\
\text{cmp}(10000000000) &= 16 + \_\_ \\
\text{cmp}(10000...0000) &= 16 + \_\_ \\
\end{align*}
\]

Extra: What's the largest x with \(\text{cmp}(x) = 20\)?

Extra Extra: Are there any integers \(x\) with \(\text{cmp}(x) > 1,000,000\)?

Extra Extra Extra! How big is the SMALLEST such integer?

http://www.googolplexwrittenout.com/