This green function seems alien!!!
Two *uncomputable* functions

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

- **cmp**
  - `cmp(5) == 17`

- **hc**
  - `hc(CS5) == True`

**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 only what Turing Machines can...

though some are faster than others

Quantum computation

Molecular computation
http://www.arstechnica.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!
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 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!

Now that's a frustrating program to debug!
(Many) more **functions** than **programs**!

Uncountably infinite

This is how many int-bool **functions** there are.

Countably infinite

This is how many **programs** there are.
No one shall expel us from the paradise that Cantor has created.
Thanks, Neel!

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

- **Uncountably infinite**: This is how many int-bool **functions** there are.
- **Countably infinite**: This is how many **programs** there are.

The Reals from 0 to 1: $\mathbb{R}$

Positive integers: $\mathbb{N}$
Let's get real

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

\{ 0.0 \leq r \leq 1.0 \}

.1
.2
.3
...
.11
.12
...

**The integers**
(positive)

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

1
2
3
...
11
12
...

Why is this NOT a valid matching?

Are these sets equal-sized?

Essence? Finite patterns!
Cantor Diagonalization

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

\[ R \] \hspace{1cm} \text{the Reals from 0 to 1} \hspace{1cm} N \hspace{1cm} \text{Positive integers}

\[ r_0 = .4424 \]

\[ r_1 = .3333333333333333... \hspace{1cm} 1 \]
\[ r_2 = .42424242424242424... \hspace{1cm} 2 \]
\[ r_3 = .314159426535897... \hspace{1cm} 3 \]
\[ r_4 = .420000000000000... \hspace{1cm} 4 \]
\[ r_5 = ... \hspace{1cm} 5 \]
Well-defined mathematical functions (uncountable)

uncountably many functions

What's swimming around out here?

countably many programs

Wow ~ I can't even describe the stuff out here!

Programs (countable)
Well-defined mathematical functions (uncountable)

What's swimming around out here?

Programs (countable)

Human-describable things...

uncountably many functions

countably many programs

Wow ~ I can't even describe the stuff out here!

more functions than programs
Well-defined mathematical functions (uncountable)

What's swimming around out here?

Wow ~ I can't even describe the stuff out here!

Programs (countable)

Human-describable things...

More functions than programs

Uncountably many functions

Countably many programs

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

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?
What is the *complexity* of an integer?

- Each of these integers has the same number of digits.
- 100 zeros total.

**Intuition:**
The *complexity* or *compressibility* of $x$ is the length of the shortest description of $x$.

Which "feels like" the more complex - or more compressible - number... *Why*?
The complexity, $\text{cmp}$, of an integer $x$ is the length of $x$'s shortest description.
The complexity of \( x \), \( \text{cmp}(x) \), is the length of the shortest zero-input function that outputs \( x \).

"description"

```python
def f():
    return 5
```

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

So, \( 42 \) has a complexity of 18.

Why 18?
The **complexity** of \( x \), \( \text{cmp}(x) \), is the length of the shortest zero-input function that outputs \( x \).

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

**Kolmogorov Complexity**

```python
def f():
    return 42
```

... because, Python has 16 characters of overhead and **2 more are needed**
"def f(): return 100"… Python has 16 characters of overhead, so "cmp(100) is 19"
... Python has 16 characters of overhead, so \( \text{cmp}(1000) \) is 20
... Python has 16 characters of overhead, so \( \text{cmp}(10000) \) is 21
\[ \begin{align*}
\text{Inputs} & \quad (x) \\
5 & \rightarrow 17 \\
42 & \rightarrow 18 \\
43 & \rightarrow 18 \\
100 & \rightarrow 19 \\
1000 & \rightarrow 20 \\
10000 & \rightarrow 21 \\
100000 & \rightarrow 21 \\
\end{align*} \]

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

```python
def f():
    return 100000
```

... Python has 16 characters of overhead, but \textit{we can do better here!}
... Python has 16 characters of overhead, and \texttt{cmp(100000)} is also 21
\[
\text{cmp}(42) = 16 + 2 = 18 \\
\text{cmp}(9001) = 16 + 4 = 20 \\
\text{cmp}(1000000) = 16 + 5 = 21 \\
\text{cmp}(1000042) = 16 + 7 = 23 \\
\text{cmp}(1000000000) = 16 + 6 = 22 \\
\text{cmp}(31415926...) = 16 + 1 = 17 \\
\text{cmp}(86753098...) = 16 + 1000 = 1016 \\
\]

**Quiz!**

What does \text{cmp}(x) return for each of these integers, \( x \)?

**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?
\[
\begin{align*}
\text{cmp}(42) &= 16 + 2 = 18 \\
\text{cmp}(9001) &= 16 + 4 = 20 \\
\text{cmp}(1000000) &= 16 + \frac{5}{6 \text{ zeros}} = 21 \\
\text{cmp}(1000042) &= 16 + \frac{7}{a \text{ million and } 42} = 23 \\
\text{cmp}(1000000000) &= 16 + \frac{5}{a \text{ billion}} = 21 \\
\text{cmp}(10000...000) &= 16 + \frac{7}{a \text{ googol}} = 23 \\
\text{cmp}(100...000) &= 16 + \frac{11}{a \text{ googolplex}} = 27 \\
\text{cmp}(31415926...) &= 16 + \frac{9}{1 \text{ billion digits of pi, as an integer}} \\
\text{cmp}(86753098...) &= 16 + \frac{1000}{1 \text{ thousand patternless digits}} \\
\end{align*}
\]

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

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

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

What does `cmp(x)` return for each of these integers, x?

- `cmp(42) = 16 + 2 = 18`
- `cmp(9001) = 16 + 4 = 20`
- `cmp(10000000) = 16 + 5 = 21`
- `cmp(1000042) = 16 + 7 = 23`
- `cmp(10**6) vs. 10**6+42`
- `cmp(10000000000) = 16 + 5 = 21`
- `cmp(10000...0000) = 16 + 7 = 23`

**Extra:** What's the largest x with `kc(x) == 20`?

**Extra Extra:** Are there any integers x with `cmp(x) > 1,000,000`?

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

- `cmp(100...000) = 16 + 11 = 27`
- `cmp(31415926...) = 16 + ___ = 27`
- `cmp(86753098...) = 16 + 1000 = 23`

Extra: 9**9

Try this on the back page first...

Whoa! 1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 googolplex

1 googol zeros

1 billion digits of pi, as an integer

for i in range(10**10**100): throwDart()...

86753098... [through all 1,000 digits]
Although \texttt{cmp (x)} is a well-defined mathematical function, with an int output for each int input \texttt{x},

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

the complexity of \texttt{x}  
the compressability of \texttt{x}

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

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

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

the complexity of \texttt{x}  
the \textit{compressability} of \texttt{x}  

\textbf{How!?!}

You'll show \textbf{every possible program for} \texttt{cmp} has a bug! 

\texttt{you} we will prove \textit{can't be programmed}...
Although $\text{cmp}(x)$ is a well-defined mathematical function, with an int output for each int input $x$, 

\textbf{cmp}(x) \text{ is not a computable function.}

the complexity of $x$
the compressability of $x$

You'll show \textit{every possible} program for $\text{cmp}$ has a bug!

that we will prove \textit{can't be computed}...

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

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

Looks good to me!

\textbf{bug(s)?}
**cmp version #1**

We know `cmp(5) == 17`, so we propose this:

```python
def cmp( x )
    return x + 12
```

Find a bug:

- `cmp(42)` should be 18.
- But our `cmp(42)` outputs 60.

*This cmp has a bug!* (actually, lots of bugs!)
def cmp(x):
    return 16 + len(str(x))

bug(s)?

5 → 17
42 → 18
47 → 18
100 → 19
1000 → 20
10000 → 21
100000 → 21
1000005 → 23

We know \texttt{cmp}(\texttt{x}) \text{ is } 16+___, so, let's try this:

It works for 5, 42, and 47!
def cmp(x):
    return 16 + len(str(x))

def cmp(x):
    return 16 + len(str(x))

Find a bug: cmp(100000) is actually 21.
But, this cmp(100000) isn't 21. 6 digits
So, this cmp also has a bug! (lots, in fact...)
\texttt{cmp} \sim \textit{any version}

Aargh! Let's try this \texttt{cmp(x)}:

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

I guess you'd call this part of the code a GRAY AREA...
**cmp** ~ *any* version

Aargh! Let's try this **cmp**(*x*):

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

The challenge: we have to find a bug in *any* possible version of **cmp**(*x*)!
We need to prove that this `cmp(x)` function contains a bug.

We're going to write a function to find and show that bug!
def BFF():
    def cmp(x):
        do stuff and then...
        return answer
    x = 0
    while cmp(x) < 1000000:
        x += 1
    return x

bug = BFF()
```python
def BFF():
    def cmp(x):
        do stuff and then...
        return answer
    x = 0
    while cmp(x) < 1000000:
        x += 1
    return x

bug = BFF()  # BFF?
```

This version of `cmp` is **900,000** chars long and claims to return the complexity of `x`.
def BFF():
    def cmp(x):
        do stuff and then...
        return answer

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

bug = BFF()

Note that BFF is a zero-input function that is 900,058 characters long... 
... and it returns an integer, bug!

x's complexity \leq 900,058

yet...

The complexity of x is the length of the shortest zero-input function that outputs x.
def BFF():
    def cmp(x):
        do stuff and then...
        return answer
    x = 0
    while cmp(x) < 1000000:
        x += 1
    return x

bug = BFF()

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

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

(1) When does this while loop stop?
   [A] when cmp(x) < 1,000,000 or
   [B] when cmp(x) >= 1,000,000 or
   [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 :-)

The complexity of x is the length of the shortest zero-input function that outputs x.
def BFF():
    def cmp(x):
        do stuff and then...
        return answer
    x = 0
    while cmp(x) < 1000000:
        x += 1
    return x
bug = BFF()

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

Even this implementation of cmp(x) contains a bug!

zero-input f'n of size less than a million

number with complexity over a million!

cmp failed!

so, every implementation of cmp(x) contains a bug!

but, we allowed it to do anything at all!
Although `cmp(x)` is a well-defined mathematical function, with an int output for each int input `x`,

`cmp(x)` is not a computable function.

`cmp(x)` can't be debugged!
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.
Compression connection?

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

\begin{verbatim}
Compressed compressed
8=1 2!
9=3 4
*=like
/=them
+=here
1=hat
2=Sam-I-Am
3=I do not *
4=green eggs and ham
5=you *
6=+ or t+
7=I would not * /
T8
T8
3 t8
Do 5 4?
3 /, 2.
9!
Would 5 / 6?
7 6.
7 anyw+. 9.
7 /, 2.
\end{verbatim}

\textit{Total characters: 187 (95\% compression ratio)}

An impressive compression result!
Compression connection?

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

Uncomputable!
**Kolmogorov Directions**

*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.*

People get really grumpy when they realize you're giving them directions for how to go to the store and buy a GPS.
insight?

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
insight?

not all infinities are created equal!

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...

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

\(~\) the \textit{complexity} of an integer \(x\)

\[ (\text{compressibility}) \]

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

\(~\) does \(f()\) halt or loop forever?
Haltchecking is uncomputable.

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

\[ \sim \text{returns whether } f() \text{ halts or not} \]

\textcolor{blue}{\text{hc}} always has a bug!
Haltchecking is uncomputable.

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 \textbf{True} if \( f() \) halts and
- \( \text{hc}(f) \) returns \textbf{False} if \( f() \) loops infinitely
Suppose $hc(f)$ worked for all $f$ Create this **BFF**:

```python
def BFF():
    if $hc(BFF) == True$:  # don't halt
        while $1+1==2$: print 'Ha!'
    else:  # if BFF loops forever
        return # halt
```

Is $hc(BFF) == True$ ?

Is $hc(BFF) == False$ ?

$hcallways$ has a bug  

**Proven!**
And this is important because ...
∞ loops are undetectable

some are detectable, but some are not
– and there's no way to know!
And this is important because ...

∞ 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

Rice's Theorem: CS81
And this is important because ...

\[ \infty \text{ loops are } \text{undetectable} \]

\textit{some} are detectable, but \textit{some} are not – and there's no way to know!

bugs are \textit{inevitable}

infinite loops are just \textit{one} type of bug...
In general, they're \textit{all} undetectable
\sim all \textit{behavioral}, not syntactic, bugs

programming is \textit{not automatable}...

at least, not \textit{bug-free} programming

Rice's Theorem: CS81
Let's celebrate!

https://www.youtube.com/watch?v=vmINGWsyWX0
Good luck with final projects

They will **halt** all too soon!

Tutoring hours and labs through Friday...!

*xkcd's take is... ...
... all **too True**, in fact.*