5 → 16		
Anguments		Results
Arguments	$42 \rightarrow 17$	MESUICS
43 → 17		
$100 \rightarrow 18$		
1000 → 19		
10000 → 20		
100000 → 20		
100005 → 22		
100042 → 23		
$10000000 \rightarrow 20$		
$10000005 \rightarrow 22$		
$_{a'_{100000}}$ 10000000000 $ ightarrow$ 22		This seems alien!!!









it's computable!

Anything expressed w/math...

it's computable!

Human-discernable patterns...

The *complexity* of an integer



$$kc(5) == 16$$



hc

hc (CS5)

Two uncomputable functions

These would be useful—if only they were possible

Final projects:

• Help! lab times and evening hours all week...

•Final project due 5/3 under hw "final"

Final exam:

- Check out the online practice problems...
- Two pages of notes are welcome...
- Exam: Wed. 5/12 @ 2 PM

What is the *complexity* of an integer?

100,000 zeros total

Each of these integers has the same number of digits

170117684...20006872822488857785601

Intuition:

The *complexity* or *compressibility* of \mathbf{x} is the length of the **shortest description** of x.

Which one "feels like" the more complicated number... **Why**?

The *complexity* of ${\bf x}$ is the length of the shortest zero-argument function that returns ${\bf x}$.

"description"

The complexity, kc, of a number x is the length of x's shortest description

```
def BFF():
    def kc(x):
        do stuff and then...
        return answer

x = 0
while kc(x) < 50000:
        x += 1
return x</pre>
BFF?
```

```
Arguments
(x)

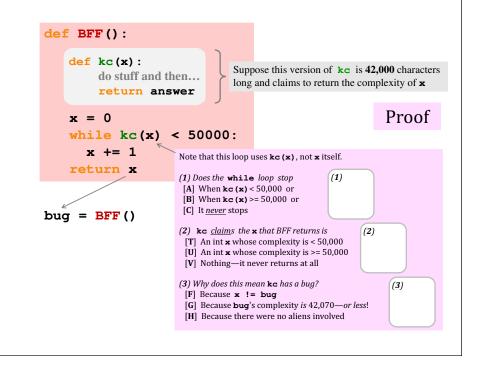
\begin{array}{c}
5 \to 16 \\
42 \to 17 \\
43 \to 17
\end{array}

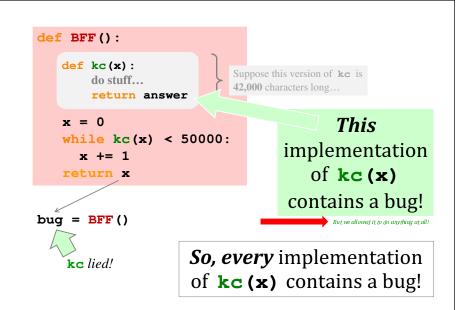
Results

\begin{array}{c}
1000 \to 18 \\
10000 \to 19 \\
100000 \to 20 \\
1000000 \to 20
\end{array}

?
```

... because Python has 15 characters of overhead and 2 more are needed





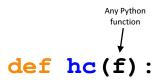
Although kc (x) is a well-defined mathematical function, with an int result for each int argument x,

kc(x) is *not* a computable function.

Every implementation of **kc(x)** contains a bug!

Proven!

Halt checking is uncomputable.



It is <u>impossible</u> to write a (bug-free) function **hc** (**f**) that determines whether a function **f** halts when run:

- hc(f) returns True if f() halts and
- hc(f) returns False if f() loops infinitely

Suppose **hc(f)** worked for all **f** Create this **BFF**:

```
def BFF():
   if hc(BFF) == True:
     while 1+1==2: print 'Ha!'
   else:
     return # halt!
```

hc always 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! Bugs are inevitable Infinite loops are just **one** type of bug... In general, they're <u>all</u> undetectable Rice's Theorem: CS81 (all behavioral, not syntactic, bugs) Programming is *not automatable*... At least, not <u>bug-free</u> programming

$$kc(42) = 15 + 2 = 17$$

Name(s)

estimate?

What does **kc** (**x**) return for

each of these integers, **x**?

$$kc(9001) = 15 + 4 = 19$$

$$kc(1\underline{000000}) = 15 + \underline{\hspace{1cm}}$$

kc (31415926...)

$$kc(1000042) =$$

$$kc(100000000) = \frac{9 \text{ zeros}}{}$$

1 billion digits of pi, as an integer

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

Extra Extra: Are there any integers x with kc(x) > 50,000?