Arguments

Results

5 → 16
42 → 17
43 → 17
100 → 18
1000 → 19
10000 → 20
100000 → 20
100005 → 22
100042 → 23
10000000 → 20
10000005 → 22
a "googol" → 100000...00000 (100 total zeros) → 22
a "googolplex" → 100000...00000 (1 googol total zeros) → 26

This seems alien!!!

`kc`	`kc(5) == 16`
`hc`	`hc(CS5) == True`

These are fun f'ns!

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

There *are* well-defined *mathematical functions* no program can compute!

This seems impossible...

An example!?

~~Anything expressed w/math...~~ it's computable!

~~Human-discernable patterns...~~ it's computable!

The ***complexity*** of an integer

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

100,000 zeros total

100000000...0000000000000000000000000

Each of these integers has the same number of digits

170117684...2000687282248885778560 1

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

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

The *complexity* of **x** is the length of the shortest zero-argument function that returns **x**.

"description"

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

Arguments (x)		Results kc(x)
5	→	16
42	→	17
43	→	17
100	→	18
1000	→	19
10000	→	20
100000	→	20

?

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

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

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

```
def BFF():

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

    x = 0
    while kc(x) < 50000:
        x += 1
    return x

bug = BFF()
```

Suppose this version of **kc** is **42,000** characters long and claims to return the complexity of **x**

BFF?

```
def BFF():

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

    x = 0
    while kc(x) < 50000:
        x += 1
    return x

bug = BFF()
```

Suppose this version of **kc** is **42,000** characters long and claims to return the complexity of **x**

Proof

Note that this loop uses **kc(x)**, not **x** itself.

(1) Does the **while** *loop stop*
- [A] When **kc(x)** < 50,000 or
- [B] When **kc(x)** >= 50,000 or
- [C] It *never* stops

(1)

(2) **kc** *claims the* **x** *that BFF returns is*
- [T] An int **x** whose complexity is < 50,000
- [U] An int **x** whose complexity is >= 50,000
- [V] Nothing—it never returns at all

(2)

(3) Why does this mean **kc** *has a bug?*
- [F] Because **x != bug**
- [G] Because **bug**'s complexity *is* 42,070—*or less*!
- [H] Because there were no aliens involved

(3)

```
def BFF():

    def kc(x):
        do stuff…
        return answer

    x = 0
    while kc(x) < 50000:
        x += 1
    return x

bug = BFF()
```

Suppose this version of **kc** is 42,000 characters long…

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

But we allowed it to do anything at all!

kc *lied!*

So, every implementation of **kc(x)** contains a bug!

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.

Any Python function

```
def hc(f):
```

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!
```

Is **hc(BFF) == True** ?

Is **hc(BFF) == False** ?

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
(all *<u>behavioral</u>*, not syntactic, bugs)

Rice's Theorem: CS81

Programming is ***not automatable***…

At least, not *<u>bug-free</u>* programming kc(42) = 15 + 2 = 17

Name(s) ____________________

Quiz!

kc(9001) = 15 + 4 = 19

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

kc(1000000) = 15 + ____
(a million; 6 zeros)

kc(100...000) =
(a googolplex; 1 googol zeros)

kc(1000042) =
(a million and 42)

kc(31415926...) =
(1 billion digits of pi, as an integer)

estimate?

kc(1000000000) =
(a billion; 9 zeros)

kc(10000...0000) =
(a googol; 100 zeros)

kc(86753098...) =
(1 million patternless digits)

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

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

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