The Halting Problem

- it's important to know what we can and can't compute
- It turns out that we cannot create a program that can check *all* other programs for infinite loops
- see, e.g., http://en.wikipedia.org/wiki/Halting_problem
- First: demonstrate that we can write programs that create and execute new programs/functions. testProgramOnInput.py
- Informal proof that we can't write doesItHalt
  - why can't we create fully correct doesItHalt function? (doesItHalt.py)
  - To see why, consider function test in doesItHaltTest.py

Halting Problem

- Consider trying to write program

  doesItHalt(programString, dataString)
  where
  programString is a string representation of a program, e.g. “def foo(n): \n\treturn(n+1)”
  and dataString represents the input to that program

  such that doesItHalt returns
  “Yes” if the program would halt on the specified input, and
  “No” if the program would not halt (i.e. would go into an infinite loop)

- *It is not obvious that such a program can’t be written.* But it should be clear that doesItHalt can’t simply execute the specified program on the specified input. (Why?) Instead, doesItHalt would need to rely on more sophisticated analysis

- *HOWEVER, we can prove that doesItHalt cannot exist*

doesItHalt cannot exist

Informal proof:

> Suppose doesItHalt exists. I.e. doesItHalt correctly determines/prints, for any possible program and input, whether or not the program halts on that input

Given assumption that doesItHalt exists,
we'll define function test as follows:

```
def test(programString):

  result = doesItHalt(programString, programString)

  if result == "No":
    print("I'm done (hey, in fact, I halt)")

  else:
    loopFinished = False
    while(not loopFinished):
      print ("I'm gonna live forever ...")
```

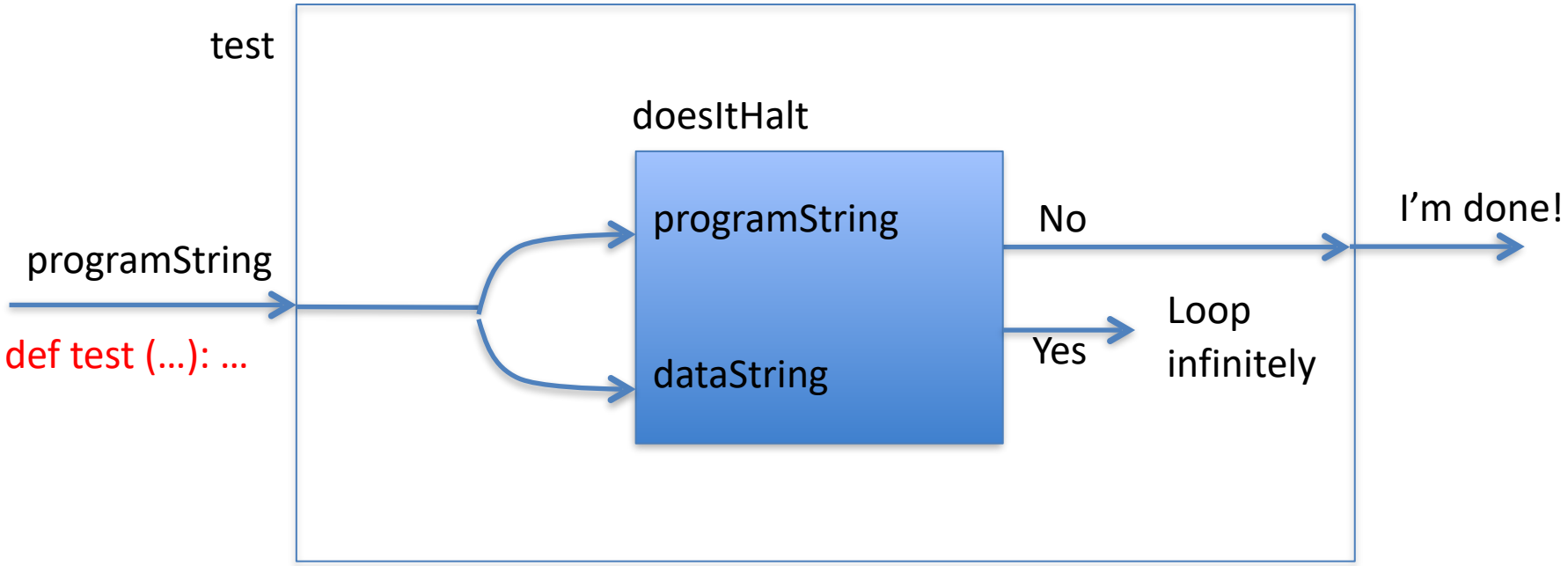
Consider: what happens when you execute test("def test …")?

doesItHalt does not exist

Informal proof:

1. Suppose doesItHalt exists (i.e. correctly states, for any possible program and input, whether or not program halts on that input)
2. Create function 'test' of previous slide. This is real Python code that works.
3. Now consider test("def test …")
   a. test("def test …") first executes doesItHalt("def test ..", "def test .."), saving returned value in variable result
   b. if result was "No" test("def test ..") clearly halts and returns.
   c. If result was "Yes" test("def test ..") clearly loops forever.
   d. BUT NOTICE! result would be "No" if doesItHalt determined that test("def test …") would not halt! And would be "Yes" if doesItHalt determined that test("def test …") would halt!
   e. THUS, test("def test …") halts if and only if test(def test …") does not halt!!
4. This is a contradiction, so we must conclude that the original assumption, that doesItHalt exists, is false.

Depiction of this on next slide might be easier to follow



When does test('def test …') print 'I'm done' (i.e. when does it halt)?

It halts when doesItHalt('def test …', 'def test …') returns No

But, by assumption, doesItHalt('def test …', 'def test …') returns No if and only if it determines that function test would not halt given 'def test …' as input.

Thus, test('def test …') halts if and only if test('def test…') does not halt.

When does test('def test …') loop infinitely? (i.e. when does it not halt)?

It loops infinitely when doestItHalt('def test …', 'def test …') returns Yes

But, by assumption, doesItHalt('def test …', 'def test …') returns Yes if and only if function test would halt given 'def test …' as input.

Thus, test('def test …') does not halt if and only if test('def test …') halts.

BOTH SITUATIONS LEAD TO A CONTRADICTION, SO THE ASSUMPTION THAT (A CORRECT) doesItHalt HALT EXISTS MUST BE FALSE.