The Counter Example

Consider the following Python code and discuss the questions below. The code is in the examples-state branch of the release repository.

```python
1 def mkInc(start=0):
2     i = start
3     def doit(ignore=0): -- for something later
4         nonlocal i
5         i = i + 1
6         return i
7     return doit
8
9 c1 = mkInc(0)
10 c2 = mkInc(10)

Problem 1) What is the scope of i in the mkInc function?

Problem 2) If we call c1(), it returns 1. If we call it again, it returns 2. If we then call c2(), we get 11. The variables i that were created in mkInc seem to be persistent. How does that happen?
The Delay Class

Here is a very special class. If you aren’t familiar with Python, the \_\_init\_\_ method is the constructor, and all methods have an initial argument self that refers to the current object. The format function replaces the {} in a string with its argument.

```python
class Delay:
    def \_\_init\_\_(self, action):
        self.action = action
        self.status = 0

    def report(x):
        print("Thunk executed: {}".format(x))
        return x

    def force(self):
        if self.status == 2:
            return self.value
        elif self.status == 0:
            self.status = 1
            self.value = Delay.report(self.action())
            self.status = 2
            return self.value
        else:
            return Exception("It broke!"
```

**Problem 3)** Python has dynamic typing, but what is the expected type of action?

**Problem 4)** The status variable has three possible values: 0,1, and 2. What do they mean?

**Problem 5)** What do you think this code will print?

```python
def plus(a, b, c):
    return a.force() + b.force()

d1 = Delay(lambda: 1+1)
d2 = Delay(lambda: 2+2)
d3 = Delay(lambda: 3+3)
plus(d1, d2, d3)
plus(d1, d2, d3)
```
Lazy Lists

Consider this function and list definition.

```python
def lazyTake(n, x):
    if x==() or n<1:
        return ()
    else:
        return (x[0], lazyTake(n-1, x[1].force()))
```

```python
l1 = (2, Delay(lambda: (3, Delay(lambda: (5, Delay(lambda: ()))))))
one = (1, Delay(lambda: ones))
```

**Problem 6)** How does this code implement lists? Note there are lazy lists and eager lists represented here.

**Problem 7)** How does that definition of `ones` work? Python normally does not allow recursive definitions...

**Problem 8)** Using this technique, write `lazyTail` and `lazyMap`
Problem 9) Write `lazyZipWith`.

Problem 10) Use these functions to make the infinite list of natural numbers `nats` and Fibonacci numbers `fib`. 