Objectives

In this activity, you will:

- implement objects using two separate techniques
- implement inheritance

Part 1 --- A Counter

Here is a counter, similar to the one in the state lecture.

```python
def mkInc(init=0):
    ct = init

def inc(delta=1):
    nonlocal ct
    ct = ct + delta
    return ct

return inc

s1 = mkInc()
```

**Problem 1)** We mentioned last time (or should have, anyway) that objects and closures are very similar. In what way does `mkInc` resemble an object? What is the constructor? What are methods? Is there private and public data?

**Problem 2)** What is missing from this story about objects?
Part 2 --- Multiple Methods

Here is a trick to introduce multiple methods.

```python
0 def mkInc(init=0):
1    ct = init
2    def inc(delta=1):
3        nonlocal ct
4        ct = ct + delta
5        return ct
6    def reset(init=0):
7        nonlocal ct
8        ct = init
9        return ct
10    return (inc,reset)
11
12 (c2,r2) = mkInc()
```

**Problem 3)** We now have multiple methods! Would you be happy programming with an object system like this? Why or why not?

**Problem 4)** Add a method `dec` to this that decrements a counter.
Dictionaries greatly improve our quality of life.

```python
0 def mkInc(init=0):
1    ct = init
2    def inc(delta=1):
3       nonlocal ct
4       ct = ct + delta
5       return ct
6    def reset(init=0):
7       nonlocal ct
8       ct = init
9       return ct
10   return { "inc": inc, "reset": reset }
```

**Problem 5)** How would you add `dec` to this version of objects?

**Problem 6)** Using a dictionary allows us to simulate inheritance. Here is some starter code: try to write a ```class``` `mkFastInc` that doubles the increment each time the `inc` method is called. It should call the superclass methods whenever possible.

```python
0 def mkFastInc(init=0):
1    superInc = mkInc(init)
2    --- your code here
```

**Problem 7)** Suppose you wanted to be able to access the state of the superclass directly. What options do you have to do that?