OBJECT-ORIENTED PROGRAMMING AND NONLOCALITY

COMPUTER SCIENCE MENTORS 61A

October 3 to October 7, 2016

1 Object Oriented Programming

1. **(H)OOP**
   Given the following code, what will Python output for the following prompts?

```
class Baller:
    all_players = []
    def __init__(self, name, has_ball = False):
        self.name = name
        self.has_ball = has_ball
        Baller.all_players.append(self)

    def pass_ball(self, other_player):
        if self.has_ball:
            self.has_ball = False
            other_player.has_ball = True
            return True
        else:
            return False

class BallHog(Baller):
    def pass_ball(self, other_player):
        return False
>>> anwar = Baller('Anwar', True)
>>> jerry = BallHog('Jerry')
>>> len(Baller.all_players)
```

Solution: 2

```
>>> Baller.name
```

Solution: Error

```
>>> len(jerry.all_players)
```

Solution: 2

```
>>> anwar.pass_ball()
```

Solution: Error

```
>>> anwar.pass_ball(jerry)
```

Solution: True

```
>>> anwar.pass_ball(jerry)
```

Solution: False

```
>>> BallHog.pass_ball(jerry, anwar)
```

Solution: False

```
>>> jerry.pass_ball(anwar)
```

Solution: False

```
>>> jerry.pass_ball(jerry, anwar)
```

Solution: Error

2. **TeamBaller**
   Write `TeamBaller`, a subclass of `Baller`. An instance of `TeamBaller` cheers on the team every time it passes a ball.
   Hint: What can we use to avoid writing duplicate code?
   "Super" Hint: There are two ways to implement `pass_ball`.

```
>>> cheerballer = TeamBaller('Thomas', has_ball=True)
>>> cheerballer.pass_ball(jerry)
```

```
Yay!!!!
True
>>> cheerballer.pass_ball(jerry)
I dont have the ball :(
False
```

```
class TeamBaller(_________):
    def pass_ball(_________, _________):
        """*** Enter solution below ***"""
```

Solution:

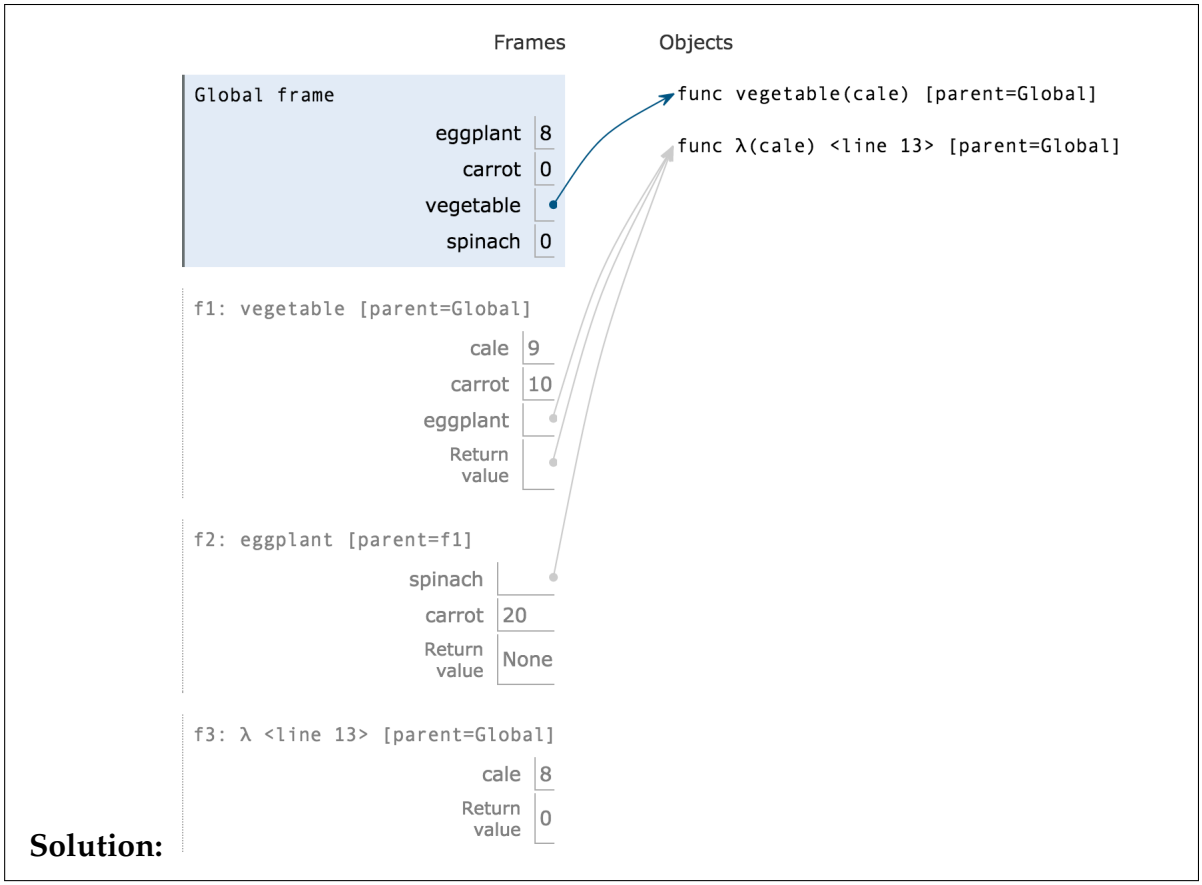
```
class TeamBaller(Baller):
    def pass_ball(self, other_player):
        did_pass = Baller.pass_ball(self, other_player)
        if did_pass:
            print('Yay!!!!')
        else:
            print("I dont have the ball :(")
        return did_pass
```

2 Nonlocal

1. **Nonlocal Kale**
   Draw the environment diagram for the following code.

```
eggplant = 8
carrot = 0
def vegetable(kale):
    carrot = 10
    def eggplant(spinach):
        nonlocal eggplant
        nonlocal kale
        kale = 9
        carrot = 20
        eggplant = spinach
    eggplant(kale)
    return eggplant
spinach = vegetable(lambda kale: carrot*kale)(eggplant)
```

Solution:



2. **Pingpong again...**
   Implement a function `make_pingpong_tracker` that returns the next value in the pingpong sequence each time it is called.

```
def has_seven(k): # Use this function for your answer below
    if k % 10 == 7:
        return True
    elif k < 10:
        return False
    else:
        return has_seven(k // 10)
```

Solution:

```
def make_pingpong_tracker():
    index, current, add = 1, 0, True
    def pingpong_tracker():
        nonlocal index, current, add
        if add:
            current = current + 1
        else:
            current = current - 1
        if has_seven(index) or index % 7 == 0:
            add = not add
        index += 1
        return current
    return pingpong_tracker
```

3. **(Optional)** Instead of using nonlocal for pingpong, let's use OOP!

Solution:

```
class PingPongTracker:
    def __init__(self):
        self.current = 0
        self.index = 1
        self.add = True

    def next(self):
        if self.add:
            self.current += 1
        else:
            self.current -= 1
```

```
        if has_seven(self.index) or self.index % 7 == 0:
            self.add = not self.add
        self.index += 1
        return self.current
```

Notice how the OOP approach is very similar to the nonlocal solution. Instead of using `nonlocal`, we use `self.variable` and the code becomes exactly the same. We just store the data in a slightly different way. This implies that OOP and functions are pretty similar, and it turns out you can even write your own OOP framework using just functions and `nonlocal`!

In addition, there are a lot of Python specific features that can be written using functions or using classes. If you are interested, check out the powerful Python feature decorators, and note how we can write them both as functions and as classes!