EXAM PREPARATION SECTION 5

NONLOCAL AND OBJECT-ORIENTED PROGRAMMING

March 7 to March 8, 2018

1 Nonlocal Environment Diagram

1. **Vulcans (Sp15 Midterm 2 Q2)**
   Draw the environment diagram that results from executing the following code.

```
def live(long):
    def prosper(spock, live):
        nonlocal long
        if len(long) == 1:
            return spock+1
        long[1] = live(long[0])
        long = long[1:]
        prosper(long[0], abs)
        return spock[0]+1
    prosper(long, lambda trek: trek-3)
live([1, 4])
```

2 What Would Python Print?

1. **Class Hierarchy (Fa14 Midterm 2 Q1)**
   For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write `Error`., but include all output displayed before the error.

```
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'My job is to gather wealth'

class Proletariat(Worker):
    greeting = 'Comrade'
    def work(self, other):
        other.greeting = self.greeting + ' ' + other.greeting
        other.work() # for revolution
        return other

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

Expression	Interactive Output
`5*5`	25
`1/0`	Error
`Worker().work()`	
`jack`	
`jack.work()`	

Expression	Interactive Output
`john.work()[10:]`	
`Proletariat().work(john)`	
`john.elf.work(john)`	

3 Code Writing

1. **What color is it? (Sp15 Midterm 2 Q4a)** Implement the `look` method of the `Dress` class. The `look` method returns a `Dress` instance's current `color` when the number of times that the instance's `look` method has ever been invoked evenly divides the total number times that the `look` method of any `Dress` instance has ever been invoked. Otherwise, the instance's `color` changes to the most recently returned color from any call to `look`, and `None` is returned.

```
class Dress:
    """What color is the dress?
    >>> blue = Dress('blue')
    >>> blue.look()
    'blue'
    >>> gold = Dress('gold')
    >>> gold.look()
    'gold'
    >>> blue.look()  # 2 does not evenly divide 3; changes to gold
    >>> Dress('black').look()
    'black'
    >>> gold.look()  # 2 does not evenly divide 5; changes to black
    >>> gold.look()  # 3 evenly divides 6
    'black'
    >>> Dress('white').look()
    'white'
    >>> gold.look()  # 4 evenly divides 8
    'black'
    >>> blue.look()  # 3 evenly divides 9
    'gold'
    """
    seen = 0
    color = None

    def __init__(self, color):
        self.color = color
        self.seen = 0

    def look(self):

        _____ = _____

        _____ = _____

        if _____:

            _____ = _____

            return _____

        else:
            _____ = _____
```

2. **Cucumber (Fa15 Midterm 2 Q4)** Cucumber is a card game. Cards are positive integers (no suits). Players are numbered from 0 up to `players` (0, 1, 2, 3 in a 4-player game). In each `Round`, the players each `play` one card, starting with the `starter` and in ascending order (player 0 follows player 3 in a 4-player game). If the `card` played is as high or higher than the `highest` card played so far, that player takes `control`. The winner is the last player who took control after every player has played once. Implement `Round` so that `play_round` behaves as described in the doctests below. Part of your score on this question will be assigned based on *composition* (don't repeat yourself).

```
def play_round(starter, cards):
    """Play a round and return all winners so far. Cards is a list of pairs.
    Each (who, card) pair in cards indicates who plays and what card they play.
    >>> play_round(3, [(3, 4), (0, 8), (1, 8), (2, 5)])
    [1]
    >>> play_round(1, [(3, 5), (1, 4), (2, 5), (0, 8), (3, 7), (0, 6), (1, 7)])
    It's not your turn, player 3
    It's not your turn, player 0
    The round is over, player 1
    [1, 3]
    >>> play_round(3, [(3, 7), (2, 5), (0, 9)]) # Round is never completed
    It's not your turn, player 2
    [1, 3]
    """
    r = Round(starter)
    for who, card in cards:
        try:
            r.play(who, card)
        except AssertionError as e:
            print(e)
    return Round.winners

class Round:
    players, winners = 4, []

    def __init__(self, starter):
        self.starter, self.player, self.highest = starter, starter, -1

    def play(self, who, card):

        assert _____________________, 'The round is over, player '+str(who)

        assert _____________________, "It's not your turn, player "+str(who)

        self.player = ____________________________________________________

        if card >= self.highest:

            _____________ , _____________ = _____________ , _____________

        if ____________________________________________________________:

            self.winners.append(self.control)

    def complete(self):

        return ____________________________________________________________
```