Vulcans - Nonlocal Environment Diagram (Spring 2015 Midterm 2 Q2)

Video Walkthrough: https://www.youtube.com/watch?v=cBemCW_uV8g

2. (12 points) Vulcans

(a) (8 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You may not need to use all of the spaces or frames.

A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

Remember: Do not add a new frame when calling a built-in function (such as abs). The built-in abs function is always written as func abs(...) [parent=Global].

```
1 def live(long):
2   def prosper(speck, live):
3     nonlocal long
4     if len(long) == 1
5     return speck+1
6     long[1] = live(long[1])
7     long = long[1]
8     prosper(long[1], abs)
9     return speck+1
10    prosper(long, lambda trek: trek-1)
11    live(1, 41)
```
1. (12 points) Class Hierarchy

For each row below, write the output displayed by the interactive Python interpreter when the expression is evaluated. Expressions are evaluated in order, and expressions may affect later expressions.

Whenever the interpreter would report an error, write ERROR. You should include any lines displayed before an error. Reminder: The Interactive Interpreter displays the repr string of the value of a successfully evaluated expression, unless it is None. Assume that you have started Python 3 and executed the following:

```python
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'
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5*5</td>
<td>25</td>
</tr>
<tr>
<td>1/0</td>
<td>ERROR</td>
</tr>
<tr>
<td>Worker().work()</td>
<td>'Sir, I work'</td>
</tr>
<tr>
<td>jack</td>
<td>Peon</td>
</tr>
<tr>
<td>jack.work()</td>
<td>'Maam, I work'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>john.work()[10:]</td>
<td>'Peon, I work</td>
</tr>
<tr>
<td></td>
<td>'to gather wealth'</td>
</tr>
<tr>
<td>Proletariat().work(john)</td>
<td>Comrade Peon, I work</td>
</tr>
<tr>
<td></td>
<td>Peon</td>
</tr>
<tr>
<td>john.elf.work(john)</td>
<td>'Comrade Peon, I work'</td>
</tr>
</tbody>
</table>
4. (12 points) What color is it?

(a) (6 pt) 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):
    Dress.seen += 1
    self.seen += 1

    if Dress.seen % self.seen == 0:
        Dress.color = self.color
        return self.color
    else:
        self.color = Dress.color
4. (8 points) Cucumber

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).

```python
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, 6), (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 r.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 not self.complete(), 'The round is over, player ' + str(who)
        assert who == self.player, 'It’s not your turn, player ' + str(who)
        self.player = (who + 1) % self.players
        if card >= self.highest:
            self.highest, self.control = card, who
        if self.complete():
            self.winners.append(self.control)
    def complete(self):
        return self.player == self.starter and self.highest > -1
```