CS61A Exam Prep 5

We will start at Berkeley Time!

Spring 2019 Midterm 2 Q3

3. (5 points) Deep lists

Implement `in_nested` which takes in a value `v` and a nested list or an individual value `L` and returns whether the value is contained in the list.

Hint: The built-in function `type` takes an object and returns the type of that object.

```
def in_nested(v, L):
    """
    >>> in_nested(5, [1, 2, [[3], 4]])
    False
    >>> in_nested(9, [[[1], [6, 4, [5, [9]]], 7], 7, 7])
    True
    >>> in_nested(1, 1)
    True
    """
    if type(L) != list:
        return L == v
    else:
        return any([is_nested(v,x) for x in L])
```

Fall 2015
Final
Q4

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).

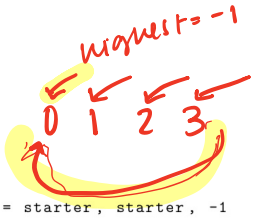
```
def play_round(starter, cards):
    """Play a round and return all winners so far. Cards is a list of pairs.
    Each (who, card) pair in cards indicates who plays and what card they play.

    >>> 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 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
```



Fall 2015 Final Q5a

5. (14 points) Grouper

(a) (4 pt) Implement `group`, which takes a one-argument function `f` and a list `s`. It returns a list of groups. Each group is a list that contains all the elements `x` in `s` that return equal values for `f(x)`. The elements in a group appear in the same order that they appeared in `s`. The groups are ordered by the order in which their first elements appeared in `s`.

```
def group(f, s):
    """Return a list of groups that contain all x with equal f(x).
    >>> five = [3, 4, 5, 2, 1]
    >>> group(lambda x: x % 2, five)
    [[3, 5, 1], [4, 2]]
    >>> group(lambda x: x % 3, five)
    [[3], [4, 1], [5, 2]]
    """
    a = []
    for b in map(f, s):
        if ____b not in a____:
            a.append(b)
    return [[____x for x in s if f(x) == b____] for b in a]
```

a = [1, 0]

b = 1

Fall 2020 Midterm 2 Q3

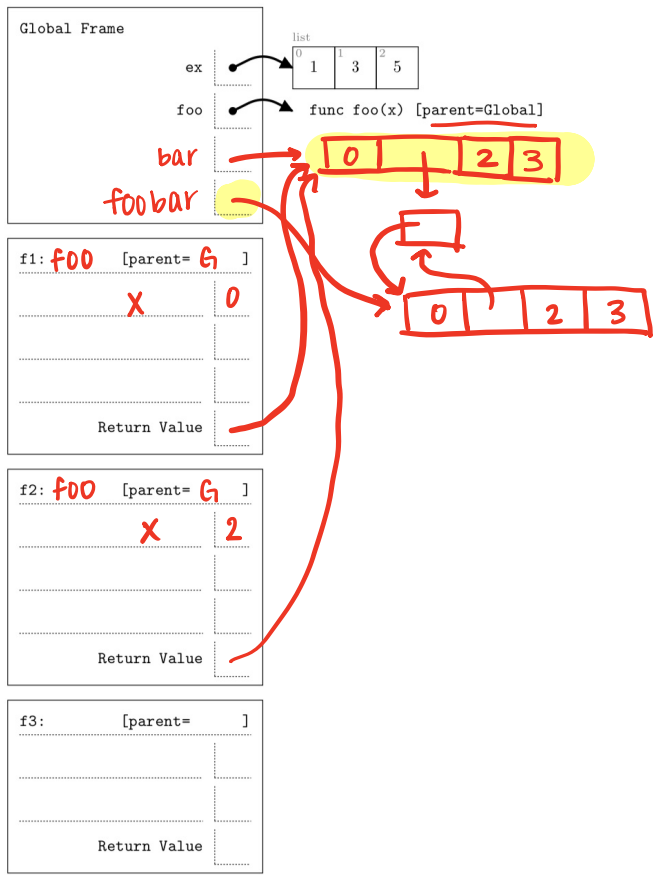
2. (15 points) Fly, you foos!

(a) (7 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 only need to show the final state of each frame. *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.

```
1  ex = [1, 3, 5]
2
3  def foo(x):
4      bar[x] = x
5      bar.append(len(bar))
6      return bar
7
8  bar = [1, [2]]
9
10 foo(0)
11 foo(2)
12 foobar = bar[:]
13 foobar[1][0] = foobar
```



go.cs61a.org/melanie-feedback