UNOFFICIAL QUIZ *for* PRACTICE SOLUTIONS

QUIZ 6

Computer Science 61A . October 8, 2015 . alvinwan.com/cs61a

This quiz will not count towards your grade. It exists to simply gauge your understanding. You will have 5 minutes to complete this quiz. In that timespan, your goal is to complete one question and at least attempt the other two.

01. OOP AND NONLOCAL

We have two ways of maintaining state: `nonlocals` and objects. In the following question, we will see how either approach can be used for the same functionality. Implement both `class Lo` and the function `lo`, so that both yield the same results. (**Hint**: Like `str(obj)` is equivalent to `obj.__str__()`, we know that `obj()` is equivalent to `obj.__call__()`)

```
>>> yo = lo()  # should be able to replace lo with Lo for identical results
>>> yo('report') # called once
1
>>> yo()('report') # called 1 + 2 = 3 times
3
>>> yo()()('report') # called 3 + 3 = 6 times
6
```

```
def lo():
  """ lo returns a function that
  prints the number of times
  the function was called.
  """
  n = 0
  def helper(report=None):
    nonlocal n
    n += 1
    if report:
        return n
    return helper
  return helper
```

```
class Lo:
  """ Creates objects that returns the
  number of times it was called, when
  asked to report.
  """

  def __init__(self):
    self.n = 0

  def __call__(self, report=None):
    self.n += 1
    if report:
      return self.n
    return self
```

UNOFFICIAL QUIZ *for* PRACTICE SOLUTIONS

02. OOP AND MUTABILITY

Kristin and Sammy are responsible for the Ants project, but they've inherited faulty code from me! They can't identify the bugs in the following code. For all three bugs, identify (1) the error, (2) a scenario that would expose the bug, and (3) how to fix it.

```
create_creature = lambda name, food=[]): [name, food] # all food lists are the same one
change_creature_name = lambda creature, new_name: [new_name] + creature[1:]
give_creature_food = lambda creature, food: creature['food'].append(food)

class Game:
  creatures = []

  def start(self):
    creatures.append(create_creature('Sumukh')) # all games share a list of creatures

  def bobify_all(self):
    for creature in creatures:
      creature = change_creature_name(creature, name) # does not modify original list

  def feed_all(self, food):
    for creature in creatures:
      give_creature_food(creature, food)
```

03. TREES

Assume that we have the standard functions for a tree abstraction (`tree(root, branches=[])`, `root(t)`, `branches(t)`, `is_leaf(t)`). Except, each node is a location, and each leaf is a tourist attraction. The value of a node represents its elevation above sea level. Write a function that gives us the maximum elevation for the path to a specified destination.

```
def get_max_elevation(t, dest):
    """
    t = tree(5, [tree(6, [tree(7), tree(4)]), tree(3, [tree(2), tree(1)])])
    >>> get_max_elevation(t, 2)
    5
    >>> get_max_elevation(t, 5)  # 5 is not a valid destination
    -1
    """
    if is_leaf(t) and root(t) == dest:
        return root(t)
    for b in branches(t):
        highest = get_max_elevation(b, dest)
        if highest:
            return max(root(t), highest)
    return False
```