Solutions to Prelim 2 Review Questions

May 16, 2016

See also the associated python file(s), which in some cases provide(s) testing code that you can run on your own implementations, or sample solutions you can try horsing around with — we encourage you to make changes and see their effects.

1 Functions on Lists

(a) The *even-odd* sort of a list that has even length permutes entries so that all the even-index entries come first followed by all the odd-indexed entries. To illustrate, suppose we have the following length-8 list:

'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'

Here are the length-4 lists of the even-indexed entries and the odd-indexed entries:

'a'	'c'	'e'	'g'

'b'	'd'	'f'	'h'

And here is the even-odd sort of the above length-8 list:

'a'	'c'	'e'	'g'	'b'	'd'	'f'	'h'

This operation *could* — but for this question you are *not* allowed to do so — can be carried out very simply using list slicing and list concatenation: indeed, if `x` has length `n` and `n` is even, then the list `x[0:n:2] + x[1:n:2]` is the even-odd sort of `x`. Implement the following procedure so that it performs as specified, *using just for-loops and subscripting. No list slicing or list concatenation allowed.*

```
def EvenOddSort(x):
    """ Performs an even-odd sort of x

    Precondition: x is a list with even length"""
```

Note that `EvenOddSort` does not return any values. Again, no list slicing or list concatenation allowed.

For a hint on important and potentially common errors, see this footnote.[1]

Solution:

```
evens = []; odds = []
for i in range(len(x)/2): # Ask yourself: why not ``for i in x''?
    # ['a', ...'h']: range(4) == [0,1,2,3]
    ind = i*2 # The even index we want to look at
    evens.append(x[ind])
    odds.append(x[ind+1])

### An alternate implementation of the above loop (not clear whether
```

[1] Don't change the list *while* constructing an even-odd-sorted version! And if y is your even-odd sorted version, don't just do `x=y`! (Why?)

```
    ### the use of ``range'' violates the ``no slicing'' rule)
        evens.append(x[ind])
        odds.append(x[ind+1])



    ## In order for the changes to ``last'' outside this function,
    # we need to change each entry of x.
    ###
    ### YOU CAN'T JUST DO ``x = evens.extend(odds)'' AND GET THE CHANGES
    ### TO ``LAST''!
    ###

    for ind in range(len(evens)):
        x[ind] = evens[ind]
        x[ind + len(evens)] = odds[ind]
```

(b) Assuming that the procedure `EvenOddSort` is available, implement the following function so that it performs as specified:

```
def MultipleSort(x,N):
    """ Returns a list obtained by performing N even-odd
    sorts of the list x. The list x is not altered.

    Precondition: x is a list with  even length and N is a positive int.
    """
```

Use a loop that calls `EvenOddSort` `N` times. (Don't try to do some fancy "if `N` is even, I'll get the same list back" type of reasoning.)

Some notes on potential errors in this footnote.[2]

Solution:

```
   copy = list(x)  # Ask yourself: why make a copy?
    for i in range(N):
        EvenOddSort(copy)
    return copy
```

The line `copy = list(x)` above is equivalent to the following for-loop:

```
    copy = []
    for ind in range(len(x)):
        copy.append(x[ind])
```

2 Farthest Point

Assume the existence of the following class, and that the command `import math` has been included beforehand.

```
    class Point(object):
       """ Attributes:
              x   the x-coordinate   [float]
```

[2] `EvenOddSort` doesn't return anything. Don't operate on `x` or you'll change it.

```
           y   the y-coordinate   [float]
    """
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def Dist(self,other):
        """ Returns a float that is the distance from self to other.

        Precondition: other is a Point
        """
        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)
```

Complete the following function so that it performs as specified

```
def FarthestPt(L,idx,P)
   """ Returns an integer j with the property that the distance from
   L[j] to P is maximum among all the ***unvisited*** points.

   If idx[i] = 1, then we say that L[i] has been visited. If idx[i] = 0, then
   we say that L[i] is unvisited.

   Preconditions: L is a list of references to Point objects, P is a reference
   to a point object, and idx is a list of ints that are either zero or 1. The
   lists idx and L have the same length and idx has at least one zero entry.
   """
```

Solution: One implementation:

```
ind = idx.index(0) # find index of first unvisited (guaranteed to exist)
max_d = P.Dist(L[ind]) # initialize max found so far
max_d_ind = ind

for ind in range(max_d_ind+1, len(L)):
    if idx[ind] == 0: # unvisited, check
        if P.Dist(L[ind]) > max_d:
            max_d = P.Dist(L[ind])
            max_d_ind = ind
return max_d_ind
```

Alternate version, different approach to initialization

```
d = 0 # max distance found so far
for j in range(len(L)):
    dj = P.Dist(L[j])
    if idx[j]==0 and dj>d:
        k = j # index of the farthest point found so far
        d = dj
return k # will be defined because PreC guarantees an unvisited node
```

3 Nested Loops

1. What is the output if the following is executed?

```
s = "abcd"
for i in range(4):
    for j in range(i+1,4):
         print i, j, s[i]+s[j]
```

Solution: "For each position i in s, print letter pairs from s where the first letter is at position i and the next letter is one after position i".

```
0 1 ab
0 2 ac
0 3 ad
1 2 bc
1 3 bd
2 3 cd
```

2. For each key in dictionary D, write down the key and corresponding value in D.

```
D1 = {'a':'one', 'b':'two', 'c': 'three', 'd':'four'}
D2 = {'c':'five', 'd':'six', 'e': 'seven' ,'f':'eight'}
D = {}
for d in D1:
   D[d] = D1[d]
for d in D2:
   D[d] = D2[d]
```

Solution:

(It wouldn't matter what order you put the following lines.)

```
a one
b two
c five
d six
e seven
f eight
```

4 More work with lists, which are objects

(a) If the following is executed, then what are the first five lines of output?

```
x = [10,20,30]
for k in range(1000):
     print "k:", k, "x in the loop", x
     x.append(x[0])
     x = x[1:4]
```

Solution:

```
k: 0 x in the loop [10, 20, 30]
k: 1 x in the loop [20, 30, 10]
k: 2 x in the loop [30, 10, 20]
k: 3 x in the loop [10, 20, 30]
k: 4 x in the loop [20, 30, 10]
```

(b) If the following is executed, then what is the output? For full credit you must also draw two state diagrams. The first should depict the situation just after the `Q.x = 0` statement and the second should depict the situation just after the `P = Point(7,8)` statement.

```
P = Point(3,4)
Q = P
Q.x = 0
print Q.x, Q.y, P.x, P.y
P = Point(7,8)
print Q.x, Q.y, P.x, P.y
```

Solution: After `Q.x = 0` statement[3]:

```
P ---->   a point with x:0 (no longer 3), y:4
           ^
           |
Q ---------|
```

So the print output is: 0 4 0 4

After the `P = Point(7,8)` statement:

```
P ---> a new point with x:7, y:8

Q ---> the previous point with x:0, y:4
```

So the print output is 0 4 7 8

(c) If the following is executed, then what is the output?

```
x = [10,20,30,40]
y = x
for k in range(4):
   print "x is", x
   print "y is", y
   print "...."
   x[k] = y[3-k]
print x
```

[3] Nope, not gonna give up our day jobs to become ASCII artists.

Solution: Changes to x affect y. So x and y are always the same.

```
x is [10, 20, 30, 40]
y is [10, 20, 30, 40]
....
x is [40, 20, 30, 40]
y is [40, 20, 30, 40]
....
x is [40, 30, 30, 40]
y is [40, 30, 30, 40]
....
x is [40, 30, 30, 40]
y is [40, 30, 30, 40]
....
[40, 30, 30, 40]
```

5 Dictionaries

(a) Complete the following function so that it performs as specified

```
    def  F(s,D):
       """ Returns True if s is a key for D and every element in D[s] is also
       a key in D. Otherwise returns False.

       Precondition: s is a nonempty string and D is a dictionary whose keys are strings
       and whose values are lists of strings.

       """
```

Solution:

```
    if s not in D:
        return False
    for item in D[s]:
        if item not in D:
            return False
    return True
```

Incidentally, Python allows the following one-line version:

```
     return (s in D) and all(item in D for item in D[s])
```

6 Methods and Lists of Objects

Assume the availability of the following class:

```
class City(object):
    """
    attributes:
        name      the name of a city [str]
        high:     the record high temeratures [length-12 list of int]
        low:      the record low temperatures [length-12 list of int]
    """

    def __init__(self,cityName,theHighs,theLows):
        """Returns a reference to a City object
        PreC: cityName is a string that names a city.
        theHighs is a length 12 list of ints.
        theHighs[k] is the record high for month k (Jan is month 0)
        theLows is a length 12 list of ints
        theLowss[k] is the record high for month k (Jan is month 0)
        """
        self.name = cityName
        self.high = theHighs
        self.low  = theLows
```

(a) Complete the following method for the class `City` so that it performs as specified.

```
    def HotMonths(self):
        """ Returns the number of months where the record high
        is strictly greater than 80.
        """
```

Solution:

```
        T = self.high
        # T is the list of temperature highs
        n = 0
        for temp in T:
            if temp>80:
                n+=1
        return n
```

(b) Complete the following method for the class `City` so that it performs as specified. Your implementation must make effective use of the method above.

```
    def Hotter(self,other):
        """Returns True if the city encoded in self has strictly more
        hot months than the city encoded in other.

        A month is hot if the record high for that month is > 80

        PreC: other is a city object
        """
```

Solution:

```
        # What if you didn't have the parentheses?
        return self.HotMonths() > other.HotMonths()
```

(c) Complete the following method for the class `City` so that it performs as specified.

```
    def Variation(self):
        """ Returns a length 12 list of ints whose k-th entry
        is the record high for month k minus the record low for month k.
        """
```

Solution:

```
        d = []
        for k in range(12):
            diff = self.high[k]-self.low[k]
            d.append(diff)
        return d
```

(d) Complete the following method for the class `City` so that it performs as specified.

```
    def Exaggerate(self):
        """ Modifies self.high so that each entry is increased by 1
        and modifies self.low so that each entry is decreased by 1
        .
        """
```

Solution: This question tests whether you can access and change attributes.

```
         for k in range(12):
            self.high[k] += 1 # what happens if you delete "self."?
            self.low[k] -= 1
```

(e) Complete the following function so that it performs as specified. Assume that the methods in parts (a) and (b) are available; your implementation must make effective use of them.

```
    def Hottest(C):
        """ Returns an item from C that represents the city that
        has the most hot months.
        PreC: C is a list of references to City objects
        """
```

Solution:

```
    cMax = C[0]

    for c in C: # redundant check for C[0] but who cares?
        if c.Hotter(cMax):
            cMax = c
    return cMax  # Note that you can return objects
```

Note that there's no need to explicitly keep track of what the max temperature found so far actually is; keeping track of the object that "scored" the max temp suffices.