CS 88 Midterm Solutions [Fa 20]

Blank exam

Question 1: What Made Python Print That

A)
```
>>> vals = [10, 50, 60, 20, 40, 30]
>>> y = [vals[idx] + idx for idx in range(len(vals))]
>>> y
[10, 51, 62, 23, 44, 35]
```

B)
```
>>> def mystery1(n):
...     if n > 20:
...             return n - 15
...     else:
...             return n * 3
>>> mystery1(mystery1(secret_num))
36
```

What are three (out of 4 total) possible values for the variable *secret_num*?

Answer: any 3 of the following: 66, 27, 17, 4

We can solve this problem by working backwards. We first need to find out what values satisfy mystery1(x) = 36. This is satisfied when x = 51 and x = 12.

Now we proceed to satisfying the outer mystery1 call. We now need to find out what values satisfy mystery1(y) = 51 and mystery1(z) = 12. For y, we will see that it can either be 66 or 17, and for z we will see that it can either be 4 or 27.

Verifying our answer below:

mystery(mystery(4)) = mystery(12) = 36
mystery(mystery(17)) = mystery(51) = 36
mystery(mystery(27)) = mystery(12) = 36
mystery(mystery(66)) = mystery(51) = 36

C) Assume lst is a list of 3 integers.

```
>>> lst = [_____, _____, _______]
>>> (lst[2] - lst[0]) or (lst[1] < 15) or (lst[0] % 10 != 2) or "hi"
```

Part 1: If lst[1] was 33, the expression would Sometimes evaluate to True.

Part 2: If lst[0] was 21, the above expression would Never evaluate to False.

Part 3: Fill in the list lst with 3 integers such that the expression on the second line in the original question evaluates to "hi".

Express your answers as [... , ... , ...]. Briefly explain how you approached this problem (answers without a proper explanation will not receive credit).

Answer:
One example: [2, 33, 2]

To evaluate to 'hi' which is a Truthy value, all the previous expressions must evaluate to a Falsy value.

To do this,
lst[0] and lst[2] should be equal, lst[1] should be greater than or equal to 15, and the last digit of lst[0] should be 2,

Question 2: Chi-pot-el (Reverse Environment Diagram)

A)

```
f1 = lambda x: x + 1
f2 = lambda y: [y]
x = f2(f1(2))
```

B)

```
def chipotle(rice, bowl):
   bowl.append(2)
   def burrito(chips):
       if chips > 6:
           beans = rice(bowl)
           return beans * chips
       else:
           return bowl[:]
   return burrito
dip = chipotle(sum, [1, 1, 1])
guac = dip(10)
salsa = dip(4)
```

Question 3: I've had enHOF of HOFs

In this question you will be building a function that checks if calling 3 different functions on the same input will yield the same result.

A) The comparer function takes in a number x and returns a HOF. This HOF returns another function that takes in the first input function and returns a second function. This second function takes in a second input function and returns a third function. The third function takes in a third input function and returns True if all 3 input functions return the same value when called on x.

Fill out the skeleton code for the comparer function.

```
def comparer(x):
    """
    >>> f1 = lambda x: 2*x
    >>> f2 = lambda x: x**2
    >>> f3 = lambda x: x + 2
    >>> comp = comparer(2)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    True
    >>> comp = comparer(10)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    False
    """
    def hof1(f1):
        def hof2(f2):
            def hof3(f3):
                return f2(x) == f3(x) == f1(x)
           return hof3
        return hof2
    return hof1
```

B) Rewrite the question in one line using lambda expressions:
(You can use both lines, but your answer must be one expression)

```
def comparer(x):
    return lambda f1: lambda f2: lambda f3: f2(x)==f3(x)==f1(x)
```

Question 4: Subset Summing

Given a list and an index, find the sum of all the summations of each subset of the list starting from the given index. For example, subsetSum([1, 2, 3], 1) would return 7. This is because the two subsets of the list starting at index 1 are [2] and [2, 3]. Then summing the sums of each subset we get 2 + 5 = 7. You can assume that a valid index will always be passed in.

```
def subsetSum(lst, i):
    """
    >>> subsetSum([1, 2, 3], 1)
    7
    >>> subsetSum([1], 0)
    1
    >>> subsetSum([1, 2, 3, 4, 5], 2) #the subsets are [3],
[3, 4], and [3, 4, 5]
    22
    """
    total = 0
    for j in range(i + 1, len(lst) + 1):
        currentSum = sum(lst[i:j])
        total += currentSum
    return total
```

Question 5: Eat, Sleep, Debug, Repeat

Question:

You are trying to make a function `tester` which takes in an integer `n`, a list of functions `fn_lst`, and a conditional function (a one argument function that returns a boolean) `condition`. The function tester runs each of the functions from `fn_list` on the integer `n`, and returns a list of all of the values for which `condition` returns `True`. However, as you are programming you find that your program is buggy. Find 3 bugs in the following program.

```
def tester(n, fn_list, condition):
    """
    >>> add_one = lambda x: x + 1
    >>> negate = lambda x: -x
    >>> double = lambda x: x * 2
    >>> is_positive = lambda x: x > 0
    >>> tester(88, [add_one, negate, double], is_positive)
    [89, 176] #-88 is not included because it is negative
    """
    new_list =[]
    for fn in fn_list:
        i = fn(n)
        if condition(n):
            new_list + i
        else:
            return new_list
```

Solution:

```
def tester(n, fn_list, condition):
    new_list = []
    for fn in fn_list:
        i = fn(n)
        if condition(i):
            new_list.append(i)
    return new_list
```

Explanation:

The three bugs are

1) condition(i) instead of condition(n)
    a) we want to call the condition function on the fn(n) instead of n
2) new_list.append(i) instead of new_list + i
    a) new_list + i does not mutate new_list
    b) Also you cannot add a list and an integer
    c) Other acceptable answers: new_list.extend([i]) new_list+ = [i]
3) Return new_list after the for loop
    a) If we return inside the for loop the function will potentially terminate early
    b) We want the for loop to run through each i before returning
4) Mismatched/misnamed arguments
    a) This was an unintentional bug on some versions of the exam, but we still allowed it to be used as one of the 3 bugs

Question 6: DDDuplicate EEElements

In this question you will be building a recursive function that, given a list and a non-negative integer count, will return a new list with count duplicates of each item in list.

Hint: `['a'] * 3` will return `['a', 'a', 'a']`

```
def duplicateElements(lst, count):
    """
    >>> duplicateElements([1, 2, 3, 4], 3)
    [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]
    >>> duplicateElements(['a', 'b', 'c'], 2)
    ['a', 'a', 'b', 'b', 'c', 'c']
    """
    if len(lst) == 0:
        return []

    first = [lst[0]] * count
    rest = duplicateElements(lst[1:], count)

    return first + rest
```

Question 7: The FinaLIST

You are in the finale of an exciting game called Treasure. You're given a board (represented as a list `lst`) that contains the amount of gold you can collect at each index, and two starting indices that you are considering (`start_idx_1` and `start_idx_2`).

After choosing a starting index, you must move towards the right of the board to collect gold, skipping one spot every time, (see the graphic below). Find which of the two starting indices will earn you more treasure.

Your final return value should be a 2 element list containing the better starting index between the two options and the amount of treasure you would accumulate starting from there. You can assume the input list only contains positive numbers.

```
def better_starting_position(lst, start_idx_1, start_idx_2):
      """
    >>> board1 = [10, 11, 12, 13, 14, 15]
    >>> better_starting_position(board1, 2, 3)
    [3, 28]
    >>> board2 = [10, 100, 10, 90, 10, 100, 10]
    >>> better_starting_position(board2, 1, 4)
    [1, 290]
    """
    total1 = 0
    total2 = 0

    for i in range(start_idx_1, len(lst), 2):
        total1 += lst[i]
    for i in range(start_idx_2, len(lst), 2):
        total2 += lst[i]

    if total1 > total2:
        return [start_idx_1, total1]
    else:
        return [start_idx_2, total2]

    ------------ ALTERNATIVE SOLUTION ------------

    total1 = sum([lst[i] for i in range(start_idx_1, len(lst), 2)])
    total2 = sum([lst[i] for i in range(start_idx_2, len(lst), 2)])
    if total1 > total2:
        return [start_idx_1, total1]
    else:
        return [start_idx_2, total2]
```

Question 8: Cycloaddition

Create a function `cycle_add` that takes in a 2d list of ints in which the int lists can be of different lengths. It returns a list of the result of summing up the values of each of the lists elementwise, cycling if necessary (if the list is too short). For example cycle_add([[1,2], [3,5,7]]) will return [4, 7, 8], the 4 is the result of 1 + 3, the 7 is the result of 2 + 5, 8 is the result of 1 + 7 (notice how we cycled back to the first value of the first list to get 1).

```
def cycle_add(lsts):
    """
    >>> cycle_add([[1,2,3], [1]])
    [2, 3, 4] # [1+1, 2+1, 3+1]
    >>> cycle_add([[1,2], [3,5,7]])
    [4, 7, 8] # [1+3, 2+5, 1+7]
    >>> cycle_add([[4,2,0], [1,2], [2]])
    [7, 6, 3] # [4+1+2, 2+2+2, 0+1+2]
    """
    new_list = []
    longest = max([len(lst) for lst in lsts])
    for i in range(longest):
        value = 0
        for lst in lsts:
           index = i % len(lst)
            value += lst[index]
        new_list.append(value)
    return new_list
```