CIS 210
Winter 2014 Final Exam

Write your name at the top of **each page** before you begin. [5 points]

1. [5 points] What does `q1( )` print? (Recall that range(6) produces the integers from 0 to 5.)

```
def q1( ):
    result = 0
    for x in range(6):
        for y in range(6):
            if x == y:
                result += x
    print(result)
```

15. The most common error, despite the note in the question, was summing the integers from 1 to 6 instead of 0 to 5; a couple people summed 0 to 4 instead of 0 to 5. A second fairly common error was counting rather than summing. I typically gave 3 points partial credit if I could tell that the only error was one of these, and I took off just 1 point for simple arithmetic errors (e.g., adding the right numbers but getting the wrong result). I gave no partial credit if an answer indicated that someone didn't understand nested loops accumulating a single result.

2. [5 points] What does `q2( )` print?

```
VALUES = [("Red", 7), ("Blue", 5), ("Green", 2)]

def value_of(col):
    for entry in VALUES:
        c, v = entry
        if c == col:
            return v
    return 1

def score( color_list ):
    result = 0
    for color in color_list:
        result += value_of(color)
    return result

def q2():
    print(score(["Red", "Purple", "Green", "Silver"]))
```

11. This question was to see if you could read the functions and form simple, precise abstractions of them to use in reading other code. Most of you did pretty well. Several people did ok except that they seem to have equated Purple and Blue; I gave 3 points partial credit in that case. I gave only 1 point if you missed the default value of 1 for Purple and Silver, and I gave no points if you didn't seem to understand the overall logic.

(score)

3. [5 points] What does `q3( )` print?

```
def swap_elem(x, y, i):
    tmp = x[i]
    x[i] = y[i]
    y[i] = tmp

def sift(a, b):
    for i in range(len(a)):
        if i < len(b):
            if a[i] < b[i]:
                swap_elem(a, b, i)

def q3():
    a = [1, 2, 3, 4, 5]
    b = [0, 0, 0, 8, 8]
    c = a
    sift(a, b)
    print(c)
```

[1, 2, 3, 8, 8]
This question is mostly checking your understanding of multiple references to an object. You needed to understand that c and a are references to the same list object, and that they can be altered by swap_elem because the reference (pointer) is passed rather than a copy of the object. I did not give partial credit if it looked like you didn't understand those concepts. I did give partial credit for a couple of simple mistakes — notably the answer $[0, 0, 0, 4, 5]$ because I think it was from just reversing a and b or else reversing the test $a[i] < b[i]$ in sift.

(score)

4. [5 points] What does q4( ) print?

```
 class BigRedButton:
    """
    A big red button.  It can be hooked up to various machines.
    """

    def __init__(self):
        self.machines = [ ]

    def connect(self, machine):
        """Connect to a machine.
        Args:
           machine: a function with no arguments
        """
        self.machines.append(machine)

    def push(self):
        for machine in self.machines:
            machine()


def horn():
    print("Beep beep")

def hammer():
    print("Clang clang")

def q4():
    button = BigRedButton()
    button.connect(horn)
    button.push()
    button.connect(hammer)
    button.push()
```

Beep beep
Beep beep
Clang clang

I think most of you recognized this as a variation on the way we implemented the model-view-controller patter in the Sudoku project. An important point is that the notifiers (self.machines) are in a list, and all of the currently installed notifiers are called by button.push. I was pleased that most of you got this. The most common error was not repeating the "Beep beep", which could have been either from not understanding that the notifiers are in a list, or from not understanding the code at all and just printing the results of horn and hammer in the order they are written. (In retrospect, I should have placed hammer before horn so that I could distinguish these errors.) I gave 2 points partial credit if you printed "Beep beep" once instead of twice. I don't think any other answers got partial credit.

(score)

5. [10 points] Finish function `partition`, consistent with its docstring.

```
def partition(li, pivot):
    """
    Partition list li into two lists, containing the elements of li at
       most pivot and the elements of li greater than pivot,
       respectively.
    Args:
       li:  A list of integers
       pivot:  An integer
    Returns:
       A list L containing two sub-lists.  L[0] is a list of elements
       of li that are less than or equal to pivot.  L[1] is a list of
       elements of li that are greater than pivot.  Each element of li
       appears in exactly one of the two sub-lists of L.

    Examples:
        partition([-3, -2, -1, 0, 1, 2, 3], 0)  =  [[ -3, -2, -1, 0], [1, 2, 3]]
        partition([ 7, -3, 4, 16, -13, 12, 1 ], 2)  =  [[-3, -13, 1], [ 7, 4, 16, 12]]
        partition([ 1, 2, 3 ], 3)  =  [[ 1, 2, 3], [ ] ]
        partition([ ], 7)  =  [[ ], [ ]]
    """
    smaller = [ ]
    larger = [ ]
    for e in li:
        if e <= pivot:
            smaller.append(e)
        else:
            larger.append(e)
    result = [ smaller, larger ]
    return result
```

There were many reasonable variations on this, and many unreasonable variations too. A couple of people complained about L as a variable name, but L is not a variable name, it's just part of the description of the result. A common error was creating special cases that weren't necessary and actually gave incorrect results, e.g., appending extra items to the result if li is empty.

(score)

6. [15 points] Sometimes we want to know if we can pick a subset of list elements that add up to a certain target amount. Finish the following (recursive) function `can_pick`.

It may be useful to note that the empty list sums to 0, and that if any subset of $[a, b, c, ...]$ sums to k, then there must be a subset of $[b, c, ...]$ that sums to either k or $k - a$. Also, in Python, if $li = [a, b, c, ...]$, then $li[1:] = [b, c, ...]$

```
def can_pick(li, target):
    """Does some subset of elements in li sum to target?
    Args:
       li:  A list of positive integers
       target: a positive integer
    Returns:
       True iff there is a subset of elements in li whose sum is target.
    Examples:
       can_pick([2, 4, 6, 8], 10) = True because 4+6 = 10 (also 8+2)
       can_pick([1, 7, 9], 12) = False
       can_pick([8, 7, 6], 0) = True   because we can select none of the elements
       can_pick([ ], 0) = True, but can_pick([ ], 5) = False
    """
    if target == 0:
        return True
    if li == [ ]:
        return False
    if can_pick( li[1:], target - li[0]):
        return True
    if can_pick( li[1:], target):
        return True
    return False
```

I expected this problem to be of intermediate difficulty, between questions 5 and 7, because I essentially gave you the basis and progress cases in the question (*"It may be useful to note that . . . "*). Instead, this seems to have been the hardest problem for many of you. There were far too many approaches to list here. In general, I gave some credit if I could see that you were breaking down some of the basis and progress cases correctly. For example, if you got one of the basis cases and one of the progress cases, you might have received 5 points partial credit, or anywhere from 3 to 7 depending on how reasonable the rest of your code was.

(score)

7. [15 points] Write function `merge_squish` without using Python's built-in sort functions.

```
def merge_squish( a, b ):
    """Merge two sorted lists, keeping only one copy of duplicated elements.
    Args:
      a:  A list of integers, in order from smallest to largest, without duplicates
      b:  A list of integers, in order from smallest to largest, without duplicates
    Returns:
      A list of all the integers from a and b, in order from smallest to largest,
      without duplicates.
    Examples:
      merge_squish([1, 2, 4, 7 ], [2, 3, 5, 7]) = [1, 2, 3, 4, 5, 7]
      merge_squish([1, 2, 3], [1, 2, 3]) = [1, 2, 3]
      merge_squish([ ], [7, 8]) = [7, 8]
    """
    a_pos = 0
    b_pos = 0
    result = [ ]
    while a_pos < len(a) and b_pos < len(b):
        if a[a_pos] == b[b_pos]:
            result.append(a[a_pos])
            a_pos += 1
            b_pos += 1
        elif a[a_pos] < b[b_pos]:
            result.append( a[a_pos] )
            a_pos += 1
        else:
            result.append( b[b_pos] )
            b_pos += 1
    while b_pos <len(b):
            result.append( b[b_pos] )
            b_pos += 1
    while a_pos < len(a):
            result.append( a[a_pos] )
            a_pos += 1
    return result
```

Beginning programmers often have difficult designing loops that maintain more than one index. We've seen a few of these during the term, including the palindrome problem (indices for the beginning and end of the palindrome), and most recently for a very closely related problem, merging two sorted lists in linear time (but without squishing out duplicates). I thought many of you would remember our linear time merge algorithm from lecture, and be able to extend it to add the squishing. Only a few of you managed to do that.

I should have required a linear time algorithm (proportional to the sum of the lengths of a and b) for full credit. I didn't anticipate that many of you concatenate the lists (or their non-duplicate elements) and then try to sort them. Some of you put the items in order using the classic *insertion sort* algorithm, which requires quadratic time but is simple enough that you mostly got it right. Some of you built a variation of the *bubble sort* algorithm, but I don't think anyone got that right. Because I did not specify the required efficiency of your code, I gave full credit for those of you who wrote correct but very inefficient code. I gave partial credit, typically around 7 points, for reasonable but incorrect attempts at bubble sort.

(score)