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.)

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

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

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

```python
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)
```

```
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()
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) = [[], []]
    """
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 \(k - a\). Also, in Python, if \(li = [a, b, c, ...]\), then \(li[1:] = [b, c, ...]\)

```python
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
    """
```


```python
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]
    """
```