CIS 210
Winter 2014 Midterm Exam

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

1. [5 points] What does `q1( )` print?

```
def q1():
    li = [1, 2, 3, 4, 5, 6 ]
    ev = 0
    od = 0
    for n in li:
        if n % 2 == 0:
            ev += n
        else:
            od += n
    print( ev - od )
```

3

The most common mistake on this problem was treating `ev += n` as `ev += 1` (counting instead of summing), and similarly for `od`. To get the correct answer, we sum the even numbers $2 + 4 + 6 = 12$ and subtract the odd numbers $1 + 3 + 5 = 9$.

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

```
def cnt(el, li):
    """I'll surely lose points for this terrible docstring"""
    c = 0
    for i in range(len(li)):
        if li[i] == el:
            c += 1
    return c

def q2():
    ar_x = [ 0, 1, 0, 2, 2, 3, 0 ]
    ar_y = [ "alpha", "beta", "gamma", "delta" ]
    sum = cnt(2, ar_x) + cnt("beta", ar_y)
    print(sum)
```

3

I basically wanted to make sure you had a basic understanding of passing values to a function, and returning values from a function. I didn't notice any particular pattern in the wrong answers.

(score)

3. [5 points] What does `q3( )` print? (Recall that `//` is integer division.)

```
def i_scale(ar, sf):
    for i in range(len(ar)):
        ar[i] = ar[i] // sf
    return

def q3():
    li = [ 4, 5, 6, 7 ]
    i_scale(li, 2)
    sum = 0
    for n in li:
        sum += n
    print(sum)
```

10

This question is mainly to make sure that you understand how shared references work when pass objects to functions (i.e., that `i_scale` can modify `li`). If you didn't understand that, you may have given 22 as an answer, and received no points. It also asks you to use the integer division operation `//`, and you may have lost a couple points if you got 11 (by using floating-point division) or 12 (by rounding up).

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

```
def rec(li,i):
    if i >= len(li):
        return 0
    else:
        return li[i] + rec(li, i+1)

def q4():
    ar = [ 1, 2, 3, 4, 5 ]
    t = rec(ar,0)
    print(t)
```

15

`rec` is a recursive function for summing the elements of an array. The most common error was just forgetting that the elements of `li` are numbered from 0 to `len(li) - 1`, and so getting 10 by adding up all but the last element. I took off a couple points for that. Two other fairly common errors indicate a misunderstanding of recursion: getting 0 (apparently because you thought the base case in the recursive procedure would return directly to `q4`), or printing each of the returned values from `rec` (when only the final sum is actually returned to `q4`). I did not give partial credit for those answers.

5. [11 points] Complete the function `phone_to_int`, without using the Python built-in function `int( )`. The Python quick reference sheet includes a reminder of how to use a `dict` structure like `DIG_VAL`. It may also be useful to remember that you can build up integers by multiplying and adding, e.g., 10*42 + 7 = 427.

```
DIG_VAL = { "0": 0, "1":1, "2":2, "3":3, "4":4,
            "5":5, "6":6, "7":7, "8":8, "9":9 }

def phone_to_int(ph):
    """
    Convert phone number to integer.
    Args:
       ph: A string representing a phone number. ph may contain
         digits 0-9, spaces, punctuation, and other characters.
    Returns:
       an integer representing just the digits in ph
    Examples:
       phone_to_int("(341) 556-9897") = 3415569897
       phone_to_int("34-45-(442).22") = 344544222
       phone_to_int("000-000-92") = 00000092 = 92
       phone_to_int("there are no digits here") = 0
       phone_to_int("") = 0
    """
    result = 0
    for d in ph:
        if d in DIG_VAL:
            result = result * 10
            result += DIG_VAL[d]
    return result
```

Fewer people succeeded on this question, or really got close, than I expected. I expected it to be less difficult than #6. I thought you would remember how we disassembled integers into their digits in the very first assignment, and would be able to generalize that to building up an integer digit by digit (especially with the hint). Maybe project 1 was too long ago, or maybe in the first week there were too many things going on for you to really think about how we were decomposing integers.

Unfamiliarity with the dict structure may also have played a role. We have used dicts in live-coding, and it's part of project 5, but it hasn't been part of the reading until this week. I generally didn't take off for mistakes in using the dict structure, and several of you created your own list data structures to use in place of `DIG_VAL`; I don't think it was the *main* problem for anyone, but the extra element of confusion may have contributed to difficulty in thinking clearly about how one would build up an integer value from characters.

Partial credit is always complicated on a coding problem. I generally gave 5 points for an answer that filtered out the non-digit characters but returned a string instead of an integer, 5 points for an answer that added digit values but didn't account for place value, 7-9 points for having the place value idea but making mistakes in its implementation. I adjusted points up or down based on other aspects of the code, and reduced by 50% if you printed the result instead of returning it.

(score)

6. [14 points] Finish the function `respace` below, consistent with the docstring.

```
def respace(s):
    """
    Collapse runs of spaces.
    Args:
       s:  a string
    Returns:
       a copy of s, except that every run of 2 or more spaces has been
       replaced by a single space
    Examples (with _ representing a space):
       respace("a__b_c__") = "a_b_c_"
       respace("__a___b") = "_a_b"
       respace("abcde") = "abcde"
       respace("") =  ""
       respace("_____") = "_"
    """
    result = ""
    prev = ""
    for ch in s:
        if prev != " " or ch != " ":
            result += ch
        prev = ch
    return result
```

This is similar to the "dedup" question that has appeared on a prior exam, but instead of compressing all runs down to a single occurrence, it compresses just runs of blanks.

Several students gave essentially the solution to the "dedup" question (not treating blanks specially), for which I awarded 5 points partial credit. Had I anticipated that error, I would have included an example to to catch it. Examples and test suites are never exhaustive, so it is important to think carefully about the specification and not only the test cases.

The most common error was trying to erase spaces by assigning an empty string to a position in the string. Strings are immutable, so you really need to create a copy instead. In most cases I gave 5 points partial credit for a solution that tried to "erase" blanks in a string.

It seems a number of students are still confused about the difference between printing a value and returning a value. I took off 5 points or 50% for that, whichever was less.

Other errors are too many to list here. For some minor confusions about syntax, or for a potential index error, I took off a couple points. For some larger conceptual errors, I took off more points or awarded no credit.

(score)