CIS 210
Winter 2013 Final Exam

Write your name at the top of **each page** before you begin. 1 point for each page.

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

```
def q1():
    x = 5498
    count = 0
    while x > 0:
        count += 1
        x = x // 10
    print(count)
```

4

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

```
def perimeter( llx, lly, urx, ury ):
    """There should be a good docstring here,
    but it's an exam so I left it off.
    """
    width = urx - llx
    if width < 0:
        width = 0 - width
    height = ury - lly
    if height < 0:
        height = 0 - height
    p = (2 * width) + (2 * height)
    return p


def q2():
    pr = perimeter(9, 9, 5, 14)
    print(pr)
```

18

(score)

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

```
def q3():
    lis = [ -5, 7, -23, 3, 98, 10 ]
    clamp(lis, 0, 10)
    s = total(lis)
    print(s)


def clamp(ar, min, max):
    for i in range(len(ar)):
        if ar[i] < min:
            ar[i] = min
        if ar[i] > max:
            ar[i] = max
    return

def total(ar):
    t = 0
    for elem in ar:
        t += elem
    return t
```

30

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

```
class Character:
    def __init__(self, name, power, friendly):
        self.name = name
        self.power = power       #Integer, 1 = human
        self.friendly = friendly #Integer, negative = unfriendly

    def __str__(self):
        return self.name

    def danger(self):
        if self.friendly < 0 :
            return 0 - (self.power * self.friendly)
        else:
            return 1

    def helpful(self):
        return self.power * self.friendly

def q4():
    moro   = Character("Moro, the wolf", 8, 2)
    kiki   = Character("Kiki, the witch", 1, 5)
    totoro = Character("Totoro, the forest spirit", 8, 5)
    chihiro = Character("Chihiro, the child", 1, 1)
    yubaba = Character("Yubaba, the witch", 8, -5)

    characters = [ chihiro, kiki, moro, totoro, yubaba ]
    helper = characters[0]
    for ch in characters:
        if ch.danger() > 1 :
            print("Watch out for ", ch.name)
        if ch.helpful() > helper.helpful():
            helper = ch
    print("Get help from ", helper)
```

Watch out for Yubaba, the witch.
Get help from Totoro, the forest spirit

5. [12 points] This question uses the Character class from the previous question. Complete the function, consistent with its docstring.

```
def count_helpful( lis ):
    """Number of Characters in lis that are helpful.
    Args:
        lis:  a list of Character objects (see question 4).
    Returns:
        An integer count of the number of characters in lis
        with helpfulness > 0.
    Example:
        For the list characters = [ chihiro, kiki, moro, totoro, yubaba ]
        from question 4, count_helpful(characters) == 4  (everyone in the
        list except Yubaba is helpful).
    """
    # Your code here
    count = 0
    for ch in lis:
        if ch.helpful() > 0:
            count += 1
    return count
```

6. [13 points] Recall the "split" function that divides a single string into a list of strings, breaking it around blanks or another character. In this problem you will write a similar function, but instead of splitting up a string of text, it splits a list of numbers. For example, `nsplit([5, 7, 7, 4, 3, 7], 7)` would return `[[ 5 ], [4, 3]]`. Complete the function, consistent with its docstring.

```
def nsplit( lis,  border ):
    """Split a list of integers into sub-lists, splitting at border.
    Args:
        lis:  A list of integers
        border:  Break the list at each occurrence of one of more instances
            of this value.
    Returns:
        A list of non-empty sub-lists of lis,
        containing maximal sub-sequences of lis excluding the border value.
    Examples:
         nsplit( [5, 4, 3, 0, 2, 1, 0, 5], 0 ) = [[5, 4, 3], [2, 1], [5]]
         nsplit( [7, 7, 0, 0, 0, 5, 0], 0 ) = [[7, 7], [5]]
         nsplit( [ 0, 0, 0 ], 0) = [ ]
    """
    result = [ ]
    current_run = [ ]
    for n in lis:
         if n == border:
            if len(current_run) > 0:
                result.append(current_run)
                current_run = [ ]
         else:
            current_run.append(n)
    if len(current_run) > 0:
        result.append(current_run)
    return result
```


(score)

7. [15 points] In the game Scrabble, the score for a word is based on the sum of the values of letters in that word. Function `highest_scrabble_score` takes a list of words, and chooses the word from the list that is worth the most points. The list `SCRABBLE_POINTS` describes the point values of all the English letters; each sub-list is a value followed by a string containing the letters with that value.

Complete `highest_scrabble_score`. You may optionally write additional functions to break your code down into simpler pieces.

```
SCRABBLE_POINTS = [[ 1, "eaionrtlsu"], [2, "dg"], [3, "bcmp"],
                   [ 4, "fhvwy"], [5, "k"], [8, "jx"], [10, "qz"]]

def highest_scrabble_score( lis ):
    """Select the word from lis with the highest Scrabble score,
    based on SCRABBLE_POINTS. For example, highest_scrabble_score(["totoro", "kiki" ])
    returns "kiki" because "totoro" scores 6 (1+1+1+1+1+1) and "kiki" scores 12 (5+1+5+1).
    Args:
       lis:  A list of strings containing only lower case letters.
    Returns:
       The string from lis with the largest sum of letter values
       from SCRABBLE_POINTS.  (Ties may be broken arbitrarily.)
    """
    best = ""
    best_score = 0
    for word in lis:
        score = 0
        for letter in word:
            score += value(letter)
        if score > best_score:
            best = word
            best_score = score
    return best

def value(letter):
    for pair in SCRABBLE_POINTS:
        if letter in pair[1]:
            return pair[0]
    print("CANT HAPPEN")
    return -100
```

(Additional room for Question 7, if needed.)

A more efficient way to find the value of each letter is to first unpack the *SCRABBLE_POINTS* list into a table:

```
def unpack_table( letter_scores ):
    """Unpack letter scores into a dictionary.
    Args:
        letter_scores:  List of two-element lists.  Each item in the list
             has two elements.  The first is a point score, and the second is
             a string of letters with that point score.
    Returns:
        A dictionary associating a score with each letter from letter_scores.
    """
    table = { }  # A dictionary
    for pair in letter_scores:
        value = pair[0]
        letters = pair[1]
        for letter in letters:
            table[letter] = value
    return table
```

We would call *unpack_table* just once before any calls to *highest_scrabble_score*. With a table in this form, the *value* function can be rewritten to make a single access to the table, instead of a loop through the entries — approximately 26 times faster, which is "only a constant factor" but enough to make a difference if we were, for example, using this as part of a program to try a large number of possibilities and suggest good Scrabble plays.