CIS 210
Fall 2012 Final Exam

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

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

```
def q1():
    x = 10
    y = 7
    x = x - y
    if x > y:
        x = 42
    if y > x:
        y = 42
    print(x, y)
```

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

```
def rescale(x, low, high):
    if x < low:
        return 0
    if x > high:
        return 100
    x = (100 * (x - low)) // (high - low)
    return x

def q2():
    minscore = 10
    maxscore = 30
    x = rescale(35, minscore, maxscore)
    y = rescale(20, minscore, maxscore)
    print(x, y)
```


(score)

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

```
class Account:

    def __init__(self, account_name):
        self.name = account_name
        self.value = 0

    def deposit(self, amt):
        self.value += amt

    def withdraw(self, amt):
        self.value = self.value - amt

def q3():
    travel = Account("Travel")
    travel.deposit(100)
    books = Account("Books")
    books.deposit(100)
    travel.withdraw(30)
    books.withdraw(50)
    print("Travel funds available:", travel.value)
    print("Book funds available:", books.value)
```

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

```
def swap(ar, i, j):
    t = ar[i]
    ar[i] = ar[j]
    ar[j] = t

def q4():
    a_lis = [ 10, 20, 30 ]
    b_lis = a_lis
    c_lis = [ 1, 2, 3 ]
    swap(a_lis, 0, 2)
    swap(b_lis, 1, 2)
    swap(c_lis, 0, 1)
    print(a_lis[0] + c_lis[0])
    print(a_lis[1] + c_lis[1])
    print(a_lis[2] + c_lis[2])
```

5. [10 points] Write the body of function span, consistent with its docstring, and without using Python's standard min, max, or sort functions.

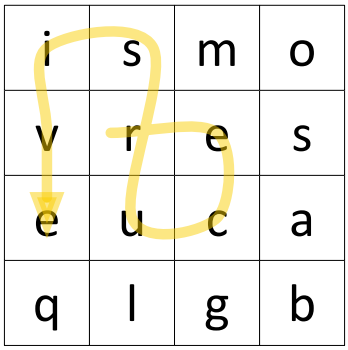
```
def span(ar):
    """Find the difference between the maximum and minimum elements of
    a list of integers.
    Arguments:
        ar: a list of integers (not modified)
    Returns:
        The difference between the largest and smallest elements of ar.
        For lists with 0 elements, span(ar) is defined to be 0.
    Examples:
        span([ 0, 1, 2, 3, 4, 5 ]) = 5
        span([ -2, -6, 4, 10 ]) = 16
    """
```

6. [15 points] Write the body of function dedup, consistent with its docstring. It may help to remember that you can build up strings by appending each character to the end using "+", e.g., `"cod" + "e" == "code"`.

```
def dedup(w):
    """Return a copy of w with runs of identical characters collapsed.
       Arguments:
           w:  A string of letters
       Returns:
           A string identical to w, except that wherever two or more
           identical characters appeared consecutively in w, only
           one copy appears in the returned string.
        Examples:
           dedup("abbcdefffgfgfgxxx") => "abcdefgfgfgx"
           dedup("abcdef") => "abcdef"
           dedup("xxxxxxxxxx") => "x"
           dedup("") => ""
    """
```


(score)

7. [15 points] This problem (continued on the next page) may remind you of our Boggler project, but it has been considerably simplified. We search for a single word in a board with four rows and four columns. A word must be spelled out by moving left, right, up, or down. There are no diagonal moves, and it is ok to use a letter more than once, as 'r' is used twice to find"recursive" in this board:



Most of the `WordBoard` class is provided, but you must fill in the recursive depth-first search for matching a word. Note the method `match_letter` will work and return False if you check for a letter matching a position that is not on the board (e.g., if you look for the letter '*a*' in row -3, column 5), so you should not make redundant checks in your search method. Think of the variable `pos` as a finger keeping track of which letter of the word to match next.

```
class WordBoard():
    """A word puzzle board, 4 rows (0..3) by 4 columns (0..3)"""

    def __init__( self, text ):
        """Create a 4x4 puzzle board from 16 characters of text."""
        self.board = [ text[0:4], text[4:8], text[8:12], text[12:16] ]

    def match_letter(self, letter, row, col):
        """Check for letter match at row, col.
           Returns True iff row, col is on the board
                   and the character at board[row][col] matches letter.
        """
        if row < 0 or row > 3 or col < 0 or col > 3:
            return False
        return self.board[row][col] == letter
```

(Continued on next page. You can tear this page out if you like.)

(*Class* `WordBoard` *continued from previous page. Complete the method* `find_suffix_at`.)

```
    def find(self, word):
        """Can word be spelled out on the board, starting
        anywhere and moving only up, down, left, right?
        Arguments:
            word: The word we're searching for
        Returns:
            True iff word can be spelled out by a series of
            only left, right, up, and down movements on the board.
        """
        for row in range(4):
           for col in range(4):
               if self.find_suffix_at(row, col, word, 0):
                   return True
        return False

    def find_suffix_at(self, row, col, word, pos):
        """Depth-first search for a word, continuing from
        board position (row, col) and index pos in word.
        If pos >= len(word), then it has been fully matched.
        Note self.match_letter(letter,row,col) returns False
        if row,col is not on the board.
        Arguments:
            row, col:  Looking for it starting in this position.
            word: The word we're looking for
            pos:  Looking for the suffix that starts at word[pos].
        Returns:
            True if the letters from word[pos] to word[len(word)-1]
            can be found starting at board[row,col].
        """
```


(score)