CIS 210
Winter 2015 Final Exam KEY

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

```
def trans(li, delta):
    for i in range(len(li)):
        li[i] = li[i] + delta
    return

def q1( ):
    a = [ 1, 2, 3 ]
    b = [ 1, 2, 3 ]
    c = b
    trans(a,1)
    trans(b,1)
    trans(c,1)
    sum = 0
    for i in range(len(a)):
        sum = sum + a[i] + b[i]
    print(sum)
```

21

The key to getting this problem right is understanding that c and b are references to the same object, which therefore gets passed to `trans` twice. If you didn't understand that, you probably got 18.

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

```
def q2():
    count = 0
    for i in range(10):
        for j in range(10):
            count += 1
    print(count)
```

100

The number of iterations of nested loops is the product of the inner and the outer. I was surprised that several people had trouble with this one.

(score)

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

```
def compress(li):
    """You'll have to figure it out"""
    result = [ ]
    if len(li) == 0:
        return result
    prev = li[0]
    result.append(li[0])
    for item in li:
        if item != prev:
            result.append(item)
            prev = item
    return result

def q3():
    ar = [ 1, 1, 2, 3, 3, 3, 3, 4, 5, 5, 5, 5 ]
    ar = compress(ar)
    sum = 0
    for item in ar:
        sum += item
    print(sum)
```

15
'Compress' is like the 'dedup' function from a prior exam — it makes a copy in which adjacent duplicate elements have been dropped. The answer is therefore $1+2+3+4+5 = 15$.

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

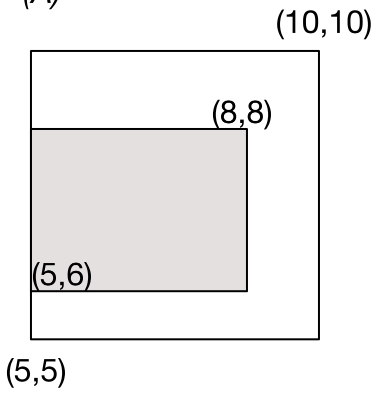
```
def pointwise(li, f):
    """Yes, yes we can"""
    result = [ ]
    for item in li:
        result.append( f(item) )
    return result

def plus_two(x):
    return x + 2

def q4():
    before = [1, 2, 3]
    after = pointwise(before, plus_two)
    print(after)
```

[3, 4, 5]
We've pass functions to other functions (or methods) both in lecture and in the Sudoku project; this was a check to see if you understood that. Most of you got it.

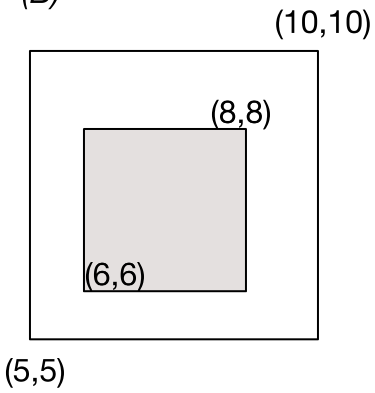
(score)

On the next page you will be asked to complete a a `covers` method that returns True iff a rectangle represented by a Rect object completely covers another Rect object. For example, consider cases (A), (B), and (C) below:

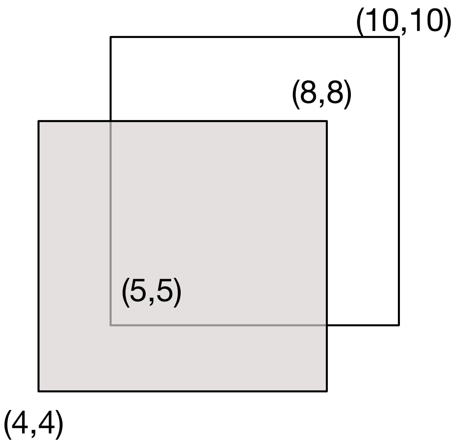
(A)

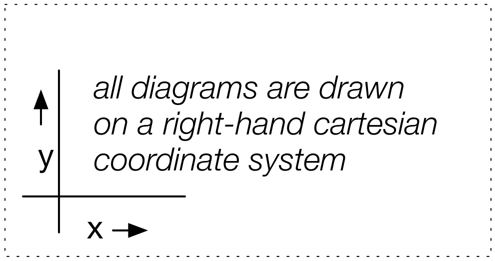


(B)



(C)





In situation *(A)* the white rectangle does completely cover the grey rectangle, and also in situation *(B)*. However, in situation *(C)* parts of the grey rectangle are outside the white rectangle, so `Rect(Point(5,5),Point(10,10)).covers(Rect(Point(4,4),Point(8,8)))` should return False.

5. [15 points] Complete the `covers` method. The previous page explains what it means for a rectangle to cover another rectangle.

```
class Point:
    """Two public fields representing x and y coordinates"""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rect:
    """Defined by lower left and upper right points"""
    def __init__(self, ll, ur):
        assert ll.x < ur.x and ll.y < ur.y
        self.ll = ll
        self.ur = ur

    def covers(self, other):
        """
        Does this Rect entirely cover other?
        Args:
           other: Rect
        Returns:
           True iff this Rect entirely contains other.
        Examples:
           Rect(Point(5,5),Point(10,10)).covers(Rect(Point(5,6),Point(8,8)))  = True
           Rect(Point(5,5),Point(10,10)).covers(Rect(Point(4,4),Point(8,8)))  = False
           Rect(Point(2,2),Point(4,4)).covers(Rect(Point(2,2),Point(4,4))) = True
        """
        return (self.ll.x <= other.ll.x and
                self.ll.y <= other.ll.y and
                self.ur.x >= other.ur.y and
                self.ur.y >= other.ur.y)
```

Another perfectly valid way to do this is with a series of tests, e.g.,

```
        if self.ll.x > other.ll.x:
           return False
        if self.ll.y > other.ll.y:
           return False
        if self.ur.x < other.ur.y:
           return False
        if self.ur.y < other.ur.y:
           return False
        return True
```

A common error was to write this logic with nested 'if' statements, and leave a path on which no result was returned.

(score)

6. [15 points]

Recall that we have used lists of lists to represent grids or matrices. For example, we can write $[[8, 3, 4, 5], [1, 2, 7, 9], [12, 5, 3, 7]]$ to represent this matrix:

8	3	4	5
1	2	7	9
12	5	3	7

This problem asks you to find the *minimum* sum of the columns of such a matrix (e.g., 10 for this example). Column 0 of the matrix above has a sum of 21 $(8 + 1 + 12)$, column 1 has a sum of 10 $(3 + 2 + 5)$ and column 2 has a sum of 14 $(4 + 7 + 3)$, column 4 has sum 21 $(5 + 9 + 7)$, and the minimum of these is 10.

Finish function `min_col` on the next page. I believe it will be easiest if you also write a function to sum the values in one column; you can use this page to do that. (A docstring is not required.)

(score)

```
def min_col(ar):
    """
    Find the minimum sum of items in a column of a matrix (list of lists).
    Args:
       ar:  A list of lists of integers representing a rectangular matrix.
            ar has at least one row, and at least one column, and each
            row has the same number of columns.
    Returns:
       The smallest sum of values in a column of ar.
    Examples:
       min_col( [[9, 2, 3], [4, 5, 8], [9, 6, 9]]) = 13  (2+5+6)
       min_col( [[1, 2], [9, 2]] ) = 4 (2+2)
       min_col( [[ 8, 7 ]]) = 7
    """
    min = col_sum(ar, 0)
    for col in range(len(ar[0])):
        sum = col_sum(ar, col)
        if sum < min:
            min = sum
    return min

def col_sum(ar, col):
    sum = 0
    for row in ar:
        sum += row[col]
    return sum
```

There are several ways to solve this one — some of you transposed the matrix and then summed the rows. However, many of you had trouble accessing elements in a list of lists, even though we have had several projects that used this representation of matrices or grids. I was surprised by how much trouble this gave even some of you who are quite good at designing algorithms for lists.

I explicitly allowed you to omit the docstring if you wrote a helper function. In retrospect, that was a mistake. Many of you seem to have gotten mixed up and written functions that did not do what your `min_col` functioned called them to do. Perhaps if you had written a docstring with a very clear definition of what your function did, it would have helped you think more clearly about writing and using your function.

(score)

7. [15 points] An s-expression is a fully parenthesized expression in prefix notation. In Python, we can easily express s-expressions using tuples, e.g., expressing $(5 + 3) - 4$ as `(’-’,(’+’,5,3),4)`. This problem asks you to complete an evaluator for s-expressions representing only addition and subtraction of integers. You can use the built-in `isinstance` function to distinguish leaves (integers) from inner nodes (tuples that represent operations) by checking `isinstance(exp,tuple)` or `isinstance(exp,int)`.

```
def s_eval( exp ):
    """
    Evaluate an ’s-expression’.
    Args:
       exp is either an integer or a tuple (op,left,right)
             where op is one of ’+’ or ’-’
             left is an s-expression
             right is an s-expression
    Returns:
       The result of evaluating exp, where
          an int evaluates to itself  (e.g., s_eval of 5 is 5)
          a tuple (op,left,right) evaluates to the result of applying
             the corresponding operation to the values of the left and right operands.
    Examples:
       s_eval( 5 ) = 5
       s_eval( (’+’, 5, 4) ) = 9
       s_eval( (’-’, (’+’, 5, 4), (’-’, 5, 3)) ) = 7
    """
    if isinstance(exp, int):
        return exp
    else:
        op, left, right = exp
        if op == ’+’:
            return s_eval(left) + s_eval(right)
        elif op == ’-’:
            return s_eval(left) - s_eval(right)
```

Experience with exam questions involving recursion told me that you would probably find this problem difficult, so I made it the last question. As expected, a lot of you aren’t quite comfortable yet designing recursive functions. More than half of you didn’t get any points on this problem, and only a handful of you got full credit. Since we haven’t used ’isinstance’ in a project, I did not take off points for anything that had the basic idea of deciding whether exp was a tuple or an integer (including some pretty crazy code); but I did take off if you didn’t get the basic idea of recursively evaluating sub-expressions.