CSc 110 Sample Final Exam #1

1. Collections Mystery

Consider the following function:

```
def mystery(list1, list2):
    result = {}
    for i in range(0, len(list1)):
        result[list1[i]] = list2[i]
        result[list2[i]] = list1[i]
    return result
```

The three entries below have specific values for the first and second parameters to function `mystery` (when you see text like the letter b, that indicates that the list stores the string "b" in that position). For each entry, indicate what values would be stored in the dictionary returned by function `mystery` if the given lists are passed as parameters. Dictionary elements should be listed with "key:value" elements, as in {b:z, d:e}.

list1: `[b, l, u, e]` list2: `[s, p, o, t]`

dictionary returned:_____

list1: `[k, e, e, p]` list2: `[s, a, f, e]`

dictionary returned:_____

list1: `[s, o, b, e, r]` list2: `[b, o, o, k, s]`

dictionary returned:_____

2. 2D List Mystery

Consider the following method:

```
def mystery(data, pos, n):
    result = set()
    for i in range(0, n):
        for j in range(0, n):
            result.add(data[i + pos][j + pos])
    return result
```

Suppose that a variable called grid has been declared as follows:

```
grid = [[8, 2, 7, 8, 2, 1], [1, 5, 1, 7, 4, 7],
        [5, 9, 6, 7, 3, 2], [7, 8, 7, 7, 7, 9],
        [4, 2, 6, 9, 2, 3], [2, 2, 8, 1, 1, 3]]
```

which means it will store the following 6-by-6 grid of values:

```
8   2   7   8   2   1
1   5   1   7   4   7
5   9   6   7   3   2
7   8   7   7   7   9
4   2   6   9   2   3
2   2   8   1   1   3
```

For each call below, indicate what value is returned. If the function call results in an error, write "error" instead.

Function Call	Contents of Set Returned
`mystery(grid, 2, 2)`	_____
`mystery(grid, 0, 2)`	_____
`mystery(grid, 3, 3)`	_____

3. **Searching and Sorting**.

(a) Suppose we are performing a **binary search** on a sorted list called `numbers` initialized as follows:

```
#  index    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
numbers = [-1,  0,  5, 11, 18, 29, 53, 57, 61, 64, 78, 82, 85, 89, 93]

# search for the value 86
index = binary_search(numbers, 86)
```

Write the indexes of the elements that would be examined by the binary search (the `mid` values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

- Indexes examined: _____
- Value Returned: _____

(b) Write the state of the elements of the list below after each of the first 3 passes of the outermost loop of the selection sort algorithm.

```
numbers = [8, 5, -9, 14, 0, -1, -7, 3]
selection_sort(numbers)
```

(c) Trace the complete execution of the merge sort algorithm when called on the list below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-lists that are created by the algorithm and show the merging of sub-lists into larger sorted lists.

```
numbers = [8, 5, -9, 14, 0, -1, -7, 3]
merge_sort(numbers)
```

4. String Programming

Write a function called `same_pattern` that returns true or false depending upon whether two strings have the same pattern of characters. More precisely, two strings have the same pattern if they are of the same length and if two characters in the first string are equal if and only if the characters in the corresponding positions in the second string are also equal. Below are some examples of patterns that are the same and patterns that differ (keep in mind that the method should return the same value no matter what order the two strings are passed).

1st String	2nd String	Same Pattern?
`""`	`""`	`True`
`"a"`	`"x"`	`True`
`"a"`	`"ab"`	`False`
`"ab"`	`"ab"`	`True`
`"aa"`	`"xy"`	`False`
`"aba"`	`"+-+"`	`True`
`"---"`	`"aba"`	`False`
`"abcabc"`	`"zodzod"`	`True`
`"abcabd"`	`"zodzoe"`	`True`
`"abcabc"`	`"xxxxxx"`	`False`
`"aaasssccccn"`	`"aaabbbcccd"`	`True`
`"asasasasas"`	`"xyxyxyxyxy"`	`True`
`"ascneencsa"`	`"aeiouuoiea"`	`True`
`"aaasssccccn"`	`"aaabbbcccd"`	`True`
`"asasasasas"`	`"xxxxxyyyyy"`	`False`
`"ascneencsa"`	`"aeiouaeiou"`	`False`
`"aaasssccccn"`	`"xxxyyyzzzz"`	`False`
`"aaasssiiii"`	`"gggdddfffh"`	`False`

Your function should take two parameters: the two strings to compare. You are allowed to create new strings, but otherwise you are not allowed to construct extra data structures to solve this problem (no list, set, dictionary, etc). You are limited to the string functions on the cheat sheet.

5. 2-d Lists

Write a function called `find_max` that takes a two dimensional list as a parameter and returns the number of the row that sums to the greatest value. For example if you had the following list of lists:

```
list = [[1, 2, 3], [2, 3, 3], [1, 3, 3]]
```

The first row would be 6, the second 8 and the third 7. The function would therefore return 1.

You can assume the passed in list of lists has at least one row and one column. You cannot assume that it is square.

6. Collections Programming

Write a function called `remove_duplicates` that takes a list of tuples and that removes duplicates from the list, returning a `set` of the tuples from the list. For example, if a list called `points` contains the following values:

```
[(1,3), (7,7), (1,3), (7,8), (1,3), (7,7), (7,8), (4, 2), (7, 8), (7, 8), (4, 2), (4, 2)]
```

then the call `remove_duplicates(points)` should leave the list storing:

```
[(1, 3), (7, 7), (7, 8), (4, 2)]
```

and the set returned should store the same values. Your function should preserve the order of values in the list (removing duplicates that come later in the list), but the values in the set can appear in any order.

You may construct the set to be returned, but you are not allowed to construct other structured objects (no string, set, list, etc.).

7. Collections Programming

Write a function called `count_in_area_code` that accepts two parameters, a `dictionary` from names (strings) to phone numbers (strings) and an area code (as a string), and returns how many unique phone numbers in the map use that area code. For example, if a map `m` contains these pairs:

```
{Marty=206-685-2181, Rick=520-206-6126, Beekto=206-685-2181,
 Jenny=253-867-5309, Stuart=206-685-9138, DirecTV=800-494-4388,
 Bob=206-685-9138, Benson=206-616-1246, Hottline=900-520-2767}
```

The call of `count_in_area_code(m, "206")` should return 3, because there are 3 unique phone numbers that use the 206 area code: Marty/Beekto's number of `"206-685-2181"`, Stuart/Bob's number of `"206-685-9138"`, and Benson's number of `"206-616-1246"`.

You may assume that every phone number value string in the dictionary will begin with a 3-digit numeric area code, and that the area code string passed will be a numeric string exactly 3 characters in length. If the dictionary is empty or contains no phone numbers with the given area code, your function should return 0.

You may create one collection (list, dictionary, set) of your choice as auxiliary storage to solve this problem. You can have as many simple variables as you like. You should not modify the contents of the dictionary passed to your function.

8. List Programming

Write a function called `rearrange` that takes a list of integer values as a parameter and that moves to the end of the list all values originally appearing in odd index positions in the list, preserving the relative order of these values. For example, suppose that a variable called `list` stores the following sequence of values:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

and we make the following call:

```
rearrange(list)
```

Afterwards the list should store the following sequence of values:

```
[0, 2, 4, 6, 8, 1, 3, 5, 7]
```

Notice that the values originally stored in odd indexes have been moved to the end of the list (1, 3, 5, and 7). This example involved sequential integers to make it easier to see exactly which values are moved, but this won't always be the case. For example, if the list had instead stored:

```
[14, 8, 6, 3, 1, 4, 2, 19, 42, 9]
     ^     ^     ^     ^       ^
     |     |     |     |       |
     +-----+-----+-----+-------+
         values to be moved
```

then it would store the following values after the function executes:

```
[14, 6, 1, 2, 42, 8, 3, 4, 19, 9]
```

Notice that what matters is whether values are in odd index positions, not whether they are odd numbers (the values 8, 3, 4, 19, and 9 in the example).

You may not construct any extra data structures to solve this problem. You must solve it by manipulating the list you are passed as a parameter.

9. Programming

Write a function called `interleave` that takes two lists as parameters and returns a new list that contains the result of interleaving the elements of the two lists. It should not alter either of its parameters. Two lists are interleaved by taking elements in the following order:

1st element of 1st list
1st element of 2nd list
2nd element of 1st list
2nd element of 2nd list
3rd element of 1st list
3rd element of 2nd list
and so on

The following table shows the results of calling interleave given the lists defined below:

```
list1 = [1, 8, 3, 9]
list2 = [2, 12, 6, 14]
list3 = [82, 7, 4, 2, 10, 20, 30, 40]
```

call	**returned list**
`interleave(list1, list2)`	`[1, 2, 8, 12, 3, 6, 9, 14]`
`interleave(list2, list1)`	`[2, 1, 12, 8, 6, 3, 14, 9]`
`interleave(list1, list3)`	`[1, 82, 8, 7, 3, 4, 9, 2, 10, 20, 30, 40]`

If the lists are not the same length, the elements of the longer list get appended after the interleaved elements as shown in last example above. If either list is empty, the result should contain the values from the other list.

Solutions

1. Collections Mystery

```
{'o': 'u', 't': 'e', 'p': 'l', 'b': 's', 'l': 'p', 's': 'b', 'e': 't', 'u': 'o'}
{'a': 'e', 'p': 'e', 's': 'k', 'e': 'p', 'k': 's', 'f': 'e'}
{'o': 'b', 'r': 's', 'b': 'o', 's': 'r', 'e': 'k', 'k': 'e'}
```

2. 2D List Mystery

Function Call	Contents of Set Returned
mystery(grid, 2, 2)	{6, 7}
mystery(grid, 0, 2)	{1, 2, 5, 8}
mystery(grid, 3, 3)	{1, 2, 3, 7, 9}

3. Searching and Sorting

```
(a) Indexes examined: 7, 11, 13, 12 Value returned: -14
(b) {-9, 5, 8, 14, 0, -1, -7, 3}
    {-9, -7, 8, 14, 0, -1, 5, 3}
    {-9, -7, -1, 14, 0, 8, 5, 3}
(c) { 8, 5, -9, 14, 0, -1, -7, 3}
    { 8, 5, -9, 14} { 0, -1, -7, 3} split
    { 8, 5} {-9, 14} { 0, -1} {-7, 3} split
    { 8} { 5} {-9} {14} { 0} {-1} {-7} { 3} split
    { 5, 8} {-9, 14} {-1, 0} {-7, 3} merge
    {-9, 5, 8, 14} {-7, -1, 0, 3} merge
    {-9, -7, -1, 0, 3, 5, 8, 14} merge
```

4. String Programming

```
    def same_pattern(s1, s2):
        if (len(s1) != len(s2)):
            return False
        for i in range(0, len(s1)):
            for j in range(i + 1, len(s1)):
                if (s1[i] == s1[j] and s2[i] != s2[j]):
                    return False
                if (s2[i] == s2[j] and s1[i] != s1[j]):
                    return False
        return True
```

5. 2d List Programming

```
def find_max(lis):
    max_sum = 0
    max_row = 0
    for i in range(0, len(lis)):
        cur_sum = 0
        cur_row = i
        for j in range(0, len(lis[i])):
            cur_sum += lis[i][j]
        if cur_sum > max_sum:
            max_sum = cur_sum
            max_row = cur_row
    return max_row
```

6. Collections Programming

```
  def remove_duplicates(lis):
      result = set(lis)
      i  = 0
      while i < len(lis):
          tup = lis[i]
          if tup in result:
              lis.pop(i)
          else:
              i += 1
      return result
```

7. Collections Programming

```
   def count_in_area_code(numbers, area_code):
       unique_numbers = set()
       for name, phone in numbers.items():
           if (phone[0:3] == area_code):
               unique_numbers.add(phone)
       return len(unique_numbers)
```

8. List Programming

```
def rearrange(lis):
    for i in range(1, len(lis) // 2 + 1):
        n = lis[i]
        lis.pop(i)
        lis.append(n)
```

9. Programming

```
   def interleave(lis1, lis2):
       result = []
       short = min(len(lis1), len(lis2))
       for i in range(0, short):
           result.append(lis1[i])
           result.append(lis2[i])

       for i in range(short, len(lis1)):
           result.append(lis1[i])

       for i in range(short, len(lis2)):
           result.append(lis2[i])

       return result
```