CSc 110 Sample Final Exam #1

1. While Loop Simulation

For each call of the function below, write the output that is printed:

```
def mystery(i, j):
    while (i != 0 and j != 0):
        i = i // j
        j = (j - 1) // 2
        print(str(i) + " " + str(j) + " ", end='')
    print(i)
```

Function Call	Output
`mystery(5, 0)`	_____
`mystery(3, 2)`	_____
`mystery(16, 5)`	_____
`mystery(80, 9)`	_____
`mystery(1600, 40)`	_____

2. Inheritance Mystery

Assume that the following classes have been defined:

```
class A(B):
    def method2(self):
        print("a 2  ", end='')
        self.method1()

class B(C):
    def __str__(se;f):
        return "b"

    def method2(self):
        print("b 2  ", end='')
        super(B, seld).method2()
```

```
class C:
    def __str__(self):
        return "c"

    def method1(self):
        print("c 1  ", end='')

    def method2(self):
        print("c 2  ", end='')

class D(B):
    def method1(self):
        print("d 1  ", end='')
        self.method2()
```

Given the classes above, what output is produced by the following code?

```
elements = [A(), B(), C(), D()]
for i in range(0, len(elements)):
    print(elements[i])
    elements[i].method1()
    print()
    elements[i].method2()
    print()
    print()
```

3. Collections 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)`	_____

4. List Programming

Write a function named `is_unique` that takes a list of integers as a parameter and that returns a boolean value indicating whether or not the values in the list are unique (`True` for yes, `False` for no). The values in the list are considered unique if there is no pair of values that are equal. For example, if a variable called `list` stores the following values:

```
list = [3, 8, 12, 2, 9, 17, 43, -8, 46, 203, 14, 97, 10, 4]
```

Then the call of `is_unique(list)` should return `True` because there are no duplicated values in this list.

If instead the list stored these values:

```
list = [4, 7, 2, 3, 9, 12, -47, -19, 308, 3, 74]
```

Then the call should return `False` because the value 3 appears twice in this list. Notice that given this definition, a list of 0 or 1 elements would be considered unique.

5. Dictionary/Set 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.

6. 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.

7. 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.

8. Critters

Write a class `Ostrich` that extends the `Critter` class from the Critters assignment, including its `get_move` and `get_color` methods. An `Ostrich` object first stays in the same place for 10 moves, then moves 10 steps to either the WEST or the EAST, then repeats. In other words, after sitting still for 10 moves, the ostrich randomly picks to go west or east, then walks 10 steps in that same direction. Then it stops and sits still for 10 moves and repeats. Whenever an `Ostrich` is moving (that is, whenever its last call to `get_move` returned a direction other than `DIRECTION_CENTER`), its color should be white (`"white"`). As soon as it stops moving, and initially when it first appears in the critter world, its color should be cyan (`"cyan"`). When randomly choosing west vs. east, the two directions should be equally likely.

You may add anything needed (fields, other methods) to implement the above behavior appropriately. All other critter behavior not discussed here uses the default values.

9. Classes and Objects

Suppose that you are provided with a pre-written class `Date` as described at right. (The headings are shown, but not the method bodies, to save space.) Assume that the fields, constructor, and methods shown are already implemented. You may refer to them or use them in solving this problem if necessary.

```
# Each Date object stores a single
# month/day such as September 19.
# This class ignores leap years.

class Date:
    # Constructs a date with
    # the given month and day.
    def __init__(self, m, d):
        self.__month = m
        self.__day = d

    # Returns the date's day.
    def get_day(self)

    # Returns the date's month.
    def get_month(self)

    # Returns the number of days
    # in this date's month.
    def days_in_month(self)

    # Modifies this date's state
    # so that it has moved forward
    # in time by 1 day, wrapping
    # around into the next month
    # or year if necessary.
    # example:  9/19 -> 9/20
    # example:  9/30 -> 10/1
    # example: 12/31 -> 1/1
    def next_day()


    # your method would go here
```

Write an instance method named **`compare`** that will be placed inside the `Date` class to become a part of each `Date` object's behavior. The `compare` method accepts another `Date` as a parameter and compares the two dates to see which comes first in chronological order. It returns an integer with one of the following values:

- a negative integer (such as -1) if the date represented by this `Date` comes before that of the parameter
- 0 if the two `Date` objects represent the same month and day
- a positive integer (such as 1) if the date represented by this `Date` comes after that of the parameter

For example, if these `Date` objects are declared in client code:

```
sep19 = Date(9, 19)
dec15 = Date(12, 15)
temp =  Date(9, 19)
sep11 = Date(9, 11)
```

The following `boolean` expressions should have `True` results.

```
sep19.compare(sep11) > 0
sep11.compare(sep19) < 0
temp.compare(sep19) == 0
dec15.compare(sep11) > 0
```

Your method should not modify the state of either `Date` object (such as by changing their `day` or `month` field values).

Solutions

1. While Loop Simulation

Function Call	Output
`mystery(5, 0)`	`5`
`mystery(3, 2)`	`1 0 1`
`mystery(16, 5)`	`3 2 1 0 1`
`mystery(80, 9)`	`8 4 2 1 2 0 2`
`mystery(1600, 40)`	`40 19 2 9 0 4 0`

2. Inheritance Mystery

```
b
c 1
a 2  c 1

b
c 1
b 2  c 2

c
c 1
c 2

b
d 1  b 2  c 2
b 2  c 2
```

3. Collections 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]`

4. List Programming

```
def is_unique(list):
    for i in range(1, len(list)):
        for j in range(i, len(list)):
            if (list[i - 1] == list[j]):
                return False
    return True
```

5. 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)
```

6. 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
```

7. 2d Lists

```
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
```

8. Critters

```
   class Ostrich(Critter):
       def __init__(self):
           super(Ostrich, self).__init__()
           self.__hiding = True
           self.__steps = 0
           self.__west = randint(0, 1) == 0

       def get_color(self):
           if (self.__hiding):
               return "cyan"
           else:
               return "white"

       def get_move(self):
           if (self.__steps == 10):
               self.__steps = 0   # Pick a new direction and re-set the steps counter
               self.__hiding = not self.__hiding
               self.__west = randint(0, 1) == 0

           self.__steps += 1
           if (self.__hiding):
               return DIRECTION_CENTER
           elif (self.__west):
               return DIRECTION_WEST
           else:
               return DIRECTION_EAST
```

9. Classes

```
   def compare(other):
       if (self.__month < other.__month or (self.__month == other.__month and
            self.__day < other.__day)):
           return -1
       elif (self.__month == other.__month and self.__day == other.__day):
           return 0
       else:
           return 1
```