CSE101 – Introduction to Computers
Python Programming Assignment # 4
(25 points, Submission due date: Friday 10 May 2019)

Instructions

For each of the following problems, create an error free efficient Python program. Each program should be submitted in a separate Python file respectively that follows a particular naming convention. (E.g. The Python program for Question 1 should be in .py file with name Assign4Answer1.py. The Python program for question 2 should be in .py file with name Assign4Answer2.py. Include one or two input cases in your program. The program should execute properly in PyCharm).

Problems

Problem 1: Gray codes (6 points)

Write a function that returns a list of Gray codes of a specified length. A Gray code is a special type of binary code organized so that any two successive bit patterns differ by only one bit. For example, a 2-bit Gray code is [’00’, ’01’, ’11’, ’10’]. Notice how in the transition from the second code to the third code only the first bit in the code changes. Here is a recursive algorithm for generating an n-bit Gray code:

- If n = 1 return the list with two strings [’0’,’1’]
- Otherwise let a be the list of Gray codes with n - 1 bits; the output should be a list of codes with a 0 in front of every code in a followed by codes that have a 1 in front of every code in a but in reverse order.

Here is an example that shows what your function should produce for n = 3:

```
>>> graycode(3)
[’000’, ’001’, ’011’, ’010’, ’110’, ’111’, ’101’, ’100’]
```

Notice how the first four codes have 0’s in front of the four 2-bit codes and the last four codes have a 1 in front of the 2-bit codes in the opposite order. Some hints:

The + operator can be used to concatenate two strings and to append two lists.

A builtin function named reversed will invert the order of items in a list.

Problem 2: Cafe Day (7 points)

Complete the function cafe_day that takes one parameter, orders, which is a list of drink orders for the day at a cafe.

```
def cafe_day(orders):
    return None  # Replace this line with your solution
```

The list orders contains one or more sub-lists, in which each sub-list represents a particular drink order. In each sub-list, there are four elements, in this order:

1. A string that indicates the customer's **membership**, which will be one of these three categories:
    - **Platinum** membership is represented by the letter 'P'
    - **Gold** membership is represented by the letter 'G'
    - **Silver** membership is represented by the letter 'S'
2. An integer that represents the number of **large** drinks for the particular order.
3. An integer that represents the number of **medium** drinks for the particular order.
4. An integer that represents the number of **small** drinks for the particular order.

Your function should process all orders, then return a floating-point number that represents the revenue generated for the day in dollars. *Don't round the return value!* The prices for different sizes are as follows:

Drink Size	Price
Large	$3.50
Medium	$2.50
Small	$1.25

In addition, there will be different privileges depending on different memberships, as described below:

1. For a **Platinum** member: If the customer purchases 3 or more large drinks, **OR** 4 or more medium drinks, then he/she gets 3 free small drinks, but *at most* 3.
2. For a **Gold** member: If the customer purchases at least 10 drinks (in any combination of small, medium and large), then he/she gets a 20% discount.
3. For a **Silver** member: The customer receives 2% off of total price regardless of number of drinks ordered.

Invalid values: Your function should be able to handle invalid values outlined below. In these cases, skip the *entire* drink order and continue to the next one. Valid drink orders have the following characteristics:

- Each sub-list has **exactly** four values.
- The first element in the sub-list, which is the **membership**, is one of the three strings 'P', 'G' or 'S' (all lowercase letters are invalid).
- The number of drinks for all three sizes combined is greater than or equal to zero.
- If the *orders* list is empty, the function should simply return 0.0.

Example calls:

```
orders1 = [['P', 5, 0, 4]]
orders2 = [['S', 1, 2, 3], ['P', 5, 0, 4], ['G', 4, 4, 2]]
orders3 = [['G', 4, 3, 2], ['S', 0, 0, 10], ['P', 1, 4, 3]]

>>>print("cafe_day(orders1): " + str(cafe_day(orders1)))
   18.75
>>>print("cafe_day(orders2): " + str(cafe_day(orders2)))
   51.955
>>>print("cafe_day(orders3): " + str(cafe_day(orders3)))
   49.75
```

Problem 3: Recursive functions for numbers (6 points)

A. Complete the following function that uses recursion to concatenate the numbers in forward order to form a string.

```
# If n = 9, it returns '0123456789'
# If n = 13, it returns '012345678910111213'
# Pre-condition: n >= 0
#
def concat_to(n):
    return None  # Replace this with your implementation
```

B. Complete the following function that uses recursion to concatenates the numbers in reverse order to form a string.

```
# If n = 9, it returns '9876543210'
# If n = 13, it returns '1312111098765432l0'
# Pre-condition: n >= 0
#
def concat_reverse_to(n):
    return None  # Replace this with your implementation
```

C. Write a recursive function gcd(m, n) to calculate gcd of two numbers with the following rules:

```
# If m = n, it returns n
# If m < n, it returns gcd(n-m, n)
# If m > n, it returns gcd(m, n-m)
#
def gcd(m,n):
    return None  # Replace this with your implementation
```

Problem 4: Recursive functions for lists (6 points)

A. Complete the following function that uses recursion to find and return the even elements in the list u.

```
# find_evens([1, 2, 3, 4] returns [2, 4]
# find_evens([1, 2, 3, 4, 5, 6, 7, 8, 9, 10] returns [2, 4, 6, 8, 10]
#
def find_evens(u):
    return None  # Replace this with your implementation
```

B. Complete the following function that returns the zip of two lists u and v of the same length.

```
# zip([1, 2, 3], [4, 5, 6]) returns [1, 4, 2, 5, 3, 6]
#
def zip(u, v):
    return None  # Replace this with your implementation
```