CSC1001: Introduction to Computer Science
Programming Methodology
Assignment 2

Assignment description:
This assignment will be worth 8% of the final grade.

You should write your code for each question in a .py file (please name it using the question name, e.g. q1.py). Please pack all your .py files into a single .zip file, name it using your student ID (e.g. if your student ID is 123456, then the file should be named as 123456.zip), and then submit the .zip file via Blackboard.

Please also write a text file, which provide the details about how to run your code for each question. The text file should be included in the .zip file as well.

Please note that, the teaching assistant may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we may check whether your program is too similar to your fellow students' code using Blackboard.

This assignment is due on 5:00PM, 18 November (Friday). For each day of late submission, you will lose 10% of your mark in this assignment. If you submit more than three days later than the deadline, you will receive zero in this assignment.

Question 1 (*10% of this assignment*):
(*Math: approximate the square root*) There are several techniques for implementing the sqrt function in the math module. One such technique is known as the Babylonian function. It approximates the square root of a number, n, by repeatedly performing a calculation using the following formula:

nextGuess = (lastGuess + (n / lastGuess)) / 2

When nextGuess and lastGuess are almost identical, nextGuess is the approximated square root. The initial guess can be any positive value (e.g., 1). This value will be the starting value for lastGuess. If the difference between nextGuess and lastGuess is less than a very small number, such as 0.0001, you can claim that nextGuess is the approximated square root of n. If not, nextGuess becomes lastGuess and the approximation process continues. Implement the following function that returns the square root of n.

```
def sqrt(n):
```

Question 2 (*15% of this assignment*):
(*Emirp*) An emirp (*prime* spelled backward) is a nonpalindromic prime number whose reversal is also a prime. For example, both 17 and 71 are prime numbers, so 17 and 71 are emirps. Write a program that displays the first 100 emirps. Display 10 numbers per line and align the numbers properly, as follows:

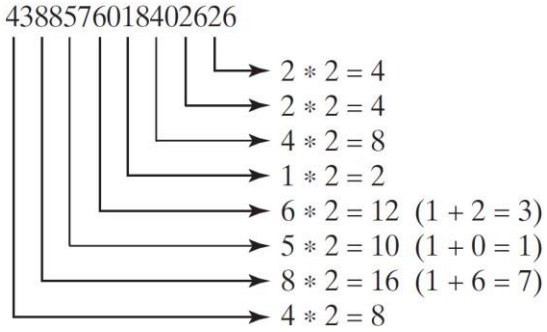
```
 13   17   31   37   71   73   79   97  107  113
149  157  167  179  199  311  337  347  359  389
...
```

Question 3 (*15% of this assignment*):
(*Financial: credit card number validation*) Credit card numbers follow certain patterns: It must have between 13 and 16 digits, and the number must start with:

- 4 for Visa cards
- 5 for MasterCard credit cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a twodigit number, add up the two digits to get a single-digit number.



2. Now add all single-digit numbers from Step 1.

$$4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37$$

3. Add all digits in the odd places from right to left in the card number.

$$6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38$$

4. Sum the results from Steps 2 and 3.

$$37 + 38 = 75$$

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as an integer. Display whether the number is valid or invalid. Design your program to use the following functions:

```
# Return true if the card number is valid
def isValid(number):

# Get the result from Step 2
def sumOfDoubleEvenPlace(number):

# Return this number if it is a single digit, otherwise, return
# the sum of the two digits
def getDigit(number):

# Return sum of odd place digits in number
def sumOfOddPlace(number):
```

Question 4 (*15% of this assignment*):
(*Anagrams*) Write a function that checks whether two words are anagrams. Two words are anagrams if they contain the same letters. For example, silent and listen are anagrams. The header of the function is:

def isAnagram(s1, s2):

(Hint: Obtain two lists for the two strings. Sort the lists and check if two lists are identical.)

Write a test program that prompts the user to enter two strings and, if they are anagrams, displays is an anagram; otherwise, it displays is not an anagram.

Question 5 (*20% of this assignment*):

(*Game: locker puzzle*) A school has 100 lockers and 100 students. All lockers are closed on the first day of school. As the students enter, the first student, denoted S1, opens every

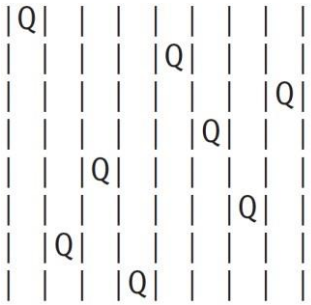
locker. Then the second student, S2, begins with the second locker, denoted L2, and closes every other locker. Student S3 begins with the third locker and changes every third locker (closes it if it was open, and opens it if it was closed). Student S4 begins with locker L4 and changes every fourth locker. Student S5 starts with L5 and changes every fifth locker, and so on, until student S100 changes L100.

After all the students have passed through the building and changed the lockers, which lockers are open? Write a program to find your answer.

(Hint: Use a list of 100 Boolean elements, each of which indicates whether a locker is open (**True**) or closed (**False**). Initially, all lockers are closed.)

Question 6 (*25% of this assignment*):

(*Game: Eight Queens*) The classic Eight Queens puzzle is to place eight queens on a chessboard such that no two queens can attack each other (i.e., no two queens are in the same row, same column, or same diagonal). There are many possible solutions. Write a program that displays one such solution. A sample output is shown below:



Note: you cannot just pre-define a solution and display it.
Please use algorithm to display a possible solution.