**Module 4 Labs**

**COMPSCI 115 – Python Programming**

**Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***Instructions:***

* ***All codes should be saved in this word document***
* ***Your code for each question should be included in this document.***
* ***Test your code and take a screenshot of the output***
* ***Upload the phyton file to BOLT***

Problem Description:

Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:

4 for Visa cards

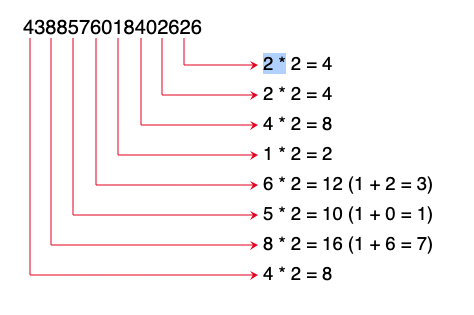
5 for Master cards

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 if a card number is entered correctly or if a credit card is scanned correctly by a scanner. Almost all 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 two-digit number, add up the two digits to get a single-digit number.



2 \* 2 = 4

2 \* 2 = 4

4 \* 2 = 8

1 \* 2 = 2

6 \* 2 = 12 (1 + 2 = 3)

5 \* 2 = 10 (1 + 0 = 1)

8 \* 2 = 16 (1 + 6 = 7)

4 \* 2 = 8

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 Step 2 and Step 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 a long 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):

# Return true if the digit d is a prefix for number

**def** prefixMatched(number, d):

# Return the number of digits in d

**def** getSize(d):

# Return the first k number of digits from number. If the

# number of digits in number is less than k, return number.

**def** getPrefix(number, k):

Here are sample runs of the program:

Sample 1:

Enter a credit card number as a long integer: 4246345689049834

4246345689049834 is invalid

Sample 2:

Enter a credit card number as a long integer: 4388576018410707

4388576018410707 is valid

Output:

