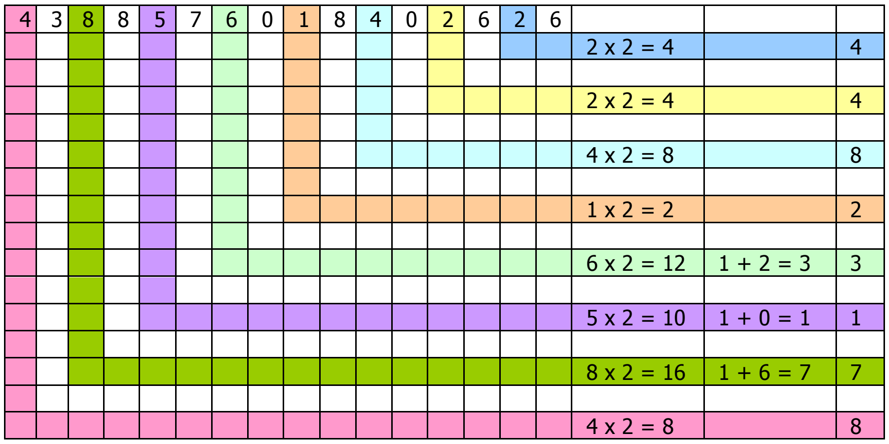
Q1) Credit card number validation: CreditCardValidation.py (20 Points)

Write a program that prompts the user to enter a credit card number as a string. Display whether the number is valid or invalid.

- Credit card number must be between 13 and 16 digits
- Credit card number must start with
  - 4 for Visa
  - 5 for MasterCard
  - 37 for American Express
  - 6 for Discover

Credit card numbers are generated following a validity check proposed by Hans Luhn of IBM in 1954. It is commonly known as the Luhn check or the Mod 10 check. 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. Now add all single-digit numbers from Step 1
   a. 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.
   a. 6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38
4. Sum the results from Steps 2 and 3.
   a. 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.

Design your program to use the following functions:

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

def sumOfDoubleEvenPlace(card_number): # Get the result from Step 1 & 2

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

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

def prefixMatched(card_number, d): # Return true if the digit d is a prefix for card number (d=4,5,6, or 37)

def getSize(card_number): # Return the number of digits in card_number.

Print the output as follows:

If the number is invalid:

Your credit card number is invalid

If the number is valid

Your card is : Visa

The number is valid

Q2: Check password: CheckPassword.py (20 points)

Write a program with a main and a sub function called checkPassword that checks whether a string is a valid password. The main function should get the user input as a string.

Define the function as def checkPassword(password). This function returns whether the password is valid or invalid. Use string methods.

Password rules: The password must

- have at least eight characters.
- contain at least two digits.
- must contain at least one uppercase letter
- must contain at least one lowercase letter
- must contain at least one of [!,@,#,$,%,^,&,*,(,)] (non-letter or non-number character)
    - Create a list for these special characters

Expected output: Print one of the following based on the value returned from the function.

The password you entered is valid

The password you entered is invalid

Q3: Eliminate Duplicates: EliminateDuplcates.py (10 Points)

Write a function that receives list_a and returns a new list, list_b by eliminating the duplicate values in the list.

Use the following function header: def eliminateDuplicates(list_a):

The main function reads a list of integers, invokes the function, and displays the result.

Hint: Read all the numbers and store them in list_a. Create a new list, list_b. Add a number in list_a to list_b. If the number is already in the list, ignore it.)

Sample output:

Original list: [8 2 3 2 8 6 3 4 5 7 2 5 8]

List without duplicates: [8 2 3 6 4 5 7]

Q4: (Anagrams) Anagram.py (10 Points)

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: Create two lists from the two strings entered by the user. Sort the lists using sort method and check if two lists are identical.)

Write a main function that prompts the user to enter two strings and displays the ouput:

Sample output:

The words silent and listen are anagrams

The words silent and learning are not anagrams

Q5: (Statistics: compute mean and standard deviation): (40 points)

Write a program WriteMajor.py that reads user input and creates a file called majors.txt. The program should read a six-digit code for the major and a description from the user and write to the output file. The major code and description are separated by comma. The output file contains only data no header.

Code	Description
200126	Business Honors Program
200500	Transitional Students
200800	Accounting
237800	Engineering Route to Business
240900	Finance
253700	Management
264000	Management Information Systems
264600	Marketing
275000	Supply Chain Management
298000	International Business

Write a program ComputeStats.py that reads two files and computes min, max, mean, and standard deviation of student scores. The program prompts the user to enter the two files scores.txt and majors.txt.

The file scores.txt contains lines of data. In each line, the first element is a 10-digit student ID, the second element is six-digit major, and the remaining elements are scores obtained in various courses. These elements are separated by space. Read each line in scores.txt, store the values in a list, and compute minimum score, maximum score, mean score, and standard deviation for each student.

$$mean = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$deviation = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean)^2}{n-1}}$$

Where n is the number of scores for each student. To compute the standard deviation, store the individual scores in a list, so that they can be used after the mean is obtained. For the major in each line, obtain its description from majors.txt

Write the output in a new file called results.txt. The output contains student ID, Description for Major, Minimum score, Maximum score, Mean score, and Standard Deviation.

Include the following functions in your program:

def deviation(x): # Compute the standard deviation of values.

def mean(x): # Compute the mean of a list of values.

You need not write a program to create scores.txt. Assume that it is given to you. For testing purposes, create a scores.txt file manually with your own data. Each line in scores.txt looks like this:

1232514280 200800 88.2 92.5 98.6 78.5 85.6 75.2 86.9