ITMD 513-01

Lab 4

Programmer: FNU Tripti (A20503656)

Date: 04/07/2022

**Run Time Output**

**Screenshot #1:**

**Validating if the entered Credit Card Number contains only alphabets:**

Graphical user interface, text

Description automatically generated

**Screenshot #2:**

**Validating if the entered Credit Card Number contains alpha numeric values:**

**A screenshot of a computer

Description automatically generated**

**Screenshot #3:**

**Validating if the entered Credit Card Number’s length is less than 13 digits**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Screenshot #4:**

**Validating the entered Credit Card Number’s length is more than 16 digits**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Screenshot #5:**

**Valid Credit Card starting with “37” – AMERICAN EXPRESS Scenario – Satisfying Luhn’s formula**

**Text

Description automatically generated**

**Screenshot #6:**

**Valid Credit Card starting with “5” – MASTER CARD Scenario – Satisfying Luhn’s formula**

**Text

Description automatically generated**

**Screenshot #7:**

**Valid Credit Card starting with “6” – DISCOVER Scenario – Satisfying Luhn’s formula**

**Graphical user interface, text, application

Description automatically generated**

**Screenshot #8:**

**Valid Credit Card starting with “4” – VISA Scenario – Satisfying Luhn’s formula**

**Graphical user interface, text, application

Description automatically generated**

**Screenshot #9:**

**Invalid Cards not satisfying the Luhn’s formula ~ AMERICAN EXPRESS scenario**

**Graphical user interface, text, application

Description automatically generated**

**Screenshot #10:**

**Invalid Cards not satisfying the Luhn’s formula ~ DISCOVER scenario**

**Graphical user interface, text

Description automatically generated**

**Source Code:**

# This application validates the  
  
# Programmer Information  
# Name: FNU Tripti  
# A-ID: A20503656  
# Course: ITMD-513  
# Date: 07/04/2022  
# Lab #: 4  
  
  
# if the card number is valid, this function returns true  
import sys  
from datetime import date  
  
def isValid(number):  
 if number.isalpha():  
 return -1  
 elif any(c.isalpha() for c in number):  
 return -2  
 elif number.isnumeric():  
 # Check for the length of the credit card numbers using the getSize method  
 if getSize(number) < 13:  
 return -3  
 elif getSize(number) > 16:  
 return -4  
 else:  
 return 4  
  
  
# obtain the result from Step 2  
def sumOfDoubleEvenPlace(number):  
 # Reversing the string to calculate the right to left calculations (Reference: Textbook)  
 reverse\_number = number[::-1]  
  
 # Iterating over the digits in a number  
 place = 1  
 double\_some\_even\_digits = 0  
 for i in reverse\_number:  
 if place % 2 == 0:  
 # Accumulating the doubled sum also calling the function if the digit becomes 2-digit number  
 double\_some\_even\_digits = double\_some\_even\_digits + getDigit(2 \* int(i))  
 place = place + 1  
 return double\_some\_even\_digits  
  
  
# if variable number is a single digit, then return the number  
# otherwise, return number as the sum of the two digits  
def getDigit(number):  
 if number < 10:  
 return number  
 else:  
 sum\_of\_digits = (number % 10) + (number // 10)  
 return sum\_of\_digits  
  
  
# this function returns the sum of odd place digits in variable number  
def sumOfOddPlace(number):  
 # Reversing the string to calculate the right to left calculations (Reference: Textbook)  
 reverse\_number = number[::-1]  
  
 # Iterating over the digits in a number  
 place = 1  
 some\_odd\_digits = 0  
 for i in reverse\_number:  
 if place % 2 != 0:  
 some\_odd\_digits = some\_odd\_digits + int(i)  
 place = place + 1  
 return some\_odd\_digits  
  
  
# if the digit d is a prefix for variable number, then return true  
def prefixMatched(number):  
 prefix = ""  
 for x in number:  
 if int(x) == 3:  
 prefix = getPrefix(number, 2)  
 break  
 else:  
 prefix = getPrefix(number, 1)  
 break  
 return prefix  
  
  
# this function returns the number of digits in variable d  
def getSize(d):  
 return len(d)  
  
  
# returns the first k number of digits from variable number but if the  
# number of digits in variable number is less than k, return number  
def getPrefix(number, k):  
 prefix = ""  
 if int(number) < k:  
 return k  
 else:  
 count = 1  
 for x in number:  
 prefix += x  
 if count >= k:  
 break  
 count = count + 1  
 return prefix  
  
  
def getFinalValidation(number):  
 if number % 10 == 0:  
 return True  
 else:  
 return False  
  
  
# This method is used to the input the credit card number.  
def inputCreditCardNumber():  
 print("Please input the credit card number:")  
 cc\_number = input()  
 return cc\_number.replace(" ", "")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 print("This program for course ITMD-513 is executed by FNU Tripti (A20503656) on : ", date.today())  
  
 cc\_number = inputCreditCardNumber()  
  
 # Calling the isValid function to validate the inout number  
 is\_valid = isValid(cc\_number)  
  
 if is\_valid == -1:  
 print("The Credit Card Number contains only alphabets; This is not a valid Credit Card.")  
 elif is\_valid == -2:  
 print("The Credit Card Number contains alphabets; This is not a valid Credit Card.")  
 elif is\_valid == -3:  
 print("The length of the card is less than 13 digits. This is not a valid Credit Card.")  
 elif is\_valid == -4:  
 print("The length of the card is more than 16 digits. This is not a valid Credit Card.")  
 elif is\_valid == 4:  
 print("The Credit Card entered has a valid length and does not contain any alphabets.")  
  
 # Calculation of the Company which issued the card.  
 prefix = prefixMatched(cc\_number)  
  
 if prefix == "37":  
 print("The Card is an AMERICAN EXPRESS Card.")  
 elif prefix == "6":  
 print("The Card entered is a DISCOVER Card.")  
 elif prefix == "5":  
 print("The Card entered is a MASTER Card.")  
 elif prefix == "4":  
 print("The Card entered is a VISA Card.")  
 else:  
 print("The Card is Invalid; And Is not issued by a valid Company in the list")  
 sys.exit()  
  
 sum\_of\_doubled\_even\_places = sumOfDoubleEvenPlace(cc\_number)  
 print("The sum of even digits doubled is : ", sum\_of\_doubled\_even\_places)  
 sum\_of\_odd\_place\_digits = sumOfOddPlace(cc\_number)  
 print("The sum of odd digits is : ", sum\_of\_odd\_place\_digits)  
 total\_sum = sum\_of\_doubled\_even\_places + sum\_of\_odd\_place\_digits  
 final\_validation = getFinalValidation(total\_sum)  
 if final\_validation:  
 print("The Card is Valid; As it satisfies all the conditions and Luhn's Formula as well.")  
 else:  
 print("The Card is Invalid; Because it Fails to satisfy the Luhn's formula.")

**QUESTIONNAIRE:**

1. **Explain how placing your code into a separate module may benefit other python files involving credit card transactions.**

**Ans.** Placing the code in a separate module will package this code for me as a one shop stop for validating my credit card and receiving the information about my credit card, that whether it is an AMEX, DISCOVER, MASTERCARD or VISA.

1. **Investigate the link below. What are some other methods you can use in your project to verify a credit card transaction? Include at least three methods that may help validate user input of credit card data**.

**Ans.** All the methods that I have written in my program can be used to validate the card for this page.

* **isValid(int)** method can be used to check alphanumeric characters in the number and also the length of the card whether it is between 13 and 16 digits.
* **sumOfDoubleEvenPlace(number)** method can be used to validate sum the doubled even placed digits of the number to perform part 2 of the Luhn’s formula (Validation).
* **getDigit(number)** As this function gets used by the **sumOfDoubleEvenPlace(number)** this will be used to.
* **sumOfOddPlaces(number)** method will also be used to calculate the step 3 of the Luhn’s formula.
* **prefixMatched(number)** can be used to determine the institution that issued the card.
* **getSize(number) and getPrefix(number, k)** methods can also be used.

1. **How would the use of regular expressions aid in reducing any bloated code to verify credit card data within an application? Give an example.**

**Ans.** I felt the need of using regex code to imply all the validation. For example matching alpha numeric values etc. can be calculated using the regex. Length of the credit can also be validated using the regex. Moreover, complex regex along with the for loop can be used to calculate the step 2 & step 3 of the Luhn’s formula.

1. **Describe some ways to ultimately check your code to validate various card numbers for any faulty entry and / or invalid card numbers or even valid card numbers.**

**Ans.** Following are some ways to check the validate the code:

I have even mentioned in the screenshots. I have visited the website provided by the professor and got some fake numbers from there. The code is working fine for those generated numbers as they satisfy the Luhn’s formula. I also incremented the odd place digits one to change the sum of the digits in the step 4 and then the card was invalidated by the code.

Also: there are some other valid cards that we were not supposed to accept for example:

American Express Card starting with 34. This card will be rejected by our code.

Some other methods will be giving less than 13 and more than 16-digit values etc.

1. **Why would there be a reason to store any invalid numbers that were input by a particular user?**

**Ans.** It may help us in following ways:

* Caching those numbers will help us in the Fraud detection. We will know that somebody is trying to crack the system.
* Creating such list before running our program will save us number of iterations to valid the card, if the number appears in search, we will know that it is an invalid card.
* This storage will provide a good dataset for the new functionality that we add to the code. These values will be good for the robust testing of the code.