**Program source Code:**

**Lab\_04\_guru\_module2.py**

# If the card number is valid, this function returns True  
def isValid(number):  
 total = sumOfDoubleEvenPlace(number) + sumOfOddPlace(number)  
 return (total % 10 == 0)  
  
def sumOfDoubleEvenPlace(number):  
 total = 0  
 num = str(number) # Iterate through the digits in the number, starting from the second to last digit  
 for i in range(len(num) - 2, -1, -2):  
 total += getDigit(int(num[i]) \* 2)  
 return total # If the doubled digit is a single digit, return it  
  
# Otherwise, return the sum of the two digits  
def getDigit(number):  
 if number < 10: # If the number is less than 10, return it  
 return number  
 else:  
 # Otherwise, return the sum of the two digits  
 return number // 10 + number % 10  
# This function returns the sum of odd place digits in variable number  
def sumOfOddPlace(number):  
 total = 0  
 num = str(number) # Iterate through the digits in the number, starting from the last digit  
 for i in range(len(num) - 1, -1, -2):  
 total += int(num[i])  
 return total  
# If the digit d is a prefix for variable number, then return True  
def prefixMatched(number, prefix):  
 return str(number).startswith(prefix)

**Main.py**

from lab\_04\_guru\_module2 import \* #importing the module from the same package  
import time #Importing Time to print time stamp  
timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime()) #Time stamp declaration  
print("Program execution at:", timestamp) #print statement for timestamp  
print('GURUTEJA\_KANDERI') #Author name  
print('A20526883 \nOpen Source Programming Python') #Hawk\_ID  
print('LAB\_04--Using Functions : Luhn’s Formula for Credit Card Validation') #lab number and lab name  
card\_number = input("Enter credit card number: ") #Prompts user to enter the credit card number  
  
if card\_number.strip() == "":  
 print("No credit card number entered.") # prints this message if user clicked enter without entering the credit card number  
else:  
 try:  
 if isValid(int(card\_number)): # if user enters the number checks the code which type of card it is  
 card\_type = ""  
 if prefixMatched(card\_number, "37"):  
 card\_type = "American Express"  
 elif prefixMatched(card\_number, "6"):  
 card\_type = "Discover"  
 elif prefixMatched(card\_number, "5"):  
 card\_type = "MasterCard"  
 elif prefixMatched(card\_number, "4"):  
 card\_type = "Visa"  
 else:  
 card\_type = "Unknown"  
  
 print("Valid", card\_type, "credit card number.") #prints the valid and card type if it is satisfying the code  
 else:  
 print("Invalid credit card number. Please enter a valid number.")  
 except ValueError:  
 print("Invalid credit card number. Please enter a valid numeric value.") # prints if the card number is invalid

**OUTPUT and VALIDATION:**

**VISA Validation:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**MASTERCARD:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**AMEX Card:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**DISCOVER CARD:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**INVALID CARD NUMBER**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**When No Credit card Number:**

**A screenshot of a computer program

Description automatically generated with medium confidence**

**Question and Answers:**

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

* Maintaining different modules helps a developer to read and understand the code properly.
* The main advantage is reusability of the code, where you keep the functions in one module, and you can import that module which has functions and make the code run in each module differently.
* The modular approach will help your team to understand the code, and if it requires to change anything in the future. You can change the exact module and it will reflect in the entire code.

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.**

To verify the card transaction, we can use CVV, Valid thru which is month and year (01/26)

, and Name and Address verification on Card:

**1.CVV: Card validation Value:** This is the number which is printed on the backside of the card and we can use it for further verification. It make sure that user has the complete card information and decreases the fraudulent actions.

**2. Valid Thru:** This is another concept which we generally use to verify the card. Which has the information of card valid through the year and month of like 06/24. This information also makes the card transaction more secure.

**3. Name on the Card or address Verification:**

This enables the user to provide more secure way of transacting through card, It will prompt the user to enter the Name printed on the card or Address that has been provided while creating the card. Which is another method to validate user input data for making secure transactions.

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.**

Within an application, Regular expressions helps to reduce the bloated code while verifying credit card data

Simplified Validation Logic:

To perform a variety of validation tasks, such as checking the length of a string, verifying that a string contains only numeric characters, or ensuring that a string matches a specific format.

Regular expressions can be reused throughout the codebase, which promotes modular and reusable code. This reduces duplication and improves maintainability.

def isValid(number):

"""Validates a credit card number using regular expressions.

Args:

number: The credit card number to be validated.

Returns:

True if the card number is valid, False otherwise.

"""

regex = r"^[0-9]{13,16}$"

if not re.match(regex, number):

return False

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.**

**Check with Lunh’s Formula:**

The Luhn algorithm is a simple but effective way to validate credit card numbers. It is used by many credit card companies to help prevent fraudulent transactions.

**Card Number Length for different card types:**

For Example, Amex cards will have only 15 digits Discover and master card will have 16 digits. We can validate the cards with their length and what number it starts with and other parameters.

**Non Numeri’s issues:**

We can include the validation to check if the number entered is number numeric or non-Numeric.

If it is Non -Numeric it should prompt, Enter correct Credit card Number, Invalid Numbers.

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

**Error tracking and troubleshooting**: Storing invalid numbers allows you to track and investigate any errors or issues related to credit card validation. When users encounter errors with their input, having access to the invalid numbers can assist in troubleshooting and resolving the problem more effectively.

**User feedback and improvement:** Storing invalid numbers provides an opportunity to provide feedback to users regarding their input errors. By informing users about the invalidity of their credit card numbers and providing guidance on correct input formats, you can help them improve their future interactions with the system.

Along with that this feedback and information from customers helps the developers to understand more about the code and improve from their end to provide better user experience.