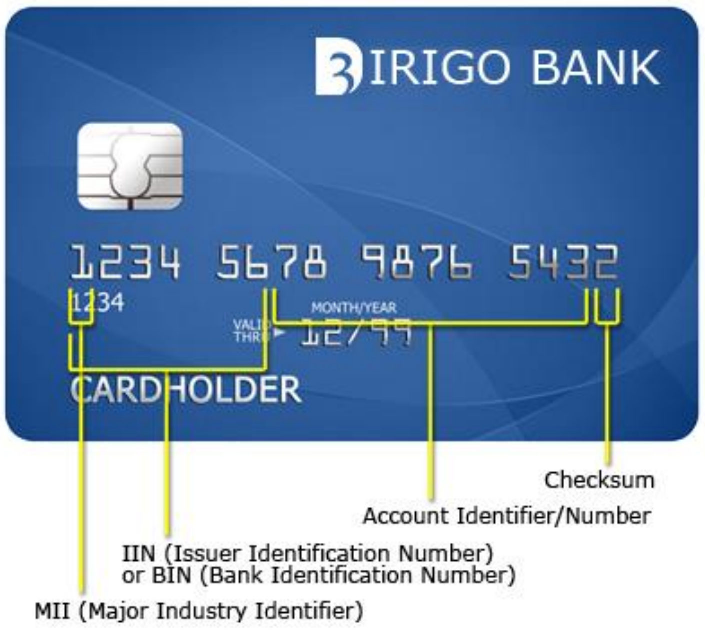
Payment Card Validator

Introduction

In most developed countries, both online and in everyday life, everyone has switched to non-cash payments, using payment cards such as credit and debit cards, as well as saved value cards, gift cards and other similar cards. Each of these cards has its own number, embossed on the face of the payment card, and encoded on a magnetic stripe and a chip. These cards are numbered not randomly, but in accordance with the ISO/IEC 7812 standard using a certain pattern:


IRIGO BANK
1234 5678 9876 5432
1234
VALID THRU
MONTH/YEAR
12/99
CARDHOLDER
Checksum
Account Identifier/Number
IIN (Issuer Identification Number)
or BIN (Bank Identification Number)
MII (Major Industry Identifier)

For example, a plastic card has a sixteen-digit number. The identifier is divided into four blocks each containing 4 characters. All codes carry certain information required for processing, clearing or authorization.

The first 6 digits represent the BIN (Bank Identification Number) - a bank identifier designed to encode data. The Bank Identification Number (BIN) is a numbering system developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) to identify card issuing institutions.

The BIN number consists of the Major Industry Identifier (Major Industry Identifier), the first digit, and the scheme or payment system on which the card operates (Visa®, Mastercard®, American Express®, and others).

Such qualifier is unique for each type of cards of a particular financial institution. This parameter also allows one to find out information about the issuing bank, which country the card belongs to and what type of product it corresponds to.

The correctness of the sequence of digits in card numbers is compared using the Luhn Algorithm, a system for calculating the check digit in the number of payment cards.

The Luhn algorithm or Luhn formula, also known as the “modulus 10” or “mod 10” algorithm, named after its creator, IBM scientist Hans Peter Luhn, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers, National Provider Identifier numbers, etc.

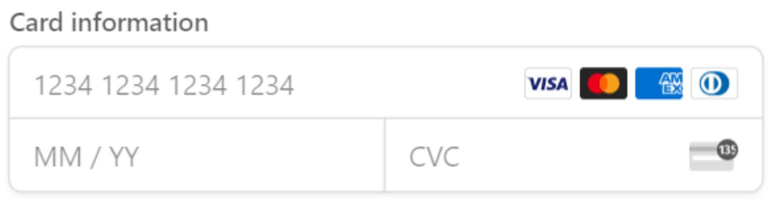
Payment cards validation

Processing when paying by payment cards online or offline consists of the following steps:

1. The processing service, having processed the payment card number, determines the payment system and sends a corresponding request to complete the transaction.

2. The payment system determines the issuing bank, sends a request, and then the issuing bank, conducting its own security check and authentication, gives permission for the transaction or rejects it in case of errors.

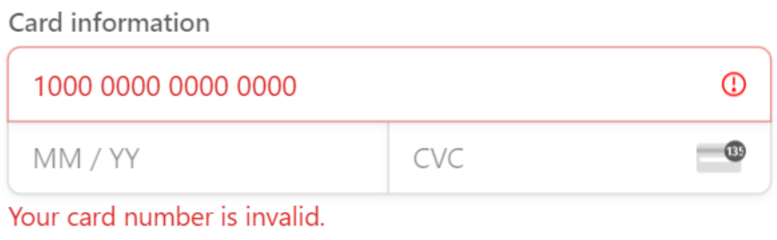
However, there is also a preliminary step 0, the stage of checking the validity of the payment card number by the Luhn Algorithm. If the card number does not match the pattern, the processing service will simply refuse to send a request to the payment system. Validation of the payment card number on the client's side allows to prevent overloading of the processing service, payment systems, as well as the servers of issuing banks. Client-side validation serves as a means of preventing Denial-of-Service (DoS) attacks - when the processing power of the server cannot handle the overload of incoming requests and fails. For example, one of the largest processing services, Stripe®, uses client-side validation of payment card numbers. If the user entered the wrong card number, the server simply would not accept such a number:


Card information

1234 1234 1234 1234

MM / YY

CVC


Card information

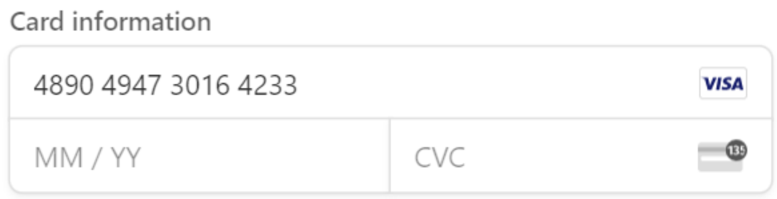
1000 0000 0000 0000

MM / YY

CVC

Your card number is invalid.

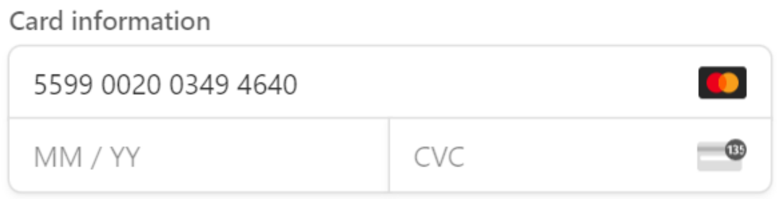
In addition to checking the validity, Stripe® also shows the payment system of the entered card. This entire procedure takes place on the client side, in this case in the user's browser.


Card information

4890 4947 3016 4233

MM / YY

CVC


Card information

5599 0020 0349 4640

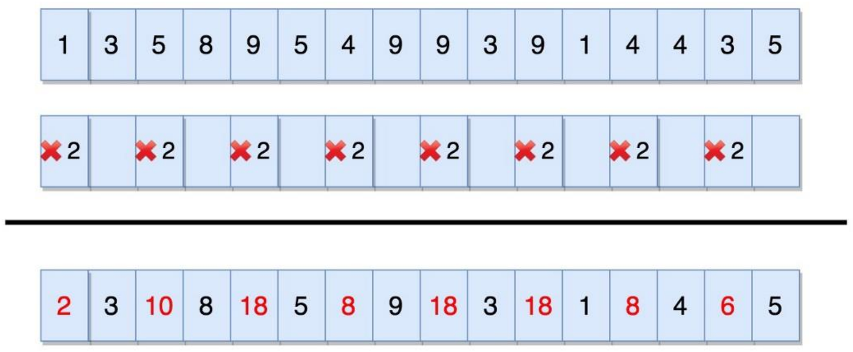
MM / YY

CVC

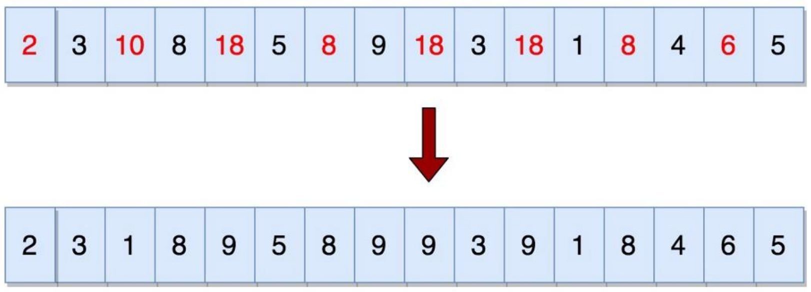
Validation check. Luhn Algorithm

Luhn Algorithm for validating payment card numbers works as follows:

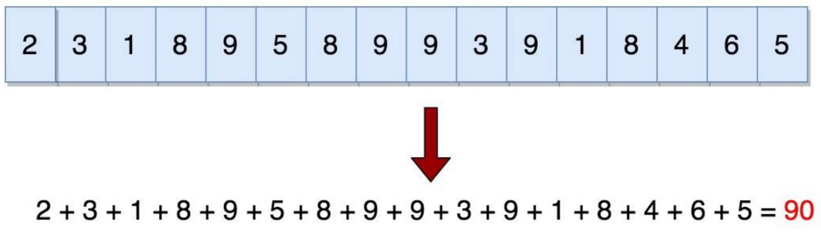
1. Starting from the rightmost side, every other digit is doubled.



2. If the result is a two-digit number, add the individual digits of the two-digit number to produce a single-digit number.



3. Next, all single-digit numbers are summed up. If the received number is a multiple of 10 (if the modulus of the sum to 10 is zero), then the card number is valid. If not, the card number is invalid.



In this case, the result 90 is a multiple of 10, which means that the card is valid.

Payment Card Validator

The operating principle of this software is based on the rules and methods described above. The software operates in two different modes, command line mode and graphical interface mode. One or another operating mode is started depending on the user's choice. The software package consists of three separate python files and an Assets folder containing the necessary data for work:

The **cmd_mode.py** file is responsible for working in command line mode.

The **gui_mode.py** file is responsible for working in GUI mode. The program's graphical interface was developed using Tkinter, a cross-platform event-driven graphics library.

The **run.py** file is the main startup file that gives the user a choice about which program mode to use. The file contains the following code:

```
# PAYMENT CARD VALIDATOR RUN FILE
APP_MODE = input("TYPE 'c' TO RUN IN COMMAND-LINE MODE, TYPE 'g' FOR GUI MODE OR
ANY OTHER KEY TO EXIT THE APP: \n")

if APP_MODE == "c":
    # run CMD Mode file
    import cmd_mode

elif APP_MODE == "g":
    # run GUI Mode file
    import gui_mode
else:
    # quit the app
    print("Quitting...")
```

After running run.py, the program prompts the user to type "c" to run in command line mode or "g" to start the graphical interface.

Command line mode

The first function that is launched in command line mode is the input_validation () function, which is prompting the user to enter the payment card number and then checking the entered information for errors. Thus, if the user input is not a number or contains letters or symbols, the program returns the error "Sorry your input is not a digit!" In addition, the program removes all unnecessary spaces entered by the user when typing. The program also shows the error "You are out of range (11-19 digits)" if the user input is less than 11 or more than 19 digits, since the payment card numbers are in this range. Also, the program will not accept input consisting of all zeros. After the user's input is pre-validated, it is stored in the CARD_NUMBER variable as an integer.

```
# Get user input (card number), and check errors
def input_validation():

    # initital
    card_num = "testnum"
    in_range = False

    while card_num.isdigit() == False or in_range == False or int(card_num) == 0:
        # get user input and remove spaces
        card_num = input("Please enter payment card number (11-19 digits): 
").replace(" ", "")

        # digit check
        if card_num.isdigit() == False:
            print("Sorry your input is not a digit!")

        # range check
        if card_num.isdigit() == True:
            if len(card_num) >= 11 and len(card_num) <= 19:
                in_range = True
            else:
                print("You are out of range (11-19 digits)")
                in_range == False

    return int(card_num)

CARD_NUMBER = input_validation()
```

Next, the IIN / BIN of the **potentially** valid card is calculated. For this, the integer from the CARD_NUMBER variable is converted to a string and stored in the *num* variable. After that, the first six digits of the card number are selected by the slicing method, after which the bin_check () function converts the string back to an integer and the result is stored in the CARD_BIN variable for later use:

```
num = [int(x) for x in str(CARD_NUMBER)] # convert integers

# Calculate IIN (Issuer Identification Number) / BIN (Bank identification number)
eight = num[:8]
six = num[:6]

def bin_check(bintype): # eight for 8 digit BIN, six for six digit BIN
    return int("".join([str(i) for i in bintype]))

CARD_BIN = bin_check(six)
```

After, the program uses the first digit of the BIN number to search for Major industry identifier using the local database in a dictionary format:

```
# Major industry identifier
def mii_check():
    mii_base = {"ISO/TC 68 and other industry assignments": 0,
                "Airlines": 1,
                "Airlines, financial and other future industry assignments": 2,
                "Travel and entertainment": 3,
                "Banking and financial.": 4,
                "Banking and financial": 5,
                "Merchandising and banking/financial": 6,
                "Petroleum and other future industry assignments": 7,
                "Healthcare, telecommunications and other future industry
assignments": 8,
                "For assignment by national standards bodies": 9}

    for key, value in mii_base.items():
        if num[0] == value:
            return key

INDUSTRY = mii_check()
```

The function is called and the returned result is stored in the INDUSTRY variable.

At this stage, the payment system of the card is calculated, again using the local database:

```
# Calculate Card Brand (Visa, MasterCard, Maestro, etc.)
def card_brand():

    brands_database = {"American Express": [34,37],
                       "China T-Union": [31],
                       "China UnionPay": [62],
                       "Diners Club International": [36],
                       "Diners Club United States & Canada": [54],
                       "Discover Card": [6011,65]+list(range(644,650)),
                       "Discover Card & China UnionPay co-branded":
list(range(622126,622926)),
                       "UkrCard": list(range(60400100,60420100)),
                       "RuPay": [60,65,81,82,508],
                       "RuPay-JCB co-branded": [353,356],
                       "InterPayment": [636],
                       "InstaPayment": [637,638,639],
                       "JCB": list(range(3528,3590)),
                       "Laser": [6304, 6706, 6771, 6709],
                       "Maestro UK": [6759, 676770, 676774],
                       "Maestro": [5018, 5020, 5038, 5893, 6304, 6759, 6761, 6762,
6763],
                       "Dankort": [5019],
                       "Dankort & Visa co-branded": [4571],
                       "Mir": list(range(2200,2205)),
                       "NPS Pridnestrovie": list(range(6054740,6054745)),
                       "MasterCard": list(range(2221,2721))+list(range(51,56)),
                       "Troy": [65, 9792],
                       "Visa": [4],
                       "Visa Electron": [4026, 417500, 4508, 4844, 4913, 4917],
                       "UATP": [1],
                       "Verve":
list(range(506099,506199))+list(range(650002,650028)),
                       "LankaPay": [357111],
                       "UzCard": [8600],
                       "Humo": [9860],}

    card = str(CARD_NUMBER)

    matches = []
    for key, value in brands_database.items():
        for item in value:
            if card.startswith(str(item)):
                matches.append(item)
            else:
                matches.append(0)

    for key, value in brands_database.items():
        for item in value:
            if item == max(matches):
                return key
            else:
                return "Not found"
```

```
BRAND = card_brand()
```

If the BIN number of the card matches the data in the database, the function returns the name of the corresponding payment system, if the data does not match, the function returns the string "Not found." The result is stored in the BRAND variable.

Major industry identifier and Card Brand databases were compiled manually using information from open sources.

The next function is the most important in the work of the program, it is the verification of the entered number using the Luhn Algorithm. The algorithm works in the same way as described above. First, the digits of the payment card number from the rightmost digit are selected, after which they are sorted and stored in two different variables. One variable stores the selected digits following the rightmost digit in increments of two, the other variable stores the digits selected with the inclusion of the rightmost digit in increments of two. Next, the digits of one variable are summed. The digits are doubled with a step of two in the second variable, with further split into single-digit numbers. After that, the sum of all the numbers from the first variable is added to the sum of the split numbers plus the rest from the second variable, and the result is checked for a multiple of 10 by the modulo operator. This function returns boolean values True or False, depending on whether the validation of the card number is successful or not:

```
# Card validation by Luhn Algorithm
def check_luhn():

    odd_digs = num[-1::-2] # reversed digits from the end with step 2
    even_digs = num[-2::-2] # reversed digits from the end+1 with step 2

    odd_summ = 0
    odd_summ = odd_summ + sum(odd_digs)

    # we multiply each item in reversed_even_digits_list by 2, joining the digits,
and converting it into integers
    evensplitted = int("".join(map(str, [i * 2 for i in even_digs])))

    # we add integers to list and take sum of them
    even_summ = sum([int(a) for a in str(evensplitted)])

    return (odd_summ + even_summ) % 10 == 0  # if sum of the numbers mod 10 is
equal to 0 the card is valid
```

All that happens next depends on whether the payment card number is validated by the Luhn Algorithm or not. It would be more logical to check the validation with this algorithm at the very beginning before calculating the BIN number and other data, however, as it turned out, not all payment cards from our databases are generated and validated according to the ISO / IEC 7812 standard, which means there are potentially some card numbers that will fail validation by the Luhn Algorithm however, the program will still calculate their BIN number, payment system and Major industry identifier.

```
if check_luhn():
    print(f"Card number {CARD_NUMBER} is valid \nIIN/BIN number: {CARD_BIN} \nCard
industry: {INDUSTRY} \nCard Brand: {BRAND} \n")
    #----------------------------Online-Checks---------------------------------#
    # check internet connection
    def connection_check():
        url='http://www.google.com/'
        timeout=5
        try:
            net = requests.get(url, timeout=timeout)
            return True
        except requests.ConnectionError:
            print("Internet connection error")
        return False

    if connection_check():
        try:
            cardinfo = get(f"https://lookup.binlist.net/{CARD_BIN}",
headers={'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36',
"Accept-Version": "3"}).json()
            card_country = card_currency = card_bank = bank_url = bank_phone =
"Not found"
            try:
                card_country = cardinfo["country"]["name"]
            except:
                pass
            try:
                card_currency = cardinfo["country"]["currency"]
            except:
                pass
            try:
                card_bank = cardinfo["bank"]["name"]
            except:
                pass
            try:
                bank_url = cardinfo["bank"]["url"]
            except:
                pass
            try:
                bank_phone = cardinfo["bank"]["phone"]
            except:
                pass
            print(f"DATABASE SEARCH: \nCard Country: {card_country} \nCurrency:
{card_currency} \nIssuing Bank: {card_bank} \nBank URL: {bank_url} \nBank
contacts: {bank_phone}")

        except:
            print("Error searching online database. Make sure your input is
correct")
    else:
        print("Check your connection and try again")
else:
    print("Card number is not valid, or it is not generated by Luhn Algorithm.")
```

At this stage, a check is made if the number of the entered payment card is valid, and if so, the program, using the BIN number of the card, makes a request to the online database to obtain the remaining parameters of the payment card. However, before that, the program will print out all the data that was received by offline calculation.

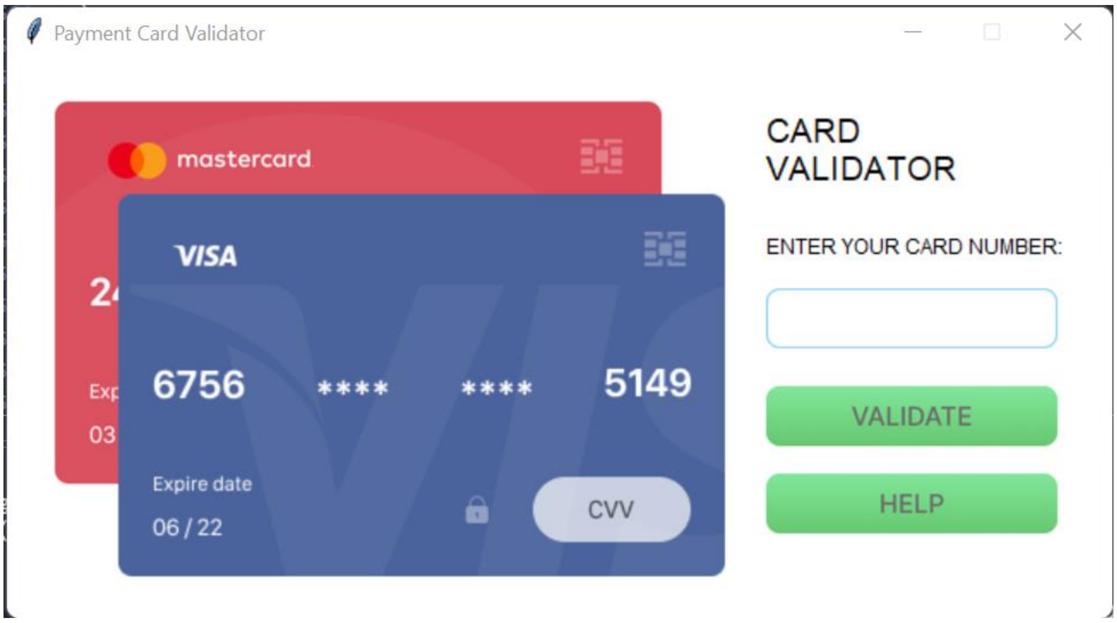
Before accessing the online database, the program performs a preliminary check for the presence of an Internet connection on the user's device. The program will try to check the connection to the google.com server and, if the connection is successful, it will start extracting data from the online database. This method was chosen primarily because the response time of the Google server is significantly faster than the response time of the online database. In addition, the number of queries per minute in accessing the online database is limited.

Thus, if the Internet on the user's device is unavailable, the program will show the "Internet connection error", if the online database is unavailable for any reasons, the program will show "Error searching online database. Make sure your input is correct", and if the entered card number do not pass the validation by Luhn Algorithm, the program will give the error "Card number is not valid, or it is not generated by Luhn Algorithm." To avoid errors with termination of the program, try except methods are used. To ensure that the variables where the data will be stored after receiving from the online database are not empty, they initially store the "Not found" string.

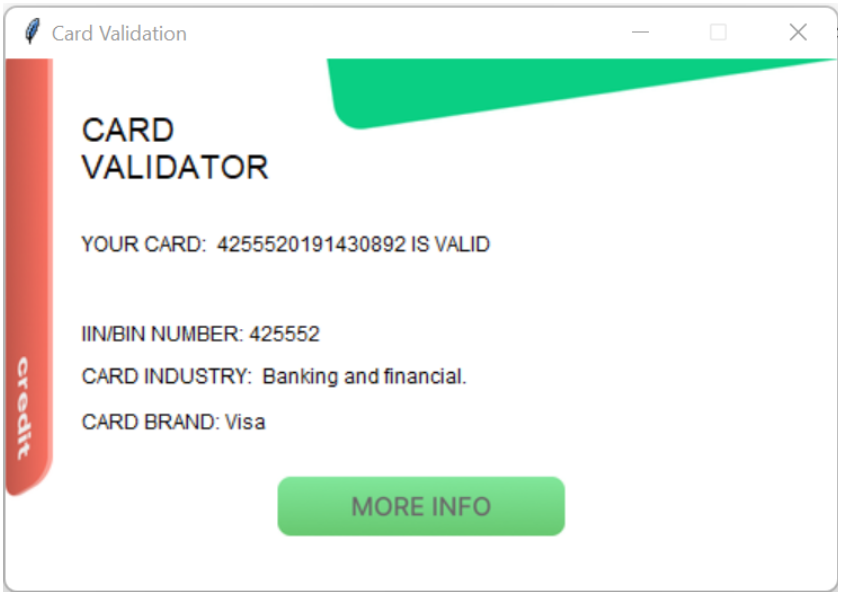
GUI Mode

Functions in the graphical interface mode work in a similar way, except that in the graphical mode there are additionally hundreds of lines of code responsible for the graphical arrangement of elements. In addition, in the graphical mode, input validation functions are implemented to filter data in real time, which allows the program to control the type of data entered by the user. Therefore, in graphical mode, users cannot enter letters or symbols in the input field. There is also a limitation when entering a payment card number, the user cannot enter more than 19 digits.

This is how the main program window looks like on Windows 11:



The main window of the program has two buttons clicking on each of which the corresponding windows open. This is how the validation window looks like, it opens after entering the card number and clicking on the validate button:



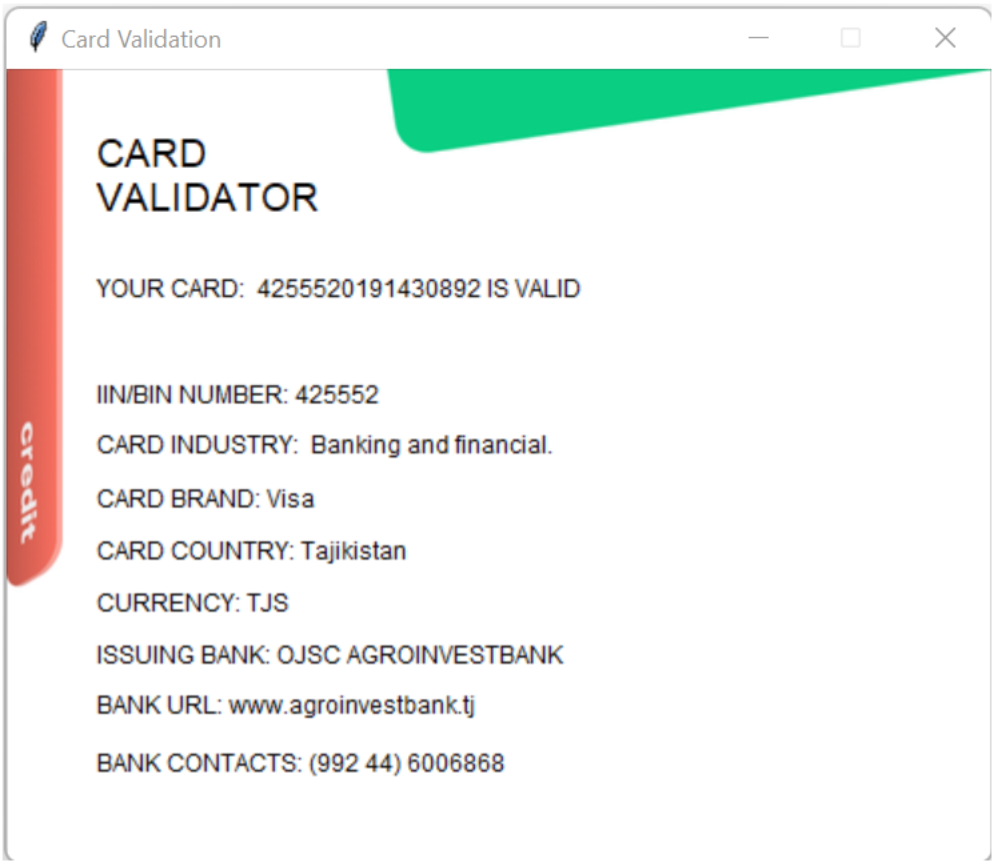
YOUR CARD: 4255520191430892 IS VALID

IIN/BIN NUMBER: 425552

CARD INDUSTRY: Banking and financial.

CARD BRAND: Visa

After clicking on the More Info button, the program shows additional data about the number of the entered payment card obtained from the online database:



CARD
VALIDATOR

YOUR CARD: 4255520191430892 IS VALID

IIN/BIN NUMBER: 425552

CARD INDUSTRY: Banking and financial.

CARD BRAND: Visa

CARD COUNTRY: Tajikistan

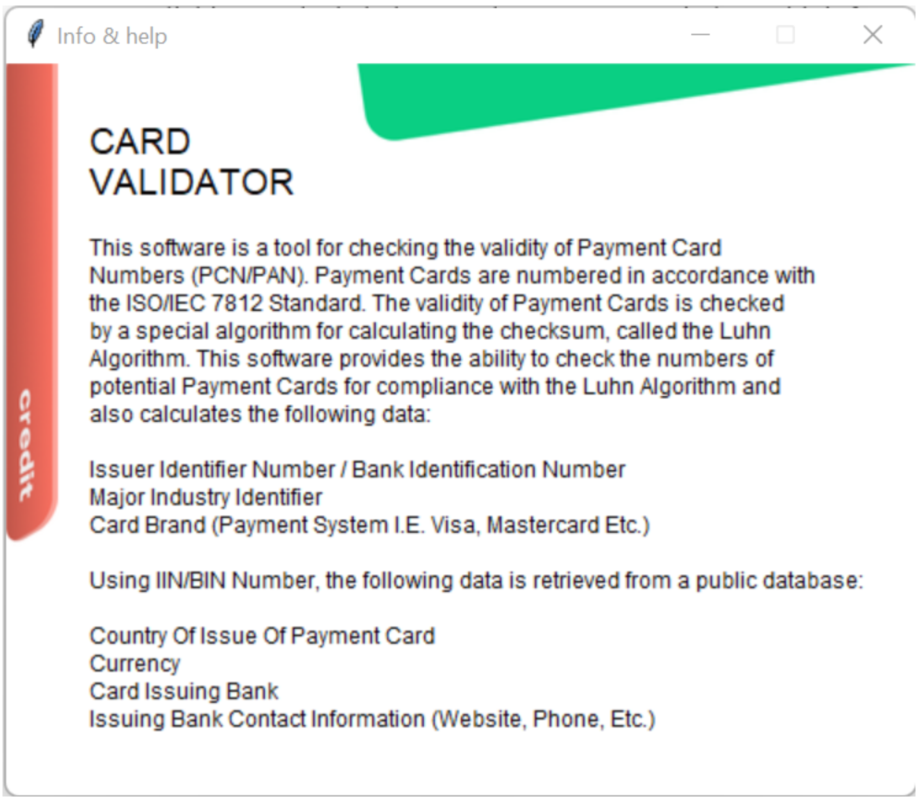
CURRENCY: TJS

ISSUING BANK: OJSC AGROINVESTBANK

BANK URL: www.agroinvestbank.tj

BANK CONTACTS: (992 44) 6006868

By clicking on the help button, the user sees a window with information on the program:



CARD
VALIDATOR

This software is a tool for checking the validity of Payment Card
Numbers (PCN/PAN). Payment Cards are numbered in accordance with
the ISO/IEC 7812 Standard. The validity of Payment Cards is checked
by a special algorithm for calculating the checksum, called the Luhn
Algorithm. This software provides the ability to check the numbers of
potential Payment Cards for compliance with the Luhn Algorithm and
also calculates the following data:

Issuer Identifier Number / Bank Identification Number
Major Industry Identifier
Card Brand (Payment System I.E. Visa, Mastercard Etc.)

Using IIN/BIN Number, the following data is retrieved from a public database:

Country Of Issue Of Payment Card
Currency
Card Issuing Bank
Issuing Bank Contact Information (Website, Phone, Etc.)

The following function is responsible for filtering the input:

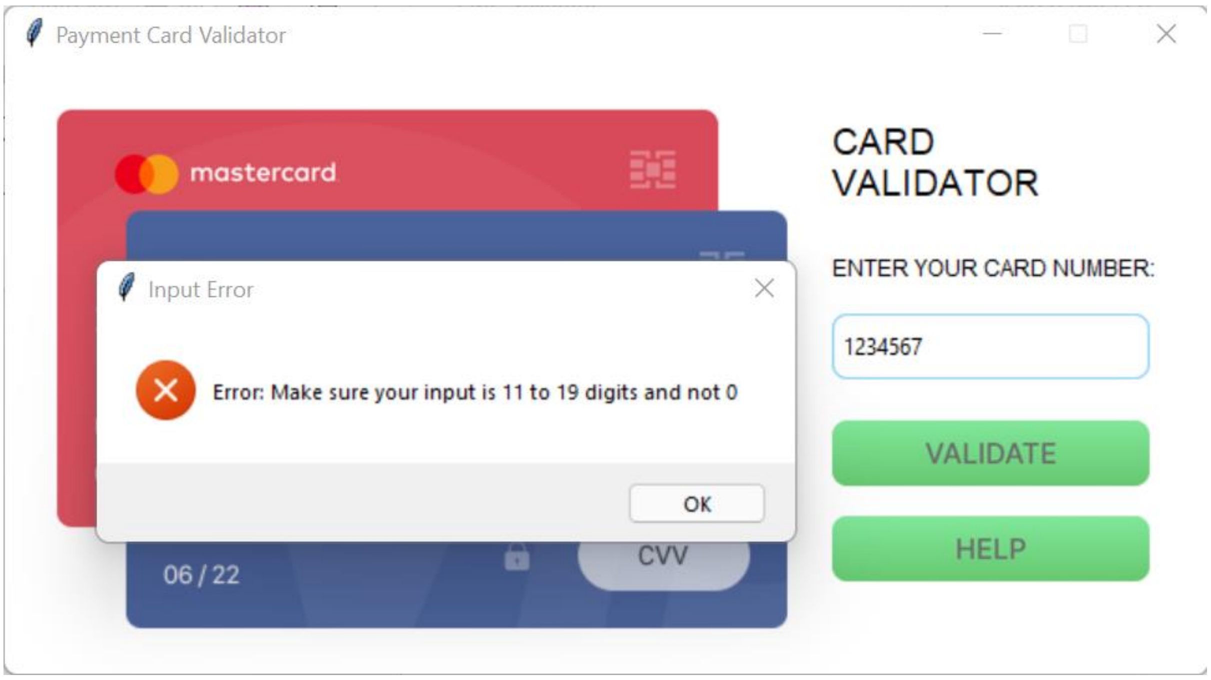
```
# GUI mode input isdigit validation
def check_input(inp):
    if inp.isdigit() and len(inp) <= 19:
        return True
    elif inp == "":
        return True
    else:
        return False

digit_validation = window.register(check_input)
entry_1.config(validate="key", validatecommand=(digit_validation, "%P"))

and:

# validate button
def input_validation():
    #print("Validate button clicked")
    if len(entry_1.get()) >= 11 and int(entry_1.get()) != 0:
          #A LOT OF CODE HERE PLEASE CHECK THE FILE
```

Tkinter code of the validation window:

```
# THIS IS NEW WINDOW WHEN VALIDATE BUTTON CLICKED

        validate_window = Toplevel()   # Tk() for main window Toplevel() for
subwindow
        validate_window.geometry("492x315") # initial window size before more info
button click 492x315
        validate_window.configure(bg = "#FFFFFF")
        validate_window.title("Card Validation")

        # More info button resizing subwindow
        def more_info_resizer():
            validate_window.geometry("492x396") # size change after clicking more
info button, default value 492x396
            more_info_btn.destroy() # deleting button after click

            # Making visible additional information
            subw_canvas.create_text(
                45.0,
                233.0,
                anchor="nw",
                text=f"CARD COUNTRY: {card_country}",
                fill="#060405",
                font=("Inter Regular", 12 * -1)
            )

            subw_canvas.create_text(
                45.0,
                259.0,
                anchor="nw",
                text=f"CURRENCY: {card_currency}",
                fill="#060405",
                font=("Inter Regular", 12 * -1)
            )

            subw_canvas.create_text(
                45.0,
                285.0,
                anchor="nw",
                text=f"ISSUING BANK: {card_bank}",
                fill="#060405",
                font=("Inter Bold", 12 * -1)
            )

            subw_canvas.create_text(
                45.0,
                310.0, # 312
                anchor="nw",
                text=f"BANK URL: {bank_url}",
                fill="#060405",
                font=("Inter Bold", 12 * -1)
            )

            subw_canvas.create_text(
                45.0,
                339.0,
                anchor="nw",
                text=f"BANK CONTACTS: {bank_phone}",
                fill="#060405",
                font=("Inter Bold", 12 * -1)
            )

        subw_canvas = Canvas(
            validate_window,
            bg = "#FFFFFF",
            height = 396,
            width = 492,
            bd = 0,
            highlightthickness = 0,
            relief = "ridge"
        )

        subw_canvas.place(x = 0, y = 0)
        more_info_btn_image = PhotoImage(
            file=relative_to_assets("more_info_btn.png"))
        more_info_btn = Button(validate_window,
            image=more_info_btn_image,
            borderwidth=0,
            highlightthickness=0,
            command=more_info_resizer,
            relief="flat"
        )
        more_info_btn.place(
            x=161.0,
            y=247.0,
            width=170.0,
            height=35.0
        )

        subw_canvas.create_text(
            45.0,
            102.0,
            anchor="nw",
            text=f"YOUR CARD:  {STATUS}",
            fill="#060405",
            font=("Inter Regular", 12 * -1)
        )

        subw_canvas.create_text(
            45.0,
            155.0,
            anchor="nw",
            text=f"IIN/BIN NUMBER: {CARD_BIN}",
            fill="#060405",
            font=("Inter Regular", 12 * -1)
        )

        subw_canvas.create_text(
            45.0,
            180.0,
            anchor="nw",
            text=f"CARD INDUSTRY:  {INDUSTRY}",
            fill="#060405",
            font=("Inter Regular", 12 * -1)
        )

        subw_canvas.create_text(
            45.0,
            207.0,
            anchor="nw",
            text=f"CARD BRAND: {BRAND}",
            fill="#060405",
            font=("Inter Regular", 12 * -1)
        )

        subw_canvas.create_text(
            45.0,
            31.0,
            anchor="nw",
            text="CARD \nVALIDATOR",
            fill="#060405",
            font=("Inter Black", 19 * -1)
        )

        image_decor_1 = PhotoImage(
            file=relative_to_assets("decor_1.png"))
        decor_1 = subw_canvas.create_image(
            336.0,
            21.0,
            image=image_decor_1
        )
```

```
        image_decor_2 = PhotoImage(
            file=relative_to_assets("decor_2.png"))
        decor_2 = subw_canvas.create_image(
            14.0,
            131.0,
            image=image_decor_2
        )
        validate_window.resizable(False, False)
        validate_window.mainloop()
```

In the code of the graphical mode, the methods of showing errors are also different, in contrast to the command line mode where errors were printed in the console, in the graphical mode all errors appear in pop-up windows:



Error: Make sure your input is 11 to 19 digits and not 0

1234567

The piece of code responsible for showing this error:

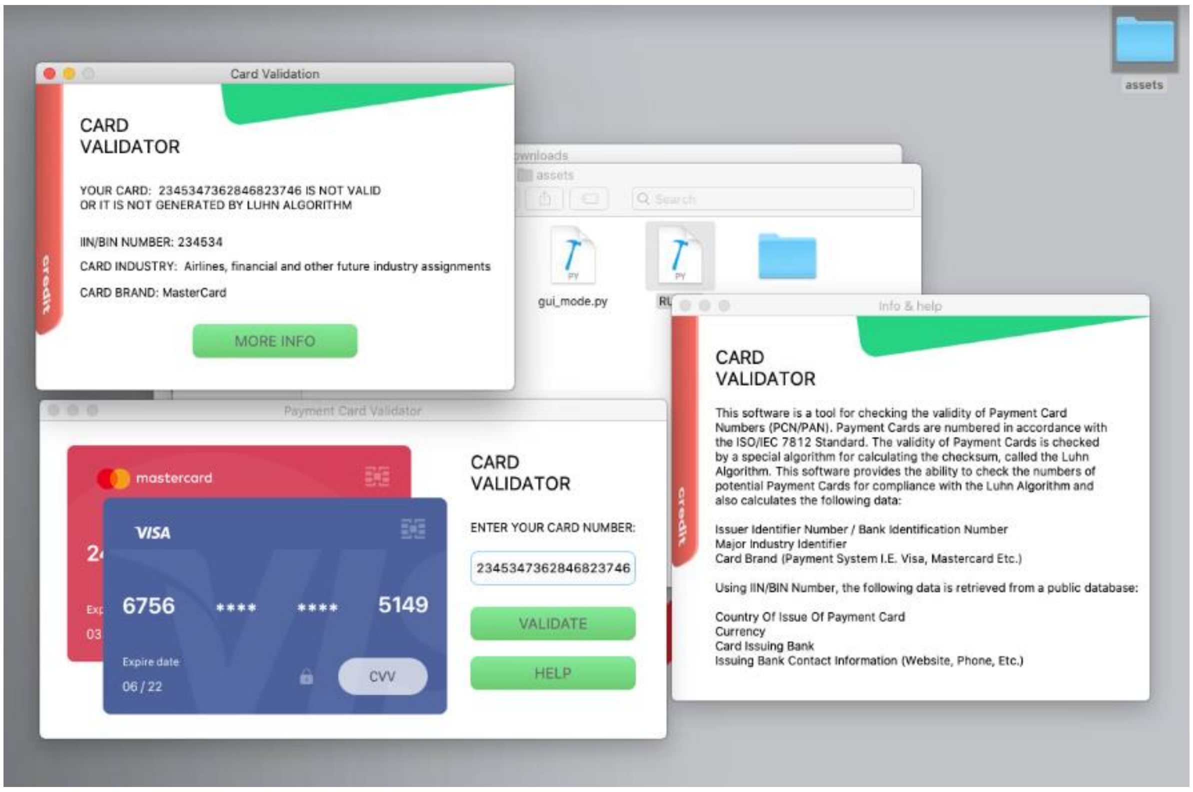
```
messagebox.showerror("Input Error", "Error: Make sure your input is 11 to 19
digits and not 0")
```

It is difficult to enumerate all the subtleties of the work of one or another part of the program in the graphical interface mode because of the abundance of code responsible for the design and arrangement of elements, but the user, having familiarized himself with the program code in the command line mode, easily navigates the other.

The performance of the program in both modes of operation has been tested and confirmed on the following operating systems:

Windows 11, Windows 10, Mac OS (Catalina), and Linux (Arch based Manjaro).

This is how the program interface looks like on Mac OS Catalina:



CARD
VALIDATOR

YOUR CARD: 2345347362846823746 IS NOT VALID
OR IT IS NOT GENERATED BY LUHN ALGORITHM

IIN/BIN NUMBER: 234534

CARD INDUSTRY: Airlines, financial and other future industry assignments

CARD BRAND: MasterCard

CARD
VALIDATOR

ENTER YOUR CARD NUMBER:

2345347362846823746

CARD
VALIDATOR

This software is a tool for checking the validity of Payment Card Numbers (PCN/PAN). Payment Cards are numbered in accordance with the ISO/IEC 7812 Standard. The validity of Payment Cards is checked by a special algorithm for calculating the checksum, called the Luhn Algorithm. This software provides the ability to check the numbers of potential Payment Cards for compliance with the Luhn Algorithm and also calculates the following data:

Issuer Identifier Number / Bank Identification Number
Major Industry Identifier
Card Brand (Payment System I.E. Visa, Mastercard Etc.)

Using IIN/BIN Number, the following data is retrieved from a public database:

Country Of Issue Of Payment Card
Currency
Card Issuing Bank
Issuing Bank Contact Information (Website, Phone, Etc.)

Below are some valid credit card numbers to check if the program is working:

4890494730164233

4255520191430892

5599002003494640

4797693370340565

Please note that validating credit card numbers does not guarantee the functionality of the cards themselves. Validation is just a method of checking whether such a card can exist or is just a bunch of numbers. This program does not make any requests to the issuing bank or payment system.