A Minor Project Report on

“ ATM SITMULATION MACHINE”

SESSION 2021-2024

****

INTERNATIONAL INSTITUTE OF TECHNOLOGY AND MANAGEMENT

AFFILIATED TO DCRUST, MURTHAL

SONIPAT-131001

HARYANA

**Under the Supervision of: Submitted by:**

Ms. Preeti Name: Priyanka Devra

Roll No:21015041069

Name: Ritu Kumari Gupta

Roll No:2105041069

Name: Sonia

Roll No:21015041100

Name: Shalu

Roll No:21015041093

# ACKNOWDLEGMENT

I take this opportunity to express my gratitude and respect to all those who helped me throughout my training period on the project. This project would not have been possible without the valuable guidance of my course instructor “Ms. Preeti " who taught me the basic concepts of Python and Visual Code handling whose guidance throughout the entire work enabled me to complete this project successfully. I owe my regards to the entire faculty of the department of BCA at IITM from where I have learnt the basics of computer and I express my sincere thanks to all my fellow course mates who supported me in my project through various informal discussions which were very valuable to the successful completion of my project.

# DECELERATION

I do, hereby declare that the interactive "ATM STIMULATION SYSTEM " application is an authentic wok developed by me at IITM affiliated by DCRUST university under the guidance of “Ms. Preeti” submitted as a partial fulfillment for the degree of Bachelor of Computer Application.

I am highly obliged to IITM, MURTHAL ( Haryana)

I also declare that, any or all contents incorporated in this dissertation have not been submitted in any form for the award of any degree or diploma of any other institution or university.

# CERTIFICATE

This is to clarify that the project / dissertation entitled ............................................................ is a Bonafide work done by Ms. ....................... (BCA Roll No. ..............................) in partial fulfillment of fulltime BCA course. Project has been carried out under my supervision and guidance. She has submitted the report in time. She has done good work and has fulfilled all the requirements.

Signature of Student Internal Guide

# Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Particulars** | **Page No.** | **Signature** |
| **1.** | Introduction | 6 |  |
| **2.** | Abstract | 7 |  |
| **3.** | Objective and Purpose | 8 |  |
| **4.** | Application | 9 |  |
| **5.** | Introduction to programming environment.   * Python * Visual Code | 10-14 |  |
| **6.** | Hardware and Software requirements | 15 |  |
| **7.** | Flowchart | 16 |  |
| **8.** | DFD | 17 |  |
| **9.** | Problem Definition | 18 |  |
| **10.** | Implementation (Coding) | 19-31 |  |
| **11.** | Results | 32-39 |  |
| **12.** | System Implementation | 40-41 |  |
| **13.** | Conclusion | 42 |  |
| **14.** | Future Scope | 43 |  |
| **15.** | Bibliography | 44 |  |

# INTRODUCTION

The aim of ATM Simulation System project is to build a Python based ATM (Automated Teller Machine) Simulation System. The Introduction of ATM by various banks have brought about freedom from the interminable queues in front of withdrawal counters at banks. This ATM Simulation System requires the constant updating of records between the bank servers and a spread out network of ATM. Security is the foundation of a good ATM system. This system will provide for secure authenticated connections between users and the bank servers. The whole process will be automated right from PIN (Personal Identification Number) validation to transaction completion ATM Simulation System will enable two important features of an ATM, reduction of human error in the banking system and the possibility of 24 hours personal banking. The card details and PIN database will be a secure module that will not be open to routine maintenance, the only possibility of access to this database will be through queries raised from an ATM in the presence of valid bank ATM card.

# ABSTRACT

The ATM is used by customers of a bank. Each customer has two accounts, a checking account and a savings account. Each customer has a customer number and a Personal dentification Number (PIN). Both must by typed into the simulation to gain access to the accounts. Once they have gained access, the customers can select an account (checking or savings). The balance of the selected account is displayed (initially zero). Then the customers can deposit and withdraw money and the balance will be updated accordingly. The application terminates when the user selects exit rather than an account. Since this isa simulation, the ATM does not actually communicate with the bank. It simply loads a list of customer numbers and PINs from a data file. The data file is maintained externally to this application. This Application should interact with the user via a simple swing GUI.

ATM Simulator project is written in Python. The project file contains a python script (atm.py). This is a simple console based system which is very easy to use. Talking about the system, it contains various functions which include Account Statement, Withdrawing, Depositing amount and changing the pin. Here, at first the user has to enter an existing username, when the username matches the system proceed toward the next procedure i.e. asking pin number. When a user passes all these sign-in procedures, he/she can use all those features. It is too easy to use, he/she can check their respective account statements.

# OBJECTIVE AND PURPOSE

An automated teller machine (ATM) is a specialized computer that allows you to complete bank transactions without the need of a bank representative. Many ATMs are conveniently accessible any time of day or night and can be used for everything from withdrawing or depositing money to checking your account balance to transferring money between accounts

**PURPOSE:**

An ATM is a machine that allows you to withdraw money, deposit cash or checks, view your balance or transfer money between accounts.

Many ATMs are accessible around the clock and eliminate the need to see a bank teller for transactions.

1. It pays to avoid ATM fees by only using ATMs in your bank's network.
2. A data interface with two input signals and four outputs is all that an automated teller machine is.
3. The processor communicates with these devices.
4. The ATM's processor is its beating heart.
5. A unified database system underpins all ATMs in operation around the world.

# APPLICATION OF ATM MACHINE

Automated Teller Machines (ATMs) are mainly of two types. One is a simple basic unit that allows you to withdraw cash, check your balance, change the PIN, get mini statements and receive account updates. The more complex units provide facilities for cash or cheque deposits and line of credit & bill payments.

There are also onsite and offsite Automated Teller Machines- the onsite ATMs are within the bank premises, unlike the offsite ones which are present in different nooks and corners of the country to assure that people have basic banking facilities and instant cash withdrawals if they can’t go to a bank branch.

ATMs can also be categorized based on the labels assigned to them. Some of these labels are listed below-

* Green Label ATMs- Used for agricultural purposes
* Yellow Label ATMs- Used for e-commerce transactions
* Orange Label ATMs- Used for share transactions
* Pink Label ATMs- Specifically for females to help avoid the long queues and waiting time
* White Label ATMs – Introduced by the TATA group, white label ATMs are not owned by a particular bank but by entities other than the bank
* Brown Label Banks- Operated by a third party other than a bank.

# INTRODUCTION TO PROGRAMMING ENVIRONMENT

The project is developed by Python.

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

What can Python do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.

Good to know

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as NetBeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Syntax compared to other programming languages

Python was designed for readability, and has some similarities to the English language with influence from mathematics.

Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

# VISUAL STUDIO CODE

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.

First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment, and more time executing on your ideas

Key feature of visual code:

1. **Available for macOS, Linux, and Windows**

Visual Studio Code supports macOS, Linux, and Windows - so you can hit the ground running, no matter the platform.

1. **Edit, build, and debug with ease.**

At its heart, Visual Studio Code features a lighting fast source code editor, perfect for day-to-day use. With support for hundreds of languages. VS code helps you be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more.

1. **Make it your own**

Customize every feature to your liking and install any number of third-party extensions. While most scenarios work "out of the box" with no configuration, VS Code also grows with you, and we encourage you to optimize your experience to suit your unique needs

1. **Built with love for the Web**

VS Code includes enriched built-in support for Node.js development with JavaScript and TypeScript, powered by the same underlying technologies that drive Visual Studio.

1. **Robust and extensible architecture**

Architecturally, Visual Studio Code combines the best of web, native, and language-specific technologies.

1. **Ready, set, code**

If you prefer a code editor-centric development tool or are building cross-platform web and cloud applications, we invite you to try out Visual Studio Code.

# ADVANTAGES OF VISUAL STUDIO CODE

* Excellent support for multiple programming languages: Visual Studio Code offers great support for a wide range of programming languages, including Java, Python, C++, JavaScript, and more.
* A wide range of features: Visual Studio Code offers many features that make it a powerful code editor.
* High customizability: Visual Studio Code is highly customizable, allowing you to configure the interface and keyboard shortcuts to your liking.
* Large community of developers: Visual Studio Code has a large community of developers who create and maintain extensions and plugins that add new functionality to the editor.
* Fast and efficient: Visual Studio Code is fast and efficient, with a small footprint. This makes it a great choice for developers who want a code editor that won’t slow down their computer.

# HARDWARE AND SOFTWARE REQUIREMENT

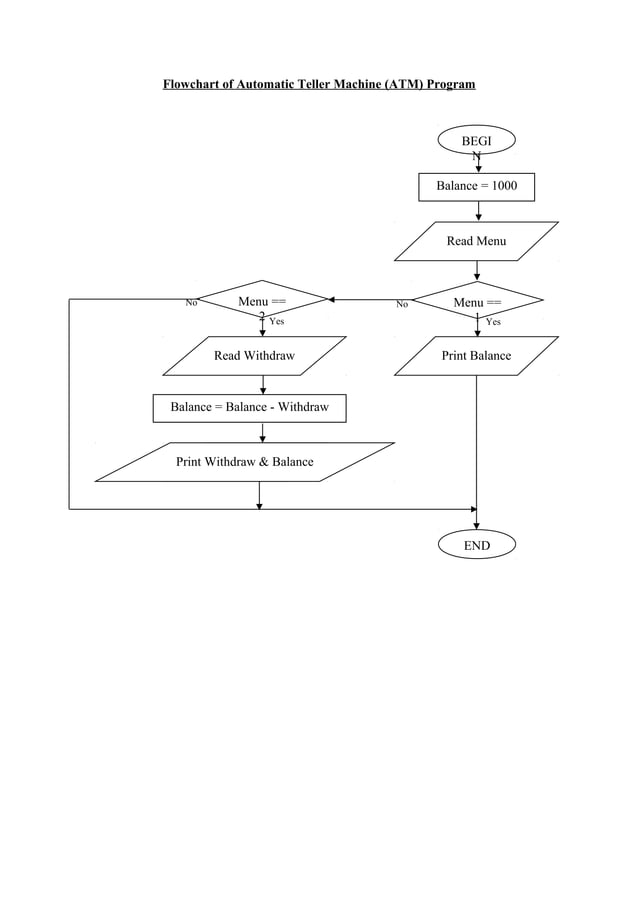
Minium Software Requirement

|  |  |
| --- | --- |
| Operating System | Window 11 |
| Front-End tools | Visual Studio Code |
| Back-End Tools | Python |

Minium Hardware Requirement

|  |  |
| --- | --- |
| Processor | 11th Gen Intel(R) Core (TM) i3-1115G4 @ 3.00GHz 2.19 GHz |
| Hard Disk | 22.3 GB Free Space |
| RAM | 8 GB |

# FLOW CHART



# DATA FLOW DIAGRAM

# PROBLEM DEFINITION

The source code declared above for the program of ATM Machine has been tested and it has been found that the above source code is okay and correct. the program involves many type of conversions. these conversions has to done carefully.

mainly there are two types of testing:

1.System testing

2.Intergration testing system testing involves whole testing of program at once and integration testing involves the breaking of program into modules & then test.

ALGORITHM/ STEP (Explain the different modules/functions used)

1.We have run the program in python IDLE

2.Our project is based on ATM Machine

3.The functions of ATM Machine are:

* Enter the correct user name and password to login
* Lodgement : This function helps us to deposit the money
* Withdrawal: Through this function we can take our money from the machine
* Statement: We get the mini statement of our account
* Pin Change: Through this function we can change the pin of our credit card.
* Quit: Function helps us to terminate the process.

# IMPLEMENTATION

print("""

       \*       \* \* \* \* \* \*     \*         \*        \*         \*       \*         \* \* \* \*    \*       \*    \* \* \*    \*        \*   \* \* \* \* \*

      \* \*           \*          \* \*     \* \*        \* \*     \* \*      \* \*       \*           \*       \*      \*      \* \*      \*   \*

     \*   \*          \*          \*  \*   \*  \*        \*  \*   \*  \*     \*   \*      \*           \*       \*      \*      \*  \*     \*   \*

    \*     \*         \*          \*   \* \*   \*        \*   \* \*   \*    \*     \*     \*           \* \* \* \* \*      \*      \*   \*    \*   \* \* \* \* \*

   \* \* \* \* \*        \*          \*    \*    \*        \*    \*    \*   \* \* \* \* \*    \*           \*       \*      \*      \*    \*   \*   \*

  \*         \*       \*          \*         \*        \*         \*  \*         \*   \*           \*       \*      \*      \*     \*  \*   \*

 \*           \*      \*          \*         \*        \*         \* \*           \*   \* \* \* \*    \*       \*    \* \* \*    \*      \* \*   \* \* \* \* \*

                                                                                                                                                               """)

try:

    # read the 'Accounts.txt' file

    # if you try to open non existing file in read mode, this will throw an error

    f = open('Accounts.txt', 'r')

    f.close()

except FileNotFoundError:

    # if 'Accounts.txt' file is not found, create it

    f = open('Accounts.txt', 'w')

    f.close()

'-------------------------------------------------'

import os

import time

import tkinter as tk

import tkinter.ttk as ttk

from tkinter import messagebox

def clear\_screen():

    # This function clear the output of the screen

    os.system('clear')

    print()  # print blank line after clearing the screen

def read\_file(file\_name):

    opened\_file = open(file\_name, 'r')

    lines\_list = []

    for line in opened\_file:

        line = line.split()

        lines\_list.append(line)

    # print(lines\_list)

    return lines\_list

def print\_process(process):

    date = '{}'.format(process[2:7])

    print('{0}\t{1}\t{2}{3: ^10} {4: ^10}'.format(

        process[0],

        process[1].center(len('change\_password')),

        date.center(len(date)),

        process[7],

        process[8]

    )

    )

def withdraw(ls, acc\_list):

    # ls is a list of the information of the account

    # ls[0] id

    # ls[1] name

    # ls[2] password

    # ls[3] balance

    def witClicked(current\_balance, ls):

        withdraw\_amount = int(witAmountEntry.get())

        if withdraw\_amount > current\_balance:

            messagebox.showerror("withdraw","You can't withdraw more than your current balance")

        else:

            current\_balance -= abs(withdraw\_amount)  # to guarantee the entered value

        file\_name = ls[0] + '.txt'

        process\_list = read\_file(file\_name)

        id\_file = open(file\_name, 'a')

        if len(process\_list) == 0:

            # if there are no processes in the file

            last\_id = 1

        else:

            last\_id = int(process\_list[len(process\_list) - 1][0]) + 1

            # get last id and increment it

        id\_file.write(

            '{0}\twithdraw\t\t\t{1}\t{2}\t{3}\n'.format(str(last\_id), str(time.ctime()), ls[3], str(current\_balance)))

        id\_file.close()

        ls[3] = str(current\_balance)

        bal = "Your current balance: " + ls[3]

        tk.Label(witTop, text=bal, font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=3,columnspan=2, padx=(0, 10), pady=(10, 10))

        acc\_file = open('Accounts.txt', 'w')

            # after logging out of the account

            # write changes to accounts.txt file

        for acc in acc\_list:

            for elements in acc:

                acc\_file.write("%s\t" % elements)

            acc\_file.write('\n')

        return ls

    witTop = tk.Toplevel()

    center\_window((300,250), witTop)

    current\_balance = int(ls[3])

    cBalance = "Current Balance: "+ls[3]

    tk.Label(witTop, text=cBalance, font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=0, pady=(10,10), padx=(10,10))

    tk.Label(witTop, text="Withdraw Amount ", font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=1, pady=(10,10), padx=(10,10))

    witAmountEntry = tk.Entry(witTop, font=("ubuntu", 12), fg="red" ,width = 5)

    witAmountEntry.grid(column=1, row=1, padx=(10,10), pady=(10,10))

    tk.Button(witTop, text="Withdraw", font=("ubuntu", 12), fg="white", bg="#4267B2",

              command=lambda: witClicked(current\_balance, ls)).grid(column=0, row=2, columnspan=2)

def deposit(ls, acc\_list):

    # ls is a list of the information of the account

    # ls elements are of type string

    # ls[0] id

    # ls[1] name

    # ls[2] password

    # ls[3] balance

    def confirmClicked(current\_balance, ls):

        deposit\_amount = int(depositAmountEntry.get())

        current\_balance += abs(deposit\_amount)  # to guarantee the entered value

        file\_name = ls[0] + '.txt'

        process\_list = read\_file(file\_name)

        id\_file = open(file\_name, 'a')

        if len(process\_list) == 0:  # if there are no processes in the file

            last\_id = 1

        else:

            last\_id = int(process\_list[len(process\_list) - 1][0]) + 1  # get last id and increment it

        id\_file.write(

            '{0}\tdeposit\t\t\t\t{1}\t{2}\t{3}\n'.format(str(last\_id), str(time.ctime()), ls[3], str(current\_balance)))

        id\_file.close()

        ls[3] = str(current\_balance)

        bal = "Your current balance: " + ls[3]

        tk.Label(depositTop, text=bal, font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=3,columnspan=2, padx=(0, 10), pady=(10, 10))

        acc\_file = open('Accounts.txt', 'w')

            # after logging out of the account

            # write changes to accounts.txt file

        for acc in acc\_list:

            for elements in acc:

                acc\_file.write("%s\t" % elements)

            acc\_file.write('\n')

        return ls

    depositTop = tk.Toplevel()

    center\_window((300,250), depositTop)

    current\_balance = int(ls[3])  # make changes to another variable to keep the previous balance

    # to print it later, then save ls[3] = current\_balance

    cBalance = 'Current balance: ' + ls[3]

    tk.Label(depositTop, text=cBalance, font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=0, pady=(10,10), padx=(10,10))

    tk.Label(depositTop, text="Deposit Amount ", font=("ubuntu", 12), fg="#4267B2").grid(column=0, row=1, pady=(10,10), padx=(10,10))

    depositAmountEntry = tk.Entry(depositTop, font=("ubuntu", 12), fg="red" ,width = 5)

    depositAmountEntry.grid(column=1, row=1, padx=(10,10), pady=(10,10))

    tk.Button(depositTop, text="Confirm", font=("ubuntu", 12), fg="white", bg="#4267B2",

              command=lambda: confirmClicked(current\_balance, ls)).grid(column=0, row=2, columnspan=2)

def show\_history(ls):

    # ls is the list contains account data

    # ls[0] id

    # ls[1] name

    # ls[2] password

    # ls[3] balance

    hisTop = tk.Toplevel()

    label = tk.Label(hisTop, text="History", font=("ubuntu", 33), bg="#4267B2", fg="#FFFFFF", width=100)

    label.place(relx=0.5, rely=0.08, anchor="center")

    # choice = int(input('1) show deposit processes\n2) show withdraw processes\nchoice>> '))

    #

    file\_name = ls[0] + '.txt'

    id\_list = read\_file(file\_name)

    top\_line = '\nID\t   ' + 'Type'.center(len('change\_password')+4) + 'Date and Time'.center(40+7) + 'before'.center(

        15) + 'after'.center(20)

    # print(top\_line)

    linet = ('-' \* (len(top\_line)+10))

    depline=""

    tk.Label(hisTop, text=top\_line, font=("ubuntu", 10), fg="#4267B2").grid(column=0, row=0, pady=(60, 0))

    tk.Label(hisTop, text=linet, font=("ubuntu", 10), fg="#4267B2").grid(column=0, row=1)

    for line in id\_list:

        date = '{}'.format(line[2:7])

        hline = ('{0}\t{1}\t{2}{3: ^10} {4: ^10}'.format(line[0],line[1].center(len('change\_password')), date.center(len(date)),line[7],line[8]))

        depline = depline + "\n\n "+hline

    tk.Label(hisTop, text=depline, font=("ubuntu", 10), fg="#4267B2").grid(column=0, row=2, pady=(0,20))

def login(acc\_list, loginWindow, LoginButton):

    def loginButtonClicked():

        Id = entryid.get()

        password = entryPass.get()

        login\_id = Id

        login\_password = password

        found = False

        for account in acc\_list:

            if account[0] == login\_id and account[2] == login\_password:

                found = True

                LoginButton['text'] = "Log Out"

                menu2(account, acc\_list, loginWindow)

                break

            else:

                continue

        if not found:

            messagebox.showerror("ATM Machine", "Wrong ID or Password")

        else:

            acc\_file = open('Accounts.txt', 'w')

            print('Saving changes...')

            # after logging out of the account

            # write changes to accounts.txt file

            for acc in acc\_list:

                for elements in acc:

                    acc\_file.write("%s\t" % elements)

                acc\_file.write('\n')

    frame = tk.Frame(loginWindow, bg="#4267B2")

    labelId = tk.Label(frame, text="Enter ID", font=("ubuntu", 14), bg="#4267B2", fg="white")

    labelPass = tk.Label(frame, text="Password", font=("ubuntu", 14), bg="#4267B2", fg="white")

    entryid = tk.Entry(frame, font=("ubuntu", 14), fg="#4267B2", width=8)

    entryPass = tk.Entry(frame, font=("ubuntu", 14), fg="#4267B2", width=15)

    loginButton = tk.Button(frame, text="login", font=("ubuntu", 14), bg="white", fg="#4267B2",

                            command=loginButtonClicked)

    labelId.grid(column=0, row=1, padx=(10, 10), pady=(10, 10))

    labelPass.grid(column=0, row=2, padx=(10, 10), pady=(10, 10))

    entryid.grid(column=1, row=1, padx=(10, 10), pady=(10, 10))

    entryPass.grid(column=1, row=2, padx=(10, 10), pady=(10, 10))

    loginButton.grid(column=1, row=3, columnspan=2, padx=(10, 10), pady=(10, 10))

    frame.grid(column=0, row=0, padx=(0, 0), pady=(60, 0))

def create\_account(ls, CreatedWindow):

    # ls is a list of lists of lines in accounts file

    # ls is the accounts\_list

    def createdClicked():

        name = entryName.get()

        password = entryPass.get()

        account\_name = name

        account\_password = password

        accounts\_file = open('Accounts.txt', 'a')

        if len(ls) == 0:

            new\_last\_id = 1

        else:

            new\_last\_id = int(ls[len(ls) - 1][0]) + 1

        line = '{0}\t{1}\t{2}\t0\n'.format(str(new\_last\_id), account\_name, account\_password)

        accounts\_file.write(line)

        id\_file\_name = str(new\_last\_id) + '.txt'

        id\_file = open(id\_file\_name, 'w')

        accountConf = ("Your Account Has Been Created \n Your Id is")

        tk.Label(CreatedWindow, text=accountConf, font=("ubuntu", 12), bg="#D9D9D9", fg="#4267B2").grid(column=0, row=1,

                                                                                                        padx=(20, 20),

                                                                                                        pady=(10, 10),

                                                                                                        columnspan=2)

        tk.Label(CreatedWindow, text=str(new\_last\_id), font=("ubuntu", 40), bg="#D9D9D9", fg="#4267B2").grid(column=0,

                                                                                                             row=2,

                                                                                                             padx=(

                                                                                                             20, 20),

                                                                                                             pady=(

                                                                                                             10, 10),

                                                                                                             columnspan=2)

        id\_file.close()

        accounts\_file.close()

        ls.append([str(new\_last\_id), account\_name, account\_password, '0'])

    frame = tk.Frame(CreatedWindow, bg="#4267B2")

    labelName = tk.Label(frame, text="Enter Name", font=("ubuntu", 14), bg="#4267B2", fg="white")

    labelPass = tk.Label(frame, text="Password", font=("ubuntu", 14), bg="#4267B2", fg="white")

    entryName = tk.Entry(frame, font=("ubuntu", 14), fg="#4267B2", width=11)

    entryPass = tk.Entry(frame, font=("ubuntu", 14), fg="#4267B2", width=13)

    CreatedButton = tk.Button(frame, text="Create Account", font=("ubuntu", 14), bg="white", fg="#4267B2",

                              command=createdClicked)

    labelName.grid(column=0, row=1, padx=(10, 10), pady=(10, 10))

    labelPass.grid(column=0, row=2, padx=(10, 10), pady=(10, 10))

    entryName.grid(column=1, row=1, padx=(10, 10), pady=(10, 10))

    entryPass.grid(column=1, row=2, padx=(10, 20), pady=(10, 10))

    CreatedButton.grid(column=0, row=3, columnspan=2, padx=(10, 10), pady=(10, 10))

    frame.grid(column=0, row=0, padx=(0, 0), pady=(60, 0))

def menu2(account,acc\_list, loginWindow):

    # account is the list of single logined user

    # account[0] id

    # account[1] name

    # account[2] password

    # account[3] balance

    def infoClicked():

        infoTop = tk.Toplevel(loginWindow)

        info = ("ID: {}\nName: {}\nBalance: {}\n".format(account[0], account[1], account[3]))

        tk.Label(infoTop, text=info, font=("ubuntu", 14), bg="#D9D9D9", fg="#4267B2").grid(column=0, row=0)

        center\_window((200, 100), infoTop)

    def hisClicked():

        show\_history(account)

    def depClicked():

        deposit(account, acc\_list)

    def witClicked():

        withdraw(account, acc\_list)

    wel = ("\nHello, {0}".format(account[1]))

    manu2Frame = tk.Frame(loginWindow, bg="white")

    labelWelcome = tk.Label(loginWindow, text=wel, font=("ubuntu", 20), bg="#D9D9D9", fg="#4267B2")

    labelWelcome.grid(column=0, row=1)

    tk.Button(manu2Frame, text="InFo", font=("ubuntu", 12), bg="#4267B2", fg="white", width=8, height=1,

              command=infoClicked).grid(column=0, row=0)

    tk.Button(manu2Frame, text="History", font=("ubuntu", 12), bg="#4267B2", fg="white", width=8, height=1,

              command=hisClicked).grid(column=0, row=1)

    tk.Button(manu2Frame, text="Deposit", font=("ubuntu", 12), bg="#4267B2", fg="white", width=8, height=1,

              command=depClicked).grid(column=0, row=2)

    tk.Button(manu2Frame, text="Withdraw", font=("ubuntu", 12), bg="#4267B2", fg="white", width=8, height=1,

              command=witClicked).grid(column=0, row=3)

    manu2Frame.grid(column=0, row=2, padx=(0, 0), pady=(0, 0))

accounts\_list = read\_file('Accounts.txt')

# GUI functions

def splash():

    root = tk.Tk()

    # show no frame

    root.overrideredirect(True)

    width = root.winfo\_screenwidth()

    height = root.winfo\_screenheight()

    root.geometry('%dx%d+%d+%d' % (width \* 0.8, height \* 0.8, width \* 0.1, height \* 0.1))

    # take a .jpg picture you like, add text with a program like PhotoFiltre

    # (free from http://www.photofiltre.com) and save as a .gif image file

    image\_file = "image.png"

    # assert os.path.exists(image\_file)

    # use Tkinter's PhotoImage for .gif files

    image = tk.PhotoImage(file=image\_file)

    canvas = tk.Canvas(root, height=height \* 0.8, width=width \* 0.8, bg="#CC5320")

    canvas.create\_image(width \* 0.8 / 2, height \* 0.8 / 2, image=image)

    canvas.pack()

    # show the splash screen for 5000 milliseconds then destroy

    root.after(3000, root.destroy)

    root.mainloop()

    return 0

def center\_window(size, window):

    window\_width = size[

        0]  # Fetches the width you gave as arg. Alternatively window.winfo\_width can be used if width is not to be fixed by you.

    window\_height = size[

        1]  # Fetches the height you gave as arg. Alternatively window.winfo\_height can be used if height is not to be fixed by you.

    window\_x = int(

        (window.winfo\_screenwidth() / 2) - (window\_width / 2))  # Calculates the x for the window to be in the centre

    window\_y = int(

        (window.winfo\_screenheight() / 2) - (window\_height / 2))  # Calculates the y for the window to be in the centre

    window\_geometry = str(window\_width) + 'x' + str(window\_height) + '+' + str(window\_x) + '+' + str(

        window\_y)  # Creates a geometric string argument

    window.geometry(window\_geometry)  # Sets the geometry accordingly.

    return

def mainWindow():

    def loginClicked():

        window.destroy()

        loginWindow = tk.Tk()

        height = 500

        width = 300

        center\_window((width, height), loginWindow)

        def backLoginClicked(\*args):

            loginWindow.destroy()

            mainWindow()

        label = tk.Label(loginWindow, text="ATM Machine", font=("ubuntu", 33), bg="#4267B2", fg="#FFFFFF", width=100)

        label.place(relx=0.5, rely=0.05, anchor="center")

        backLogout = "Back"

        LoginButton = tk.Button(loginWindow, text=backLogout, font=("ubuntu", 33), bg="#4267B2", width=100, fg="#FFFFFF",

                                command=backLoginClicked)

        LoginButton.place(relx=0.5, rely=0.95, anchor="center")

        login(accounts\_list, loginWindow, LoginButton)

        loginWindow.mainloop()

    def createAccountClicked():

        window.destroy()

        CreatedWindow = tk.Tk()

        height = 500

        width = 300

        center\_window((width, height), CreatedWindow)

        def backCreatedClicked():

            CreatedWindow.destroy()

            mainWindow()

        label = tk.Label(CreatedWindow, text="ATM Machine", font=("ubuntu", 33), bg="#4267B2", fg="#FFFFFF", width=100)

        label.place(relx=0.5, rely=0.05, anchor="center")

        CreatedButton = tk.Button(CreatedWindow, text="Back", font=("ubuntu", 33), bg="#4267B2", width=100,

                                  fg="#FFFFFF", command=backCreatedClicked)

        CreatedButton.place(relx=0.5, rely=0.95, anchor="center")

        create\_account(accounts\_list, CreatedWindow)

        CreatedWindow.mainloop()

    def exitClicked():

        res = messagebox.askyesnocancel("ATM machine", "Do You want to EXIT!")

        if (res == True):

            os.system('clear')

            window.destroy()

        else:

            pass

    window = tk.Tk()

    height = 500

    width = 300

    center\_window((width, height), window)

    label = tk.Label(window, text="ATM Machine", font=("ubuntu", 33), bg="#4267B2", fg="#FFFFFF", width=100)

    label.place(relx=0.5, rely=0.05, anchor="center")

    loginButton = tk.Button(window, text="Login", font=("ubuntu", 33), bg="#4267B2", fg="#FFFFFF", command=loginClicked)

    loginButton.place(relx=0.5, rely=0.2, anchor="center")

    createAccountButton = tk.Button(window, text="Create\nAccount", font=("ubuntu", 33), bg="#4267B2",

                                    command=createAccountClicked, fg="#FFFFFF")

    createAccountButton.place(relx=0.5, rely=0.5, anchor="center")

    exitButton = tk.Button(window, text="Exit", font=("ubuntu", 33), bg="#4267B2", command=exitClicked, fg="#FFFFFF")

    exitButton.place(relx=0.5, rely=0.8, anchor="center")

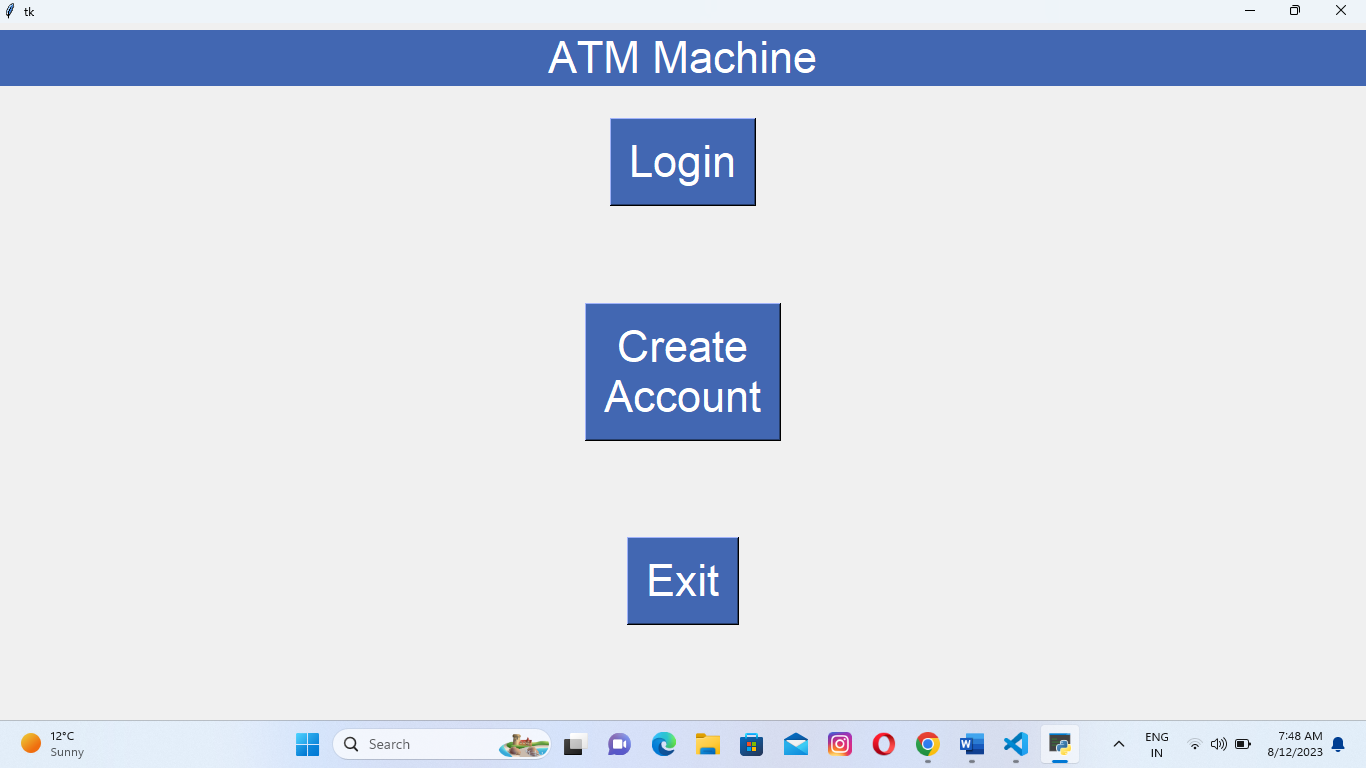
    window.mainloop()

splash()

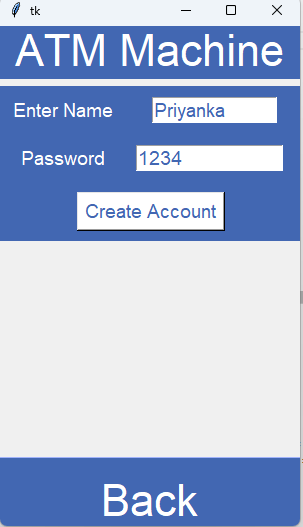
mainWindow()

# RESULTS

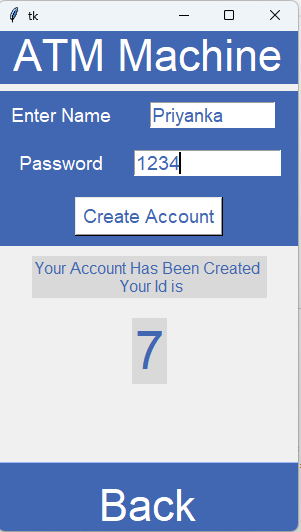
The following is the final output:

1. First we have to create the account.

# 2.Now we have to enter the name and create a password.

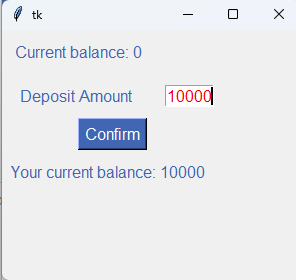


# 3.The system will provide ID.

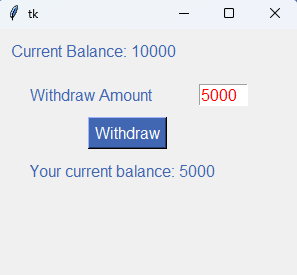


# 4.After enter the ID and password the system will login.

# 5.Deposit the amount.



# 6.Withdraw the amount.



# 7.History

# 8.INFORMATION



# SYSTEM IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system. The implementation phase constructs, installs and operates the new system. The most crucial stage in achieving a new successful system is that it will work efficiently and effectively. Implementation is a process of ensuring that the information system is operational.

It involves –

-Constructing a new system from scratch

-Constructing a new system from the existing one.

Implementation allows the users to take over its operation for use and evaluation. It involves training the users to handle the system and plan for a smooth conversion.

There are several activities involved while implementing a new project. They are

• End user training

• End user Education

• Training on the application software

• System Design

• Parallel Run and To New System

• Post implementation Review

**End user Training**: The successful implementation of the new system will purely upon the involvement of the officers working in that department. The officers will be imparted the necessary training on the new technology.

**End User Education**: The education of the end user start after the implementation and testing is over. When the system is found to be more difficult to understand and complex, more effort is put to educate the end used to make them aware of the system, giving them lectures about the new system and providing them necessary documents and materials about how the system can do this.

**Training of application software**: After providing the necessary basic training on the computer awareness, the users will have to be trained upon the new system such as the screen flows and screen design type of help on the screen, type of errors while entering the data, the corresponding validation check at each entry and the way to correct the data entered. It should then cover information needed by the specific user or group to use the system.

**Post Implementation View:** The department is planning a method to know the states of the past implementation process. For that regular meeting will be arranged by the concerned officers about the implementation problem and success.

**Software requirement analysis:** The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product.

**Objectives of the project:** The main objective of the ATM Stimulation Machine is to manage the details of Accounts, Deposit, Logins, Withdrawal, and have the facility of ID creation.

# CONCLUSION

The main complaint heard about ATM machines is that while they are convenient, they are expensive to use. However, if we look at it from a banking perspective, business is business. Regardless of what we think of ATM machines, there is no doubt that they have changed the world and the way in which we do things. For example, think how many times we have been out somewhere only to discover we have no cash and we are out of checks, ah, but in the corner, there is an ATM machine. In the blink of an eye, we swipe the card and now have cash on hand. In addition to pulling money out, the ATM machine also makes it convenient to deposit money. transfer money, and check balances. Best of all, to use an ATM machine, we do not have to go to the bank. We will find ATM machines at other banks, grocery stores, shopping malls, along the roadside.

# FUTURE SCOPE

The scope of ATM Stimulation Machine are:

* ATM Management system reduces the workload of the bank's staff.
* ATMs reduce the work pressure on the bank's staff and avoids queues in bank premises.
* Atm also reduces the time of customer which is possible through the ATM.

# BIBLIOGRAPHAY

* Teach yourself Python programming
* Software engineering
* Wikipedia, URL: http:/www.wikipedia.org.
* Answers.com, online dictionary, encyclopedia and much more
* Google
* Visual Studio Code.