**ABSTRACT**

This report elaborates on a simple python project as titled “Offline ATM Simulator”. By the means of the simulator, the user will be able to experience the features of a standard ATM machine (and more) virtually in a computer system. The project in its entirety is coded in Python 3.7.3, with no need of downloading external libraries. All the functions shall be invoked through the baseline Python library. The resultant system incorporates various python modules like OS, Time, their associated functions as well as a couple of user defined functions. Some features of the simulator include the ability to login, create an account, deposit, withdraw, change password etc. The system will be initiated with a simple user interface with a list of options awaiting user selection. The simulator has a total of 10 modules from which 2 menus offer distinct features of its own. File handling mechanics are integrated to store and retrieve user details. The details include the user’s name, password. It also contains additional information like process history, deposit process details, withdraw process details. All the aforementioned information can be displayed in conjunction by selecting a single option. Furthermore, the user can delete all process from the file and start from the scratch. For each and every user, particular files are created such that it contains information specific to the individual. The system is properly organized and equipped with a lot of features that one could exploit for personal use, or other kinds of experimentation.

**INTRODUCTION**

The ATM Simulator is a virtual environment that attempts to replicate a realistic ATM machine and further improvises on it as a solid package. It provides a simple user-interface offering a variety of features that can be found in an actual ATM machine. In fact, the simulator expands on the features of an ATM machine by including the ability for a new user to register themselves as a bank user. The other (common) features include the ability to login, create an account, deposit, withdraw, change password etc. Each customer has a customer number and a Personal Identification Number (PIN). Both must be typed into the simulation to gain access to the accounts. Once they have gained access, the customer can select an account (checking or savings). The balance of the selected account is displayed (initially zero). Then the customer 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 is a simulation, the ATM does not actually communicate with the bank. It simply loads a list of customer numbers and PINs from a data file. This data file is maintained externally to this application. This application interacts with the user through the means of a generic Python interface and is dedicated to provide a mirror like experience.

**AIM:** The purpose of this program is to simulate a simple Automatic Teller Machine (ATM)

**ALGORITHM:**

This is a generic ATM simulator and this system uses the concept of Files in Python. All of the modules implemented in this system use files to store all the data given by the user and the data manipulated by the system.

**FILE HANDLING IN PYTHON:**

Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

**PSEUDO CODE FOR THE SYSTEM:**

* File\_object = open(‘filename.txt’,’w’);
* File\_object.close()

The above two lines is used to read the data from the user and open the file to read and write the values. ‘w’ indicates the file is opened in the write format and ‘r’ indicates the files is opened in the read mode.

Set old\_password = ls[2] (Ask to old password to enter new one)

flag = gate\_x(old\_password) (Security flag get the output flag)

if (flag == '0')

set new\_password = input("\nEnter the new password: ")('''Get the new password''')

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)

set last\_id = 1

else:

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

id\_file.write(

'{0}\tchange\_password\t\t{1}\t{2}\t{3}\n'.format(str(last\_id), str(time.ctime()), old\_password, str(new\_password)))

(write process id type before after)

id\_file.close()

This is the pseudo code for changing the password of the user. This is implemented in the second menu which comes after the user logs in into their account.

if (len(ls) == 0)

set new\_last\_id = 1

else

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

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

This is the pseudo code for creating an account.

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)))

write-> process\_id process\_name process\_date\_and\_time before\_process after\_process

id\_file.close()

set ls[3] = str(current\_balance)

This is the pseudo code for deposition of money in the account.

login\_id = input('Please, Enter your info(press "Ctrl+C" to go back) \n>>ID: ')

login\_password = input('>>Password: ')

set found = False

for account in acc\_list:

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

* + set found = True
  + clear screen()
  + menu2(account)
  + break

else:

* + continue

This is the pseudo code for Login module.

**CHAPTER 2**

**SYSTEM ANALYISIS AND DESIGN**

**2.1. SOFTWARE USED**

The operating system used in the design of the application is Windows 10. The software used in building the application is Python 3.7.3. It is the third maintenance release of Python 3.7. The Python 3.7 series is the newest major release of the Python language and contains many new features and optimizations. Among the major new features in Python 3.7 are:

* PEP 539, new C API for thread-local storage
* PEP 545, Python documentation translations
* New documentation translations: Japanese, French, and Korean.
* PEP 552, Deterministic pyc files
* PEP 553, Built-in breakpoint()
* PEP 557, Data Classes
* PEP 560, Core support for typing module and generic types
* PEP 562, Customization of access to module attributes
* PEP 563, Postponed evaluation of annotations
* PEP 564, Time functions with nanosecond resolution
* PEP 565, Improved Deprecation Warning handling
* PEP 567, Context Variables
* Avoiding the use of ASCII as a default text encoding (PEP 538, legacy C locale coercion and PEP 540, forced UTF-8 runtime mode)
* The insertion-order preservation nature of dict objects is now an official part of the Python language spec.
* Notable performance improvements in many areas.

**2.3. HARDWARE USED**

* Processor: Intel® Core™ i5-7300HQ CPU @2.50 GHz
* Installed memory (RAM): 16.0 GB
* GPU: NVIDIA GeForce GTX 1050 Ti @4.0 GB

**2.4 ARCHITECTURE DIAGRAM**

****

Fig 2.4 Architecture Diagram

The Architecture diagram describes the system in a more simple and pictographic manner. The user can either start with providing credentials or creating an account. Then the system provides different functionalities for the user to perform in which the user can select any option. Finally after all the desired functions are performed, the user can logout and exit.

**2.5. SEQUENCE DIAGRAM**

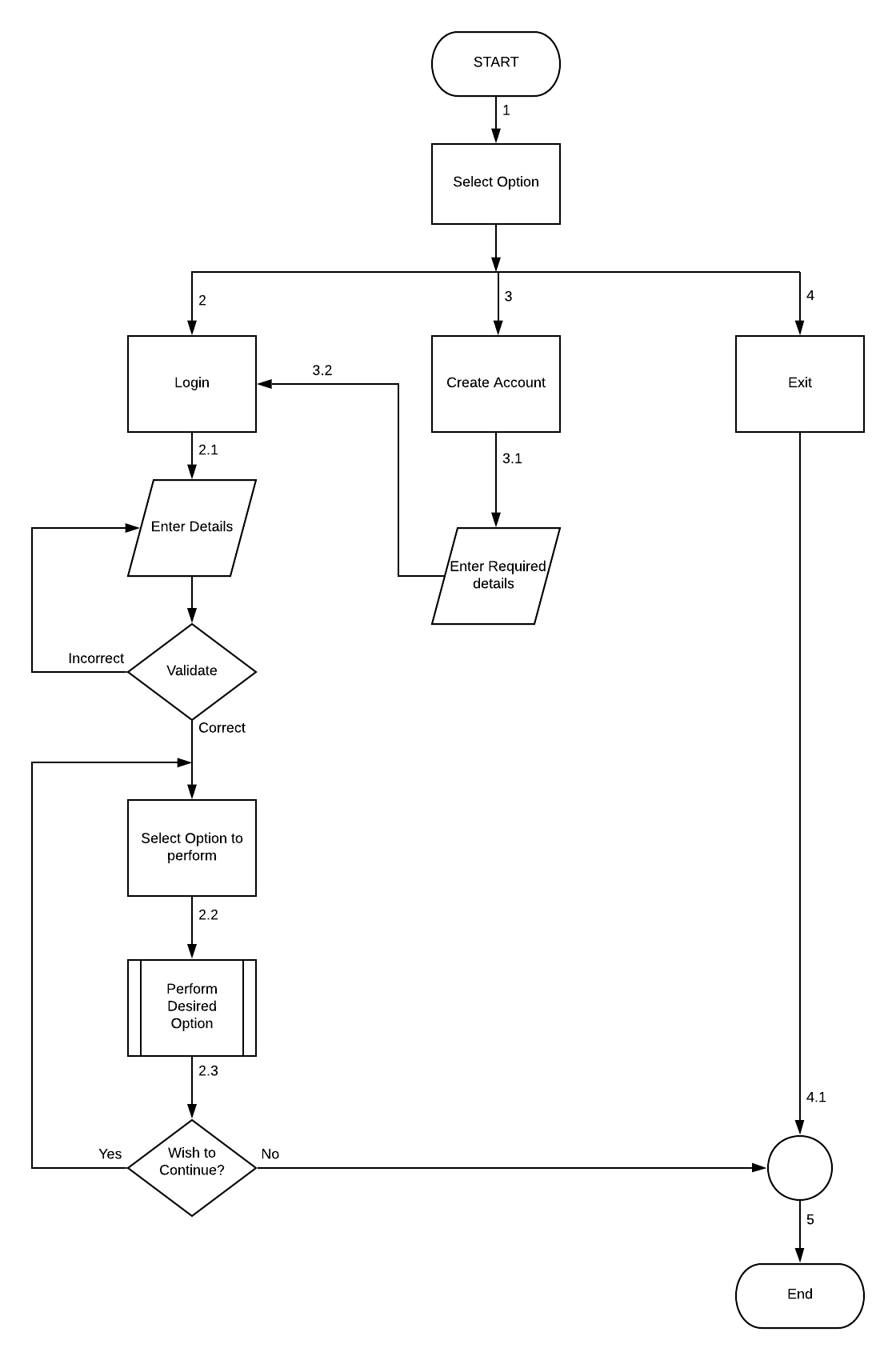
****

Fig 2.5 Sequence Diagram

**2.5.1. DESCRIPTION**

In the menu page 1, there are three options: login, create account and exit. Inside the login page, the user can enter their ID and password. If they are valid, the system approaches menu page 2. Otherwise, it prompts to enter ID and password again. The menu page 2 contains the following options: Show info, Show Process History, Deposit, Withdraw, Change Password and logout. If the user selects the “show info” option, all the current user information will be displayed. Inside show process, menu 3 is approached with the following options: show deposit processes, show withdraw processes, show all processes and clear processes. Inside option 1, the user can view their deposit process details. Inside option 2, the user can view their withdraw process details. Inside option 3, password details can be viewed.

Option 4 allows a user to view all the aforementioned details simultaneously. Finally, option 5 allows deletion of all the processes from the file. Reverting back to menu page 2, the option 3 (deposit) asks the user the amount to be entered and thus updates the balance accordingly. Under change password, the User is asked to enter the old password and update the new password. This change is displayed in the “change password” process. The last option is Logout which takes the user back to menu 1. Under the create account option, the user is asked to provide name and password. The last option in menu 1 is exit. It kills the program and exits the python shell.

**CHAPTER 3**

**SYSTEM IMPLEMENTATION**

**3.1. PLATFORM USED**

The platform used in the implementation of the application is Windows 10, Professional edition.

**3.2. CODING**

The code work of the application shall be split in the form of10 modules:

* Main module
* Create an account
* Change Password
* Deposit
* Login
* Menu 1
* Menu 2
* Read File
* Show History
* Withdraw

**3.2.1. MAIN MODULE**

* This is the main module of this project.
* To run this ATM simulator project, we have to execute this python code (program).
* This code uses an exception handling mechanism.
* If the file Account.txt already exist it read from that file else a FileNotFoundError exception occurs
* The Account.txt file is created in the forthcoming section of ‘creating account function’.
* This code contains a module named ‘menu1’ which will be called once we run the code.
* Exception handling: When an error occurs, or exception as we call it, Python will normally stop and generate an error message. These exceptions can be handled using the try statement
* The try block lets you test a block of code for errors.
* The except block lets you handle the error.

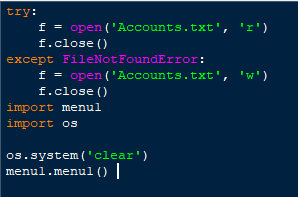


Fig 3.2.1. Main Module

* Create Accounts.txt if not exist
* The try block defines:

Read the 'Accounts.txt' file. if you try to open non existing file in readmode, this will throw an error

* The except block defines:

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

* Import modules

import menu1

import os

* menu1.menu1() is used to start the program, it calls the menu1() function

**3.2.2 CREATE AN ACCOUNT**

* Import OS: The OS module in python provides functions for interacting with the operating system. OS, comes under Python’s standard utility module
* In this module we are creating an account. A text file name Accounts.txt is created that is we are appending (creating a file if it does not exist) the file with account: name, id, password and balance
* Every time a new account is created the id get appended by 1.
* ls is the account list i.e. the lists of lines in account file
* Id\_file is maintained to keep record of the process that takes place through the account

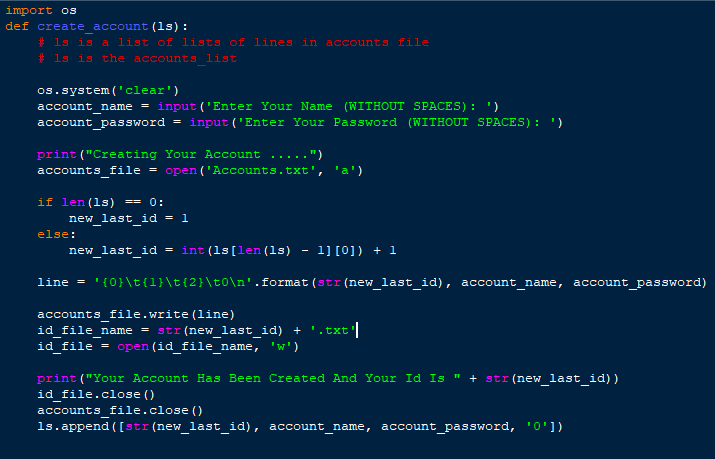


Fig 3.2.2 Create an Account

**3.2.3 CHANGE PASSWORD**

This python file change\_password.py intends to change the password of the user account.

This python code can be divided into 2 paths:

* Get old password or exit (if you want to go back)
* Change the password

The process:

1. Get the old password
2. Enter a new password
3. Append the new password corresponding to the matched id in the ‘Account.txt’ file

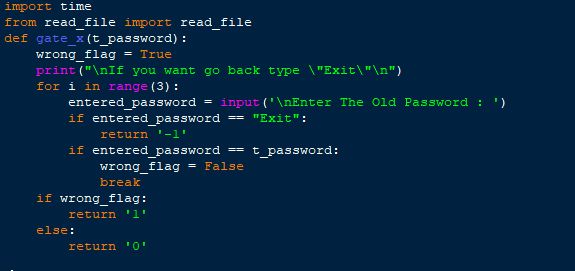


Fig 3.2.3 Change Password1

* import time handles time-related tasks
* from read\_file import read\_file

Is used to import the read\_file module

* def gate\_x(t\_password):

t\_password stands for “true password”

* Enter old password to confirm the new password
* wrong\_flag = True

True if end all tries wrong

* for i in range(3):

Limit the try to enter the old password

* if entered\_password == "Exit":

Return the Exit flag i.e. it reverts back to the menu

* if entered\_password == t\_password:

Compere if the Entered password = the True password

* wrong\_flag = False

Set to false mean the entered password confirmed

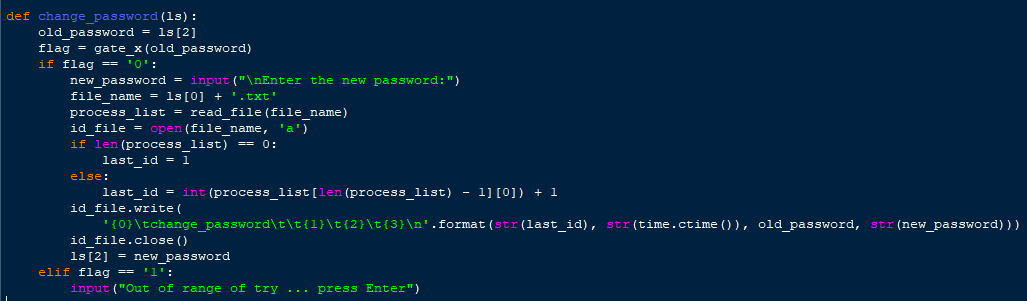


Fig 3.2.4 Change Password 2

* def change\_password(ls):

A function to change the password; It is used Get the old password

* old\_password = ls[2]

Ask to old password to enter new one

* flag = gate\_x(old\_password)

Security flag get the output flag

* if len(process\_list) == 0:

if there are no processes in the file

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

gets the last id and increments it

**3.2.4 DEPOSIT**

* **time** : An inbuilt python module that intends to handle time-related tasks.
* **read\_file**: A user defined/custom module that involves file processing.
* **ctime()** : A function that takes seconds passed since epoch (1 January, 12:00 am, 1970 ) as an argument and returns a string representing a local time.
* This module will accept user input involving money deposit such that a user’s current balance would be updated accordingly.
* The module is only triggered after the successful completion of the preliminary modules.

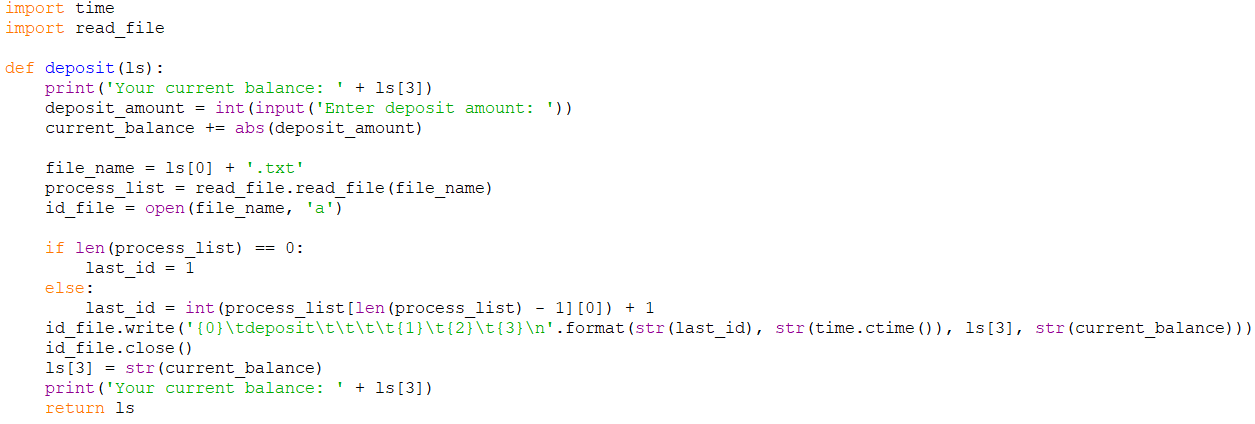


Fig 3.2.4 Deposit

* def deposit(ls): Receives a list of information of an account, and returns the account information in a 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
* 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
* current\_balance += abs(deposit\_amount): guarantees the entered value
* if len(process\_list) == 0: : if there are no processes in the file
* 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)))

write->process\_idprocess\_nameprocess\_date\_and\_timebefore\_processafter\_process

**3.2.5. LOGIN**

* This module imports the functionalities of the module menu2 (user/custom defined)
* Particularly, the function clear\_screen() is utilized to clear the output of the screen
* In this module, the user is prompted to enter his/her personal information like login id and login password
* If the input user is found, the module will function, otherwise it would terminate right and there.
* Of-course, a non-existent user also has an option to register themselves as a new user

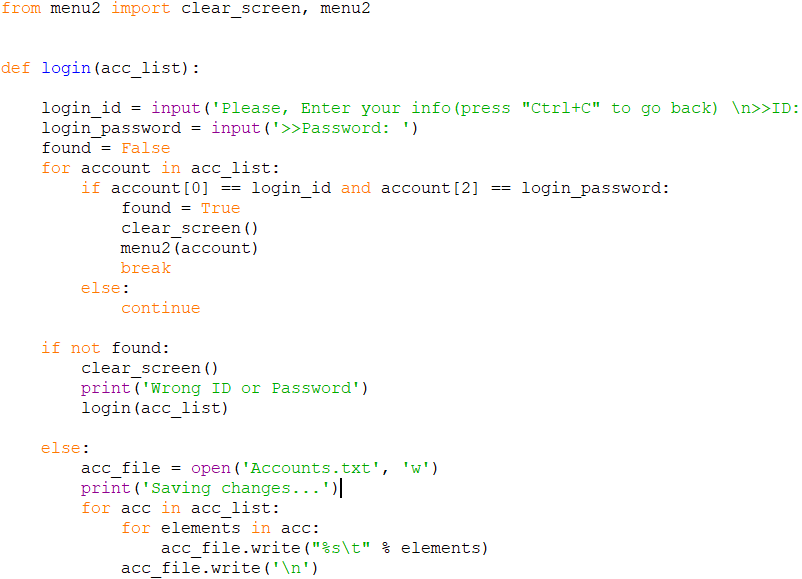


Fig 3.2.5 Login

* def login(acc\_list): This function will carry out the user login process. It incorporates acc\_list as an argument
* for account in acc\_list: The loop will traverse the account list and verify if the user exists or not
* If a user is not found, the system will prompt incorrect username/password
* for account in acc\_list:
* if account[0] == login\_id and account[2] == login\_password:

found = True

clear\_screen()

menu2(account)

The loop checks for user’s existence with respect to account id and password

* for acc in acc\_list:

for elements in acc:

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

acc\_file.write('\n')

The loop saves changes to an account after logging out of it

**3.2.6. MENU 1**

* This module relies on the presence of the login module, particularly the login() function which collects user information.
* Other modules required are create account – from which the create\_account() is imported. The read-file module is necessary as well with the read-file() function.
* Lastly, menu2’s clear\_screen() is used to clear the output screen.
* This module provides a menu or a convenient user interface encompassing features like login, create account and exit.
* The user is given choices to perform the aforementioned features so that the system would perform in accordance to the user selection

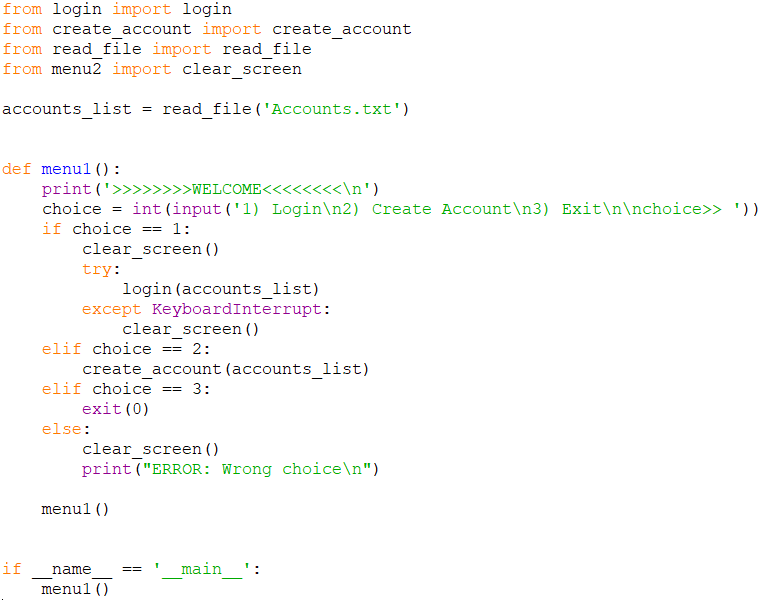


Fig 3.2.6 Menu 1

* accounts\_list = read\_file('Accounts.txt’) : The file (Account.txt) is triggered in read operation
* try:

to enable the option of (ctrl+c) to go back

* login(accounts\_list)
* except KeyboardInterrupt:

If a python program is stopped using an interrupt key, the interpreter throws this exception

* clear\_screen()

To deal with the keyboard-interrupt exception, we handle it with the except-clause

**3.2.7 MENU 2**

* This is the main module that acts as the front page of the user account
* The code of this module is implemented in the Login page, especially the clear screen method
* This module provides more functionalities than the menu 1 provides.
* The user can choose any of the following options to perform
* The system invokes appropriate methods for the chosen option
* The other modules imported in this module are withdraw, show\_history and change password

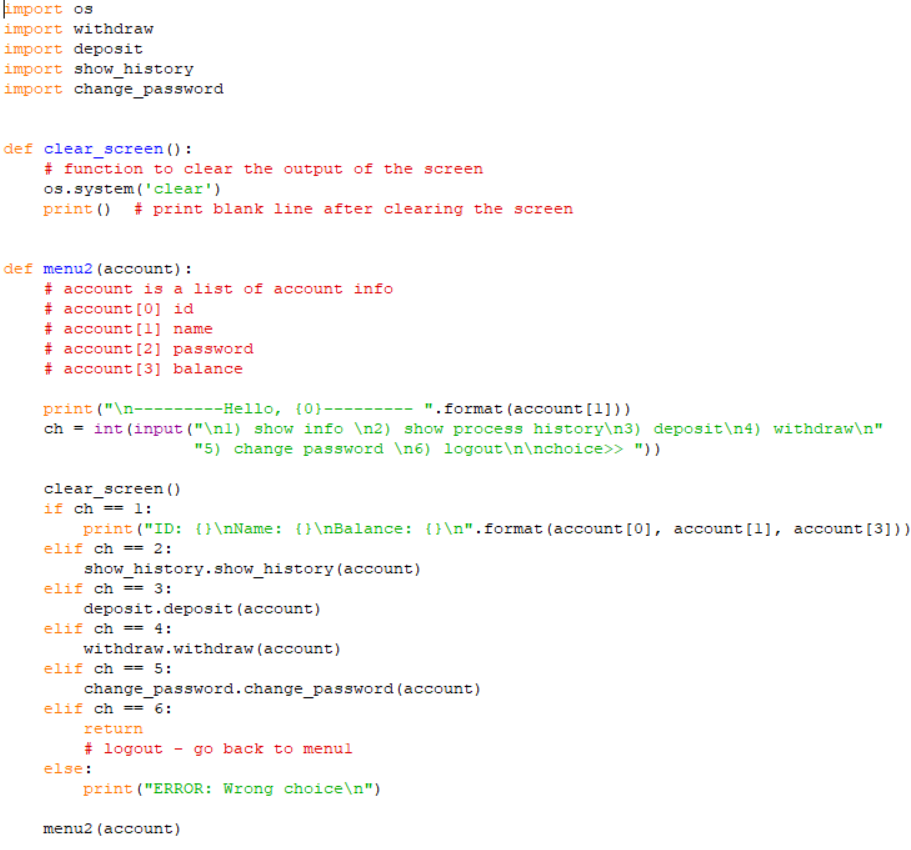


Fig 3.2.7 MENU 2

* def clear\_screen(): This method is used to clear te previous output on the screen and print a blank line on the screen.
* def menu2(account): This is the major method in this module. This method takes a list as its argument and perform the operations.
* account is a list of account information
* account[0] id
* account[1] name
* account[2] password
* account[3] balance
* In this method, this user is provided with 6 options where user can perform one option at a time.
* Option 1 – Show info – prints the details of the account such as id, name and balance
* Option 2 – Show process history – invokes the show\_history() method which is discussed in another module
* Option 3 – Deposit – invokes the deposit() method
* Option 4 – Withdraw – invokes the withdraw method
* Option 5 – Change Password – invokes the change\_password() method
* Option 6 – Logout – Returns to the main menu (menu 1) page

**3.2.8 READ FILE**

* This module contains the read\_file method that takes in the user input and stores it in a file which can be reused lated when necessary
* This function appends the content of the file. i.e it does not overwrite the content of the new with new content, instead it just adds the new content as a part of the file with the existing content

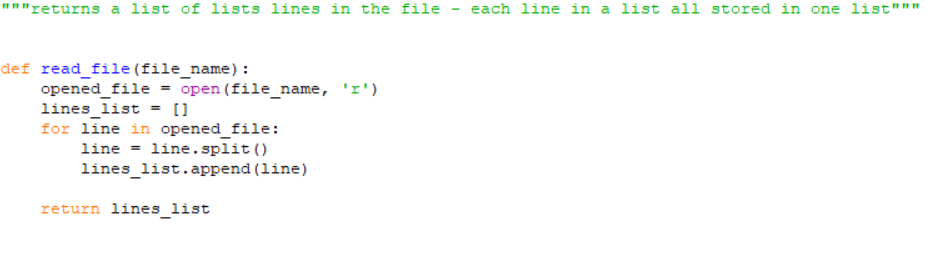


Fig 3.2.8 Read File

**3.2.8 SHOW HISTORY**

* This module is used to display the previous transactions of the user
* This module is implemented in the menu 2 module as one of its functionalities
* This module also invokes the read\_file module to display the previous records performed by the user

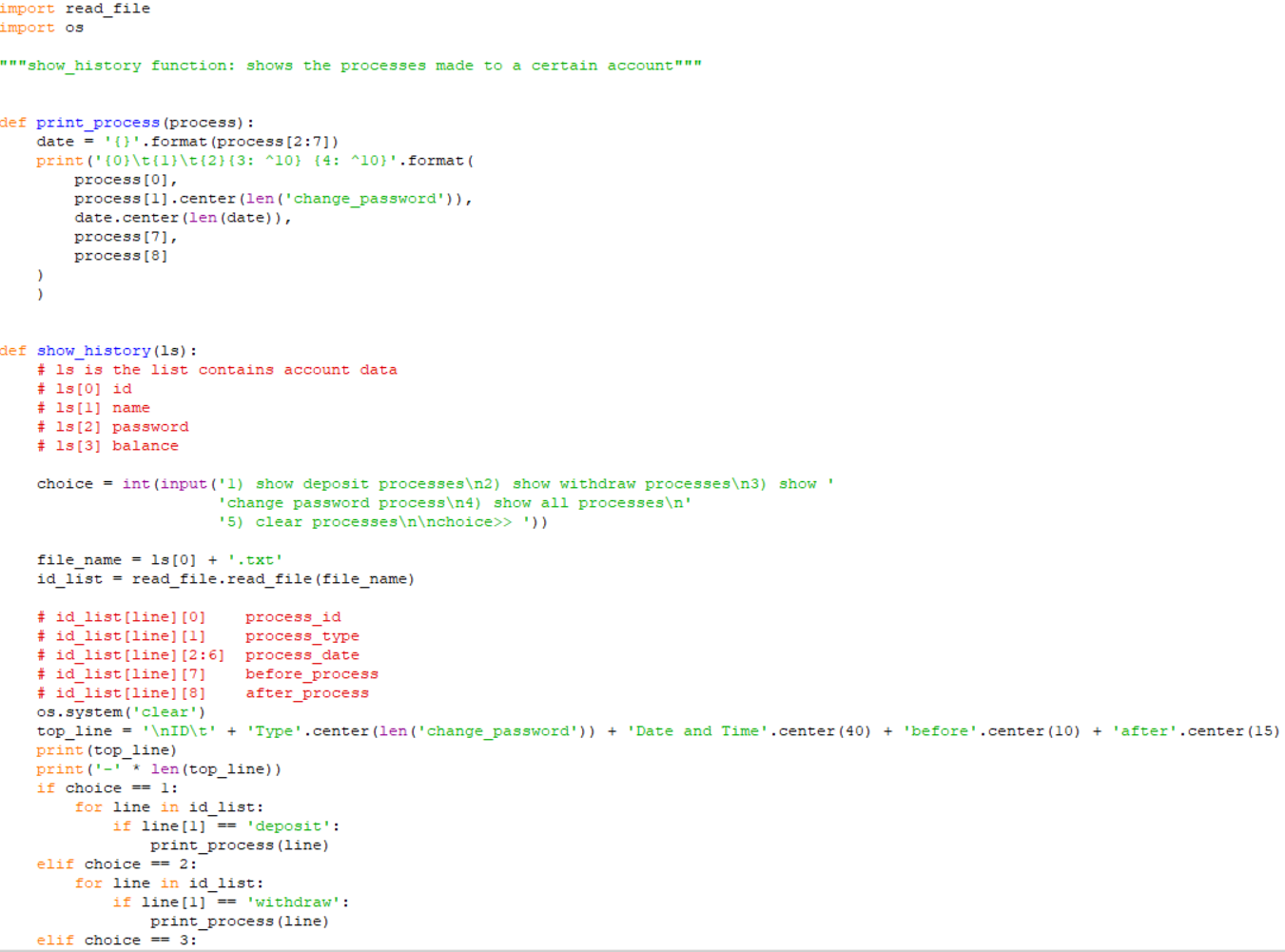


Fig 3.2.8 Show History

* The print\_process() method prints the the date, process id, before process and after process details. Id\_list is the list that contains the following details
  + - id\_list[line][0] process\_id
    - id\_list[line][1] process\_type
    - id\_list[line][2:6] process\_date
    - id\_list[line][7] before\_process
    - id\_list[line][8] after\_process
* The show\_history() method also provides the user with different functionalities to perform
* Option 1 – Show Deposit Process – Shows the deposit process by invoking the print\_process() method
* Option 2 – Show Withdraw Process – Shows the withdraw process by invoking the print\_process() method
* Option 3 – Show Change Password Process – Shows the previous changed passwords
* Option 4 – Show all processes – Shows all of the above mentioned processes
* Option 5 – Clear Process – Clears all the above processes

**3.2.9 WITHDRAW**

* This module implements the withdrawal part of the system
* This module also invokes the read\_file module to read the existing data and withdraw the amount
* There is only one function defined in this module that is invoked in the menu 2 module of the system

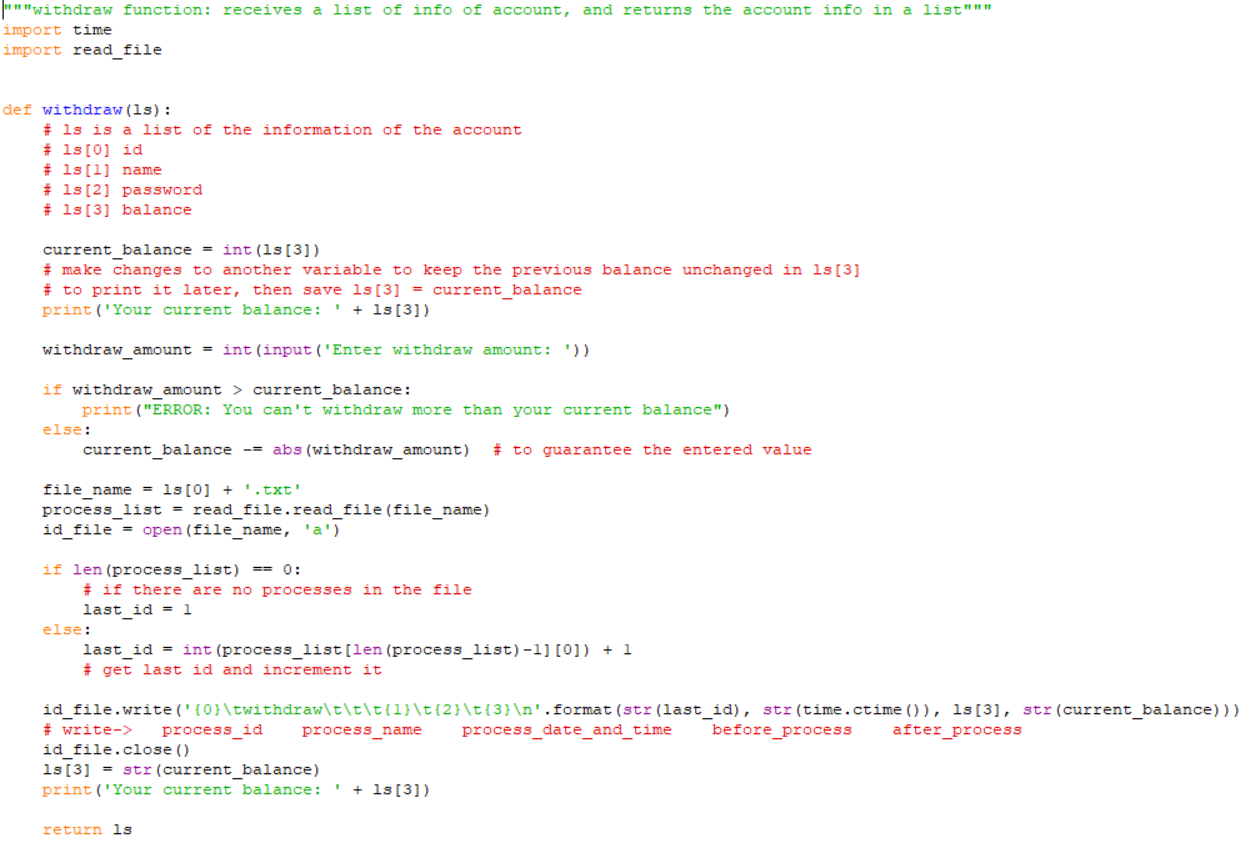


Fig 3.2.9 Withdraw

* withdraw function: receives a list of info of account, and returns the account info in a list
* withdraw() method takes a list ls as its argument and performs the withdrawal
* It asks the user to enter the amount to be withdrawn
* If the amount to be withdrawn is more than the current balance of the account then the system throws out an error
* If the amount to be withdrawn is with the range of the current balance of the account, then the amount to be withdrawn isusbtracted from the current balance and is updated as the current balance
* This method also displays the current balance.

**CHAPTER 4**

**SYSTEM TESTING**

**4.1 EXPECTED INPUT**

For the application to deliver its full capabilities, the user is expected to enter his/her valid credentials (Pin and Password). If the user fails at satisfying the aforementioned criteria, the application would terminate then and there.

**4.2 EXPECTED OUTPUT**

The application’s output is proportional to the output selected by the user. The application is ready to display all the details requested by the user via the option provided. For instance, if the user requests the application to deliver his/her account balance, the application would rightfully exhibit the corresponding values.

**4.3 TEST PLAN TABLE**

|  |  |  |
| --- | --- | --- |
| Expected Input | Input | Test Status |
| (1/2/3) | 4 | Fail |
| (1/2/3/4/5) | 7 | Fail |
| 2000 | 2300 | fail |
| 2300 | 2000 | Success |

In the first trial, if the input is 4, then it would exceed the threshold option. And hence we meet failure. In the next text case, there are wider input options, however 7 is still out of the threshold, thus the test case fails. In the third trial, the appropriate option is chosen to Withdraw some money. However, the test status is a failure as withdraw money (2300) can’t exceed that of the money is deposited. The vice-verse of this however is successful.

**4.3 SCREENSHOTS**

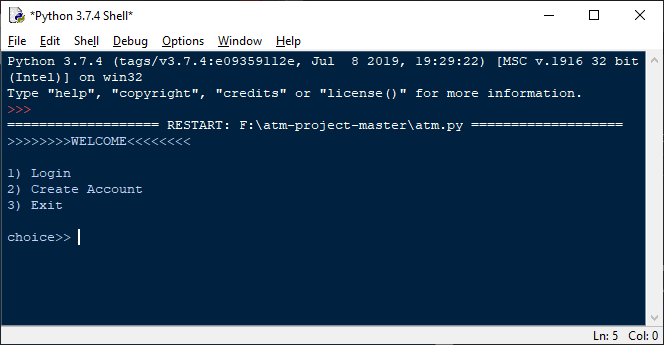


Fig 4.3.1 Menu 1

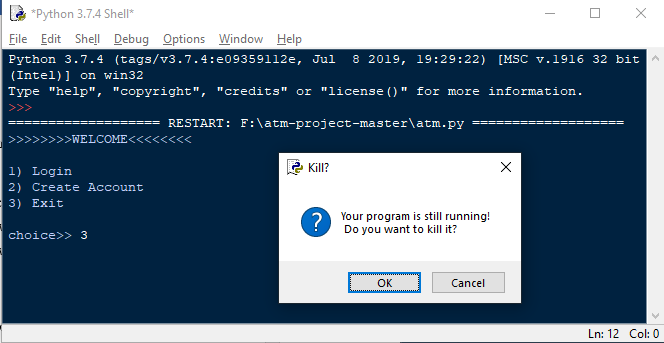


Fig 4.3.2 Kill the Program

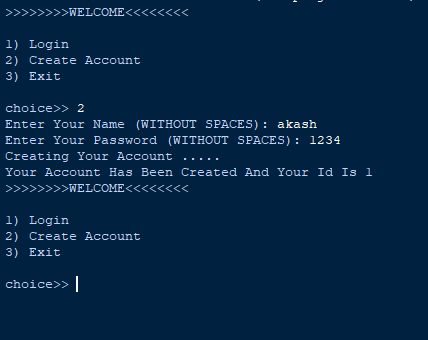


Fig 4.3.3 Create Account

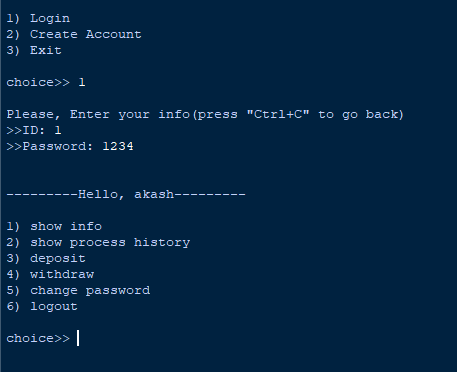


Fig 4.3.4 Login page

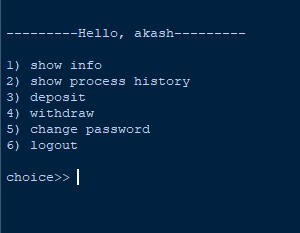


Fig 4.3.5 Menu 2

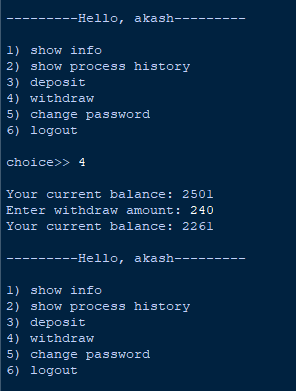


Fig 4.3.6 Withdrawal Amount

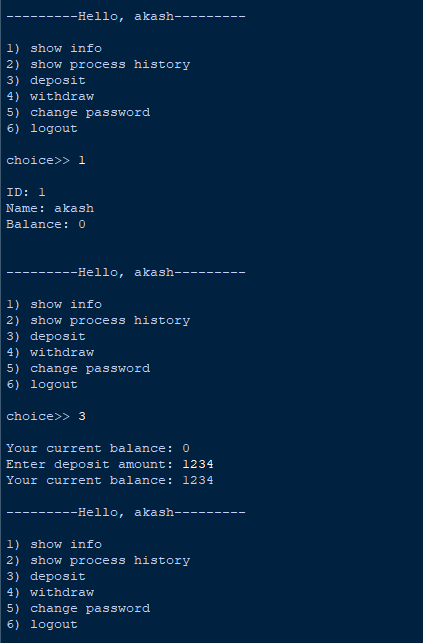


Fig 4.3.7 Deposit and Show info

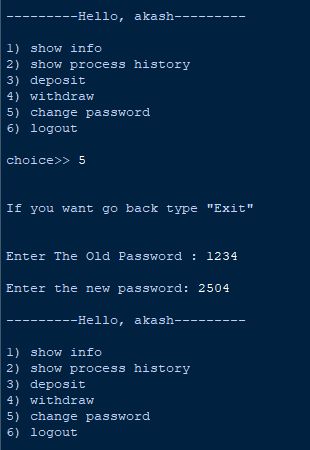


Fig 4.3.8 Change Password

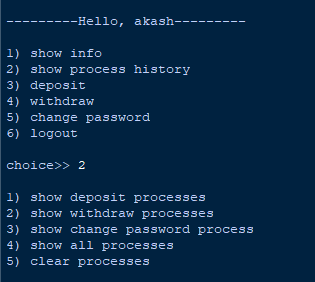


Fig 4.3.9 Show Process history

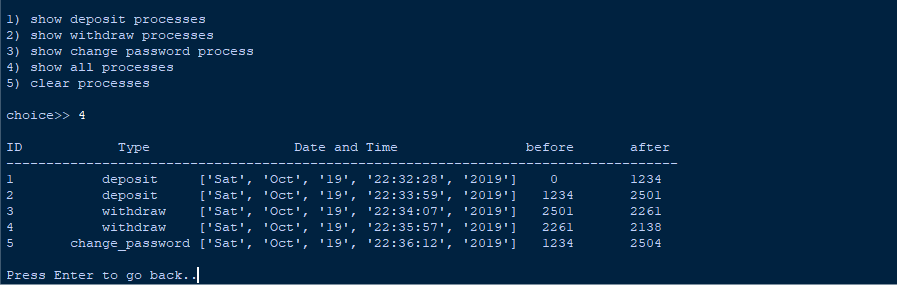


Fig 4.3.10 Process History

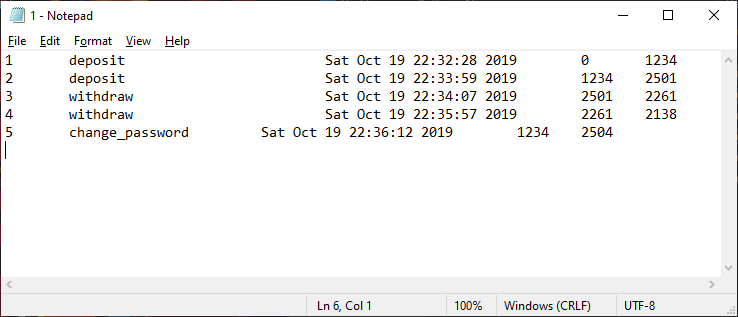


Fig 4.3.11 Process History text file

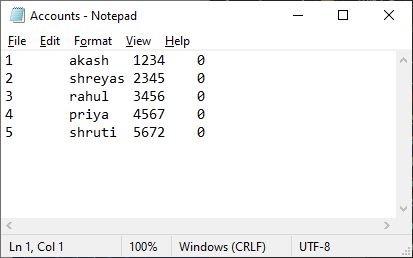


Fig 4.3.12 accounts created

**CHAPTER 5**

**CONCLUSION AND FUTURE ENHANCEMENTS**

**5.. CONCLUSION**

The application is robust and versatile as not only does it upgrade the underlying features of a realistic Automatic Teller Machine, it’s easy to access and comprehend. With 10 modules encompassing over 20 functionalities, a user can run through numerous trails as he/she deems necessary. The system can be implemented as a reference for the design of an actual ATM. Furthermore, it can be used to assess, test and make a note of the proper work flow an ATM, so that the customers do not encounter any kinds of bugs or any trivial issues. We solemnly hope that the system contributes to the aforementioned or anywhere else relevant.

**5.2. FUTURE ENHANCEMENTS**

As the system has a generic user-interface, it could be further enhanced by implemented GUI (Graphical User Interface) to make the user-interface seem fancy and compelling as a simulator. The system can further integrate DBMS (Database Management System) capabilities to be capable of handling the abundance of users that could potentially register themselves as a member of the bank. The simulator in the future can also be enhanced with a 2-step security verification in order to drastically augment the security protocol. The 2-step security verification can be either in the form of an OTP (One Time Password), or a face scanner. OTP would require cloud connectivity, and the face scanner would demand an external hardware such as a webcam; it could be programmed authenticate the validity of the user through image processing techniques or convolutional neural networks.