## ASSIGNMENT - II

## NAME : PARISHKAR SINGH

## USN : 2GI20CS081

## DATE - 29- MAY-2022

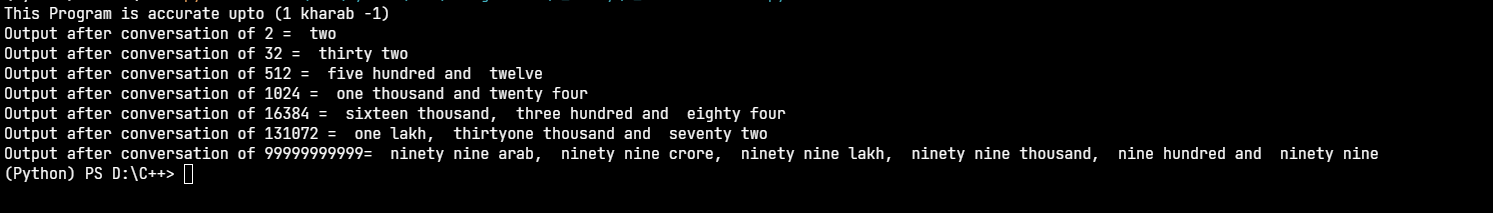
## DIV: 4-B

## *Q1. Write a python program to convert amount in digits to words:*

## *Example 15467: Fifteen thousand four hundred sixty seven*

## Code:

import inflect  
  
  
def convertOnes():  
 # given input 1 2 3 4 5 6 7 8 9  
 string = ','+inflect.engine().number\_to\_words(123456789, group=1)  
 # output list containing words from ‘’ ’’ to nine  
 return string.split(',')  
  
  
def convertTwos():  
 # given input 10 11 12 13 14 15 16 17 18 19  
 string = inflect.engine().number\_to\_words(10111213141516171819, group=2)  
 # output list containing words form “ ” , ten to nineteen   
 return string.split(',')  
  
  
def convertTens():  
 # given input 20 30 40 50 60 70 80 90  
 string = ','+' ,'+inflect.engine().number\_to\_words(2030405060708090, group=2)  
 # output list containing words from “ ” ,twenty to 80   
 return string.split(',')  
  
  
# global list of words .............................................................................  
ones = convertOnes() # contains a list of zero-nine  
twos = convertTwos() # contains a list of ten-twenty words  
tens = convertTens() # contains a list of twenty-hundred  
lessThanTwenty = ones+twos  
# ..................................................................................................  
  
  
def calc(n, value):  
 if n == 0:  
 return ''  
  
 if n > 19:  
 return tens[n // 10] + lessThanTwenty[n % 10] + value  
  
 return lessThanTwenty[n] + value  
  
  
def convertToWords(n):  
 res = calc((n // 10\*\*9 % 100), ' arab, ')  
 res += calc((n // 10\*\*7 % 100), ' crore, ')  
 res += calc((n // 10\*\*5 % 100), ' lakh, ')  
 res += calc((n // 10\*\*3 % 100), ' thousand, ')  
 res += calc((n // 10\*\*2 % 10), ' hundred ')  
  
 if n > 100 and n % 100:  
 res += 'and '  
  
 # for twos and one  
 res += calc(n % 100, '')  
  
 return res.strip().replace(', and', ' and')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("This Program is accurate upto (1 kharab -1) ")  
 print("Output after conversation of 2 = ", convertToWords(2))  
 print("Output after conversation of 32 = ", convertToWords(32))  
 print("Output after conversation of 512 = ", convertToWords(512))  
 print("Output after conversation of 1024 = ", convertToWords(1024))  
 print("Output after conversation of 16384 = ", convertToWords(16384))  
 print("Output after conversation of 131072 = ", convertToWords(131072))  
 print("Output after conversation of 99999999999= ",  
 convertToWords(99999999999))



## *Q2. Write a Python program to generate following pattern*

*\**

*\* \**

*\* \* \**

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

*# Question 2 Pattern Printing  
  
def* printPattern():  
 *for* i *in* range(1, 5):  
  
 print("\t", "\t"\*(4-i), end="")  
 *if* i == 4:  
 *for* j *in* range(1, i+2):  
 print("\*", end="")  
 *if* j *in* (2, 3):  
 print(end="\t"+" "\*(j+1))  
 *else*:  
 print(end="\t\t")  
 *else*:  
 *for* j *in* range(1, i+1):  
 print("\*", end="\t\t")  
 print("\n")  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 printPattern()



## *Q3. Design a phone book that stores Name, CellNo. Landline no. email ids as contact info. Identify what data structures to use to store and search for phone/email details given the Name of the person. Demonstrate the working with 5 contact details. :*

Code:--🡪  
# Phone Directory  
  
# This program can do selection and deletion using index or directly by name  
# Directory is sorted alphabetically and Contacts are added dynamically  
# Searching is possible (Case insensitive! uppercase and lowercase are treated the same)  
  
# The Global dictionary  
phoneBook = {}  
# temp dictionary is for indexing the contents  
tempDict = {}  
  
  
# to sort contacts alphabetically  
def sortDict():  
 return {key: value for key, value in sorted(phoneBook.items())}  
  
# Some Preloaded Contacts  
def preloadedContract():  
 phoneBook["tollfree"] = [18001028411, "0831-123124", "support@oneplus.com"]  
 phoneBook["hdfc bank"] = [18002026161, "0831-358483", "support@hdfc.com"]  
 phoneBook["emergency"] = [100, "0831-2401556", "emergency@police.com"]

# clear and bind the phonebook to index's and update tempDict  
def showContact():  
 phoneBook = sortDict()  
 tempDict.clear()  
 count = 1  
 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CONTACT LIST\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
 for count,(key, value) in enumerate(phoneBook.items()):  
 print(count+1, ":", key.capitalize())  
 tempDict[count+1] = [key, value]  
 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

# selecting contacts  
def selectContact():  
 x = input("Enter Index/Name -> ")  
 if x.isdigit():  
 z = int(x)  
 print(  
 f"Name: {tempDict[z][0].capitalize()} PhoneNo: {tempDict[z][1][0]} LandlineNo{tempDict[z][1][1]} Email: {tempDict[z][1][2]}\n")  
 else:  
 x = x.lower()  
 if x in phoneBook:  
 print(  
 f"Name:{x.capitalize()} PhoneNo: {phoneBook[x][0]} LandlineNo:{phoneBook[x][1]} Email: {phoneBook[x][2]}\n")  
 else:  
 print("No Contact Found")

# To add contacts  
def addContact():  
 name = input("Enter the Name -> ")  
 phoneno, landline, email = input(  
 "Enter Phone, Landline and email -> ").split()   
 phoneBook[name.lower()] = [phoneno, landline, email]  
 print("contact added succesfull")

# To delete contacts  
def deleteContact():  
 x = input("Enter Index/Name -> ")  
 if x.isdigit():  
 z = int(x)  
 phoneBook.pop(tempDict[z][0])  
 phonebook = sortDict()  
 print("Deleted Contact Successfully\n")  
 else:  
 x = x.lower()  
 if x in phoneBook:  
 phoneBook.pop(x)  
 phoneBook = sortDict()  
 print("Deleted Contact Successfully\n")  
 else:  
 print("No Contact Found")

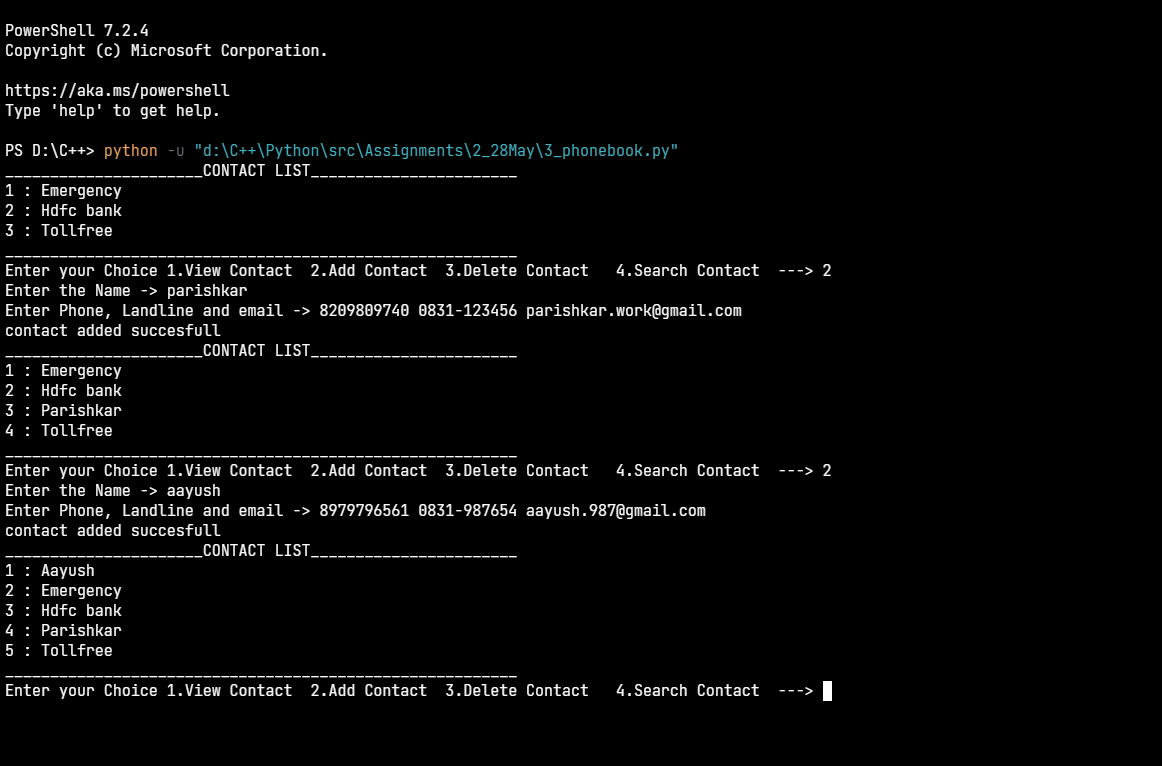
# To Search Contact  
def searchContact():  
 x = input("Enter the name to Search -> ").lower()  
  
 if x in phoneBook:  
 print("Contact Found\n-->", x.capitalize(),  
 phoneBook[x][0], phoneBook[x][1])  
 else:  
 print("not found ")

# main method  
if \_\_name\_\_ == '\_\_main\_\_':  
 preloadedContract()  
 while True:  
 showContact()  
 x = input(  
 "Enter your Choice 1.View Contact 2.Add Contact 3.Delete Contact 4.Search Contact ---> ")  
 if x == '1':  
 selectContact()  
 elif x == '2':  
 addContact()  
 elif x == '3':  
 deleteContact()  
 elif x == '4':  
 searchContact()

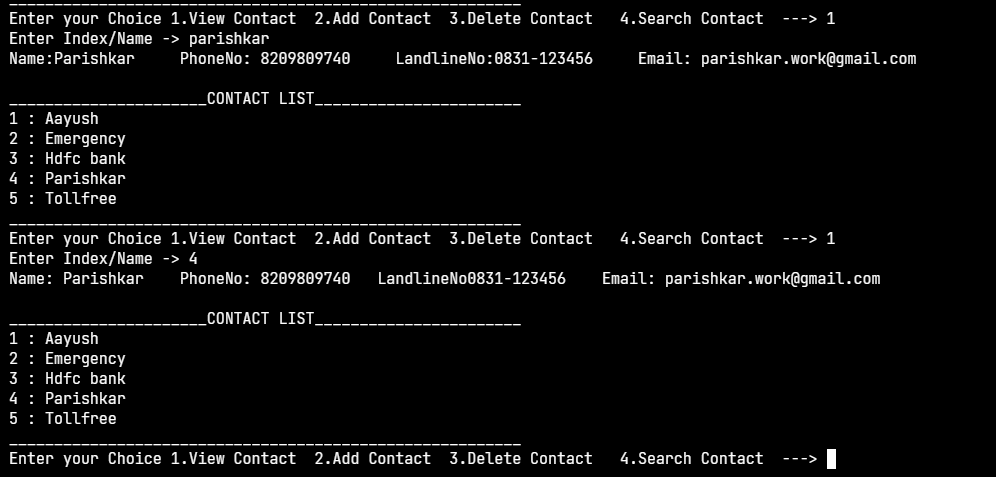
elif x == '5':  
 break

Output:

1. Adding contacts



2.selecting contacts and viewing then using index and name (case insensitive)



3.Searching contact for phone no and email using name (case insensitive)



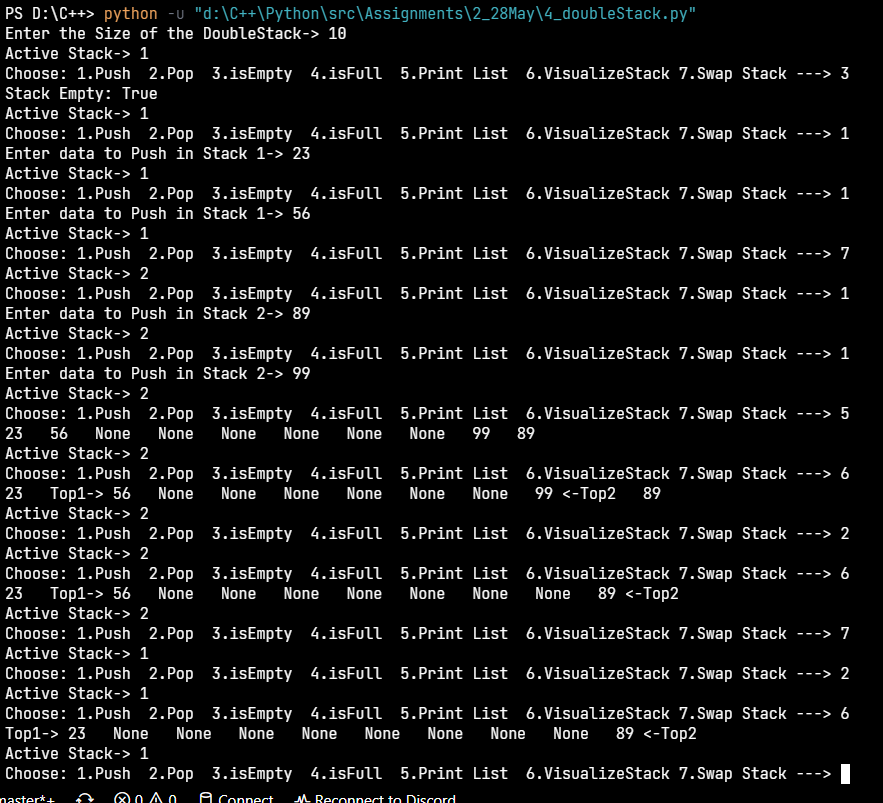
4.deleting the contents using index number or name of contract(case insensitive)



## *Q4.* *Write a program to implement a double stack using appropriate data structure and demonstrate its working. In one storage structures there should be two stacks with their own top indices.*

*# stack 1 left to right  
# stack 2 right to left  
  
class* doubleStack:  
 capacity = 0  
 list = []  
 Top1 = 0  
 Top2 = 0  
  
 *def* \_\_init\_\_(*self*, size):  
 *self*.capacity = size  
 *self*.Top1 = -1  
 *self*.Top2 = size  
 *self*.list = [*None*]\*size  
  
 *def* push(*self*, stackType, data):  
 *if* stackType == 1 *and* (*self*.Top1+1) < *self*.Top2:  
 *self*.Top1 += 1  
 *self*.list[*self*.Top1] = data  
 *elif* stackType == 2 *and* (*self*.Top2-1) > *self*.Top1:  
 *self*.Top2 -= 1  
 *self*.list[*self*.Top2] = data  
 *else*:  
 print("DoubleStack Overflow")  
  
 *def* pop(*self*, stackType):  
 *if* stackType == 1 *and self*.Top1 > -1:  
 *self*.list[*self*.Top1] = *None  
 self*.Top1 -= 1  
 *elif* stackType == 2 *and self*.Top2 < *self*.capacity:  
 *self*.list[*self*.Top2] = *None  
 self*.Top2 += 1  
 *else*:  
 print("Stack Underflow")  
  
 *def* isStackEmpty(*self*):  
 *return not* any(*self*.list)  
  
 *def* isStackFull(*self*):  
 *return* any(*self*.list)  
  
 *def* printStack(*self*):  
 *for* i *in self*.list:  
 print(i, end=" ")  
 print()  
  
 *def* visualizeStack(*self*):  
 *for* i, val *in* enumerate(*self*.list):  
 *if* i == *self*.Top1:  
 print("Top1->", val, end=" ")  
 *elif* i == *self*.Top2:  
 print(val, "<-Top2", end=" ")  
 *else*:  
 print(val, end=" ")  
 print()  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 x = int(input("Enter the Size of the DoubleStack-> "))  
 stack = doubleStack(x)  
 n = 1  
 *while True*:  
 print("Active Stack->", n)  
 x = input(  
 "Choose: 1.Push 2.Pop 3.isEmpty 4.isFull 5.Print List 6.VisualizeStack 7.Swap Stack ---> ")  
 *if* x == '1':  
 data = input(f"Enter data to Push in Stack {n}-> ")  
 stack.push(n, data)  
 *elif* x == '2':  
 stack.pop(n)  
 *elif* x == '3':  
 print("Stack Empty:", stack.isStackEmpty())  
 *elif* x == '4':  
 print("Stack Full:", stack.isStackFull())  
 *elif* x == '5':  
 stack.printStack()  
 *elif* x == '6':  
 stack.visualizeStack()  
 *elif* x == '7':  
 n = n == 2 *and* 1 *or* 2

Output:-



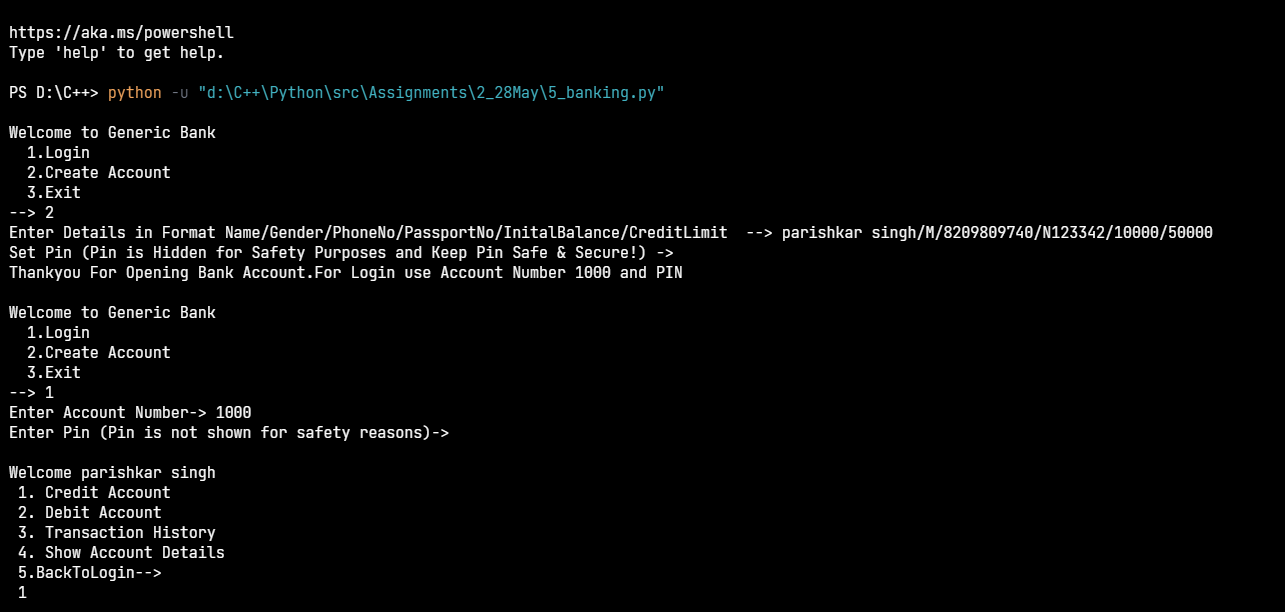
## *Q5.* *Design a appropriate classes to model a banking application that has Bank, Customer, Account, and Transaction details to be stored. Demonstrate its working with options to add a customer, create an account and perform debit and credit in the account.*

Code->

*# The code has been Tested and debugged!  
#Features:  
# 1. Creating Multiple bank accounts as per wish  
# 2. Credit Card and Debit card both are supported  
# 3. Authentication using Machine generated Account Number and Custom PIN  
# 4. Detailed Transaction Log  
# 5. Login to access bank accounts using reference's to objects(bankAccounts)*

*import* getpass  
*from* datetime *import* datetime  
  
*# This dictionary contains {account numbers:bank accounts} and will be used for authentication purposes*accounts = {}  
  
  
*class* genericBank:  
 accountNumber = 1000  
  
 *def \_\_init\_\_*(*self*, *name*, *gender*, *phone*, *passport*, *balance*, *spentCredits*, *accNum*, *pin*):  
 *self*.pin = *int*(*pin*)  
 *self*.name = *name  
 self*.phone = *int*(*phone*)  
 *self*.accNum = *int*(*accNum*)  
 *self*.gender = *gender  
 self*.balance = *int*(*balance*)  
 *self*.initBalance = *balance  
 self*.passport = *passport  
 self*.initLimit = *int*(*spentCredits*)  
 *self*.spentCredits = 0  
 *self*.transactionHistory = ''  
  
 @classmethod  
 *def* createAccount(*cls*):  
 details = *input*(  
 'Enter Details in Format Name/Gender/PhoneNo/PassportNo/InitalBalance/CreditLimit --> ')  
 sec = getpass.getpass(  
 'Set Pin (Pin is Hidden for Safety Purposes and Keep Pin Safe & Secure!) ->')  
 string = details + '/' + *str*(*cls*.accountNumber) + '/' + sec  
 *print*(  
 f'Thankyou For Opening Bank Account.For Login use Account Number {*cls*.accountNumber} and PIN')  
 *cls*.accountNumber += 1  
 *return* cls(\*string.split('/'))  
  
 *def* addTransaction(*self*, *acc*, *type*, *valid*, *amt*):  
 now = datetime.now()  
 dt\_string = now.strftime('%d/%m/%Y %H:%M:%S')  
 *if acc* == 'credit':  
 *self*.transactionHistory = dt\_string + ' Credit Card: ' + *type* + \  
 ' of ' + *str*(*amt*) + ' was ' + *valid*+'! ' + *self*.printCredit() + '\n' + \  
 *self*.transactionHistory  
 *elif acc* == 'saving':  
 *self*.transactionHistory = dt\_string + ' Saving Account: ' + *type* + \  
 ' of ' + *str*(*amt*) + ' was ' + *valid* + '! ' + *self*.printDebit() + '\n' + \  
 *self*.transactionHistory  
  
 *def* printCredit(*self*):  
 *return* f'Available Credit : {*self*.initLimit - *self*.spentCredits}'  
  
 *def* printDebit(*self*):  
 *return* f'Saving Account Balance : {*self*.balance}'  
  
 *def* getCurrentCredit(*self*):  
 *return self*.initLimit - *self*.spentCredits  
  
 *def* creditAccount(*self*):  
 *while True*:  
 *print*('\n', *self*.printCredit())  
 x = *input*(  
 '\nEnter your Choice 1. Withdraw Credit 2. Repay Credit Card Bill 3. Show Credit Limit 4.Logout Current Account ->')  
 *if* x == '1':  
 amt = *int*(*input*('Enter Amount to Withdraw:'))  
 *if* amt <= *self*.getCurrentCredit():  
 *self*.spentCredits += amt  
 *print*('Successfull withdrawn')  
 *self*.addTransaction(  
 'credit', 'withdrawal', 'succesfull', amt)  
 *else*:  
 *print*('Insufficient Credit ')  
 *self*.addTransaction(  
 'credit', 'withdrawal', 'unsucessfull', amt)  
 *elif* x == '2':  
 *if self*.getCurrentCredit() == *self*.initLimit:  
 *print*('No Bill due.')  
 *else*:  
 *print*(f'Bill Generated : Due ₹{*self*.spentCredits}')  
 amt = 0  
 *while not int*(amt) *in range*(1, *self*.spentCredits + 1):  
 amt = *int*(*input*(  
 f'Pay in Full: {*self*.spentCredits} or Custom Ammount(less <=Total Limit) -> '))  
 *break  
 self*.spentCredits -= amt  
 *self*.addTransaction(  
 'credit', 'repayment', 'succesfull', amt)  
 *print*(  
 f'Paid: {amt}, Remaining Due: {*self*.spentCredits}, Available: {*self*.getCurrentCredit()} ')  
 *elif* x == '3':  
 *print*(  
 f'Credit info:- {*self*.getCurrentCredit()}/{*self*.initLimit}')  
 *elif* x == '4':  
 *break  
  
 def* savingAccount(*self*):  
 *while True*:  
 *print*('\n', *self*.printDebit())  
 x = *input*(  
 '\nWelcome 1. Deposit Money 2. Withdraw Money 3. Show Balance 4. Logout Saving Account-> ')  
 *if* x == '1':  
 amt = *int*(*input*('Enter The amount you want to deposit-> '))  
 *self*.balance += amt  
 *self*.addTransaction(  
 'saving', 'deposition', 'succesfull', amt)  
 *elif* x == '2':  
 amt = *int*(*input*('Enter the Amount you want to withdraw-> '))  
 *if* amt <= *self*.balance:  
 *self*.balance -= amt  
 *self*.addTransaction(  
 'saving', 'withdrawal', 'succesfull', amt)  
 *else*:  
 *print*('Insufficient Balance')  
 *self*.addTransaction(  
 'saving', 'withdrawal', 'unsuccesfull', amt)  
 *elif* x == '3':  
 *print*(*self*.printDebit())  
 *elif* x == '4':  
 *break  
  
 def* printDetails(*self*):  
 *print*()  
 *print*('Bank Account Details : ')  
 *print*('Customer Account Number is : ', *self*.accNum)  
 *print*('Customer Name is : ', *self*.name)  
 *print*('Customer Gender is : ', *self*.gender)  
 *print*('Customer Phone Number is : ', *self*.phone)  
 *print*('Customer Passport Number is : ', *self*.passport)  
 *print*('Customer Credit Card limit is : ', *self*.initLimit)  
 *print*('Customer Saving Account balance is : ',  
 *self*.initBalance, *end*='\n\n')  
  
  
*def* addAccount(*cls*):  
 accounts[*cls*.accNum] = *cls  
  
  
def* login():  
 acc = *int*(*input*('Enter Account Number-> '))  
 pss = getpass.getpass('Enter Pin (Pin is not shown for safety reasons)-> ')  
 *if* acc *in* accounts.keys():  
 *if* accounts.get(acc).pin == *int*(pss):  
 *return* accounts.get(acc)  
 *return False  
  
  
if* \_\_name\_\_ == '\_\_main\_\_':  
  
 *while True*:  
 x = *input*(  
 '\nWelcome to Generic Bank\n 1.Login\n 2.Create Account\n 3.Exit\n--> ')  
 *if* x == '1':  
 referencePointer = login()  
 *if* referencePointer:  
 *while True*:  
 r = *input*(  
 f'\nWelcome {referencePointer.name}\n 1. Credit Account\n 2. Debit Account\n 3. Transaction History\n 4. Show Account Details\n 5.BackToLogin-->\n ')  
 *if* r == '1':  
 referencePointer.creditAccount()  
 *elif* r == '2':  
 referencePointer.savingAccount()  
 *elif* r == '3':  
 *print*(referencePointer.transactionHistory)  
 *elif* r == '4':  
 referencePointer.printDetails()  
 *elif* r == '5':  
 *break  
 else*:  
 *print*('Account Not Found\n')  
 *elif* x == '2':  
 referencePointer = genericBank.createAccount()  
 addAccount(referencePointer)  
 *elif* x == '3':  
 *break  
# END*

output:

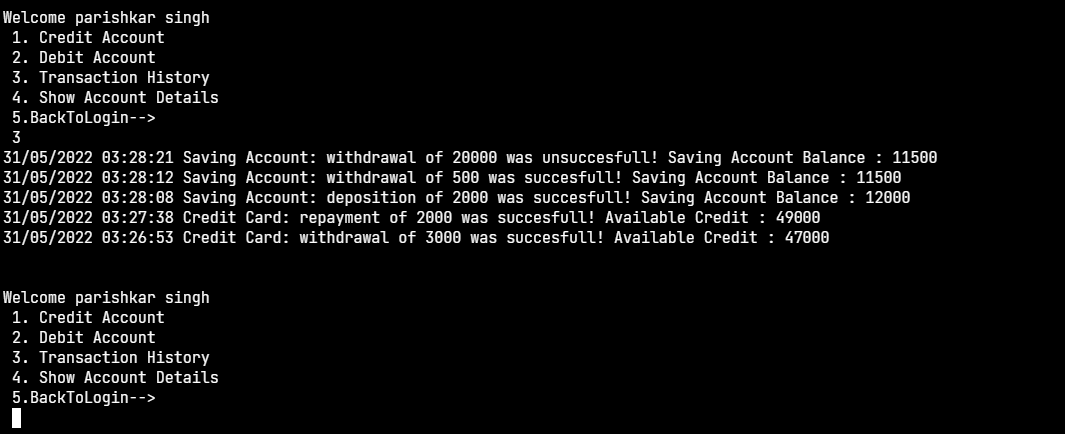
1.creating account 

2.using credit section

3.Using Debit Section



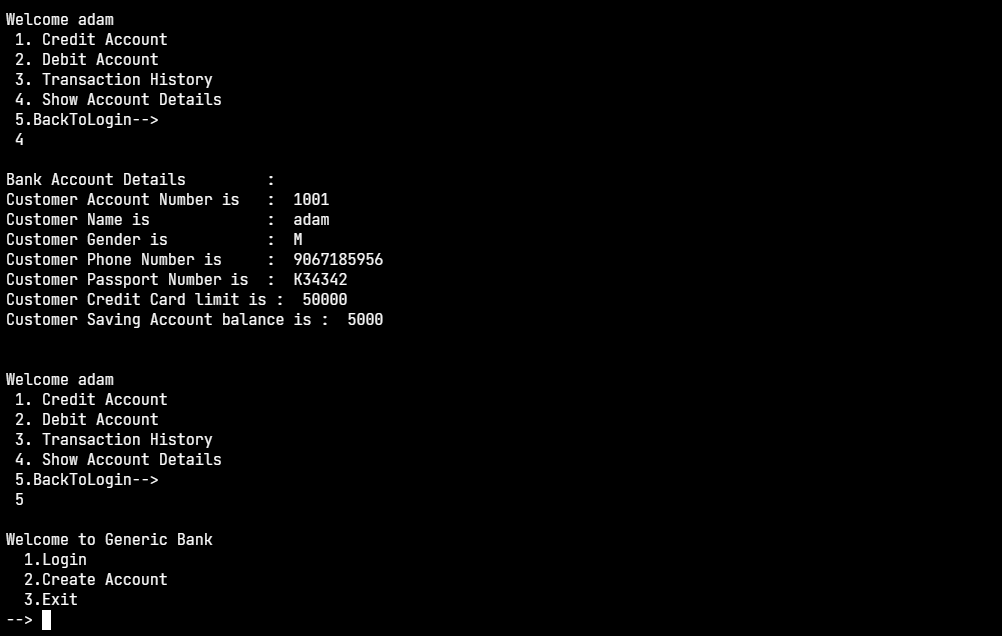
3.Transaction log for bankaccount:



4.Going back to Main Bank window and creating another bank account simultaneously



5.after successful login we’ll show the account details for adam:



#end