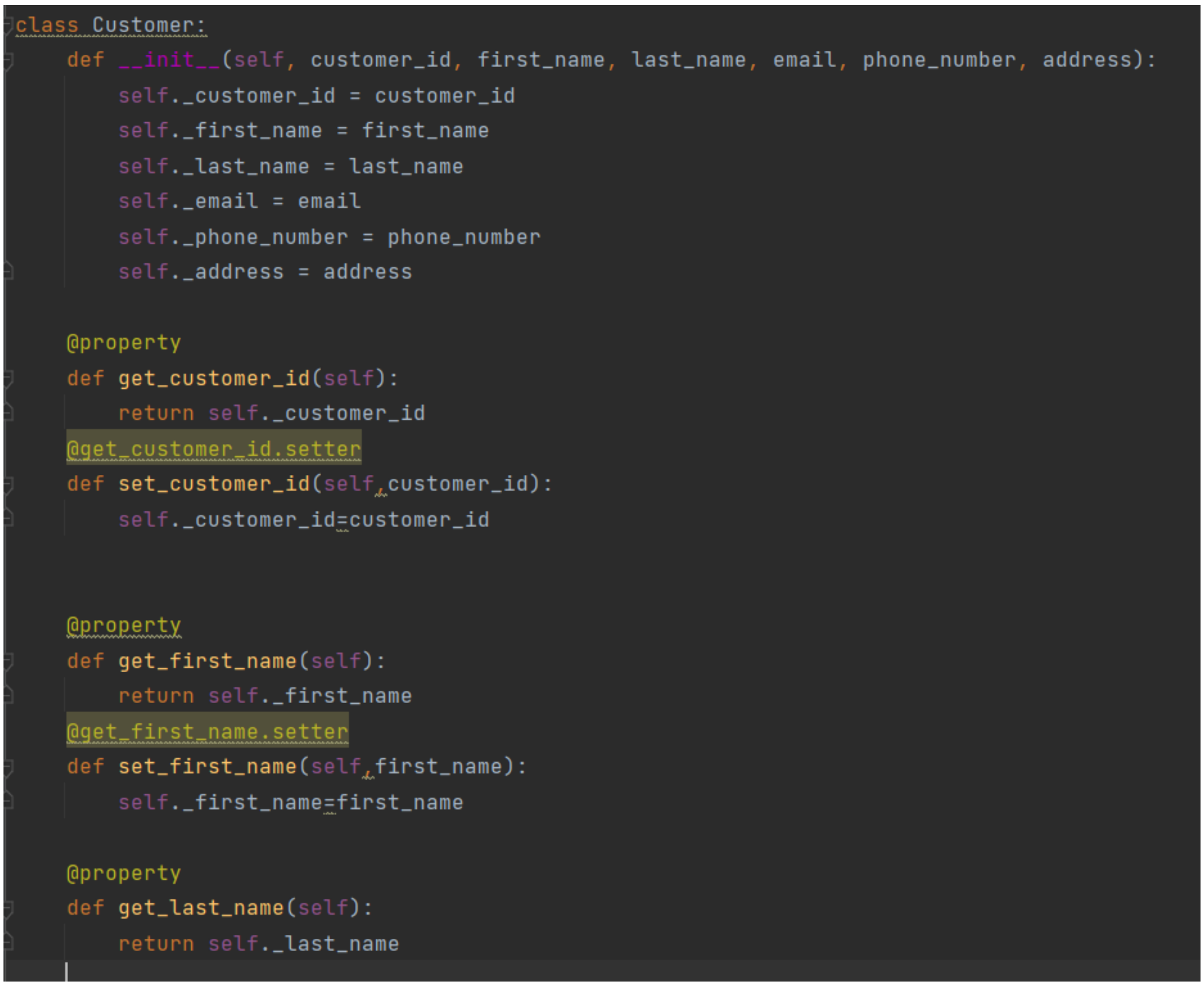
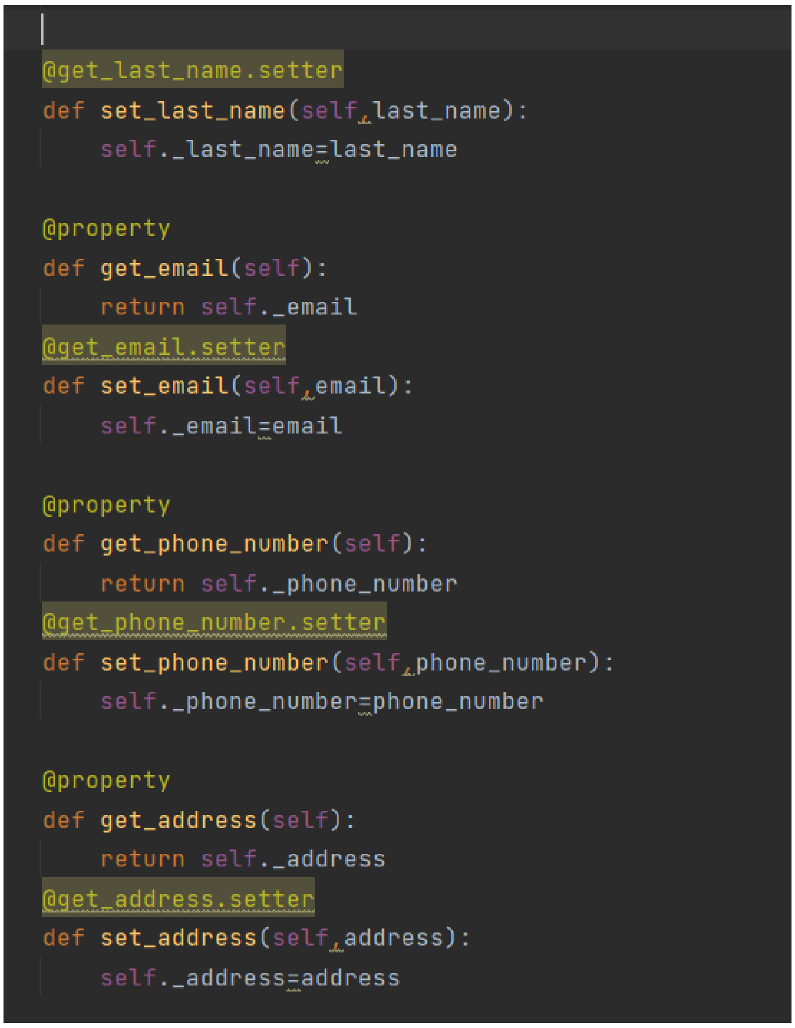
Assignment-3

Create a ‘Customer’ class as mentioned above task.

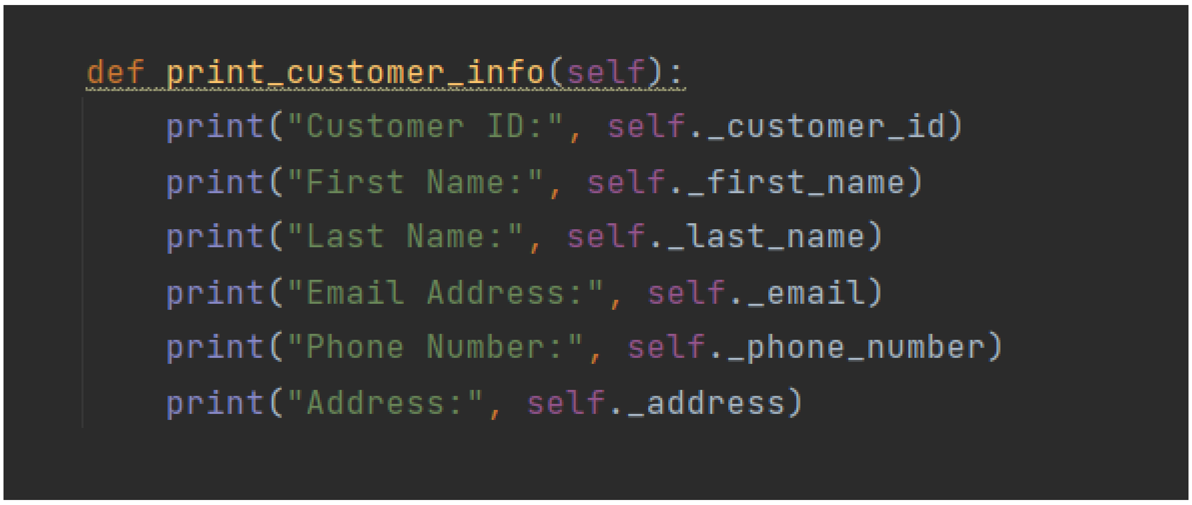


```python
class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number, address):
        self._customer_id = customer_id
        self._first_name = first_name
        self._last_name = last_name
        self._email = email
        self._phone_number = phone_number
        self._address = address

    @property
    def get_customer_id(self):
        return self._customer_id
    @get_customer_id.setter
    def set_customer_id(self,customer_id):
        self._customer_id=customer_id


    @property
    def get_first_name(self):
        return self._first_name
    @get_first_name.setter
    def set_first_name(self,first_name):
        self._first_name=first_name

    @property
    def get_last_name(self):
        return self._last_name
```



```
@get_last_name.setter
def set_last_name(self,last_name):
    self._last_name=last_name

@property
def get_email(self):
    return self._email
@get_email.setter
def set_email(self,email):
    self._email=email

@property
def get_phone_number(self):
    return self._phone_number
@get_phone_number.setter
def set_phone_number(self,phone_number):
    self._phone_number=phone_number

@property
def get_address(self):
    return self._address
@get_address.setter
def set_address(self,address):
    self._address=address
```

Print:



```
def print_customer_info(self):
    print("Customer ID:", self._customer_id)
    print("First Name:", self._first_name)
    print("Last Name:", self._last_name)
    print("Email Address:", self._email)
    print("Phone Number:", self._phone_number)
    print("Address:", self._address)
```

Create an class ‘Account’ that includes the following attributes. Generate account number using static variable.

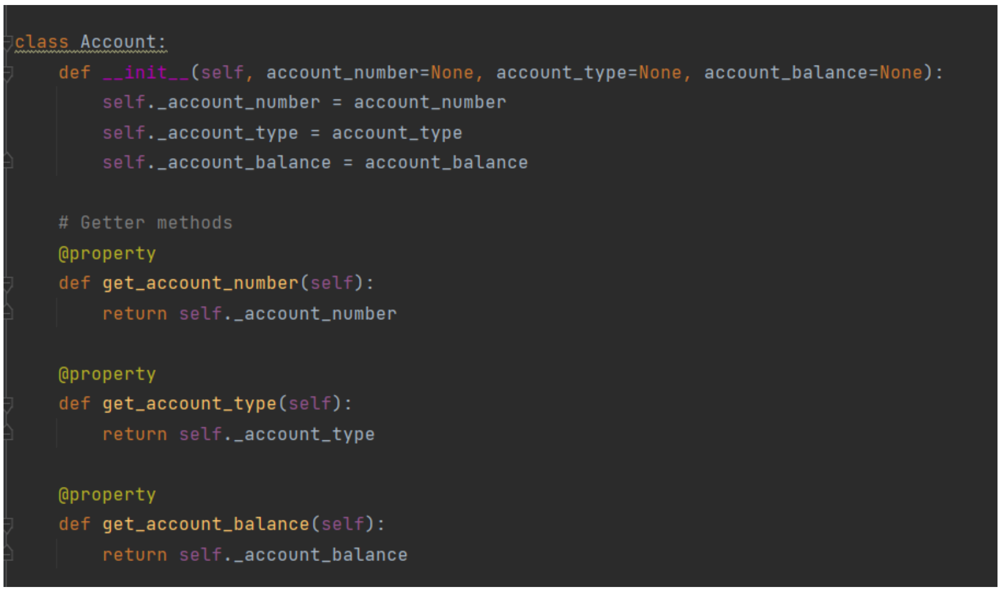
• Account Number (a unique identifier).

• Account Type (e.g., Savings, Current)

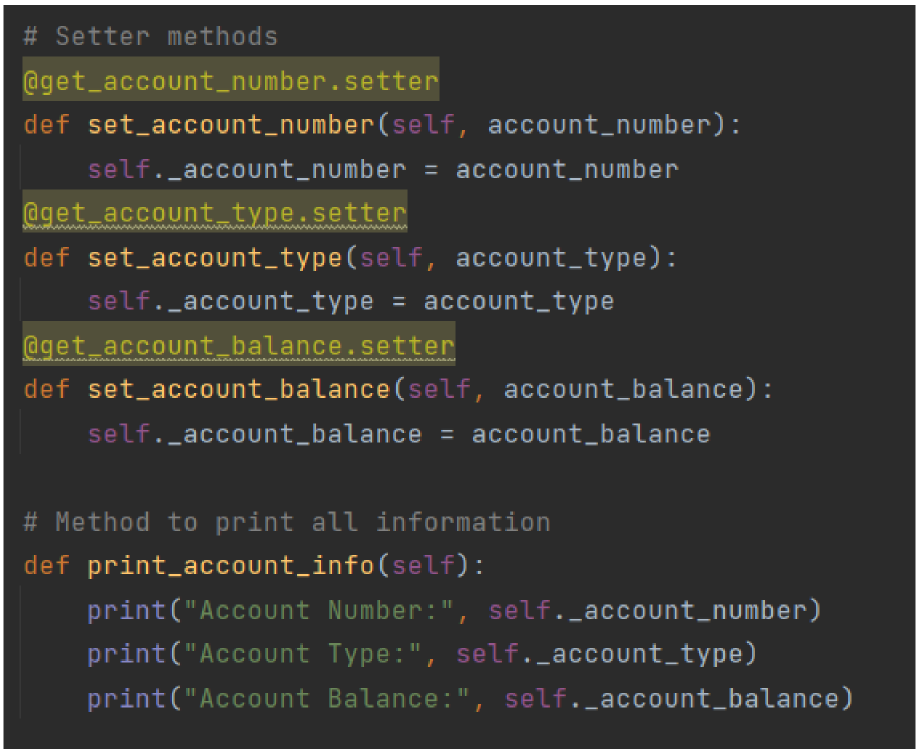
• Account Balance

• Customer (the customer who owns the account)

• lastAccNo



```
class Account:
    def __init__(self, account_number=None, account_type=None, account_balance=None):
        self._account_number = account_number
        self._account_type = account_type
        self._account_balance = account_balance

    # Getter methods
    @property
    def get_account_number(self):
        return self._account_number

    @property
    def get_account_type(self):
        return self._account_type

    @property
    def get_account_balance(self):
        return self._account_balance
```



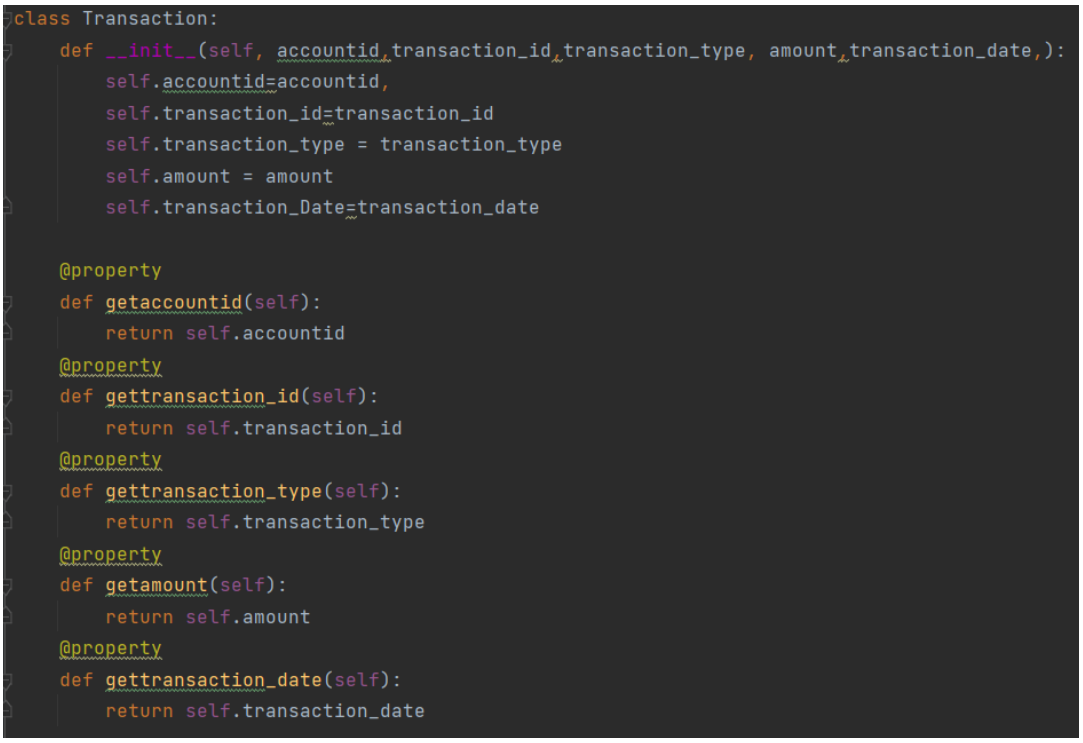
```
# Setter methods
@get_account_number.setter
def set_account_number(self, account_number):
    self._account_number = account_number
@get_account_type.setter
def set_account_type(self, account_type):
    self._account_type = account_type
@get_account_balance.setter
def set_account_balance(self, account_balance):
    self._account_balance = account_balance

# Method to print all information
def print_account_info(self):
    print("Account Number:", self._account_number)
    print("Account Type:", self._account_type)
    print("Account Balance:", self._account_balance)
```

Create a class 'TRANSACTION' that include following attributes

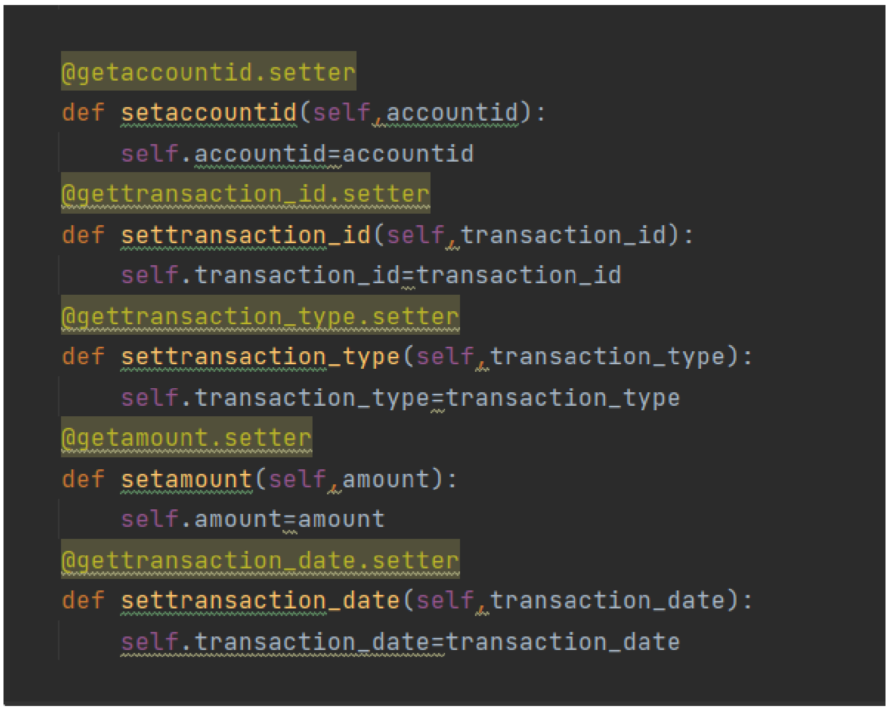
• Account • Description • Date and Time

• TransactionType(Withdraw, Deposit, Transfer)

• TransactionAmount



```python
class Transaction:
    def __init__(self, accountid,transaction_id,transaction_type, amount,transaction_date,):
        self.accountid=accountid,
        self.transaction_id=transaction_id
        self.transaction_type = transaction_type
        self.amount = amount
        self.transaction_Date=transaction_date

    @property
    def getaccountid(self):
        return self.accountid
    @property
    def gettransaction_id(self):
        return self.transaction_id
    @property
    def gettransaction_type(self):
        return self.transaction_type
    @property
    def getamount(self):
        return self.amount
    @property
    def gettransaction_date(self):
        return self.transaction_date
```



```python
    @getaccountid.setter
    def setaccountid(self,accountid):
        self.accountid=accountid
    @gettransaction_id.setter
    def settransaction_id(self,transaction_id):
        self.transaction_id=transaction_id
    @gettransaction_type.setter
    def settransaction_type(self,transaction_type):
        self.transaction_type=transaction_type
    @getamount.setter
    def setamount(self,amount):
        self.amount=amount
    @gettransaction_date.setter
    def settransaction_date(self,transaction_date):
        self.transaction_date=transaction_date
```

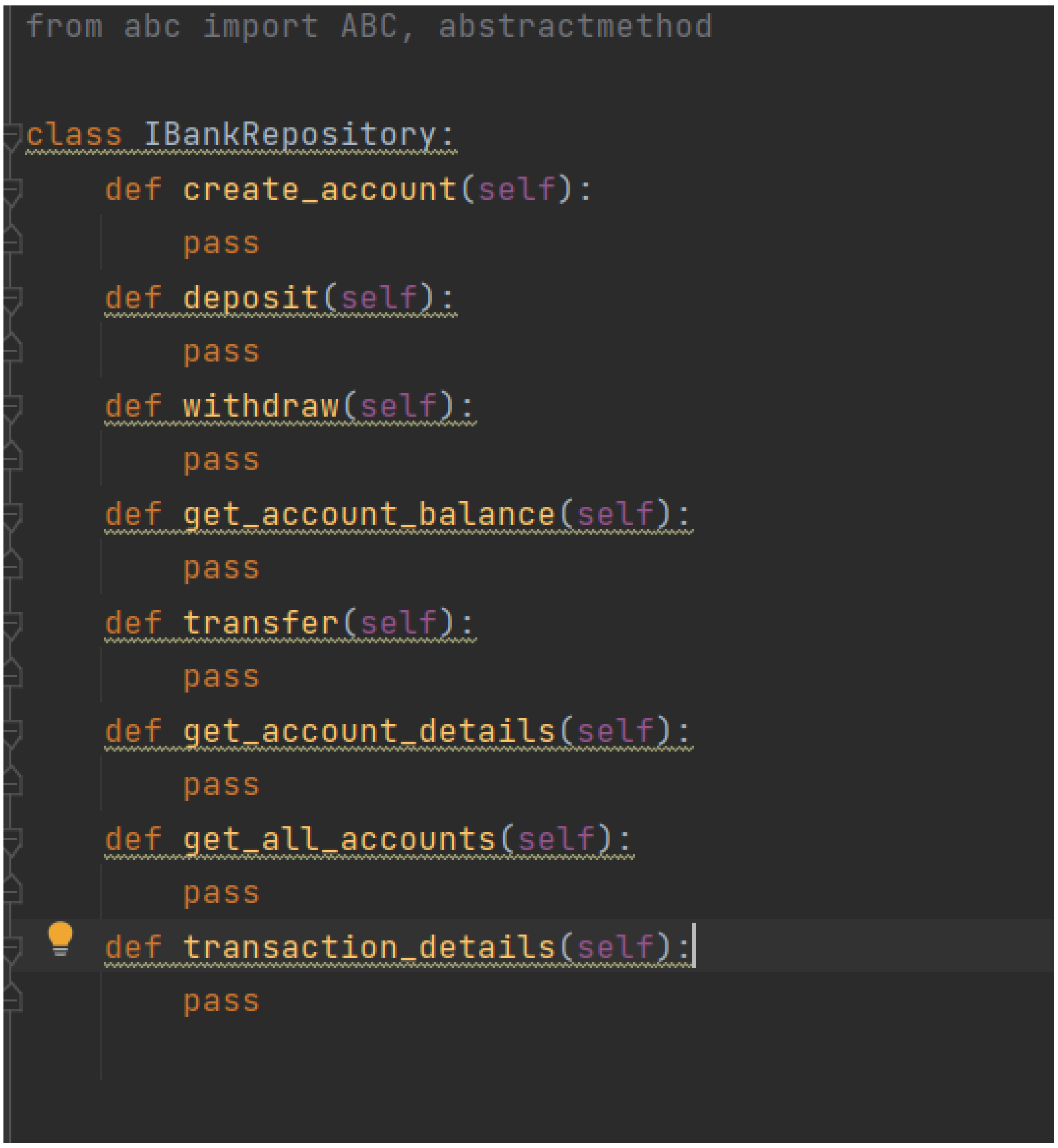
Create IBankRepository interface/abstract class with following functions:

• create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.

• listAccounts(): Array of BankAccount: List all accounts in the bank.(List[Account] accountsList)

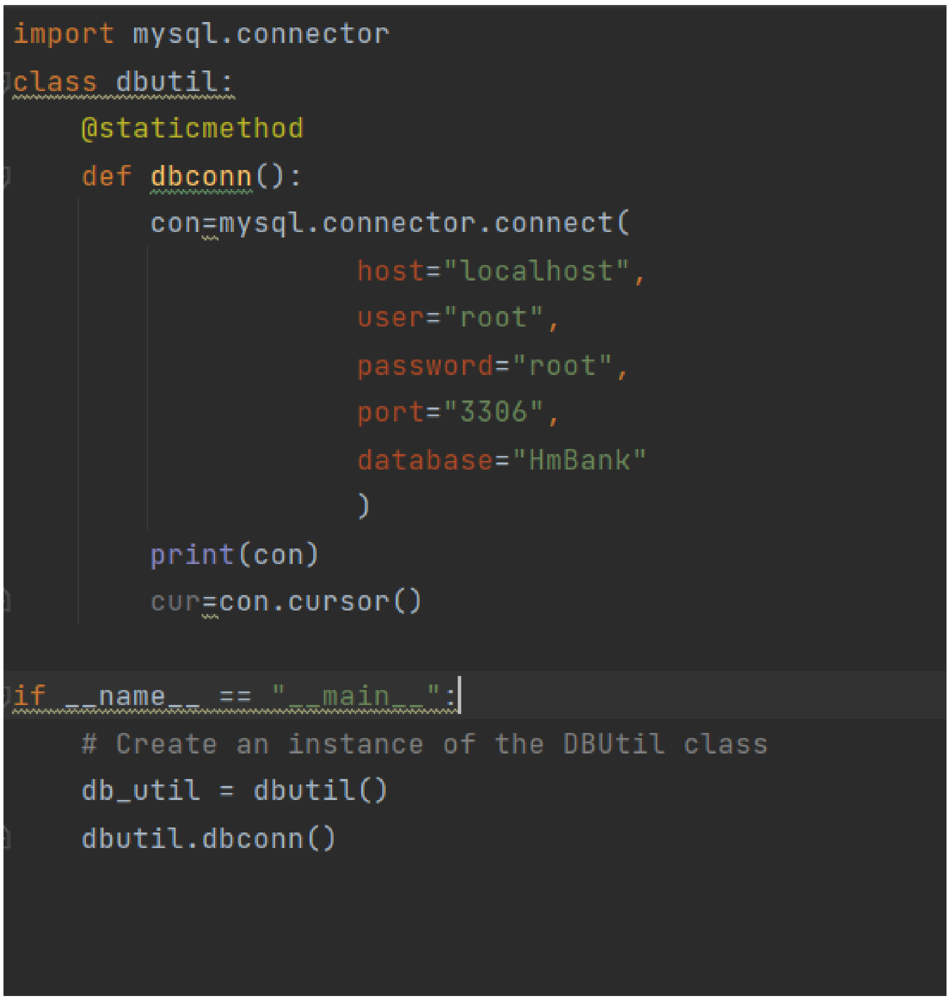
• getAccountDetails(account_number: long): Should return the account and customer details.

• calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.



```python
from abc import ABC, abstractmethod


class IBankRepository:
    def create_account(self):
        pass
    def deposit(self):
        pass
    def withdraw(self):
        pass
    def get_account_balance(self):
        pass
    def transfer(self):
        pass
    def get_account_details(self):
        pass
    def get_all_accounts(self):
        pass
    def transaction_details(self):
        pass
```

Create DBUtil class and add the following method. • static getDBConn():Connection Establish a connection to the database and return Connection reference



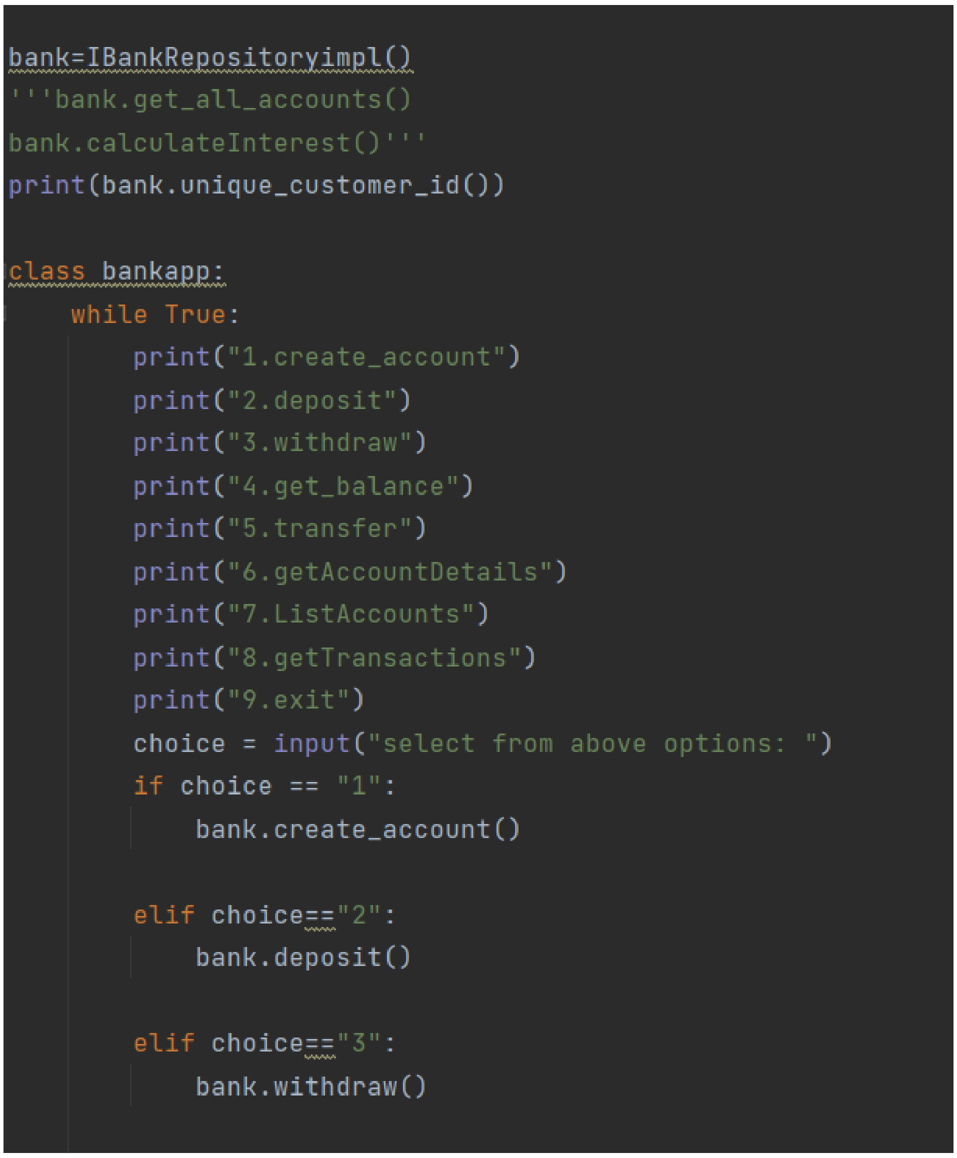
```
import mysql.connector
class dbutil:
    @staticmethod
    def dbconn():
        con=mysql.connector.connect(
                    host="localhost",
                    user="root",
                    password="root",
                    port="3306",
                    database="HmBank"
                    )
        print(con)
        cur=con.cursor()

if __name__ == "__main__":
    # Create an instance of the DBUtil class
    db_util = dbutil()
    dbutil.dbconn()
```

Create BankApp class and perform following operation:

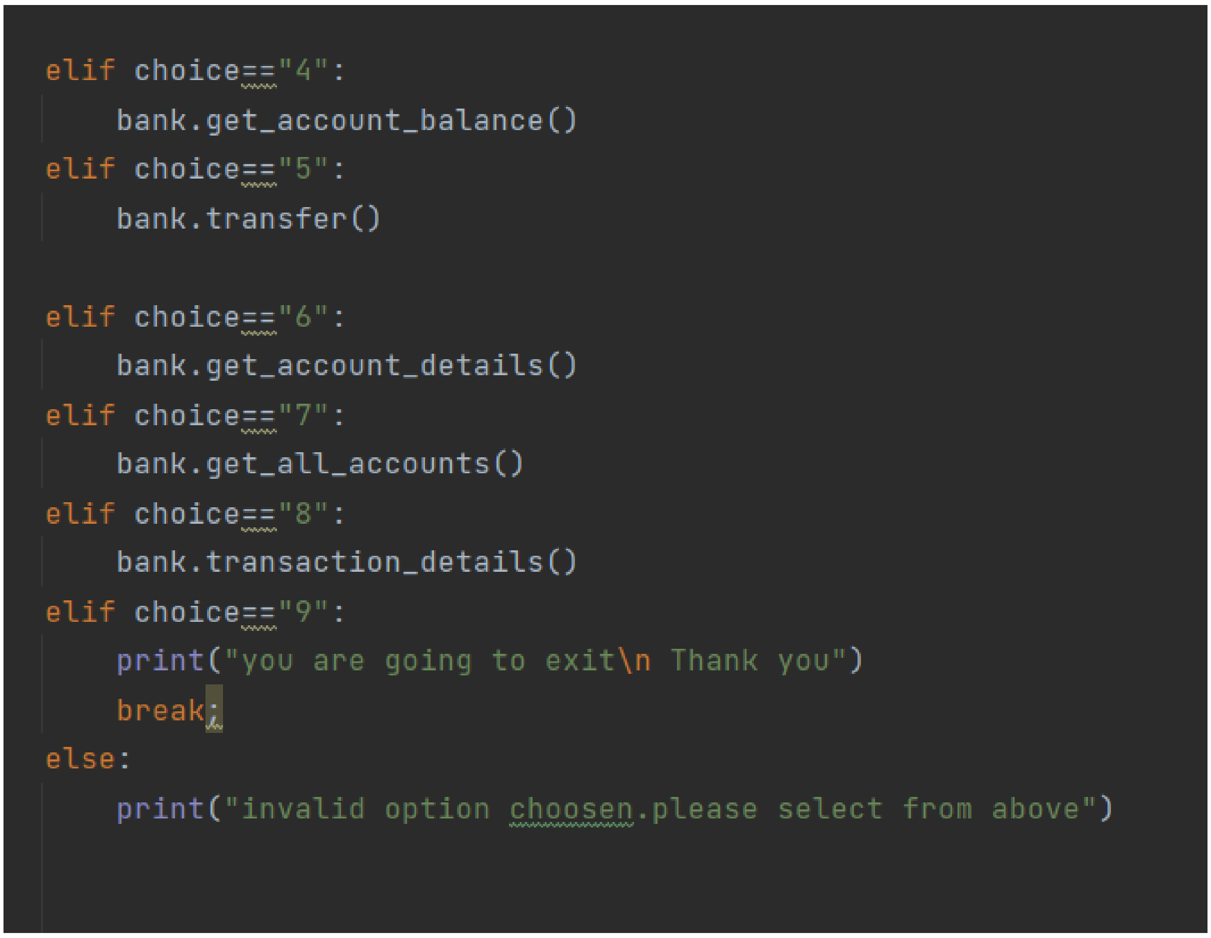
• main method to simulate the banking system. Allow the user to interact with the system by entering choice from menu such as "create_account", "deposit", "withdraw", "get_balance", "transfer", "getAccountDetails", "ListAccounts", "getTransactions" and "exit."

• create_account should display sub menu to choose type of accounts and repeat this operation until user exit.

Bank App:



```
bank=IBankRepositoryimpl()
'''bank.get_all_accounts()
bank.calculateInterest()'''
print(bank.unique_customer_id())

class bankapp:
    while True:
        print("1.create_account")
        print("2.deposit")
        print("3.withdraw")
        print("4.get_balance")
        print("5.transfer")
        print("6.getAccountDetails")
        print("7.ListAccounts")
        print("8.getTransactions")
        print("9.exit")
        choice = input("select from above options: ")
        if choice == "1":
            bank.create_account()

        elif choice=="2":
            bank.deposit()

        elif choice=="3":
            bank.withdraw()
```

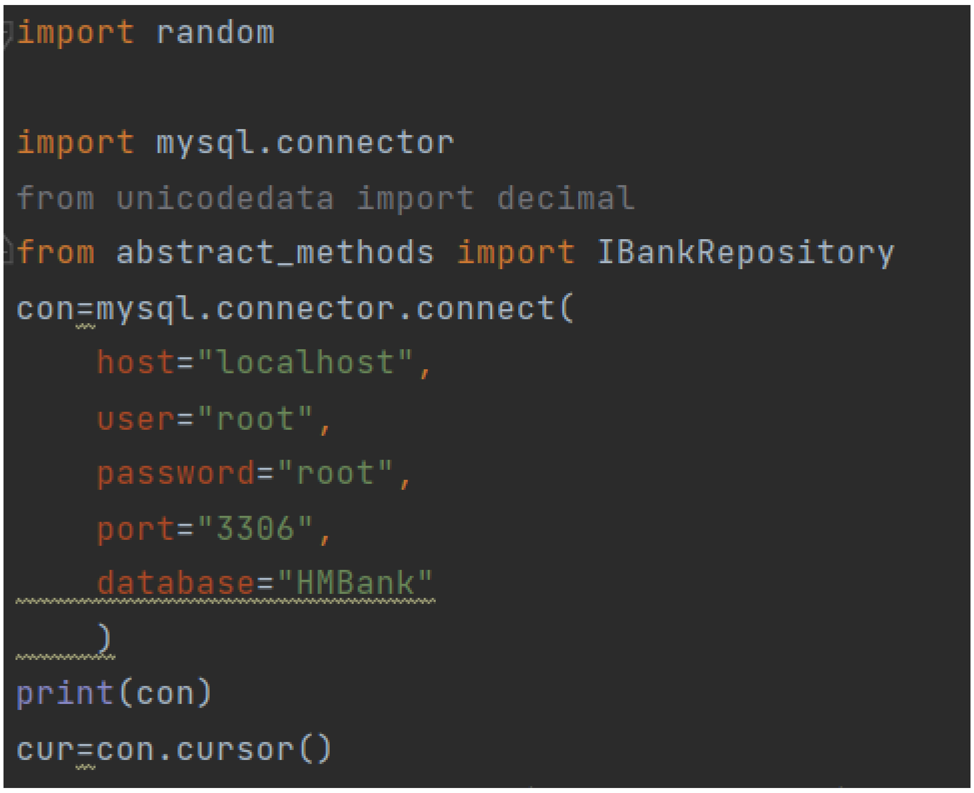


```
elif choice=="4":
    bank.get_account_balance()
elif choice=="5":
    bank.transfer()

elif choice=="6":
    bank.get_account_details()
elif choice=="7":
    bank.get_all_accounts()
elif choice=="8":
    bank.transaction_details()
elif choice=="9":
    print("you are going to exit\n Thank you")
    break;
else:
    print("invalid option choosen.please select from above")
```

Create IBankRepositoryimpl class which include following methods to interact with database.

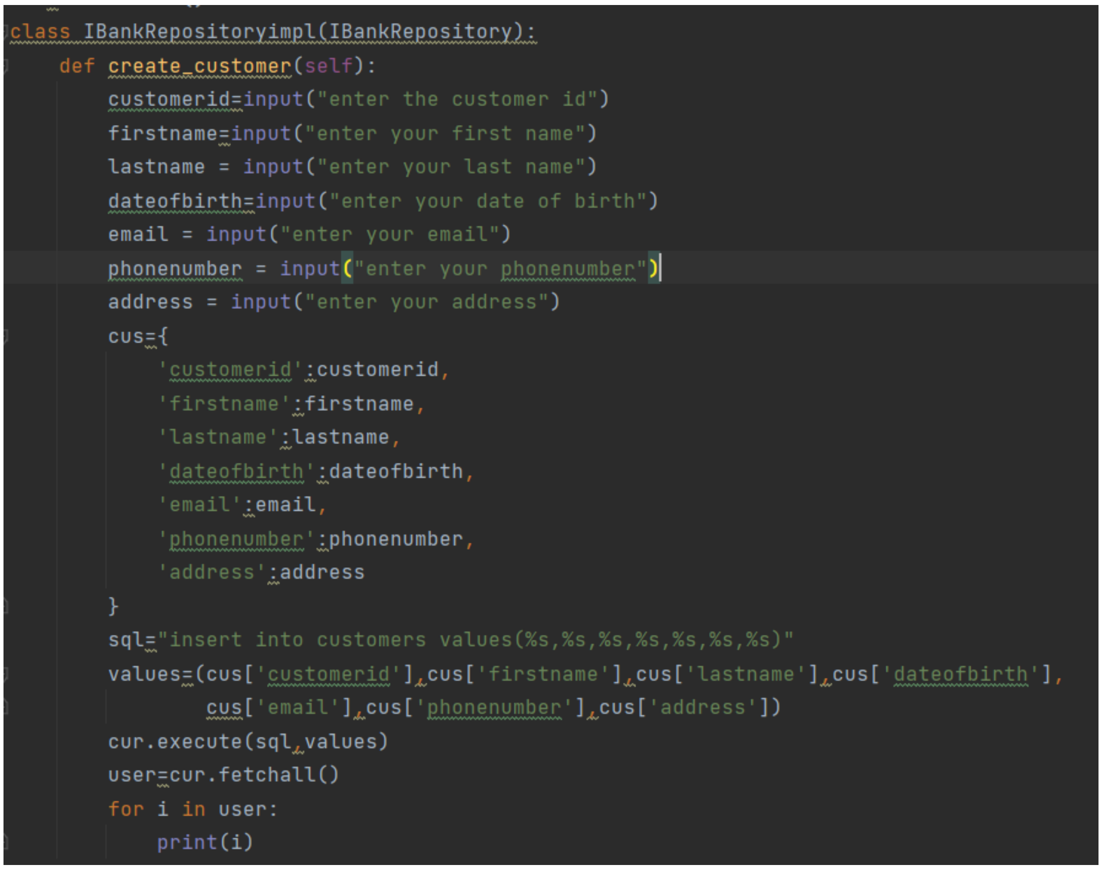
create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance. • listAccounts(): Array of BankAccount: List all accounts in the bank.(List[Account] accountsList) • getAccountDetails(account_number: long): Should return the account and customer details. • calculateInterest(): the calculate_interest() method to calculate interest based on the balance and interest rate.

Database connection:

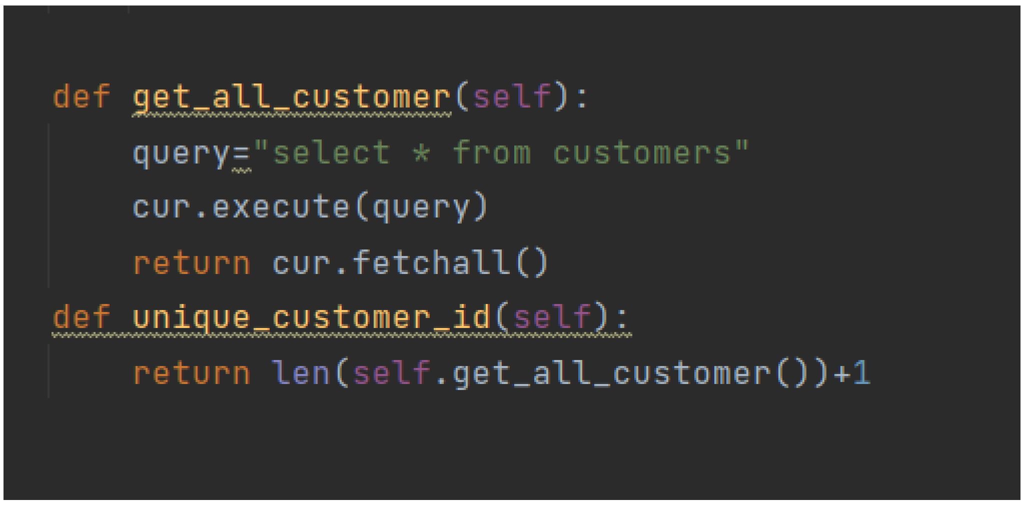


```
import random

import mysql.connector
from unicodedata import decimal
from abstract_methods import IBankRepository
con=mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    port="3306",
    database="HMBank"
    )
print(con)
cur=con.cursor()
```

Create customer:

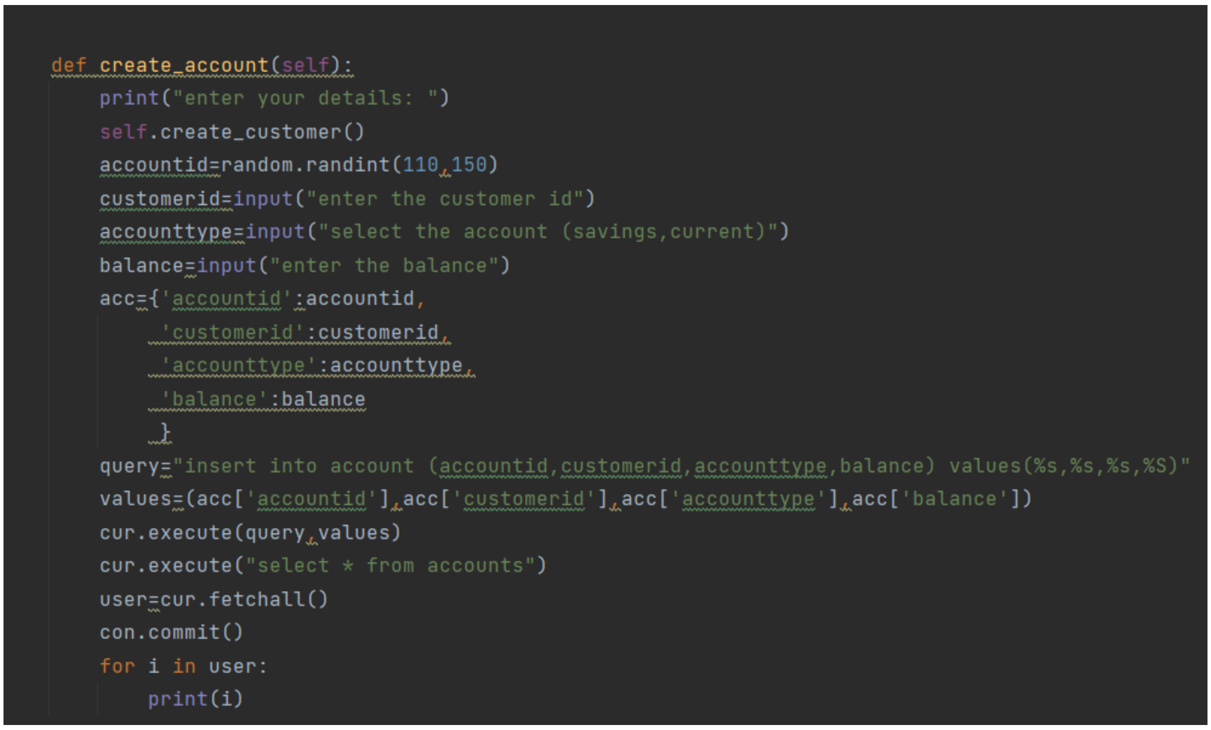


```
class IBankRepositoryimpl(IBankRepository):
    def create_customer(self):
        customerid=input("enter the customer id")
        firstname=input("enter your first name")
        lastname = input("enter your last name")
        dateofbirth=input("enter your date of birth")
        email = input("enter your email")
        phonenumber = input("enter your phonenumber")
        address = input("enter your address")
        cus={
            'customerid':customerid,
            'firstname':firstname,
            'lastname':lastname,
            'dateofbirth':dateofbirth,
            'email':email,
            'phonenumber':phonenumber,
            'address':address
        }
        sql="insert into customers values(%s,%s,%s,%s,%s,%s,%s)"
        values=(cus['customerid'],cus['firstname'],cus['lastname'],cus['dateofbirth'],
                cus['email'],cus['phonenumber'],cus['address'])
        cur.execute(sql,values)
        user=cur.fetchall()
        for i in user:
            print(i)
```



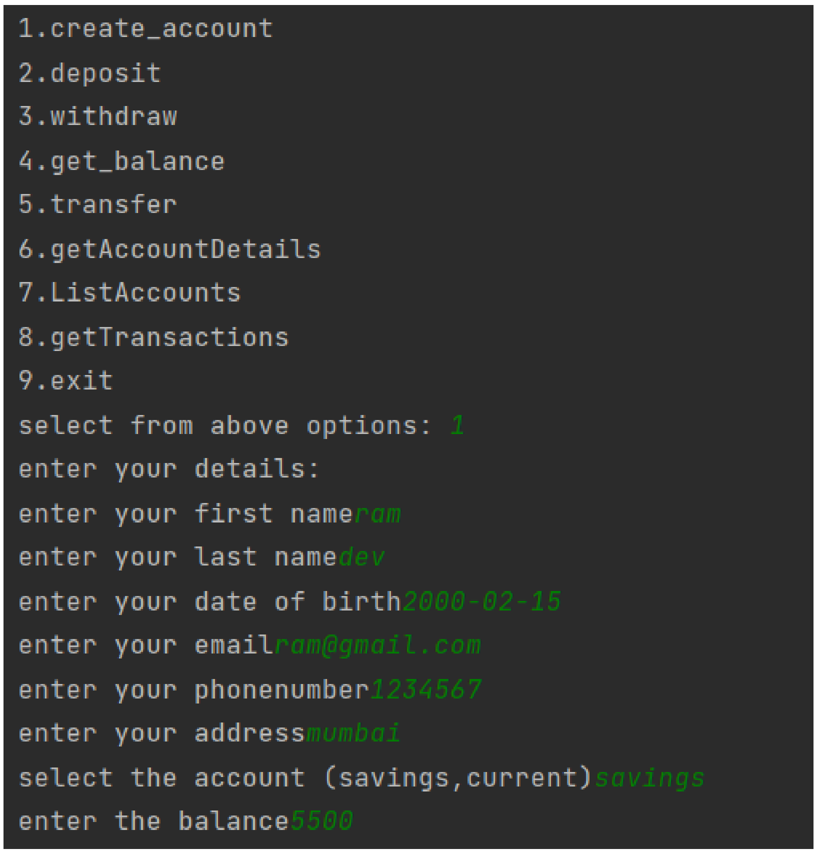
```python
def get_all_customer(self):
    query="select * from customers"
    cur.execute(query)
    return cur.fetchall()
def unique_customer_id(self):
    return len(self.get_all_customer())+1
```

Create account:



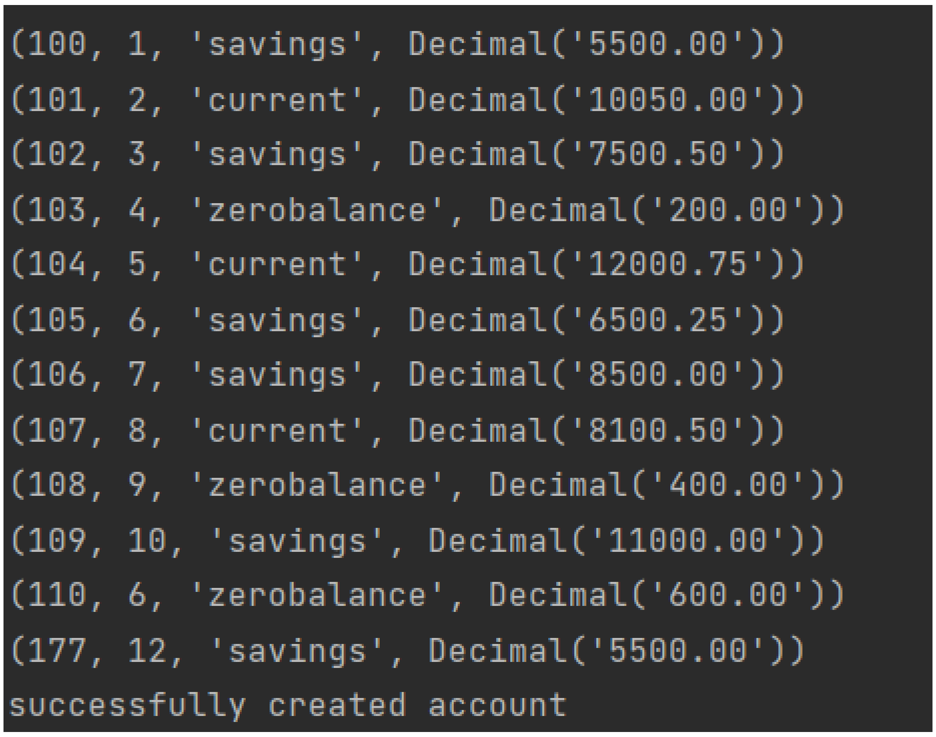
```python
def create_account(self):
    print("enter your details: ")
    self.create_customer()
    accountid=random.randint(110,150)
    customerid=input("enter the customer id")
    accounttype=input("select the account (savings,current)")
    balance=input("enter the balance")
    acc={'accountid':accountid,
         'customerid':customerid,
         'accounttype':accounttype,
         'balance':balance
         }
    query="insert into account (accountid,customerid,accounttype,balance) values(%s,%s,%s,%S)"
    values=(acc['accountid'],acc['customerid'],acc['accounttype'],acc['balance'])
    cur.execute(query,values)
    cur.execute("select * from accounts")
    user=cur.fetchall()
    con.commit()
    for i in user:
        print(i)
```

Outputs:



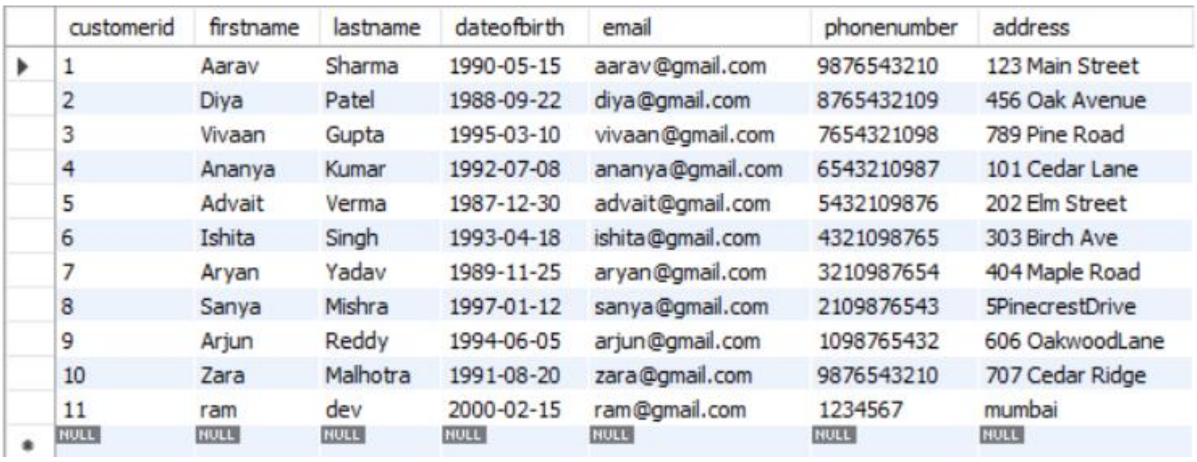
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 1
enter your details:
enter your first nameram
enter your last namedev
enter your date of birth2000-02-15
enter your emailram@gmail.com
enter your phonenumber1234567
enter your addressmumbai
select the account (savings,current)savings
enter the balance5500
```

Accounts table:



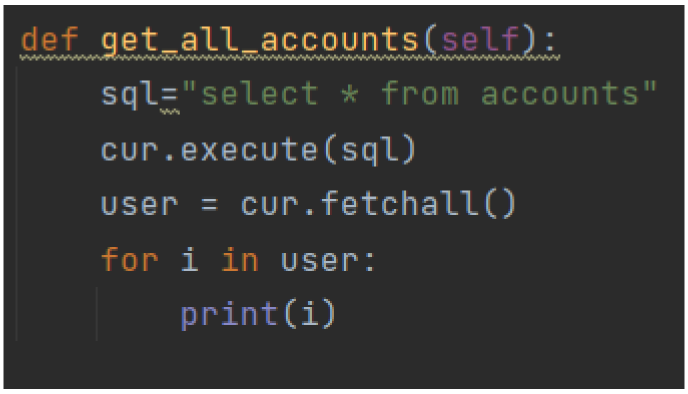
```
(100, 1, 'savings', Decimal('5500.00'))
(101, 2, 'current', Decimal('10050.00'))
(102, 3, 'savings', Decimal('7500.50'))
(103, 4, 'zerobalance', Decimal('200.00'))
(104, 5, 'current', Decimal('12000.75'))
(105, 6, 'savings', Decimal('6500.25'))
(106, 7, 'savings', Decimal('8500.00'))
(107, 8, 'current', Decimal('8100.50'))
(108, 9, 'zerobalance', Decimal('400.00'))
(109, 10, 'savings', Decimal('11000.00'))
(110, 6, 'zerobalance', Decimal('600.00'))
(177, 12, 'savings', Decimal('5500.00'))
successfully created account
```

Customers table:



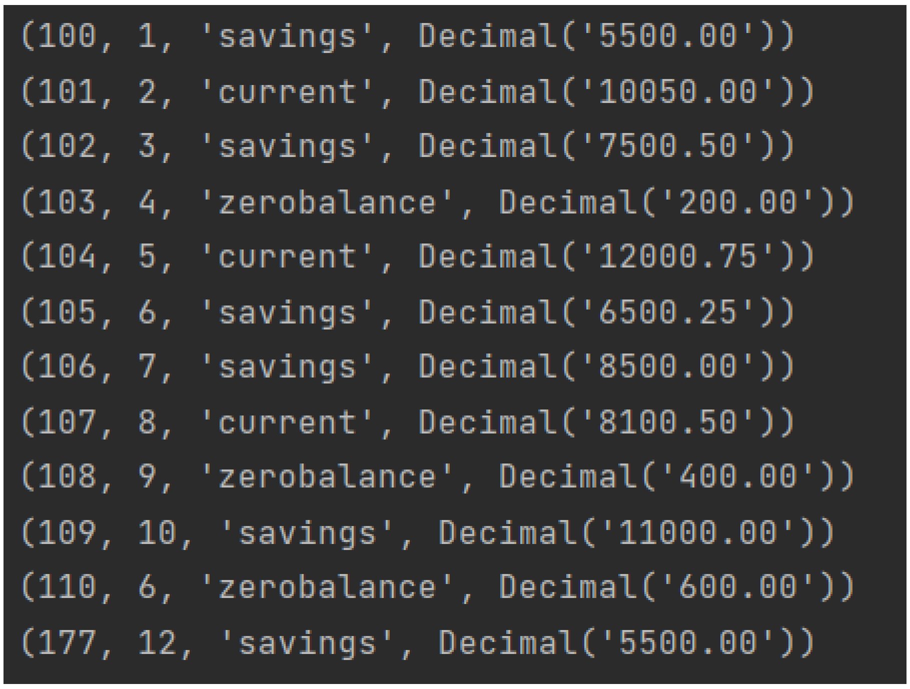
customerid	firstname	lastname	dateofbirth	email	phonenumber	address
1	Aarav	Sharma	1990-05-15	aarav@gmail.com	9876543210	123 Main Street
2	Diya	Patel	1988-09-22	diya@gmail.com	8765432109	456 Oak Avenue
3	Vivaan	Gupta	1995-03-10	vivaan@gmail.com	7654321098	789 Pine Road
4	Ananya	Kumar	1992-07-08	ananya@gmail.com	6543210987	101 Cedar Lane
5	Advait	Verma	1987-12-30	advait@gmail.com	5432109876	202 Elm Street
6	Ishita	Singh	1993-04-18	ishita@gmail.com	4321098765	303 Birch Ave
7	Aryan	Yadav	1989-11-25	aryan@gmail.com	3210987654	404 Maple Road
8	Sanya	Mishra	1997-01-12	sanya@gmail.com	2109876543	5PinecrestDrive
9	Arjun	Reddy	1994-06-05	arjun@gmail.com	1098765432	606 OakwoodLane
10	Zara	Malhotra	1991-08-20	zara@gmail.com	9876543210	707 Cedar Ridge
11	ram	dev	2000-02-15	ram@gmail.com	1234567	mumbai
NULL	NULL	NULL	NULL	NULL	NULL	NULL

List accounts:



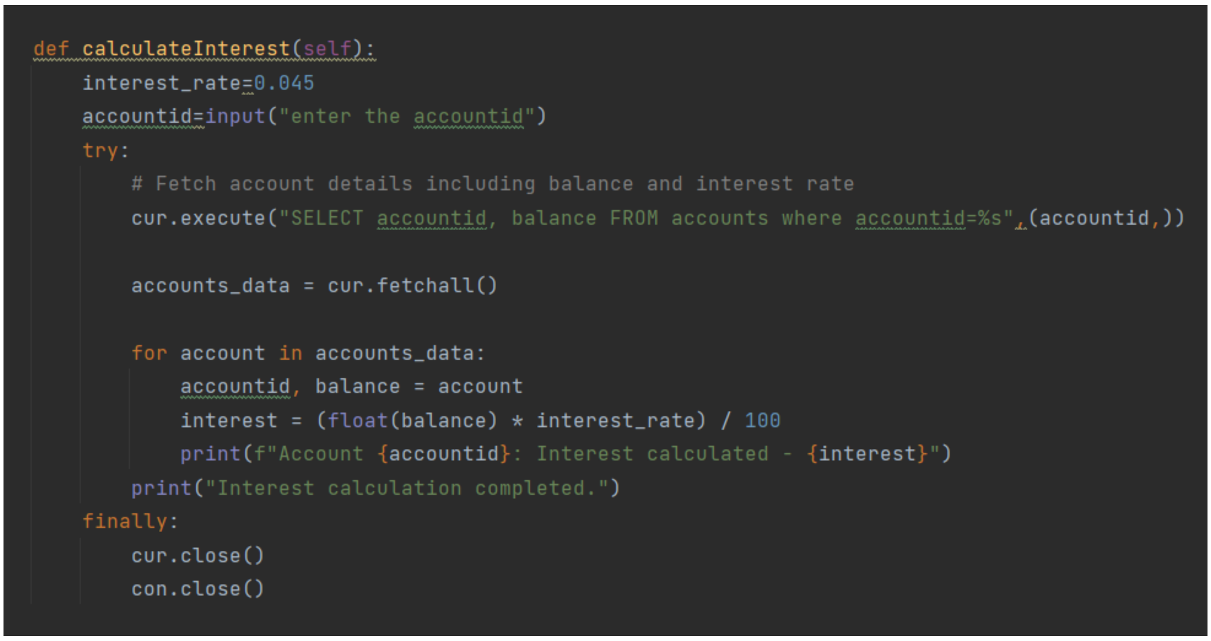
```
def get_all_accounts(self):
    sql="select * from accounts"
    cur.execute(sql)
    user = cur.fetchall()
    for i in user:
        print(i)
```

Output:



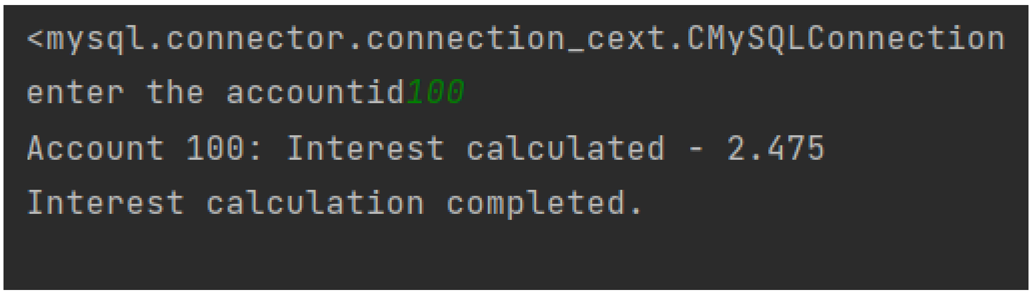
```
(100, 1, 'savings', Decimal('5500.00'))
(101, 2, 'current', Decimal('10050.00'))
(102, 3, 'savings', Decimal('7500.50'))
(103, 4, 'zerobalance', Decimal('200.00'))
(104, 5, 'current', Decimal('12000.75'))
(105, 6, 'savings', Decimal('6500.25'))
(106, 7, 'savings', Decimal('8500.00'))
(107, 8, 'current', Decimal('8100.50'))
(108, 9, 'zerobalance', Decimal('400.00'))
(109, 10, 'savings', Decimal('11000.00'))
(110, 6, 'zerobalance', Decimal('600.00'))
(177, 12, 'savings', Decimal('5500.00'))
```

Calculate interest:



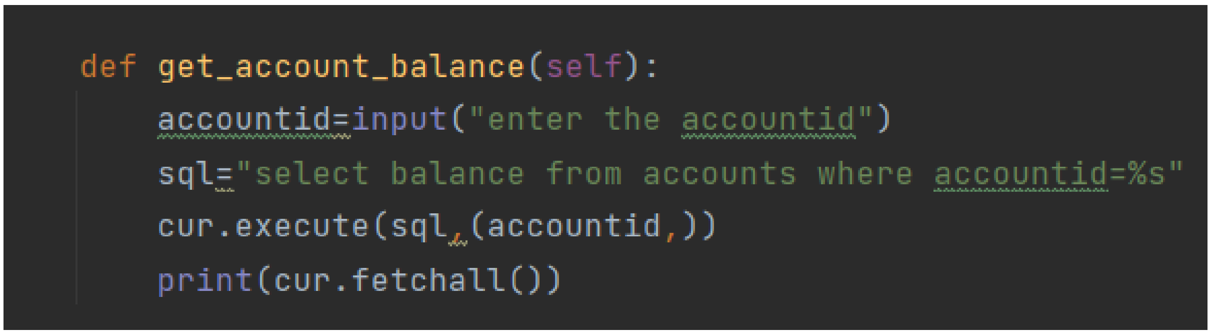
```python
def calculateInterest(self):
    interest_rate=0.045
    accountid=input("enter the accountid")
    try:
        # Fetch account details including balance and interest rate
        cur.execute("SELECT accountid, balance FROM accounts where accountid=%s",(accountid,))

        accounts_data = cur.fetchall()

        for account in accounts_data:
            accountid, balance = account
            interest = (float(balance) * interest_rate) / 100
            print(f"Account {accountid}: Interest calculated - {interest}")
        print("Interest calculation completed.")
    finally:
        cur.close()
        con.close()
```

Output:



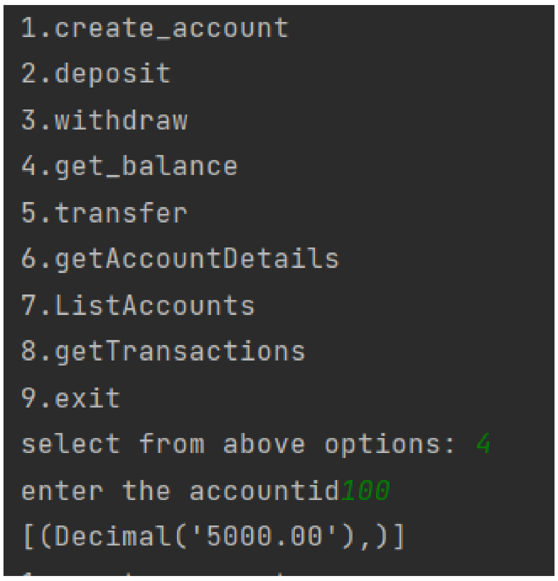
```
<mysql.connector.connection_cext.CMySQLConnection
enter the accountid100
Account 100: Interest calculated - 2.475
Interest calculation completed.
```

Get account Balance:



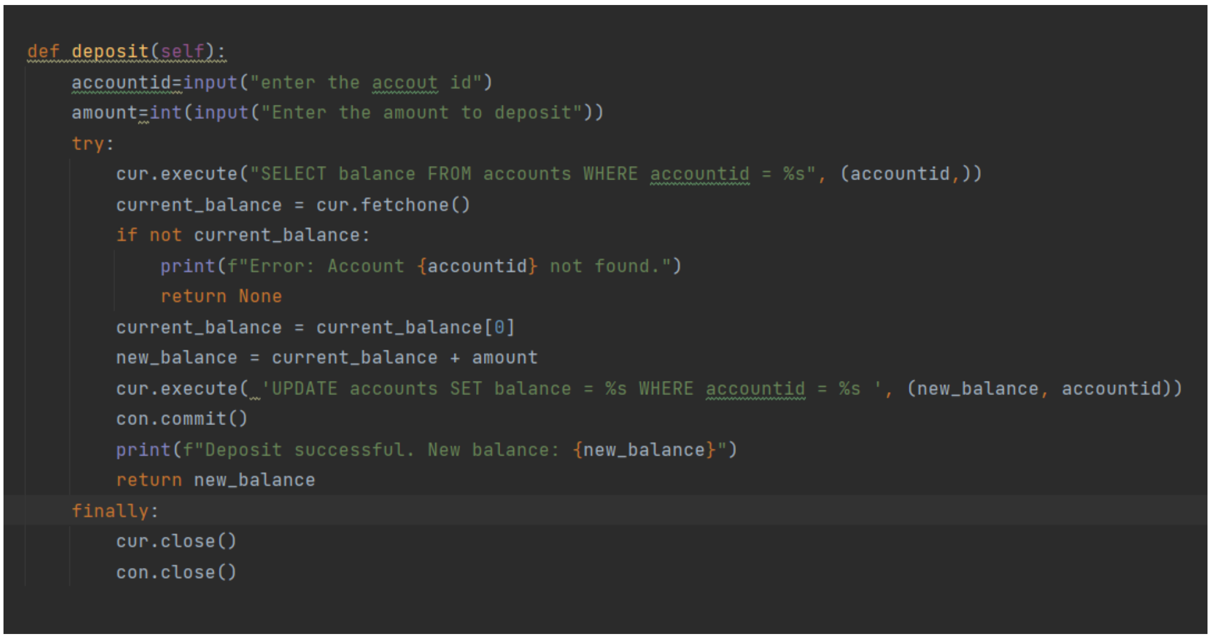
```python
def get_account_balance(self):
    accountid=input("enter the accountid")
    sql="select balance from accounts where accountid=%s"
    cur.execute(sql,(accountid,))
    print(cur.fetchall())
```

Output:



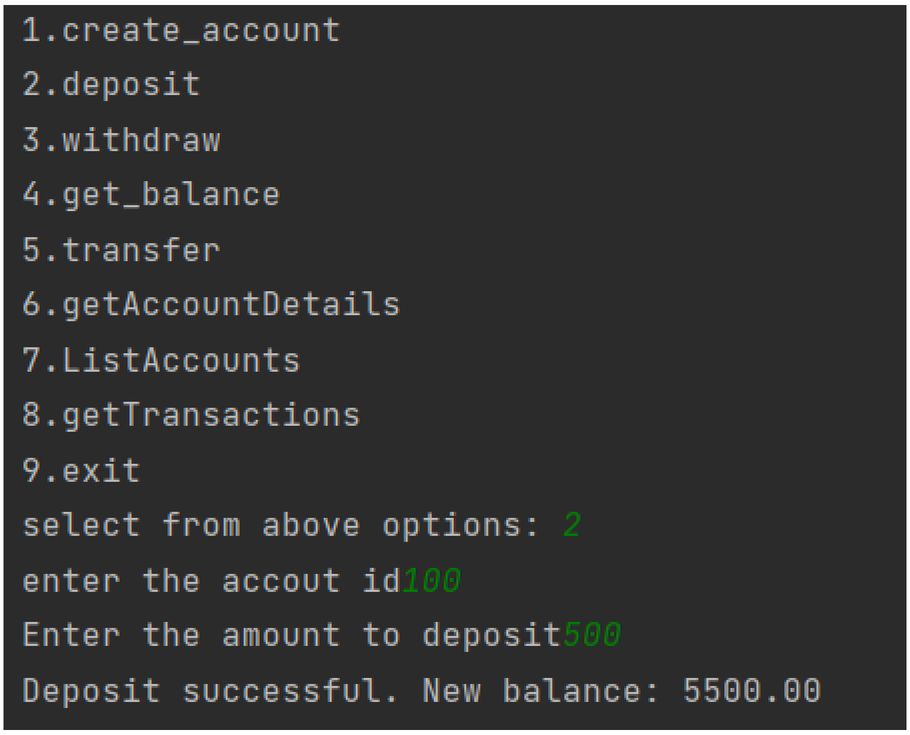
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 4
enter the accountid100
[(Decimal('5000.00'),)]
```

Deposit:



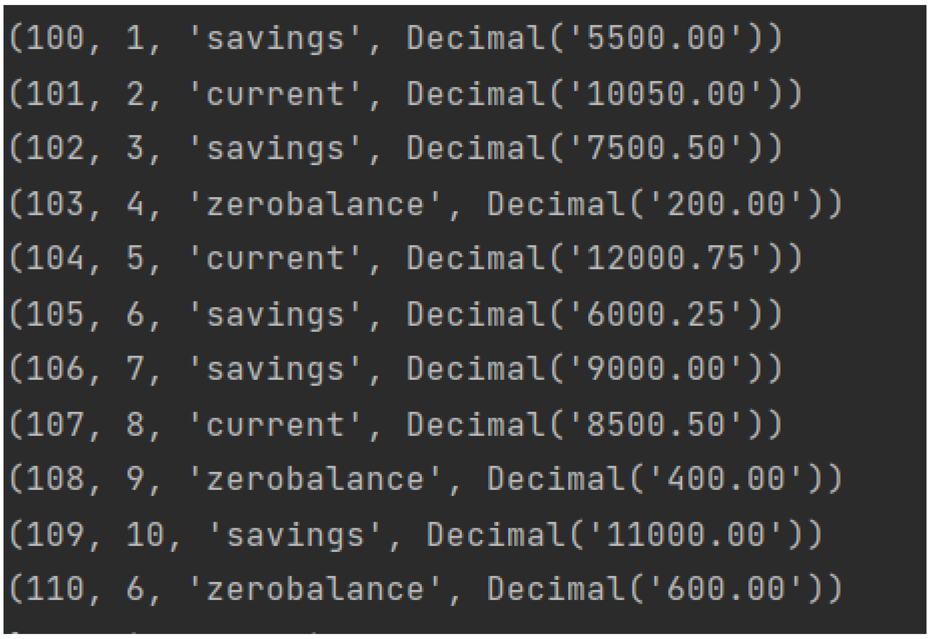
```python
def deposit(self):
    accountid=input("enter the accout id")
    amount=int(input("Enter the amount to deposit"))
    try:
        cur.execute("SELECT balance FROM accounts WHERE accountid = %s", (accountid,))
        current_balance = cur.fetchone()
        if not current_balance:
            print(f"Error: Account {accountid} not found.")
            return None
        current_balance = current_balance[0]
        new_balance = current_balance + amount
        cur.execute( 'UPDATE accounts SET balance = %s WHERE accountid = %s ', (new_balance, accountid))
        con.commit()
        print(f"Deposit successful. New balance: {new_balance}")
        return new_balance
    finally:
        cur.close()
        con.close()
```

Output:



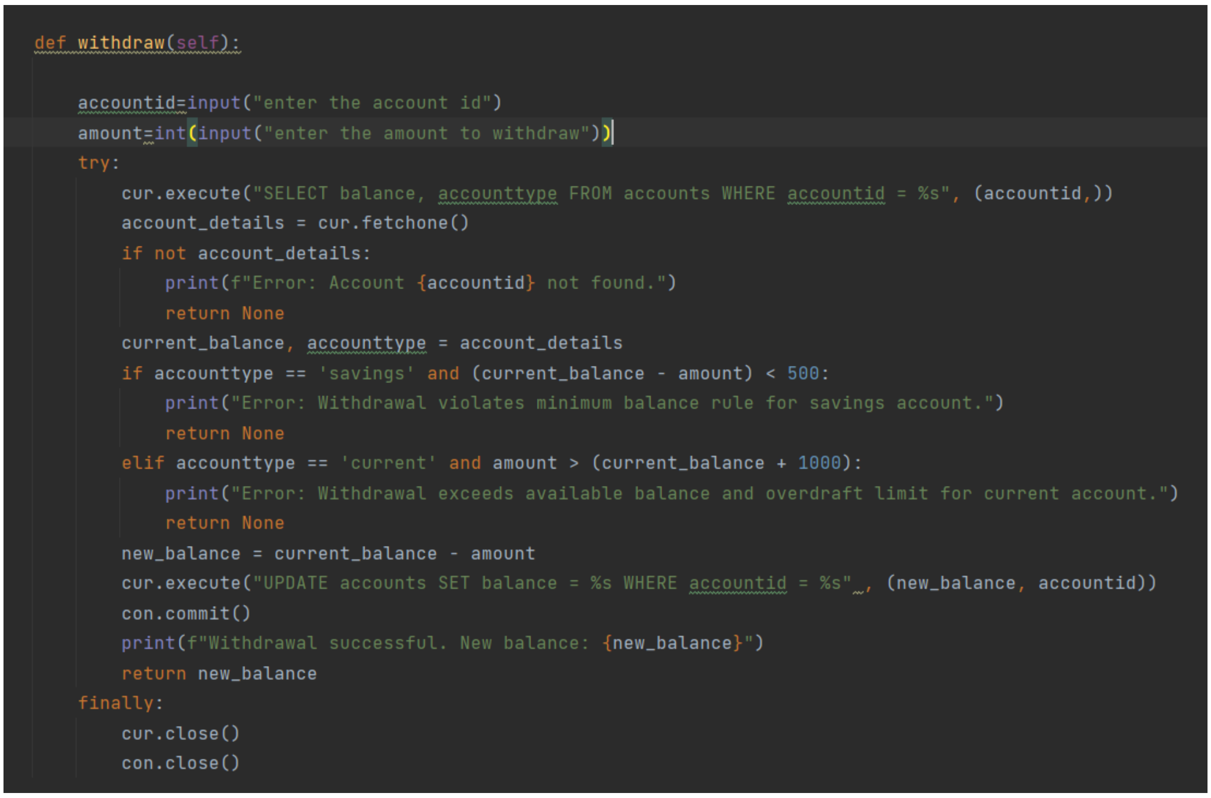
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 2
enter the accout id100
Enter the amount to deposit500
Deposit successful. New balance: 5500.00
```

Accounts Table:



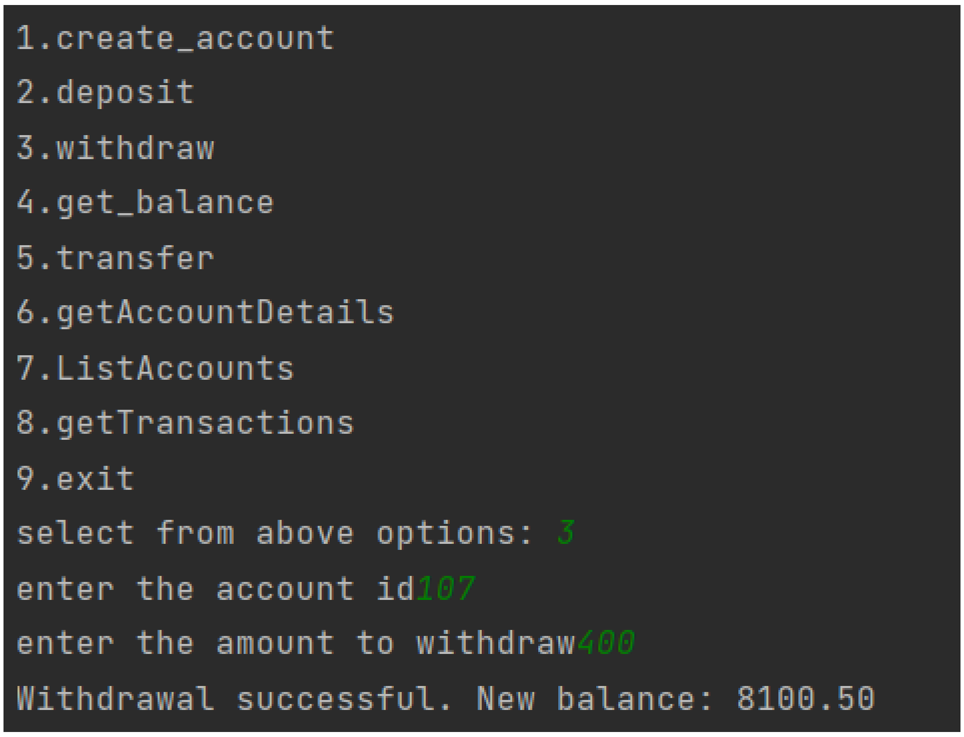
```
(100, 1, 'savings', Decimal('5500.00'))
(101, 2, 'current', Decimal('10050.00'))
(102, 3, 'savings', Decimal('7500.50'))
(103, 4, 'zerobalance', Decimal('200.00'))
(104, 5, 'current', Decimal('12000.75'))
(105, 6, 'savings', Decimal('6000.25'))
(106, 7, 'savings', Decimal('9000.00'))
(107, 8, 'current', Decimal('8500.50'))
(108, 9, 'zerobalance', Decimal('400.00'))
(109, 10, 'savings', Decimal('11000.00'))
(110, 6, 'zerobalance', Decimal('600.00'))
```

Withdraw:



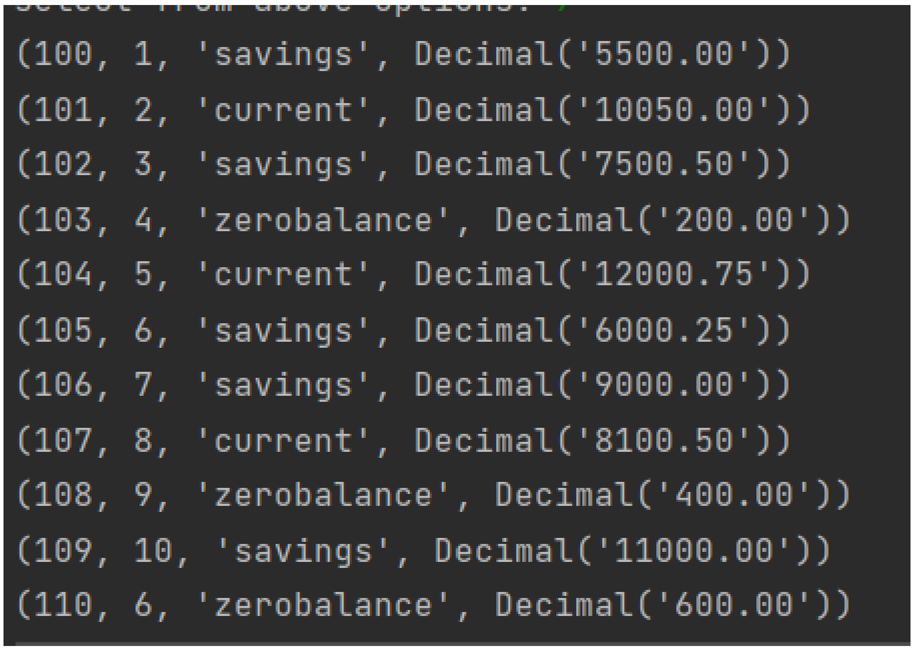
```
def withdraw(self):

    accountid=input("enter the account id")
    amount=int(input("enter the amount to withdraw"))
    try:
        cur.execute("SELECT balance, accounttype FROM accounts WHERE accountid = %s", (accountid,))
        account_details = cur.fetchone()
        if not account_details:
            print(f"Error: Account {accountid} not found.")
            return None
        current_balance, accounttype = account_details
        if accounttype == 'savings' and (current_balance - amount) < 500:
            print("Error: Withdrawal violates minimum balance rule for savings account.")
            return None
        elif accounttype == 'current' and amount > (current_balance + 1000):
            print("Error: Withdrawal exceeds available balance and overdraft limit for current account.")
            return None
        new_balance = current_balance - amount
        cur.execute("UPDATE accounts SET balance = %s WHERE accountid = %s", (new_balance, accountid))
        con.commit()
        print(f"Withdrawal successful. New balance: {new_balance}")
        return new_balance
    finally:
        cur.close()
        con.close()
```

Output:



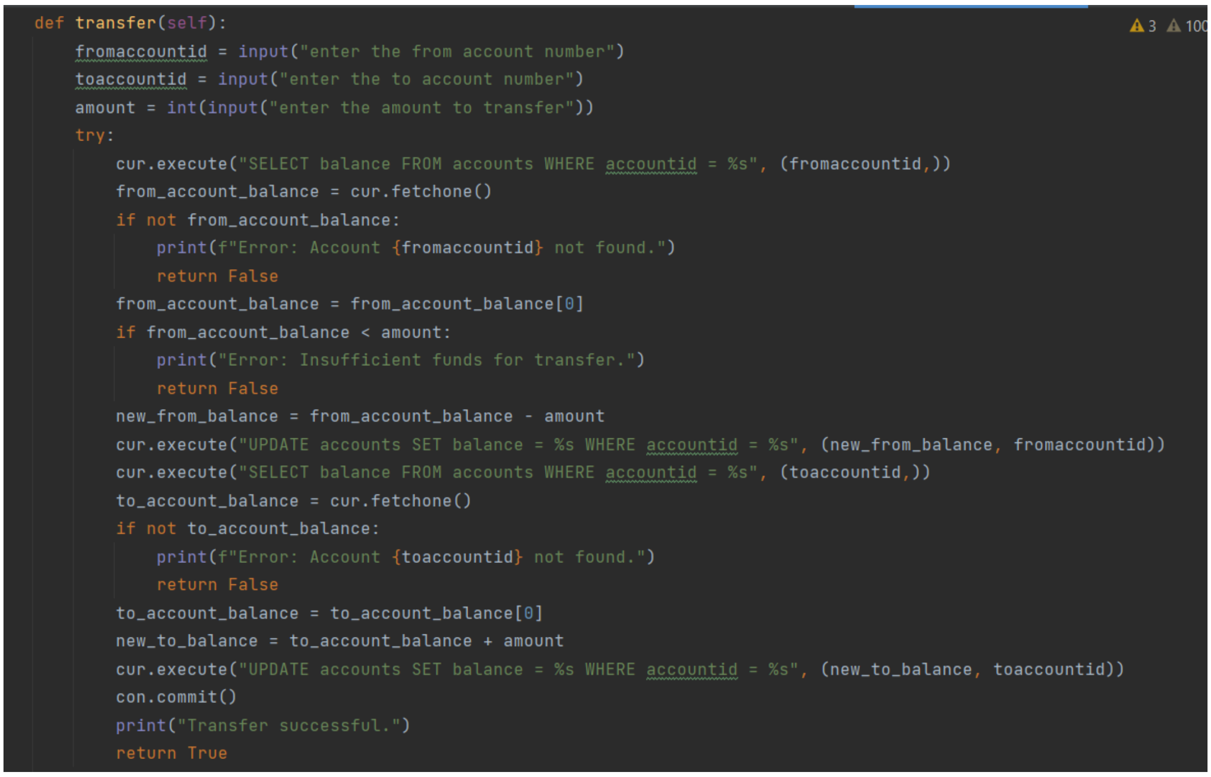
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 3
enter the account id107
enter the amount to withdraw400
Withdrawal successful. New balance: 8100.50
```

Accounts Table:

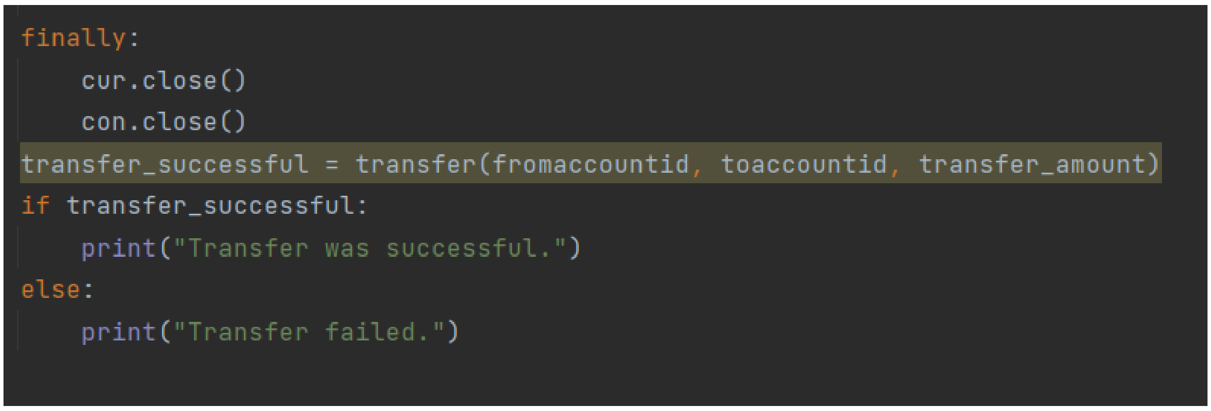


```
select from above options:
(100, 1, 'savings', Decimal('5500.00'))
(101, 2, 'current', Decimal('10050.00'))
(102, 3, 'savings', Decimal('7500.50'))
(103, 4, 'zerobalance', Decimal('200.00'))
(104, 5, 'current', Decimal('12000.75'))
(105, 6, 'savings', Decimal('6000.25'))
(106, 7, 'savings', Decimal('9000.00'))
(107, 8, 'current', Decimal('8100.50'))
(108, 9, 'zerobalance', Decimal('400.00'))
(109, 10, 'savings', Decimal('11000.00'))
(110, 6, 'zerobalance', Decimal('600.00'))
```

Transfer:

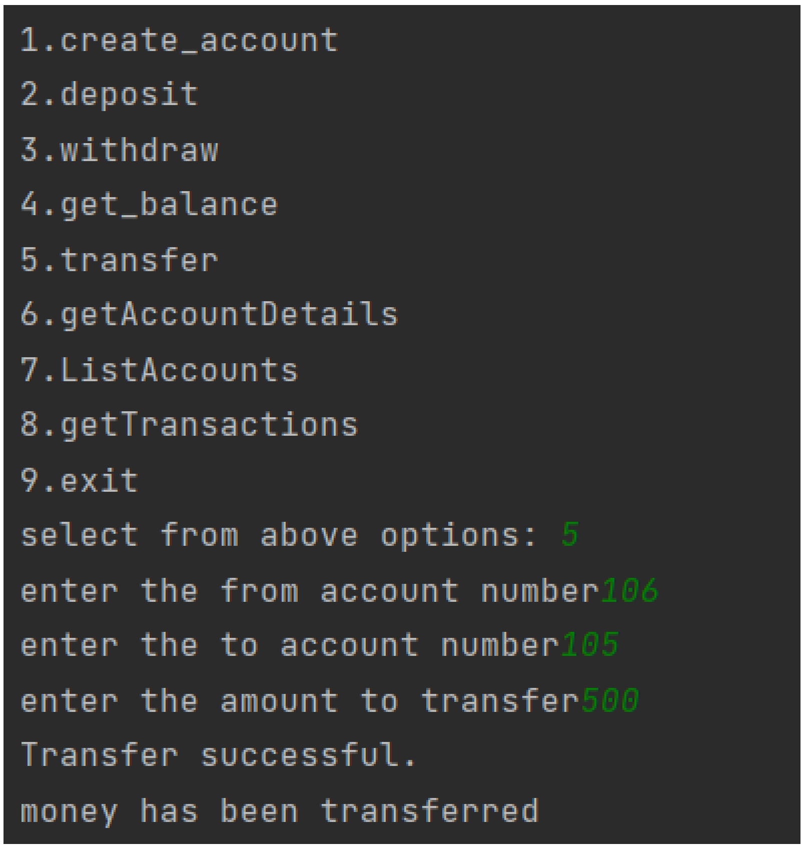


```python
def transfer(self):
    fromaccountid = input("enter the from account number")
    toaccountid = input("enter the to account number")
    amount = int(input("enter the amount to transfer"))
    try:
        cur.execute("SELECT balance FROM accounts WHERE accountid = %s", (fromaccountid,))
        from_account_balance = cur.fetchone()
        if not from_account_balance:
            print(f"Error: Account {fromaccountid} not found.")
            return False
        from_account_balance = from_account_balance[0]
        if from_account_balance < amount:
            print("Error: Insufficient funds for transfer.")
            return False
        new_from_balance = from_account_balance - amount
        cur.execute("UPDATE accounts SET balance = %s WHERE accountid = %s", (new_from_balance, fromaccountid))
        cur.execute("SELECT balance FROM accounts WHERE accountid = %s", (toaccountid,))
        to_account_balance = cur.fetchone()
        if not to_account_balance:
            print(f"Error: Account {toaccountid} not found.")
            return False
        to_account_balance = to_account_balance[0]
        new_to_balance = to_account_balance + amount
        cur.execute("UPDATE accounts SET balance = %s WHERE accountid = %s", (new_to_balance, toaccountid))
        con.commit()
        print("Transfer successful.")
        return True
```



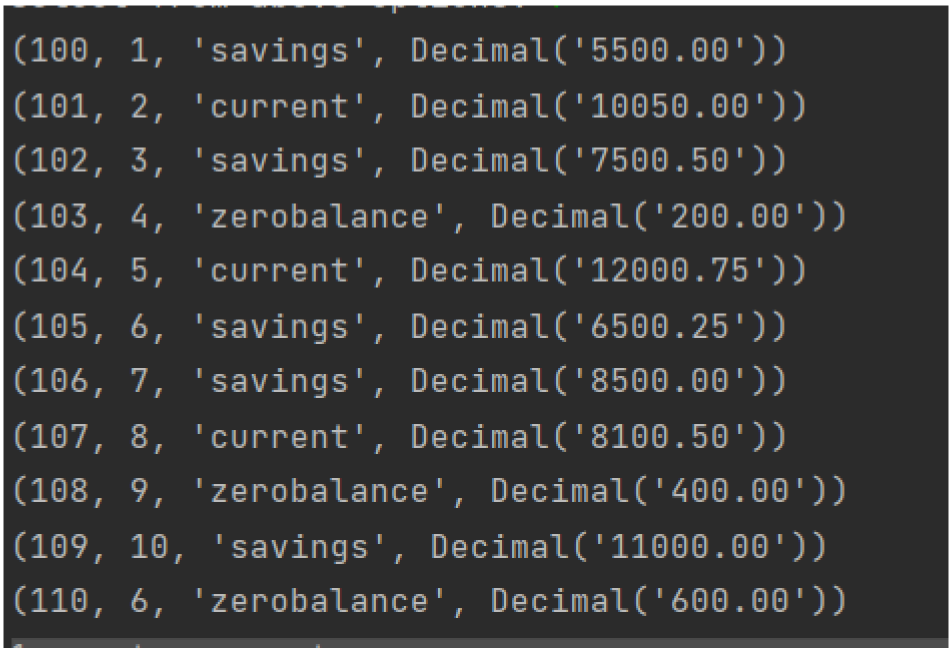
```
finally:
    cur.close()
    con.close()
transfer_successful = transfer(fromaccountid, toaccountid, transfer_amount)
if transfer_successful:
    print("Transfer was successful.")
else:
    print("Transfer failed.")
```

Output:



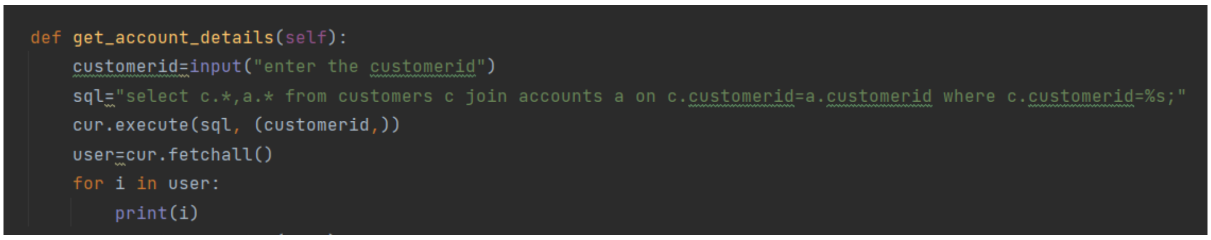
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 5
enter the from account number106
enter the to account number105
enter the amount to transfer500
Transfer successful.
money has been transferred
```

Accounts Table:



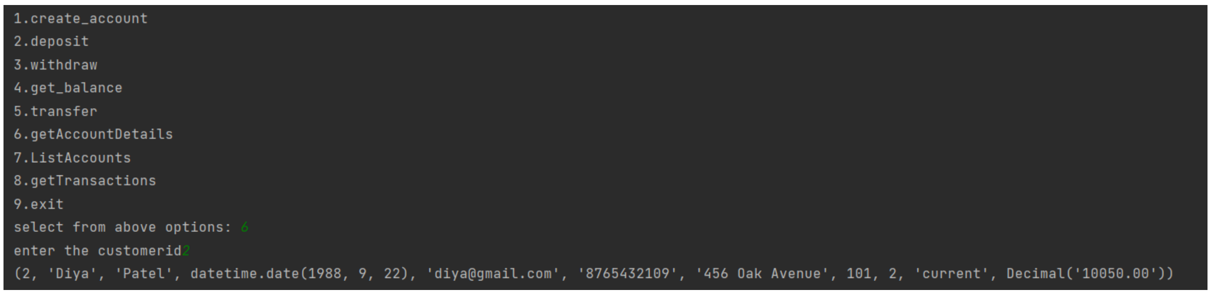
```
(100, 1, 'savings', Decimal('5500.00'))
(101, 2, 'current', Decimal('10050.00'))
(102, 3, 'savings', Decimal('7500.50'))
(103, 4, 'zerobalance', Decimal('200.00'))
(104, 5, 'current', Decimal('12000.75'))
(105, 6, 'savings', Decimal('6500.25'))
(106, 7, 'savings', Decimal('8500.00'))
(107, 8, 'current', Decimal('8100.50'))
(108, 9, 'zerobalance', Decimal('400.00'))
(109, 10, 'savings', Decimal('11000.00'))
(110, 6, 'zerobalance', Decimal('600.00'))
```

Get account details:



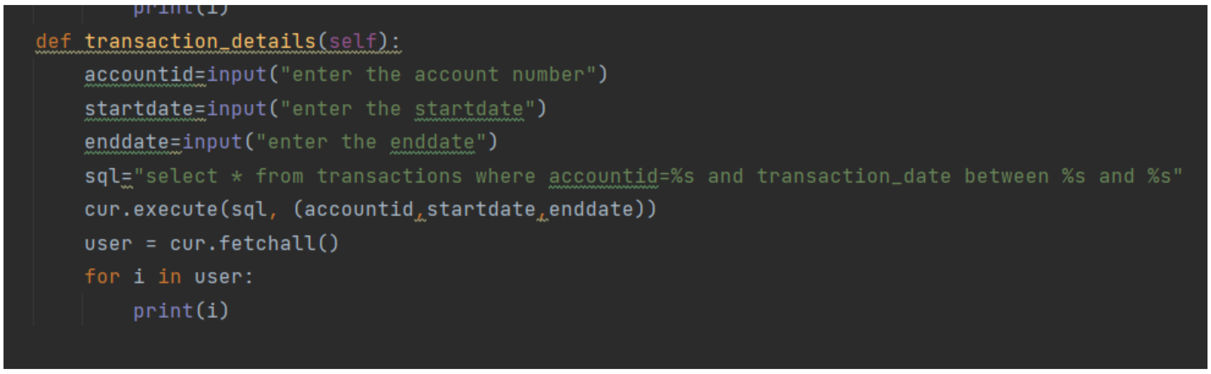
```python
def get_account_details(self):
    customerid=input("enter the customerid")
    sql="select c.*,a.* from customers c join accounts a on c.customerid=a.customerid where c.customerid=%s;"
    cur.execute(sql, (customerid,))
    user=cur.fetchall()
    for i in user:
        print(i)
```

Output:



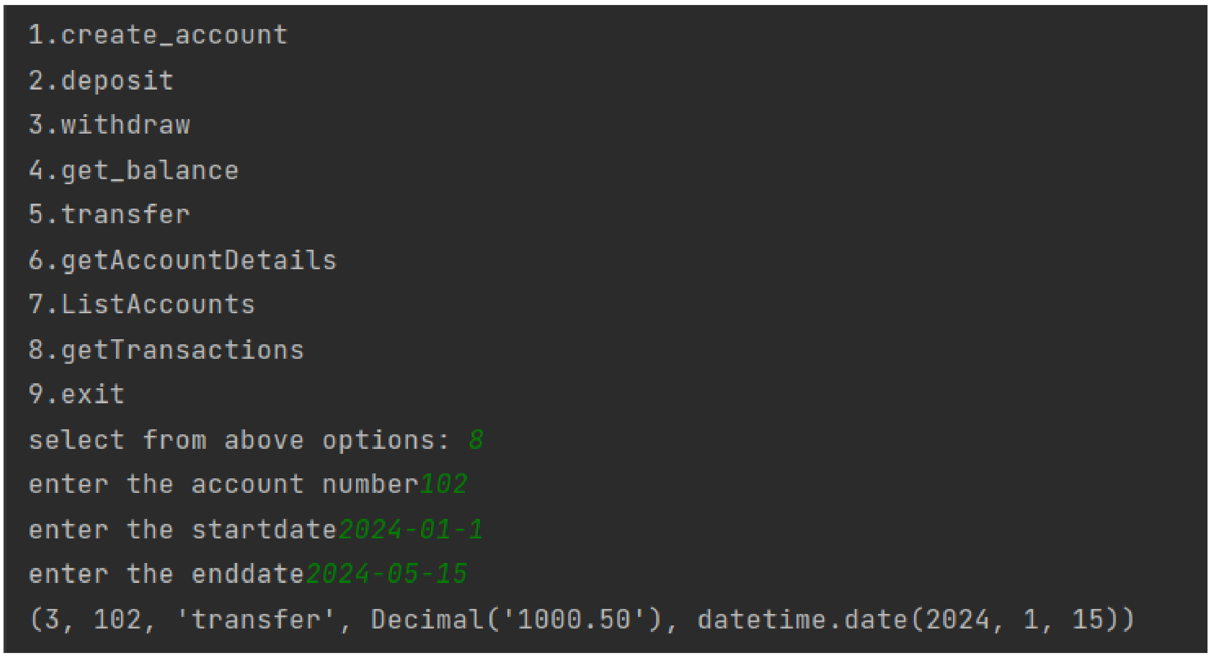
```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 6
enter the customerid2
(2, 'Diya', 'Patel', datetime.date(1988, 9, 22), 'diya@gmail.com', '8765432109', '456 Oak Avenue', 101, 2, 'current', Decimal('10050.00'))
```

Get transactions:



```
def transaction_details(self):
    accountid=input("enter the account number")
    startdate=input("enter the startdate")
    enddate=input("enter the enddate")
    sql="select * from transactions where accountid=%s and transaction_date between %s and %s"
    cur.execute(sql, (accountid,startdate,enddate))
    user = cur.fetchall()
    for i in user:
        print(i)
```

Output:



```
1.create_account
2.deposit
3.withdraw
4.get_balance
5.transfer
6.getAccountDetails
7.ListAccounts
8.getTransactions
9.exit
select from above options: 8
enter the account number102
enter the startdate2024-01-1
enter the enddate2024-05-15
(3, 102, 'transfer', Decimal('1000.50'), datetime.date(2024, 1, 15))
```

Conditional Statements In a bank, you have been given the task is to create a program that checks if a customer is eligible for a loan based on their credit score and income. The eligibility criteria are as follows:

• Credit Score must be above 700.

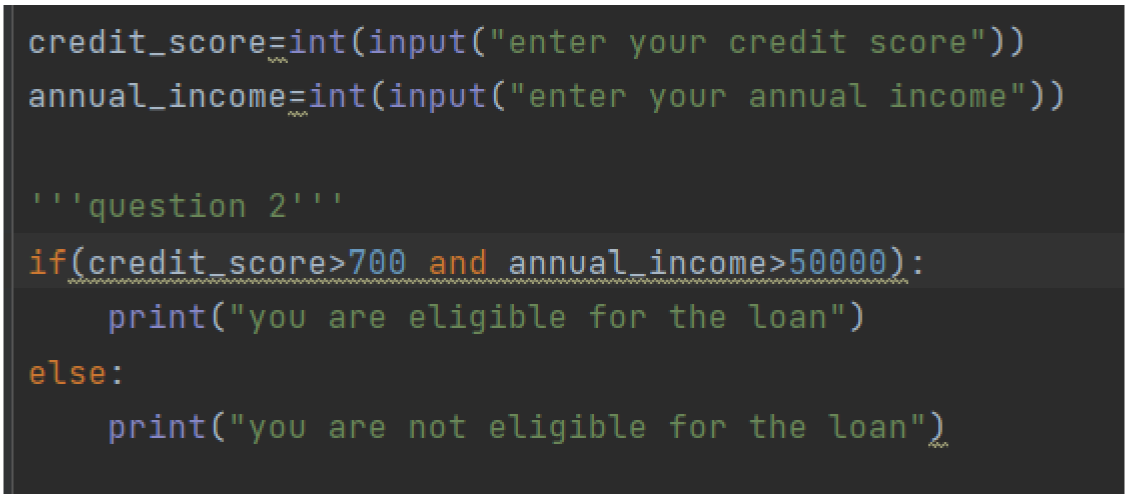
• Annual Income must be at least $50,000.

Tasks:

1. Write a program that takes the customer's credit score and annual income as input.

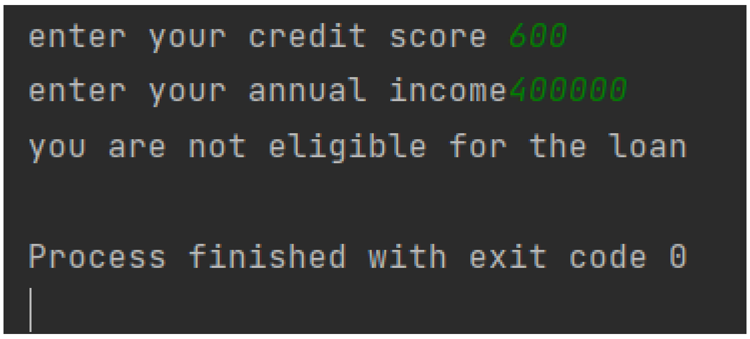
2. Use conditional statements (if-else) to determine if the customer is eligible for a loan.

3. Display an appropriate message based on eligibility.



```
credit_score=int(input("enter your credit score"))
annual_income=int(input("enter your annual income"))

'''question 2'''
if(credit_score>700 and annual_income>50000):
    print("you are eligible for the loan")
else:
    print("you are not eligible for the loan")
```

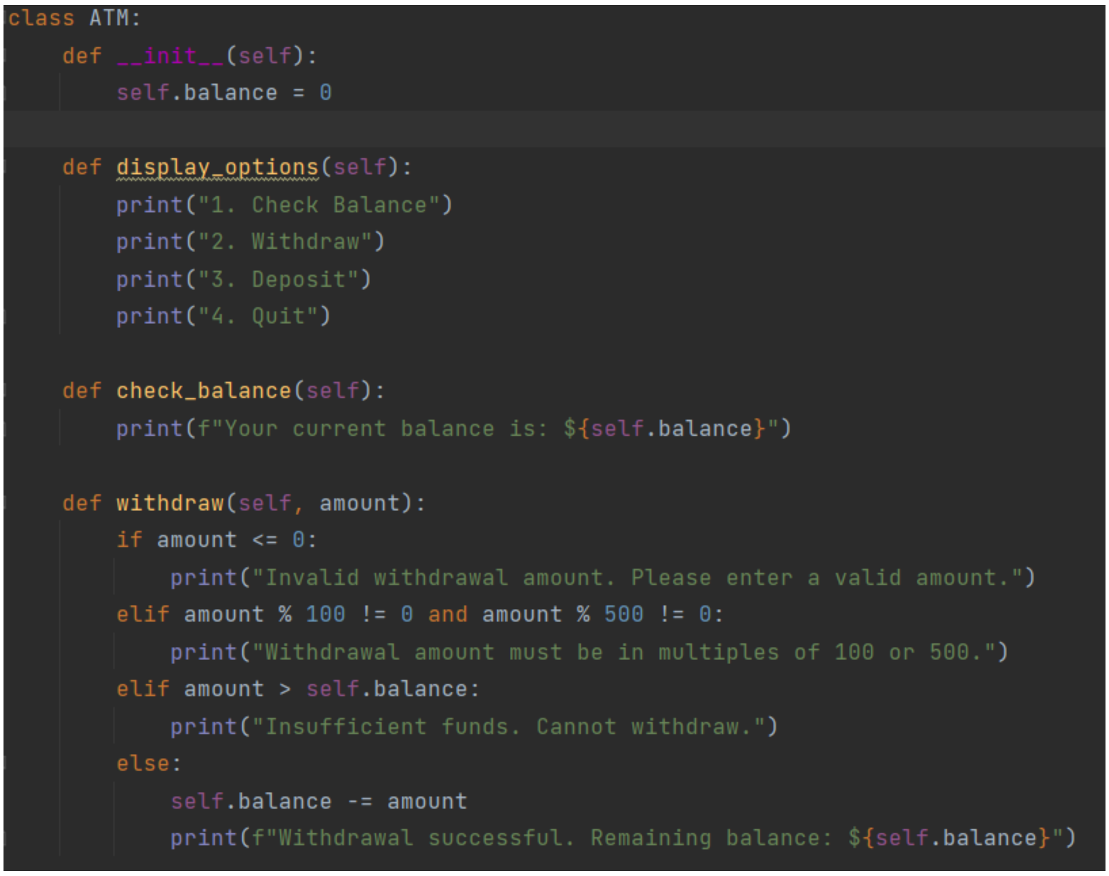
Output:



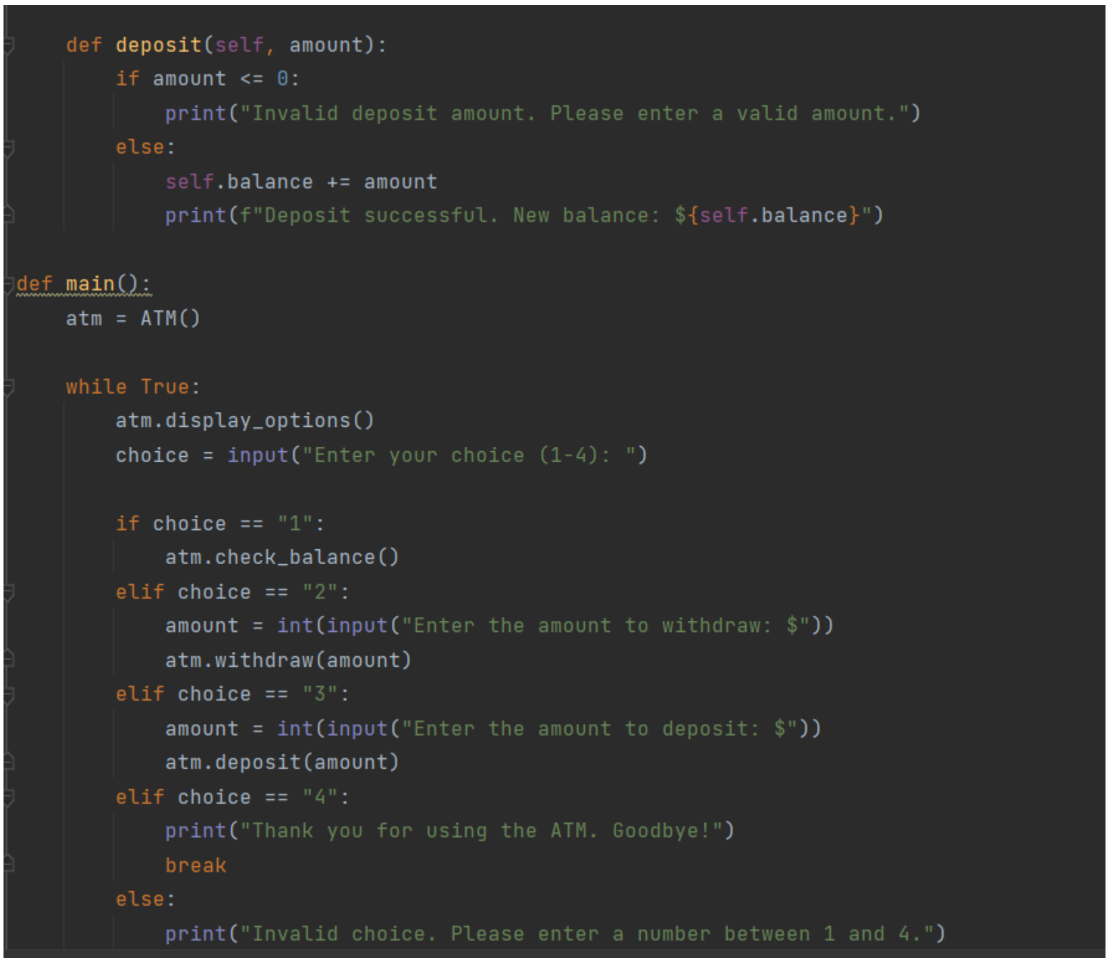
```
enter your credit score 600
enter your annual income400000
you are not eligible for the loan

Process finished with exit code 0
```

Nested Conditional Statements

Create a program that simulates an ATM transaction. Display options such as "Check Balance," "Withdraw," "Deposit,". Ask the user to enter their current balance and the amount they want to withdraw or deposit. Implement checks to ensure that the withdrawal amount is not greater than the available balance and that the withdrawal amount is in multiples of 100 or 500. Display appropriate messages for success or failure.

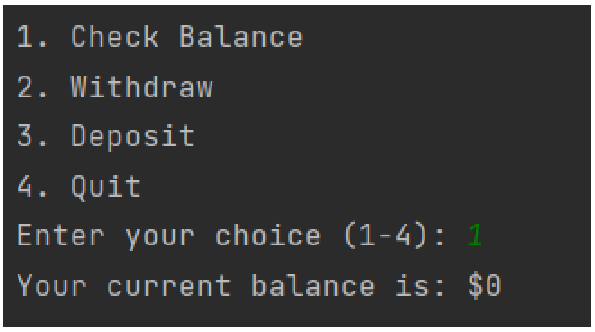


```python
class ATM:
    def __init__(self):
        self.balance = 0

    def display_options(self):
        print("1. Check Balance")
        print("2. Withdraw")
        print("3. Deposit")
        print("4. Quit")

    def check_balance(self):
        print(f"Your current balance is: ${self.balance}")

    def withdraw(self, amount):
        if amount <= 0:
            print("Invalid withdrawal amount. Please enter a valid amount.")
        elif amount % 100 != 0 and amount % 500 != 0:
            print("Withdrawal amount must be in multiples of 100 or 500.")
        elif amount > self.balance:
            print("Insufficient funds. Cannot withdraw.")
        else:
            self.balance -= amount
            print(f"Withdrawal successful. Remaining balance: ${self.balance}")
```

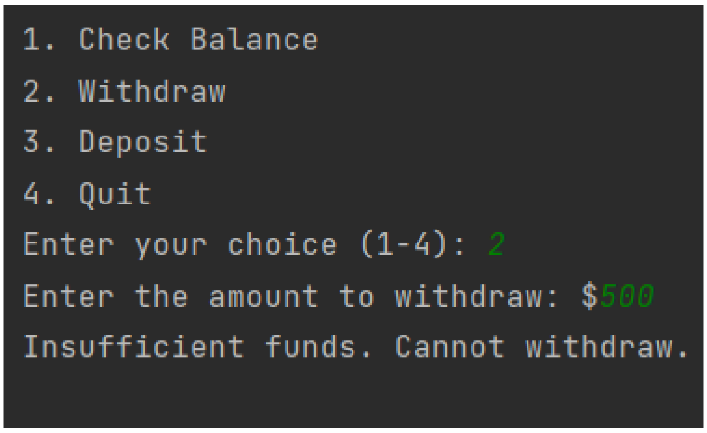


```python
    def deposit(self, amount):
        if amount <= 0:
            print("Invalid deposit amount. Please enter a valid amount.")
        else:
            self.balance += amount
            print(f"Deposit successful. New balance: ${self.balance}")

def main():
    atm = ATM()

    while True:
        atm.display_options()
        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            atm.check_balance()
        elif choice == "2":
            amount = int(input("Enter the amount to withdraw: $"))
            atm.withdraw(amount)
        elif choice == "3":
            amount = int(input("Enter the amount to deposit: $"))
            atm.deposit(amount)
        elif choice == "4":
            print("Thank you for using the ATM. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")
```

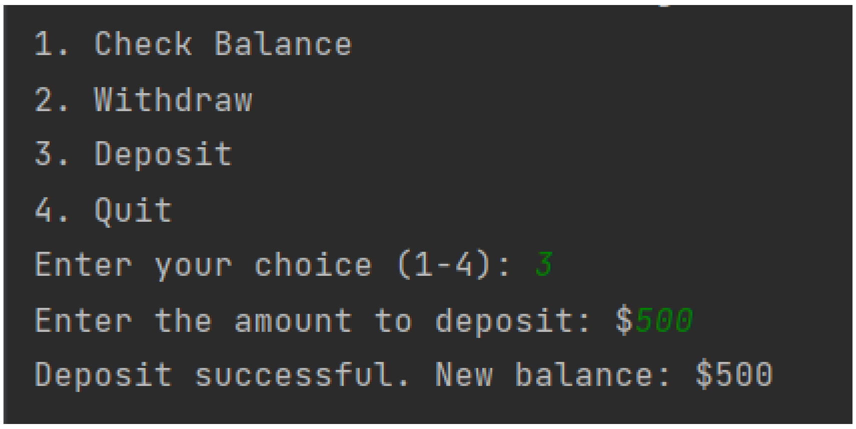
Outputs:



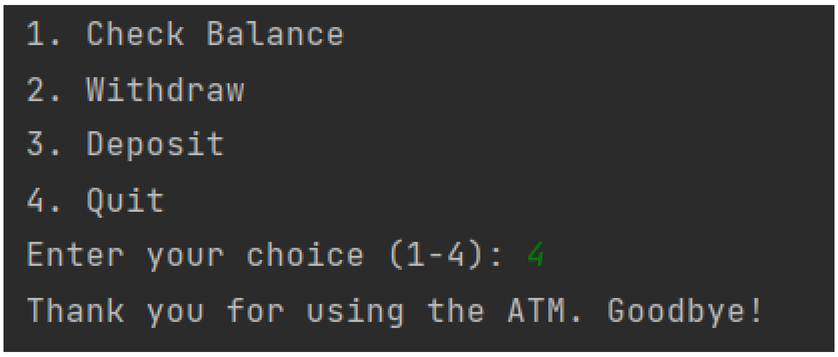
```
1. Check Balance
2. Withdraw
3. Deposit
4. Quit
Enter your choice (1-4): 1
Your current balance is: $0
```



```
1. Check Balance
2. Withdraw
3. Deposit
4. Quit
Enter your choice (1-4): 2
Enter the amount to withdraw: $500
Insufficient funds. Cannot withdraw.
```



```
1. Check Balance
2. Withdraw
3. Deposit
4. Quit
Enter your choice (1-4): 3
Enter the amount to deposit: $500
Deposit successful. New balance: $500
```



```
1. Check Balance
2. Withdraw
3. Deposit
4. Quit
Enter your choice (1-4): 4
Thank you for using the ATM. Goodbye!
```

Loop Structures

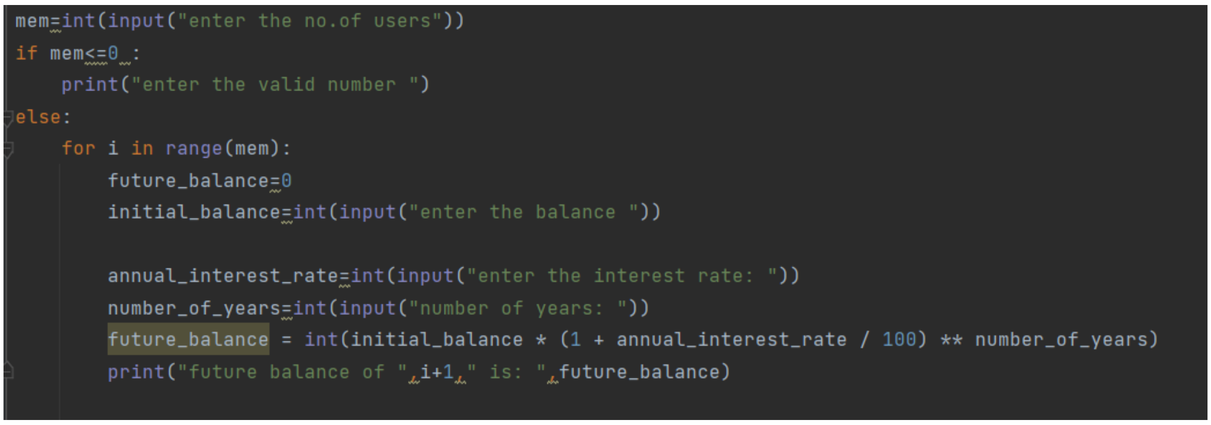
You are responsible for calculating compound interest on savings accounts for bank customers. You need to calculate the future balance for each customer's savings account after a certain number of years.
Tasks:

1. Create a program that calculates the future balance of a savings account.

2. Use a loop structure (e.g., for loop) to calculate the balance for multiple customers.

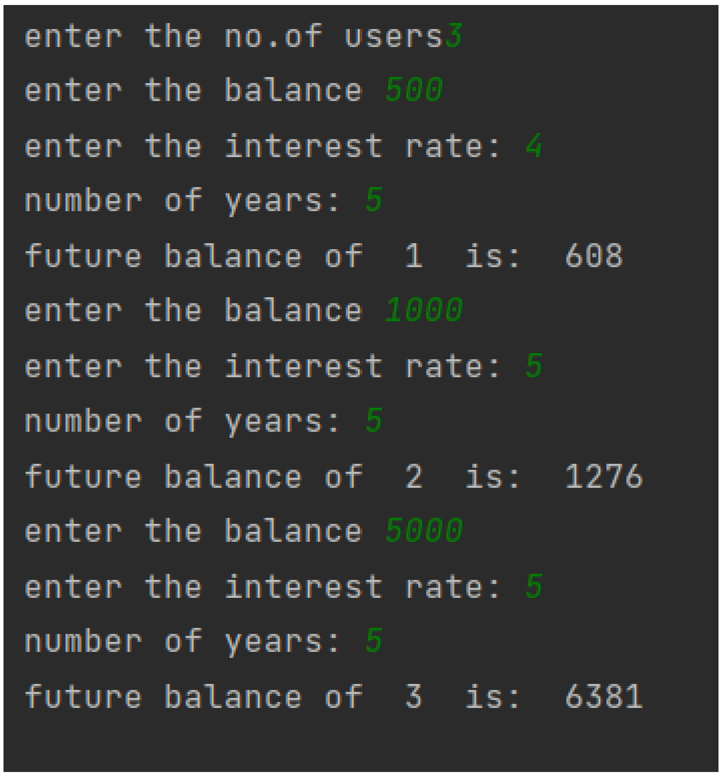
3. Prompt the user to enter the initial balance, annual interest rate, and the number of years.

4. Calculate the future balance using the formula: future_balance = initial_balance * (1 + annual_interest_rate/100)^years. 5. Display the future balance for each customer.



```python
mem=int(input("enter the no.of users"))
if mem<=0 :
    print("enter the valid number ")
else:
    for i in range(mem):
        future_balance=0
        initial_balance=int(input("enter the balance "))

        annual_interest_rate=int(input("enter the interest rate: "))
        number_of_years=int(input("number of years: "))
        future_balance = int(initial_balance * (1 + annual_interest_rate / 100) ** number_of_years)
        print("future balance of ",i+1," is: ",future_balance)
```

Output:



```
enter the no.of users3
enter the balance 500
enter the interest rate: 4
number of years: 5
future balance of  1  is:  608
enter the balance 1000
enter the interest rate: 5
number of years: 5
future balance of  2  is:  1276
enter the balance 5000
enter the interest rate: 5
number of years: 5
future balance of  3  is:  6381
```

Looping, Array and Data Validation

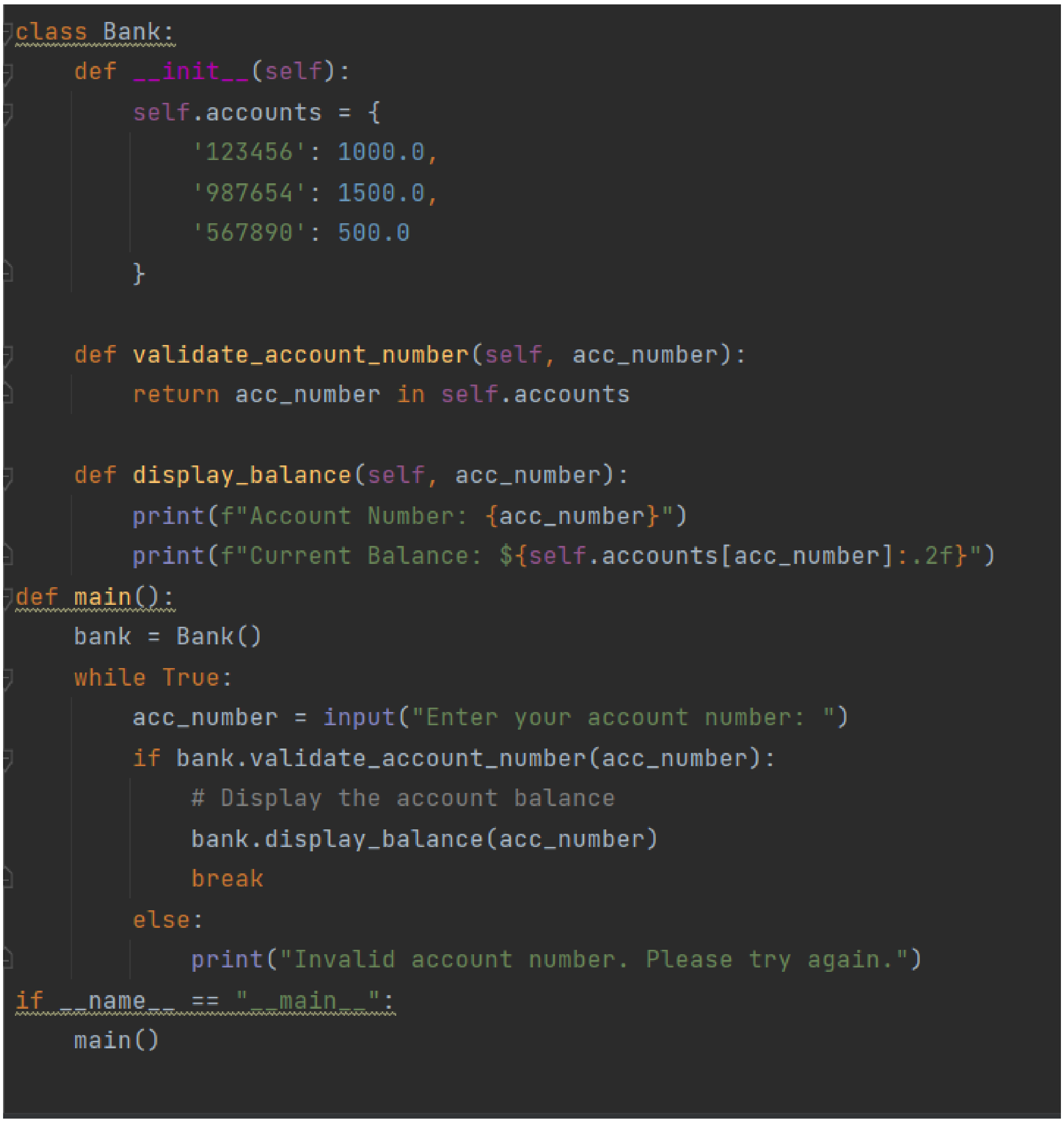
You are tasked with creating a program that allows bank customers to check their account balances. The program should handle multiple customer accounts, and the customer should be able to enter their account number, balance to check the balance.

Tasks:

1. Create a Python program that simulates a bank with multiple customer accounts.

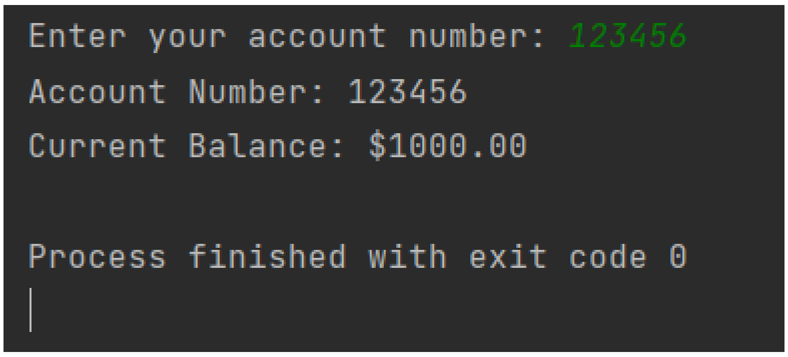
2. Use a loop (e.g., while loop) to repeatedly ask the user for their account number and balance until they enter a valid account number.
3. Validate the account number entered by the user.

4. If the account number is valid, display the account balance. If not, ask the user to try again.

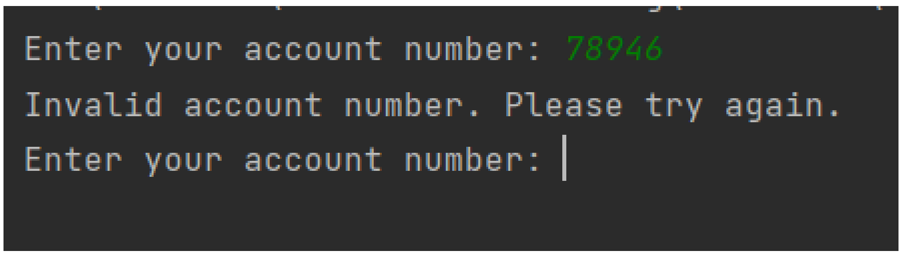


```python
class Bank:
    def __init__(self):
        self.accounts = {
            '123456': 1000.0,
            '987654': 1500.0,
            '567890': 500.0
        }

    def validate_account_number(self, acc_number):
        return acc_number in self.accounts

    def display_balance(self, acc_number):
        print(f"Account Number: {acc_number}")
        print(f"Current Balance: ${self.accounts[acc_number]:.2f}")
def main():
    bank = Bank()
    while True:
        acc_number = input("Enter your account number: ")
        if bank.validate_account_number(acc_number):
            # Display the account balance
            bank.display_balance(acc_number)
            break
        else:
            print("Invalid account number. Please try again.")
if __name__ == "__main__":
    main()
```

Output:



```
Enter your account number: 123456
Account Number: 123456
Current Balance: $1000.00

Process finished with exit code 0
```



```
Enter your account number: 78946
Invalid account number. Please try again.
Enter your account number: 
```

Password Validation

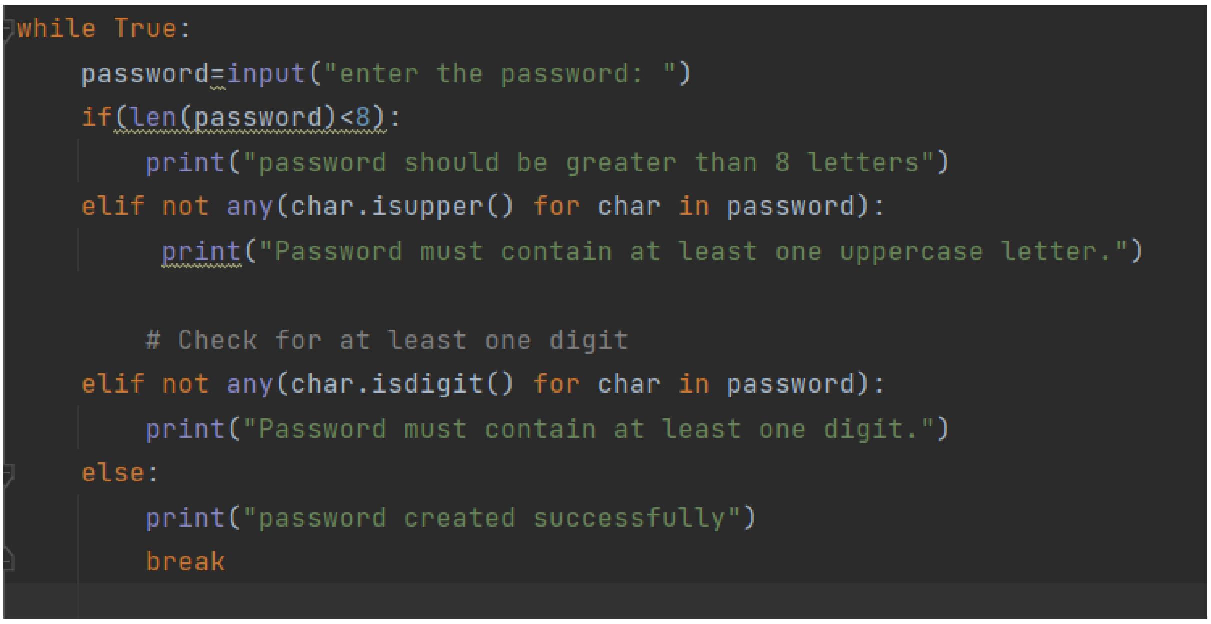
Write a program that prompts the user to create a password for their bank account. Implement if conditions to validate the password according to these rules:

• The password must be at least 8 characters long.

• It must contain at least one uppercase letter.

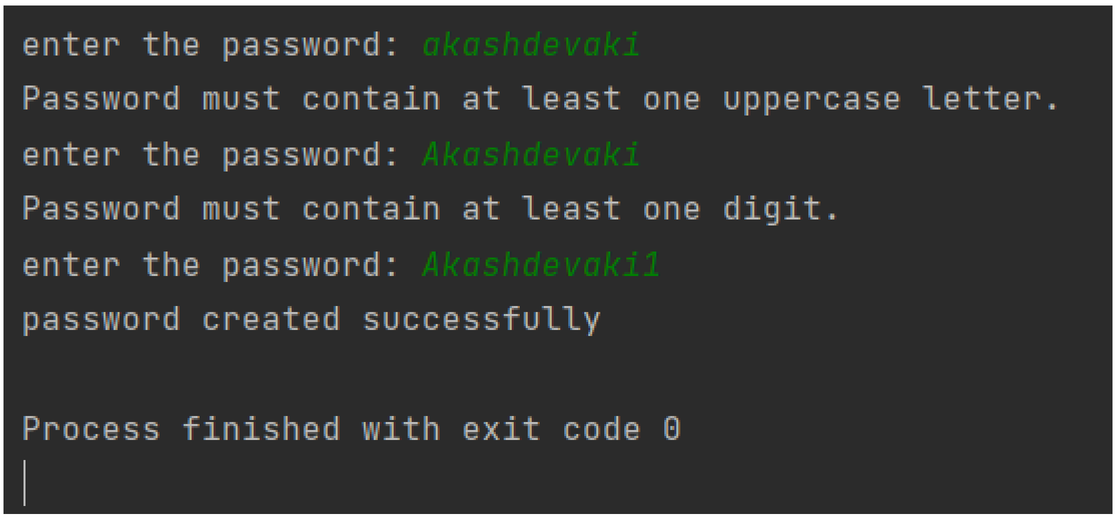
• It must contain at least one digit.

• Display appropriate messages to indicate whether their password is valid or not.



```python
while True:
    password=input("enter the password: ")
    if(len(password)<8):
        print("password should be greater than 8 letters")
    elif not any(char.isupper() for char in password):
         print("Password must contain at least one uppercase letter.")

        # Check for at least one digit
    elif not any(char.isdigit() for char in password):
        print("Password must contain at least one digit.")
    else:
        print("password created successfully")
        break
```

Output:



```
enter the password: akashdevaki
Password must contain at least one uppercase letter.
enter the password: Akashdevaki
Password must contain at least one digit.
enter the password: Akashdevaki1
password created successfully

Process finished with exit code 0
```

Submitted By:

Devaki Akash