# ASSIGNMENT 3

# BANKING SYSTEM

**AKILESH K**

## Task1

```
def check_loan_eligibility(credit_score, annual_income):


    credit_score_threshold = 700

    income_threshold = 50000



    if credit_score > credit_score_threshold and annual_income >= income_threshold:

        print("Congratulations! You are eligible for a loan.")

    else:

        print("Sorry, you are not eligible for a loan at this time.")


credit_score_input = int(input("Enter your credit score: "))

annual_income_input = float(input("Enter your annual income: $"))


check_loan_eligibility(credit_score_input, annual_income_input)
```

```
PS C:\Users\Akilesh K\OneDrive\Documents\python class\co
hallenge\petpals'; & 'C:\Users\Akilesh K\AppData\Local\P
-2023.22.1\pythonFiles\lib\python\debugpy\adapter/../..\
sign\hm bank\bank.py'
Enter your credit score: 800
Enter your annual income: $100000
Congratulations! You are eligible for a loan.
```

```
PS C:\Users\Akilesh K\OneDrive\Documents\python class\coding ch
hallenge\petpals'; & 'C:\Users\Akilesh K\AppData\Local\Programs
-2023.22.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy
sign\hm bank\bank.py'
Enter your credit score: 600
Enter your annual income: $40000
Sorry, you are not eligible for a loan at this time.
```

## Task 2

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

    def check_balance(self):
        print(f"Current Balance: ${self.balance:.2f}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds. Withdrawal failed.")
        elif amount % 100 != 0 or amount % 500 != 0:
            print("Withdrawal amount must be in multiples of 100 or 500. Withdrawal failed.")
        else:
            self.balance -= amount
            print(f"Withdrawal successful. New Balance: ${self.balance:.2f}")

    def deposit(self, amount):
        if amount <= 0:
            print("Deposit amount must be greater than zero. Deposit failed.")
        else:
            self.balance += amount
            print(f"Deposit successful. New Balance: ${self.balance:.2f}")
```

```
def main():
    initial_balance = float(input("Enter your current balance: $"))
    atm = ATM(initial_balance)

    while True:
        print("\nATM Options:")
        print("1. Check Balance")
        print("2. Withdraw")
        print("3. Deposit")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            atm.check_balance()
        elif choice == "2":
            amount_to_withdraw = float(input("Enter the amount to withdraw: $"))
            atm.withdraw(amount_to_withdraw)
        elif choice == "3":
            amount_to_deposit = float(input("Enter the amount to deposit: $"))
            atm.deposit(amount_to_deposit)
        elif choice == "4":
            print("Exiting ATM. Thank you!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

```
Enter your current balance: $1000

ATM Options:
1. Check Balance
2. Withdraw
3. Deposit
4. Exit
Enter your choice (1-4): 1
Current Balance: $1000.00

ATM Options:
1. Check Balance
2. Withdraw
3. Deposit
4. Exit
Enter your choice (1-4): 2
Enter the amount to withdraw: $200
Withdrawal amount must be in multiples of 100 or 500. Withdrawal failed.
```

```
ATM Options:
1. Check Balance
2. Withdraw
3. Deposit
4. Exit
Enter your choice (1-4): 3
Enter the amount to deposit: $300
Deposit successful. New Balance: $1300.00

ATM Options:
1. Check Balance
2. Withdraw
3. Deposit
4. Exit
Enter your choice (1-4): 2
Enter the amount to withdraw: $500
Withdrawal successful. New Balance: $800.00
```

## Task 3

```
def calculate_future_balance(initial_balance, annual_interest_rate, years):
    # Calculate future balance using compound interest formula
    future_balance = initial_balance * (1 + annual_interest_rate / 100) ** years
    return future_balance
```

```
def main():
    num_customers = int(input("Enter the number of customers: "))

    for customer in range(1, num_customers + 1):
        print(f"\nCustomer {customer}:")
        initial_balance = float(input("Enter the initial balance: $"))
        annual_interest_rate = float(input("Enter the annual interest rate (%): "))
        years = int(input("Enter the number of years: "))

        # Calculate and display future balance
        future_balance = calculate_future_balance(initial_balance, annual_interest_rate, years)
        print(f"Future Balance for Customer {customer}: ${future_balance:.2f}")

if __name__ == "__main__":
    main()
```

```
Enter the number of customers: 2

Customer 1:
Enter the initial balance: $500
Enter the annual interest rate (%): 10
Enter the number of years: 3
Future Balance for Customer 1: $665.50

Customer 2:
Enter the initial balance: $1000
Enter the annual interest rate (%): 8
Enter the number of years: 5
Future Balance for Customer 2: $1469.33
```

## Task 4

```
class Bank:
    def __init__(self):
```

```
        # Sample customer accounts (replace with actual customer data)
        self.customer_accounts = {
            "123456": 1000.50,
            "789012": 500.25,
            "345678": 1500.75
        }

    def check_balance(self, account_number):
        if account_number in self.customer_accounts:
            return self.customer_accounts[account_number]
        else:
            return None

def main():
    bank = Bank()

    while True:
        account_number = input("Enter your account number (or 'exit' to quit): ")

        if account_number.lower() == 'exit':
            print("Exiting the bank. Thank you!")
            break

        # Validate the account number
        balance = bank.check_balance(account_number)
        if balance is not None:
            print(f"Account Balance for Account {account_number}: ${balance:.2f}")
        else:
            print("Invalid account number. Please try again.")

if __name__ == "__main__":
```

```
    main()
```

```
Enter your account number (or 'exit' to quit): 123456
Account Balance for Account 123456: $1000.50
Enter your account number (or 'exit' to quit): 5252462
Invalid account number. Please try again.
Enter your account number (or 'exit' to quit): 
```

## Task 5

```
def validate_password(password):


    if len(password) < 8:

        return False, "Password must be at least 8 characters long."



    if not any(char.isupper() for char in password):

        return False, "Password must contain at least one uppercase letter."



    if not any(char.isdigit() for char in password):

        return False, "Password must contain at least one digit."



    return True, "Password is valid."


def main():

    while True:

        user_password = input("Create a password for your bank account: ")



        is_valid, message = validate_password(user_password)
```

```
        print(message)


        if is_valid:
            break


if __name__ == "__main__":
    main()
```

```
Create a password for your bank account: will2274jacab
Password must contain at least one uppercase letter.
Create a password for your bank account: willja
Password must be at least 8 characters long.
Create a password for your bank account: willjacab
Password must contain at least one uppercase letter.
Create a password for your bank account: WILL2203jacab
Password is valid.
```

## Task 6

```
class BankAccount:
    def __init__(self):
        self.transaction_history = []

    def deposit(self, amount):
        self.transaction_history.append(("Deposit", amount))

    def withdraw(self, amount):
        self.transaction_history.append(("Withdrawal", amount))

    def display_transaction_history(self):
        print("\nTransaction History:")
        for transaction_type, amount in self.transaction_history:
```

```
        print(f"{transaction_type}: ${amount:.2f}")

def main():
    account = BankAccount()

    while True:
        print("\nOptions:")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Display Transaction History")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            deposit_amount = float(input("Enter the deposit amount: $"))
            account.deposit(deposit_amount)
        elif choice == "2":
            withdrawal_amount = float(input("Enter the withdrawal amount: $"))
            account.withdraw(withdrawal_amount)
        elif choice == "3":
            account.display_transaction_history()
        elif choice == "4":
            print("Exiting. Transaction History:")
            account.display_transaction_history()
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()
```

```
Options:
1. Deposit
2. Withdraw
3. Display Transaction History
4. Exit
Enter your choice (1-4): 1
Enter the deposit amount: $100

Options:
1. Deposit
2. Withdraw
3. Display Transaction History
4. Exit
Enter your choice (1-4): 2
Enter the withdrawal amount: $500

Options:
1. Deposit
2. Withdraw
3. Display Transaction History
4. Exit
Enter your choice (1-4): 3

Transaction History:
Deposit: $100.00
Withdrawal: $500.00
```

## Task 7

```
class Customer:

    def __init__(self, customer_id=None, first_name=None, last_name=None, email=None,
phone_number=None, address=None):

        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
```

```
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
```

## Task 8

```
class SavingsAccount(Account):
    def __init__(self, account_number, account_balance, interest_rate):
        super().__init__(account_number, "Savings", account_balance)
        self.interest_rate = interest_rate

class CurrentAccount(Account):
    OVERDRAFT_LIMIT = 1000  # Example overdraft limit

    def __init__(self, account_number, account_balance):
        super().__init__(account_number, "Current", account_balance


def create_account():
    print("\nChoose Account Type:")
    print("1. SavingsAccount")
    print("2. CurrentAccount")

    choice = input("Enter your choice (1 or 2): ")

    if choice == "1":
        account_number = input("Enter account number: ")
        balance = float(input("Enter initial balance: $"))
        interest_rate = float(input("Enter interest rate: "))
```

```
        return SavingsAccount(account_number, balance, interest_rate)
    elif choice == "2":
        account_number = input("Enter account number: ")
        balance = float(input("Enter initial balance: $"))
        return CurrentAccount(account_number, balance)
    else:
        print("Invalid choice. Creating a default SavingsAccount.")
        return SavingsAccount("DefaultSavings", 0.0, 2.0)
```

## TASK 9

```
class BankAccount(ABC):
    def __init__(self, account_number=None, customer_name=None, balance=0.0):
        self.account_number = account_number
        self.customer_name = customer_name
        self.balance = balance

    @abstractmethod
    def calculate_interest(self):
        pass

    @abstractmethod
    def withdraw(self, amount):
        pass
```

## TASK 10

```
def main(self):
        while True:
            print("\nBanking System Commands:")
```

```
        print("1. Create Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Get Balance")
        print("5. Transfer")
        print("6. Get Account Details")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")

        if choice == "1":
            self.create_account()
        elif choice == "2":
            self.deposit()
        elif choice == "3":
            self.withdraw()
        elif choice == "4":
            self.get_balance()
        elif choice == "5":
            self.transfer()
        elif choice == "6":
            self.get_account_details()
        elif choice == "7":
            print("Exiting BankApp. Thank you!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    bank_app = BankApp()
    bank_app.main()
```

## TASK 11

```
class ICustomerServiceProvider(ABC):
    @abstractmethod
    def get_account_balance(self, account_number):
        pass

    @abstractmethod
    def deposit(self, account_number, amount):
        pass

    @abstractmethod
    def withdraw(self, account_number, amount):
        pass

    @abstractmethod
    def transfer(self, from_account_number, to_account_number, amount):
        pass

    @abstractmethod
    def get_account_details(self, account_number):
        pass

class IBankServiceProvider(ICustomerServiceProvider):
    @abstractmethod
    def create_account(self, customer, acc_no, acc_type, balance):
        pass
```

```
    @abstractmethod
    def list_accounts(self):
        pass

    @abstractmethod
    def calculate_interest(self, account):
        pass

class CustomerServiceProviderImpl(ICustomerServiceProvider):
    def __init__(self):
        self.accounts = []



class BankServiceProviderImpl(CustomerServiceProviderImpl, IBankServiceProvider):
    def __init__(self, branch_name, branch_address):
        super().__init__()
        self.branch_name = branch_name
        self.branch_address = branch_address
```

## TASK 12

```
class InsufficientFundException(Exception):
    pass

class InvalidAccountException(Exception):
    pass

class OverDraftLimitExceededException(Exception):
    pass

class NullPointerException(Exception):
    pass
```

## TASK 13

```
class CustomerServiceProviderImpl(ICustomerServiceProvider):
    def __init__(self):
        self.accounts_list: List[Account] = []
        self.accounts_set: Set[Account] = set()
        self.accounts_map: Dict[int, Account] = {}
```

## TASK 14

### METHODS AND ITS IMPLEMENTATION USING MYSQL CONNECTOR

```
import mysql.connector
from mysql.connector import Error
from datetime import datetime

# Database connection
def create_connection():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            port='3306',
            database="hmbank"
        )
        return connection
    except Error as e:
        print(f"Error connecting to the database: {e}")
        return None
```

1. **createAccount()**

```
counter = 10


def generate_cus_number():
    global counter
    counter += 1
    return counter




def record_cus():
    connection = create_connection()
    if connection:
        try:
            customer_number=generate_cus_number()
            first_name = input("Enter first name: ")
            last_name = input("Enter last name: ")
            email = input("Enter email: ")
            phno = input("Enter phone number: ")
            c_date = datetime.now().strftime("%Y-%m-%d")


            cursor = connection.cursor()
            cursor.execute("INSERT INTO customers (customerid, firstname, lastname, date,email,phone)
VALUES (%s, %s, %s, %s,%s,%s)",
                        (customer_number,first_name,last_name,c_date,email,phno ))
            connection.commit()


            print("customer recorded successfully!")


        except (Error, ValueError) as e:
```

```
        print(f"Error recording : {e}")
    finally:
        connection.close()
```

```
sign\hm bank\bank.py'
Enter first name: aki
Enter last name: k
Enter email: aki@gmail
Enter phone number: 32784797
customer recorded successfully!
```

```
mysql> select * from customers;
+------------+-----------+-----------+------------+------------------+----------------+
| customerid | firstname | lastname  | date       | email            | phone          |
+------------+-----------+-----------+------------+------------------+----------------+
|          1 | Gretchen  | Bird      | 1995-04-22 | bird@icloud.com  | (552) 253-2923 |
|          3 | Macy      | Travis    | 1990-01-29 | macy@outlook.com | (157) 375-9678 |
|          5 | Jael      | Mcfarland | 1987-06-08 | jael@aol.couk    | (403) 558-6094 |
|          7 | Jamal     | Walls     | 1989-11-16 | jamal@yahoo.ca   | 1-344-653-4977 |
|          8 | Slade     | Boone     | 1999-03-28 | slade@aol.couk   | 1-377-644-1576 |
|         11 | aki       | k         | 2023-12-25 | aki@gmail        | 32784797       |
+------------+-----------+-----------+------------+------------------+----------------+
6 rows in set (0.01 sec)
```

## 2. listAccounts()

```
def display_acc_listings():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM accounts")
            cus = cursor.fetchall()

            print("accounts:")
            for alist in cus:
                print(alist)
```

```
    except Error as e:
        print(f"Error retrieving  listings: {e}")
    finally:
        connection.close()
```

```
sign\hm bank\bank.py'
accounts:
(21, 5, 'savings', -40200.0)
(42, 8, 'savings', 7000.0)
(63, 3, 'current', 20000.0)
(72, 7, 'zero balance', 30000.0)
(97, 1, 'savings', 72000.0)
```

3. **calculateInterest()**

4. getAccountBalance()

```
def get_balance():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
            Account_id=int(input("enter account number:"))
            select_query=("SELECT balance FROM accounts where accountid= %s")
            cursor.execute(select_query,(Account_id,))
            cus = cursor.fetchone()

            print("balance:")

            print(cus)

        except Error as e:
```

```
        print(f"Error retrieving listings: {e}")
    finally:
        connection.close()
```

```
sign\hm bank\bank.py'
enter account number:21
balance:
(-40200.0,)
```

5. deposit()

```
t_counter = 1000


def generate_t_number():
    global t_counter
    t_counter += 1
    return t_counter




def deposit_in_table():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
            accountid=int(input("enter account id:"))
            deposit=float(input("enter deposit ammount:"))
            ttype="Deposit"
            transactionid=generate_t_number()
            t_date = datetime.now().strftime("%Y-%m-%d")
            select_query=("SELECT balance FROM accounts where accountid= %s")
            cursor.execute(select_query,(accountid,))
            cus = cursor.fetchone()
            total=sum(cus,deposit)
            update_query = ("UPDATE accounts SET balance = %s WHERE accountid = %s")
```

```
        cursor.execute(update_query, (total, accountid))


        cursor.execute("INSERT INTO transactions (transactionid,accountid,
transactiontype,amount,transactiondate) VALUES (%s,%s, %s, %s, %s)",

                (transactionid,accountid,ttype,deposit,t_date))


        connection.commit()

        print(f"Updated value in the database.")


    except Error as e:

        print(f"Error updating value in the table: {e}")


    finally:

        connection.close()
```

```
sign\hm bank\bank.py'
enter account id:21
enter deposit ammount:300
Updated value in the database.
```

```
mysql> select * from transactions;
+---------------+-----------+-----------------+--------+-----------------+
| transactionid | accountid | transactiontype | amount | transactiondate |
+---------------+-----------+-----------------+--------+-----------------+
|          1001 |        21 | Deposit         |    100 | 2023-12-25      |
|          1003 |        21 | Deposit         |    300 | 2023-12-25      |
|          1011 |        21 | Withdraw        |    100 | 2023-12-25      |
|          4412 |        42 | deposit         |   1000 | 2023-05-22      |
|          5242 |        63 | deposit         |    500 | 2023-05-22      |
|          6931 |        21 | deposit         |   3000 | 2023-05-22      |
|          7731 |        97 | withdrawal      |   2500 | 2023-05-22      |
|          8471 |        72 | transfer        |    750 | 2023-05-22      |
+---------------+-----------+-----------------+--------+-----------------+
8 rows in set (0.00 sec)

mysql> select * from accounts;
+-----------+------------+--------------+---------+
| accountid | customerid | accounttype  | balance |
+-----------+------------+--------------+---------+
|        21 |          5 | savings      |  -39900 |
|        42 |          8 | savings      |    7000 |
|        63 |          3 | current      |   20000 |
|        72 |          7 | zero balance |   30000 |
|        97 |          1 | savings      |   72000 |
+-----------+------------+--------------+---------+
5 rows in set (0.00 sec)
```

### 6. withdraw()

```
t_w = 1020


def generate_tw_number():
    global t_w
    t_w += 1
    return t_w


from functools import reduce
import operator


def with_in_table():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
```

```
        accountid=int(input("enter account id:"))
        withd=float(input("enter withdraw ammount:"))
        ttype="Withdraw"
        transactionid=generate_tw_number()
        t_date = datetime.now().strftime("%Y-%m-%d")
        select_query=("SELECT balance FROM accounts where accountid= %s")
        cursor.execute(select_query,(accountid,))
        cus = cursor.fetchone()
        total=reduce(operator.__sub__, cus,withd)
        update_query = ("UPDATE accounts SET balance = %s WHERE accountid = %s")
        cursor.execute(update_query, (total, accountid))

        cursor.execute("INSERT INTO transactions (transactionid,accountid,
transactiontype,amount,transactiondate) VALUES (%s,%s, %s, %s, %s)",
                    (transactionid,accountid,ttype,withd,t_date))

        connection.commit()
        print(f"Updated value in the database.")

    except Error as e:
        print(f"Error updating value in the table: {e}")

    finally:
        connection.close()
```

```
sign\hm bank\bank.py'
enter account id:42
enter withdraw ammount:500
Updated value in the database.
```

```
mysql> select * from transactions;
+---------------+-----------+-----------------+--------+-----------------+
| transactionid | accountid | transactiontype | amount | transactiondate |
+---------------+-----------+-----------------+--------+-----------------+
|          1001 |        21 | Deposit         |    100 | 2023-12-25      |
|          1003 |        21 | Deposit         |    300 | 2023-12-25      |
|          1011 |        21 | Withdraw        |    100 | 2023-12-25      |
|          1021 |        42 | Withdraw        |    500 | 2023-12-25      |
|          4412 |        42 | deposit         |   1000 | 2023-05-22      |
|          5242 |        63 | deposit         |    500 | 2023-05-22      |
|          6931 |        21 | deposit         |   3000 | 2023-05-22      |
|          7731 |        97 | withdrawal      |   2500 | 2023-05-22      |
|          8471 |        72 | transfer        |    750 | 2023-05-22      |
+---------------+-----------+-----------------+--------+-----------------+
9 rows in set (0.01 sec)

mysql> select * from accounts;
+-----------+------------+--------------+---------+
| accountid | customerid | accounttype  | balance |
+-----------+------------+--------------+---------+
|        21 |          5 | savings      |  -39900 |
|        42 |          8 | savings      |   -6500 |
|        63 |          3 | current      |   20000 |
|        72 |          7 | zero balance |   30000 |
|        97 |          1 | savings      |   72000 |
+-----------+------------+--------------+---------+
5 rows in set (0.00 sec)
```

### 7. transfer()

```
t_w = 2110


def generate_tr_number():
    global t_w
    t_w += 1
    return t_w


from functools import reduce
import operator


def tr_in_table():
    connection = create_connection()
    if connection:
```

```
try:
    cursor = connection.cursor()
    accountid=int(input("enter account id:"))
    withd=float(input("enter transfer ammount:"))
    ttype="Transfer"
    transactionid=generate_tr_number()
    t_date = datetime.now().strftime("%Y-%m-%d")
    select_query=("SELECT balance FROM accounts where accountid= %s")
    cursor.execute(select_query,(accountid,))
    cus = cursor.fetchone()
    total=reduce(operator.__sub__, withd,cus)
    update_query = ("UPDATE accounts SET balance = %s WHERE accountid = %s")
    cursor.execute(update_query, (total, accountid))

    cursor.execute("INSERT INTO transactions (transactionid,accountid,
transactiontype,amount,transactiondate) VALUES (%s,%s, %s, %s, %s)",
                   (transactionid,accountid,ttype,withd,t_date))

    connection.commit()
    print(f"Updated value in the database.")

except Error as e:
    print(f"Error updating value in the table: {e}")

finally:
    connection.close()
```

```
sign\hm bank\bank.py'
enter account id:21
enter transfer ammount:200
Updated value in the database.
```

```
mysql> select * from transactions;
+---------------+-----------+-----------------+--------+-----------------+
| transactionid | accountid | transactiontype | amount | transactiondate |
+---------------+-----------+-----------------+--------+-----------------+
|          1001 |        21 | Deposit         |    100 | 2023-12-25      |
|          1003 |        21 | Deposit         |    300 | 2023-12-25      |
|          1011 |        21 | Withdraw        |    100 | 2023-12-25      |
|          1021 |        42 | Withdraw        |    500 | 2023-12-25      |
|          2111 |        21 | Transfer        |    200 | 2023-12-25      |
|          4412 |        42 | deposit         |   1000 | 2023-05-22      |
|          5242 |        63 | deposit         |    500 | 2023-05-22      |
|          6931 |        21 | deposit         |   3000 | 2023-05-22      |
|          7731 |        97 | withdrawal      |   2500 | 2023-05-22      |
|          8471 |        72 | transfer        |    750 | 2023-05-22      |
+---------------+-----------+-----------------+--------+-----------------+
10 rows in set (0.00 sec)

mysql> select * from accounts;
+-----------+------------+--------------+---------+
| accountid | customerid | accounttype  | balance |
+-----------+------------+--------------+---------+
|        21 |          5 | savings      |   40100 |
|        42 |          8 | savings      |   -6500 |
|        63 |          3 | current      |   20000 |
|        72 |          7 | zero balance |   30000 |
|        97 |          1 | savings      |   72000 |
+-----------+------------+--------------+---------+
5 rows in set (0.00 sec)
```

### 8. getAccountDetails()

```
def display_cus_listings():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
            cursor.execute("SELECT * FROM customers")
            cus = cursor.fetchall()

            print("customers:")
            for clist in cus:
                print(clist)

        except Error as e:
            print(f"Error retrieving  listings: {e}")
```

```
    finally:
        connection.close()
```

```
sign\hm bank\bank.py'
customers:
(1, 'Gretchen', 'Bird', datetime.date(1995, 4, 22), 'bird@icloud.com', '(552) 253-2923')
(3, 'Macy', 'Travis', datetime.date(1990, 1, 29), 'macy@outlook.com', '(157) 375-9678')
(5, 'Jael', 'Mcfarland', datetime.date(1987, 6, 8), 'jael@aol.couk', '(403) 558-6094')
(7, 'Jamal', 'Walls', datetime.date(1989, 11, 16), 'jamal@yahoo.ca', '1-344-653-4977')
(8, 'Slade', 'Boone', datetime.date(1999, 3, 28), 'slade@aol.couk', '1-377-644-1576')
(11, 'aki', 'k', datetime.date(2023, 12, 25), 'aki@gmail', '32784797')
```

### 9. getTransations()

```
def get_transaction():
    connection = create_connection()
    if connection:
        try:
            cursor = connection.cursor()
            Account_id=int(input("enter account number:"))
            select_query=("SELECT * FROM transactions where accountid= %s")
            cursor.execute(select_query,(Account_id,))
            cus = cursor.fetchall()

            print("transactions:")

            print(cus)

        except Error as e:
            print(f"Error retrieving listings: {e}")
        finally:
            connection.close()
```

```
sign\hm bank\bank.py'
enter account number:21
transactions:
[(1001, 21, 'Deposit', 100.0, datetime.date(2023, 12, 25)), (1003, 21, 'Deposit', 300.0, datetime.date(2023, 12, 25)), (1011, 21, 'Withdraw', 100.0,
 datetime.date(2023, 12, 25)), (2111, 21, 'Transfer', 200.0, datetime.date(2023, 12, 25)), (6931, 21, 'deposit', 3000.0, datetime.date(2023, 5, 22))
]
```