1. Introduction

This report explains a simple **Banking System application** developed using **Python**.
The program allows a user to:

- Create basic account details
- Check account balance
- Deposit money
- Withdraw money

The purpose of this project is to understand **functions, conditional logic, dictionaries, and user input handling** in Python.

---

2. Problem Statement

To design a basic console-based banking system that securely allows a user to:

- Access account features using a PIN
- Perform basic banking transactions
- Display updated balance information

---

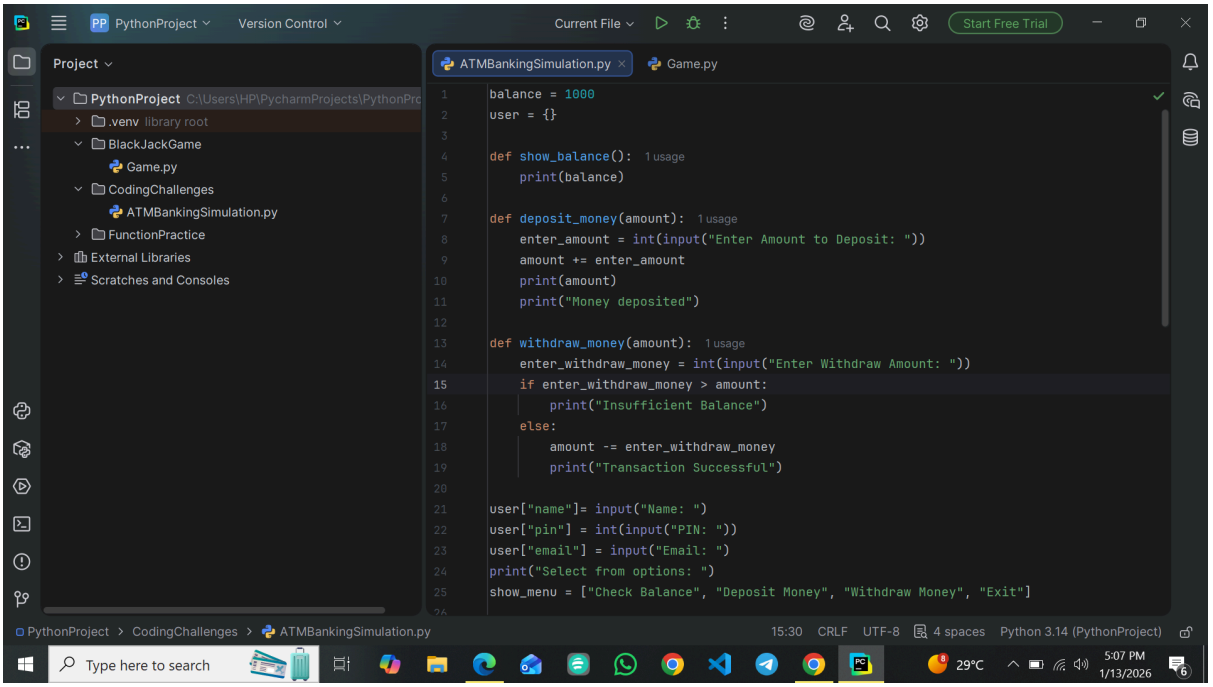
3. Technologies Used

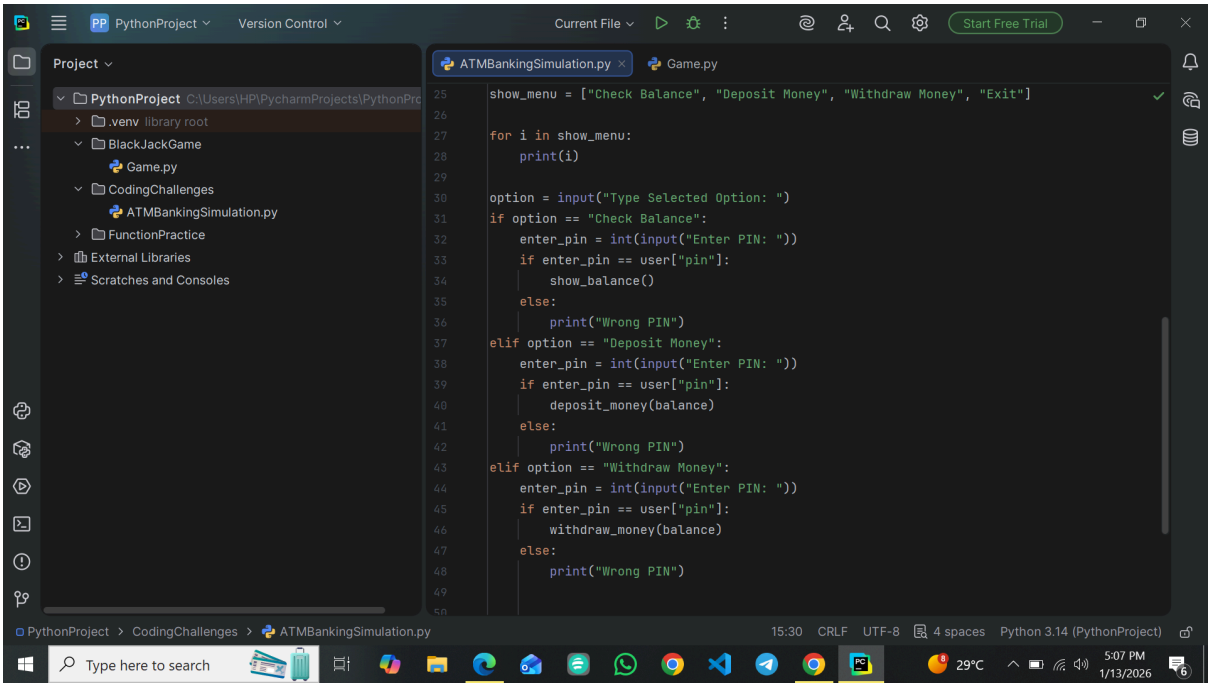
- **Programming Language:** Python
- **Concepts Used:**
    - Variables
    - Functions
    - Dictionaries
    - Conditional Statements

- Loops
- User Input Handling

---

4. Source Code





Banking System Source Code

---

5. Code Explanation (Function-wise & Logic-wise)

5.1 Global Variables

```
balance = 1000
user = {}
```

- `balance` stores the initial account balance.
- `user` dictionary stores user details like name, PIN, and email.

5.2 show_balance() Function

```
def show_balance():
    print(balance)
```

Purpose:
Displays the current balance of the user.

Feature:

- Simple function with no parameters.
- Reads the global `balance` variable.

5.3 deposit_money(amount) Function

```
def deposit_money(amount):
    enter_amount = int(input("Enter Amount to Deposit: "))
    amount += enter_amount
    print(amount)
    print("Money deposited")
```

Purpose:
Allows the user to deposit money.

Logic Explanation:

- Takes deposit amount as input.
- Adds deposit amount to the provided balance.
- Displays updated balance.

Limitation:

- Balance change does not reflect globally due to pass-by-value behavior.

5.4 withdraw_money(amount) Function

```
def withdraw_money(amount):
    enter_withdraw_money = int(input("Enter Withdraw Amount: "))
    if enter_withdraw_money < amount:
        print("Insufficient Balance")
    else:
        amount -= enter_withdraw_money
        print("Transaction Successful")
```

Purpose:
Handles withdrawal transactions.

Logic Explanation:

- Checks if withdrawal amount exceeds available balance.
- Deducts amount if sufficient balance exists.

5.5 User Registration Logic

```
user["name"]= input("Name: ")
user["pin"] = int(input("PIN: "))
user["email"] = input("Email: ")
```

Purpose:
Collects user details and stores them securely in a dictionary.

5.6 Menu Display Logic

```
show_menu = ["Check Balance", "Deposit Money", "Withdraw Money",
"Exit"]

for i in show_menu:
    print(i)
```

Purpose:
Displays available banking operations.

5.7 PIN Authentication & Option Handling

```
if option == "Check Balance":
    ...
elif option == "Deposit Money":
    ...
elif option == "Withdraw Money":
    ...
```

Purpose:

- Verifies user PIN before executing any operation.
- Enhances basic security.

---

6. Key Features of the Program

- PIN-based authentication
- Modular function-based design
- Simple user-friendly menu
- Demonstrates real-world banking logic
- Easy to understand for beginners

---

7. Limitations of the Current Code

- Balance does not update globally after deposit/withdraw
- No loop to allow multiple transactions
- No data persistence (data lost after program exits)
- No input validation for negative values
- Security is minimal (PIN visible during input)

---

8. Future Scope / Further Enhancements

The following improvements can be implemented in future versions:

1. **Global Balance Update**
    - Use `global balance` or return updated balance from functions.

2. **Transaction Loop**
    - Allow users to perform multiple transactions until exit.
3. **File Handling / Database**
    - Store user data permanently using files or databases.
4. **Enhanced Security**
    - Mask PIN input.
    - Add PIN retry limit.
5. **Multiple User Support**
    - Allow multiple accounts using lists or databases.
6. **GUI-Based Interface**
    - Convert console app to GUI using Tkinter or web-based UI.

---

9. Conclusion

This Python banking system project demonstrates fundamental programming concepts such as functions, conditionals, dictionaries, and user interaction. While basic, it provides a strong foundation for building more advanced financial applications and understanding real-world logic implementation.