과제-2 **(함수와 모듈) 은행 시스템 구현**

① 수행 과업

이 과제는 함수와 모듈의 개념을 이해하고, 이를 활용하여 실용적인 프로그램을 설계하는 데 있습니다. 은행 시스템에서 흔히 사용되는 기본 기능(입금, 출금, 잔액 조회)을 함수로 구현하고, 모듈화를 통해 코드의 재사용성과 유지보수성을 높이는 방법을 학습합니다. 이를 통해 학생들은 코드 구조화와 협업 역량을 향상시킬 수 있습니다.

② 요구사항

1. **기능 정의**
   - **입금 기능**: 사용자가 입력한 금액을 계좌 잔액에 추가
   - **출금 기능**: 사용자가 입력한 금액을 계좌 잔액에서 차감. 단, 잔액 부족시 오류 메시지를 출력
   - **잔액 조회**: 현재 계좌 잔액을 출력
2. **모듈화**
   - 은행 시스템의 각 기능(입금, 출금, 잔액 조회)을 별도의 함수로 작성
   - 모든 함수는 **main 함수**에서 호출되도록 구조화
3. **에러 처리**
   - 입력된 금액이 숫자가 아닌 경우 오류 메시지를 출력
   - 출금 시 잔액이 부족한 경우 오류 메시지를 출력
4. **확장 가능성**
   - 프로그램이 종료되지 않고 반복적으로 동작
   - 사용자가 "exit"을 입력하면 프로그램이 종료

③ 스스로 학습

1. 함수의 기본 개념

- 키워드: def, 매개변수, 반환값(return)
- 학습 방향:
  - 함수를 정의하고 호출하는 기본 방법 학습
  - 매개변수를 활용하여 데이터 전달 및 결과 반환 연습
- 예시 질문: "왜 함수를 사용하면 코드가 더 효율적이고 가독성이 좋아질까?“

2. 모듈화

- 키워드: import, 사용자 정의 모듈
- 학습 방향:
  - Python의 모듈 사용법과, 코드의 기능을 분리하여 모듈로 저장하는 방법을 학습.
  - 내장 모듈과 사용자 정의 모듈의 차이점 이해.
- 예시 질문: "모듈화를 통해 협업과 유지보수가 어떻게 더 쉬워질까?“

3. 데이터 검증과 에러 처리

- 키워드: try, except, raise
- 학습 방향: 조건에 따라 다른 결과를 실행하는 방법을 이해합니다.
  - 입력된 데이터가 올바른 형식인지 검증.
  - 에러 발생 시 사용자에게 적절한 메시지를 출력하는 방법 학습.
- 예시 질문: "왜 데이터 검증과 에러 처리가 프로그램 안정성에 중요한가?“

4. 프로그램 흐름 제어

- 키워드: while, break
- 학습 방향:
  - 프로그램이 종료되지 않고 반복적으로 동작하도록 제어하는 방법 학습.
  - 종료 조건과 반복 조건 설정 이해.

④ 코드 구현 예제를 이해하여 설명하시오.

```
# 은행 시스템 메인 프로그램
def deposit(balance, amount):
    """입금 함수: 잔액에 금액을 추가"""
    if amount <= 0:
        print("입금 금액은 0보다 커야 합니다.")
        return balance
    balance += amount
    print(f"{amount}원이 입금되었습니다. 현재 잔액: {balance}원")
    return balance


def withdraw(balance, amount):
    """출금 함수: 잔액에서 금액을 차감"""
    if amount <= 0:
        print("출금 금액은 0보다 커야 합니다.")
        return balance
    if amount > balance:
        print("잔액이 부족합니다. 출금할 수 없습니다.")
        return balance
    balance -= amount
    print(f"{amount}원이 출금되었습니다. 현재 잔액: {balance}원")
    return balance


def check_balance(balance):
    """잔액 조회 함수"""
    print(f"현재 잔액: {balance}원")


def main():
    """메인 함수: 은행 시스템 실행"""
    print("=== 은행 시스템 ===")
    print("사용 가능 명령어: deposit, withdraw, check, exit")

    balance = 0  # 초기 잔액 설정

    while True:
        user_input = input("명령어를 입력하세요: ").strip().lower()

        if user_input == "exit":
            print("프로그램을 종료합니다.")
            break
        elif user_input == "deposit":
            try:
                amount = int(input("입금 금액을 입력하세요: "))
                balance = deposit(balance, amount)
            except ValueError:
                print("유효한 숫자를 입력해주세요.")
        elif user_input == "withdraw":
```

```
            try:
                amount = int(input("출금 금액을 입력하세요: "))
                balance = withdraw(balance, amount)
            except ValueError:
                print("유효한 숫자를 입력해주세요.")
        elif user_input == "check":
            check_balance(balance)
        else:
            print("유효하지 않은 명령어입니다. 다시 입력해주세요.")


# 프로그램 실행
if __name__ == "__main__":
    main()
```

【Peer 평가 Check List】(함수와 모듈) 은행 시스템 구현

설명자 Explainer		평가자 Evaluator		날짜 date	
☐ 전혀 설명하지 못함	☐ 기본적인 설명은 했으나 부족함		☐ 완벽하게 이해하고 명확히 설명함		

【프로그램 구조에 대한 이해】

1. 전체적인 프로그램 흐름

- ☐ 프로그램의 전체 구조와 실행 흐름(main 함수와 반복문)을 명확히 이해했는가?
- ☐ 프로그램 종료 조건("exit")이 코드에서 어떻게 처리되는지 설명했는가?

2. 함수의 역할

- ☐ deposit 함수가 입금 금액을 처리하는 과정을 정확히 설명했는가?
- ☐ withdraw 함수가 출금 금액을 처리하고, 잔액 부족 시 예외를 처리하는 방식을 명확히 설명했는가?
- ☐ check_balance 함수가 현재 잔액을 출력하는 동작을 정확히 설명했는가?

3. 입력 처리 및 예외 처리

- ☐ 사용자가 잘못된 입력(예: 숫자가 아닌 값)을 했을 때, 예외 처리(try-except)가 프로그램 흐름에 어떻게 적용되는지 설명했는가?
- ☐ 잔액 부족 또는 잘못된 금액 입력 시 오류 메시지가 출력되는 원리를 설명했는가?

【코드 세부 사항에 대한 이해】

4. 함수 매개변수와 반환값

- ☐ 각 함수(deposit, withdraw, check_balance)의 매개변수와 반환값이 코드에서 어떤 역할을 하는지 정확히 설명했는가?

5. 반복문과 조건문

- ☐ while 반복문이 프로그램의 지속적인 실행을 어떻게 보장하는지 설명했는가?
- ☐ 조건문(if, elif, else)이 사용자 입력에 따라 프로그램 동작을 어떻게 결정하는지 명확히 설명했는가?

【설명 능력】

6. 명확성과 논리성

- ☐ 동료에게 프로그램의 동작 원리를 논리적으로 설명했는가?
- ☐ 어려운 개념(예: 예외 처리, 반환값 등)을 쉽게 이해할 수 있도록 설명했는가?
- ☐ 동료가 잘 이해하지 못한 부분에 대해 추가적인 예시나 설명을 제공했는가?

⑤ [도전과제] 은행 시스템 기능 확장 및 개선

- 요구사항:

1. 입출금 기록 기능 추가

- 사용자가 수행한 모든 입금/출금 기록을 리스트에 저장하고, "history" 명령어를 입력하면 지금까지의 기록을 출력하도록 기능을 추가합니다.
- 기록 형식 예

```
입금: 5000원
출금: 2000원
```

2. 일일 입출금 한도 설정

- 사용자가 하루에 입출금할 수 있는 최대 한도를 설정
- 입금 한도: 하루 최대 1,000,000원
- 출금 한도: 하루 최대 500,000원
- 사용자가 한도를 초과할 경우, 적절한 오류 메시지를 출력

3. 비밀번호 인증 기능

- 프로그램 실행 시 사용자에게 4자리 비밀번호를 입력받아, 올바른 비밀번호가 입력될 때만 은행 시스템에 접근할 수 있도록 설정
- 비밀번호가 잘못 입력되면 프로그램이 종료

【Hint】

① 입출금 기록 저장 자료 구조:

```
history = []
```

② 입출금 기록 추가 방법

```
history.append(f"입금: {amount}원")  # 입금 기록
history.append(f"출금: {amount}원")  # 출금 기록
```

③ 일일 한도 관리 변수 초기화

```
daily_deposit = 0
daily_withdraw = 0
```

④ 비밀번호 인증 로직

```
password = "1234"
user_input = input("비밀번호를 입력하세요: ")
if user_input != password:
        print("비밀번호가 틀렸습니다. 프로그램을 종료합니다.")
exit()
```