AI Assisted Coding Lab Ass-6.1

Name: D.Nithin

Batch-13

2303A51845

Task Description #1 (AI-Based Code Completion for Loops) Task:

Use an AI code completion tool to generate a loop-based

program.

Prompt:

“Generate Python code to print all even numbers between 1 and N using

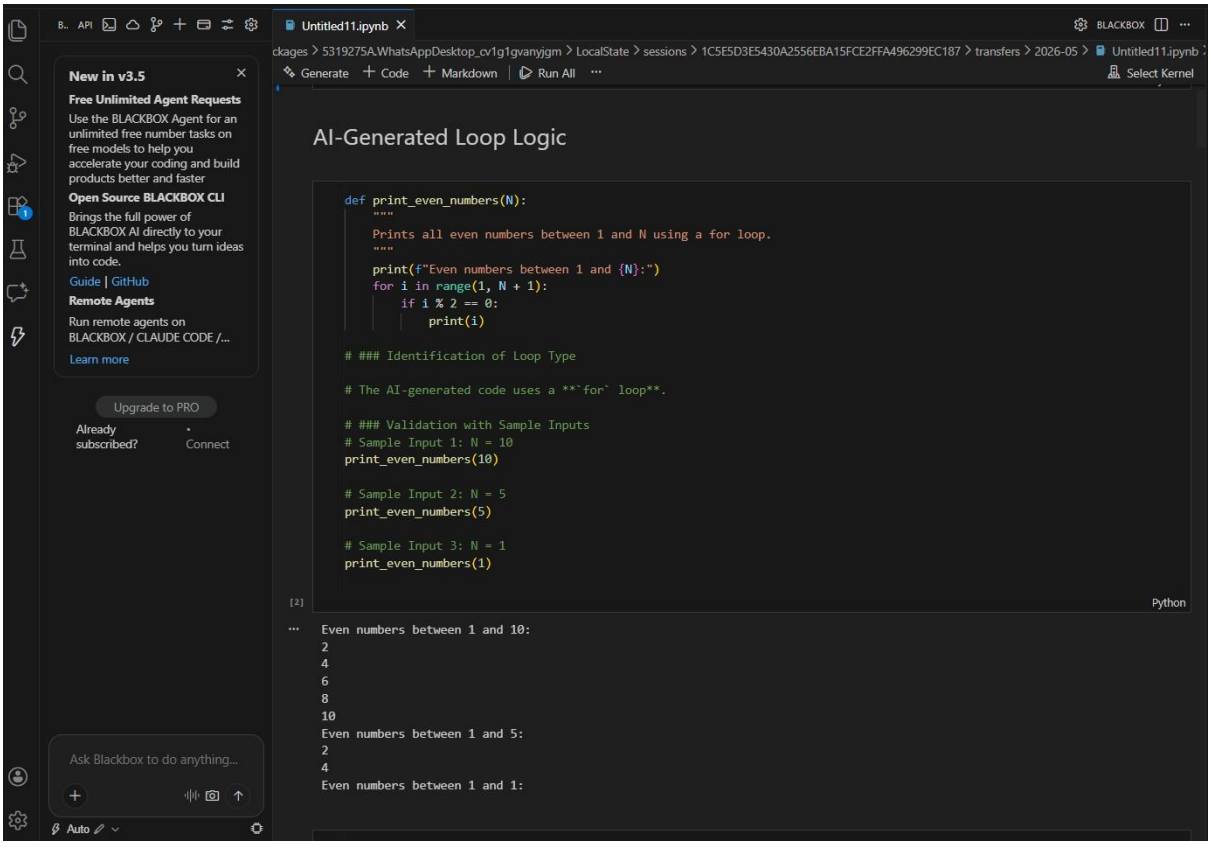
a loop.”

Expected Output:

• AI-generated loop logic.

• Identification of loop type used (for or while).

• Validation with sample inputs.



Task Description #2 (AI-Based Code Completion for Loop with

Conditionals)

Task: Use an AI code completion tool to combine loops and conditionals.

Prompt:

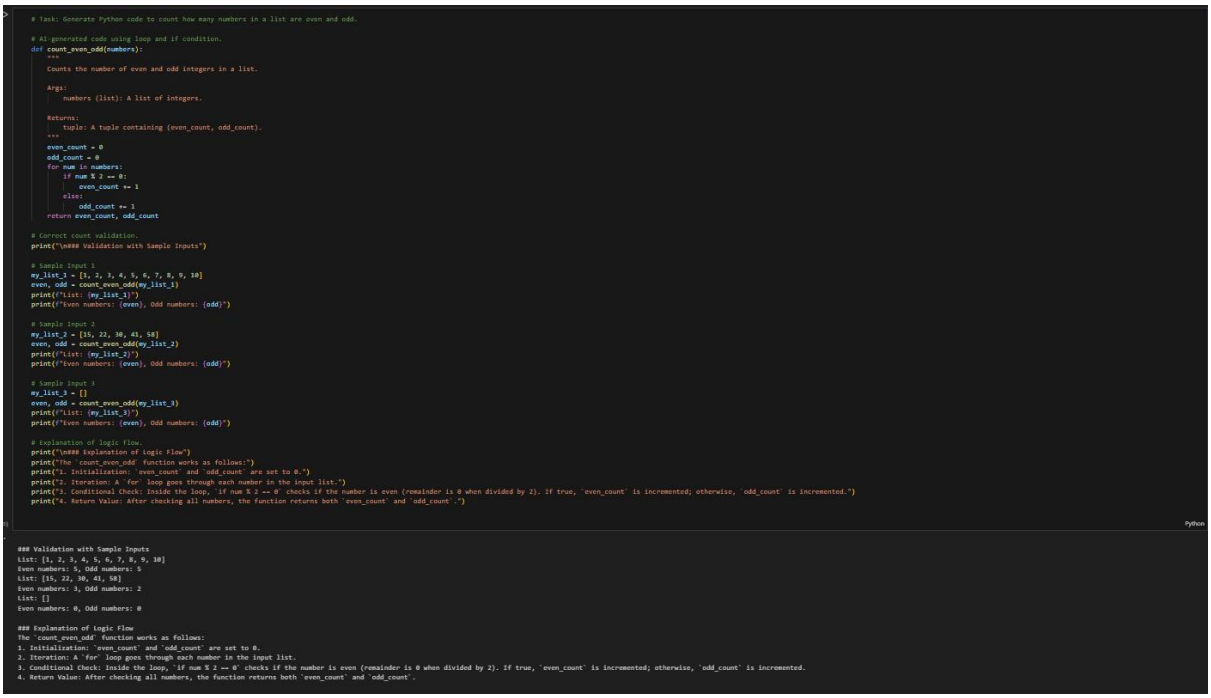
“Generate Python code to count how many numbers in a list are even and odd.”

Expected Output:

• AI-generated code using loop and if condition.

• Correct count validation.

• Explanation of logic flow.



```
# Task: Generate Python code to count how many numbers in a list are even and odd.

# AI-generated code using loop and if condition.
def count_even_odd(numbers):
    """
    Counts the number of even and odd integers in a list.

    Args:
        numbers (list): A list of integers.

    Returns:
        tuple: A tuple containing (even_count, odd_count).
    """
    even_count = 0
    odd_count = 0
    for num in numbers:
        if num % 2 == 0:
            even_count += 1
        else:
            odd_count += 1
    return even_count, odd_count

# Correct count validation.
print("\n### Validation with Sample Inputs")

# Sample Input 1
my_list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even, odd = count_even_odd(my_list_1)
print(f"List: {my_list_1}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Sample Input 2
my_list_2 = [15, 22, 38, 41, 58]
even, odd = count_even_odd(my_list_2)
print(f"List: {my_list_2}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Sample Input 3
my_list_3 = []
even, odd = count_even_odd(my_list_3)
print(f"List: {my_list_3}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Explanation of logic flow.
print("\n### Explanation of Logic Flow")
print("The `count_even_odd` function works as follows:")
print("1. Initialization: `even_count` and `odd_count` are set to 0.")
print("2. Iteration: A `for` loop goes through each number in the input list.")
print("3. Conditional Check: Inside the loop, `if num % 2 == 0` checks if the number is even (remainder is 0 when divided by 2). If true, `even_count` is incremented; otherwise, `odd_count` is incremented.")
print("4. Return Value: After checking all numbers, the function returns both `even_count` and `odd_count`.")
```

Python

```
### Validation with Sample Inputs
List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even numbers: 5, Odd numbers: 5
List: [15, 22, 38, 41, 58]
Even numbers: 3, Odd numbers: 2
List: []
Even numbers: 0, Odd numbers: 0

### Explanation of Logic Flow
The `count_even_odd` function works as follows:
1. Initialization: `even_count` and `odd_count` are set to 0.
2. Iteration: A `for` loop goes through each number in the input list.
3. Conditional Check: Inside the loop, `if num % 2 == 0` checks if the number is even (remainder is 0 when divided by 2). If true, `even_count` is incremented; otherwise, `odd_count` is incremented.
4. Return Value: After checking all numbers, the function returns both `even_count` and `odd_count`.
```

Task Description #3 (AI-Based Code Completion for Class Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input.

Prompt:

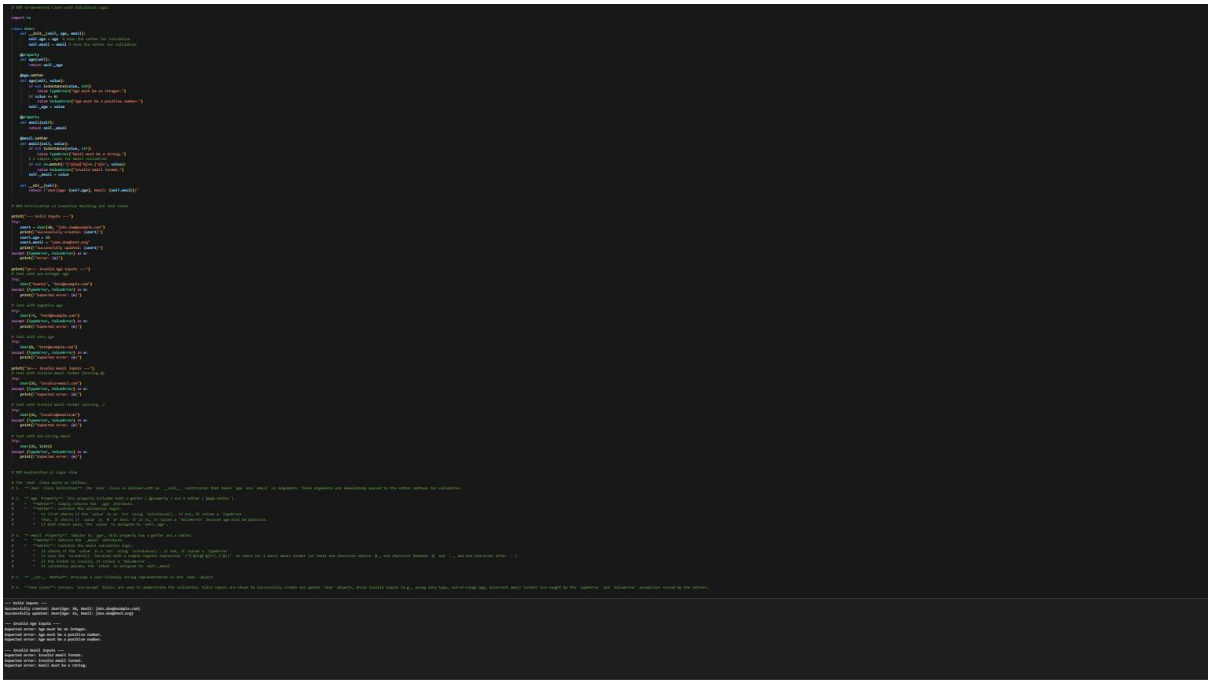
“Generate a Python class User that validates age and email using conditional statements.”

Expected Output:

• AI-generated class with validation logic.

• Verification of condition handling.

• Test cases for valid and invalid inputs.



Task Description #4 (AI-Based Code Completion for Classes) Task:

Use an AI code completion tool to generate a Python class for

managing student details.

Prompt:

“Generate a Python class Student with attributes (name, roll number,

marks) and methods to calculate total and average marks.”

Expected Output:

• AI-generated class code.

• Verification of correctness and completeness of class structure.

• Minor manual improvements (if needed) with justification.

Task Description 5 (AI-Assisted Code Completion Review) Task:

Use an AI tool to generate a complete Python program using

classes, loops, and conditionals together.

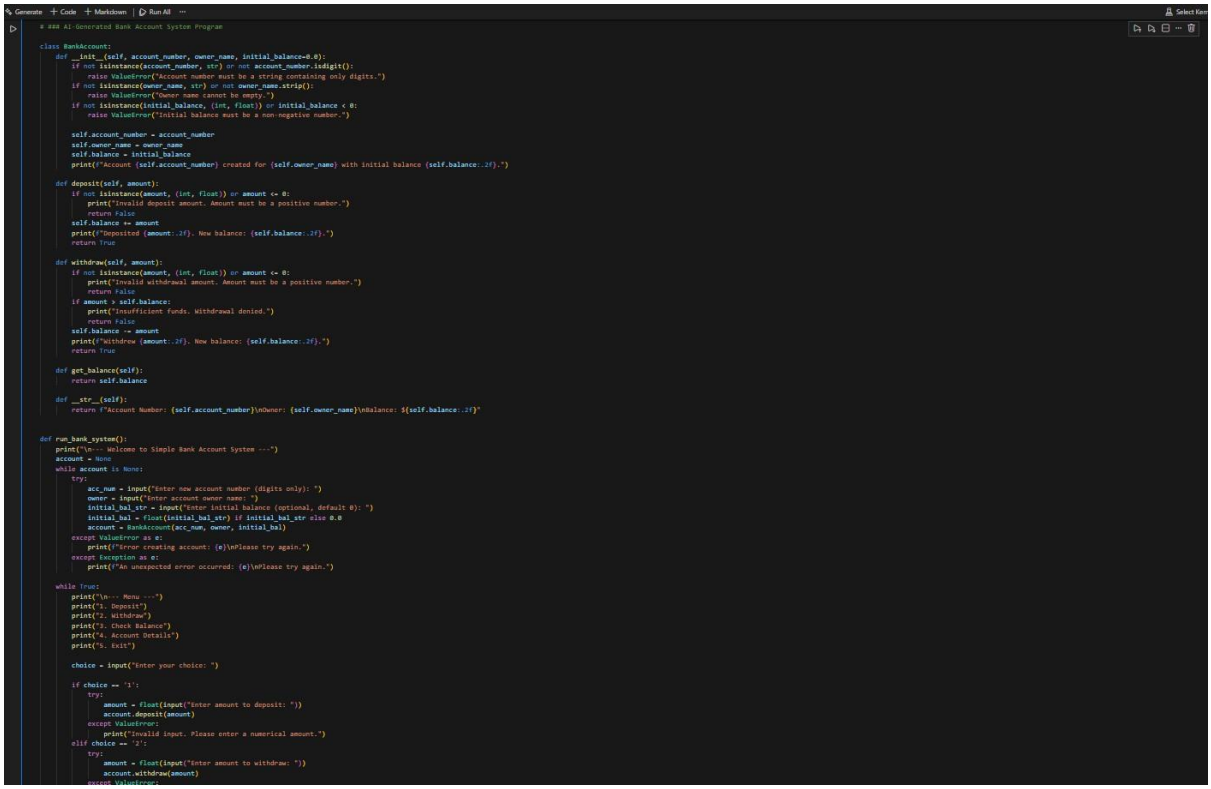
Prompt:

“Generate a Python program for a simple bank account system using

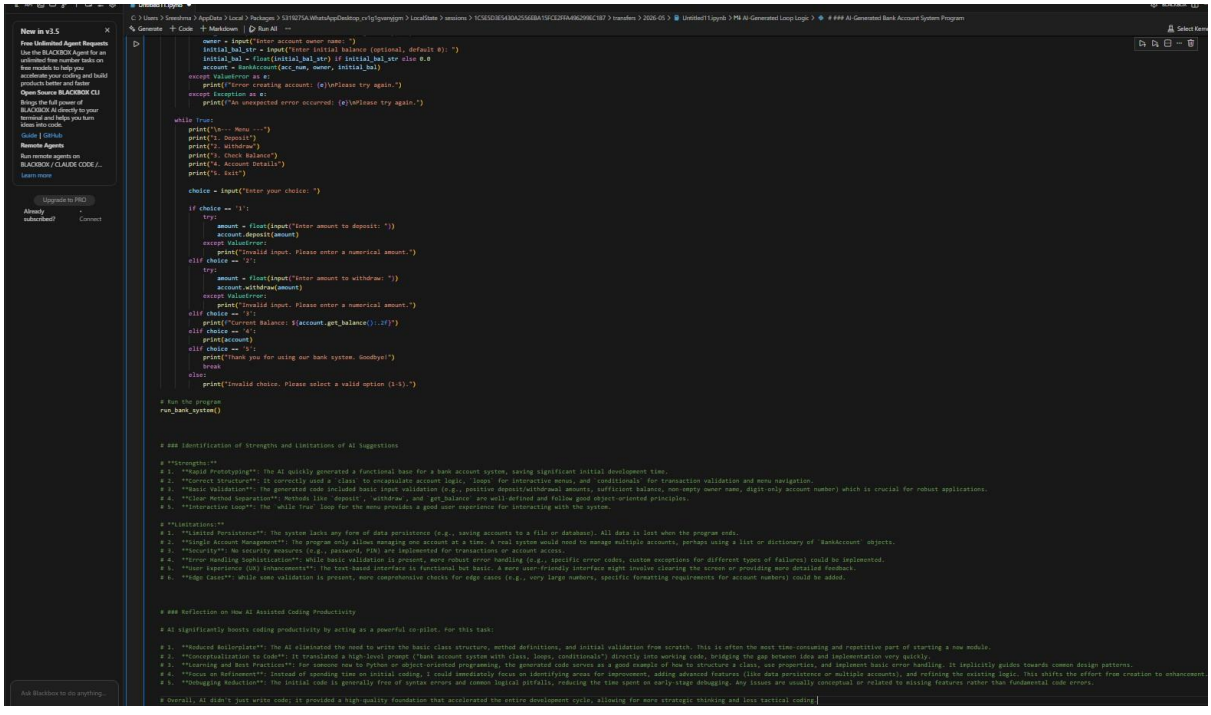
class, loops, and conditional statements.”

Expected Output:

- Complete AI-generated program.
- Identification of strengths and limitations of AI suggestions.
- Reflection on how AI assisted coding productivity.



```
# ### AI-Generated Bank Account System Program

class BankAccount:
    def __init__(self, account_number, owner_name, initial_balance=0.0):
        if not isinstance(account_number, str) or not account_number.isdigit():
            raise ValueError("Account number must be a string containing only digits.")
        if not isinstance(owner_name, str) or not owner_name.strip():
            raise ValueError("Owner name cannot be empty.")
        if not isinstance(initial_balance, (int, float)) or initial_balance < 0:
            raise ValueError("Initial balance must be a non-negative number.")

        self.account_number = account_number
        self.owner_name = owner_name
        self.balance = initial_balance
        print(f"Account {self.account_number} created for {self.owner_name} with initial balance {self.balance:.2f}.")

    def deposit(self, amount):
        if not isinstance(amount, (int, float)) or amount <= 0:
            print("Invalid deposit amount. Amount must be a positive number.")
            return False
        self.balance += amount
        print(f"Deposited {amount:.2f}. New balance: {self.balance:.2f}.")
        return True

    def withdraw(self, amount):
        if not isinstance(amount, (int, float)) or amount <= 0:
            print("Invalid withdrawal amount. Amount must be a positive number.")
            return False
        if amount > self.balance:
            print("Insufficient funds. Withdrawal denied.")
            return False
        self.balance -= amount
        print(f"Withdrew {amount:.2f}. New balance: {self.balance:.2f}.")
        return True

    def get_balance(self):
        return self.balance

    def __str__(self):
        return f"Account Number: {self.account_number}\nOwner: {self.owner_name}\nBalance: ${self.balance:.2f}"


def run_bank_system():
    print("\n--- Welcome to Simple Bank Account System ---")
    account = None
    while account is None:
        try:
            acc_num = input("Enter new account number (digits only): ")
            owner = input("Enter account owner name: ")
            initial_bal_str = input("Enter initial balance (optional, default 0): ")
            initial_bal = float(initial_bal_str) if initial_bal_str else 0.0
            account = BankAccount(acc_num, owner, initial_bal)
        except ValueError as e:
            print(f"Error creating account: {e}\nPlease try again.")
        except Exception as e:
            print(f"An unexpected error occurred: {e}\nPlease try again.")

    while True:
        print("\n--- Menu ---")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Account Details")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            try:
                amount = float(input("Enter amount to deposit: "))
                account.deposit(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
        elif choice == '2':
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
```




```
            owner = input("Enter account owner name: ")
            initial_bal_str = input("Enter initial balance (optional, default 0): ")
            initial_bal = float(initial_bal_str) if initial_bal_str else 0.0
            account = BankAccount(acc_num, owner, initial_bal)
        except ValueError as e:
            print(f"Error creating account: {e}\nPlease try again.")
        except Exception as e:
            print(f"An unexpected error occurred: {e}\nPlease try again.")

    while True:
        print("\n--- Menu ---")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Account Details")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            try:
                amount = float(input("Enter amount to deposit: "))
                account.deposit(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
        elif choice == '2':
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
        elif choice == '3':
            print(f"Current Balance: ${account.get_balance():.2f}")
        elif choice == '4':
            print(account)
        elif choice == '5':
            print("Thank you for using our bank system. Goodbye!")
            break
        else:
            print("Invalid choice. Please select a valid option (1-5).")

# Run the program
run_bank_system()


# ### Identification of Strengths and Limitations of AI Suggestions

# **Strengths:**
# 1. **Rapid Prototyping**: The AI quickly generated a functional base for a bank account system, saving significant initial development time.
# 2. **Correct Structure**: It correctly used a `class` to encapsulate account logic, `loops` for interactive menus, and `conditionals` for transaction validation and menu navigation.
# 3. **Basic Validation**: The generated code included basic input validation (e.g., positive deposit/withdrawal amounts, sufficient balance, non-empty owner name, digit-only account number) which is crucial for robust applications.
# 4. **Clear Method Separation**: Methods like `deposit`, `withdraw`, and `get_balance` are well-defined and follow good object-oriented principles.
# 5. **Interactive Loop**: The `while True` loop for the menu provides a good user experience for interacting with the system.

# **Limitations:**
# 1. **Limited Persistence**: The system lacks any form of data persistence (e.g., saving accounts to a file or database). All data is lost when the program ends.
# 2. **Single Account Management**: The program only allows managing one account at a time. A real system would need to manage multiple accounts, perhaps using a list or dictionary of `BankAccount` objects.
# 3. **Security**: No security measures (e.g., password, PIN) are implemented for transactions or account access.
# 4. **Error Handling Sophistication**: While basic validation is present, more robust error handling (e.g., specific error codes, custom exceptions for different types of failures) could be implemented.
# 5. **User Experience (UX) Enhancements**: The text-based interface is functional but basic. A more user-friendly interface might involve clearing the screen or providing more detailed feedback.
# 6. **Edge Cases**: While some validation is present, more comprehensive checks for edge cases (e.g., very large numbers, specific formatting requirements for account numbers) could be added.


# ### Reflection on How AI Assisted Coding Productivity

# AI significantly boosts coding productivity by acting as a powerful co-pilot. For this task:

# 1. **Reduced Boilerplate**: The AI eliminated the need to write the basic class structure, method definitions, and initial validation from scratch. This is often the most time-consuming and repetitive part of starting a new module.
# 2. **Conceptualization to Code**: It translated a high-level prompt ("bank account system with class, loops, conditionals") directly into working code, bridging the gap between idea and implementation very quickly.
# 3. **Learning and Best Practices**: For someone new to Python or object-oriented programming, the generated code serves as a good example of how to structure a class, use properties, and implement basic error handling. It implicitly guides towards common design patterns.
# 4. **Focus on Refinement**: Instead of spending time on initial coding, I could immediately focus on identifying areas for improvement, adding advanced features (like data persistence or multiple accounts), and refining the existing logic. This shifts the effort from creation to enhancement.
# 5. **Debugging Reduction**: The initial code is generally free of syntax errors and common logical pitfalls, reducing the time spent on early-stage debugging. Any issues are usually conceptual or related to missing features rather than fundamental code errors.

# Overall, AI didn't just write code; it provided a high-quality foundation that accelerated the entire development cycle, allowing for more strategic thinking and less tactical coding.
```

```
--- Welcome to Simple Bank Account System ---
Enter new account number (digits only): 6757
Enter account owner name: gg
Enter initial balance (optional, default 0):
Account 6757 created for gg with initial balance 0.00.

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 1
Enter amount to deposit: 6666
Deposited 6666.00. New balance: 6666.00.

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 3
Current Balance: $6666.00

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 5
Thank you for using our bank system. Goodbye!
```