AI-Assisted-Coding Week-6.1

2303A51878

B-28

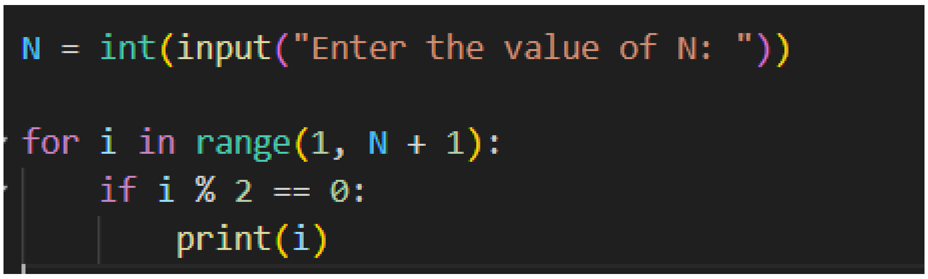
Task Description -1 (AI-Based Code Completion for Loops)

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.”

AI-Generated Python Code :-



```
N = int(input("Enter the value of N: "))

for i in range(1, N + 1):
    if i % 2 == 0:
        print(i)
```

Loop Logic Explanation :-

- The program uses a **for loop**.
- The loop iterates from **1 to N** using range(1, N + 1).
- Inside the loop, the condition i % 2 == 0 checks whether the current number is even.
- If the condition is true, the number is printed.

Identification of Loop Type Used

- **Loop Type:** for loop
- **Reason:**
  A for loop is ideal when the number of iterations (from 1 to N) is known in advance.

Validation with Sample Inputs

Sample Input 1

```
Enter the value of N: 10
```

Sample Output 1

```
2
4
6
8
10
```

Sample Input 2

```
Enter the value of N: 7
```

Sample Output 2

```
2
4
6
```

Conclusion

The AI-generated program correctly:

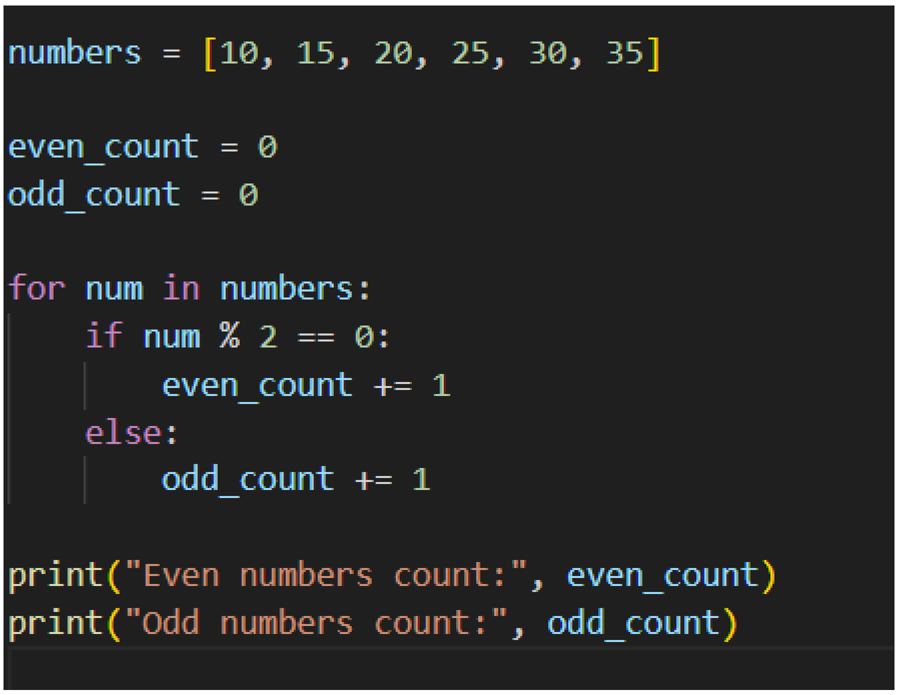
- Uses a loop structure
- Identifies even numbers using a logical condition
- Produces accurate output for different input values

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."

AI-Generated Python Code :-



```python
numbers = [10, 15, 20, 25, 30, 35]

even_count = 0
odd_count = 0

for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

print("Even numbers count:", even_count)
print("Odd numbers count:", odd_count)
```

Loop and Conditional Used

- Loop Type: for loop
- Conditional Statement: if-else

Explanation of Logic Flow

1. A list of integers is defined.
2. Two counters (even_count and odd_count) are initialized to zero.
3. The for loop iterates through each element in the list.
4. For every number:
    - If the number is divisible by 2, it is counted as even.

    - Otherwise, it is counted as odd.

5. After the loop finishes, the total count of even and odd numbers is displayed.

Validation with Sample Data

Sample Input

numbers = [10, 15, 20, 25, 30, 35]

Output

Even numbers count: 3

Odd numbers count: 3

Conclusion

The AI-generated program successfully:

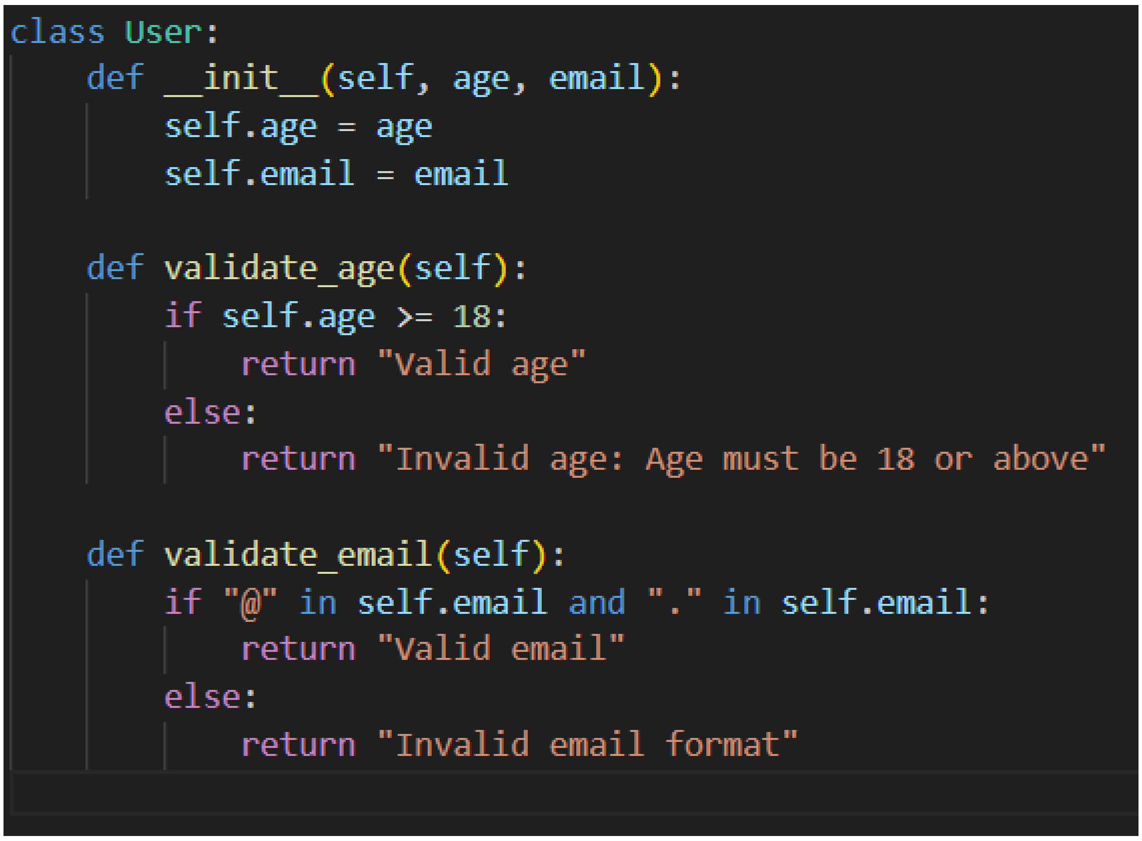
- Combines **loops** and **conditional statements**
- Accurately identifies even and odd numbers
- Produces correct counts for the given list

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.”

AI-Generated Python Class Code :-



```python
class User:
    def __init__(self, age, email):
        self.age = age
        self.email = email

    def validate_age(self):
        if self.age >= 18:
            return "Valid age"
        else:
            return "Invalid age: Age must be 18 or above"

    def validate_email(self):
        if "@" in self.email and "." in self.email:
            return "Valid email"
        else:
            return "Invalid email format"
```

Validation Logic Explanation

Age Validation

- Checks whether the user's age is 18 or above
- If true → age is valid
- Otherwise → invalid age message is returned

Email Validation

- Checks if the email contains both @ and .
- This ensures a basic valid email structure
- If not present → email is considered invalid

Verification of Condition Handling

- The program uses if-else conditional statements
- Each condition handles both valid and invalid cases
- No runtime errors occur for incorrect input formats

Test Cases

Test Case 1: Valid Input

```
user1 = User(25, "user@example.com")
print(user1.validate_age())
print(user1.validate_email())
```

Output

```
Valid age
Valid email
```

Test Case 2: Invalid Age

```
user2 = User(16, "student@example.com")
print(user2.validate_age())
```

Output

```
Invalid age: Age must be 18 or above
```

Test Case 3: Invalid Email

```
user3 = User(30, "userexample.com")
print(user3.validate_email())
```

Output

```
Invalid email format
```

Conclusion

The AI-generated class:

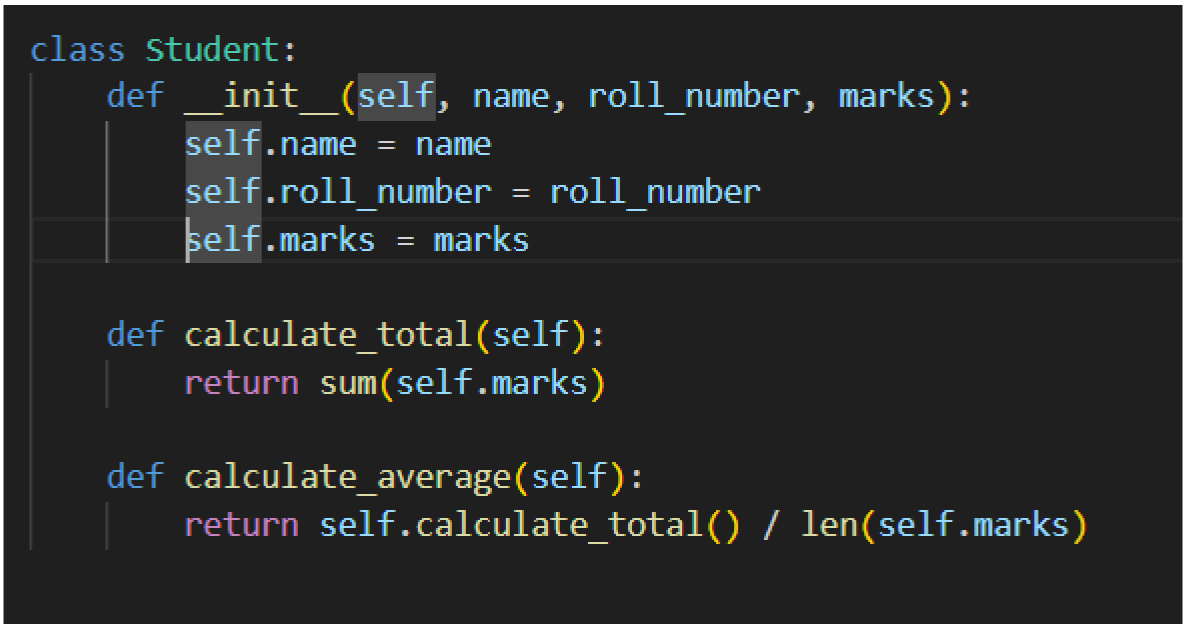
- Successfully validates age and email
- Handles both valid and invalid conditions
- Demonstrates proper use of conditional statements in a class

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."

AI-Generated Python Class Code :-



```python
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def calculate_total(self):
        return sum(self.marks)

    def calculate_average(self):
        return self.calculate_total() / len(self.marks)
```

Verification of Class Structure

Attributes

- name → stores the student's name
- roll_number → stores the student's roll number
- marks → list containing marks of subjects

Methods

- calculate_total()
    - Calculates and returns the sum of all marks
- calculate_average()
    - Calculates and returns the average marks

The class is **complete**, **well-structured**, and follows **object-oriented principles**.

Testing the Class (Correctness Validation)

Test Case

```
student1 = Student("Ravi", 101, [80, 75, 90, 85])

print("Total Marks:", student1.calculate_total())

print("Average Marks:", student1.calculate_average())
```

Output

```
Total Marks: 330

Average Marks: 82.5
```

The output confirms correct calculation of total and average marks.

Minor Manual Improvements (With Justification)

Improvement 1: Handle Empty Marks List

```
def calculate_average(self):
    if len(self.marks) == 0:
        return 0
    return self.calculate_total() / len(self.marks)
```

Justification:
Prevents **division by zero error** if no marks are provided.

Improvement 2: Basic Marks Validation

```
def calculate_total(self):
    if not all(mark >= 0 for mark in self.marks):
        return "Invalid marks detected"
    return sum(self.marks)
```

Justification:
Ensures marks are **non-negative**, improving data reliability.

Conclusion

The AI-generated Student class:

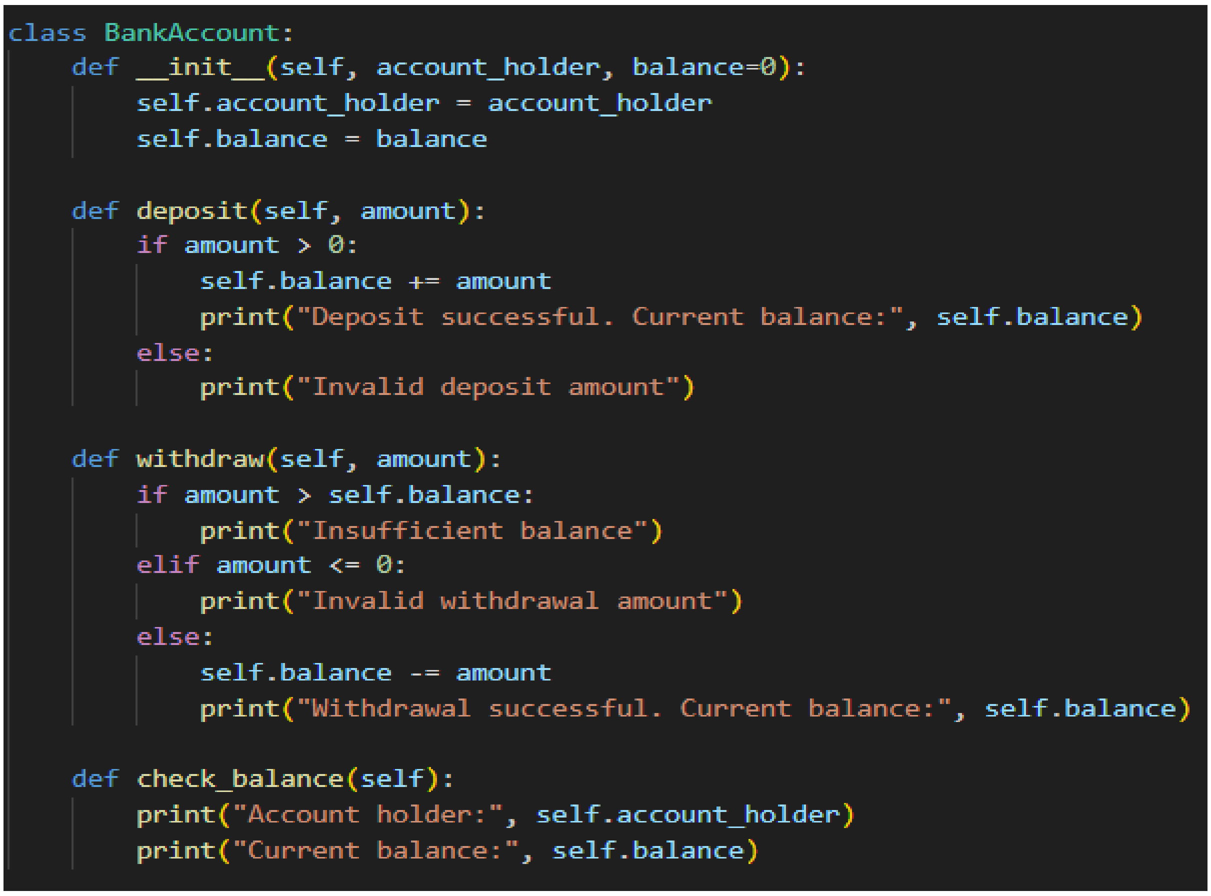
- Correctly manages student data
- Accurately calculates total and average marks
- Can be improved with small validations for robustness

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."

Complete AI-Generated Python Program :-

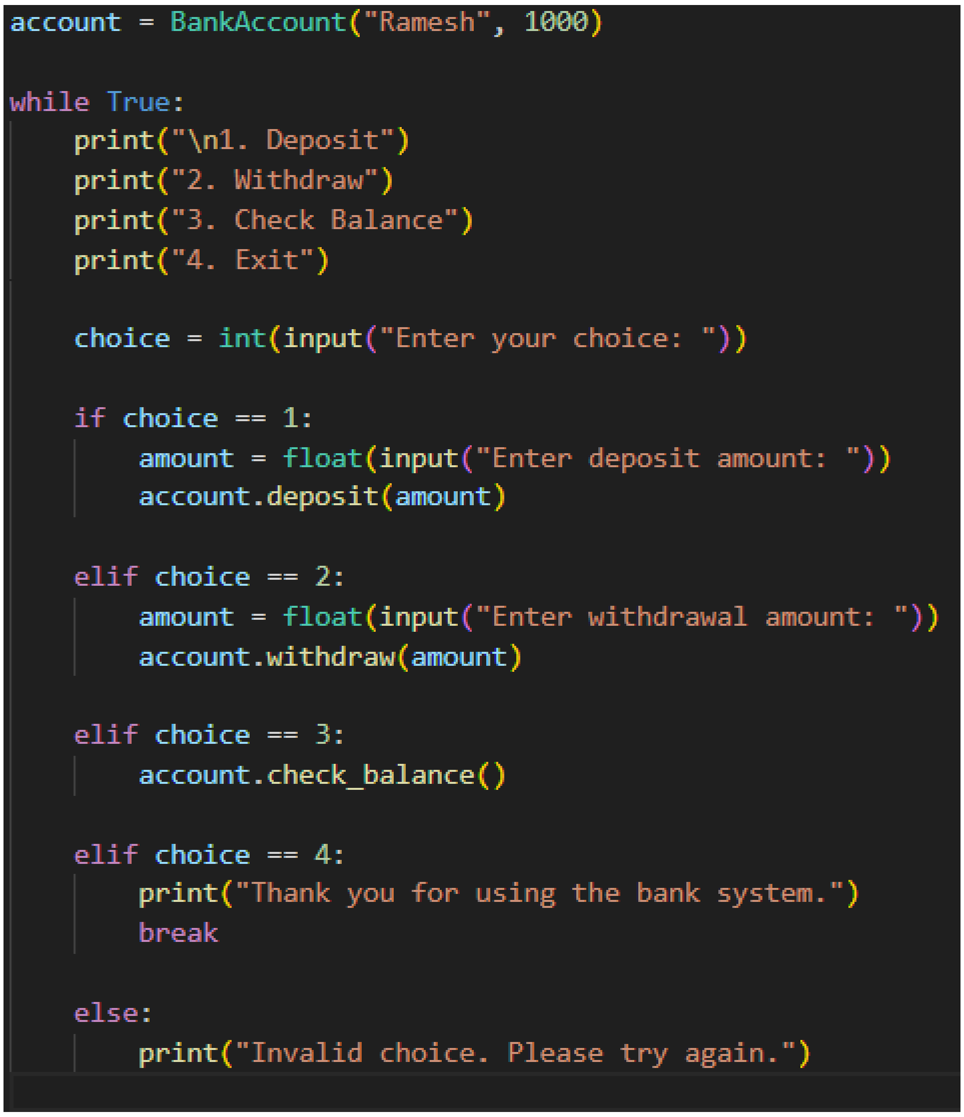


```python
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print("Deposit successful. Current balance:", self.balance)
        else:
            print("Invalid deposit amount")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance")
        elif amount <= 0:
            print("Invalid withdrawal amount")
        else:
            self.balance -= amount
            print("Withdrawal successful. Current balance:", self.balance)

    def check_balance(self):
        print("Account holder:", self.account_holder)
        print("Current balance:", self.balance)
```



```python
account = BankAccount("Ramesh", 1000)

while True:
    print("\n1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        amount = float(input("Enter deposit amount: "))
        account.deposit(amount)

    elif choice == 2:
        amount = float(input("Enter withdrawal amount: "))
        account.withdraw(amount)

    elif choice == 3:
        account.check_balance()

    elif choice == 4:
        print("Thank you for using the bank system.")
        break

    else:
        print("Invalid choice. Please try again.")
```

Explanation of Program Components

Class

- BankAccount stores account holder name and balance
- Methods: deposit(), withdraw(), check_balance()

Loops

- while True loop keeps the program running until the user exits

Conditionals

- if-elif-else used for:
    - Menu selection
    - Validating deposit and withdrawal amounts
    - Checking sufficient balance

Strengths of AI-Generated Code

- Correct use of **Object-Oriented Programming**
- Logical combination of **class, loop, and conditionals**
- Readable and well-structured code
- Handles common banking operations
- Suitable for beginners and academic assignments

Limitations of AI Suggestions

- No advanced security (PIN, authentication)
- No data persistence (data lost after program ends)
- Minimal input validation (non-numeric input may cause error)
- Single account support only

Reflection: How AI Assisted Coding Productivity

- AI quickly generated a **working program skeleton**
- Reduced development time significantly
- Helped focus on **logic rather than syntax**
- Made it easier to identify areas for improvement
- Useful as a **learning and prototyping tool**

Conclusion

The AI-assisted approach successfully generated a functional bank account system while demonstrating how AI can:

- Improve coding speed
- Support learning
- Provide a solid foundation for further enhancement