

# AI ASSISTED CODING ASS-6.1

2303A51642

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

### Loop Type Used:

- **For loop**
- The loop iterates from 1 to N using Python's range() function.

### Validation With Sample Inputs:

Sample Input:

10

Output:

2

4

6

8

**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:**

```
numbers = list(map(int, input().split()))
even_count = 0
odd_count = 0

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

print(even_count, odd_count)
```

**Correct Count Validation:****Sample Input:**

1 2 3 4 5 6

**Output:**

3 3

**Explanation of Logic Flow:**

1. The program takes a list of integers as input.
2. Two counters are initialized to track even and odd numbers.
3. A **for loop** iterates through each number in the list.
4. An **if condition** checks whether the number is divisible by 2.
  - If true, the even counter is incremented.
  - Otherwise, the odd counter is incremented.
5. Finally, the program prints the count of even and odd numbers.

## 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 with Validation Logic:

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

    def validate_age(self):
        if self.age >= 18:
            return True
        return False

    def validate_email(self):
        if "@" in self.email and "." in self.email:
            return True
        return False

    def is_valid_user(self):
        return self.validate_age() and self.validate_email()

# Example usage:
user1 = User(22, "test@example.com")
print(user1.is_valid_user())

user2 = User(16, "test@example.com")
print(user2.is_valid_user())

user3 = User(25, "testexample.com")
print(user3.is_valid_user())
```

## **Verification of Condition Handling:**

### **Age validation**

- Accepts age **18 or above**
- Rejects age below 18

### **Email validation**

- Checks for presence of @ and .
- Rejects emails missing either condition

### **Overall validation**

- User is valid only if both age and email conditions pass

## **Test Cases:**

### **Test Case 1: Valid Input**

```
user1 = User(22, "test@example.com")
print(user1.is_valid_user())
```

### **Output**

True

### **Test Case 2: Invalid Age**

```
user2 = User(16, "test@example.com")
print(user2.is_valid_user())
```

### **Output**

False

### **Test Case 3: Invalid Email**

```
user3 = User(25, "testexample.com")
print(user3.is_valid_user())
```

### **Output**

False

### **Test Case 4: Both Invalid**

```
user4 = User(15, "testexample.com")
print(user4.is_valid_user())
```

### **Output**

False

## 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:

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

    def calculate_total(self):
        total = 0
        for m in self.marks:
            total += m
        return total

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

student1 = Student("Amit", 101, [80, 75, 90])
print(student1.calculate_total())
print(student1.calculate_average())
```

### Verification of Class Structure:

#### Attributes

- name stores the student's name
- roll\_number stores the roll number

- marks stores a list of subject marks

## Methods

- calculate\_total() correctly sums all marks using a loop
- calculate\_average() calculates the average and safely handles empty marks

## Encapsulation

- Data and behavior are properly grouped inside the Student class

## Sample Test Case (Validation):

```
student1 = Student("Amit", 101, [80, 75, 90])  
print(student1.calculate_total())  
print(student1.calculate_average())
```

## Output

245

81.66666666666667

## Minor Manual Improvements (With Justification):

1. **Empty marks check in average**
  - Prevents division by zero errors
  - Makes the class more robust
2. **Reuse of calculate\_total()**
  - Avoids duplicate logic
  - Improves maintainability and readability

## 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.”

### AI-Generated Python Program:

(Simple Bank Account System using class, loops, and conditionals)

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

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

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print("Withdrawal successful")
        else:
            print("Invalid or insufficient balance")

    def show_balance(self):
        print("Current balance:", self.balance)

name = input("Enter account holder name: ")
account = BankAccount(name)

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

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

    if choice == 1:
        amount = int(input("Enter amount: "))
        account.deposit(amount)
    elif choice == 2:
        amount = int(input("Enter amount: "))
        account.withdraw(amount)
    elif choice == 3:
        account.show_balance()
    elif choice == 4:
        break
    else:
        print("Invalid choice")
```

### Sample Test Case:

#### Input (User Interaction)

Enter account holder name: Rahul

1. Deposit

Enter amount: 500

|                   |                 |
|-------------------|-----------------|
| 2. Withdraw       | 1. Deposit      |
| 3. Balance        | 2. Withdraw     |
| 4. Exit           | 3. Balance      |
| Enter choice: 1   | 4. Exit         |
| Enter choice: 2   | Enter choice: 3 |
| Enter amount: 200 | 1. Deposit      |
| 1. Deposit        | 2. Withdraw     |
| 2. Withdraw       | 3. Balance      |
| 3. Balance        | 4. Exit         |
| 4. Exit           | Enter choice: 4 |

## **Output**

Deposit successful

Withdrawal successful

Current balance: 300

## **Strengths of AI-Generated Solution:**

### **1. Clear class structure**

- Encapsulates account data and operations cleanly.
- Uses methods logically for deposit, withdrawal, and balance check.

### **2. Proper use of loops**

- while True loop allows continuous user interaction.

### **3. Effective conditionals**

- Validates deposit amount.
- Prevents withdrawal beyond available balance.
- Handles invalid menu choices.

### **4. Readable and beginner-friendly**

- Easy to understand and extend.

## **Limitations of AI Suggestions:**

### **1. No exception handling**

- Invalid inputs like letters instead of numbers may crash the program.

### **2. No transaction history**

- Only current balance is maintained.

### **3. Single account support**

- Cannot manage multiple users or accounts.

### **4. Basic security**

- No PIN or authentication mechanism.

These are acceptable limitations for a simple demonstration program.

## **Reflection: How AI Assisted Coding Productivity:**

- AI reduced development time by instantly generating a working structure.
- It helped combine **classes, loops, and conditionals** correctly without syntax errors.
- AI served as a strong starting point, allowing the programmer to focus on improvements rather than boilerplate code.
- Manual review is still essential to handle edge cases and improve robustness.