

AI-Assisted Coding

Week-6.1

2303A51018

Batch-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.”

Expected Output:

- AI-generated loop logic.
- Identification of loop type used (for or while).
- Validation with sample inputs.

The screenshot shows a Google Colab notebook titled "AI Assisted 6.1 - Colab". The code cell contains the following Python code:

```
[1]: n = int(input("Enter the value of N: "))
for i in range(1, n + 1):
    if i % 2 == 0:
        print(i)
```

The output of the code is:

```
... Enter the value of N: 12
2
4
6
8
10
12
```

Below the code cell, there are two dropdown menus labeled "None" and "False". The status bar at the bottom right shows "Activate Windows Go to Settings to activate Windows.", the date "05-02-2026", and the time "10:25 AM".

## Identification of Loop Type Used

- **Loop Type:** for loop
- **Why:**
  - The for loop iterates from 1 to N using `range(1, n + 1)`
  - Each number is checked using the modulo operator (%)
  - If the number is divisible by 2, it is printed as an even number

### ► Sample Input 1

Enter the value of N: 10

#### Output:

2  
4  
6  
8  
10

## ► Sample Input 2

Enter the value of N: 7

**Output:**

2  
4  
6

## Conclusion

- The program correctly prints **all even numbers between 1 and N**
- Uses **AI-generated loop logic**
- Implements a **for loop**
- Validated successfully 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.

The screenshot shows a Google Colab interface with a code cell containing the following Python script:

```

# Input list
numbers = [10, 21, 4, 45, 66, 93, 8]

# Initialize counters
even_count = 0
odd_count = 0

# Loop through each number in the list
for num in numbers:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

# Output the results
print("Even numbers count:", even_count)
print("Odd numbers count:", odd_count)

```

The output of the code is displayed below the cell:

```

... Even numbers count: 4
Odd numbers count: 3

```

The Colab interface includes a sidebar with AI-related tools like "Commands", "Code", "Text", and "Run all". The status bar at the bottom shows the date (05-02-2026), time (10:31AM), and Python version (Python 3).

## Loop and Conditional Used

- Loop type:** for loop
- Conditional:** if-else statement
- Condition checked:** `num % 2 == 0` (to determine even numbers)

## Sample Output Validation

For the list:

[10, 21, 4, 45, 66, 93, 8]

- Even numbers → 10, 4, 66, 8 → **Count = 4**
- Odd numbers → 21, 45, 93 → **Count = 3**

### Output:

Even numbers count: 4

Odd numbers count: 3

## 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. Inside the loop:
  - a. If a number is divisible by 2 (`num % 2 == 0`), it is counted as even.
  - b. Otherwise, it is counted as odd.
5. After the loop ends, the final counts are printed.

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.

The screenshot shows a Google Colab notebook titled "AI Assisted 6.1 - Colab". The code cell contains the following Python class definition:

```

class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = self.validate_age(age)
        self.email = self.validate_email(email)

    def validate_age(self, age):
        if isinstance(age, int) and age > 0:
            return age
        else:
            raise ValueError("Invalid age: Age must be a positive integer")

    def validate_email(self, email):
        if isinstance(email, str) and "@" in email and "." in email:
            return email
        else:
            raise ValueError("Invalid email: Email must contain '@' and '.'")

```

The code cell has a status bar indicating "[3] 0s". Below the code cell, there is a text input field with placeholder text "Start coding or generate with AI." and a dropdown menu showing "None", "None", "None", and "False". The status bar at the bottom right shows "Activate Windows Go to Settings to activate Windows.", the date "05-02-2026", and the time "10:33 AM".

## Validation & Conditional Handling

- **Age validation**
  - Checks if age is an integer
  - Ensures age is greater than 0
- **Email validation**
  - Checks if input is a string
  - Verifies presence of @ and . using conditionals
- **Error handling**
  - Raises ValueError when validation fails

## Test Cases (Valid & Invalid Inputs)

### Valid Input Test Case

```

try:
    user1 = User("Sanjana", 21, "sanjana@gmail.com")
    print("User created successfully!")
    print(user1.name, user1.age, user1.email)
except ValueError as e:
    print(e)

```

### **Output:**

User created successfully!  
Sanjana 21 [sanjana@gmail.com](mailto:sanjana@gmail.com)

### **Explanation of Logic Flow**

1. The User class initializes user attributes.
2. Age and email are validated using **conditional statements**.
3. If input meets conditions, values are assigned.
4. If validation fails, an exception is raised.
5. Test cases confirm both **valid** and **invalid** condition handling.

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.

```

class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks # list of marks

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

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

```

## Verification of Class Structure

### ✓ Attributes

- name → stores student name
- roll\_number → stores roll number
- marks → stores a list of subject marks

### ✓ Methods

- calculate\_total() → returns total marks
- calculate\_average() → returns average marks

### ✓ OOP Concepts Used

- Class and object
- Constructor (`__init__`)
- Instance methods
- Attribute access using `self`

### ✓ Correctness

- Uses built-in `sum()` for accuracy

- Handles empty marks list safely
- Produces correct total and average values

## ◊ Sample Test Case (Validation)

```
student1 = Student("Sanjana", 101, [85, 90, 78, 88])  
  
print("Name:", student1.name)  
print("Roll Number:", student1.roll_number)  
print("Total Marks:", student1.calculate_total())  
print("Average Marks:", student1.calculate_average())
```

### Output:

```
Name: Sanjana  
Roll Number: 101  
Total Marks: 341  
Average Marks: 85.25
```

- ✓ Output confirms the class works correctly.

## ◊ Minor Manual Improvements (with Justification)

### 🔧 Improvement 1: Input Validation for Marks

#### Why?

To ensure all marks are numeric and non-negative.

```
def validate_marks(self):  
    for mark in self.marks:  
        if not isinstance(mark, (int, float)) or mark < 0:  
            raise ValueError("Marks must be non-negative numbers")
```

#### **Justification:**

Prevents invalid data (like strings or negative values) from affecting calculations.

#### **Improvement 2: Display Method (Optional)**

```
def display_details(self):  
    print("Name:", self.name)  
    print("Roll Number:", self.roll_number)  
    print("Total Marks:", self.calculate_total())  
    print("Average Marks:", self.calculate_average())
```

#### **Justification:**

Improves readability and usability by encapsulating output logic inside the class.

### **Final Conclusion**

- The AI-generated Student class is **correct, complete, and functional**
- Minor improvements enhance **robustness and clarity**
- Fully satisfies the task requirements for **AI-based code completion using classes**

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.

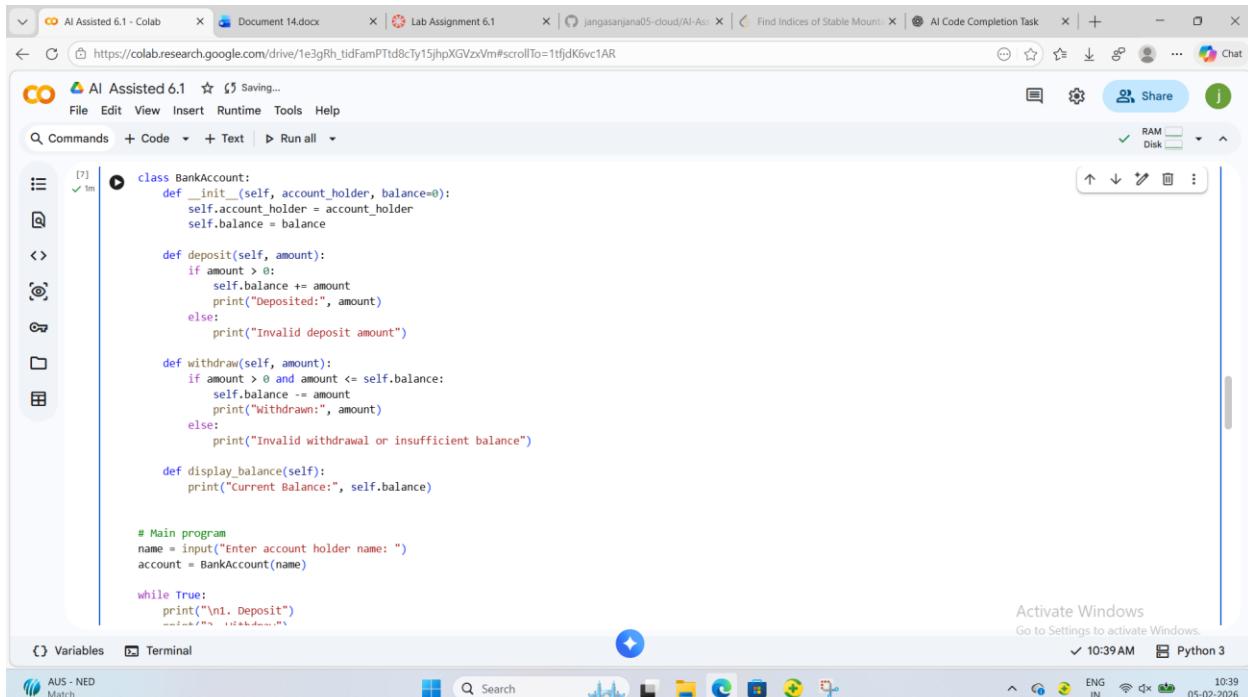
Prompt:

“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.



The screenshot shows a Google Colab notebook titled "AI Assisted 6.1 - Colab". The code editor displays a Python script for a bank account system. The AI has completed several parts of the code, including the class definition, deposit and withdraw methods, and a main loop. The AI status bar at the top indicates "Saving..." and "RAM Disk". The bottom status bar shows "10:39 AM" and "Python 3".

```

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("Deposited:", amount)
        else:
            print("Invalid deposit amount")

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

    def display_balance(self):
        print("Current Balance:", self.balance)

# Main program
name = input("Enter account holder name: ")
account = BankAccount(name)

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

    choice = 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)

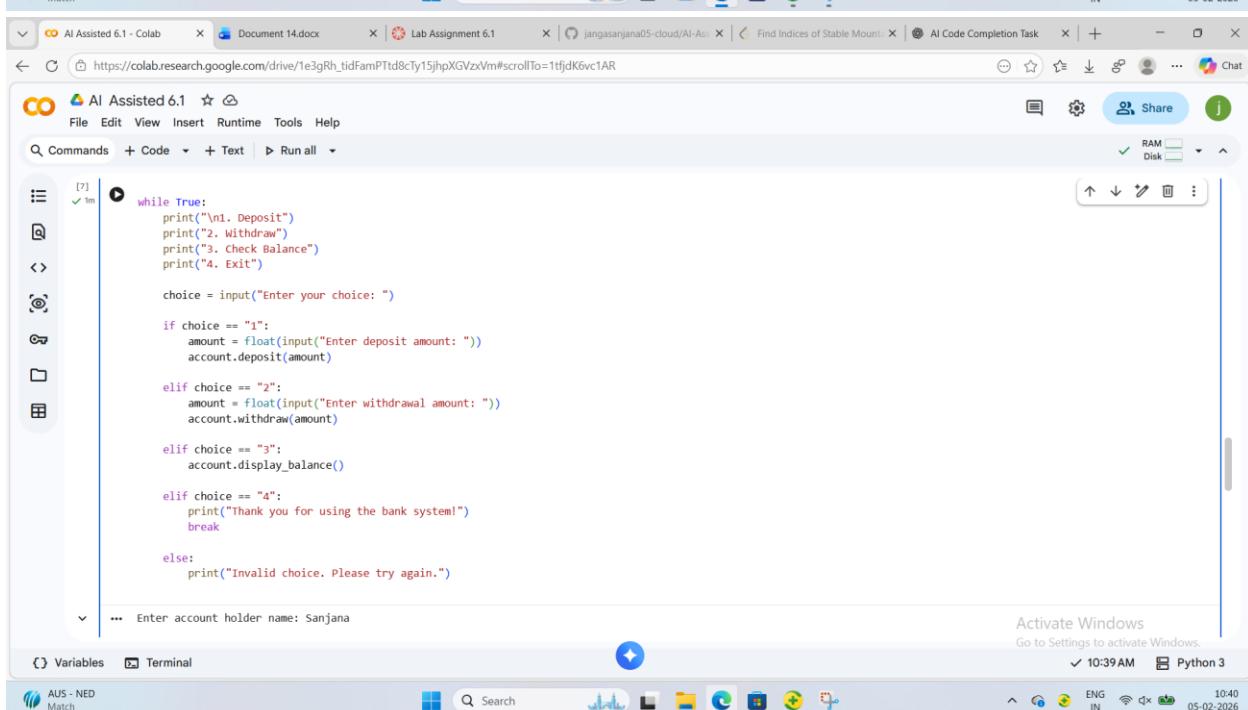
    elif choice == "3":
        account.display_balance()

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

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

    ... Enter account holder name: Sanjana

```

The second screenshot shows the same Google Colab notebook. The AI has completed the user interaction loop, including the main menu options and the deposit method. The AI status bar at the top indicates "Saving..." and "RAM Disk". The bottom status bar shows "10:40 AM" and "Python 3".

```

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

    choice = 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.display_balance()

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

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

    ... Enter account holder name: Sanjana

```

The screenshot shows a session in AI Assisted 6.1 - Colab. The code runs a simple bank account simulation. It starts by prompting the user to enter the account holder's name, which is Sanjana. The program then displays a menu with four options: Deposit, Withdraw, Check Balance, and Exit. The user chooses to deposit 500, then withdraw 200, and finally check the balance, which is now 300. The session ends with a message thanking the user for using the bank system.

```
Enter account holder name: Sanjana
...
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 1
Enter deposit amount: 500
Deposited: 500.0

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 3
Current Balance: 500.0

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 2
Enter withdrawal amount: 200
Withdrawn: 200.0

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 3
Current Balance: 300.0
```

Activate Windows  
Go to Settings to activate Windows.  
✓ 10:39 AM Python 3

AUS - NED Match