ASSIGNMENT-6.3

G.Srinidhi

2303A53023

B-46

Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals

Lab Objectives

To explore AI-powered auto-completion features for core Python constructs such as classes, loops, and conditional statements.

To analyze how AI tools suggest logic for object-oriented programming and control structures.

To evaluate the correctness, readability, and completeness of AI-generated Python code.

Lab Outcomes (LOs)

After completing this lab, students will be able to:

Use AI tools to generate and complete Python class definitions and methods.

Understand and assess AI-suggested loop constructs for iterative tasks.

Generate and evaluate conditional statements using AI-driven prompts.

Critically analyze AI-assisted code for correctness, clarity, and efficiency.

Task Description #1: Classes (Student Class) Scenario

You are developing a simple student information management module.

Task

Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.

The class should include attributes such as name, roll number, and branch.

Add a method display_details() to print student information.

Execute the code and verify the output.

Analyze the code generated by the AI tool for correctness and clarity

Expected Output #1

A Python class with a constructor (init) and a display_details() method.

Sample object creation and output displayed on the console.

Brief analysis of AI-generated code.

PROMPT:

#Generate a Python class Student with attributes name, roll number, and branch. Include constructor and display_details() method. Create sample object and print details.

CODE:

class Student:

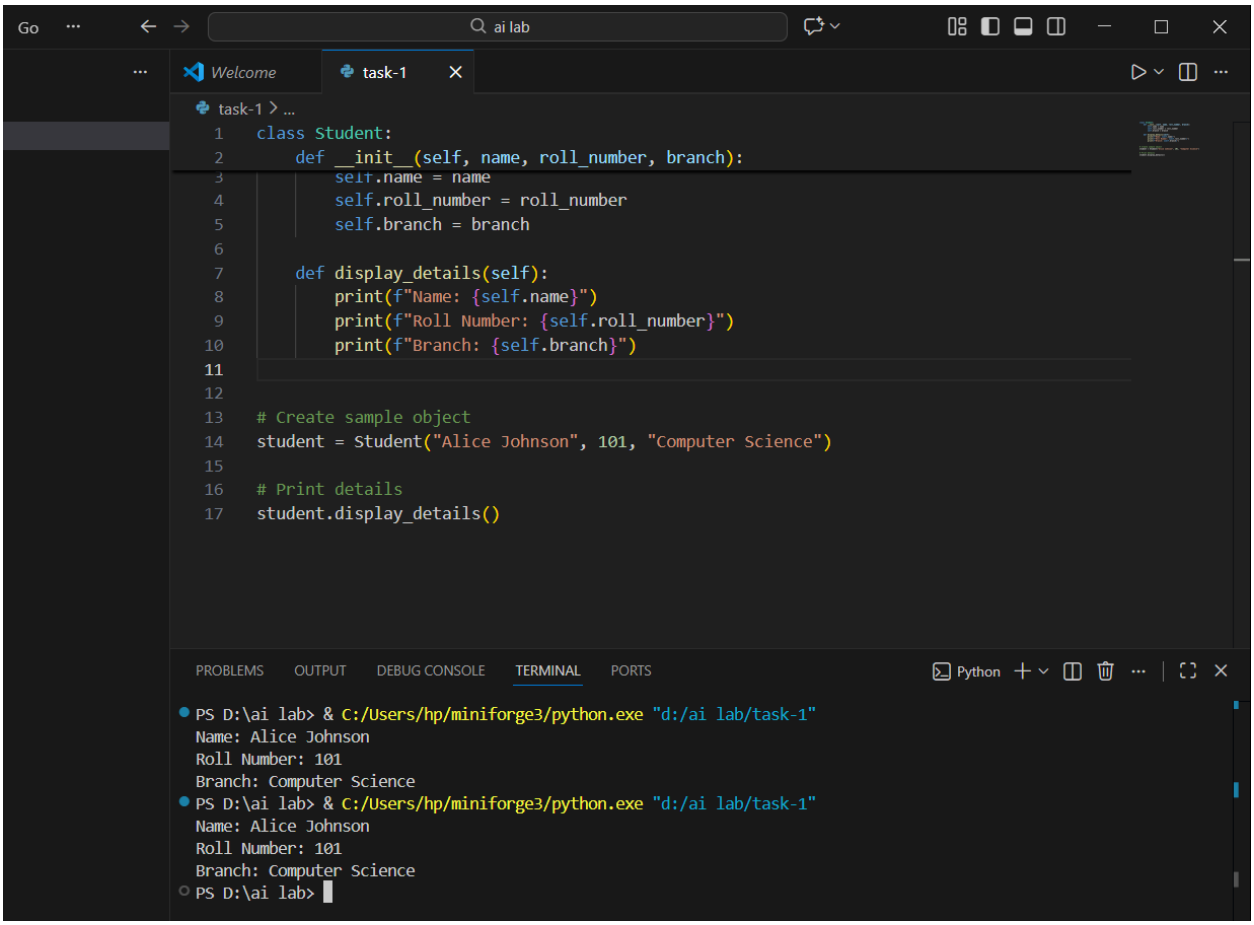
```
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")
student = Student("Alice Johnson", 101, "Computer Science")
student.display_details()
```

OUTPUT:

Name: Alice

Roll Number: 101

Branch: Computer Science



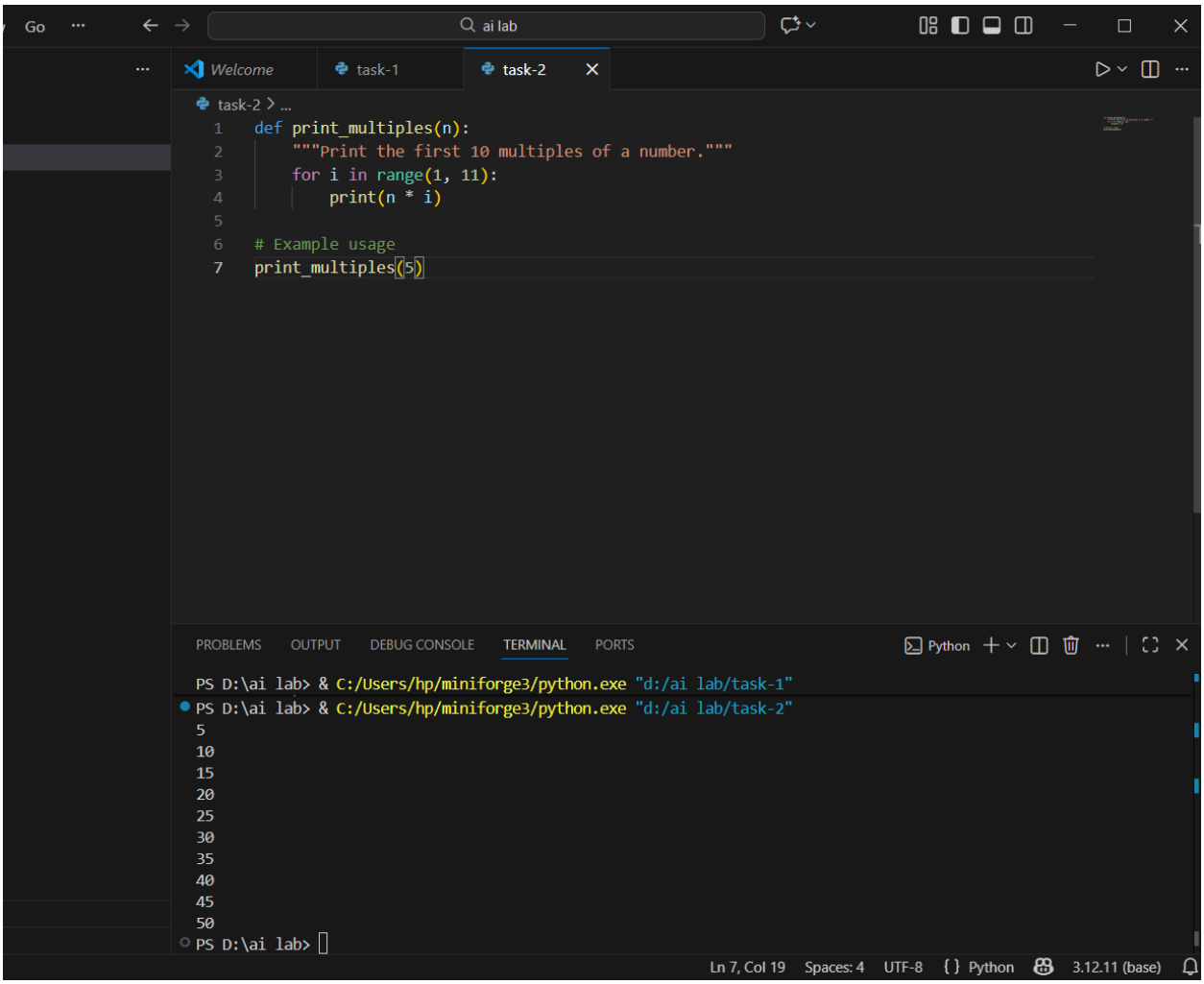
EXPLAINATION:

The Student class uses a constructor to initialize name, roll number, and branch, demonstrating basic object-oriented programming.The display_details() method prints student information clearly, showing how methods access object attributes.

Task Description #2: Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number



Task

Prompt the AI tool to generate a function that prints the first 10 multiples of a given number using a loop.

Analyze the generated loop logic.

Ask the AI to generate the same functionality using another controlled looping structure (e.g., while instead of for).

Expected Output #2

Correct loop-based Python implementation.

Output showing the first 10 multiples of a number.

Comparison and analysis of different looping approaches.

PROMPT:

#Generate Python function to print first 10 multiples of a number using loop.

CODE:

```
def print_multiples(n):
    """Print the first 10 multiples of a number."""
    for i in range(1, 11):
        print(n * i)
# Example usage
print_multiples(5)
```

EXPLAINATION:

The loop iterates from 1 to 10 and multiplies the given number each time to generate its first ten multiples. Both for and while loops produce the same result, but the for loop is simpler while the while loop gives more control.

Task Description 3: Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

Task

Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).

Analyze the generated conditions and logic.

Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).

Expected Output #3

A Python function that classifies age into appropriate groups.

Clear and correct conditional logic.

Explanation of how the conditions work.

PROMPT:

#Generate nested if-elif-else to classify age groups.

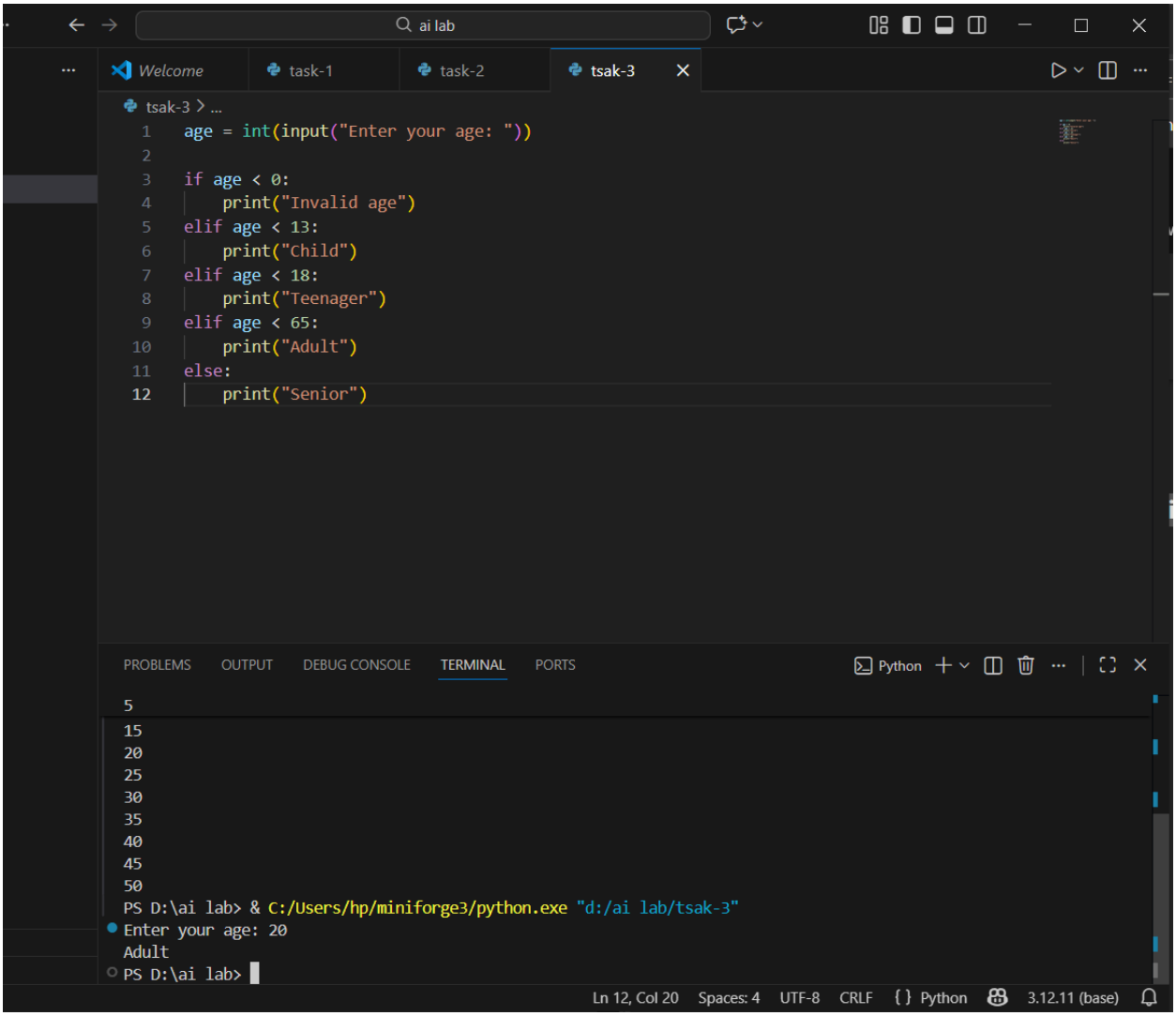
CODE:

```
age = int(input("Enter your age: "))
if age < 0:
    print("Invalid age")
elif age < 13:
    print("Child")
elif age < 18:
    print("Teenager")
elif age < 65:
    print("Adult")
```

```
else:
    print("Senior")
```

OUTPUT:

Adult



EXPLANATION:

The conditional statements check age ranges sequentially to classify a person into child, teenager, adult, or senior.

The alternative dictionary-based logic separates classification rules from conditions, improving readability and maintainability.

Task Description 4: For and While Loops (Sum of First n Numbers

Scenario

You need to calculate the sum of the first n natural numbers.

Task

Use AI assistance to generate a sum_to_n() function using a for loopAnalyze the generated code.

Ask the AI to suggest an alternative implementation using a while loop or a mathematical formula.

Expected Output #4

Python function to compute the sum of first n numbers.

Correct output for sample inputs.

Explanation and comparison of different approaches.

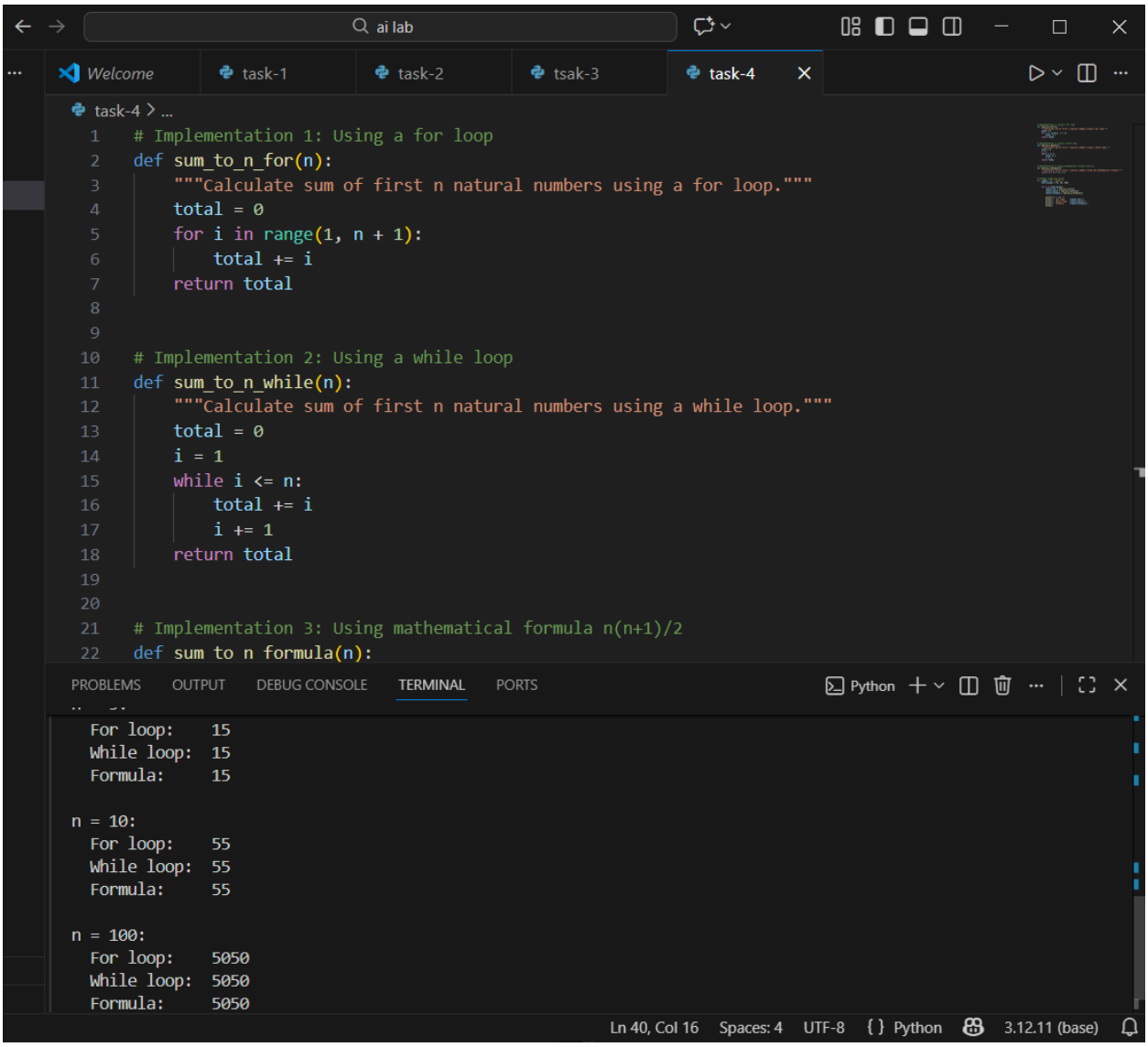
PROMPT:

#Generate a Python function sum_to_n(n) that calculates the sum of the first n natural numbers using a for loop.

#Analyze the generated code for correctness and clarity. Then provide an alternative implementation using a while loop and another using the mathematical formula n(n+1)/2, with example usage and output.

CODE:

```
def sum_to_n_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage
print(sum_to_n_for_loop(10)) # Output: 55 def
sum_to_n_while_loop(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
# Example usage
```



EXPLAINATION:

The loopbased approach adds numbers from 1 to n iteratively, demonstrating basic accumulation using control structures.

The mathematical formula computes the sum instantly in constant time, making it more efficient than loop methods.

Task Description #5: Classes (Bank Account Class) Scenario

You are designing a basic banking application.

Task

Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().

Analyze the AI-generated class structure and logic.

Add meaningful comments and explain the working of the code.

Expected Output #5

Complete Python Bank Account class.

Demonstration of deposit and withdrawal operations with updated balance.

Well-commented code with a clear explanation.

PROMPT:

#Generate BankAccount class with deposit, withdraw, and check_balance methods.

CODE:

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(f"Deposited: {amount}. New Balance: {self.balance}")
    else:
        print("Deposit amount must be positive.")
```

```
def withdraw(self, amount):
    if amount > self.balance:
        print("Insufficient funds.")
    elif amount <= 0:
        print("Withdrawal amount must be positive.")
    else:
        self.balance -= amount
        print(f"Withdrew: {amount}. New Balance: {self.balance}")

def check_balance(self):
    print(f"Current Balance: {self.balance}")
```

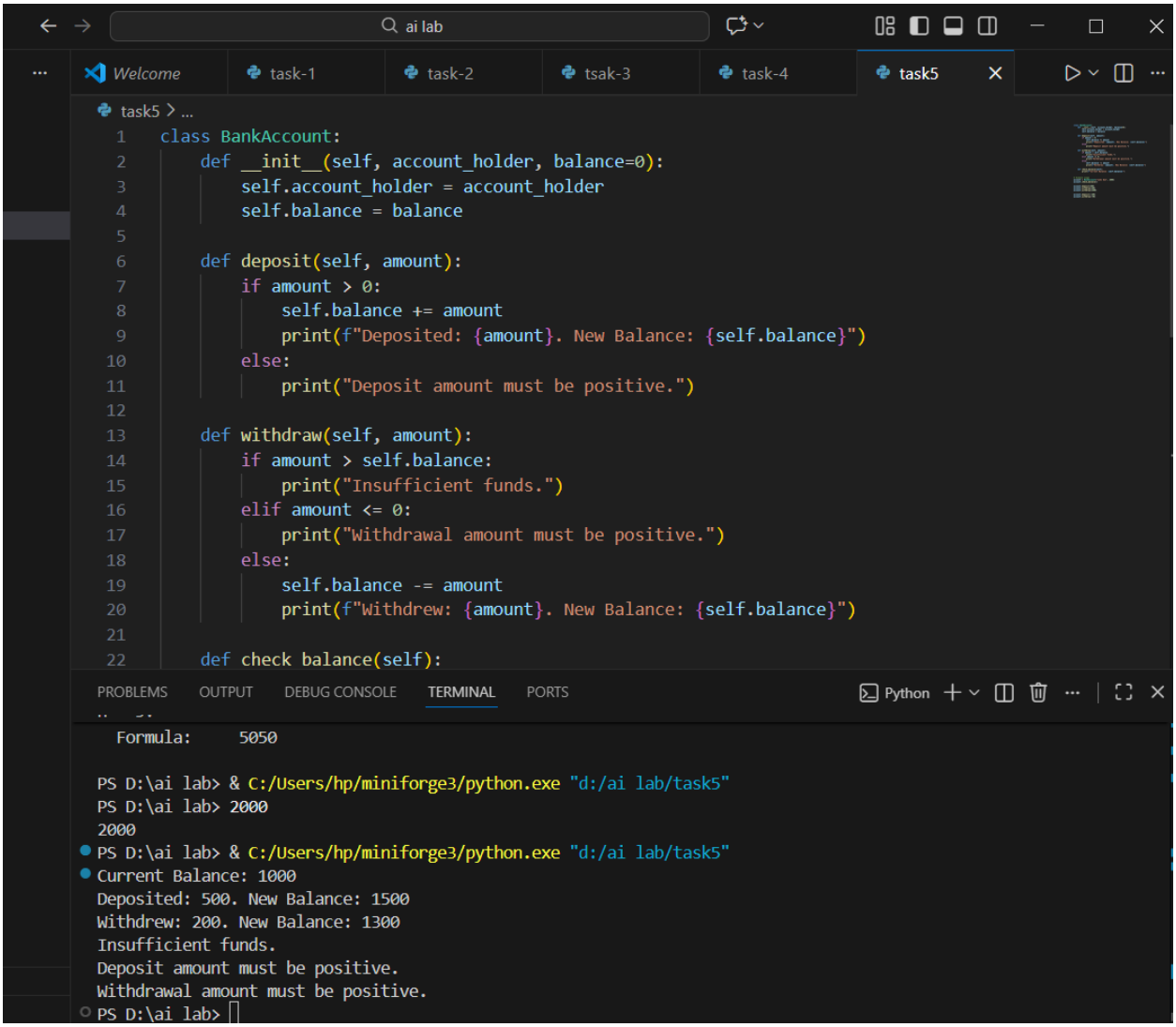
OUTPUT:

Current Balance: 1000

Deposited: 500. New Balance: 1500

Withdrew: 200. New Balance: 1300 Insufficient funds.

Deposit amount must be positive. Withdrawal amount must be positive.



EXPLANATION:

The BankAccount class models real banking operations using methods for deposit, withdrawal, and balance checking with validation.

It prevents invalid transactions and demonstrates encapsulation by controlling balance updates through class methods.