SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		**Assignment Type: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week3 - Wednesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:6.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	**Question**	***ExpectedTime to complete***
1	Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals **Lab Objectives:** • To explore AI-powered auto-completion features for core Python constructs. • To analyze how AI suggests logic for class definitions, loops, and conditionals. • To evaluate the completeness and correctness of code generated by AI assistants. **Lab Outcomes (LOs):**	Week3 - Wednesday

After completing this lab, students will be able to:

- Use AI tools to generate and complete class definitions and methods.
- Understand and assess AI-suggested loops for iterative tasks.
- Generate conditional statements through prompt-driven suggestions.
- Critically evaluate AI-assisted code for correctness and clarity.

Task Description#1 (Classes)

- Use AI to complete a Student class with attributes and a method.
- Check output
- Analyze the code generated by AI tool

Instructions:

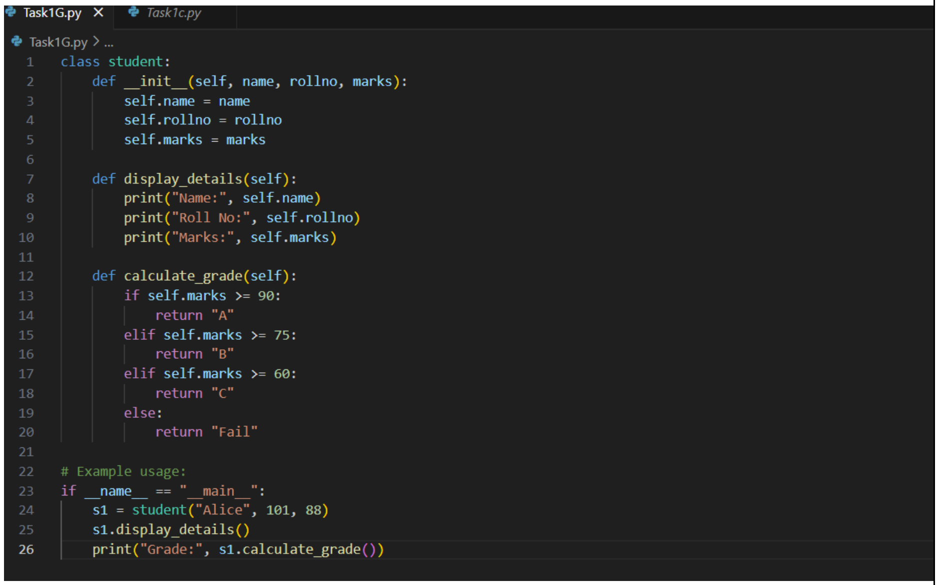
- **Initialize class with attributes like name, roll no, marks**
- **Method to display student details**
- **Method to calculate grade based on marks (A:>=90, B: >=75, C: >=60, else Fail)**

Start Writing code and auto complete using any AI tool

Expected Output#1

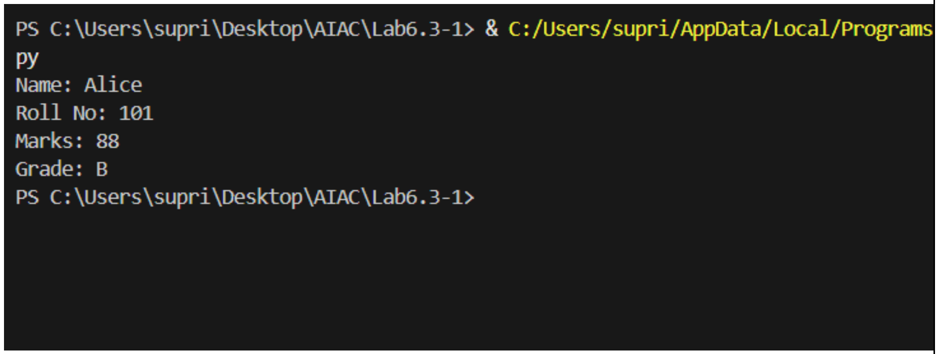
- Class with constructor and display_details() method

VS CODE:



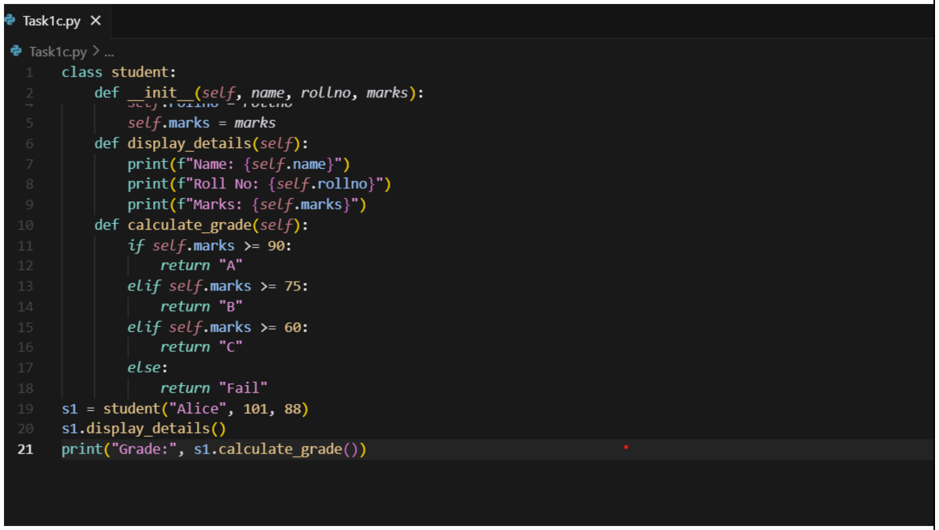
```
Task1G.py ×    Task1c.py
Task1G.py > ...
class student:
    def __init__(self, name, rollno, marks):
        self.name = name
        self.rollno = rollno
        self.marks = marks

    def display_details(self):
        print("Name:", self.name)
        print("Roll No:", self.rollno)
        print("Marks:", self.marks)

    def calculate_grade(self):
        if self.marks >= 90:
            return "A"
        elif self.marks >= 75:
            return "B"
        elif self.marks >= 60:
            return "C"
        else:
            return "Fail"

# Example usage:
if __name__ == "__main__":
    s1 = student("Alice", 101, 88)
    s1.display_details()
    print("Grade:", s1.calculate_grade())
```

OUTPUT:



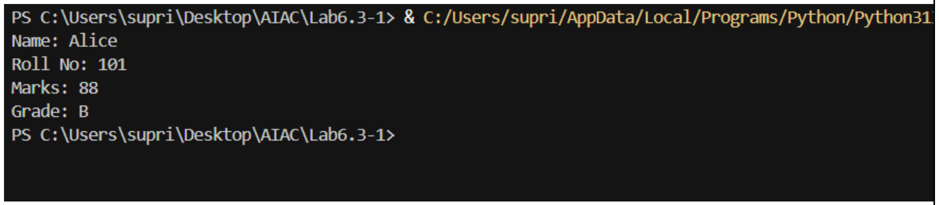
```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs
py
Name: Alice
Roll No: 101
Marks: 88
Grade: B
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

CURSOR CODE:



```
Task1c.py ×
Task1c.py > ...
class student:
    def __init__(self, name, rollno, marks):
        self.rollno = rollno
        self.marks = marks
    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.rollno}")
        print(f"Marks: {self.marks}")
    def calculate_grade(self):
        if self.marks >= 90:
            return "A"
        elif self.marks >= 75:
            return "B"
        elif self.marks >= 60:
            return "C"
        else:
            return "Fail"
s1 = student("Alice", 101, 88)
s1.display_details()
print("Grade:", s1.calculate_grade())
```

OUTPUT:



```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python31
Name: Alice
Roll No: 101
Marks: 88
Grade: B
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

Task Description#2 (Loops)

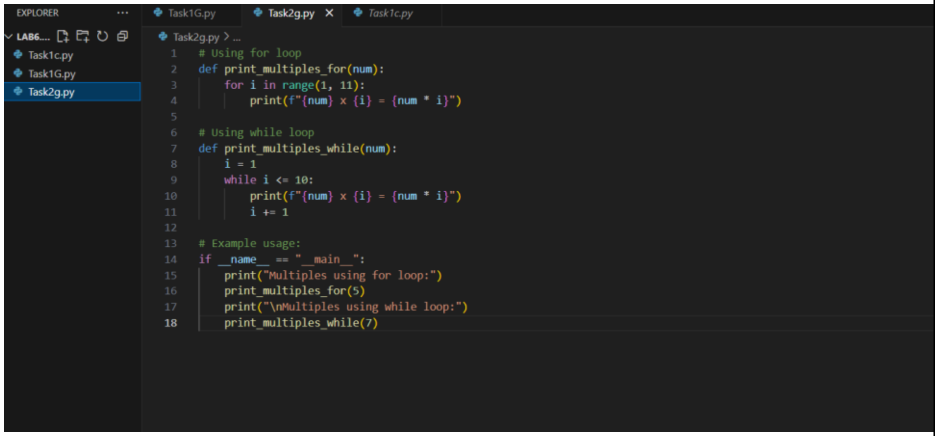
- Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.
- Analyze the generated code
- Ask AI to generate code using other controlled looping

Write code using **For** Loop, later complete code using **While** Loop

Expected Output#2

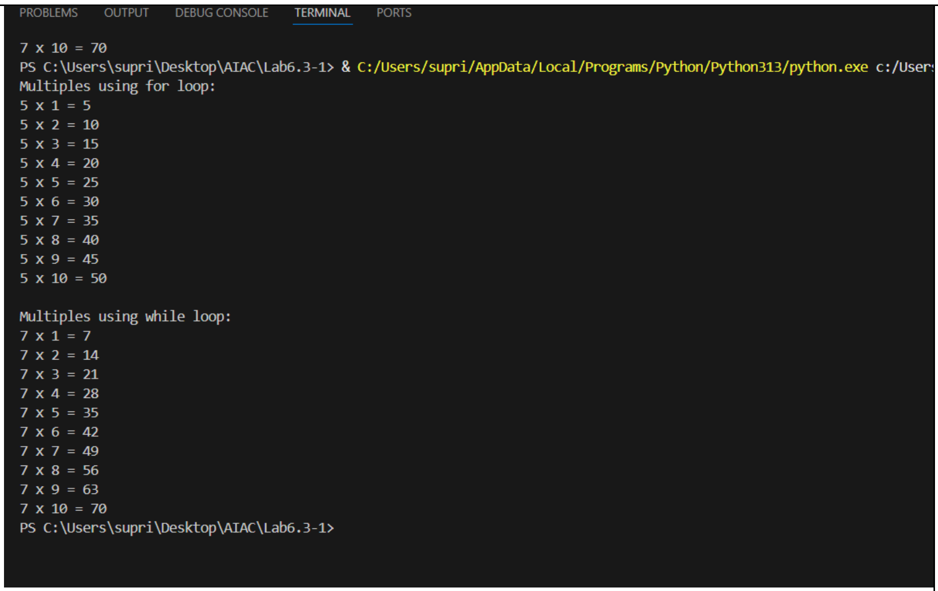
- Correct loop-based implementation

VS CODE:



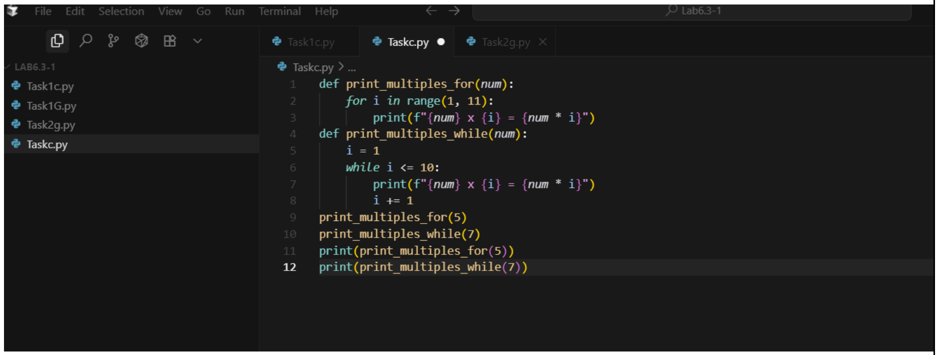
```
EXPLORER
LAB6....
Task1c.py
Task1G.py
Task2g.py

Task1G.py   Task2g.py ×   Task1c.py
Task2g.py > ...
# Using for loop
def print_multiples_for(num):
    for i in range(1, 11):
        print(f"{num} x {i} = {num * i}")

# Using while loop
def print_multiples_while(num):
    i = 1
    while i <= 10:
        print(f"{num} x {i} = {num * i}")
        i += 1

# Example usage:
if __name__ == "__main__":
    print("Multiples using for loop:")
    print_multiples_for(5)
    print("\nMultiples using while loop:")
    print_multiples_while(7)
```

OUTPUT:



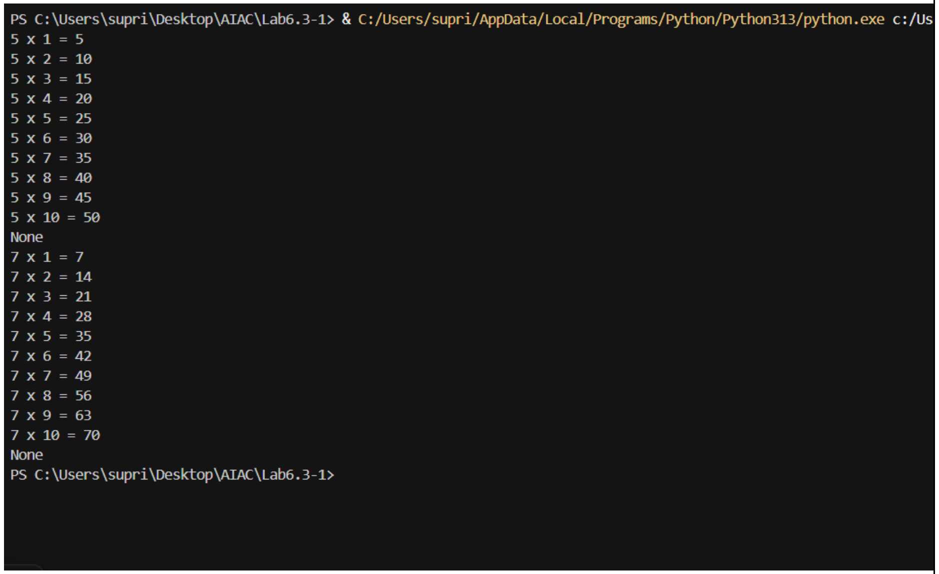
```
7 x 10 = 70
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users
Multiples using for loop:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Multiples using while loop:
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

CURSOR CODE:



```python
def print_multiples_for(num):
    for i in range(1, 11):
        print(f"{num} x {i} = {num * i}")
def print_multiples_while(num):
    i = 1
    while i <= 10:
        print(f"{num} x {i} = {num * i}")
        i += 1
print_multiples_for(5)
print_multiples_while(7)
print(print_multiples_for(5))
print(print_multiples_while(7))
```

OUTPUT:



```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Us
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
None
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
None
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

Task Description#3 (Conditional Statements)

- Ask AI to write nested if-elif-else conditionals to classify age groups.
- Analyze the generated code

- Ask AI to generate code using other conditional statements

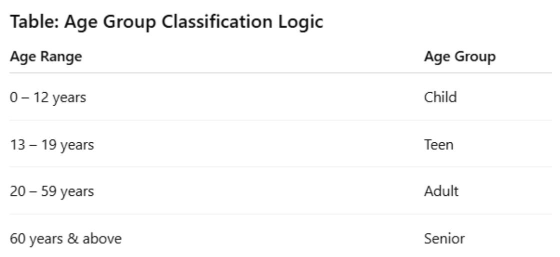


Table: Age Group Classification Logic

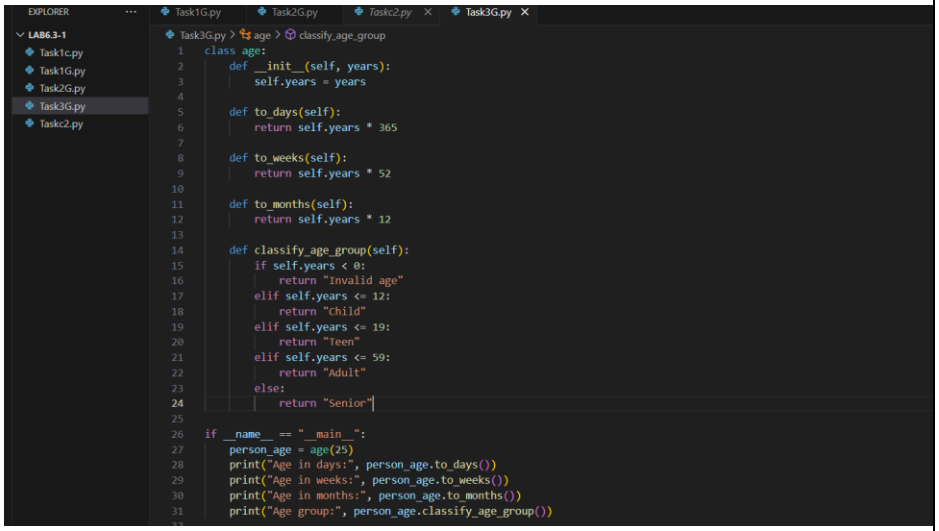
Age Range	Age Group
0 – 12 years	Child
13 – 19 years	Teen
20 – 59 years	Adult
60 years & above	Senior

-

Expected Output#3

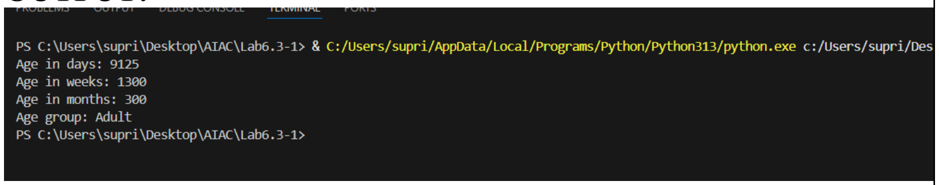
- Age classification function with appropriate conditions and with explanation

VS CODE:



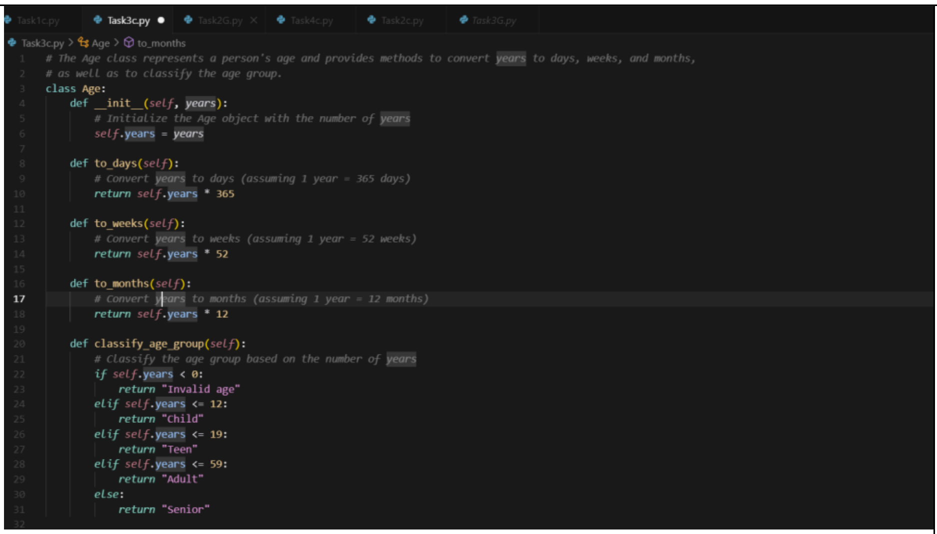
```python
class age:
    def __init__(self, years):
        self.years = years

    def to_days(self):
        return self.years * 365

    def to_weeks(self):
        return self.years * 52

    def to_months(self):
        return self.years * 12

    def classify_age_group(self):
        if self.years < 0:
            return "Invalid age"
        elif self.years <= 12:
            return "Child"
        elif self.years <= 19:
            return "Teen"
        elif self.years <= 59:
            return "Adult"
        else:
            return "Senior"

if __name__ == "__main__":
    person_age = age(25)
    print("Age in days:", person_age.to_days())
    print("Age in weeks:", person_age.to_weeks())
    print("Age in months:", person_age.to_months())
    print("Age group:", person_age.classify_age_group())
```

OUTPUT:

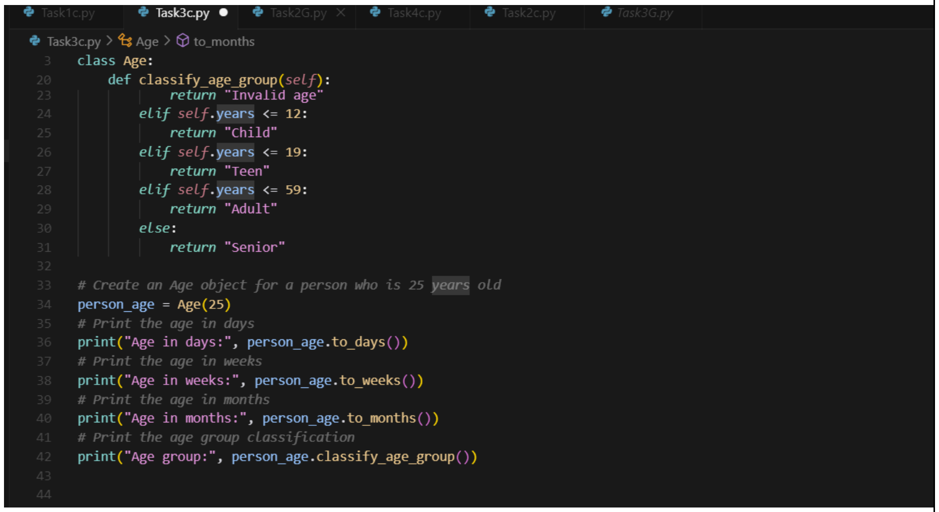


```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users/supri/Des
Age in days: 9125
Age in weeks: 1300
Age in months: 300
Age group: Adult
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

CURSOR CODE:

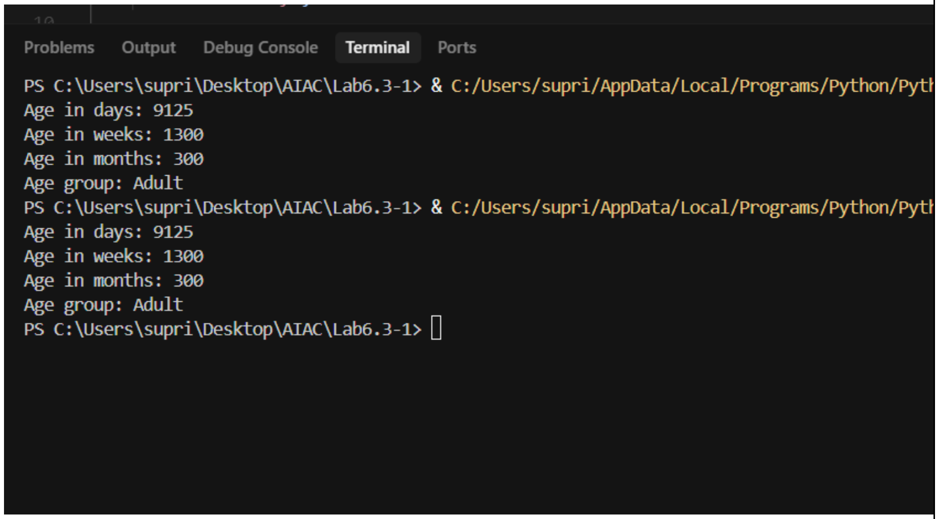


```
# The Age class represents a person's age and provides methods to convert years to days, weeks, and months,
# as well as to classify the age group.
class Age:
    def __init__(self, years):
        # Initialize the Age object with the number of years
        self.years = years

    def to_days(self):
        # Convert years to days (assuming 1 year = 365 days)
        return self.years * 365

    def to_weeks(self):
        # Convert years to weeks (assuming 1 year = 52 weeks)
        return self.years * 52

    def to_months(self):
        # Convert years to months (assuming 1 year = 12 months)
        return self.years * 12

    def classify_age_group(self):
        # Classify the age group based on the number of years
        if self.years < 0:
            return "Invalid age"
        elif self.years <= 12:
            return "Child"
        elif self.years <= 19:
            return "Teen"
        elif self.years <= 59:
            return "Adult"
        else:
            return "Senior"
```



```
class Age:
    def classify_age_group(self):
            return "Invalid age"
        elif self.years <= 12:
            return "Child"
        elif self.years <= 19:
            return "Teen"
        elif self.years <= 59:
            return "Adult"
        else:
            return "Senior"

# Create an Age object for a person who is 25 years old
person_age = Age(25)
# Print the age in days
print("Age in days:", person_age.to_days())
# Print the age in weeks
print("Age in weeks:", person_age.to_weeks())
# Print the age in months
print("Age in months:", person_age.to_months())
# Print the age group classification
print("Age group:", person_age.classify_age_group())
```

OUTPUT:



```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Pyth
Age in days: 9125
Age in weeks: 1300
Age in months: 300
Age group: Adult
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Pyth
Age in days: 9125
Age in weeks: 1300
Age in months: 300
Age group: Adult
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

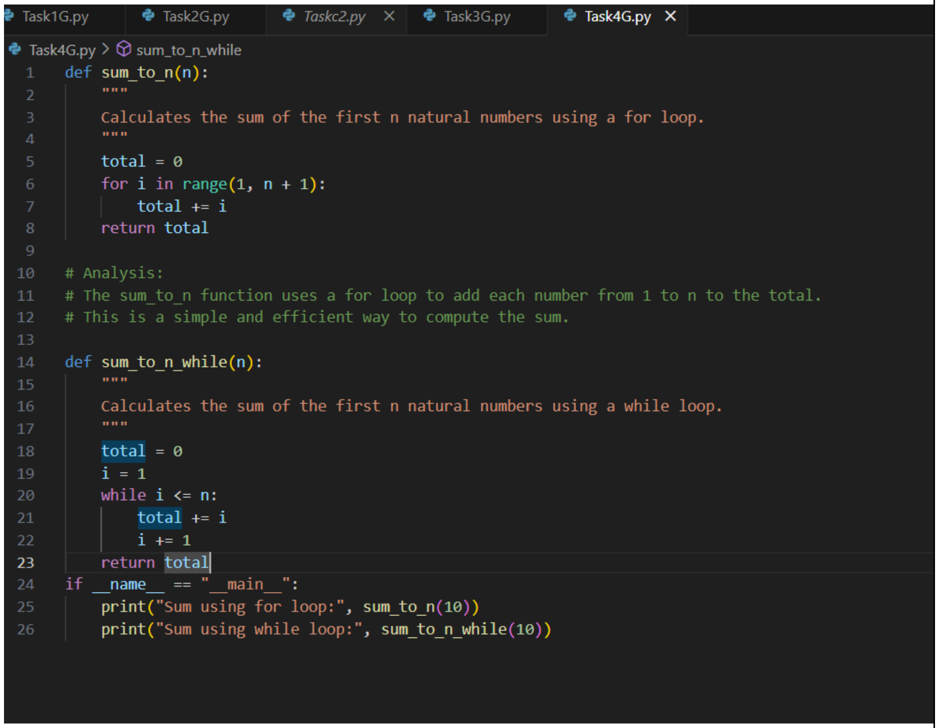
Task Description#4 (For and While loops)

- Generate a sum_to_n() function to calculate sum of first n numbers
- Analyze the generated code
- Get suggestions from AI with other controlled looping

Expected Output#4

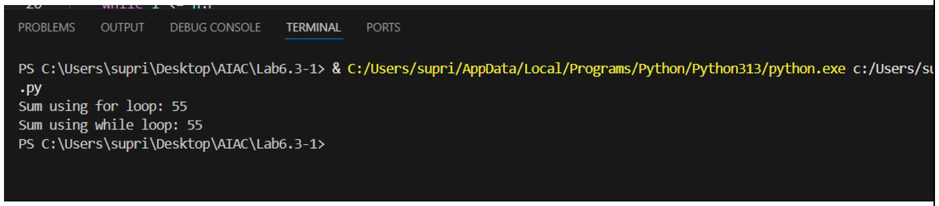
- Python code with explanation

VC CODE:



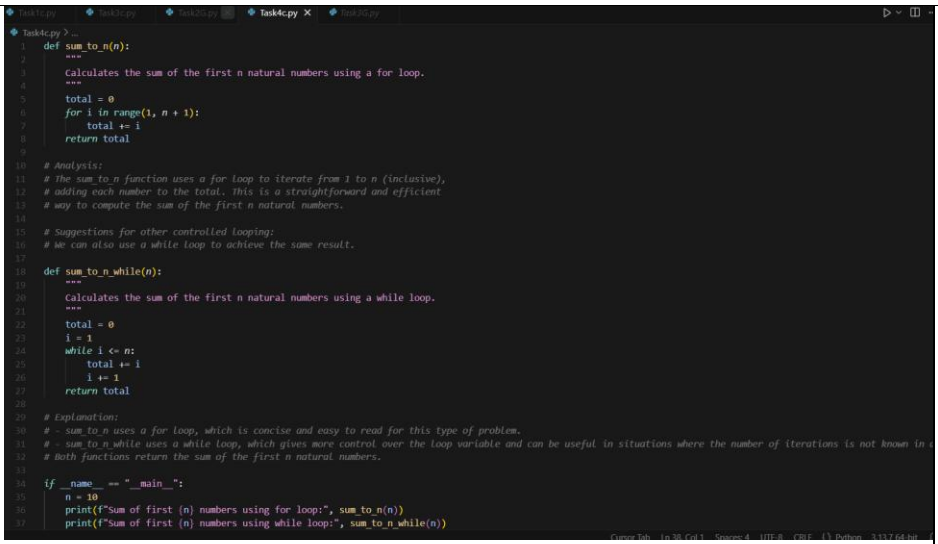
```python
def sum_to_n(n):
    """
    Calculates the sum of the first n natural numbers using a for loop.
    """
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Analysis:
# The sum_to_n function uses a for loop to add each number from 1 to n to the total.
# This is a simple and efficient way to compute the sum.

def sum_to_n_while(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.
    """
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
if __name__ == "__main__":
    print("Sum using for loop:", sum_to_n(10))
    print("Sum using while loop:", sum_to_n_while(10))
```

OUTPUT:

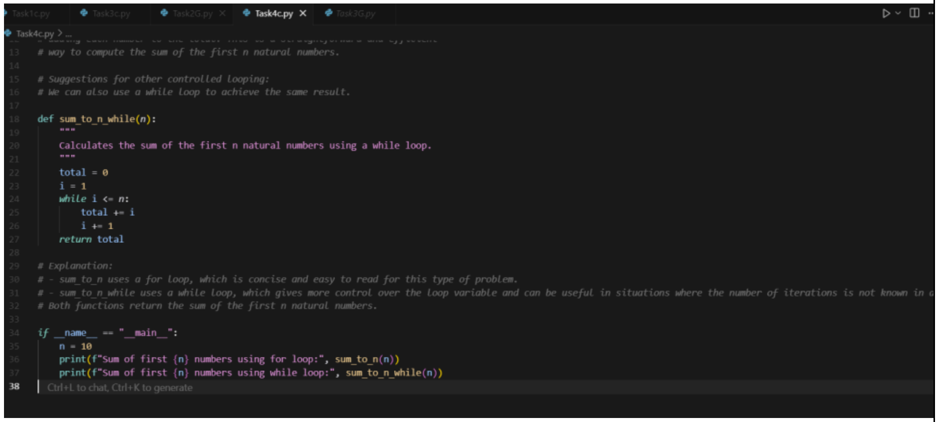


```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users/su
.py
Sum using for loop: 55
Sum using while loop: 55
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

CURSOR CODE:

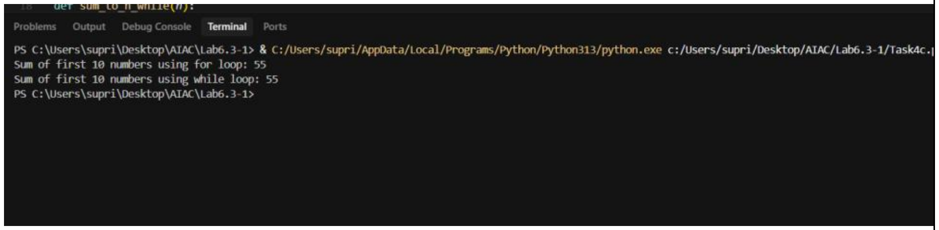


```python
def sum_to_n(n):
    """
    Calculates the sum of the first n natural numbers using a for loop.
    """
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Analysis:
# The sum_to_n function uses a for loop to iterate from 1 to n (inclusive),
# adding each number to the total. This is a straightforward and efficient
# way to compute the sum of the first n natural numbers.

# Suggestions for other controlled looping:
# We can also use a while loop to achieve the same result.

def sum_to_n_while(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.
    """
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

# Explanation:
# - sum_to_n uses a for loop, which is concise and easy to read for this type of problem.
# - sum_to_n_while uses a while loop, which gives more control over the loop variable and can be useful in situations where the number of iterations is not known in 
# Both functions return the sum of the first n natural numbers.

if __name__ == "__main__":
    n = 10
    print(f"Sum of first {n} numbers using for loop:", sum_to_n(n))
    print(f"Sum of first {n} numbers using while loop:", sum_to_n_while(n))
```



```python
# way to compute the sum of the first n natural numbers.

# Suggestions for other controlled looping:
# We can also use a while loop to achieve the same result.

def sum_to_n_while(n):
    """
    Calculates the sum of the first n natural numbers using a while loop.
    """
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

# Explanation:
# - sum_to_n uses a for loop, which is concise and easy to read for this type of problem.
# - sum_to_n_while uses a while loop, which gives more control over the loop variable and can be useful in situations where the number of iterations is not known in 
# Both functions return the sum of the first n natural numbers.

if __name__ == "__main__":
    n = 10
    print(f"Sum of first {n} numbers using for loop:", sum_to_n(n))
    print(f"Sum of first {n} numbers using while loop:", sum_to_n_while(n))
```

OUTPUT:



```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users/supri/Desktop/AIAC/Lab6.3-1/Task4c.
Sum of first 10 numbers using for loop: 55
Sum of first 10 numbers using while loop: 55
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

Task Description#5 (Class)

- Use AI to build a BankAccount class with deposit, withdraw, and balance methods.
- Analyze the generated code
- Add comments and explain code

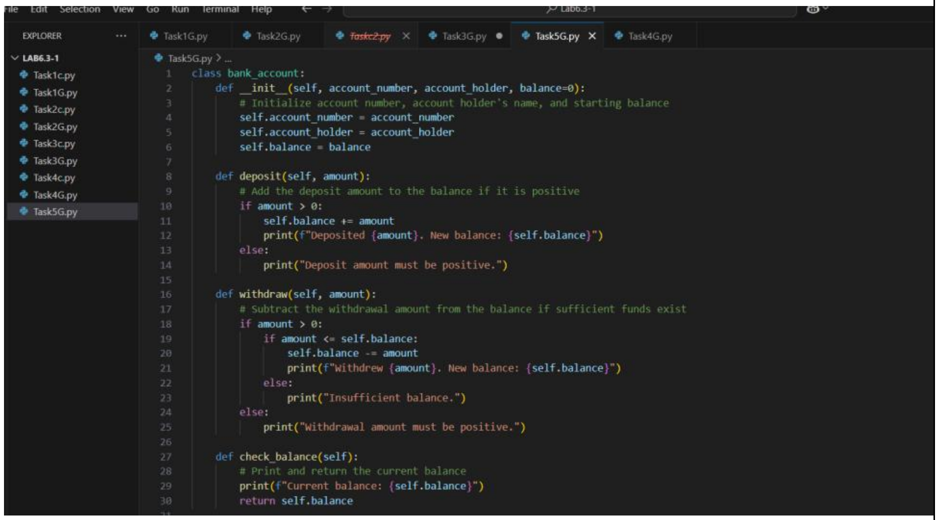
Instructions

- **Initialize BankAccount class with attributes like name, balance**
- **Method to deposit amount**
- **Method to withdraw amount**
- **Method to check balance**

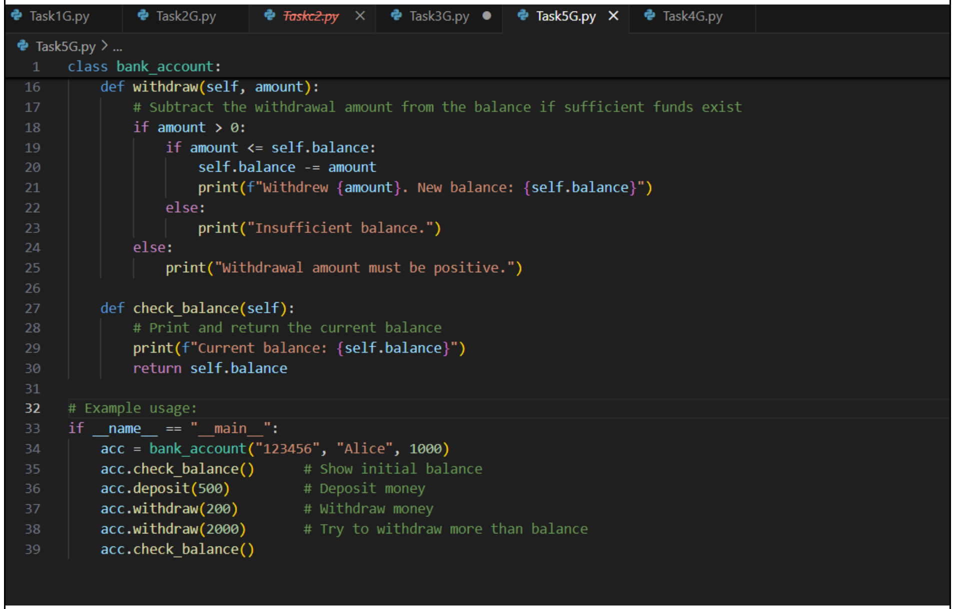
Expected Output#5

- Python code with explanation

VS CODE:

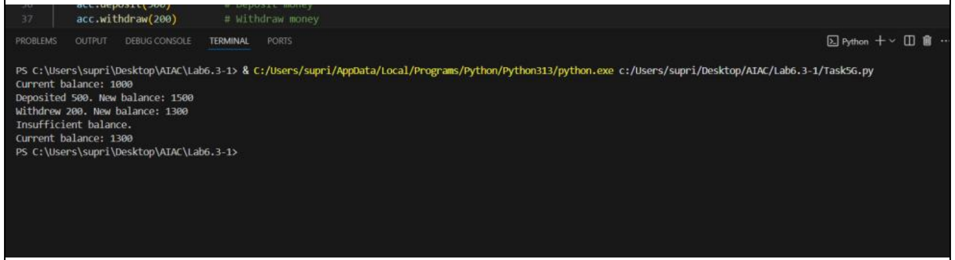


```python
class bank_account:
    def __init__(self, account_number, account_holder, balance=0):
        # Initialize account number, account holder's name, and starting balance
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        # Add the deposit amount to the balance if it is positive
        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):
        # Subtract the withdrawal amount from the balance if sufficient funds exist
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew {amount}. New balance: {self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        # Print and return the current balance
        print(f"Current balance: {self.balance}")
        return self.balance
```



```python
class bank_account:
    def withdraw(self, amount):
        # Subtract the withdrawal amount from the balance if sufficient funds exist
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew {amount}. New balance: {self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        # Print and return the current balance
        print(f"Current balance: {self.balance}")
        return self.balance

# Example usage:
if __name__ == "__main__":
    acc = bank_account("123456", "Alice", 1000)
    acc.check_balance()      # Show initial balance
    acc.deposit(500)         # Deposit money
    acc.withdraw(200)        # Withdraw money
    acc.withdraw(2000)       # Try to withdraw more than balance
    acc.check_balance()
```

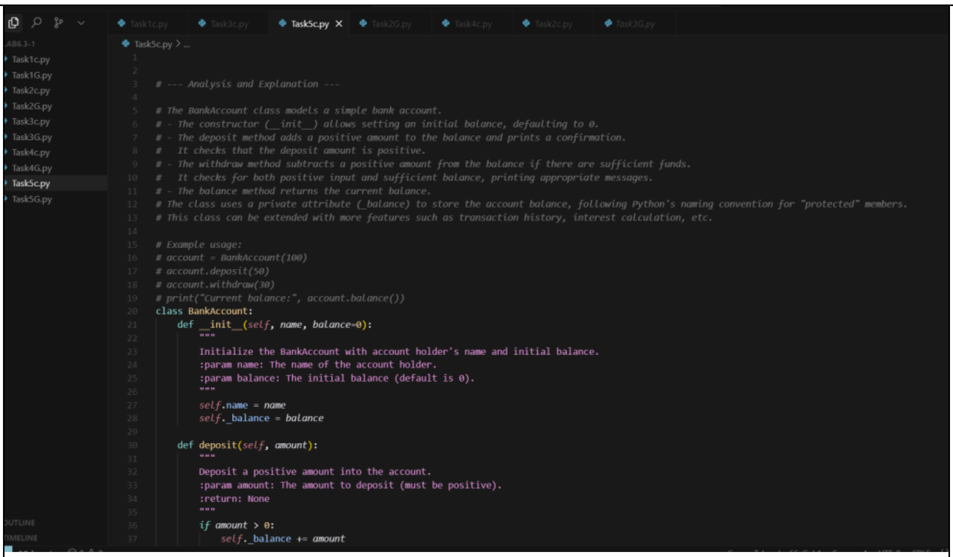
OUTPUT:



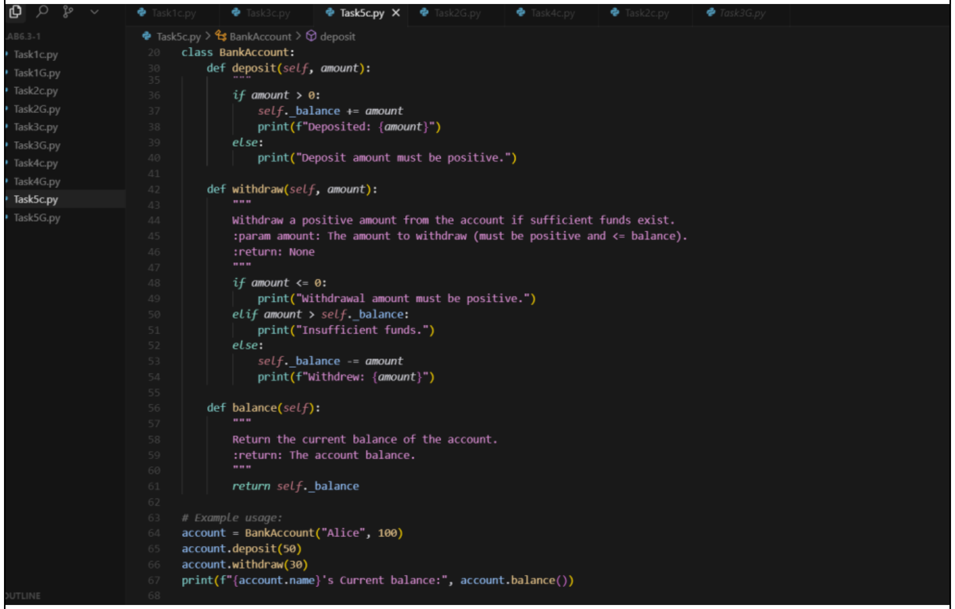
```
    acc.withdraw(200)          # Withdraw money
```

```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users/supri/Desktop/AIAC/Lab6.3-1/Task5G.py
Current balance: 1000
Deposited 500. New balance: 1500
Withdrew 200. New balance: 1300
Insufficient balance.
Current balance: 1300
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

CURSOR CODE:

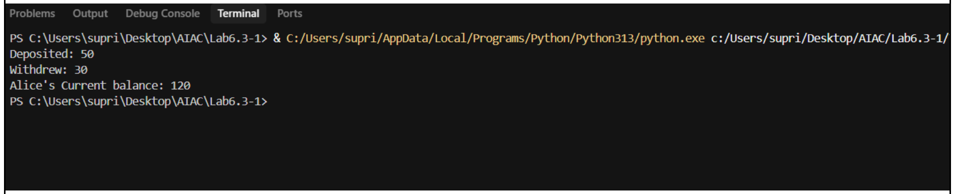


```python


# --- Analysis and Explanation ---

# The BankAccount class models a simple bank account.
# - The constructor (__init__) allows setting an initial balance, defaulting to 0.
# - The deposit method adds a positive amount to the balance and prints a confirmation.
#   It checks that the deposit amount is positive.
# - The withdraw method subtracts a positive amount from the balance if there are sufficient funds.
#   It checks for both positive input and sufficient balance, printing appropriate messages.
# - The balance method returns the current balance.
# The class uses a private attribute (_balance) to store the account balance, following Python's naming convention for "protected" members.
# This class can be extended with more features such as transaction history, interest calculation, etc.

# Example usage:
# account = BankAccount(100)
# account.deposit(50)
# account.withdraw(30)
# print("Current balance:", account.balance())
class BankAccount:
    def __init__(self, name, balance=0):
        """
        Initialize the BankAccount with account holder's name and initial balance.
        :param name: The name of the account holder.
        :param balance: The initial balance (default is 0).
        """
        self.name = name
        self._balance = balance

    def deposit(self, amount):
        """
        Deposit a positive amount into the account.
        :param amount: The amount to deposit (must be positive).
        :return: None
        """
        if amount > 0:
            self._balance += amount
```



```python
class BankAccount:
    def deposit(self, amount):
        """
        if amount > 0:
            self._balance += amount
            print(f"Deposited: {amount}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraw a positive amount from the account if sufficient funds exist.
        :param amount: The amount to withdraw (must be positive and <= balance).
        :return: None
        """
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self._balance:
            print("Insufficient funds.")
        else:
            self._balance -= amount
            print(f"Withdrew: {amount}")

    def balance(self):
        """
        Return the current balance of the account.
        :return: The account balance.
        """
        return self._balance

# Example usage:
account = BankAccount("Alice", 100)
account.deposit(50)
account.withdraw(30)
print(f"{account.name}'s Current balance:", account.balance())
```

OUTPUT:



```
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1> & C:/Users/supri/AppData/Local/Programs/Python/Python313/python.exe c:/Users/supri/Desktop/AIAC/Lab6.3-1/
Deposited: 50
Withdrew: 30
Alice's Current balance: 120
PS C:\Users\supri\Desktop\AIAC\Lab6.3-1>
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Class	1.0

	Loops	1.0		
	Conditional Statements	0.5		
	Total	**2.5 Marks**		