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

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

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

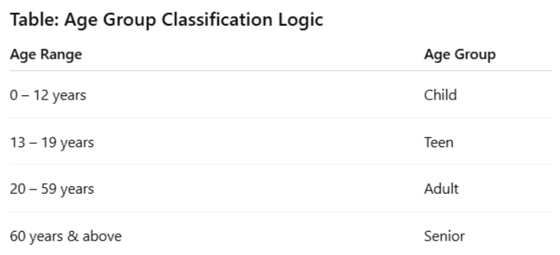


Table: Age Group Classification Logic

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

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

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

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**

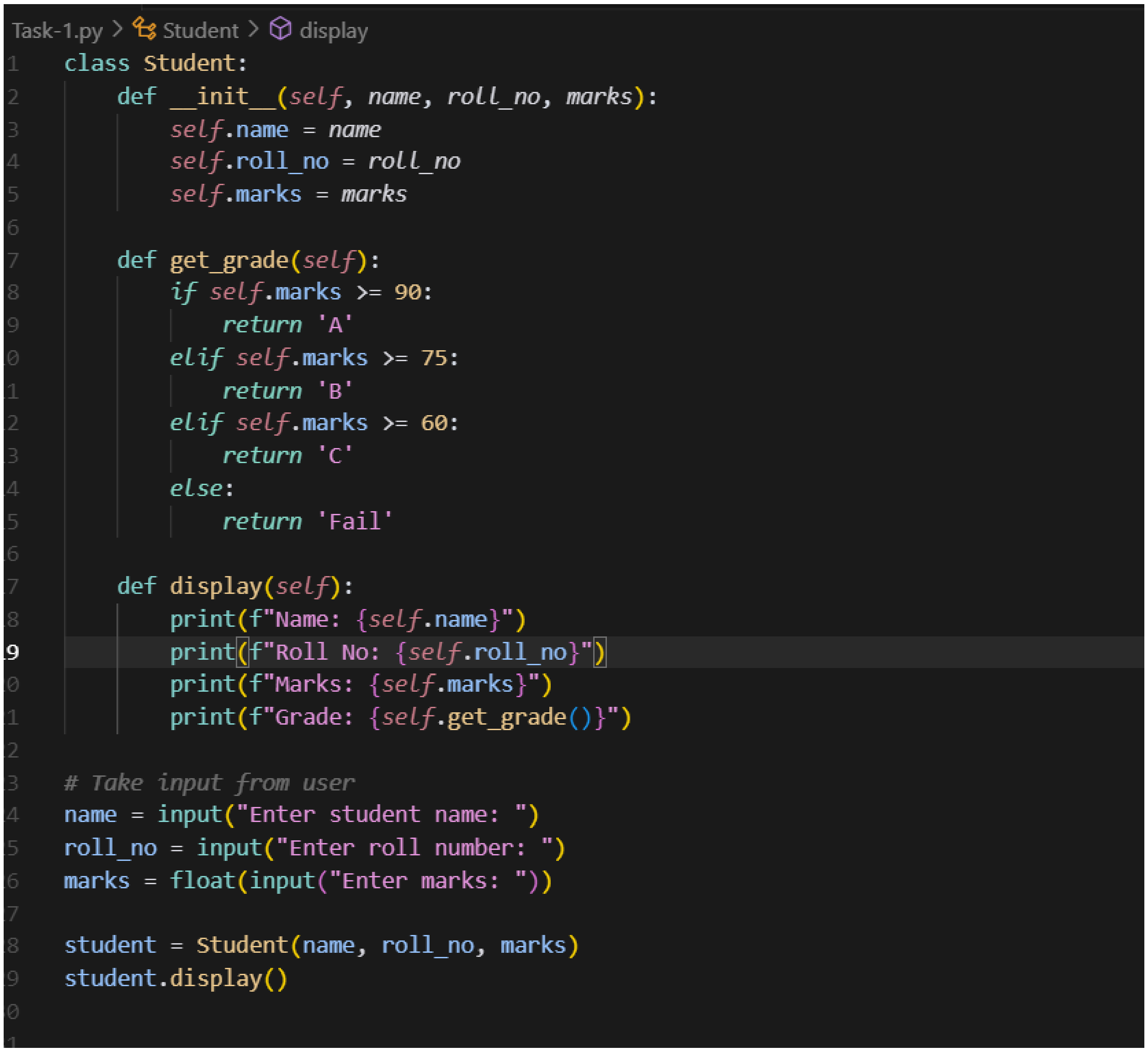
TASK_1:

PROMPT:

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

now i want a method to display this student details and give grades based on marks i.e(A:>=90, B: >=75, C: >=60, else Fail).Take input from user.

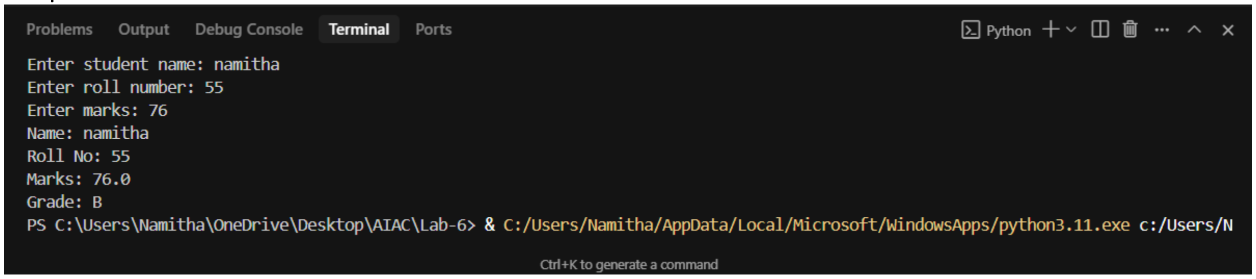
Code:



Task-1.py > Student > display

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

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

    def display(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.roll_no}")
        print(f"Marks: {self.marks}")
        print(f"Grade: {self.get_grade()}")

# Take input from user
name = input("Enter student name: ")
roll_no = input("Enter roll number: ")
marks = float(input("Enter marks: "))

student = Student(name, roll_no, marks)
student.display()
```

Output:



```
Enter student name: namitha
Enter roll number: 55
Enter marks: 76
Name: namitha
Roll No: 55
Marks: 76.0
Grade: B
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-6> & C:/Users/Namitha/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/N
```

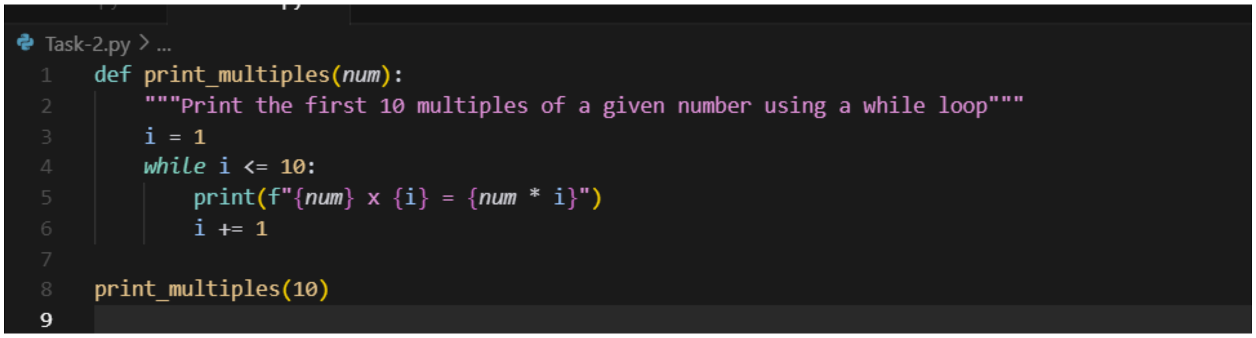
TASK-2:

PROMPT:
def print_multiples(num):
"""Print the first 10 multiples of a given number"""

for i in range(1, 11):
print(f"{num} x {i} = {num * i}")

print_multiples(10).Now generate the same code using while loop.

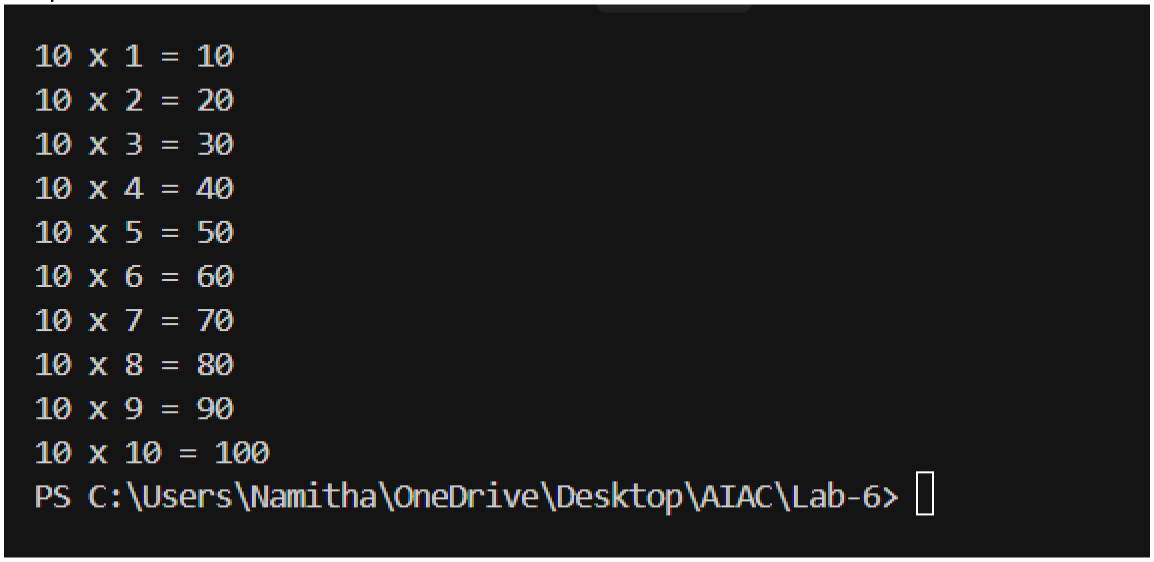
Code:



Task-2.py > ...

```python
def print_multiples(num):
    """Print the first 10 multiples of a given number using a while loop"""
    i = 1
    while i <= 10:
        print(f"{num} x {i} = {num * i}")
        i += 1

print_multiples(10)
```

Output:



```
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-6>
```

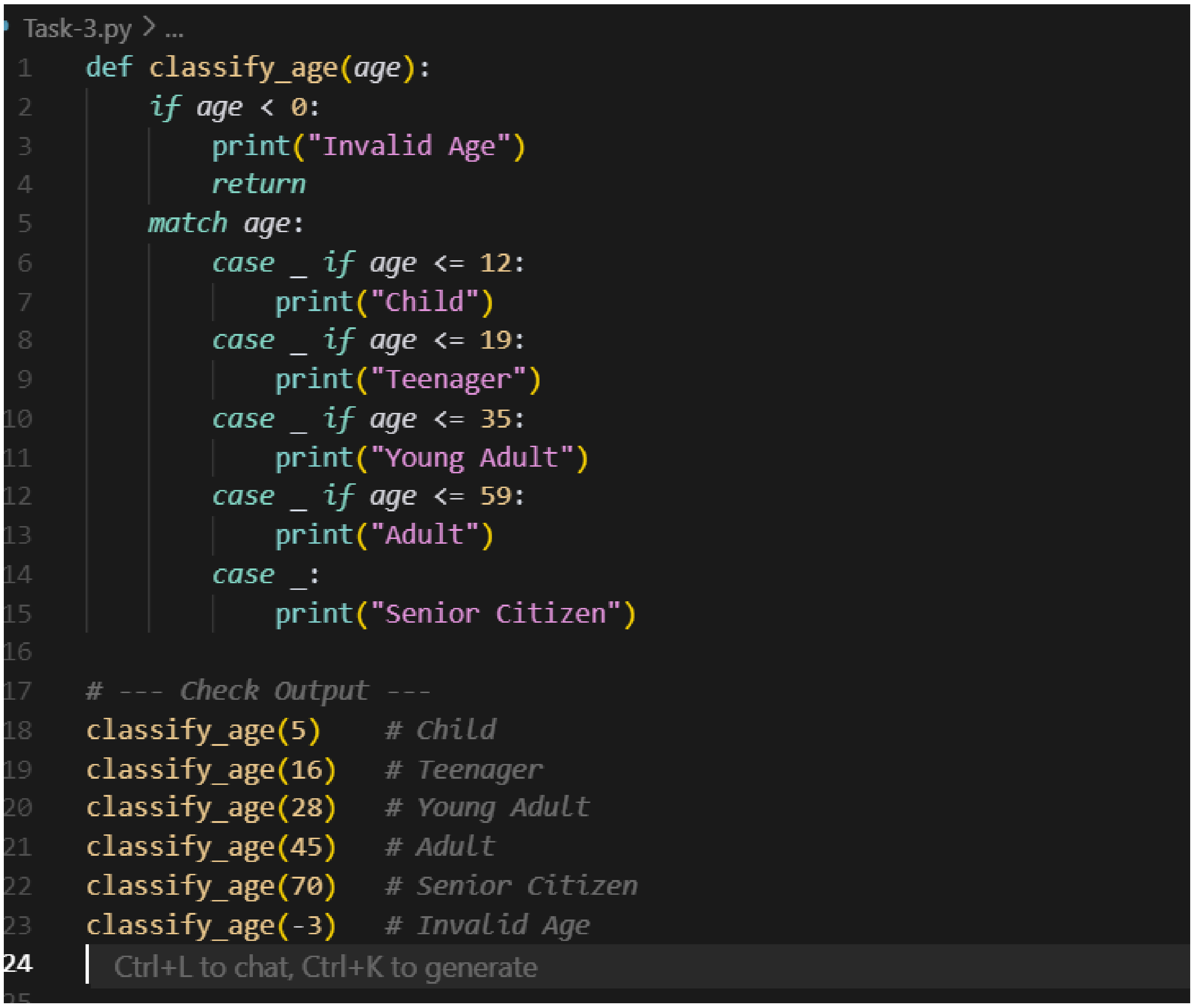
TASK-3:

PROMPT:
def classify_age(age):
if age >= 0:
if age <= 12:
print("Child")
elif age <= 19:
print("Teenager")
elif age <= 35:
print("Young Adult")

elif age <= 59:
print("Adult")
else:
print("Senior Citizen")
else:
print("Invalid Age")

--- Check Output ---
classify_age(5) # Child
classify_age(16) # Teenager
classify_age(28) # Young Adult
classify_age(45) # Adult
classify_age(70) # Senior Citizen
classify_age(-3) # Invalid Age
now write the same code with other conditional statements.

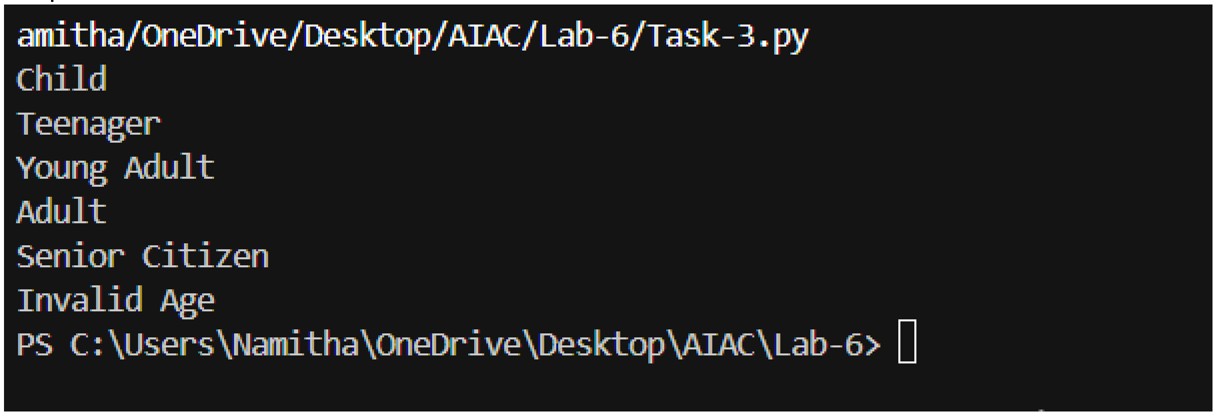
Code:



Task-3.py > ...

```python
def classify_age(age):
    if age < 0:
        print("Invalid Age")
        return
    match age:
        case _ if age <= 12:
            print("Child")
        case _ if age <= 19:
            print("Teenager")
        case _ if age <= 35:
            print("Young Adult")
        case _ if age <= 59:
            print("Adult")
        case _:
            print("Senior Citizen")

# --- Check Output ---
classify_age(5)     # Child
classify_age(16)    # Teenager
classify_age(28)    # Young Adult
classify_age(45)    # Adult
classify_age(70)    # Senior Citizen
classify_age(-3)    # Invalid Age
```

Output:



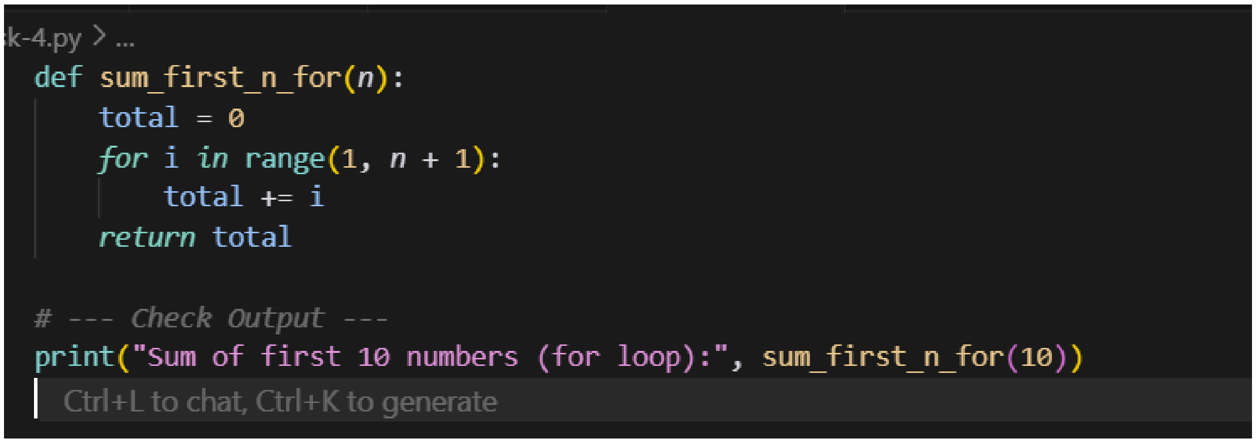
```
amitha/OneDrive/Desktop/AIAC/Lab-6/Task-3.py
Child
Teenager
Young Adult
Adult
Senior Citizen
Invalid Age
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-6>
```

TASK-4:

PROMPT:
def sum_first_n_while(n):
total = 0
i = 1
while i <= n:
total += i
i += 1
return total

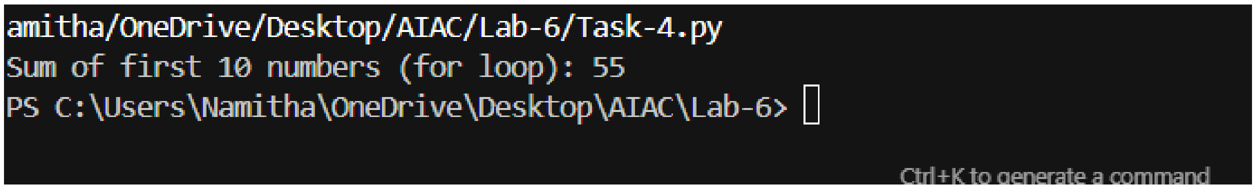
--- Check Output ---
print("Sum of first 10 numbers (while loop):", sum_first_n_while(10))
now write same code using for loop.

Code:



```
k-4.py > ...
def sum_first_n_for(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# --- Check Output ---
print("Sum of first 10 numbers (for loop):", sum_first_n_for(10))
```

Output:



```
amitha/OneDrive/Desktop/AIAC/Lab-6/Task-4.py
Sum of first 10 numbers (for loop): 55
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-6> 
```

Ctrl+K to generate a command

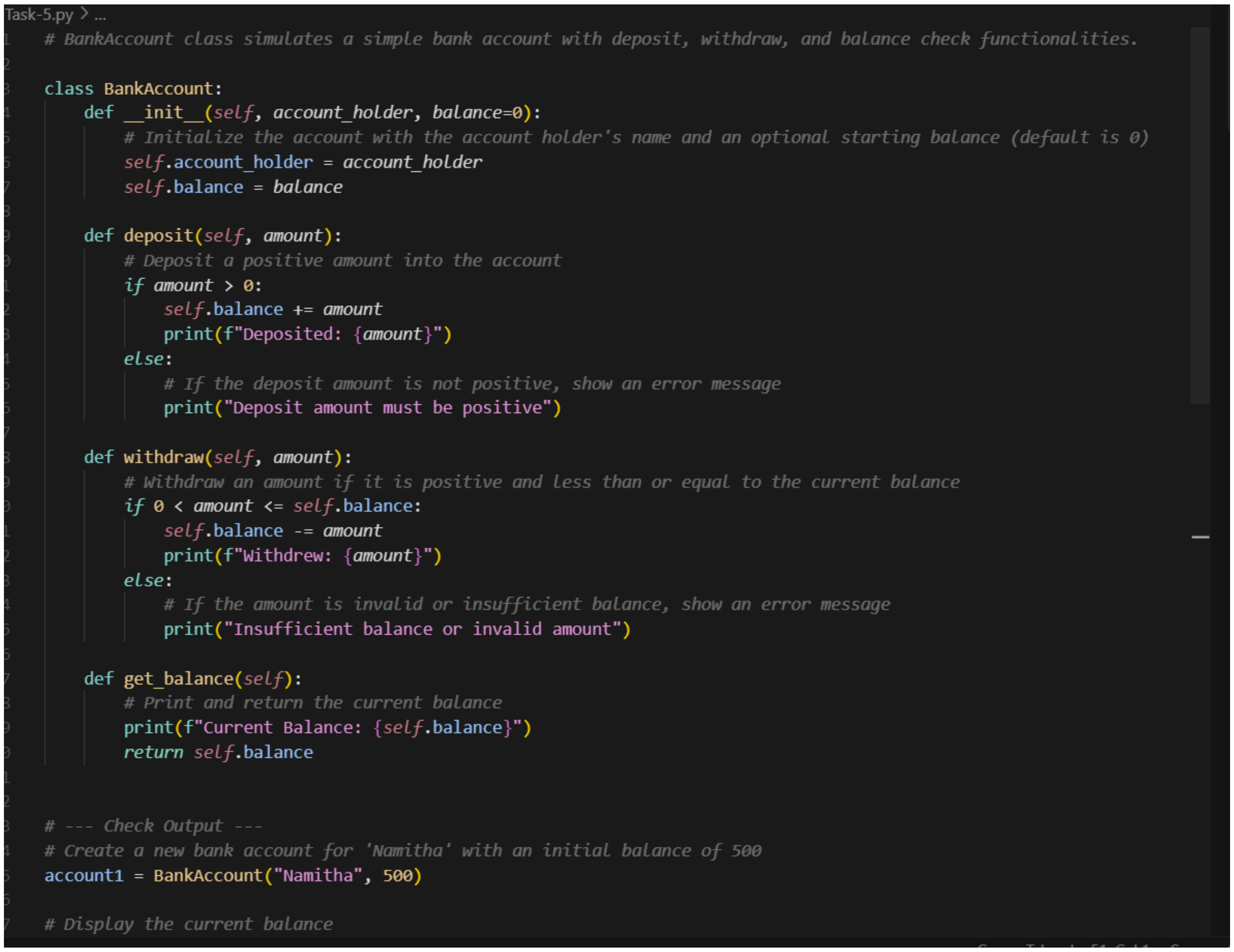
TASK-5:

PROMPT:

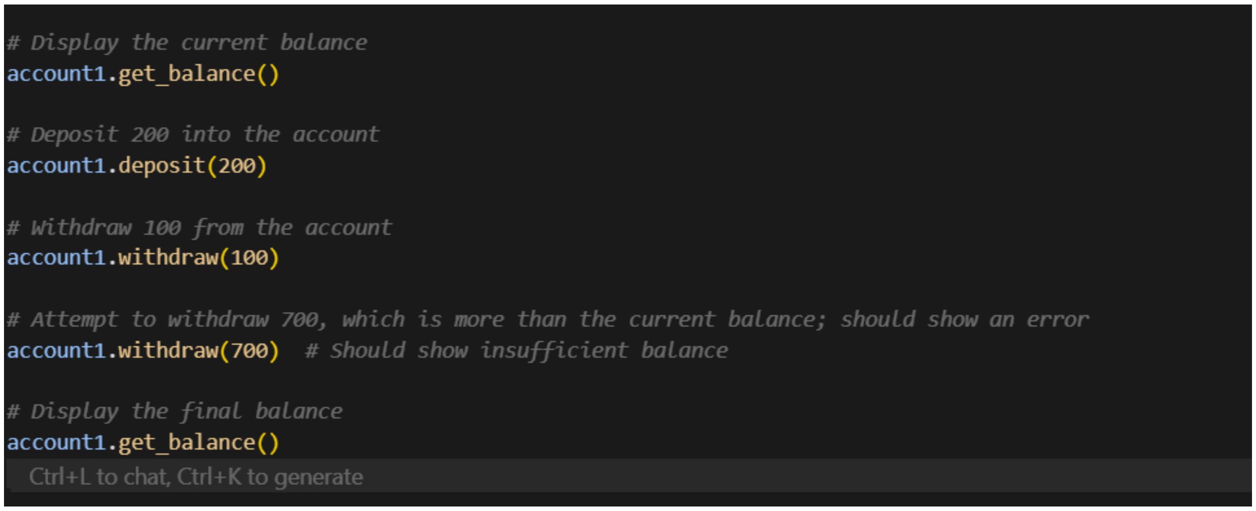
```
class BankAccount:
def __init__(self, account_holder, balance=0):
self.account_holder = account_holder
self.balance = balance

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

def withdraw(self, amount):
if 0 < amount <= self.balance:
self.balance -= amount
print(f"Withdrew: {amount}")
else:
print("Insufficient balance or invalid amount")

def get_balance(self):
print(f"Current Balance: {self.balance}")
return self.balance


# --- Check Output ---
account1 = BankAccount("Namitha", 500)

account1.get_balance()
account1.deposit(200)
account1.withdraw(100)
account1.withdraw(700) # Should show insufficient balance
account1.get_balance()
```

Expalin this code by adding comments.

Code:

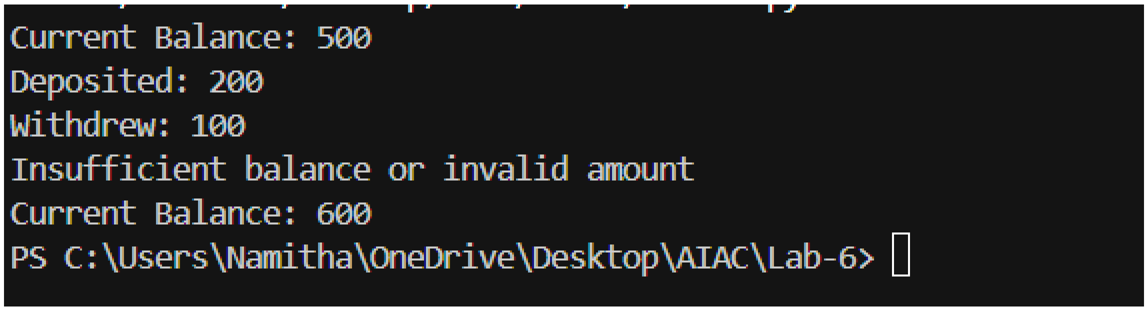


```python
# BankAccount class simulates a simple bank account with deposit, withdraw, and balance check functionalities.

class BankAccount:
    def __init__(self, account_holder, balance=0):
        # Initialize the account with the account holder's name and an optional starting balance (default is 0)
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        # Deposit a positive amount into the account
        if amount > 0:
            self.balance += amount
            print(f"Deposited: {amount}")
        else:
            # If the deposit amount is not positive, show an error message
            print("Deposit amount must be positive")

    def withdraw(self, amount):
        # Withdraw an amount if it is positive and less than or equal to the current balance
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: {amount}")
        else:
            # If the amount is invalid or insufficient balance, show an error message
            print("Insufficient balance or invalid amount")

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


# --- Check Output ---
# Create a new bank account for 'Namitha' with an initial balance of 500
account1 = BankAccount("Namitha", 500)

# Display the current balance
```



```python
# Display the current balance
account1.get_balance()

# Deposit 200 into the account
account1.deposit(200)

# Withdraw 100 from the account
account1.withdraw(100)

# Attempt to withdraw 700, which is more than the current balance; should show an error
account1.withdraw(700)  # Should show insufficient balance

# Display the final balance
account1.get_balance()
```

Output:



```
Current Balance: 500
Deposited: 200
Withdrew: 100
Insufficient balance or invalid amount
Current Balance: 600
PS C:\Users\Namitha\OneDrive\Desktop\AIAC\Lab-6>
```