LAB ASSIGNMENT-6.3

2303A51337

BATCH-10

➢ TASK-1:

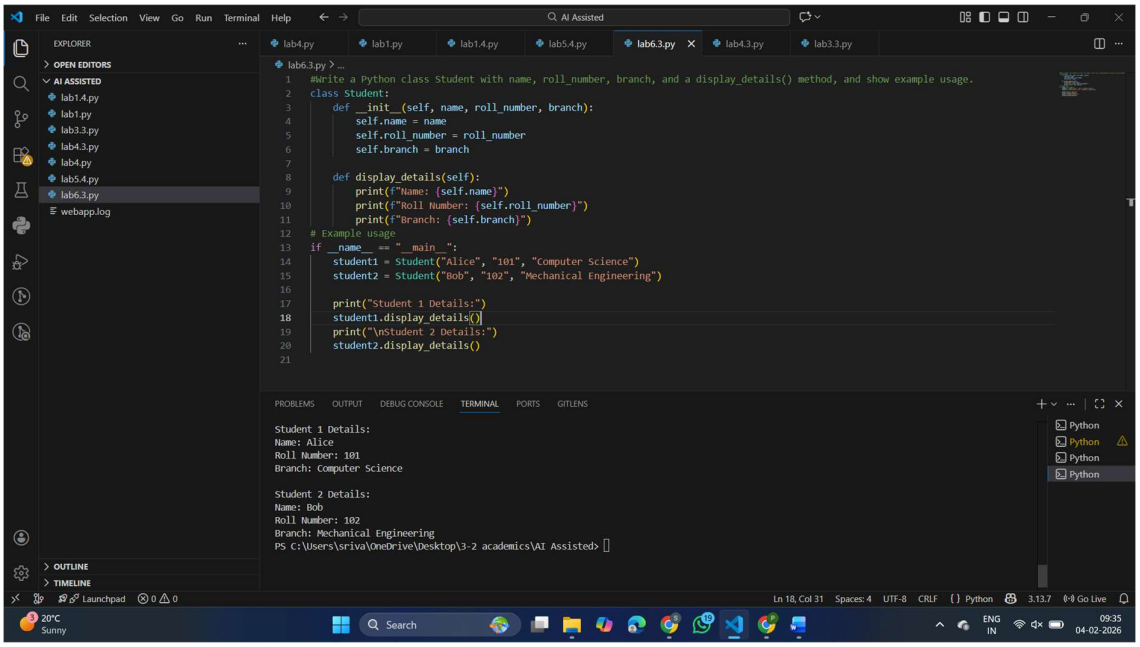
PROMPT:

Write a Python class Student with name, roll_number, branch, and a display_details() method, and show example usage.

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}")
# Example usage
if __name__ == "__main__":
    student1 = Student("Alice", "101", "Computer Science")
    student2 = Student("Bob", "102", "Mechanical Engineering")
    print("Student 1 Details:")
    student1.display_details()
    print("\nStudent 2 Details:")
    student2.display_details()
```

OUTPUT:



EXPLANATION:

- When the program runs, it creates two Student objects named student1 and student2 with their respective names, roll numbers, and branches.
- The display_details() method is called for each object, which prints the stored information in a formatted way. First, the details of the first student are displayed, followed by the details of the second student .

❖ TASK-2

PROMPT:

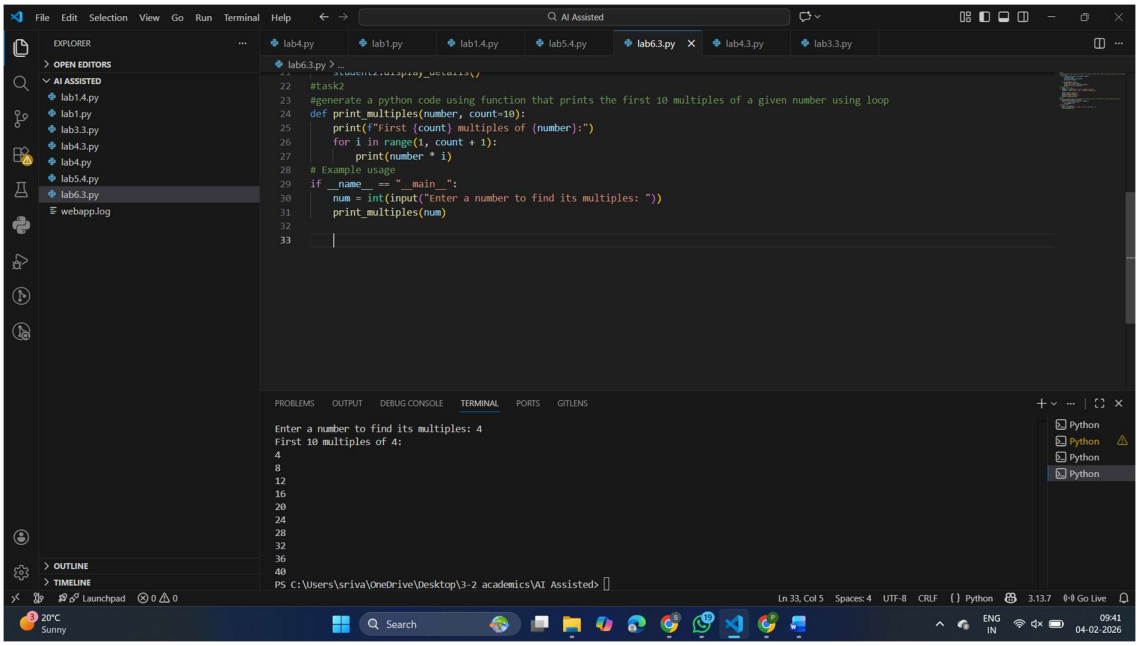
Generate a python code using function that prints the first 10 multiples of a given number using loop.

CODE:

```
def print_multiples(number, count=10):
    print(f"First {count} multiples of {number}:")
    for i in range(1, count + 1):
        print(number * i)
# Example usage
if __name__ == "__main__":
```

```
num = int(input("Enter a number to find its multiples: "))
print_multiples(num)
```

OUTPUT:



EXPLANATION:

- This program prints the first 10 multiples of a given number using a function. It takes a number from the user and uses a loop to multiply it from 1 to 10. The program shows the use of functions, loops, and user input to generate multiples of a number.

❖ TASK-3:

PROMPT:

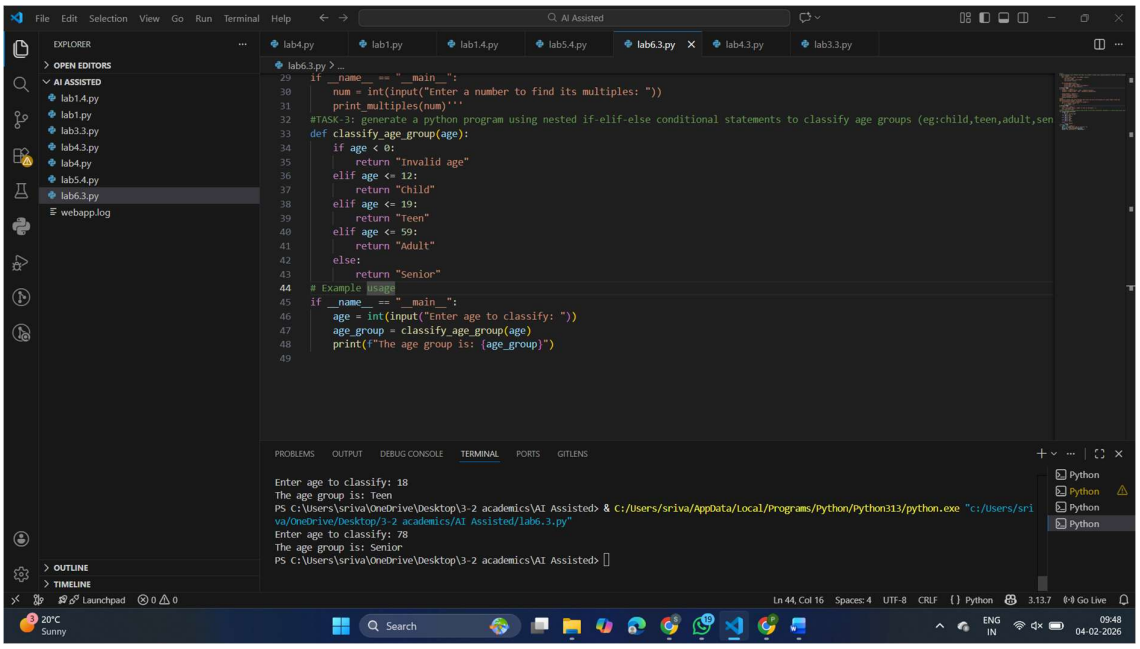
Generate a python program using nested if-elif-else conditional statements to classify age groups (eg:child,teen,adult,senior)

CODE:

```
def classify_age_group(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
```

```
    elif age <= 19:
        return "Teen"
    elif age <= 59:
        return "Adult"
    else:
        return "Senior"
# Example usage
if __name__ == "__main__":
    age = int(input("Enter age to classify: "))
    age_group = classify_age_group(age)
    print(f"The age group is: {age_group}")
```

OUTPUT:



EXPLANATION:

- This program classifies a person into an age group based on the given age. It takes the age as input from the user and passes it to the classify_age_group() function.

- The function checks different age ranges and returns the appropriate category such as Child, Teen, Adult, or Senior. If the age is negative, it returns Invalid age. Finally, the result is printed on the screen.

❖ TASK-4:

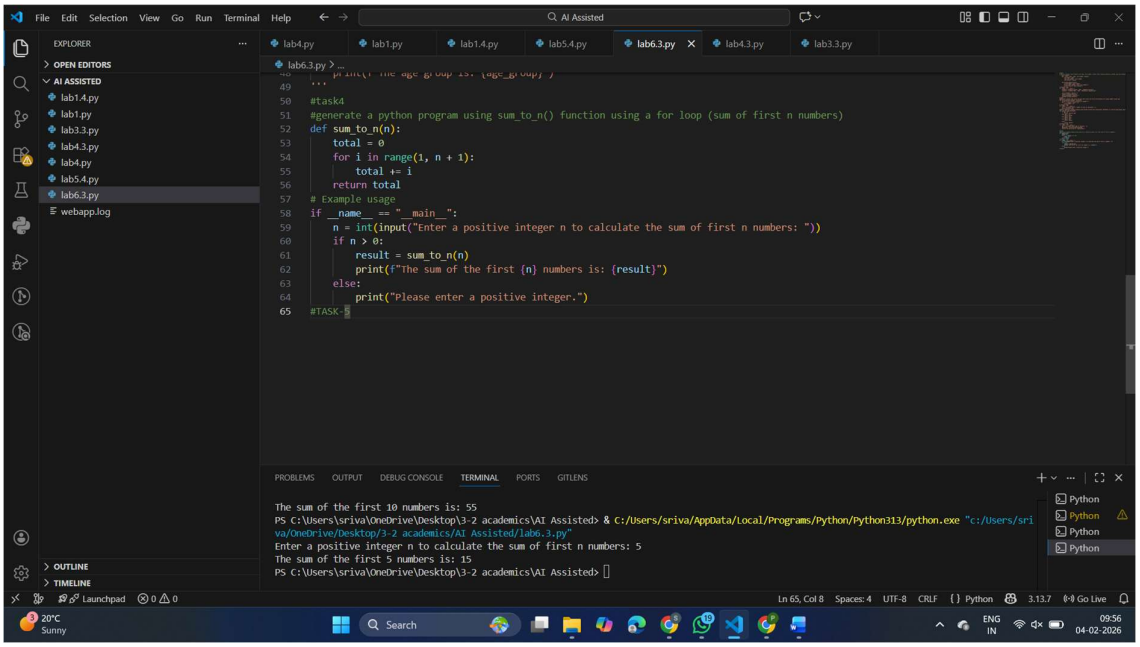
PROMPT:

Generate a python program using sum_to_n() function using a for loop (sum of first n numbers).

CODE:

```
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
# Example usage
if __name__ == "__main__":
    n = int(input("Enter a positive integer n to calculate the sum of first n numbers: "))
    if n > 0:
        result = sum_to_n(n)
        print(f"The sum of the first {n} numbers is: {result}")
    else:
        print("Please enter a positive integer.")
```

OUTPUT:



EXPLANATION:

- This program calculates the sum of the first n natural numbers using a function.
- The user enters a positive number n, which is passed to the function sum_to_n(). Inside the function, a loop adds numbers from 1 to n and stores the result in total. The function then returns this sum.
- If the user enters a value less than or equal to zero, the program displays a message asking for a positive integer.

❖ TASK-5:

PROMPT:

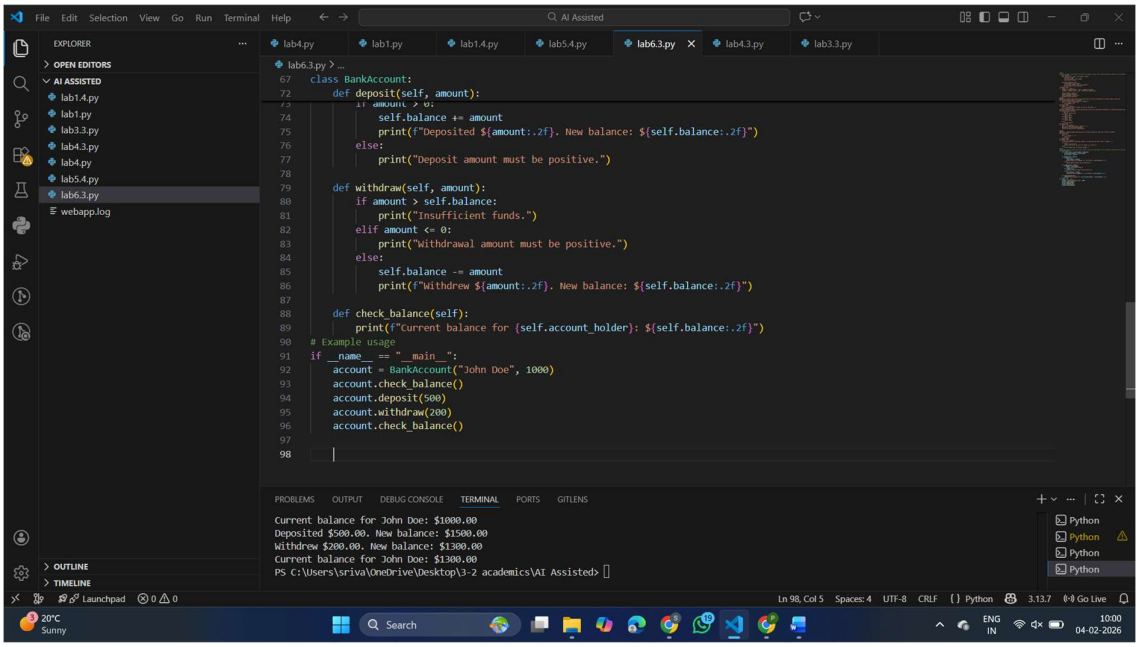
Generate a python program about bank account using class with methods such as deposit(),withdraw(),and check_balance().

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:.2f}. New balance: ${self.balance:.2f}")
        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:.2f}. New balance: ${self.balance:.2f}")

    def check_balance(self):
        print(f"Current balance for {self.account_holder}: ${self.balance:.2f}")
# Example usage
if __name__ == "__main__":
    account = BankAccount("John Doe", 1000)
    account.check_balance()
    account.deposit(500)
    account.withdraw(200)
    account.check_balance()
```

OUTPUT:



EXPLANATION:

- This program simulates a simple bank account system using a class. The BankAccount class stores the account holder's name and balance.
- It provides methods to deposit money, withdraw money, and check the current balance.
- The deposit method adds money only if the amount is positive, while the withdraw method ensures there are enough funds and the amount is valid.