

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")
```

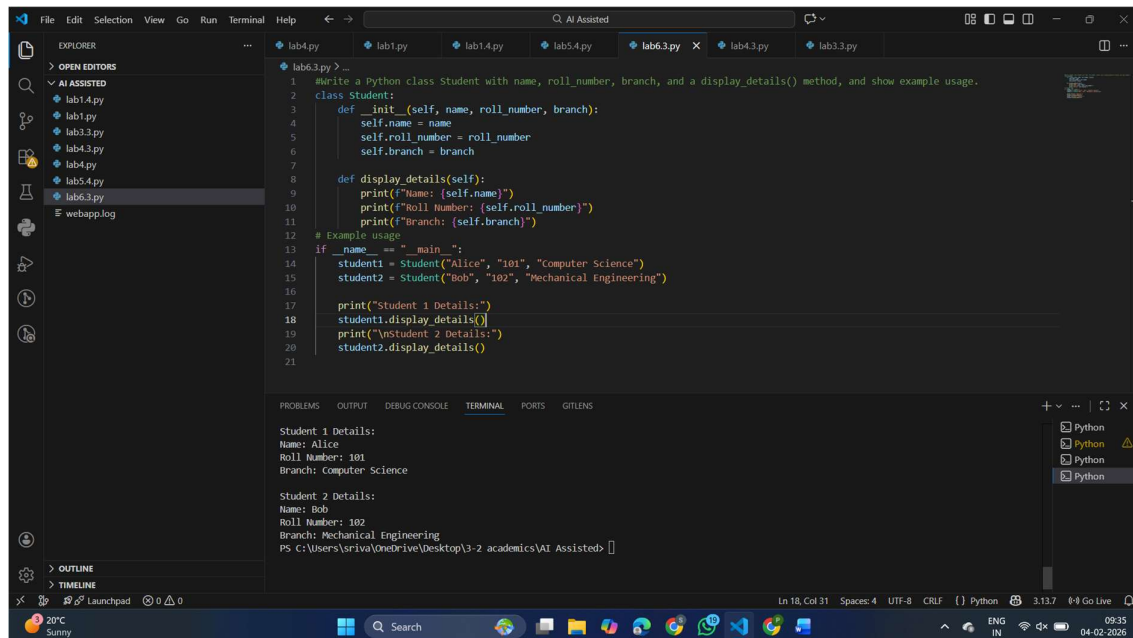
```
    print("Student 1 Details:")
```

```
    student1.display_details()
```

```
    print("\nStudent 2 Details:")
```

```
    student2.display_details()
```

OUTPUT:



```
1 #Write a Python class Student with name, roll_number, branch, and a display_details() method, and show example usage.
2 class Student:
3     def __init__(self, name, roll_number, branch):
4         self.name = name
5         self.roll_number = roll_number
6         self.branch = branch
7
8     def display_details(self):
9         print(f"Name: {self.name}")
10        print(f"Roll Number: {self.roll_number}")
11        print(f"Branch: {self.branch}")
12
13 # Example usage
14 if __name__ == "__main__":
15     student1 = Student("Alice", "101", "Computer Science")
16     student2 = Student("Bob", "102", "Mechanical Engineering")
17
18     print("Student 1 Details:")
19     student1.display_details()
20     print("\nStudent 2 Details:")
21     student2.display_details()
```

Student 1 Details:
Name: Alice
Roll Number: 101
Branch: Computer Science

Student 2 Details:
Name: Bob
Roll Number: 102
Branch: Mechanical Engineering

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.

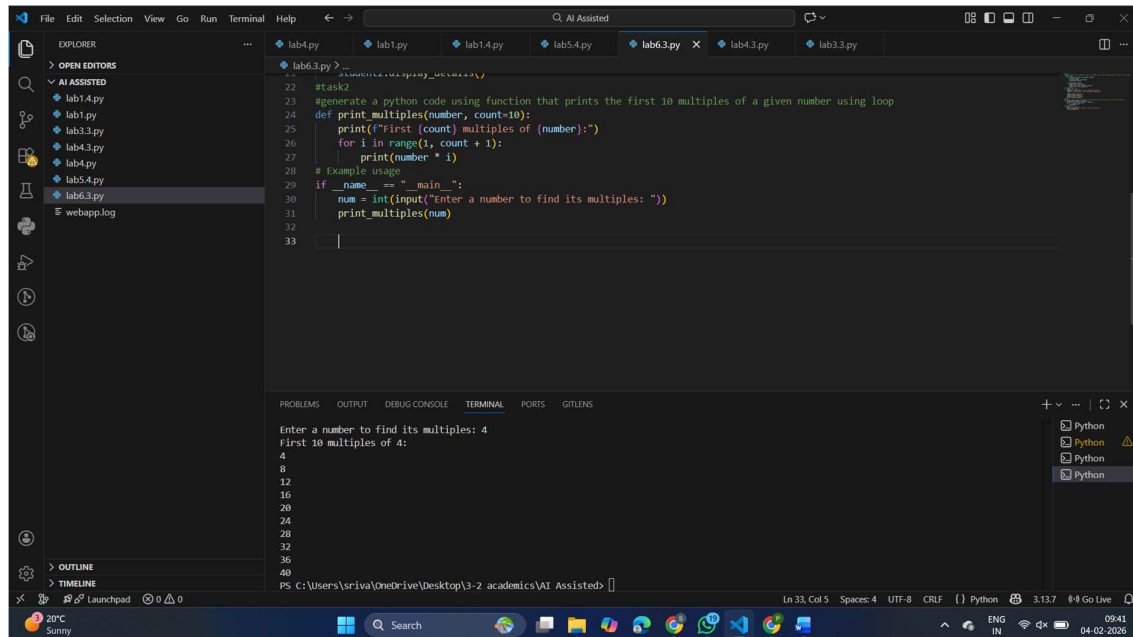
CODE:

```
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:

A screenshot of the Visual Studio Code editor interface. The Explorer panel on the left shows a project with several Python files (lab1.py to lab6.py) and a webapp.log file. The main editor window displays a Python script in 'lab6.3.py'. The script defines a function 'print_multiples' that takes a number and a count (default 10) and prints the first 'count' multiples of the number using a loop. The main block prompts the user to enter a number and calls the function. The TERMINAL panel at the bottom shows the execution output: 'Enter a number to find its multiples: 4' followed by 'First 10 multiples of 4:' and a list of multiples from 4 to 40. The status bar at the bottom indicates the file is at line 33, column 5, using UTF-8 encoding and CRLF line endings, with Python 3.13.7 installed.

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__":

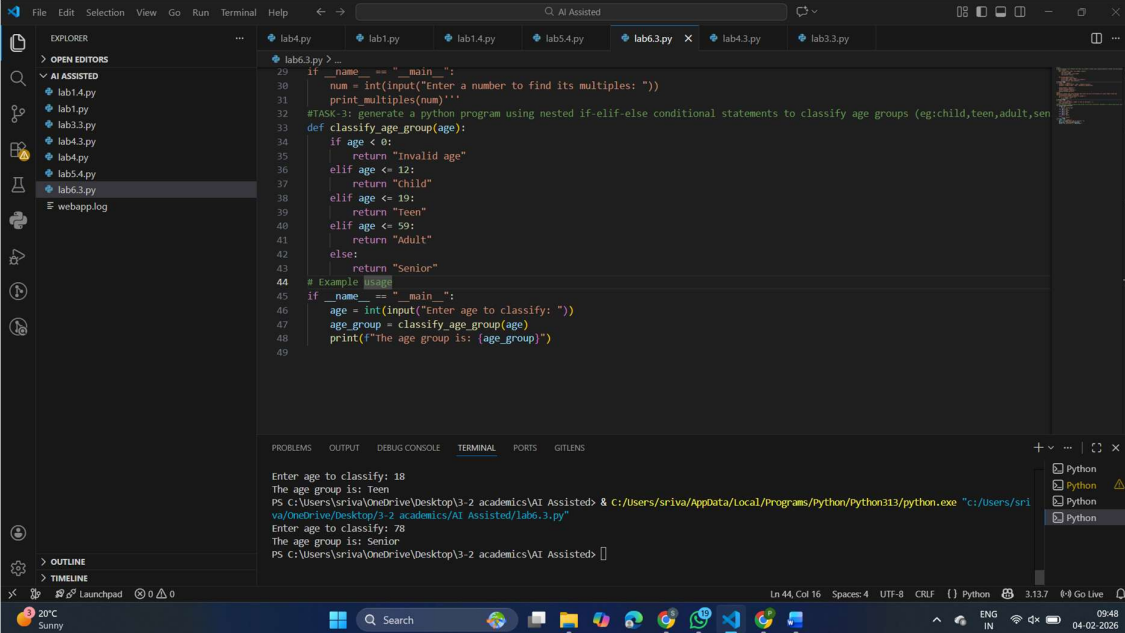
    age = int(input("Enter age to classify: "))

    age_group = classify_age_group(age)

    print(f"The age group is: {age_group}")

```

OUTPUT:



The screenshot shows a Visual Studio Code editor with a Python file named `lab6.3.py` open. The code defines a function `classify_age_group` that takes an age as input and returns a string representing the age group based on the following conditions:

- If age is less than 0, return "Invalid age".
- If age is less than or equal to 12, return "child".
- If age is less than or equal to 19, return "Teen".
- If age is less than or equal to 59, return "Adult".
- Otherwise, return "Senior".

The code also includes an example usage section that prompts the user to enter an age and prints the resulting age group.

The terminal output shows the program being executed. It prompts the user to enter an age, and two examples are shown: entering 18 results in "Teen", and entering 78 results in "Senior".

```

Enter age to classify: 18
The age group is: Teen
PS C:\Users\sriya\OneDrive\Desktop\3-2 academics\AI Assisted> & C:\Users\sriya\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/sriya/OneDrive/Desktop/3-2 academics/AI Assisted/lab6.3.py"
Enter age to classify: 78
The age group is: Senior
PS C:\Users\sriya\OneDrive\Desktop\3-2 academics\AI Assisted>

```

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:**

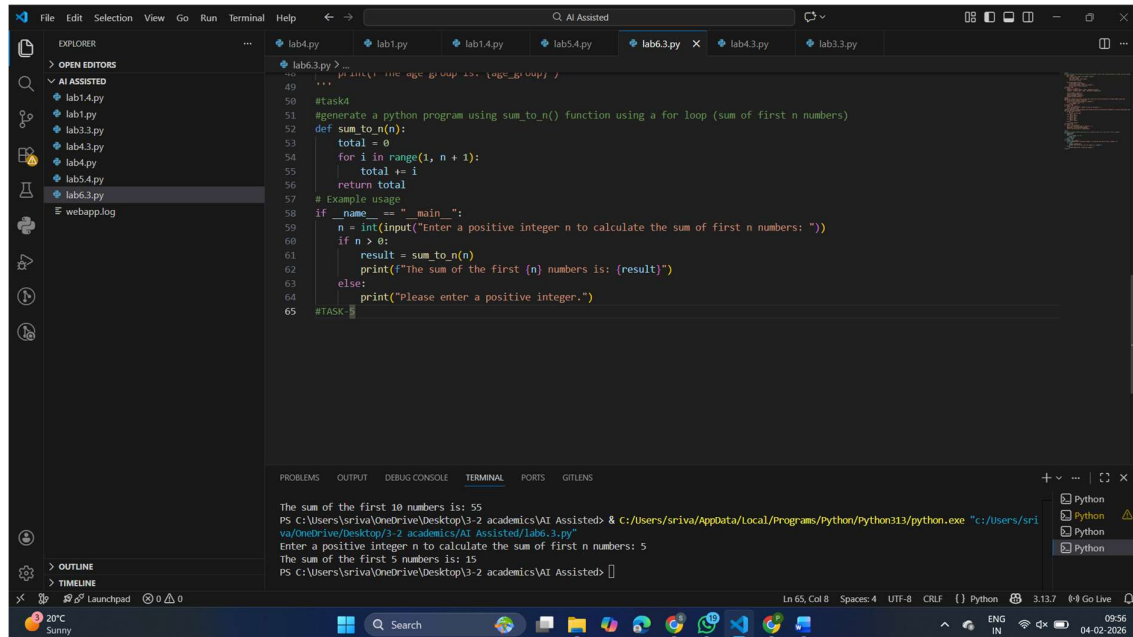
PROMPT:

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:



The screenshot shows a VS Code editor with a Python file named lab6.3.py. The code defines a function `sum_to_n(n)` that calculates the sum of the first `n` natural numbers using a for loop. The main part of the program prompts the user to enter a positive integer `n`. If `n` is greater than 0, it calls `sum_to_n(n)` and prints the result. If `n` is less than or equal to 0, it prints a message asking for a positive integer. The terminal output shows the program running successfully, with the user entering 5 and the program outputting "The sum of the first 5 numbers is: 15".

```
49 #task4
50 #generate a python program using sum_to_n() function using a for loop (sum of first n numbers)
51 def sum_to_n(n):
52     total = 0
53     for i in range(1, n + 1):
54         total += i
55     return total
56 # Example usage
57 if __name__ == "__main__":
58     n = int(input("Enter a positive integer n to calculate the sum of first n numbers: "))
59     if n > 0:
60         result = sum_to_n(n)
61         print(f"The sum of the first {n} numbers is: {result}")
62     else:
63         print("Please enter a positive integer.")
64 #TASK-5
65
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS

The sum of the first 10 numbers is: 55
PS C:\Users\sriva\OneDrive\Desktop\3-2 academics\AI Assisted> & C:/Users/sriva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/sriva/OneDrive/Desktop/3-2 academics/AI Assisted/lab6.3.py"
Enter a positive integer n to calculate the sum of first n numbers: 5
The sum of the first 5 numbers is: 15
PS C:\Users\sriva\OneDrive\Desktop\3-2 academics\AI Assisted>

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
```

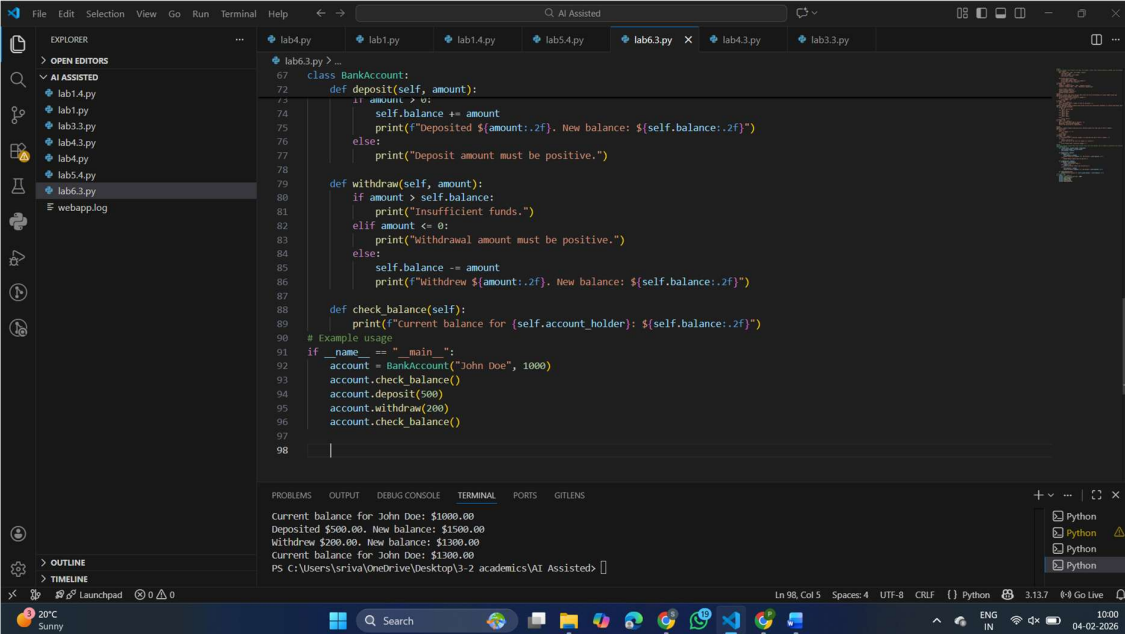
```
def deposit(self, amount):
    if amount > 0:
        self.balance += amount
        print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
        print("Deposit amount must be positive.")

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:



The screenshot shows a Visual Studio Code editor with a Python file named `lab6.3.py` open. The code defines a `BankAccount` class with methods for depositing, withdrawing, and checking the balance. The `__main__` block demonstrates the usage of the class.

```
67 class BankAccount:
68     def deposit(self, amount):
69         if amount > 0:
70             self.balance += amount
71             print(f'Deposited ${amount:.2f}. New balance: ${self.balance:.2f}')
72         else:
73             print("Deposit amount must be positive.")
74
75     def withdraw(self, amount):
76         if amount > self.balance:
77             print("Insufficient funds.")
78         elif amount <= 0:
79             print("Withdrawal amount must be positive.")
80         else:
81             self.balance -= amount
82             print(f'Withdrew ${amount:.2f}. New balance: ${self.balance:.2f}')
83
84     def check_balance(self):
85         print(f'Current balance for {self.account_holder}: ${self.balance:.2f}')
86
87 # Example usage
88 if __name__ == "__main__":
89     account = BankAccount("John Doe", 1000)
90     account.check_balance()
91     account.deposit(500)
92     account.withdraw(200)
93     account.check_balance()
94
95
96
97
98
```

The terminal output shows the execution of the program:

```
Current balance for John Doe: $1000.00
Deposited $500.00. New balance: $1500.00
Withdrew $200.00. New balance: $1300.00
Current balance for John Doe: $1300.00
PS C:\Users\sriya\OneDrive\Desktop\3-2 academics\AI Assisted>
```

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.