

# 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():
    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:

The screenshot shows a Microsoft Windows desktop environment with the Visual Studio Code (VS Code) application open. The code editor displays a Python script named `lab6.3.py`. The script defines a `Student` class with an `__init__` method to initialize name, roll number, and branch. It also includes a `display_details` method to print these details. Two student objects, `student1` and `student2`, are created and their details are printed. The terminal below the editor shows the execution of the script and the resulting output: "Student 1 Details" followed by Alice's information, and "Student 2 Details" followed by Bob's information. The status bar at the bottom right indicates the file is saved in Python, the current line is 18, column 31, and the date is 04-02-2026.

```
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("Name: " + self.name)
10        print("Roll Number: " + str(self.roll_number))
11        print("Branch: " + self.branch)
12
13 if __name__ == "__main__":
14     student1 = Student("Alice", 101, "Computer Science")
15     student2 = Student("Bob", 102, "Mechanical Engineering")
16
17     print("Student 1 Details:")
18     student1.display_details()
19     print("\nStudent 2 Details:")
20     student2.display_details()
```

## 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:

The screenshot shows the Visual Studio Code interface with the 'AI Assisted' extension open. The terminal window displays the following interaction:

```

Enter a number to find its multiples: 4
First 10 multiples of 4:
4
8
12
16
20
24
28
32
36
40

```

#### 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 the Visual Studio Code interface. The code editor displays a Python script named lab63.py. The script defines a function `classify_age_group` that takes an age as input and returns a group name based on the age range. It includes an example usage section at the bottom. The terminal below the editor shows the execution of the script and its output, which correctly classifies the input age as 'Senior'.

```

    if __name__ == "__main__":
        age = int(input("Enter age to classify: "))
        age_group = classify_age_group(age)
        print(f"The age group is: {age_group}")

```

```

Enter age to classify: 18
The age group is: Senior

```

## 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 the Visual Studio Code interface with the 'AI Assisted' extension open. In the top bar, there is a search bar with the placeholder 'Q, AI Assisted'. Below it, the 'OPEN EDITORS' section lists several files: lab4.py, lab1.py, lab1.4.py, lab1.4.py, lab6.3.py, lab4.3.py, and lab3.3.py. The main editor area displays a Python script named 'lab6.3.py' with the following code:

```
task4
#generate a python program using sum_to_n() function using a for loop (sum of first n numbers)
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.")

#TASK-5
```

The bottom status bar shows the file is in line 65, column 8, with 0 spaces, 4 tabs, and UTF-8 encoding. It also shows the Python language selected and the date/time as 04-02-2026.

## 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
```

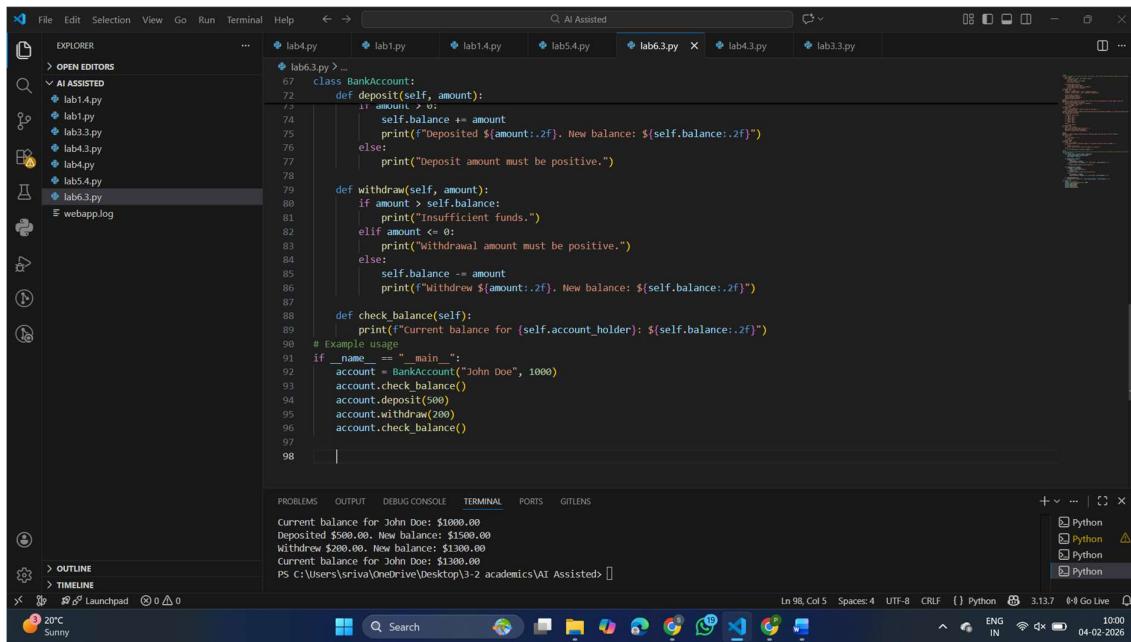
```
def deposit(self, amount):
    if amount > 0:
        self.balance += amount
        print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
    else:
        print("Deposit amount must be positive.")

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:



```
File Edit Selection View Go Run Terminal Help < > AI Assisted ... lab4.py lab1.py lab1.4.py lab5.4.py lab6.3.py lab4.3.py lab3.4.py lab5.3.py webapp.log

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
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\sriva\OneDrive\Desktop\3-2 academics\AI Assisted\b

Ln 98, Col 5 Spaces: 4 UTF-8 CRLF {} Python 3.13.7 Go Live
ENGLISH IN 1000 04-02-2026
```

## 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.