

## ASSIGNMENT – 6.3

2303A51525

Batch-10

Task-1

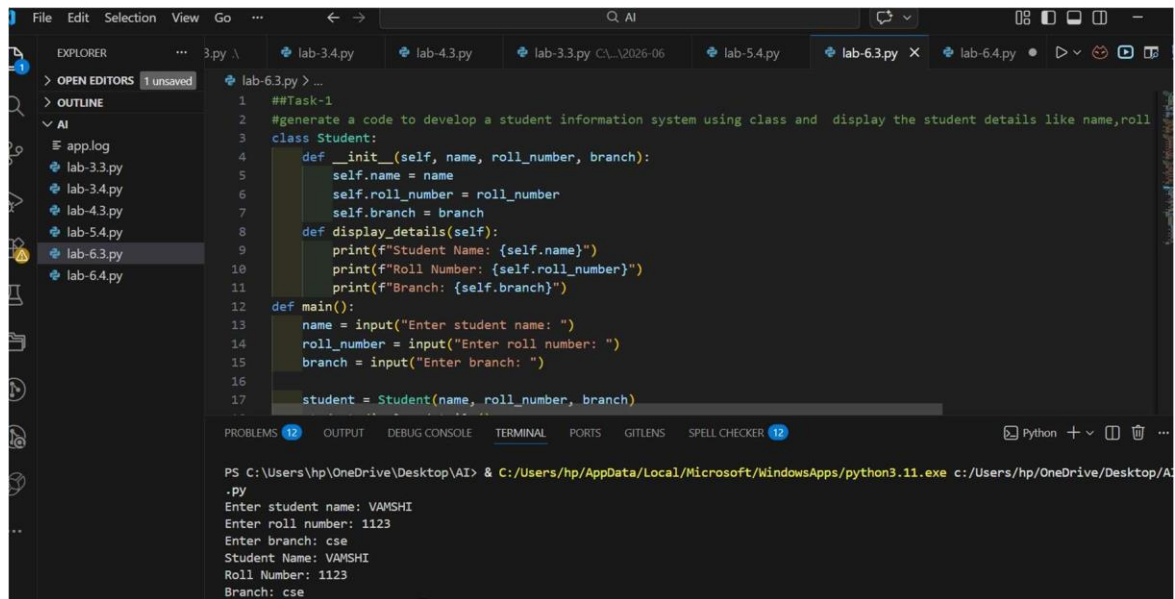
Prompt: generate a code to develop a student information system using class and display the student details like name, roll number and branch with user input.

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"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")
def main():
    name = input("Enter student name: ")
    roll_number = input("Enter roll number: ")
    branch = input("Enter branch: ")

    student = Student(name, roll_number, branch)
    student.display_details()
if __name__ == "__main__":
    main()
```

Output :



```
1 ##Task-1
2 #generate a code to develop a student information system using class and display the student details like name,roll
3 class Student:
4     def __init__(self, name, roll_number, branch):
5         self.name = name
6         self.roll_number = roll_number
7         self.branch = branch
8     def display_details(self):
9         print(f"Student Name: {self.name}")
10        print(f"Roll Number: {self.roll_number}")
11        print(f"Branch: {self.branch}")
12
13 def main():
14     name = input("Enter student name: ")
15     roll_number = input("Enter roll number: ")
16     branch = input("Enter branch: ")
17
18     student = Student(name, roll_number, branch)
19
20     student.display_details()
```

PS C:\Users\hp\OneDrive\Desktop\AI> C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-6.3.py

Enter student name: VAMSHI  
Enter roll number: 1123  
Enter branch: cse  
Student Name: VAMSHI  
Roll Number: 1123  
Branch: cse

### Code Analysis :

- ☐ Uses class and constructor (`__init__`) to store student details, showing OOP concept.
- ☐ Accepts user input dynamically, making the program interactive.
- ☐ `display_details()` method separates logic and printing, improving readability.
- ☐ Simple structure suitable for beginners to understand classes.
- ☐ Can be improved by storing multiple students using lists.

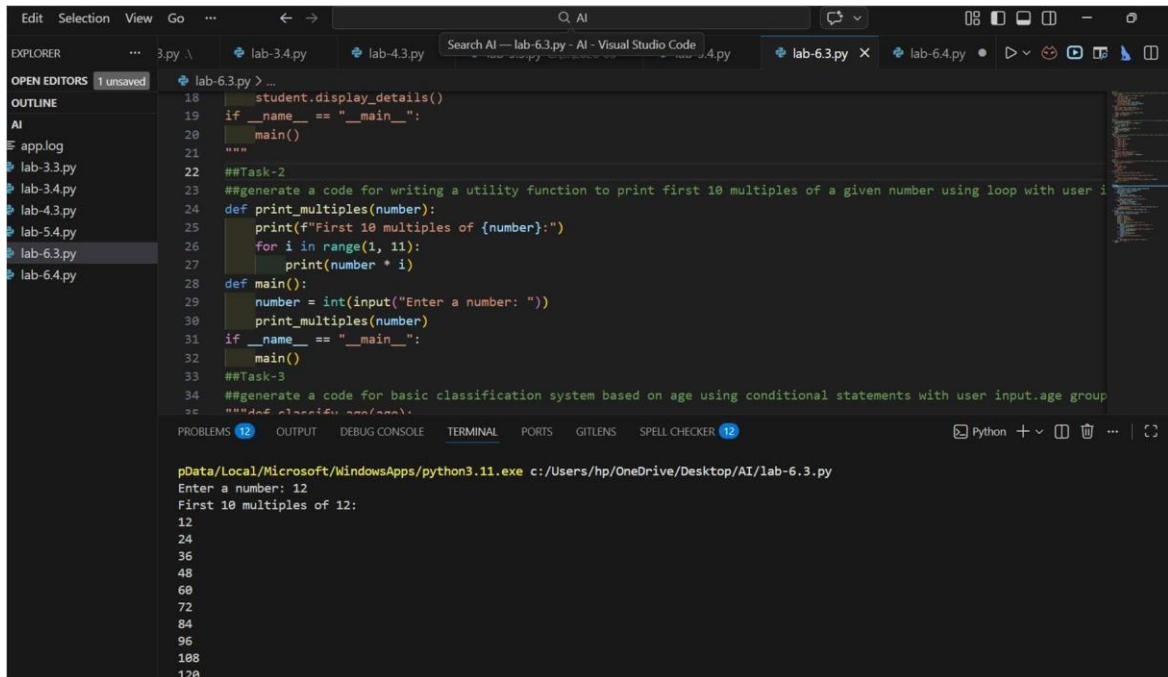
### Task-2

Prompt: generate a code for writing a utility function to print first 10 multiples of a given number using loop with user input.without using try and except block.

Code :

```
def print_multiples(number):
    print(f'First 10 multiples of {number}:')
    for i in range(1, 11):
        print(number * i)
def main():
    number = int(input("Enter a number: "))
    print_multiples(number) if __name__ ==
    "__main__":
        main()
```

Output :



```
18     student.display_details()
19 if __name__ == "__main__":
20     main()
21
22 ##Task-2
23 ##generate a code for writing a utility function to print first 10 multiples of a given number using loop with user input
24 def print_multiples(number):
25     print(f"First 10 multiples of {number}:")
26     for i in range(1, 11):
27         print(number * i)
28
29 def main():
30     number = int(input("Enter a number: "))
31     print_multiples(number)
32
33 if __name__ == "__main__":
34     main()
35
36 ##Task-3
37 ##generate a code for basic classification system based on age using conditional statements with user input.age group
38 def classify_age(age):
39     if age < 0:
40         return "Invalid age"
41     elif age <= 12:
42         return "Child"
43     elif age <= 19:
44         return "Teenager"
45     elif age <= 29:
46         return "Young Adult"
47     elif age <= 39:
48         return "Adult"
49     elif age <= 59:
50         return "Middle-aged"
51     elif age <= 69:
52         return "Senior"
53     else:
54         return "Invalid age"
```

Terminal Output:

```
pData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-6.3.py
Enter a number: 12
First 10 multiples of 12:
12
24
36
48
60
72
84
96
108
120
```

Code Analysis :

- Uses a for loop with range(1,11) to generate first 10 multiples efficiently.
- Function-based design makes the code reusable.
- Takes integer input directly without exception handling as required.
- Clear output formatting helps users understand results easily.
- Could add input validation to avoid wrong data types.

### Task-3

Prompt: generate a code for basic classification system based on age using conditional statements with user input.age groups like(child,teenager.adult and senior)without using try and except block.

Code :

```
def classify_age(age):
    if age < 0:
        return "Invalid
    age"
    elif age <= 12:
        return "Child"
    elif
    age <= 19:
```

```

        return "Teenager"
elif age <= 59:
    return "Adult"
else:
    return "Senior"

def main():
    age = int(input("Enter your age: "))
    category = classify_age(age)
    print(f"You
are classified as: {category}")
if __name__
== "__main__":
    main()

```

Output :

```

31 if __name__ == "__main__":
32     main()
33
34 ##Task-3
35 ##generate a code for basic classification system based on age using conditional statements with user input.age group
36 def classify_age(age):
37     if age < 0:
38         return "Invalid age"
39     elif age <= 12:
40         return "Child"
41     elif age <= 19:
42         return "Teenager"
43     elif age <= 59:
44         return "Adult"
45     else:
46         return "Senior"
47
48 def main():
49     age = int(input("Enter your age: "))
50     category = classify_age(age)
51     print(f"You are classified as: {category}")
52
53 if __name__ == "__main__":
54     main()

```

Terminal Output:

```

PS C:\Users\hp\OneDrive\Desktop\AI> C:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/1
.py
Enter your age: 21
You are classified as: Adult

```

Code Analysis :

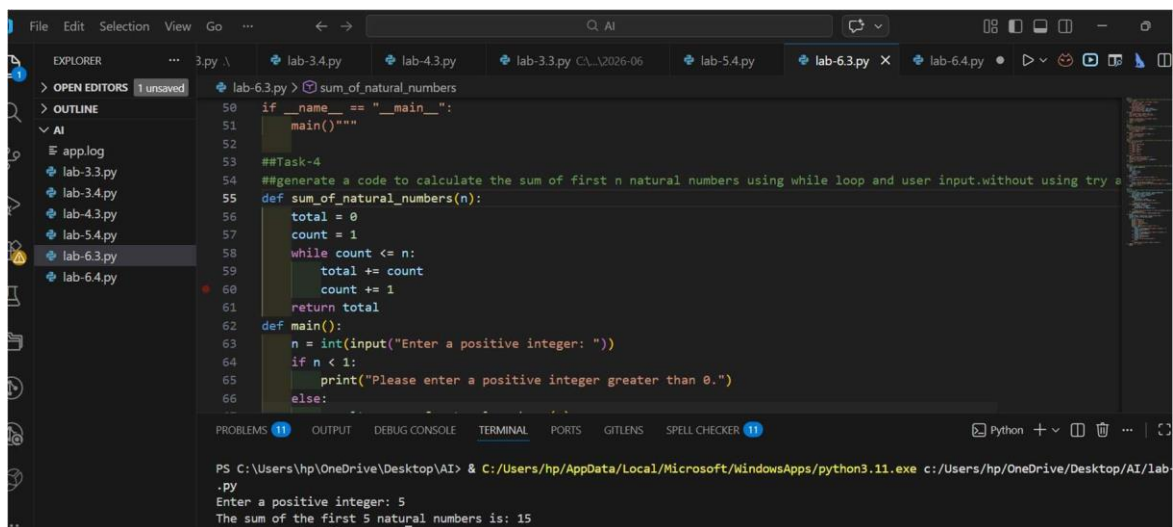
- ☐ Implements conditional statements (if-elif-else) for decision making.
- ☐ Categories like Child, Teenager, Adult, and Senior are clearly defined.
- ☐ Function returns values instead of printing directly, improving modularity.
- ☐ Handles negative age with “Invalid age” condition.
- ☐ Logic is simple but effective for basic classification problems.

Task-4

Prompt: generate a code to calculate the sum of first n natural numbers using while loop and user input without using try and except block.

Code :

```
def sum_of_natural_numbers(n):
    total = 0
    count = 1
    while count <= n:
        total += count
        count += 1
    return total
def main():
    n = int(input("Enter a positive integer: "))
    if n < 1:
        print("Please enter a positive integer greater than 0.")
    else:
        result = sum_of_natural_numbers(n)
        print(f"The sum of the first {n} natural numbers is: {result}")
if __name__ == "__main__":
    main()
Output
:
```



```
File Edit Selection View Go ... Q AI
EXPLORER 3.py \ lab-34.py lab-43.py lab-33.py C:\_2026-06 lab-54.py lab-63.py X lab-64.py
> OPEN EDITORS 1 unsaved
> OUTLINE
  AI
  app.log
  lab-33.py
  lab-34.py
  lab-43.py
  lab-54.py
  lab-63.py
  lab-64.py
lab-63.py > sum_of_natural_numbers
50 if __name__ == "__main__":
51     main()
52
53 ##Task-4
54 ##generate a code to calculate the sum of first n natural numbers using while loop and user input.without using try a
55 def sum_of_natural_numbers(n):
56     total = 0
57     count = 1
58     while count <= n:
59         total += count
60         count += 1
61     return total
62 def main():
63     n = int(input("Enter a positive integer: "))
64     if n < 1:
65         print("Please enter a positive integer greater than 0.")
66     else:
67
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER 11
Python + v [ ] ... [ ]
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab
.py
Enter a positive integer: 5
The sum of the first 5 natural numbers is: 15
```

Code Analysis :

- ☐ Uses while loop to iteratively calculate the sum step by step.
- ☐ Maintains separate variables (total, count) for clarity.
- ☐ Checks for positive input before calculation, improving correctness.
- ☐ Function returns result, making it reusable in other programs.
- ☐ Could be optimized using formula  $n*(n+1)/2$  for efficiency.

## Task 5

Prompt: #generate a code to design a bank application using class with methods like deposit,withdraw and check balance take user input. class BankAccount:

```
    def __init__(self,
account_holder,
initial_balance=0):
self.account_holder =
account_holder
        self.balance =
initial_balance
        def deposit(self,
amount):
            if amount > 0:
self.balance +=
amount
                print(f"Deposited:
${amount:.2f}")
            else:
print("Deposit amount
must be positive.")    def
withdraw(self, amount):
if 0 < amount <=
self.balance:
self.balance -=
amount
                print(f"Withdrew:
${amount:.2f}")
            else:
print("Insufficient balance
```

```

or invalid withdrawal
amount.")
def
check_balance(self):
print(f"Current balance:
${self.balance:.2f}")
def
main():
    account_holder =
input("Enter account
holder name: ")
    account =
BankAccount(account_holder)
    while True:
print("\nOptions:")
print("1. Deposit")
print("2. Withdraw")
print("3. Check
Balance")
        print("4. Exit")
choice = input("Choose an
option (1-4): ")
        if
choice == '1':
amount =
float(input("Enter amount
to deposit: "))
account.deposit(amount)
elif choice == '2':
amount =
float(input("Enter amount
to withdraw: "))

```

```

account.withdraw(a
mount)

    elif choice == '3':
account.check_bala nce()
elif choice == '4':
print("Exiting the
application.")

    break

else:

    print("Invalid
choice. Please try again.")

if __name__ ==
"__main__":

    main()

```

Output :

The screenshot shows a VS Code editor with a Python file named 'lab-6.3.py'. The code implements a menu-driven system for a bank account. It includes methods for depositing, withdrawing, and checking the balance. The terminal output shows the program running, prompting for the account holder's name ('vamshii'), displaying the menu options, and successfully depositing \$100 when option 1 is selected.

```

92 def main():
93     while True:
94         print("\nOptions:")
95         print("1. Deposit")
96         print("2. Withdraw")
97         print("3. Check Balance")
98         print("4. Exit")
99         choice = input("Choose an option (1-4): ")
100        if choice == '1':
101            amount = float(input("Enter amount to deposit: "))
102            account.deposit(amount)
103        elif choice == '2':
104            amount = float(input("Enter amount to withdraw: "))
105            account.withdraw(amount)
106        elif choice == '3':
107            account.check_balance()
108        elif choice == '4':
109            break
110
111 if __name__ == '__main__':
112     main()

```

Terminal Output:

```

.py
Enter account holder name: vamshii

Options:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Choose an option (1-4): 1
Enter amount to deposit: 100
Deposited: $100.00

```

Code Analysis :

- ☐ Demonstrates OOP with methods like deposit, withdraw, and check balance.
- ☐ Menu-driven loop allows continuous user interaction.



- ☐ Includes validation for negative deposits and insufficient balance.
- ☐ Keeps balance as an instance variable, showing encapsulation concept.
- ☐ Can be enhanced by adding PIN security or transaction history.