ASSIGNMENT – 6.3

2303A51060

Batch-10

Task-1

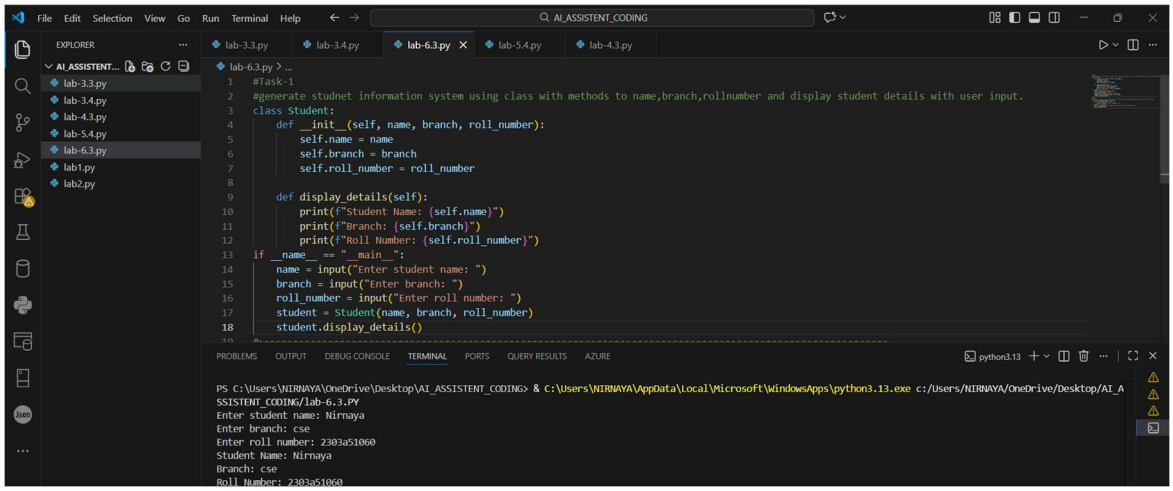
Prompt: generate studnet information system using a class with methods to name, branch,rollnumber and display student details with user input.

Code:

```
class Student:
    def __init__(self, name, branch, roll_number):
        self.name = name
        self.branch = branch
        self.roll_number = roll_number

    def display_details(self):
        print(f"Student Name: {self.name}")
        print(f"Branch: {self.branch}")
        print(f"Roll Number: {self.roll_number}")
if __name__ == "__main__":
    name = input("Enter student name: ")
    branch = input("Enter branch: ")
    roll_number = input("Enter roll number: ")
    student = Student(name, branch, roll_number)
    student.display_details()
```

Output :



Code Analysis :

- A `Student` class is created to represent student data.
- The constructor (`__init__`) initializes name, branch, and roll number.
- Instance variables store individual student details.
- `display_details()` method prints student information neatly.
- User input is used to create an object dynamically at runtime.

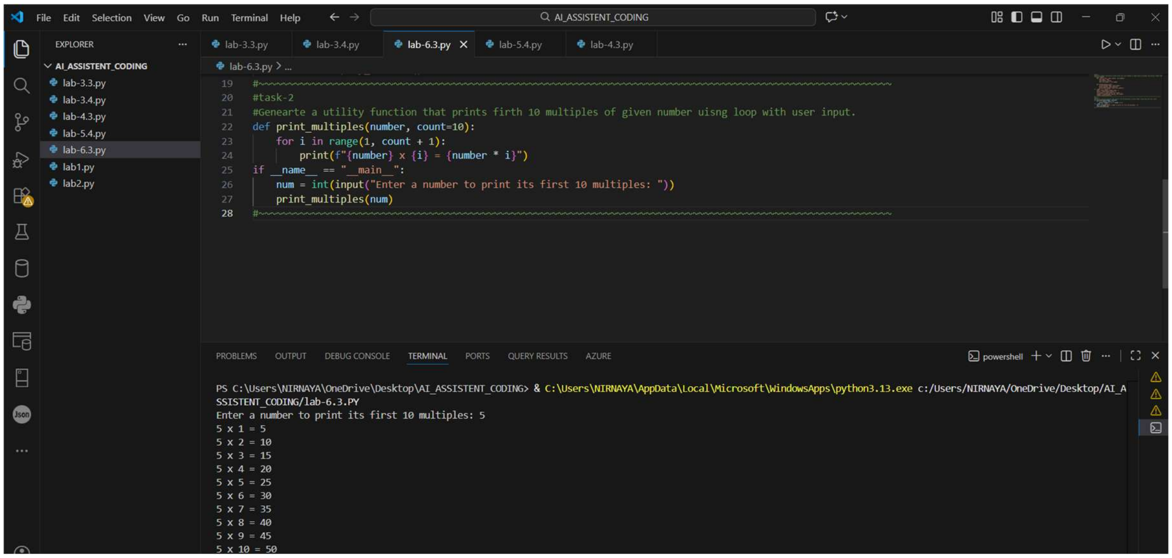
Task-2

Prompt: Generate a utility function that prints firth 10 multiples of a given number using a loop with user input.

Code:

```
def print_multiples(number, count=10):
    for i in range(1, count + 1):
        print(f"{number} x {i} = {number * i}")
if __name__ == "__main__":
    num = int(input("Enter a number to print its first 10 multiples: "))
    print_multiples(num)
```

Output :



Code Analysis :

- The function `print_multiples()` accepts a number and a count as parameters.
- A `for` loop iterates from 1 to 10 to generate multiples.
- Multiplication logic is handled inside the loop.
- The default parameter value ensures flexibility.
- Function improves reusability for different inputs.

Task-3

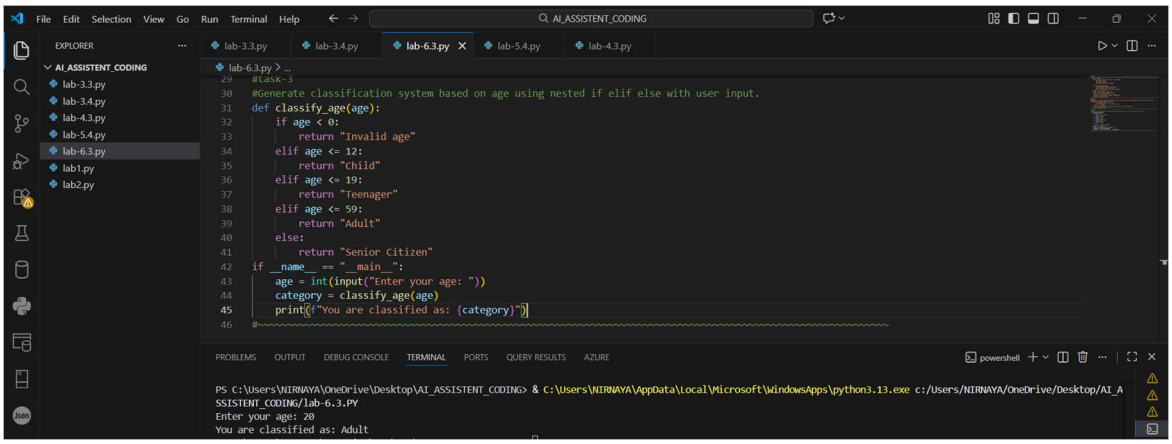
Prompt: Generate a classification system based on age using nested if, elif, and else with user input.

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

Output :



Code Analysis :

- the function `classify_age()` categorizes age groups.
- Conditional statements check age ranges sequentially.
- Invalid inputs are handled using boundary checks.
- Function returns a classification string.
- User input is processed dynamically for classification.

Task-4

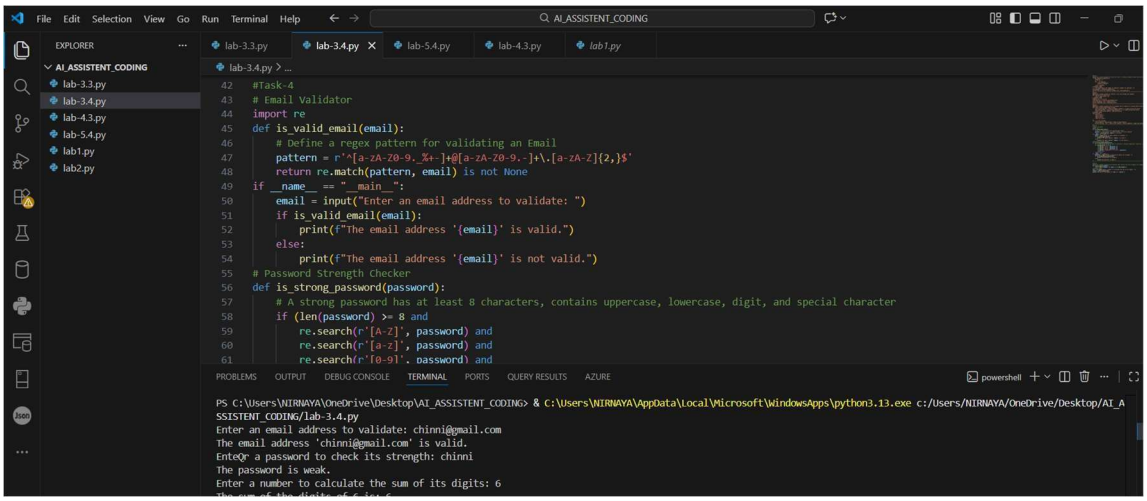
Prompt: Generate the sum of the first n numbers using for and while loops with user input.

Code:

```
def sum_of_numbers(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
```

```
if __name__ == "__main__":
    n = int(input("Enter a positive integer to calculate the sum of first n numbers: "))
    result = sum_of_numbers(n)
    print(f"The sum of the first {n} numbers is: {result}")
```

Output :



Code Analysis :

- The function `sum_of_numbers()` computes sum iteratively.
- A `for` loop runs from 1 to `n` for accumulation.
- Variable `total` stores cumulative sum.
- Function returns computed result instead of printing directly.
- Clean separation of logic and input/output handling.

Task 5

Prompt: Generate a bank application using a class with methods deposit, withdraw and display balance with user input.

Code:

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

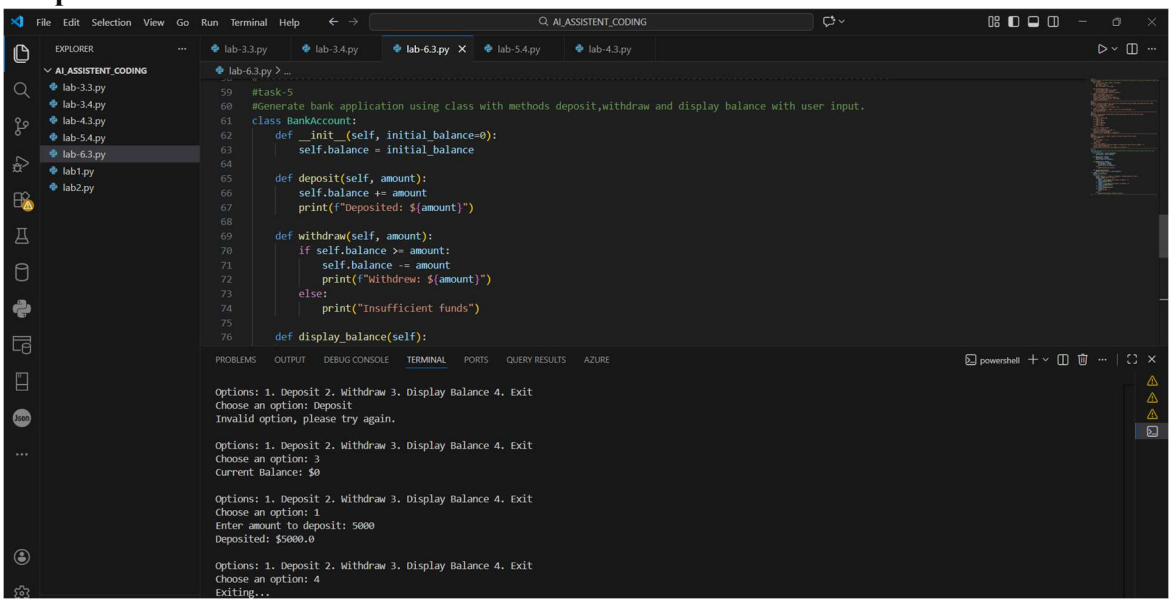
```
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: ${amount}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew: ${amount}")
        else:
            print("Insufficient funds")

    def display_balance(self):
        print(f"Current Balance: ${self.balance}")
if __name__ == "__main__":
    account = BankAccount()
    while True:
        print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
        choice = input("Choose an option: ")
        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(amount)
        elif choice == '3':
            account.display_balance()
        elif choice == '4':
            print("Exiting...")
            break
```

```
    else:
        print("Invalid option, please try again.")
```

Output :



Code Analysis :

- `BankAccount` class models a real-world bank account.
- Constructor initialises account balance.
- `deposit()` and `withdraw()` methods modify balance safely.
- Conditional checks prevent overdraft.
- A menu-driven loop allows continuous user interaction.