

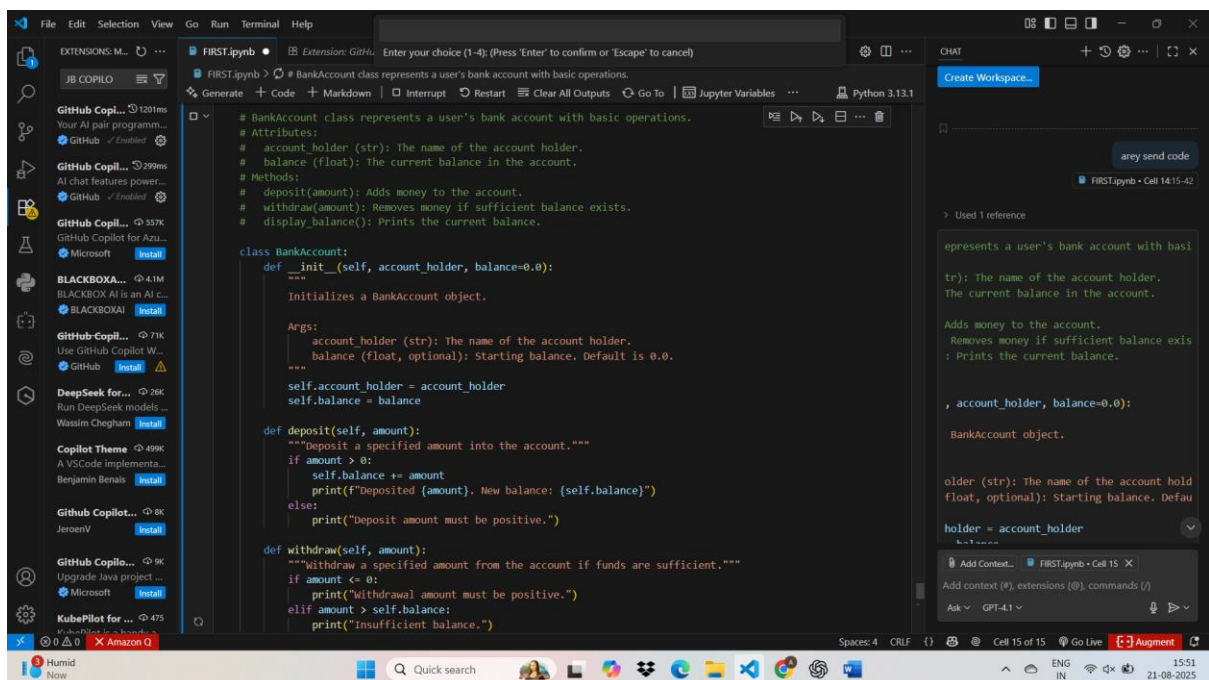
# AI Coding

## Assignment 4.4

2403A51268  
Srinija.G

### TASK-1:

### CODE:



The screenshot shows the VS Code editor with the 'FIRST.ipynb' file open. The code defines a `BankAccount` class with attributes `account_holder` and `balance`, and methods `deposit`, `withdraw`, and `display_balance`. The `display_balance` method is currently commented out. The left sidebar shows the 'EXTENSIONS' panel with various AI-related extensions like GitHub Copilot, DeepSeek, and Copilot Theme. The right sidebar shows the 'CHAT' panel with a 'Create Workspace...' button and a 'Used 1 reference' section.

```
# BankAccount class represents a user's bank account with basic operations.
# Attributes:
#   account_holder (str): The name of the account holder.
#   balance (float): The current balance in the account.
# Methods:
#   deposit(amount): Adds money to the account.
#   withdraw(amount): Removes money if sufficient balance exists.
#   display_balance(): Prints the current balance.

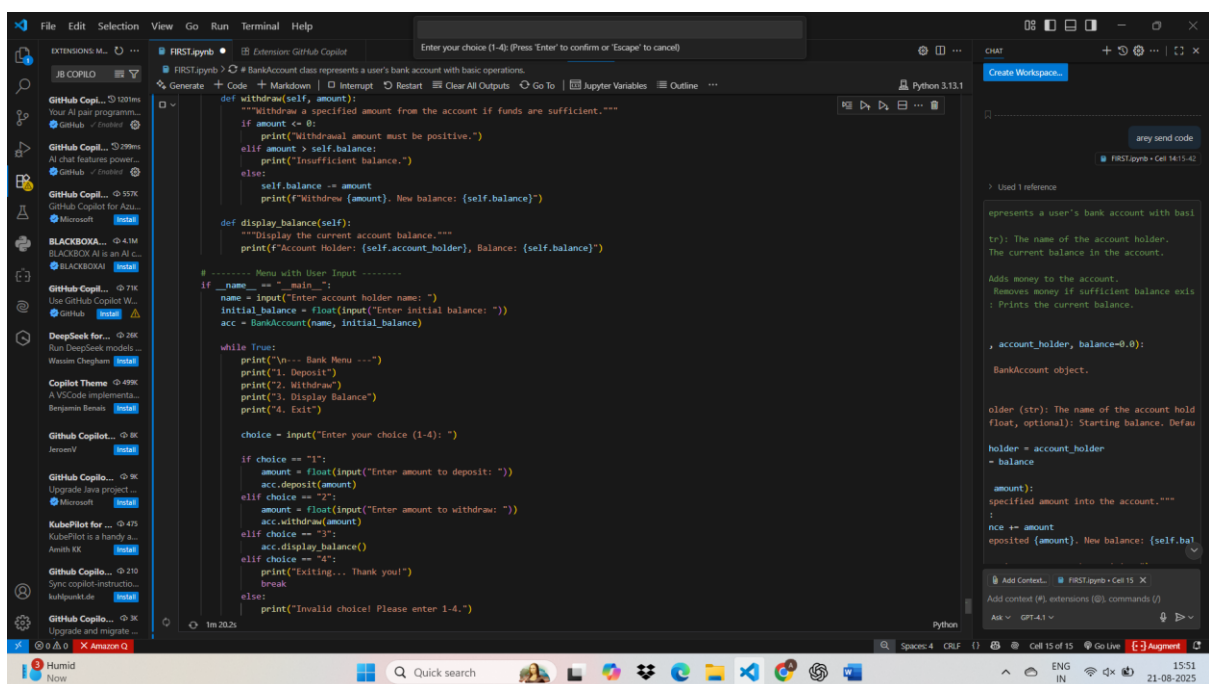
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        """
        Initializes a BankAccount object.

        Args:
            account_holder (str): The name of the account holder.
            balance (float, optional): Starting balance. Default is 0.0.
        """
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """Deposit a specified amount into the account."""
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """Withdraw a specified amount from the account if funds are sufficient."""
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance: {self.balance}")

    def display_balance(self):
        """Display the current account balance."""
        print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")
```



The screenshot shows the VS Code editor with the 'FIRST.ipynb' file open. The code is now complete, including a `__main__` block that creates a `BankAccount` object and a `while` loop that prompts the user to enter a choice (1-4) to perform deposit, withdraw, display balance, or exit. The `display_balance` method is now active. The left sidebar shows the 'EXTENSIONS' panel with various AI-related extensions like GitHub Copilot, DeepSeek, and Copilot Theme. The right sidebar shows the 'CHAT' panel with a 'Create Workspace...' button and a 'Used 1 reference' section.

```
# BankAccount class represents a user's bank account with basic operations.
# Attributes:
#   account_holder (str): The name of the account holder.
#   balance (float): The current balance in the account.
# Methods:
#   deposit(amount): Adds money to the account.
#   withdraw(amount): Removes money if sufficient balance exists.
#   display_balance(): Prints the current balance.

class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        """
        Initializes a BankAccount object.

        Args:
            account_holder (str): The name of the account holder.
            balance (float, optional): Starting balance. Default is 0.0.
        """
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """Deposit a specified amount into the account."""
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """Withdraw a specified amount from the account if funds are sufficient."""
        if amount <= 0:
            print("Withdrawal amount must be positive.")
        elif amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance: {self.balance}")

    def display_balance(self):
        """Display the current account balance."""
        print(f"Account Holder: {self.account_holder}, Balance: {self.balance}")

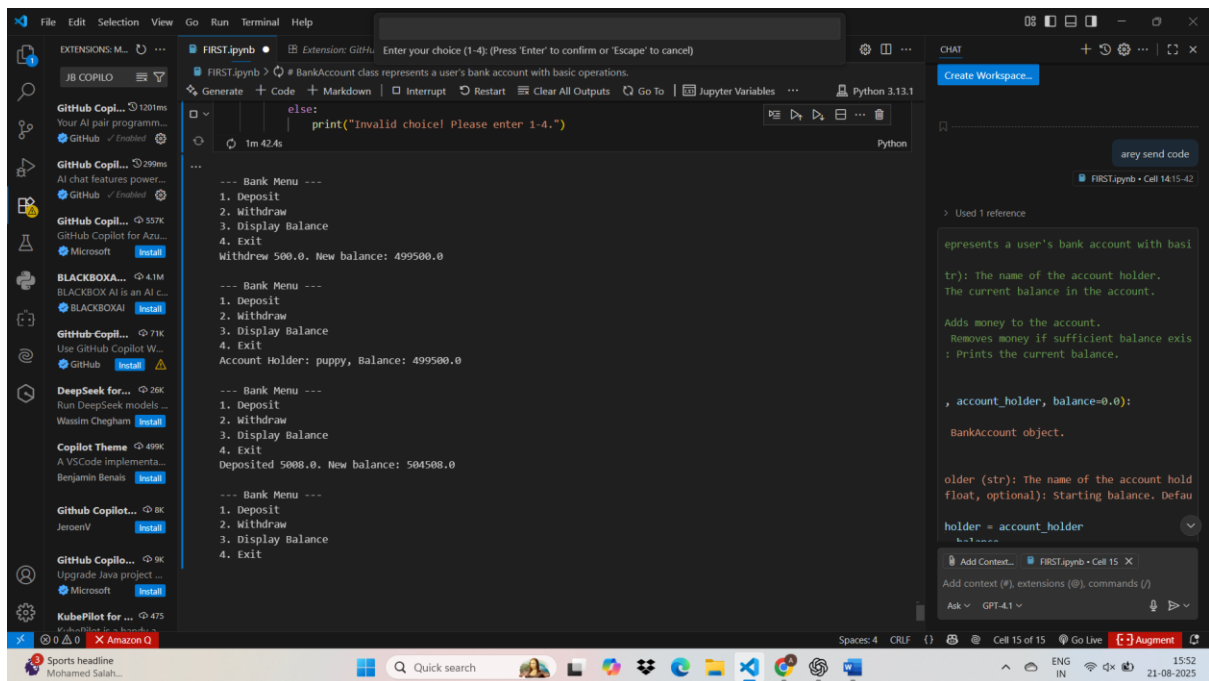
# ----- Menu with User Input -----
if __name__ == "__main__":
    name = input("Enter account holder name: ")
    initial_balance = float(input("Enter initial balance: "))
    acc = BankAccount(name, initial_balance)

    while True:
        print("\n--- Bank Menu ---")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Display Balance")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

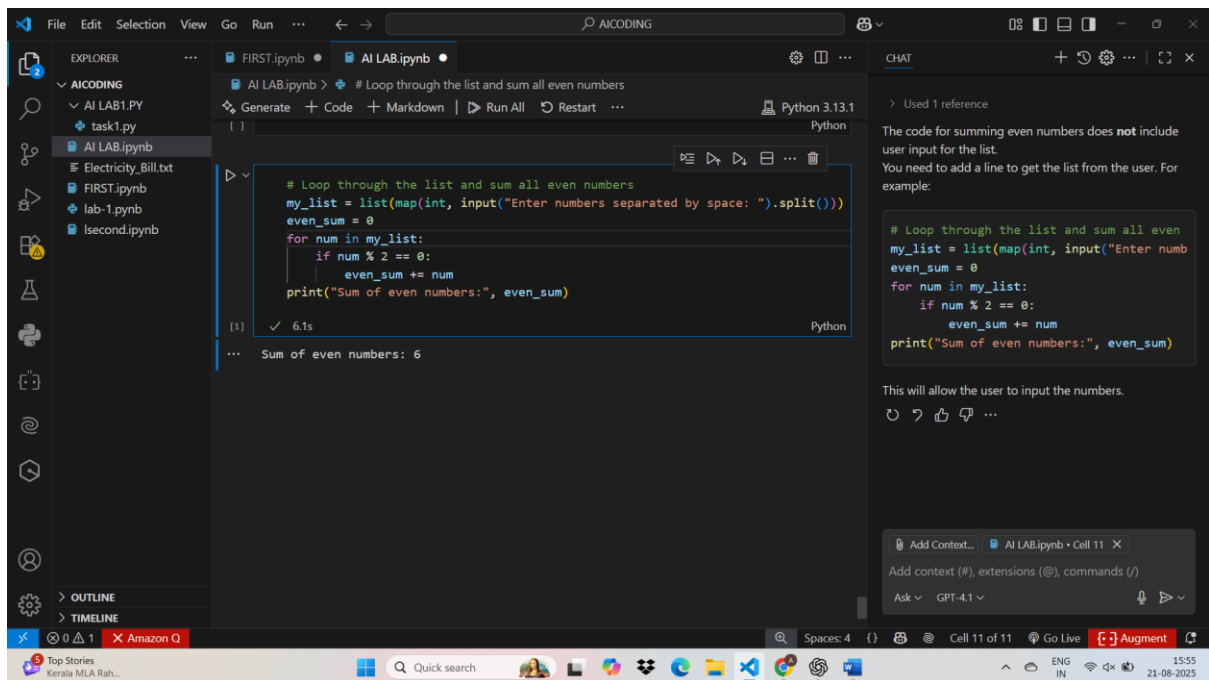
        if choice == "1":
            amount = float(input("Enter amount to deposit: "))
            acc.deposit(amount)
        elif choice == "2":
            amount = float(input("Enter amount to withdraw: "))
            acc.withdraw(amount)
        elif choice == "3":
            acc.display_balance()
        elif choice == "4":
            print("Exiting... Thank you!")
            break
        else:
            print("Invalid choice! Please enter 1-4.")
```

## OUTPUT:



## TASK-2

### CODE WITH OUTPUT:



### TASK3:

#### CODE WITH OUTPUT:

The screenshot shows a Jupyter Notebook with a file explorer on the left containing files like 'AI LAB1.PY', 'task1.py', 'Electricity\_Bill.txt', 'FIRST.ipynb', 'lab-1.pynb', and 'Isecond.ipynb'. The main editor displays a Python function `def age_group(age):` with a docstring and conditional logic to categorize ages into 'Child', 'Teenager', 'Adult', and 'Senior'. Below the function, user input is taken and the function is called. The output shows 'Adult'. A chat window on the right shows a conversation with GPT-4.1, displaying the function code and the output 'Adult, TAKE USER INPUT'.

```
def age_group(age):  
    """Return the age group for a given age."""  
    if age < 13:  
        return "Child"  
    elif age < 20:  
        return "Teenager"  
    elif age < 60:  
        return "Adult"  
    else:  
        return "Senior"  
  
    # Take user input  
    user_age = int(input("Enter age: "))  
    print(age_group(user_age))
```

Output: Adult

### TASK-4:

#### CODE WITH OUTPUT

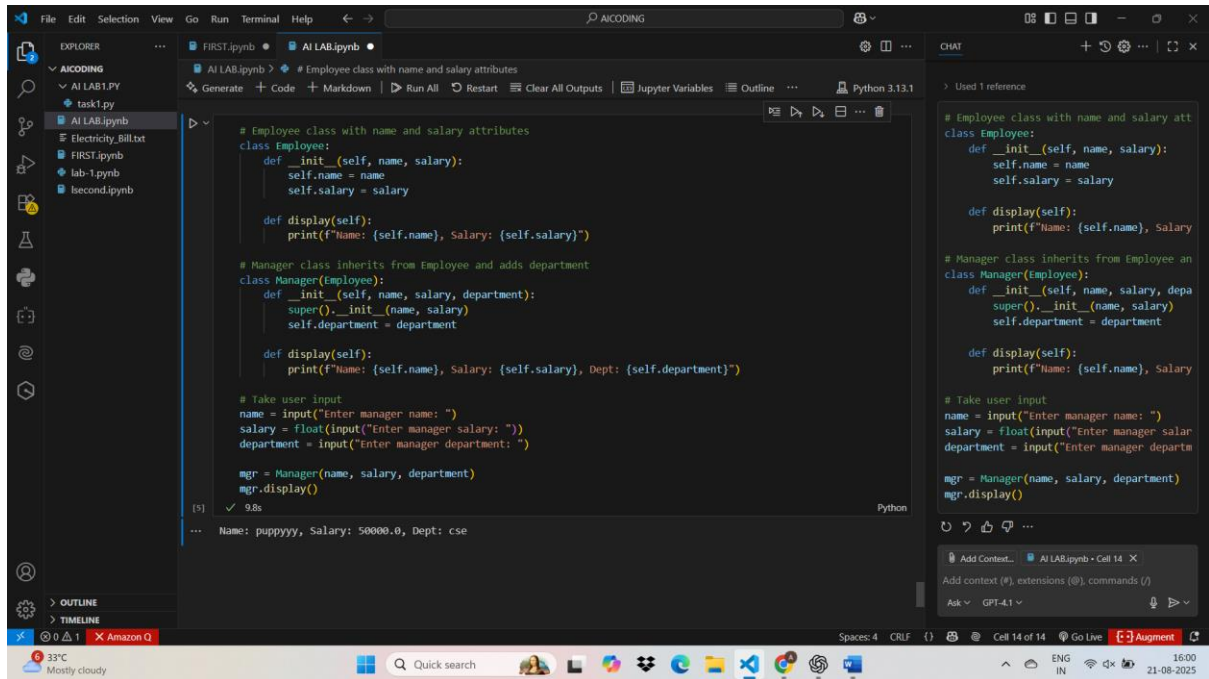
The screenshot shows a Jupyter Notebook with a file explorer on the left containing files like 'AI LAB1.PY', 'task1.py', 'Electricity\_Bill.txt', 'FIRST.ipynb', 'lab-1.pynb', and 'Isecond.ipynb'. The main editor displays a Python function `# Reverse the digits of a number using a while loop` that takes a number as input and reverses its digits using a while loop. The output shows 'Reversed number: 325'. A chat window on the right shows a conversation with GPT-4.1, displaying the function code and the output 'Reversed number: 325'.

```
# Reverse the digits of a number using a while loop  
num = int(input("Enter a number to reverse: "))  
reverse = 0  
  
while num > 0:  
    digit = num % 10  
    reverse = reverse * 10 + digit  
    num //= 10  
  
print("Reversed number:", reverse)
```

Output: Reversed number: 325

## TASK 5:

Code with output:



The screenshot displays a Jupyter Notebook environment with the following components:

- EXPLORER:** Lists files including `AI LAB1.PY`, `task1.py`, `AI LAB.ipynb`, `Electricity_Bill.txt`, `FIRST.ipynb`, `lab-1.ipynb`, and `second.ipynb`.
- Code Editor:** Contains the following Python code:

```
# Employee class with name and salary attributes
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}")

# Manager class inherits from Employee and adds department
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}")

# Take user input
name = input("Enter manager name: ")
salary = float(input("Enter manager salary: "))
department = input("Enter manager department: ")

mgr = Manager(name, salary, department)
mgr.display()
```
- Output:** The execution result shows the input values and the formatted output:

```
[5] ✓ 98s Python
... Name: puppyyy, Salary: 50000.0, Dept: cse
```
- CHAT:** Displays a reference to the code used, showing the class definitions and input/output logic.
- Bottom Bar:** Includes system status (33°C, Mostly cloudy), search bar, and task progress (Cell 14 of 14, Go Live, Augment).