

LAB ASSIGNMENT- 4.4

Name: V . Vishnu Vardhan

Enrollment No.: 2503A51L26

Course Code: 24CS002PC215

Course Title: AI Assisted Coding

Lab Number: 4.4

BRANCH: CSE

Task 1: Auto-Complete a Python Class for Bank Account

Prompt: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.

Python Code:

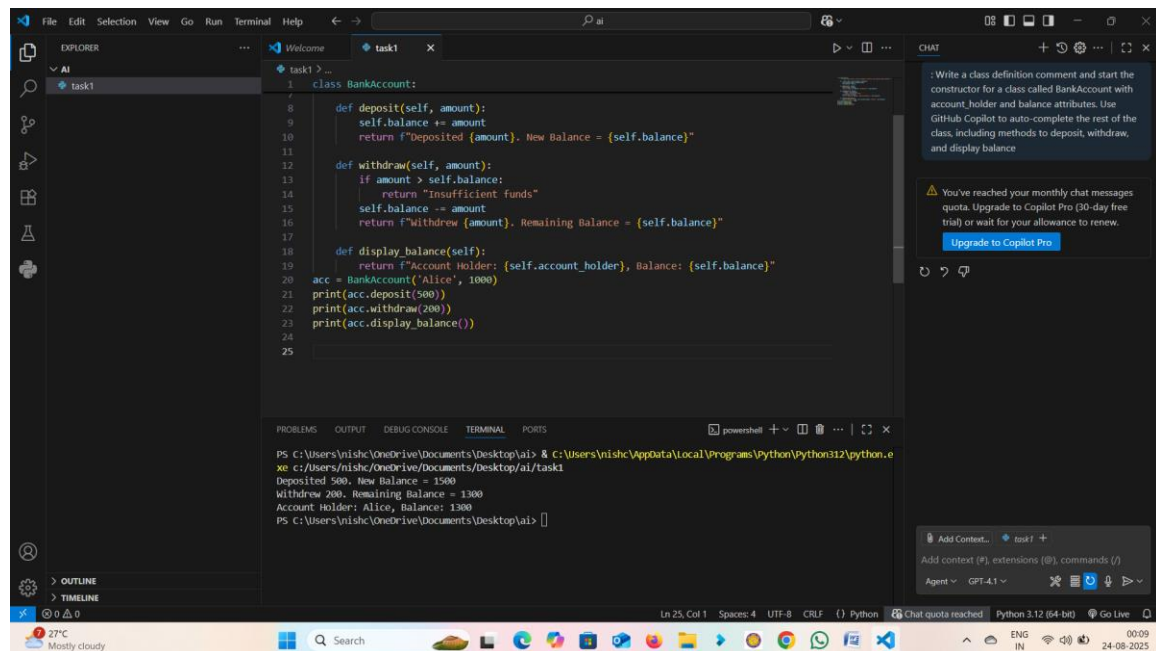
```
class BankAccount:
    """A simple Bank Account class with deposit, withdraw, and display balance methods."""
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        return f"Deposited {amount}. New Balance = {self.balance}"

    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        self.balance -= amount
        return f"Withdrew {amount}. Remaining Balance = {self.balance}"

    def display_balance(self):
        return f"Account Holder: {self.account_holder}, Balance: {self.balance}"
```

Explanation: The class has attributes for account holder and balance. Methods allow deposit, withdrawal with balance check, and displaying account details.



Sample Output:

```

>>> acc = BankAccount('Alice', 1000)
>>> print(acc.deposit(500))
Deposited 500. New Balance = 1500
>>> print(acc.withdraw(200))
Withdrew 200. Remaining Balance = 1300
>>> print(acc.display_balance())
Account Holder: Alice, Balance: 1300

```

Observation: The BankAccount class successfully handled deposits, withdrawals, and displayed balance accurately.

Task 2: Auto-Complete a For Loop to Sum Even Numbers in a List

Prompt: Write a comment and the initial line of a loop to iterate over a list. Allow GitHub Copilot to complete the logic to sum all even numbers in the list.

Python Code:

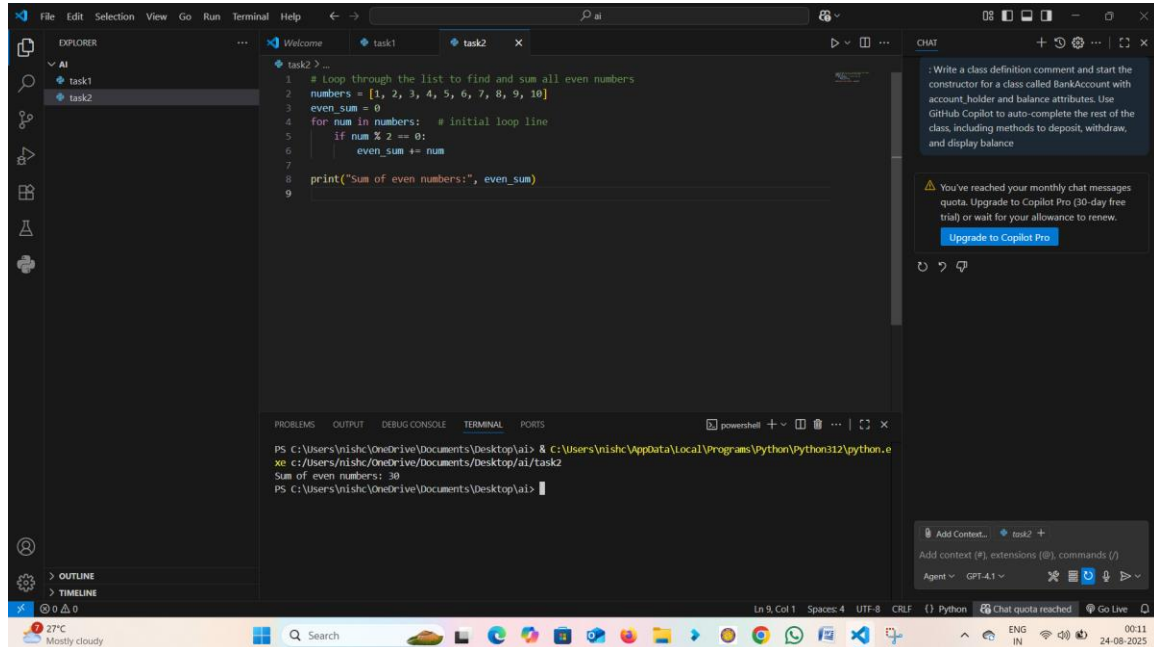
```

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num

```

```
print("Sum of even numbers:", even_sum)
```

Explanation: The loop iterates through the list, checks if each number is even, and adds it to the running sum.



```
1 # Loop through the list to find and sum all even numbers
2 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3 even_sum = 0
4 for num in numbers: # Initial loop line
5     if num % 2 == 0:
6         even_sum += num
7
8 print("Sum of even numbers:", even_sum)
9
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & c:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe c:\Users\nishc\OneDrive\Documents\Desktop\ai\task2
Sum of even numbers: 30
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai>
```

Sample Output:

Sum of even numbers: 30

Observation: The loop correctly iterated and summed even numbers from the list.

Task 3: Auto-Complete Conditional Logic to Check Age Group

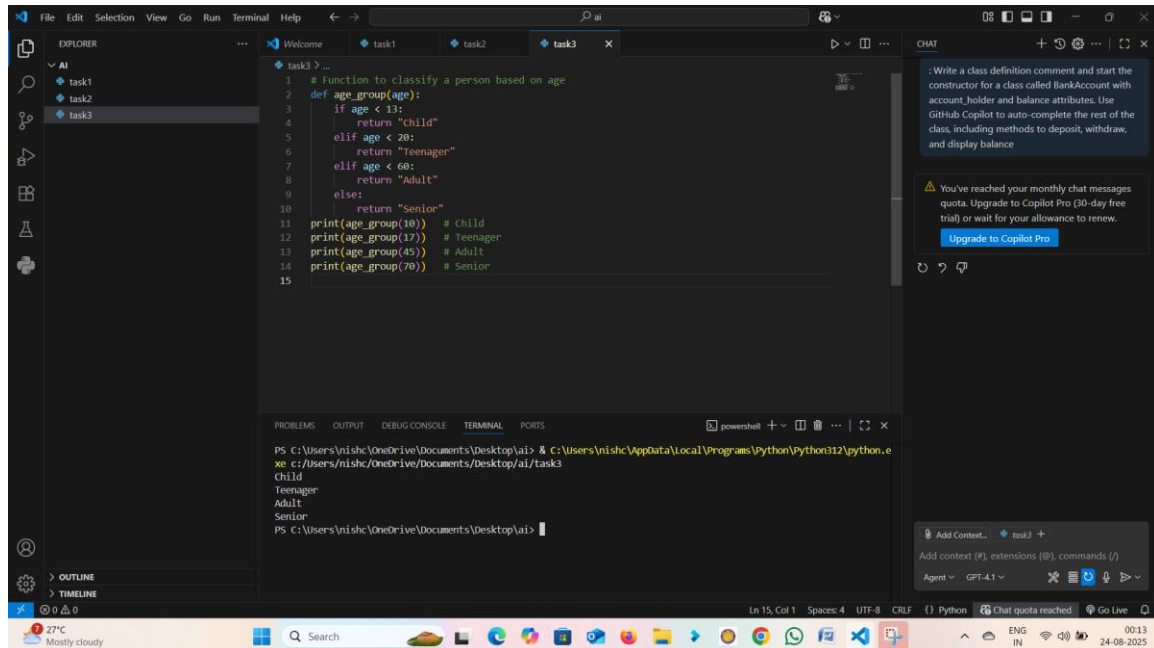
Prompt: Start a function that takes age as input and returns whether the person is a child, teenager, adult, or senior using if-elif-else.

Python Code:

```
def age_group(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
```

```
return "Senior"
```

Explanation: The function uses if-elif-else conditionals to classify age groups.



The screenshot shows a Visual Studio Code editor with a Python file named 'task3.py'. The code defines a function 'age_group' that classifies a person's age into 'Child', 'Teenager', 'Adult', or 'Senior' based on if-elif-else conditions. Below the function definition, there are four print statements testing the function with ages 10, 17, 45, and 70. The terminal at the bottom shows the output of these tests: 'Child', 'Teenager', 'Adult', and 'Senior'. The right sidebar shows the 'CHAT' panel with a message from GitHub Copilot suggesting a class definition for 'BankAccount'.

```
1 # Function to classify a person based on age
2 def age_group(age):
3     if age < 13:
4         return "Child"
5     elif age < 20:
6         return "Teenager"
7     elif age < 60:
8         return "Adult"
9     else:
10        return "Senior"
11 print(age_group(10)) # Child
12 print(age_group(17)) # Teenager
13 print(age_group(45)) # Adult
14 print(age_group(70)) # Senior
15
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe c:\Users\nishc\OneDrive\Documents\Desktop\ai\task3
Child
Teenager
Adult
Senior
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai>
```

Sample Output:

```
>>> age_group(10) -> Child
>>> age_group(17) -> Teenager
>>> age_group(45) -> Adult
>>> age_group(70) -> Senior
```

Observation: The function correctly classified age groups based on input values.

Task 4: Auto-Complete a While Loop to Reverse Digits of a Number

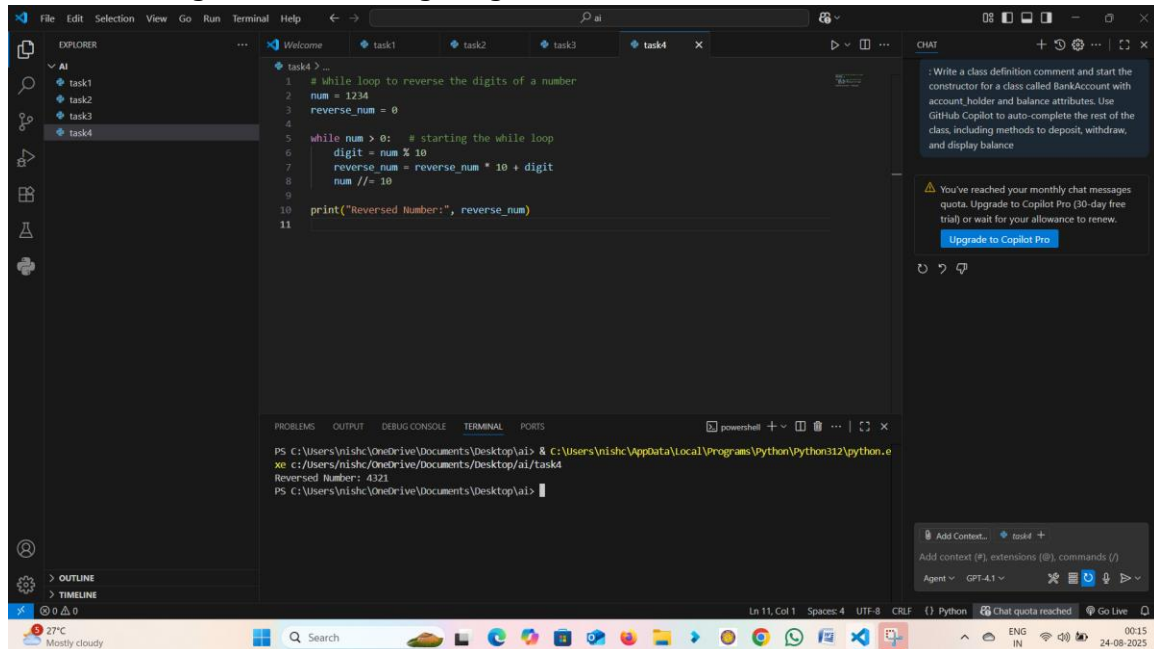
Prompt: Write a comment and start a while loop to reverse the digits of a number.

Python Code:

```
num = 1234
reverse_num = 0
while num > 0:
    digit = num % 10
    reverse_num = reverse_num * 10 + digit
    num //= 10
```

```
print("Reversed Number:", reverse_num)
```

Explanation: The loop extracts the last digit using modulo, builds the reversed number, and reduces the original number using integer division.



```
1 # while loop to reverse the digits of a number
2 num = 1234
3 reverse_num = 0
4
5 while num > 0: # starting the while loop
6     digit = num % 10
7     reverse_num = reverse_num * 10 + digit
8     num //= 10
9
10 print("Reversed Number:", reverse_num)
11
```

PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe c:\Users\nishc\OneDrive\Documents\Desktop\ai\task4
Reversed Number: 4321
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai>

Sample Output:

Reversed Number: 4321

Observation: The while loop reversed the digits of the number without errors.

Task 5: Auto-Complete Class with Inheritance (Employee → Manager)

Prompt: Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds a department.

Python Code:

class Employee:

```
def __init__(self, name, salary):
    self.name = name
    self.salary = salary
```

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

```

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

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

```

Explanation: The Manager class inherits from Employee using super() for constructor chaining and overrides the display method.

The screenshot shows a Visual Studio Code editor with a Python file open. The code defines an `Employee` class with an `__init__` method and a `display` method. A `Manager` class inherits from `Employee`, overriding the `__init__` method to include a `department` parameter and overriding the `display` method to include the department in the output string. The terminal shows the execution of the code, creating a `Manager` object and calling its `display` method, resulting in the output: `Name: John, Salary: 50000, Dept: IT`. The chat window on the right shows a message from GitHub Copilot suggesting to write a class definition comment and start the constructor for a class called `BankAccount`.

```

class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        return f'Name: {self.name}, Salary: {self.salary}'

# Derived class Manager inheriting from Employee and adding department
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

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

mgr = Manager("John", 50000, "IT")
print(mgr.display())

```

```

PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe C:\Users\nishc\OneDrive\Documents\Desktop\ai\task5
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.exe C:\Users\nishc\OneDrive\Documents\Desktop\ai\task5
Name: John, Salary: 50000, Dept: IT
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai>

```

Chat: : Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance.

You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Upgrade to Copilot Pro

Add Context... task5 +

Add context (#), extensions (@), commands (/)

Agent: GPT-4.1

Ln 21, Col 1 Spaces: 4 UTF-8 CRLF Python Chat quota reached Go Live

Sample Output:

Name: John, Salary: 50000, Dept: IT

Observation: The Manager class inherited Employee attributes and methods correctly while extending functionality.

Observation

In this lab, we explored GitHub Copilot's ability to auto-complete Python code for classes, loops, and conditionals. We practiced building classes with inheritance, loops for summing and reversing, and conditional logic for classification. This enhanced understanding of AI-assisted coding and Python fundamentals.