AI ASSISTED CODING

ASSIGNMENT-6.1

HTNO: 2303A510B2

NAME: K AKHIL

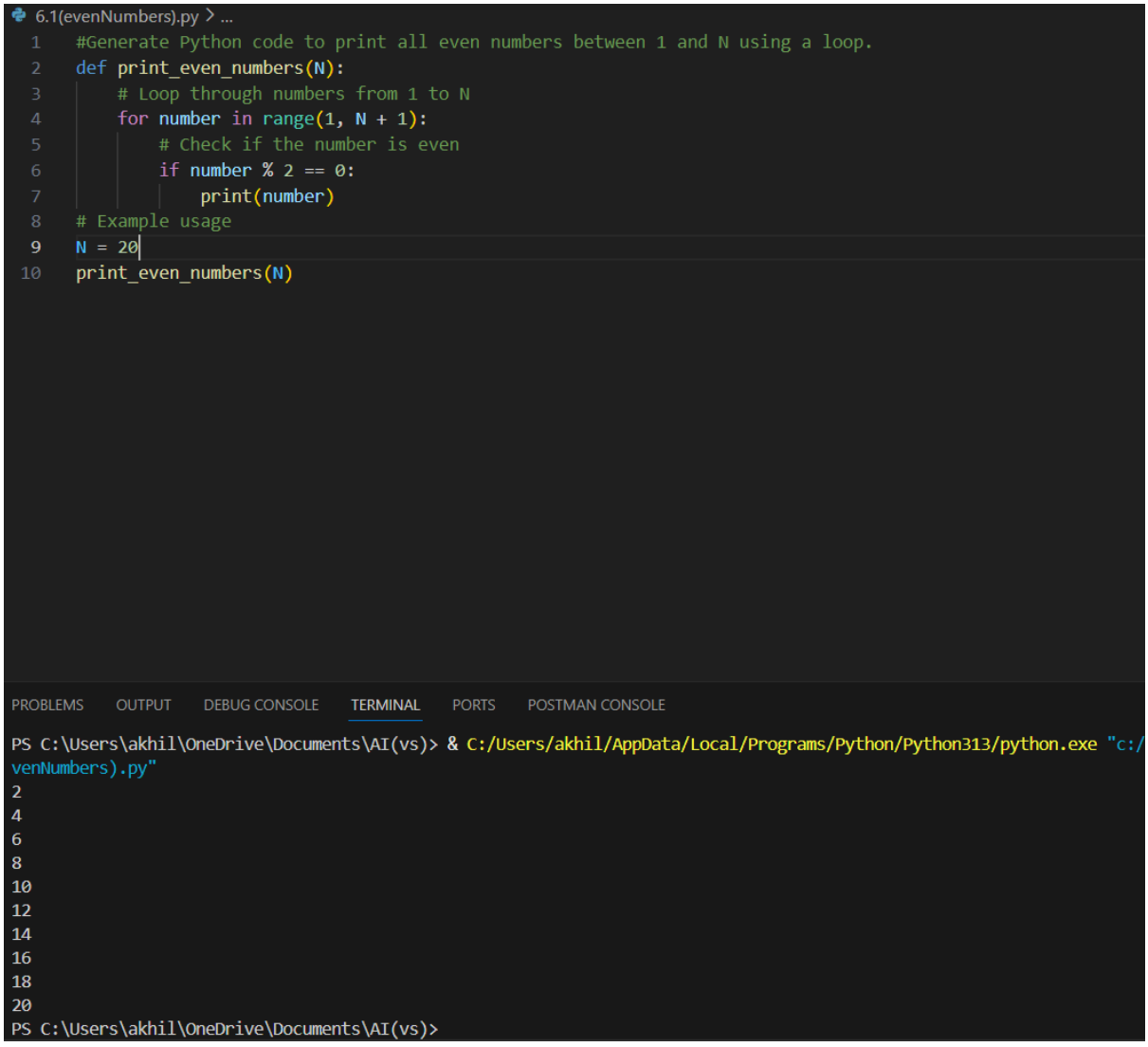
BATCH-14

Task Description #1 (AI-Based Code Completion for Loops)

Task: Use an AI code completion tool to generate a loop-based program.

PROMPT: Generate Python code to print all even numbers between 1 and N using a loop.

CODE AND OUTPUT:



JUSTIFICATION:

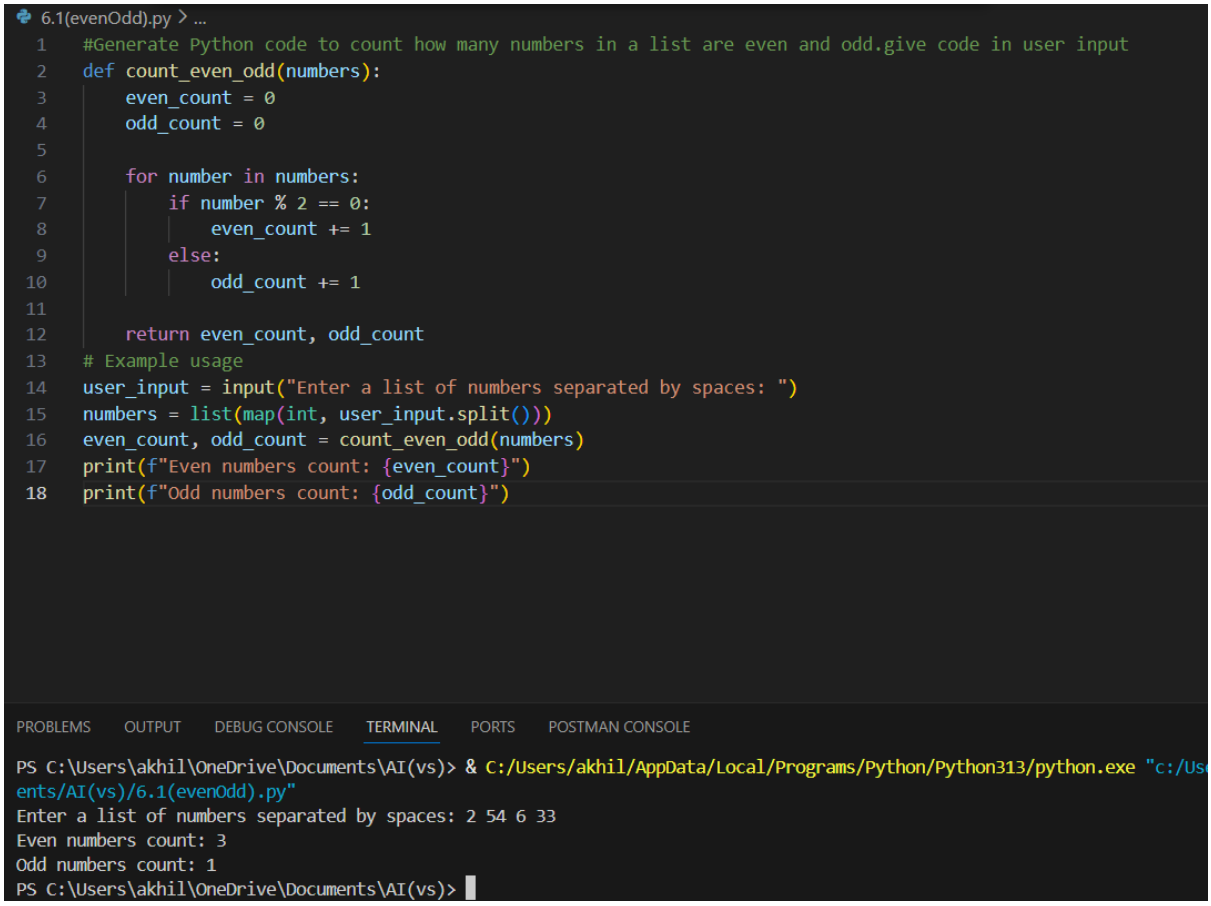
This task demonstrates the effective use of an AI tool to generate a loop-based program. The AI correctly identified that a for loop is suitable for iterating through a fixed range of numbers. Conditional logic was applied to check even numbers using the modulo operator. This reduces manual effort and ensures logical accuracy while reinforcing basic looping concepts.

Task Description #2 (AI-Based Code Completion for Loop with Conditionals)

Task: Use an AI code completion tool to combine loops and conditionals.

PROMPT: Generate Python code to count how many numbers in a list are even and odd.

CODE AND OUTPUT:



JUSTIFICATION:

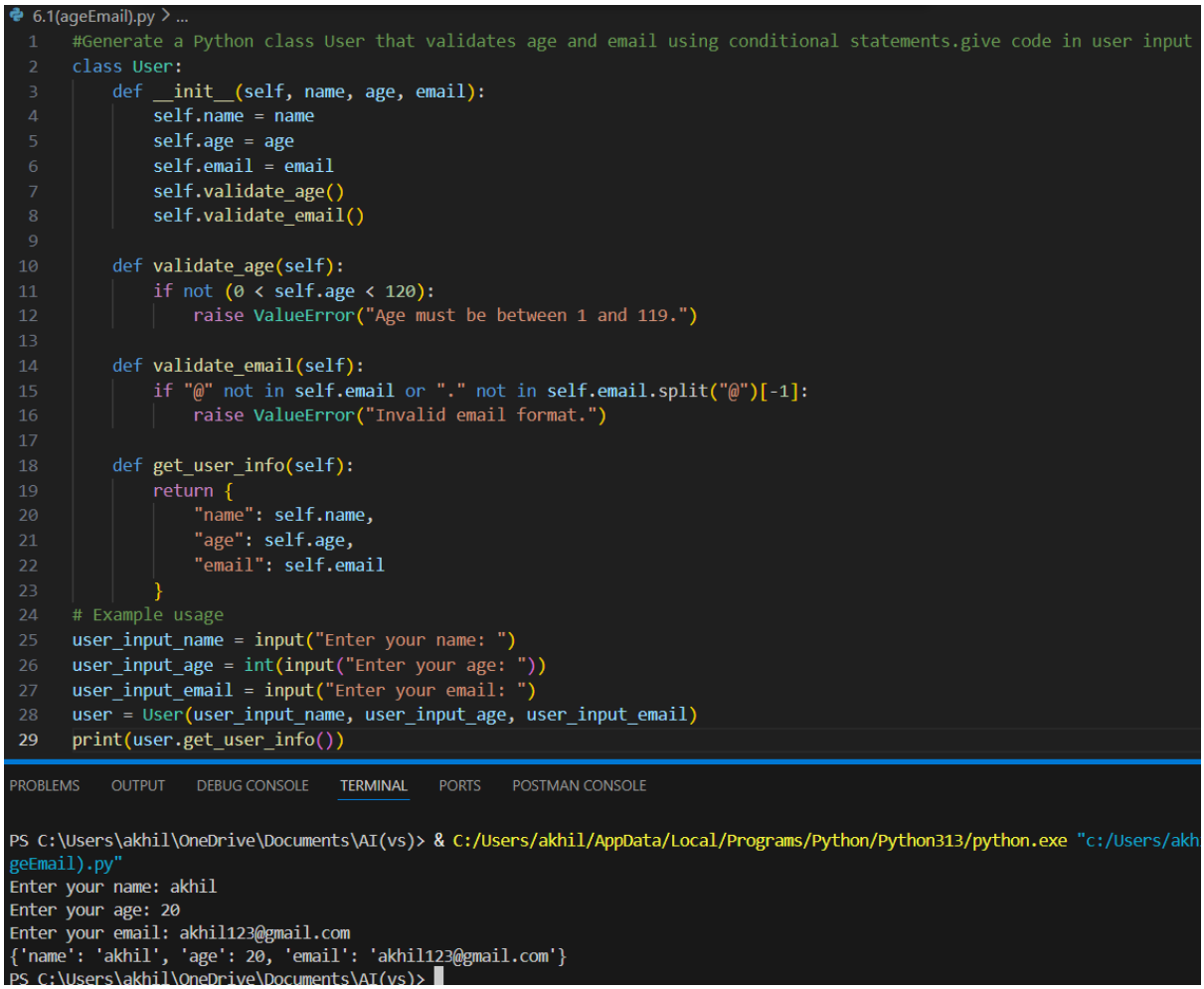
This task validates the AI's ability to combine loops and conditional statements effectively. The AI-generated solution correctly iterates through a list and classifies numbers as even or odd. The logic improves code clarity and efficiency, showing how AI can assist in writing structured programs involving decision-making and iteration.

Task Description #3 (AI-Based Code Completion for Class Attributes Validation)

Task: Use an AI tool to complete a Python class that validates user input.

PROMPT: Generate a Python class User that validates age and email using conditional statements**.**

CODE AND OUTPUT:



```
6.1(ageEmail).py > ...
#Generate a Python class User that validates age and email using conditional statements.give code in user input
class User:
    def __init__(self, name, age, email):
        self.name = name
        self.age = age
        self.email = email
        self.validate_age()
        self.validate_email()

    def validate_age(self):
        if not (0 < self.age < 120):
            raise ValueError("Age must be between 1 and 119.")

    def validate_email(self):
        if "@" not in self.email or "." not in self.email.split("@")[-1]:
            raise ValueError("Invalid email format.")

    def get_user_info(self):
        return {
            "name": self.name,
            "age": self.age,
            "email": self.email
        }
# Example usage
user_input_name = input("Enter your name: ")
user_input_age = int(input("Enter your age: "))
user_input_email = input("Enter your email: ")
user = User(user_input_name, user_input_age, user_input_email)
print(user.get_user_info())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akh
geEmail).py"
Enter your name: akhil
Enter your age: 20
Enter your email: akhil123@gmail.com
{'name': 'akhil', 'age': 20, 'email': 'akhil123@gmail.com'}
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task highlights the use of AI in generating class-based validation logic. The AI correctly implemented conditional checks for age and email format. This improves data reliability and demonstrates object-oriented programming concepts. AI assistance reduces development time while maintaining proper input validation mechanisms.

Task Description #4 (AI-Based Code Completion for Classes)

Task: Use an AI code completion tool to generate a Python class for

managing student details.

PROMPT: Generate a Python class Student with attributes (name, roll number,

marks) and methods to calculate total and average marks.

CODE :

6.1(class).py > ...

```python
#: Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total and average marks.
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks  # marks should be a list of numbers

    def calculate_total(self):
        return sum(self.marks)

    def calculate_average(self):
        if len(self.marks) == 0:
            return 0
        return self.calculate_total() / len(self.marks)

    def get_student_info(self):
        return {
            "name": self.name,
            "roll_number": self.roll_number,
            "marks": self.marks,
            "total_marks": self.calculate_total(),
            "average_marks": self.calculate_average()
        }
# Example usage
student_name = input("Enter student name: ")
student_roll_number = input("Enter roll number: ")
marks_input = input("Enter marks separated by spaces: ")
marks_list = list(map(float, marks_input.split()))
student = Student(student_name, student_roll_number, marks_list)
print(student.get_student_info())
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/D
Enter student name: akhil
Enter roll number: 10b2
Enter marks separated by spaces: 90 95 80 96
{'name': 'akhil', 'roll_number': '10b2', 'marks': [90.0, 95.0, 80.0, 96.0], 'total_marks': 361.0, 'average_marks': 90.25}
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

JUSTIFICATION:

This task shows how AI helps in designing a **complete and well-structured class** with attributes and methods. The generated class correctly calculates total and average marks. Minor manual improvements were needed to handle edge cases, showing that AI provides a strong foundation while human oversight ensures robustness.

Task Description 5 (AI-Assisted Code Completion Review)

Task: Use an AI tool to generate a complete Python program using

classes, loops, and conditionals together.

PROMPT: Generate a Python program for a simple bank account system using

class, loops, and conditional statements.

CODE :

6.1(bank).py > ...

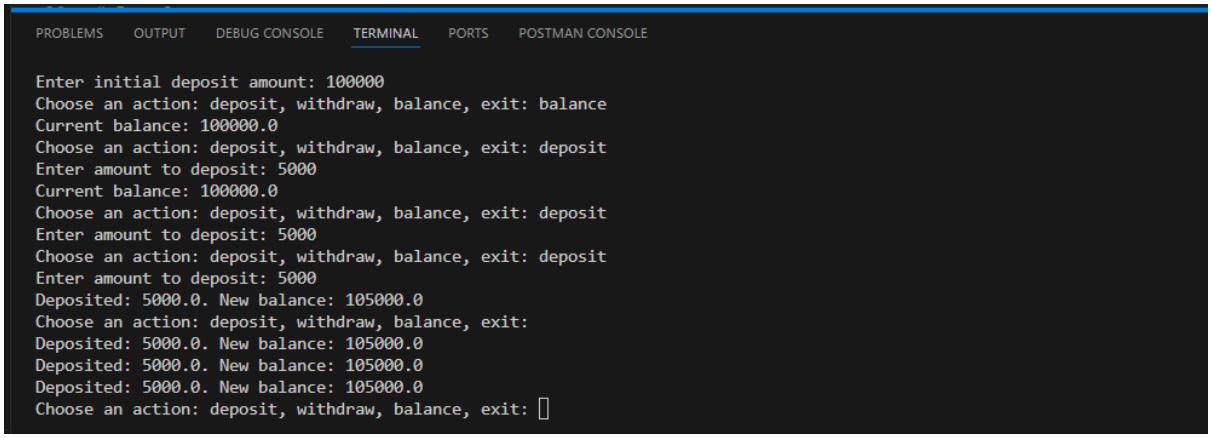
```python
#Generate a Python program for a simple bank account system using class, loops, and conditional statements.
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

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

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew: {amount}. New balance: {self.balance}")
            else:
                print("Insufficient funds for this withdrawal.")
        else:
            print("Withdrawal amount must be positive.")

    def get_balance(self):
        return self.balance
# Example usage
account_holder_name = input("Enter account holder name: ")
initial_deposit = float(input("Enter initial deposit amount: "))
account = BankAccount(account_holder_name, initial_deposit)
while True:
    action = input("Choose an action: deposit, withdraw, balance, exit: ").lower()
    if action == "deposit":
        amount = float(input("Enter amount to deposit: "))
        account.deposit(amount)
    elif action == "withdraw":
        amount = float(input("Enter amount to withdraw: "))
        account.withdraw(amount)
    elif action == "balance":
        print(f"Current balance: {account.get_balance()}")
    elif action == "exit":
        print("Exiting the banking system. Goodbye!")
        break
    else:
        print("Invalid action. Please choose again.")
```

OUTPUT:



JUSTIFICATION:

This task demonstrates AI's capability to generate a **complete working program** using classes, loops, and conditionals together. The AI-generated bank system simplifies program structure and improves coding speed. However, limitations such as missing validations highlight the importance of human judgment. Overall, AI significantly enhances coding productivity and learning efficiency.