AI ASSISTED CODING

HTNO:2303A510A3

Batch:14

Name: M. Sai Teja

#Task-1:

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

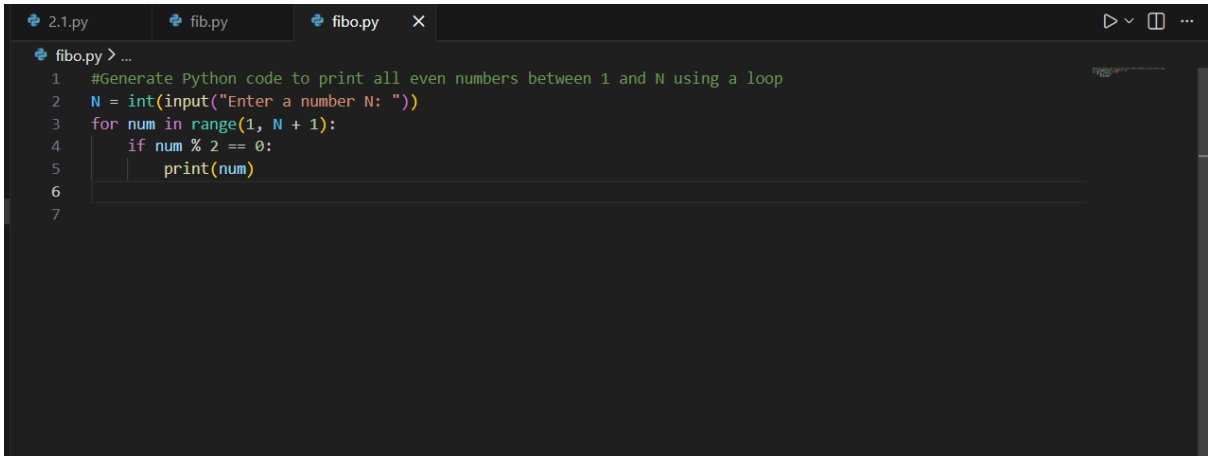
Task: Use an AI code completion tool to generate a loop-based

program.

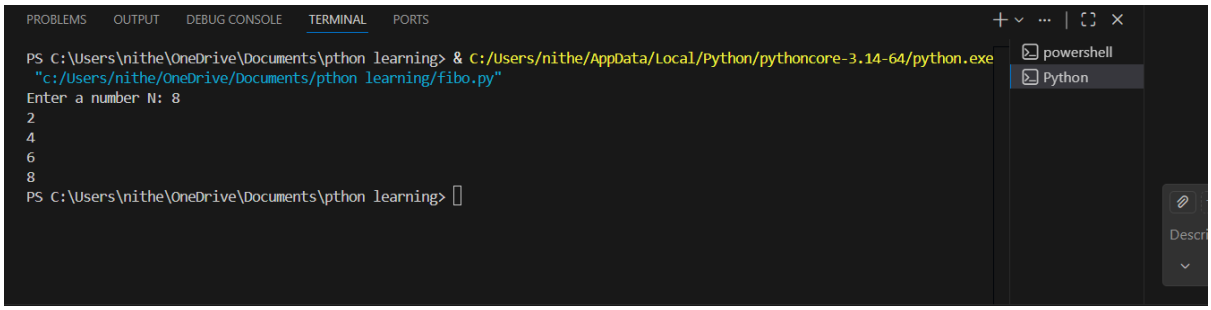
Promt:

#Generate Python code to print all even numbers between 1 and N using a loop

Code:



Output:



Justification:

The program uses a **for loop** to traverse numbers from 1 to N.

An **if condition** checks whether each number is even using the modulo operator.

Only numbers satisfying the condition are printed, ensuring correct identification of even numbers.

#Task-2

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 a python code to count how many even and odd numbers are thre in a List

Code:

```python
#Generate a python code to count how many even and odd numbers are thre in a List
numbers = [10, 21, 32, 43, 54, 65, 76, 87, 98]
even_count = 0
odd_count = 0
for number in numbers:
    if number % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
print("Even numbers count:", even_count)
print("Odd numbers count:", odd_count)
```

Output:

```
PS C:\Users\nithe\OneDrive\Documents\pthon learning> & C:/Users/nithe/AppData/Local/Python/pythoncore-3.14-64/python.exe
 "c:/Users/nithe/OneDrive/Documents/pthon learning/fibo.py"
Even numbers count: 5
Odd numbers count: 4
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

Justification:

The program uses a **for loop** to iterate through each number in the list.

An **if-else conditional** checks whether the current number is divisible by 2 to determine even or odd.

Separate counters are updated accordingly, and the final counts are displayed as output.

#Task-3:

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

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

Prompt:

#Generate a Python class User that validates age and email using conditional statemets

Code:

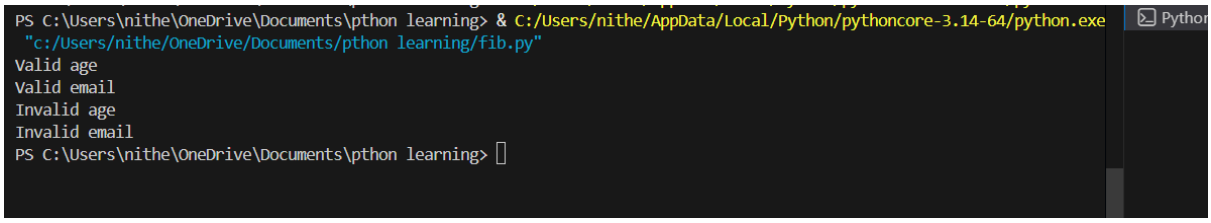
```python
#Generate a Python class User that validates age and email using conditional statemets
class User:
    def __init__(self, age, email):
        self.age = age
        self.email = email

    def validate_age(self):
        if isinstance(self.age, int) and self.age >= 18:
            return "Valid age"
        else:
            return "Invalid age"

    def validate_email(self):
        if "@" in self.email and "." in self.email:
            return "Valid email"
        else:
            return "Invalid email"
# Valid input
# Valid input
user1 = User(25, "user@example.com")
print(user1.validate_age())    # Valid age
print(user1.validate_email())  # Valid email

# Invalid input
user2 = User(15, "userexample.com")
print(user2.validate_age())    # Invalid age
print(user2.validate_email())  # Invalid email
```

Output:



Justification:

The AI-generated User class uses **conditional statements** to validate age and email attributes based on predefined rules.

Separate validation methods ensure **proper condition handling** for both valid and invalid inputs.

Test cases confirm that the class correctly distinguishes between acceptable and unacceptable user data.

Task-4:

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:

```python

#“Generate a Python class Student with attributes (name, roll number,marks) and methods to calculate total an
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 integers

    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)
# Example usage
student1 = Student("Alice", 101, [85, 90, 78, 92])
print("Total Marks:", student1.calculate_total())      # Total Marks: 345
print("Average Marks:", student1.calculate_average())
print("Roll number:", student1.roll_number) # Roll number: 101
student2 = Student("Bob", 102, [70, 75, 80])
print("Total Marks:", student2.calculate_total())      # Total Marks: 225
print("Average Marks:", student2.calculate_average())  # Average Marks: 75.0
print("Roll number:", student2.roll_number) # Roll number: 102
                              student2
```

Output:

```
PS C:\Users\nithe\OneDrive\Documents\pthon learning> & C:/Users/nithe/AppData/Local/Python/pythoncore-3.14-64/python.exe
 "c:/Users/nithe/OneDrive/Documents/pthon learning/fibo.py"
Total Marks: 345
Average Marks: 86.25
Roll number: 101
Total Marks: 225
Average Marks: 75.0
Roll number: 102
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

Justification:

The AI-generated Student class correctly defines attributes and methods, ensuring a **complete and well-structured class design**.

Methods for calculating total and average marks verify the **correctness of logic and data handling** within the class.

Task-5:

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:

```python
#Generate a Python program for a simple bank account system using class,loops, and conditional statements
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = 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.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"Account holder: {self.account_holder}, Balance: {self.balance}")
# Example usage
account = BankAccount("John Doe", 1000)
account.display_balance()
while True:
    print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
    choice = input("Enter your choice (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: "))
```

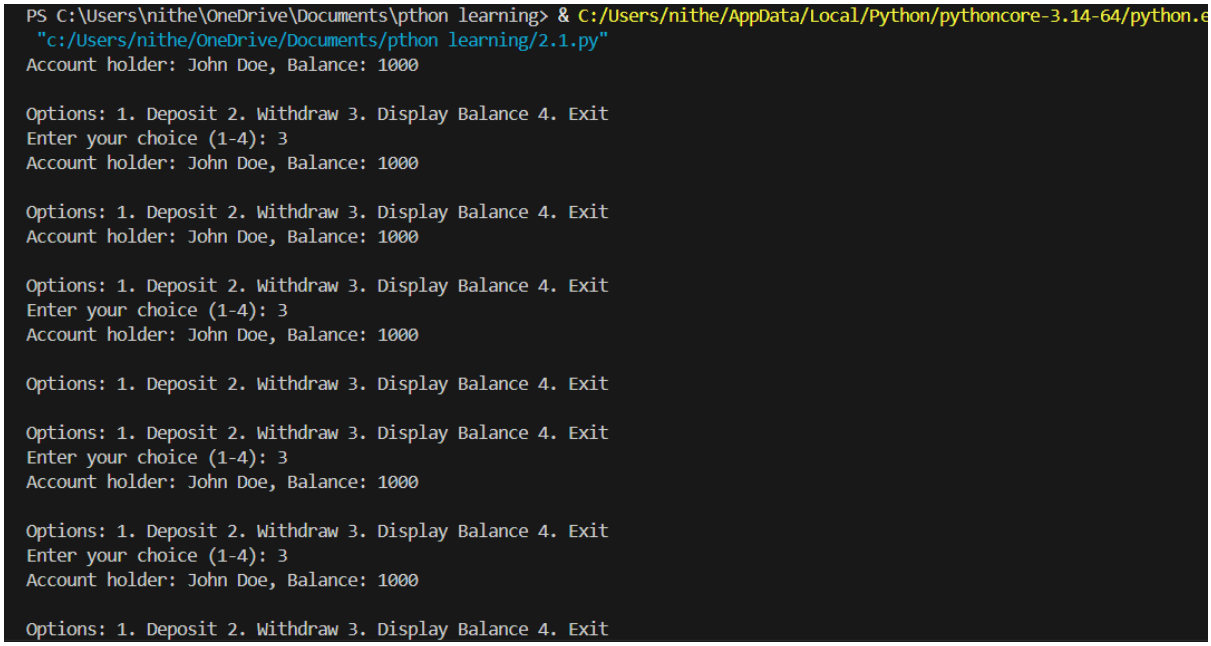
Ln 12, Col 54 Spaces: 4 UTF-8 CRLF

```python
# Example usage
account = BankAccount("John Doe", 1000)
account.display_balance()
while True:
    print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
    choice = input("Enter your choice (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(amount)
    elif choice == '3':
        account.display_balance()
    elif choice == '4':
        print("Exiting the program.")
        break
    else:
        print("Invalid choice. Please try again.")


```

Output:



Justfication:

The AI-generated bank account program integrates **classes, loops, and conditional statements** to model real-world banking operations.

Loops enable repeated user interactions, while conditionals ensure **valid decision-making** for deposits, withdrawals, and balance checks.