# AI ASSISTED CODING

# Assignment-6.1

HTNO:2303A510B4

Batch:14

Name:K.Nithin kumar

#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:

```
2.1.py        fib.py        fibo.py  ×

fibo.py > ...
    1    #Generate Python code to print all even numbers between 1 and N using a loop
    2    N = int(input("Enter a number N: "))
    3    for num in range(1, N + 1):
    4        if num % 2 == 0:
    5            print(num)
    6
    7
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

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"
Enter a number N: 8
2
4
6
8
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

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:

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
1    #Generate a Python class User that validates age and email using conditional statemets
2    class User:
3        def __init__(self, age, email):
4            self.age = age
5            self.email = email
6
7        def validate_age(self):
8            if isinstance(self.age, int) and self.age >= 18:
9                return "Valid age"
10           else:
11               return "Invalid age"
12
13       def validate_email(self):
14           if "@" in self.email and "." in self.email:
15               return "Valid email"
16           else:
17               return "Invalid email"
18   # Valid input
19   # Valid input
20   user1 = User(25, "user@example.com")
21   print(user1.validate_age())      # Valid age
22   print(user1.validate_email())  # Valid email
23
24   # Invalid input
25   user2 = User(15, "userexample.com")
26   print(user2.validate_age())      # Invalid age
27   print(user2.validate_email())  # Invalid email
28
29
```

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/fib.py"
Valid age
Valid email
Invalid age
Invalid email
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

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
1   #Generate a Python program for a simple bank account system using class,loops, and conditional statements
2   class BankAccount:
3       def __init__(self, account_holder, balance=0):
4           self.account_holder = account_holder
5           self.balance = balance
6
7       def deposit(self, amount):
8           if amount > 0:
9               self.balance += amount
10              print(f"Deposited: {amount}. New balance: {self.balance}")
11          else:
12              print("Deposit amount must be positive.")
13
14      def withdraw(self, amount):
15          if amount > 0:
16              if amount <= self.balance:
17                  self.balance -= amount
18                  print(f"Withdrew: {amount}. New balance: {self.balance}")
19              else:
20                  print("Insufficient funds.")
21          else:
22              print("Withdrawal amount must be positive.")
23
24      def display_balance(self):
25          print(f"Account holder: {self.account_holder}, Balance: {self.balance}")
26  # Example usage
27  account = BankAccount("John Doe", 1000)
28  account.display_balance()
29  while True:
30      print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
31      choice = input("Enter your choice (1-4): ")
32
33      if choice == '1':
34          amount = float(input("Enter amount to deposit: "))
35          account.deposit(amount)
36      elif choice == '2':
37          amount = float(input("Enter amount to withdraw: "))
```

```python
26   # Example usage
27   account = BankAccount("John Doe", 1000)
28   account.display_balance()
29   while True:
30       print("\nOptions: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit")
31       choice = input("Enter your choice (1-4): ")
32
33       if choice == '1':
34           amount = float(input("Enter amount to deposit: "))
35           account.deposit(amount)
36       elif choice == '2':
37           amount = float(input("Enter amount to withdraw: "))
38           account.withdraw(amount)
39       elif choice == '3':
40           account.display_balance()
41       elif choice == '4':
42           print("Exiting the program.")
43           break
44       else:
45           print("Invalid choice. Please try again.")
46
47
```

Output:

```
PS C:\Users\nithe\OneDrive\Documents\pthon learning> & C:/Users/nithe/AppData/Local/Python/pythoncore-3.14-64/python.e
 "c:/Users/nithe/OneDrive/Documents/pthon learning/2.1.py"
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Enter your choice (1-4): 3
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Enter your choice (1-4): 3
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Enter your choice (1-4): 3
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
Enter your choice (1-4): 3
Account holder: John Doe, Balance: 1000

Options: 1. Deposit 2. Withdraw 3. Display Balance 4. Exit
```

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.