

AI Assisted Coding Lab Ass-6.1

Name: K.Abbinav

Batch-13

2303A51856

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."

Expected Output:

- AI-generated loop logic.
- Identification of loop type used (for or while).
- Validation with sample inputs.

The screenshot shows a Jupyter Notebook interface with a sidebar for 'BLACKBOX' features. The main area displays a code cell titled 'AI-Generated Loop Logic' containing Python code to print even numbers. The code uses a for loop and includes validation for sample inputs (N=10, N=5, N=1). The output shows the generated code and its execution results.

```
def print_even_numbers(N):
    """
    Prints all even numbers between 1 and N using a for loop.
    """
    print(f"Even numbers between 1 and {N}:")
    for i in range(1, N + 1):
        if i % 2 == 0:
            print(i)

# ### Identification of Loop Type
# The AI-generated code uses a **for** loop**.

# ### Validation with Sample Inputs
# Sample Input 1: N = 10
print_even_numbers(10)

# Sample Input 2: N = 5
print_even_numbers(5)

# Sample Input 3: N = 1
print_even_numbers(1)
```

[2] ... Even numbers between 1 and 10:
2
4
6
8
10
Even numbers between 1 and 5:
2
4
Even numbers between 1 and 1:
1

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.”

Expected Output:

- AI-generated code using loop and if condition.
- Correct count validation.
- Explanation of logic flow.

```
> # Task: Generate Python code to count how many numbers in a list are even and odd.

# AI-generated code using loop and if condition.
def count_even_odd(numbers):
    """
    Counts the number of even and odd integers in a list.

    Args:
        numbers (list): A list of integers.

    Returns:
        tuple: A tuple containing (even_count, odd_count).
    """
    even_count = 0
    odd_count = 0
    for num in numbers:
        if num % 2 == 0:
            even_count += 1
        else:
            odd_count += 1
    return even_count, odd_count

# correct count validation.
print("#### Validation with Sample Inputs")

# Sample Input 1
my_list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even, odd = count_even_odd(my_list_1)
print(f"List: {my_list_1}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Sample Input 2
my_list_2 = [15, 22, 38, 41, 58]
even, odd = count_even_odd(my_list_2)
print(f"List: {my_list_2}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Sample Input 3
my_list_3 = []
even, odd = count_even_odd(my_list_3)
print(f"List: {my_list_3}")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Explanation of logic flow.
print("#### Explanation of logic Flow")
print("The 'count_even_odd' function works as follows:")
print("1. Initialization: 'even_count' and 'odd_count' are set to 0.")
print("2. Iteration: A 'for' loop goes through each number in the input list.")
print("3. Conditional Check: Inside the loop, 'if num % 2 == 0' checks if the number is even (remainder is 0 when divided by 2). If true, 'even_count' is incremented; otherwise, 'odd_count' is incremented.")
print("4. Return Value: After checking all numbers, the function returns both 'even_count' and 'odd_count'.")
```

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.”

Expected Output:

- AI-generated class with validation logic.
- Verification of condition handling.
- Test cases for valid and invalid inputs.

```

class Person:
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks

    def calculate_total_marks(self):
        total_marks = sum(self.marks)
        print(f"Total marks: {total_marks}")

    def calculate_average_marks(self):
        average_marks = sum(self.marks) / len(self.marks)
        print(f"Average marks: {average_marks}")

    def update_marks(self, new_marks):
        self.marks.append(new_marks)

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age

    def get_marks(self):
        return self.marks

    def set_name(self, new_name):
        self.name = new_name

    def set_age(self, new_age):
        self.age = new_age

    def set_marks(self, new_marks):
        self.marks = new_marks

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}, Marks: {self.marks}"


# Create a person object
person = Person("John Doe", 20, [85, 90, 78, 88, 92])

# Print details
print(person)

# Calculate total marks
person.calculate_total_marks()

# Calculate average marks
person.calculate_average_marks()

# Update marks
person.update_marks(95)

# Get name
name = person.get_name()
print(f"Name: {name}")

# Get age
age = person.get_age()
print(f"Age: {age}")

# Get marks
marks = person.get_marks()
print(f"Marks: {marks}")

# Set name
person.set_name("Jane Doe")

# Set age
person.set_age(22)

# Set marks
person.set_marks([90, 85, 80, 95, 88])

# Print updated details
print(person)

# Print class details
print(Person.__doc__)

```

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."

Expected Output:

- AI-generated class code.
- Verification of correctness and completeness of class structure.
- Minor manual improvements (if needed) with justification.

```

    <!-- Header -->
    <div>
        <div>Header</div>
        <div>Header</div>
        <div>Header</div>
    </div>

    <!-- Content -->
    <div>
        <div>Content</div>
        <div>Content</div>
        <div>Content</div>
    </div>

    <!-- Footer -->
    <div>
        <div>Footer</div>
        <div>Footer</div>
        <div>Footer</div>
    </div>

```

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.”

Expected Output:

- Complete AI-generated program.
 - Identification of strengths and limitations of AI suggestions.
 - Reflection on how AI assisted coding productivity.

```
% Generate + Code + Markdown | Run All - Select Region
> a simple Bank Accounts System Program

class BankAccount:
    def __init__(self, account_number, owner_name, initial_balance=0):
        if not isinstance(account_number, str) or not account_number.isdigit():
            raise ValueError("Account number must be a string containing only digits.")
        if not isinstance(owner_name, str) or not owner_name.strip():
            raise ValueError("Owner name must be a non-empty string.")
        if not isinstance(initial_balance, (int, float)) or initial_balance < 0:
            raise ValueError("Initial balance must be a non-negative number.")

        self.account_number = account_number
        self.owner_name = owner_name
        self.balance = initial_balance
        print(f"Account {self.account_number} created for {self.owner_name} with initial balance {self.balance:.2f}.")

    def deposit(self, amount):
        if not isinstance(amount, (int, float)) or amount < 0:
            print("Invalid deposit amount. Amount must be a positive number.")
            return False
        self.balance += amount
        print(f"Deposited {amount:.2f}. New balance: {self.balance:.2f}.")
        return True

    def withdraw(self, amount):
        if not isinstance(amount, (int, float)) or amount <= 0:
            print("Invalid withdrawal amount. Amount must be a positive number.")
            return False
        if amount > self.balance:
            print("Insufficient funds. Withdrawal denied.")
            return False
        self.balance -= amount
        print(f"Withdrew {amount:.2f}. New balance: {self.balance:.2f}.")
        return True

    def get_balance(self):
        return self.balance

    def __str__(self):
        return f"Account Number: {self.account_number}\nOwner: {self.owner_name}\nBalance: {self.balance:.2f}"


def run_bank_system():
    print("\nWelcome to Simple Bank Account System ---")
    account = None
    while account is None:
        try:
            ac_num = input("Enter new account number (digits only): ")
            owner = input("Enter account owner name: ")
            initial_bal_str = input("Enter initial balance (optional, default 0): ")
            initial_bal = float(initial_bal_str) if initial_bal_str else 0.0
            account = BankAccount(ac_num, owner, initial_bal)
        except ValueError as e:
            print(f"Error creating account: {e}\nPlease try again.")
        except Exception as e:
            print(f"An unexpected error occurred: {e}\nPlease try again.")

    while True:
        print("---- Menu ----")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Account Details")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            try:
                amount = float(input("Enter amount to deposit: "))
                account.deposit(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
        elif choice == '2':
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")



```

```
--- Welcome to Simple Bank Account System ---
Enter new account number (digits only): 6757
Enter account owner name: gg
Enter initial balance (optional, default 0):
Account 6757 created for gg with initial balance 0.00.

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 1
Enter amount to deposit: 6666
Deposited 6666.00. New balance: 6666.00.

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 3
Current Balance: $6666.00

--- Menu ---
1. Deposit
2. Withdraw
3. Check Balance
4. Account Details
5. Exit
Enter your choice: 5
Thank you for using our bank system. Goodbye!
```