

AI Assisted Coding Lab ass-6.1

Name:M.Mohan venkatesh

Batch:14

2303A510f0

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

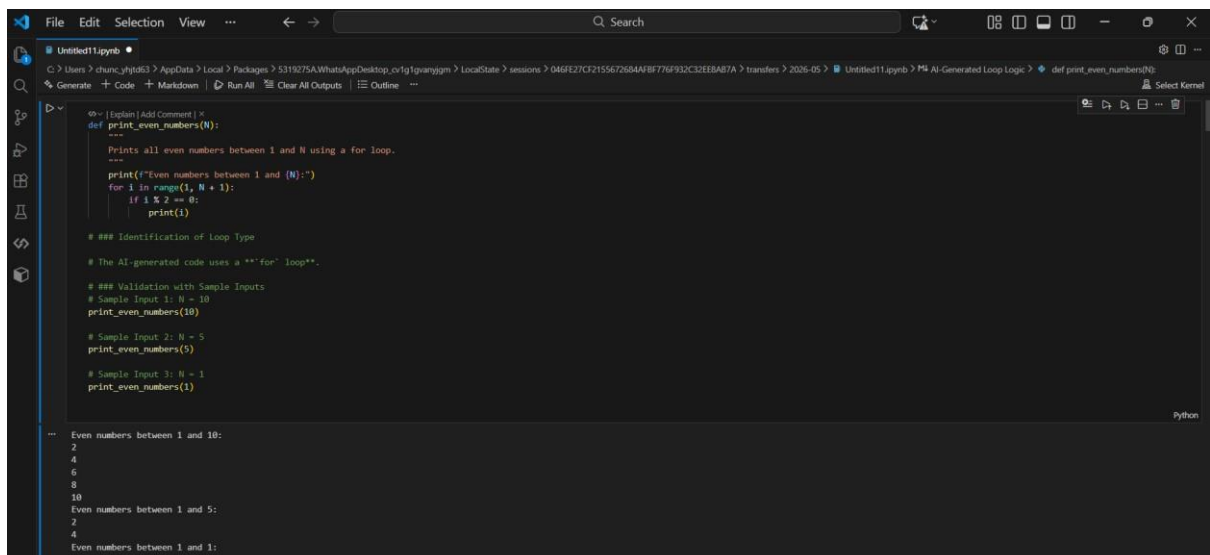
Task: Use an AI code completion tool to generate a loop-based program.

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.



```
File Edit Selection View ... Search
Untitled1.ipynb
C:\Users\chun...> AppData\Local\Packages> 5319275A...> LocalState> sessions> 046fE27CF2155672684AFBF776F32C32EBAE7A> transfers> 2025-05>
Generate Code Markdown Run All Clear All Outputs Outline
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)

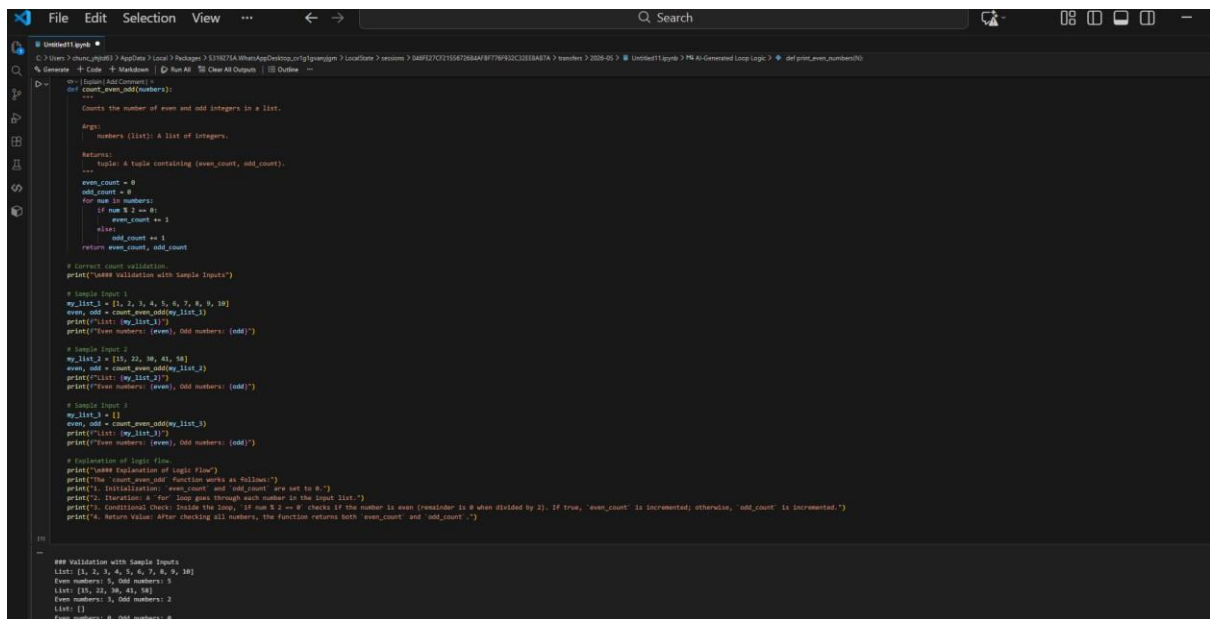
Even numbers between 1 and 10:
2
4
6
8
10
Even numbers between 1 and 5:
2
4
Even numbers between 1 and 1:
```

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.”



```
File Edit Selection View ... Search
# CountEvenOdd.py
C:\Users> python .\CountEvenOdd.py
Generate + Code + Markdown Run All Clear All Outputs Outline
D:\...> python .\CountEvenOdd.py
...
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, 54]
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.")

## Validation with Sample Inputs
list1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even numbers: 5, Odd numbers: 5
list2: [15, 22, 38, 41, 54]
Even numbers: 3, Odd numbers: 2
list3: []
Even numbers: 0, Odd numbers: 0
```

Expected Output:

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

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.”

“Generate a Python class Student with attributes (name, roll number, marks) and methods to calculate total and average marks.”

```
class Student:
    """A Python class representing a student with attributes: name, roll number, and marks.
    It includes methods to calculate total and average marks, and a string representation of the student object.
    """

    def __init__(self, name: str, roll_number: int, marks: list[int]):
        """Initialize a Student object with name, roll number, and marks.

        Args:
            name (str): The name of the student.
            roll_number (int): The roll number of the student.
            marks (list[int]): A list of marks for different subjects.

        Raises:
            ValueError: If marks is not a list of integers.
        """
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

        # Validate marks
        if not isinstance(marks, list) or not all(isinstance(mark, int) for mark in marks):
            raise ValueError("Marks must be a list of integers.")

    def calculate_total_marks(self) -> int:
        """Calculate the total marks of the student.

        Returns:
            int: The sum of all marks. Returns 0 if no marks.
        """
        return sum(self.marks)

    def calculate_average_marks(self) -> float:
        """Calculate the average marks of the student.

        Returns:
            float: The average of all marks. Returns 0.0 if no marks.
        """
        if not self.marks:
            return 0.0
        return self.calculate_total_marks() / len(self.marks)

    def __str__(self) -> str:
        """Return a string representation of the Student object.

        Returns:
            str: A string containing name, roll number, and average marks.
        """
        return f"Student: {self.name}, Roll No: {self.roll_number}, Average Marks: {self.calculate_average_marks():.2f}"

# Verification with Sample Student Data
student1 = Student("Alice", 101, [85, 78, 92])
print(f"Total Marks for {student1.name}: {student1.calculate_total_marks()}")
print(f"Average Marks for {student1.name}: {student1.calculate_average_marks():.2f}")

student2 = Student("Bob", 102, [72, 88, 75])
print(f"Total Marks for {student2.name}: {student2.calculate_total_marks()}")
print(f"Average Marks for {student2.name}: {student2.calculate_average_marks():.2f}")

student3 = Student("Charlie", 103, [65, 70, 60])
print(f"Total Marks for {student3.name}: {student3.calculate_total_marks()}")
print(f"Average Marks for {student3.name}: {student3.calculate_average_marks():.2f}")
```

```
class Student:
    """A Python class representing a student with attributes: name, roll number, and marks.
    It includes methods to calculate total and average marks, and a string representation of the student object.
    """

    def __init__(self, name: str, roll_number: int, marks: list[int]):
        """Initialize a Student object with name, roll number, and marks.

        Args:
            name (str): The name of the student.
            roll_number (int): The roll number of the student.
            marks (list[int]): A list of marks for different subjects.

        Raises:
            ValueError: If marks is not a list of integers.
        """
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

        # Validate marks
        if not isinstance(marks, list) or not all(isinstance(mark, int) for mark in marks):
            raise ValueError("Marks must be a list of integers.")

    def calculate_total_marks(self) -> int:
        """Calculate the total marks of the student.

        Returns:
            int: The sum of all marks. Returns 0 if no marks.
        """
        return sum(self.marks)

    def calculate_average_marks(self) -> float:
        """Calculate the average marks of the student.

        Returns:
            float: The average of all marks. Returns 0.0 if no marks.
        """
        if not self.marks:
            return 0.0
        return self.calculate_total_marks() / len(self.marks)

    def __str__(self) -> str:
        """Return a string representation of the Student object.

        Returns:
            str: A string containing name, roll number, and average marks.
        """
        return f"Student: {self.name}, Roll No: {self.roll_number}, Average Marks: {self.calculate_average_marks():.2f}"

# Verification with Sample Student Data
student1 = Student("Alice", 101, [85, 78, 92])
print(f"Total Marks for {student1.name}: {student1.calculate_total_marks()}")
print(f"Average Marks for {student1.name}: {student1.calculate_average_marks():.2f}")

student2 = Student("Bob", 102, [72, 88, 75])
print(f"Total Marks for {student2.name}: {student2.calculate_total_marks()}")
print(f"Average Marks for {student2.name}: {student2.calculate_average_marks():.2f}")

student3 = Student("Charlie", 103, [65, 70, 60])
print(f"Total Marks for {student3.name}: {student3.calculate_total_marks()}")
print(f"Average Marks for {student3.name}: {student3.calculate_average_marks():.2f}")
```

Expected Output:

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

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.”

```
# Generated Python Code
"""
Simple Bank Account System
"""

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 consisting only digits.")
        if not isinstance(owner_name, str) or not owner_name.strip():
            raise ValueError("Owner name cannot be empty.")
        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} | Owner: {self.owner_name} | Balance: ${self.balance:.2f}"

def run_bank_system():
    print("Welcome to Simple Bank Account System")
    account = None
    while account is None:
        try:
            acc_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(acc_num, owner, initial_bal)
        except ValueError as e:
            print(f"Invalid input: {e}")
        except Exception as e:
            print(f"Unexpected error occurred: {e}")

    while True:
        print("\n--- 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.")
            except Exception as e:
                print(f"Unexpected error: {e}")
        elif choice == "2":
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
            except Exception as e:
                print(f"Unexpected error: {e}")
        elif choice == "3":
            print(f"Current Balance: {account.get_balance():.2f}")
        elif choice == "4":
            print(account)
        elif choice == "5":
            print("Thank you for using our bank system. Goodbye!")
            break
        else:
            print("Invalid choice. Please select a valid option (1-5).")

if __name__ == "__main__":
    run_bank_system()
```

```
"""
Simple Bank Account System
"""

def run_bank_system():
    print("Welcome to Simple Bank Account System")
    account = None
    while account is None:
        try:
            acc_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(acc_num, owner, initial_bal)
        except ValueError as e:
            print(f"Invalid input: {e}")
        except Exception as e:
            print(f"Unexpected error occurred: {e}")

    while True:
        print("\n--- 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.")
            except Exception as e:
                print(f"Unexpected error: {e}")
        elif choice == "2":
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
            except Exception as e:
                print(f"Unexpected error: {e}")
        elif choice == "3":
            print(f"Current Balance: {account.get_balance():.2f}")
        elif choice == "4":
            print(account)
        elif choice == "5":
            print("Thank you for using our bank system. Goodbye!")
            break
        else:
            print("Invalid choice. Please select a valid option (1-5).")

if __name__ == "__main__":
    run_bank_system()
```

```
--- 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!
```

Expected Output:

- Complete AI-generated program.

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