

AI Assisted Coding Lab ass-6.1

Name: CH. Arush Nandhan

Batch:14

2303A51880

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

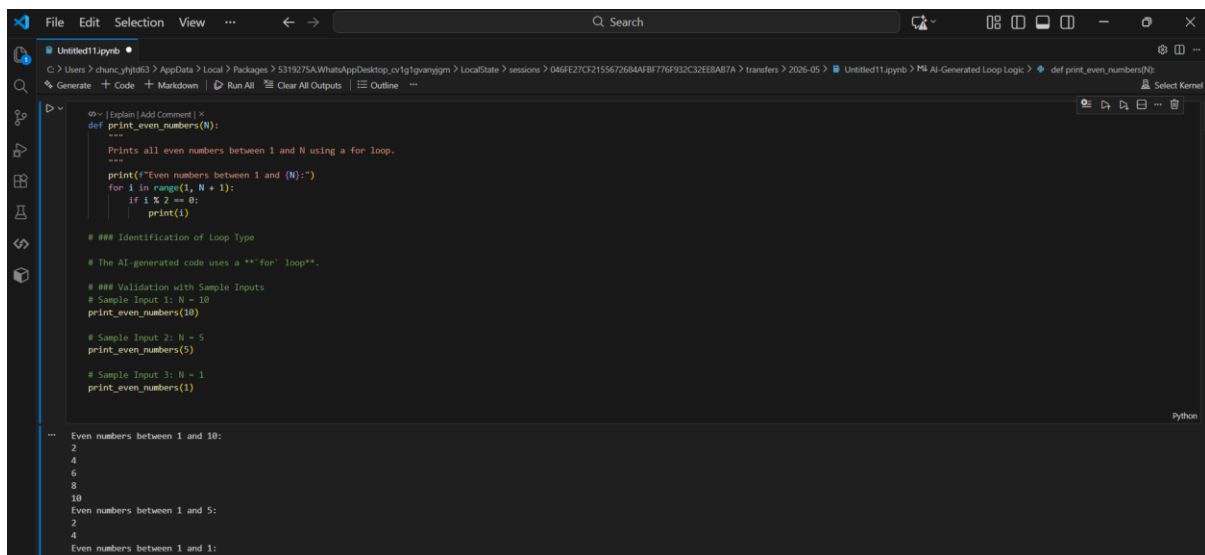
Task: Use an AI code completion tool to generate a loop-based program.

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.



```
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)
```

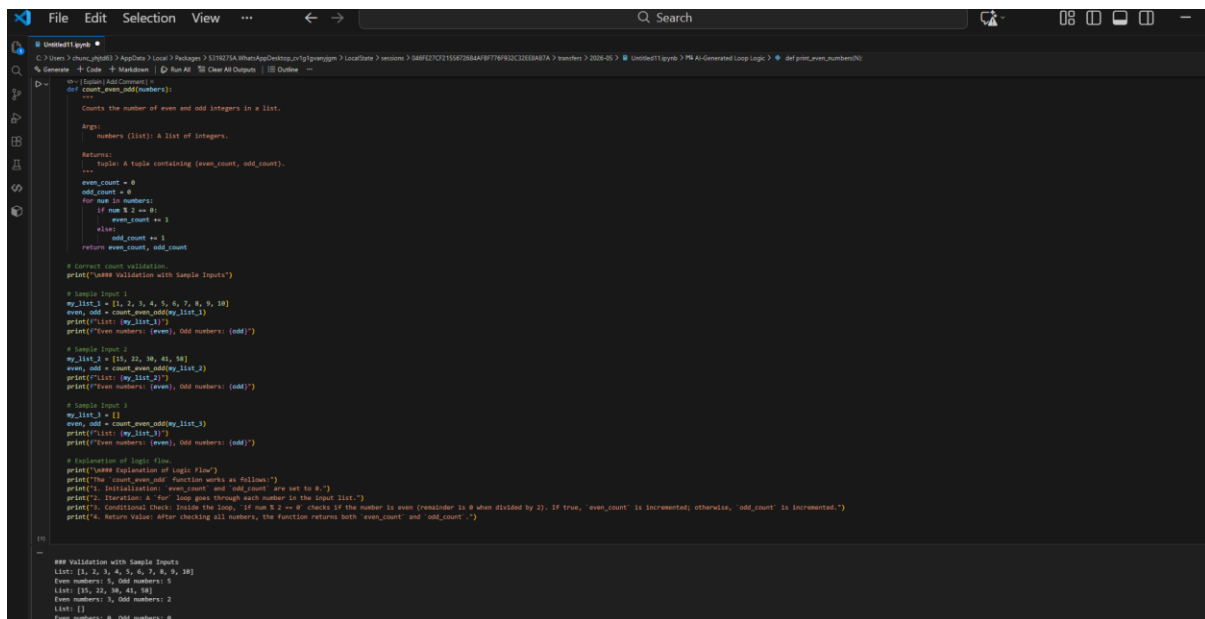
Even numbers between 1 and 10:
2
4
6
8
10
Even numbers between 1 and 5:
2
4
Even numbers between 1 and 1:

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
C:\Users\chun... > AppData\Local\Programs\Python\Python39\Scripts > LocalState > Sessions > 5d8f2f0211567285aaf9778f920c32e3a37a > Session > 2026-05 > Python AI-Generated Loop Logic > def print_even_numbers()

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 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
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, 30, 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, it 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
List: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
Even numbers: 10, Odd numbers: 0
List: [15, 22, 30, 41, 58]
Even numbers: 3, Odd numbers: 2
List: []
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.”


```
# Student class for student management
class Student:
    """
    A class designed to manage student details and provide basic mark calculations.
    """
    def __init__(self, name, roll_number, marks):
        """
        Initialize the Student object with name, roll number, and marks.
        """
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def calculate_total_marks(self):
        """
        Calculate the total marks of the student.
        """
        total_marks = sum(self.marks)
        return total_marks

    def calculate_average_marks(self):
        """
        Calculate the average marks of the student.
        """
        total_marks = self.calculate_total_marks()
        average_marks = total_marks / len(self.marks)
        return average_marks

    def __str__(self):
        """
        Return a string representation of the Student object.
        """
        return f"Student Name: {self.name}, Roll No: {self.roll_number}, Marks: {self.marks}, Total Marks: {self.calculate_total_marks()}, Average Marks: {self.calculate_average_marks()}"

# Verification with Sample Student Data ...
student = Student("John", "1001", [85, 78, 92])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 1: Student with multiple marks
student = Student("Alice", "1002", [75, 88, 90, 82])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 2: Student with a single mark
student = Student("Bob", "1003", [70])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 3: Student with no marks (empty list)
student = Student("Charlie", "1004", [])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")
```

```
# Student class for student management
class Student:
    """
    A class designed to manage student details and provide basic mark calculations.
    """
    def __init__(self, name, roll_number, marks):
        """
        Initialize the Student object with name, roll number, and marks.
        """
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def calculate_total_marks(self):
        """
        Calculate the total marks of the student.
        """
        total_marks = sum(self.marks)
        return total_marks

    def calculate_average_marks(self):
        """
        Calculate the average marks of the student.
        """
        total_marks = self.calculate_total_marks()
        average_marks = total_marks / len(self.marks)
        return average_marks

    def __str__(self):
        """
        Return a string representation of the Student object.
        """
        return f"Student Name: {self.name}, Roll No: {self.roll_number}, Marks: {self.marks}, Total Marks: {self.calculate_total_marks()}, Average Marks: {self.calculate_average_marks()}"

# Verification with Sample Student Data ...
student = Student("John", "1001", [85, 78, 92])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 1: Student with multiple marks
student = Student("Alice", "1002", [75, 88, 90, 82])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 2: Student with a single mark
student = Student("Bob", "1003", [70])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")

# Test Case 3: Student with no marks (empty list)
student = Student("Charlie", "1004", [])
print(f"Student Name: {student.name}, Roll No: {student.roll_number}, Marks: {student.marks}, Total Marks: {student.calculate_total_marks()}, Average Marks: {student.calculate_average_marks()}")
```

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 Program
# Python AI-Generated Bank Account 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 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}.")

    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}. New balance: {self.balance}.")
        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}. New balance: {self.balance}.")
        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}"

def run_bank_system():
    print("\n--- 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"Error: {e}")
        except KeyboardInterrupt:
            print("Program interrupted. Exiting.")
            return

    # Main loop
    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.")
        elif choice == "2":
            try:
                amount = float(input("Enter amount to withdraw: "))
                account.withdraw(amount)
            except ValueError:
                print("Invalid input. Please enter a numerical amount.")
        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).")

    # Run the program
    run_bank_system()

# Identification of Strengths and Limitations of AI Suggestions
def main():
    # Strengths
    1. "Rapid Prototyping": The AI quickly generated a functional base for a bank account system, saving significant initial development time.
    2. "Correct Structure": It correctly used a class to encapsulate account logic, loops for interactive menus, and conditionals for transaction validation and menu navigation.
    3. "Basic Validation": The generated code included basic input validation (e.g., positive deposit/withdrawal amounts, sufficient balance, non-empty owner name, digit-only account number) which is crucial for robust applications.
    4. "Clear Naming Conventions": Methods like deposit, withdraw, and get_balance are well-defined and follow good object-oriented principles.
    5. "Interactive Loop": The while True loop for the menu provides a good user experience for interacting with the system.

    # Limitations
    1. "Lack of Persistence": The system lacks any form of data persistence (e.g., saving accounts to a file or database). All data is lost when the program ends.
    2. "Single Account Management": The program only allows managing one account at a time. A real system would need to manage multiple accounts, perhaps using a list or dictionary of BankAccount objects.
    3. "Security": No security measures (e.g., password, PIN) are implemented for transactions or account access.
    4. "Error Handling Limitations": While basic validation is present, more robust error handling (e.g., specific error codes, custom exceptions for different types of failures) could be implemented.
    5. "User Experience (UX) Enhancement": The text-based interface is functional but basic. A more user-friendly interface might involve clearing the screen or providing more detailed feedback.
    6. "Edge Cases": While some validation is present, more comprehensive checks for edge cases (e.g., very large numbers, specific formatting requirements for account numbers) could be added.
```

```
except KeyboardInterrupt:
    print("Program interrupted. Exiting.")
    return

# Main loop
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.")
    elif choice == "2":
        try:
            amount = float(input("Enter amount to withdraw: "))
            account.withdraw(amount)
        except ValueError:
            print("Invalid input. Please enter a numerical amount.")
    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).")

# Run the program
run_bank_system()

# Identification of Strengths and Limitations of AI Suggestions
def main():
    # Strengths
    1. "Rapid Prototyping": The AI quickly generated a functional base for a bank account system, saving significant initial development time.
    2. "Correct Structure": It correctly used a class to encapsulate account logic, loops for interactive menus, and conditionals for transaction validation and menu navigation.
    3. "Basic Validation": The generated code included basic input validation (e.g., positive deposit/withdrawal amounts, sufficient balance, non-empty owner name, digit-only account number) which is crucial for robust applications.
    4. "Clear Naming Conventions": Methods like deposit, withdraw, and get_balance are well-defined and follow good object-oriented principles.
    5. "Interactive Loop": The while True loop for the menu provides a good user experience for interacting with the system.

    # Limitations
    1. "Lack of Persistence": The system lacks any form of data persistence (e.g., saving accounts to a file or database). All data is lost when the program ends.
    2. "Single Account Management": The program only allows managing one account at a time. A real system would need to manage multiple accounts, perhaps using a list or dictionary of BankAccount objects.
    3. "Security": No security measures (e.g., password, PIN) are implemented for transactions or account access.
    4. "Error Handling Limitations": While basic validation is present, more robust error handling (e.g., specific error codes, custom exceptions for different types of failures) could be implemented.
    5. "User Experience (UX) Enhancement": The text-based interface is functional but basic. A more user-friendly interface might involve clearing the screen or providing more detailed feedback.
    6. "Edge Cases": While some validation is present, more comprehensive checks for edge cases (e.g., very large numbers, specific formatting requirements for account numbers) could be added.
```

```
--- 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.