

# AI Assisted Coding Lab Ass-6.1

Name: K.MAHESH

Batch-14

2303A510H1

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 the BLACKBOX AI interface. On the left, there's a sidebar with various icons and sections like 'New in v3.5', 'Free Unlimited Agent Requests', 'Open Source BLACKBOX CLI', 'Guide | GitHub', 'Remote Agents', and a 'Learn more' button. In the center, a code editor window titled 'Untitled11.pynb' displays the following Python code:

```
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)
```

Below the code, the terminal output shows the execution of the function for different values of N:

```
[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:
```

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("List: (my_list_1)")
print(f"Even numbers: {even}, Odd numbers: {odd}")

# Sample Input 2
my_list_2 = [15, 22, 38, 45, 58]
even, odd = count_even_odd(my_list_2)
print("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("List: (my_list_3)")
print(f"Even numbers: {even}, Odd numbers: {odd}")

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

## Validation with Sample Inputs
list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even: 5, odd: 5
List: (my_list_1)
Even numbers: 5, Odd numbers: 5
list: [15, 22, 38, 45, 58]
even: 3, odd: 2
List: (my_list_2)
Even numbers: 3, Odd numbers: 2
list: []
even: 0, odd: 0
List: (my_list_3)
Even numbers: 0, Odd numbers: 0

# Explanation of logic flow
The 'count_even_odd' function works as follows:
1. Initialization: 'even_count' and 'odd_count' are set to 0.
2. Iteration: A 'for' loop goes through each number in the input list.
3. Logic: Inside the loop, it checks if the number is even ('num % 2 == 0'). If true, 'even_count' is incremented; otherwise, 'odd_count' is incremented.
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.

```

// Validation code for the User class
public class UserValidation {
    private static final String NAME_REGEX = "^[a-zA-Z][a-zA-Z\\s]{1,20}$";
    private static final String AGE_REGEX = "^(1[6-9]|[2-9]\\d)$";
    private static final String WEIGHT_REGEX = "^(1[6-9]|[2-9]\\d)\\.\\d{1,2}$";

    public void validateUser(User user) {
        validateName(user.getName());
        validateAge(user.getAge());
        validateWeight(user.getWeight());
    }

    private void validateName(String name) {
        if (!name.matches(NAME_REGEX)) {
            throw new ValidationException("Name must be a string");
        }
    }

    private void validateAge(Integer age) {
        if (!age.matches(AGE_REGEX)) {
            throw new ValidationException("Age must be an integer");
        }
    }

    private void validateWeight(Double weight) {
        if (!weight.matches(WEIGHT_REGEX)) {
            throw new ValidationException("Weight must be a positive number");
        }
    }
}

// Test code for the UserValidation class
public class UserValidationTest {
    @Test
    public void testValidateUser() {
        User user = new User("John Doe", 30, 75.5);
        UserValidation validation = new UserValidation();
        validation.validateUser(user);

        // Check if no exception was thrown
        verifyNoExceptionWasThrown();
    }

    @Test
    public void testValidateName() {
        UserValidation validation = new UserValidation();
        validation.validateName("John Doe");

        // Check if no exception was thrown
        verifyNoExceptionWasThrown();
    }

    @Test
    public void testValidateAge() {
        UserValidation validation = new UserValidation();
        validation.validateAge(30);

        // Check if no exception was thrown
        verifyNoExceptionWasThrown();
    }

    @Test
    public void testValidateWeight() {
        UserValidation validation = new UserValidation();
        validation.validateWeight(75.5);

        // Check if no exception was thrown
        verifyNoExceptionWasThrown();
    }

    private void verifyNoExceptionWasThrown() {
        // Implementation of verifyNoExceptionWasThrown()
    }
}

```

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.

```
/* An additional class from the Student Manager */
class BankAccount:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount} into account for {self.name}.")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew {amount} from account for {self.name}.")
        else:
            print("Withdrawal amount must be positive and less than or equal to the current balance.")

    def get_balance(self):
        return self.balance

    def __str__(self):
        return f"Account for {self.name}, Balance: ${self.balance:.2f}"

# Create accounts
account1 = BankAccount("Alice", 100)
account2 = BankAccount("Bob", 200)

# Deposit and withdrawal
account1.deposit(50)
account2.withdraw(100)

# Check balance
print(account1.get_balance())
print(account2.get_balance())

# Print account details
print(account1)
print(account2)

# Transfer funds
def transfer(amount, source, target):
    source.withdraw(amount)
    target.deposit(amount)

transfer(50, account1, account2)

# Print final balances
print(account1.get_balance())
print(account2.get_balance())
```

#### 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 ...
> # AI-Generated Bank Account System Program

class BankAccount:
    def __init__(self, account_number, owner_name, initial_balance=0.0):
        if not isinstance(account_number, int) or len(str(account_number)) > 10:
            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, 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:{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"(Withdraw {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("... Welcome to Simple Bank Account System ...")
    account = None
    while account is None:
        try:
            acc_num = input("Enter new account number (digits only):")
            name = input("Enter 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, name, 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("\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(f"(Account {acc_num}) details: {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).")

    print("... Exiting ...")
    print("Bye!")

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

```
New in v1.5
Free Unfinished Requests
Use the BLACKBOX AI for an
unfinished request. It provides
the code needed to help
you finish it. You can then
build products better, faster, and
with more confidence.
Open Source BLACKBOX CLI
https://blackbox.ai
BLACKBOX AI directly to your
terminal so you can get
them done faster.
Learn more

Upgrade to PRO
Already
subscribed? Connect

% Generate + Code + Markdown | Run All ...
> # AI-Generated Bank Account System Program

class BankAccount:
    def __init__(self, account_number, owner_name, initial_balance=0.0):
        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):
        self.balance += amount
        print(f"(Deposited amount:{amount:.2f}). New balance: {self.balance:.2f}.")
        return True

    def withdraw(self, amount):
        if self.balance - amount < 0:
            print("Insufficient funds. Withdrawal denied.")
            return False
        self.balance -= amount
        print(f"(Withdraw {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("... Welcome to Simple Bank Account System ...")
    account = None
    while account is None:
        try:
            acc_num = input("Enter new account number: ")
            name = input("Enter owner name: ")
            initial_bal = float(input("Enter initial balance (optional, default 0): "))
            account = BankAccount(acc_num, name, 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("\n... Menu ...")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Account Detail")
        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(f"(Account {acc_num}) details: {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).")

    print("... Exiting ...")
    print("Bye!")

if __name__ == "__main__":
    run_bank_system()

# === Identification of Strengths and Limitations of AI Suggestions

# Strengths:
# 1. **Rapid Prototyping**: The AI quickly generated a functional base for a bank account system, saving significant initial development time.
# 2. **Basic Validation**: The generated code includes basic validation logic (e.g., checking for empty strings, ensuring account numbers are integers).
# 3. **Basic Validation**: The generated code included basic input validation (e.g., positive deposit/withdraw amounts, sufficient balance, non-empty owner name, digit-only account number) which is crucial for robust applications.
# 4. **Clear Method Separation**: Methods like `deposit`, `withdraw`, and `get_balance` are well-defined and follow good object-oriented principles.
# 5. **Code Reuse**: The AI leveraged existing code for common operations like `deposit` and `withdraw` across different methods.

# Limitations:
# 1. **Limited 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. **Simple 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. **No Error Handling**: There is no explicit error handling for edge cases like invalid input or insufficient funds.
# 4. **Error Handling Sophistication**: 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 Interaction**: The AI did not generate a graphical user interface. Instead of spending time on initial coding, I could have more easily focus on identifying areas for improvement, adding advanced features (like data persistence of multiple accounts), and refining the existing logic. This shifts the effort from creation to enhancement.
# 6. **Code Style**: While the generated code is functional, it may not be the most elegant or readable. Some AI-generated code can be verbose and lack readability.

# Reflection on How AI Assisted Coding Productivity
# AI significantly boosts coding productivity by acting as a powerful co-pilot. For this task:
# 1. **Reduced Boilerplate**: The AI eliminated the need to write the basic class structure, method definitions, and initial validation from scratch. This is often the most time-consuming and repetitive part of starting a new module.
# 2. **Conceptualization to Code**: It translated a high-level prompt ("Bank account system with class, loops, conditional") directly into working code, bridging the gap between idea and implementation very quickly.
# 3. **Code Generation and Best Practices**: The AI generated code that follows best practices for object-oriented programming. It implicitly guides towards common design patterns.
# 4. **Focus on Enhancements**: Instead of spending time on initial coding, I could have more easily focus on identifying areas for improvement, adding advanced features (like data persistence of multiple accounts), and refining the existing logic. This shifts the effort from creation to enhancement.
# 5. **Debugging Reduction**: The initial code is generally free of syntax errors and common logical pitfalls, reducing the time spent on early-stage debugging. Any issues are usually conceptual or related to missing features rather than fundamental code errors.

# Overall, AI didn't just write code; it provided a high-quality foundation that accelerated the entire development cycle, allowing for more strategic thinking and less tactical coding.
```

```
---- 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!
```