

# AI ASSISTED CODING

## ASSIGNMENT-6.1

N. Madhuvani

2303A52117

Batch-40

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.”

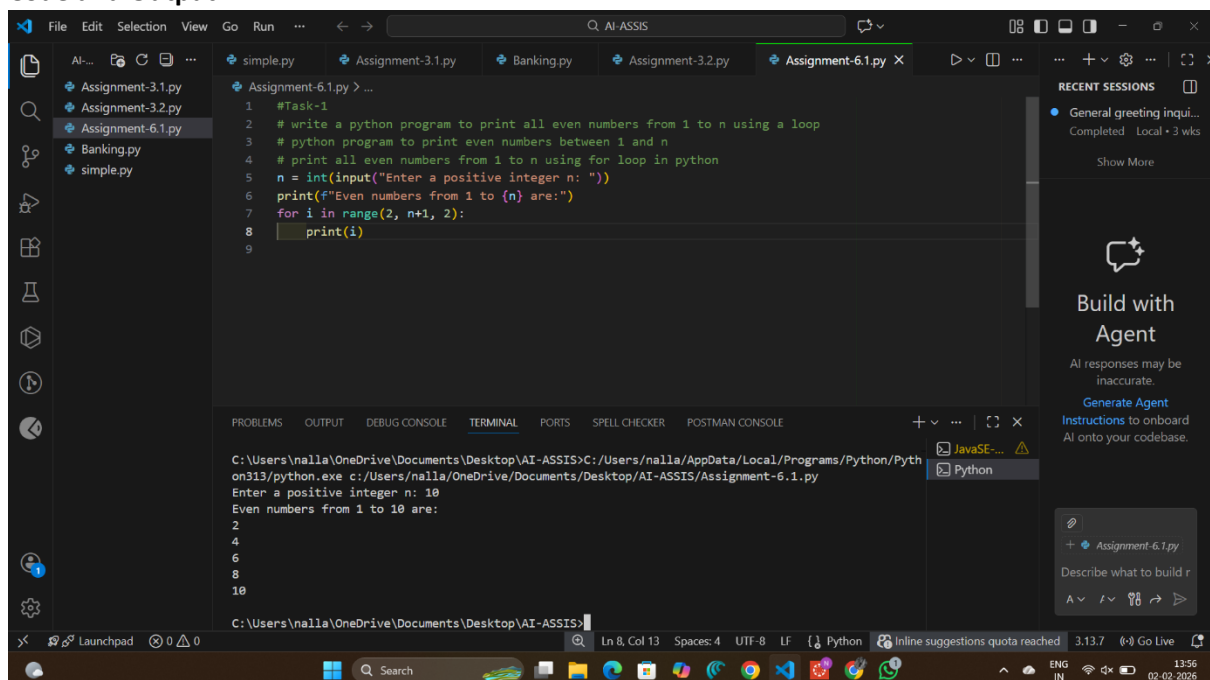
Expected Output:

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

Prompt:

# write a python program to print all even numbers from 1 to n using a loop

Code and Output:



The screenshot shows a code editor with a file explorer on the left, a main code area, and a terminal at the bottom. The file explorer lists files: Assignment-3.1.py, Assignment-3.2.py, Assignment-6.1.py, Banking.py, and simple.py. The main code area shows the following Python code:

```
1 #Task-1
2 # write a python program to print all even numbers from 1 to n using a loop
3 # python program to print even numbers between 1 and n
4 # print all even numbers from 1 to n using for loop in python
5 n = int(input("Enter a positive integer n: "))
6 print(f"Even numbers from 1 to {n} are:")
7 for i in range(2, n+1, 2):
8     print(i)
9
```

The terminal at the bottom shows the execution of the program:

```
C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>C:/Users/nalla/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-6.1.py
Enter a positive integer n: 10
Even numbers from 1 to 10 are:
2
4
6
8
10
```

The status bar at the bottom indicates the file is at line 8, column 13, with 4 spaces, UTF-8 encoding, and Python 3.13.7. The system tray shows the date and time as 02-02-2026, 13:56.

Analysis:

1. The program takes a positive integer n as input from the user.
2. A for loop is used to iterate from 2 to n.
3. The loop increments by 2, so only even numbers are printed.
4. This approach avoids unnecessary conditional checks.
5. The program demonstrates the use of loops and user input.

## 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.

**Prompt:**

# write a python program to count even and odd numbers in a list

# program to count how many numbers in a list are even and odd

**Code and Output:**

```
#Task-2
# write a python program to count even and odd numbers in a list
# program to count how many numbers in a list are even and odd
nums = list(map(int, input("Enter numbers: ").split()))
even_count = 0
odd_count = 0
for num in nums:
    if num % 2 == 0:
        even_count += 1
    else:
        odd_count += 1
print(f"Even numbers count: {even_count}")
print(f"Odd numbers count: {odd_count}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER POSTMAN CONSOLE

```
C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>C:\Users\nalla\AppData\Local\Programs\Python\Python313\python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-6.1.py
Enter numbers: 10 20 33 47 56 90
Even numbers count: 4
Odd numbers count: 2
C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>
```

RECENT SESSIONS

- General greeting inqui...  
Completed Local • 3 wks  
Show More

Build with Agent

AI responses may be inaccurate.

[Generate Agent](#)  
Instructions to onboard AI onto your codebase.

+ Assignment-6.1.py  
Describe what to build r  
A v f v i i →

Ln 15, Col 15 Spaces: 4 UTF-8 LF Python Online suggestions quota reached 3.13.7 Go Live 1401 02-02-2025

**Analysis:**

1. The program accepts multiple numbers from the user as a list.
2. Two counters are initialized to count even and odd numbers.
3. A for loop iterates through each number in the list.
4. The modulus operator (%) is used to check even or odd.
5. The final count of even and odd numbers is displayed.

**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.

**Prompt:**

#create a python class named User with attributes name, age, and email

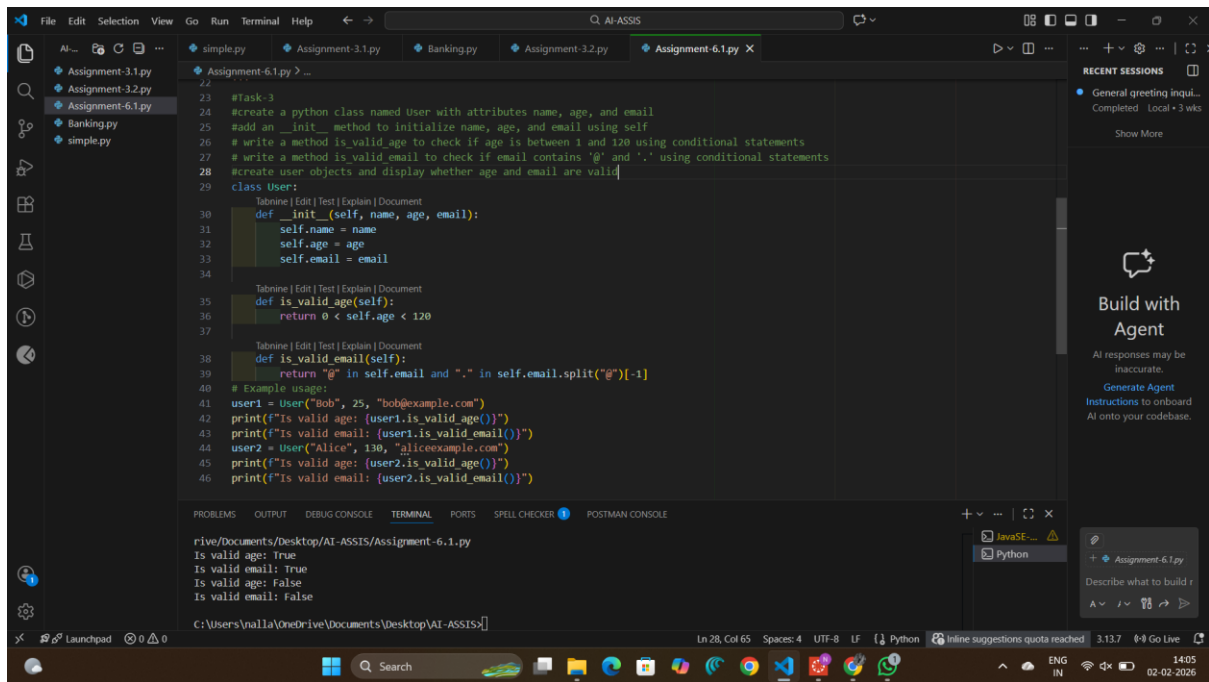
#add an \_\_init\_\_ method to initialize name, age, and email using self

# write a method is\_valid\_age to check if age is between 1 and 120 using conditional statements

# write a method is\_valid\_email to check if email contains '@' and '.' using conditional statements

#create user objects and display whether age and email are valid

**Code and Output:**



```
23 #Task-3
24 #create a python class named User with attributes name, age, and email
25 #add an __init__ method to initialize name, age, and email using self
26 # write a method is_valid_age to check if age is between 1 and 120 using conditional statements
27 # write a method is_valid_email to check if email contains '@' and '.' using conditional statements
28 #create user objects and display whether age and email are valid
29 class User:
30     def __init__(self, name, age, email):
31         self.name = name
32         self.age = age
33         self.email = email
34
35     def is_valid_age(self):
36         return 0 < self.age < 120
37
38     def is_valid_email(self):
39         return "@" in self.email and "." in self.email.split("@")[-1]
40
41 # Example usage:
42 user1 = User("Bob", 25, "bob@example.com")
43 print(f"Is valid age: {user1.is_valid_age()}")
44 print(f"Is valid email: {user1.is_valid_email()}")
45 user2 = User("Alice", 130, "alice@example.com")
46 print(f"Is valid age: {user2.is_valid_age()}")
47 print(f"Is valid email: {user2.is_valid_email()}")
```

Terminal Output:

```
rive/Documents/Desktop/AI-ASSIS/Assignment-6.1.py
Is valid age: True
Is valid email: True
Is valid age: False
Is valid email: False
```

## Analysis:

1. A User class is created with name, age, and email attributes.
2. The constructor initializes user details using the self keyword.
3. Age validation checks whether the age lies between 1 and 120.
4. Email validation checks for the presence of @ and ..
5. Objects are created to test valid and invalid user data.

## 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.

Prompt:

#write a python program and create a python class named Student with attributes name, roll\_number, and marks

#add an \_\_init\_\_ method to initialize name, roll\_number, and marks (marks should be a list)

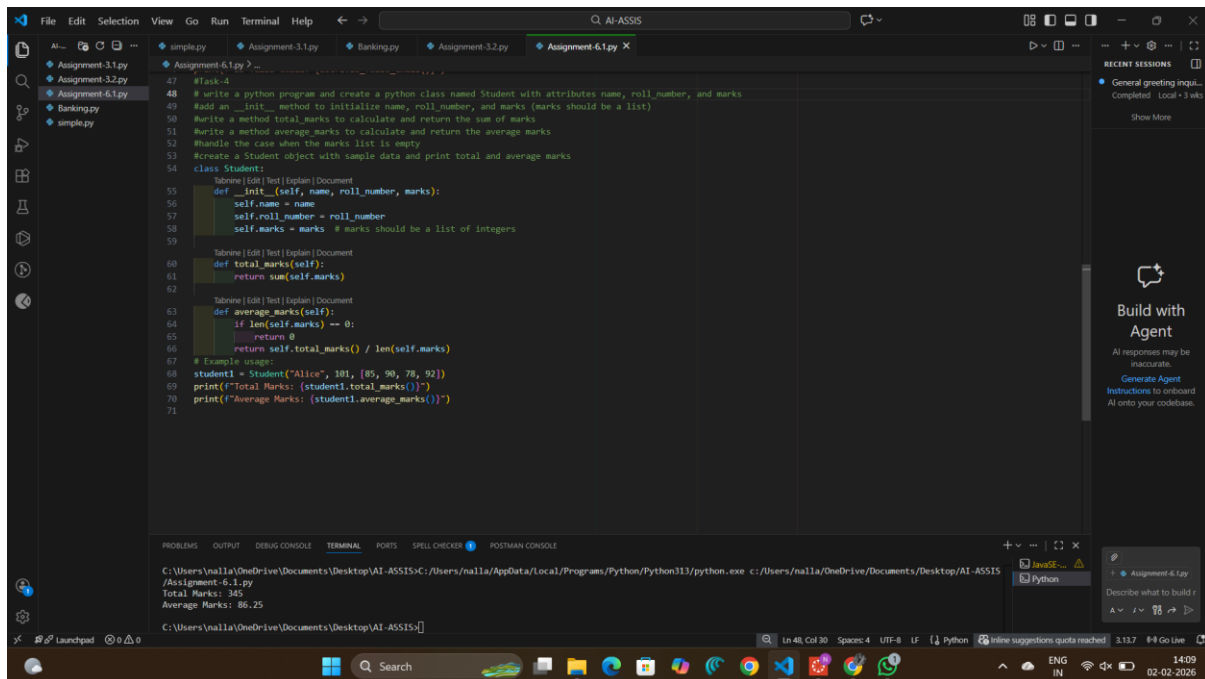
#write a method total\_marks to calculate and return the sum of marks

#write a method average\_marks to calculate and return the average marks

#handle the case when the marks list is empty

#create a Student object with sample data and print total and average marks

### Code and Output:



```
47 # Task 4
48 # write a python program and create a python class named Student with attributes name, roll_number, and marks
49 #add an __init__ method to initialize name, roll_number, and marks (marks should be a list)
50 #write a method total_marks to calculate and return the sum of marks
51 #write a method average_marks to calculate and return the average marks
52 #handle the case when the marks list is empty
53 #create a Student object with sample data and print total and average marks
54 class Student:
55     def __init__(self, name, roll_number, marks):
56         self.name = name
57         self.roll_number = roll_number
58         self.marks = marks # marks should be a list of integers
59
60     def total_marks(self):
61         return sum(self.marks)
62
63     def average_marks(self):
64         if len(self.marks) == 0:
65             return 0
66         return self.total_marks() / len(self.marks)
67
68 # Example usage:
69 student1 = Student("Alice", 101, [85, 90, 78, 92])
70 print("Total Marks: (student1.total_marks())")
71 print("Average Marks: (student1.average_marks())")
```

Output:

```
C:\Users\nalla\OneDrive\Documents\Desktop\AI-ASSIS>python.exe c:/Users/nalla/OneDrive/Documents/Desktop/AI-ASSIS/Assignment-6.1.py
Total Marks: 345
Average Marks: 86.25
```

### Analysis:

1. A Student class is defined with name, roll number, and marks.
2. Marks are stored as a list of integers.
3. The total\_marks() method calculates the sum of marks.
4. The average\_marks() method calculates the average safely.
5. A condition handles the case when the marks list is empty.

### 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.

### Prompt:

#Write a python code and create a python class named BankAccount with attributes account\_holder and balance

#add an \_\_init\_\_ method to initialize account\_holder and balance (default balance = 0)

#write deposit and withdraw methods using conditional statements to update balance

#write a method to return the current account balance

#use a loop to perform deposit and withdrawal operations and display the final balance

### Code and Output:

```

71 #Task-5
72 #Write a python code and create a python class named BankAccount with attributes account_holder and balance
73 #Add an __init__ method to initialize account_holder and balance (default balance = 0)
74 #Write deposit and withdraw methods using conditional statements to update balance
75 #Write a method to return the current account balance
76 #Use a loop to perform deposit and withdrawal operations and display the final balance
77 class BankAccount:
78     def __init__(self, account_holder, balance=0):
79         self.account_holder = account_holder
80         self.balance = balance
81     def deposit(self, amount):
82         if amount > 0:
83             self.balance += amount
84             print(f"Deposited: {amount}. New balance: {self.balance}")
85         else:
86             print("Deposit amount must be positive.")
87     def withdraw(self, amount):
88         if amount > self.balance:
89             print("Insufficient Funds.")
90         elif amount <= 0:
91             print("Withdrawal amount must be positive.")
92         else:
93             self.balance -= amount
94             print(f"Withdraw: {amount}. New balance: {self.balance}")
95     def get_balance(self):
96         return self.balance
97 # Example usage
98 account = BankAccount("John Doe", 1000)
99 account.deposit(500)
100 account.withdraw(200)
101 print(f"Final balance: {account.get_balance()}")
102

```

```

/Assignment-6.1.py
Deposited: 500. New balance: 1500
Withdraw: 200. New balance: 1300
Final balance: 1300

```

### Analysis:

1. A BankAccount class is created with account holder and balance.
2. The deposit method adds money only if the amount is positive.
3. The withdraw method checks for sufficient balance and valid amount.
4. Conditional statements ensure safe banking operations.
5. The program displays the final account balance after transactions.

