

# AI Assisted Coding Lab ass-6.1

Name: CH. Venu Gopal

Batch:13

2303A51844

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

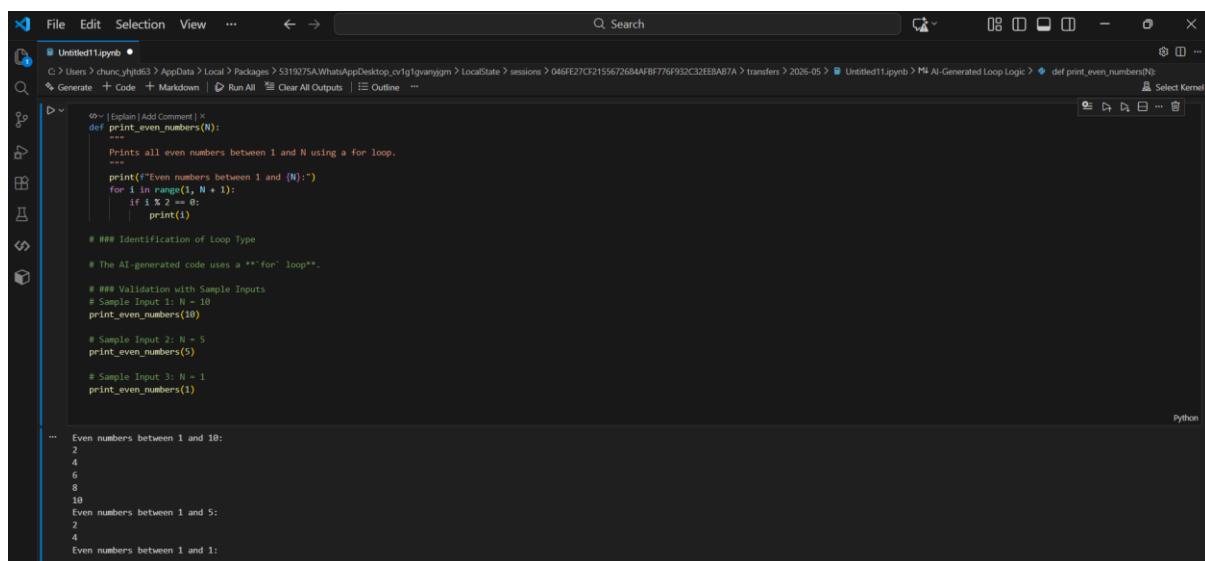
Task: Use an AI code completion tool to generate a loop-based program.

### 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 dark theme. The code cell contains 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 cell, the output is displayed in three sections:

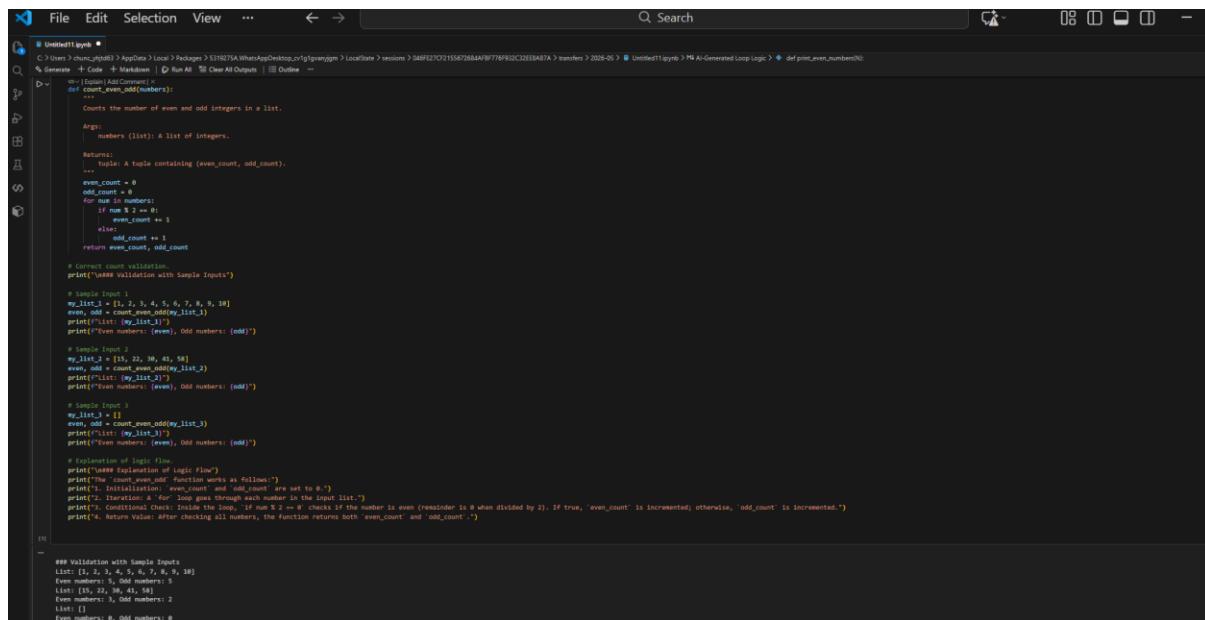
- Even numbers between 1 and 10:  
2  
4  
6  
8  
10
- Even numbers between 1 and 5:  
2  
4
- Even numbers between 1 and 1:  
None

## 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.”



The screenshot shows a code editor window with the following Python code:

```
# Undeclared loop
# C:\Users\chuck.jeph\Documents>AppData\Local\Packages>S17027CA.What'sAppDesktop_cwlgqsmjpm>LocalState>sessions>246F127C721556726844BF776F83DC2EE8A87A>transfers>2026-05>Untitled11.ipynb>M9 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 = [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("## Explaination 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: The 'for' loop goes through each element in the input list 'numbers'.")
print("3. Conditional Check: For each number, 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: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even numbers: 5
Odd numbers: 5
List: [15, 22, 38, 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.”

```

class User:
    def __init__(self, age, email):
        self._age = age
        self._email = email

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError("Age must be an integer.")
        if value < 0:
            raise ValueError("Age must be a positive number.")
        self._age = value

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        if not isinstance(value, str):
            raise TypeError("Email must be a string.")
        if not re.match(r"^\w+@[a-zA-Z]+\.\w+$", value):
            raise ValueError("Invalid email format.")
        self._email = value

    def __str__(self):
        return f"User({self.age}, {self.email})"

# Test verification of condition handling and test cases
print("... Valid Inputs ...")
try:
    user = User(10, "john.doe@example.com")
    print("Successfully created (user1)")
    user.age = 20
    print("Successfully updated (user1) age")
    user = User("10", "john.doe@example.com")
    print("Successfully created (user2) age")
except (TypeError, ValueError) as e:
    print(f"Expected error: {e}")

print("... Invalid Inputs ...")
try:
    user = User(-10, "john.doe@example.com")
    print("Successfully created (user1) age")
    user.age = -20
    print("Successfully updated (user1) age")
except (ValueError) as e:
    print(f"Expected error: {e}")

# Test with negative age
try:
    user = User(-10, "john.doe@example.com")
    print("Successfully created (user1) age")
except (ValueError) as e:
    print(f"Expected error: {e}")

# Test with non-string age
try:
    user = User("abc", "john.doe@example.com")
    print("Successfully created (user1) age")
except (TypeError) as e:
    print(f"Expected error: {e}")

# Test with empty email
try:
    user = User(10, "")
    print("Successfully created (user1) email")
except (ValueError) as e:
    print(f"Expected error: {e}")

# Test with invalid email format (missing .)
try:
    user = User(10, "john.doeexample.com")
    print("Successfully created (user1) email")
except (ValueError) as e:
    print(f"Expected error: {e}")

# Test with empty string email
try:
    user = User(10, "")
    print("Successfully created (user1) email")
except (ValueError) as e:
    print(f"Expected error: {e}")

# Test verification of logic flow
print("... User class logic as follows:")
# 1. **User** constructor: The `User` class is defined with an `__init__` constructor that takes `age` and `email` as arguments. These arguments are immediately passed to the `__setters` method for validation.
# 2. **age** Property: This property includes both a getter (dunder) and a setter (dunder).
#   - **getters**: Simply returns the attribute.
#   - **setters**: Checks the validation logic.
#     - It first checks if the value is an int using `isinstance`. If not, it raises a `TypeError`.
#     - It then checks if the value is non-negative using `value >= 0`. If it raises a `ValueError`, because `age` must be positive.
#     - If both checks pass, the value is assigned to `self._age`.
# 3. **email** Property: Similar to `age`, this property has a getter and a setter.
#   - **getters**: Contains the email validation logic.
#   - **setters**: Checks if the value is a string using `isinstance`. If not, it raises a `TypeError`.
#     - It uses the `re.match` function with a simple regular expression `r"^\w+@[a-zA-Z]+\.\w+$"` to check for a basic email format (at least one character before '@', one character between '@' and '.', and one character after '.').
#     - If validation passes, the value is assigned to `self._email`.
# 4. **__str__** Method: Provides a user-friendly string representation of the `User` object.

# ... Valid Inputs ...
# 1. **User** class definition: User(age: int, email: str) = type('User', (), {'__init__': lambda self, age, email: setattr(self, '_age', age), '__str__': lambda self: f"User({self._age}, {self._email})"})
# Successfully uploaded: User(age: int, email: str) = type('User', (), {'__init__': lambda self, age, email: setattr(self, '_age', age), '__str__': lambda self: f"User({self._age}, {self._email})"})
# ... Invalid Inputs ...
# 1. **User** class definition: User(age: int, email: str) = type('User', (), {'__init__': lambda self, age, email: setattr(self, '_age', age), '__str__': lambda self: f"User({self._age}, {self._email})"})
#   - Expected error: Age must be a positive number.
#   - Expected error: Age must be a positive number.

# Expected error: Invalid email format.
# Expected error: Invalid email format.
# Expected error: Invalid email format.

```

## Expected Output:

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

## 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."

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def __str__(self):
        return f"Name: {self.name}, Roll No: {self.roll_number}, Marks: {self.marks}"

    def calculate_total_marks(self):
        if len(self.marks) == 0:
            raise ValueError("Marks list must not be empty")
        else:
            total_marks = sum(self.marks)
            return total_marks

    def calculate_average_marks(self):
        if len(self.marks) == 0:
            raise ValueError("Marks list must not be empty")
        else:
            average_marks = self.calculate_total_marks() / len(self.marks)
            return average_marks

    def __repr__(self):
        return f"Student object for student {self.name} with roll number {self.roll_number} and marks {self.marks}"
```

```
# Test Case 1: Valid marks input (two numeric in list)
try:
    student = Student("John", "1001", [80, 75])
    print(f"Total Marks for student {student.name}: {student.calculate_total_marks()}")
    print(f"Average Marks for student {student.name}: {student.calculate_average_marks()}")
except ValueError as e:
    print(str(e))

# Test Case 2: Invalid marks input (one non-numeric type)
try:
    student = Student("John", "1001", [80, "75"])
    print(f"Total Marks for student {student.name}: {student.calculate_total_marks()}")
    print(f"Average Marks for student {student.name}: {student.calculate_average_marks()}")
except ValueError as e:
    print(str(e))

# Test Case 3: Invalid marks input (one list of non-numeric type)
try:
    student = Student("John", "1001", ["75"])
    print(f"Total Marks for student {student.name}: {student.calculate_total_marks()}")
    print(f"Average Marks for student {student.name}: {student.calculate_average_marks()}")
except ValueError as e:
    print(str(e))

# Test Case 4: Student with single mark
try:
    student = Student("John", "1001", 80)
    print(f"Total Marks for student {student.name}: {student.calculate_total_marks()}")
    print(f"Average Marks for student {student.name}: {student.calculate_average_marks()}")
except ValueError as e:
    print(str(e))

# Test Case 5: Student with no marks (empty list)
try:
    student = Student("John", "1001", [])
    print(f"Total Marks for student {student.name}: {student.calculate_total_marks()}")
    print(f"Average Marks for student {student.name}: {student.calculate_average_marks()}")
except ValueError as e:
    print(str(e))
```

## 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.”

The screenshot shows an AI-generated Python script for a bank account system. The code includes comments explaining the logic for deposit, withdrawal, and account creation. It uses classes for BankAccount and AccountHolder. The code handles user input for account number, owner name, and initial balance, and performs validation checks like digit-only account numbers and non-negative balances. It also includes a menu loop for deposit, withdrawal, and account details.

```

# %% AI-generated Bank Account System Program
# %% 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!

```

--- 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.