AI Assisted Coding

Assignment 6.3

Name: MD . Abdul Layeeq

Roll no: 2303A52217

Task Description #1: Classes (Student Class)

Scenario

You are developing a simple student information management module.

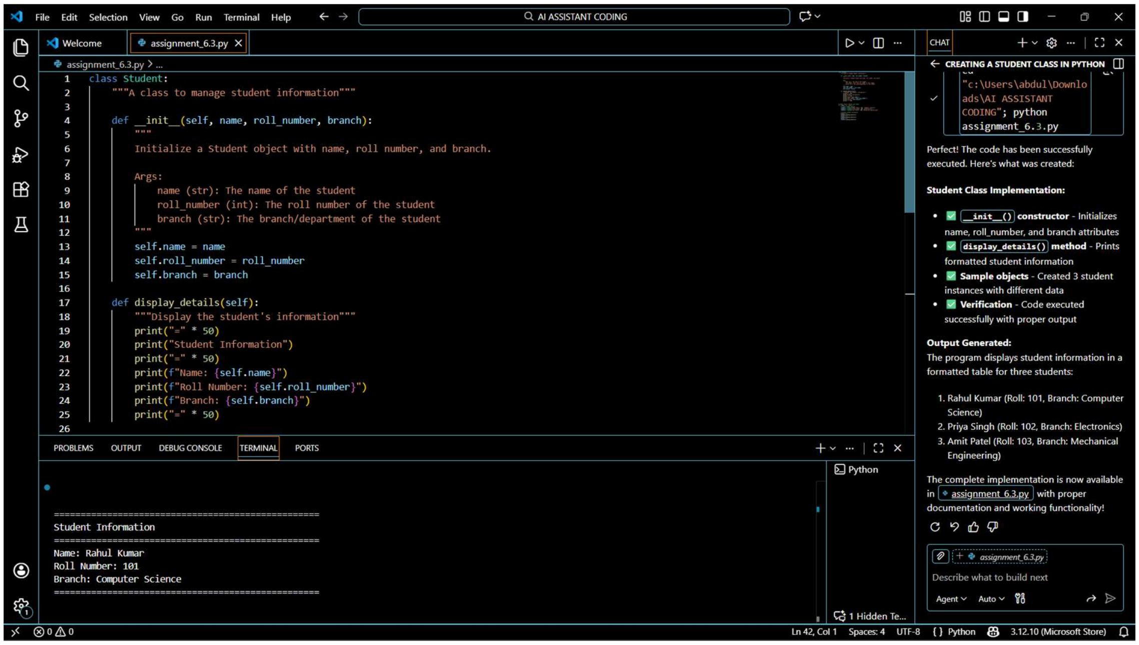
Task-1:

- Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.
- The class should include attributes such as name, roll number, and branch.
- Add a method display_details() to print student information.
- Execute the code and verify the output.
- Analyze the code generated by the AI tool for correctness and clarity.

Expected Output #1

- A Python class with a constructor (__init__) and a display_details() method.
- Sample object creation and output displayed on the console.
- Brief analysis of AI-generated code.

Output:-


File Edit Selection View Go Run Terminal Help
AI ASSISTANT CODING
Welcome
assignment_6.3.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
1 Hidden Te...
Ln 42, Col 1 Spaces: 4 UTF-8 { } Python 3.12.10 (Microsoft Store)
CHAT

```python
class Student:
    """A class to manage student information"""

    def __init__(self, name, roll_number, branch):
        """
        Initialize a Student object with name, roll number, and branch.

        Args:
            name (str): The name of the student
            roll_number (int): The roll number of the student
            branch (str): The branch/department of the student
        """
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        """Display the student's information"""
        print("=" * 50)
        print("Student Information")
        print("=" * 50)
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")
        print("=" * 50)

```

```
==================================================
Student Information
==================================================
Name: Rahul Kumar
Roll Number: 101
Branch: Computer Science
==================================================
```

CREATING A STUDENT CLASS IN PYTHON

```
"c:\Users\abdul\Downloads\AI ASSISTANT CODING"; python assignment_6.3.py
```

Perfect! The code has been successfully executed. Here's what was created:

Student Class Implementation:

- ✅ `__init__()` **constructor** - Initializes name, roll_number, and branch attributes
- ✅ `display_details()` **method** - Prints formatted student information
- ✅ **Sample objects** - Created 3 student instances with different data
- ✅ **Verification** - Code executed successfully with proper output

Output Generated:
The program displays student information in a formatted table for three students:

1. Rahul Kumar (Roll: 101, Branch: Computer Science)
2. Priya Singh (Roll: 102, Branch: Electronics)
3. Amit Patel (Roll: 103, Branch: Mechanical Engineering)

The complete implementation is now available in assignment_6.3.py with proper documentation and working functionality!

Describe what to build next

Agent Auto

Task Description #2: Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number.

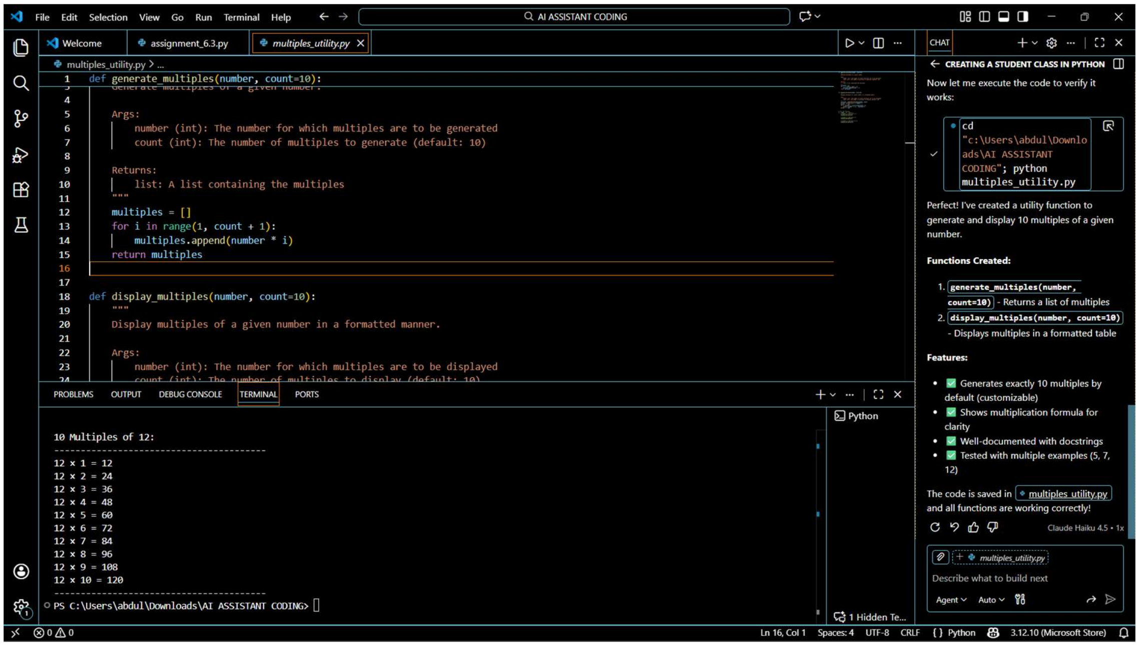
Task

- Prompt the AI tool to generate a function that prints the first 10 multiples of a given number using a loop.
- Analyze the generated loop logic.
- Ask the AI to generate the same functionality using another controlled looping structure (e.g., while instead of for).

Expected Output #2

- Correct loop-based Python implementation.
- Output showing the first 10 multiples of a number.
- Comparison and analysis of different looping approaches.

Output:-



```python
def generate_multiples(number, count=10):
    Generate multiples of a given number.

    Args:
        number (int): The number for which multiples are to be generated
        count (int): The number of multiples to generate (default: 10)

    Returns:
        list: A list containing the multiples
    """
    multiples = []
    for i in range(1, count + 1):
        multiples.append(number * i)
    return multiples


def display_multiples(number, count=10):
    """
    Display multiples of a given number in a formatted manner.

    Args:
        number (int): The number for which multiples are to be displayed
        count (int): The number of multiples to display (default: 10)
```

```
10 Multiples of 12:
----------------------------------------
12 × 1 = 12
12 × 2 = 24
12 × 3 = 36
12 × 4 = 48
12 × 5 = 60
12 × 6 = 72
12 × 7 = 84
12 × 8 = 96
12 × 9 = 108
12 × 10 = 120
----------------------------------------
PS C:\Users\abdul\Downloads\AI ASSISTANT CODING> 
```

CREATING A STUDENT CLASS IN PYTHON

Now let me execute the code to verify it works:

```
cd "c:\Users\abdul\Downloads\AI ASSISTANT CODING"; python multiples_utility.py
```

Perfect! I've created a utility function to generate and display 10 multiples of a given number.

Functions Created:

1. `generate_multiples(number, count=10)` - Returns a list of multiples
2. `display_multiples(number, count=10)` - Displays multiples in a formatted table

Features:

- ✅ Generates exactly 10 multiples by default (customizable)
- ✅ Shows multiplication formula for clarity
- ✅ Well-documented with docstrings
- ✅ Tested with multiple examples (5, 7, 12)

The code is saved in multiples_utility.py and all functions are working correctly!

Claude Haiku 4.5 • 1x

multiples_utility.py

Describe what to build next

Agent Auto

Task Description #3: Conditional Statements (Age Classification)

Scenario

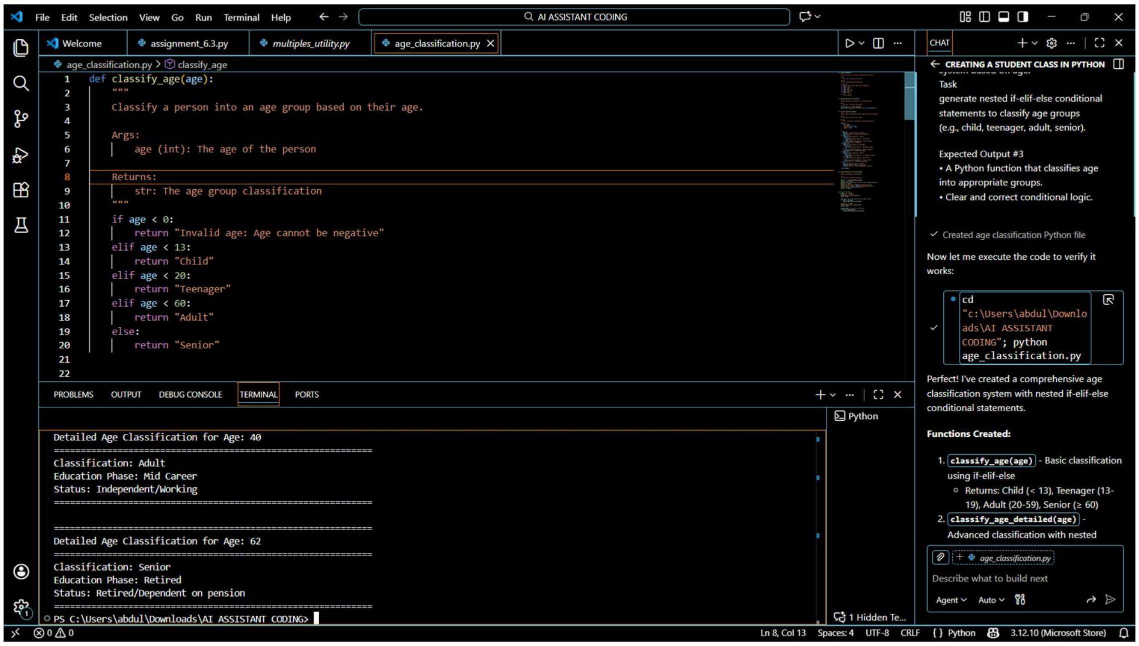
You are building a basic classification system based on age.

Task

- Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).
- Analyze the generated conditions and logic.
- Ask the AI to generate the same classification using alternative conditional structures (e.g., simplified conditions or dictionary-based logic).

Expected Output #3

- A Python function that classifies age into appropriate groups.
- Clear and correct conditional logic.
- Explanation of how the conditions work.



```python
def classify_age(age):
    """
    Classify a person into an age group based on their age.

    Args:
        age (int): The age of the person

    Returns:
        str: The age group classification
    """
    if age < 0:
        return "Invalid age: Age cannot be negative"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 60:
        return "Adult"
    else:
        return "Senior"


```

```
Detailed Age Classification for Age: 40
============================================================
Classification: Adult
Education Phase: Mid Career
Status: Independent/Working
============================================================

============================================================
Detailed Age Classification for Age: 62
============================================================
Classification: Senior
Education Phase: Retired
Status: Retired/Dependent on pension
============================================================
PS C:\Users\abdul\Downloads\AI ASSISTANT CODING>
```

Task
generate nested if-elif-else conditional statements to classify age groups (e.g., child, teenager, adult, senior).

Expected Output #3
• A Python function that classifies age into appropriate groups.
• Clear and correct conditional logic.

✓ Created age classification Python file

Now let me execute the code to verify it works:

```
cd "c:\Users\abdul\Downloads\AI ASSISTANT CODING"; python age_classification.py
```

Perfect! I've created a comprehensive age classification system with nested if-elif-else conditional statements.

Functions Created:

1. `classify_age(age)` - Basic classification using if-elif-else
    - Returns: Child (< 13), Teenager (13-19), Adult (20-59), Senior (≥ 60)
2. `classify_age_detailed(age)` - Advanced classification with nested

age_classification.py

Describe what to build next

Agent Auto

Task Description #4: For and While Loops (Sum of First n Numbers)

Scenario

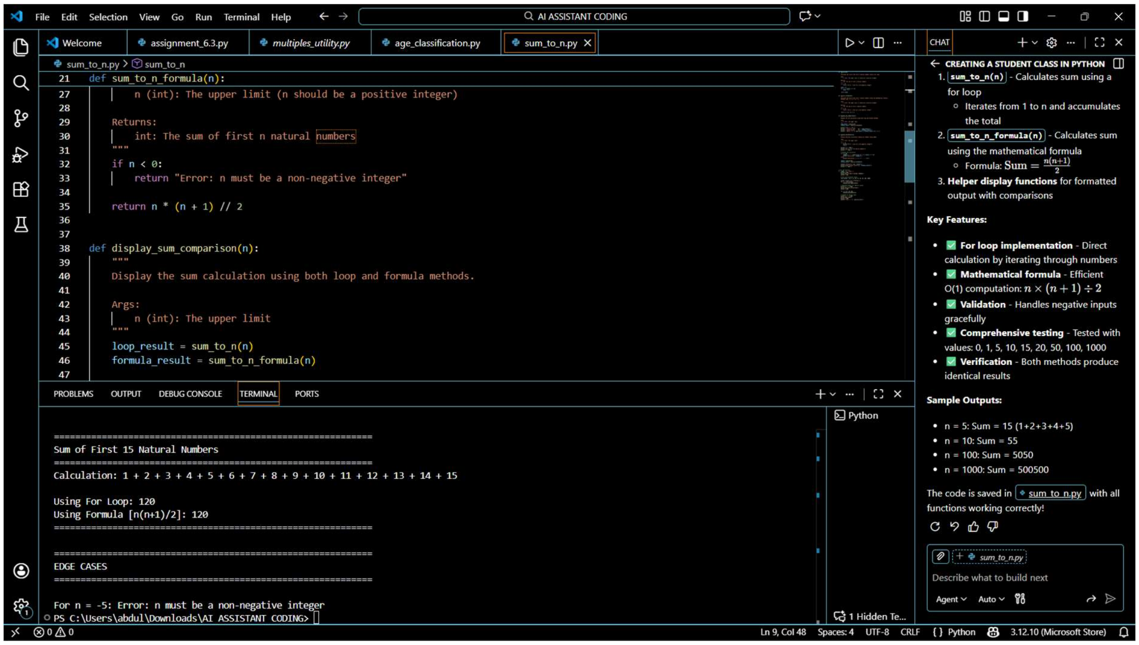
You need to calculate the sum of the first n natural numbers.

Task

- Use AI assistance to generate a sum_to_n() function using a for loop.
- Analyze the generated code.
- Ask the AI to suggest an alternative implementation using a while loop or a mathematical formula.

Expected Output #4

- Python function to compute the sum of first n numbers.
- Correct output for sample inputs.
- Explanation and comparison of different approaches.



```python
def sum_to_n_formula(n):
        n (int): The upper limit (n should be a positive integer)

    Returns:
        int: The sum of first n natural numbers
    """
    if n < 0:
        return "Error: n must be a non-negative integer"

    return n * (n + 1) // 2


def display_sum_comparison(n):
    """
    Display the sum calculation using both loop and formula methods.

    Args:
        n (int): The upper limit
    """
    loop_result = sum_to_n(n)
    formula_result = sum_to_n_formula(n)

```

```
============================================================
Sum of First 15 Natural Numbers
============================================================
Calculation: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15

Using For Loop: 120
Using Formula [n(n+1)/2]: 120
============================================================


============================================================
EDGE CASES
============================================================

For n = -5: Error: n must be a non-negative integer
PS C:\Users\abdul\Downloads\AI ASSISTANT CODING>
```

CREATING A STUDENT CLASS IN PYTHON

1. `sum_to_n(n)` - Calculates sum using a for loop
   - Iterates from 1 to n and accumulates the total
2. `sum_to_n_formula(n)` - Calculates sum using the mathematical formula
   - Formula: $\text{Sum} = \frac{n(n+1)}{2}$
3. **Helper display functions** for formatted output with comparisons

Key Features:

- ✅ **For loop implementation** - Direct calculation by iterating through numbers
- ✅ **Mathematical formula** - Efficient O(1) computation: $n \times (n+1) \div 2$
- ✅ **Validation** - Handles negative inputs gracefully
- ✅ **Comprehensive testing** - Tested with values: 0, 1, 5, 10, 15, 20, 50, 100, 1000
- ✅ **Verification** - Both methods produce identical results

Sample Outputs:

- n = 5: Sum = 15 (1+2+3+4+5)
- n = 10: Sum = 55
- n = 100: Sum = 5050
- n = 1000: Sum = 500500

The code is saved in sum_to_n.py with all functions working correctly!

sum_to_n.py

Describe what to build next

Agent Auto

Task Description #5: Classes (Bank Account Class)

Scenario

You are designing a basic banking application.

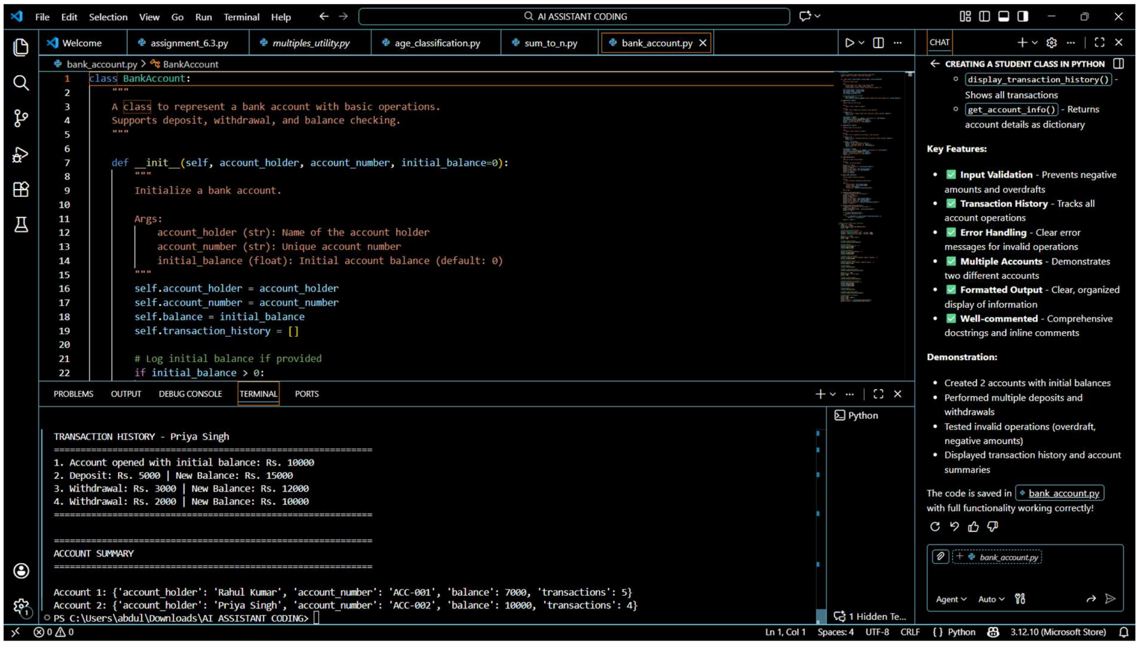
Task

- Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(), and check_balance().
- Analyze the AI-generated class structure and logic.

- Add meaningful comments and explain the working of the code.

Expected Output #5

- Complete Python Bank Account class.
- Demonstration of deposit and withdrawal operations with updated balance.
- Well-commented code with a clear explanation.



```python
class BankAccount:
    """
    A class to represent a bank account with basic operations.
    Supports deposit, withdrawal, and balance checking.
    """

    def __init__(self, account_holder, account_number, initial_balance=0):
        """
        Initialize a bank account.

        Args:
            account_holder (str): Name of the account holder
            account_number (str): Unique account number
            initial_balance (float): Initial account balance (default: 0)
        """
        self.account_holder = account_holder
        self.account_number = account_number
        self.balance = initial_balance
        self.transaction_history = []

        # Log initial balance if provided
        if initial_balance > 0:
```

```
TRANSACTION HISTORY - Priya Singh
============================================================
1. Account opened with initial balance: Rs. 10000
2. Deposit: Rs. 5000 | New Balance: Rs. 15000
3. Withdrawal: Rs. 3000 | New Balance: Rs. 12000
4. Withdrawal: Rs. 2000 | New Balance: Rs. 10000
============================================================

============================================================
ACCOUNT SUMMARY
============================================================

Account 1: {'account_holder': 'Rahul Kumar', 'account_number': 'ACC-001', 'balance': 7000, 'transactions': 5}
Account 2: {'account_holder': 'Priya Singh', 'account_number': 'ACC-002', 'balance': 10000, 'transactions': 4}
PS C:\Users\abdul\Downloads\AI ASSISTANT CODING>
```

CREATING A STUDENT CLASS IN PYTHON

  - `display_transaction_history()` - Shows all transactions
  - `get_account_info()` - Returns account details as dictionary

Key Features:

- ✅ **Input Validation** - Prevents negative amounts and overdrafts
- ✅ **Transaction History** - Tracks all account operations
- ✅ **Error Handling** - Clear error messages for invalid operations
- ✅ **Multiple Accounts** - Demonstrates two different accounts
- ✅ **Formatted Output** - Clear, organized display of information
- ✅ **Well-commented** - Comprehensive docstrings and inline comments

Demonstration:

- Created 2 accounts with initial balances
- Performed multiple deposits and withdrawals
- Tested invalid operations (overdraft, negative amounts)
- Displayed transaction history and account summaries

The code is saved in bank_account.py with full functionality working correctly!