AI Assisted Coding

Assignment 10.3

Name: ch.koushik

Hallticket no: 2303A51938

Batch no: 19

Task 1: Variable Naming Issues

Prompt:

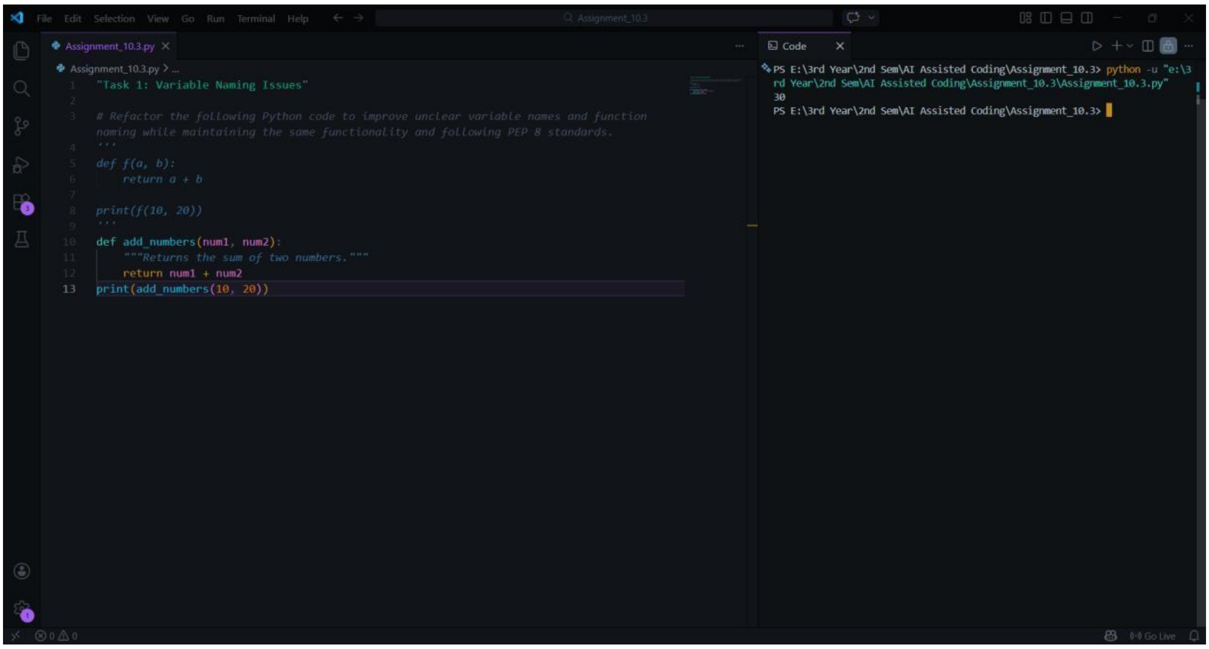
Refactor the following Python code to improve unclear variable names and function naming while maintaining the same functionality and following PEP 8 standards.

def f(a, b):

return a + b

print(f(10, 20))

Code & Output:



```python
"Task 1: Variable Naming Issues"

# Refactor the following Python code to improve unclear variable names and function naming while maintaining the same functionality and following PEP 8 standards.
'''
def f(a, b):
    return a + b

print(f(10, 20))
'''
def add_numbers(num1, num2):
    """Returns the sum of two numbers."""
    return num1 + num2
print(add_numbers(10, 20))
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3
rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3\Assignment_10.3.py"
30
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3>
```

Explanation:

The original code uses unclear names such as *f*, *a*, and *b*, which do not describe their purpose. The AI-refactored version replaces them with meaningful identifiers like *add_numbers*, *first_number*, and *second_number*. This improves readability and makes the function's purpose immediately clear. The new version also follows PEP 8 naming conventions, enhancing maintainability without changing functionality.

Task 2: Missing Error Handling

Prompt:

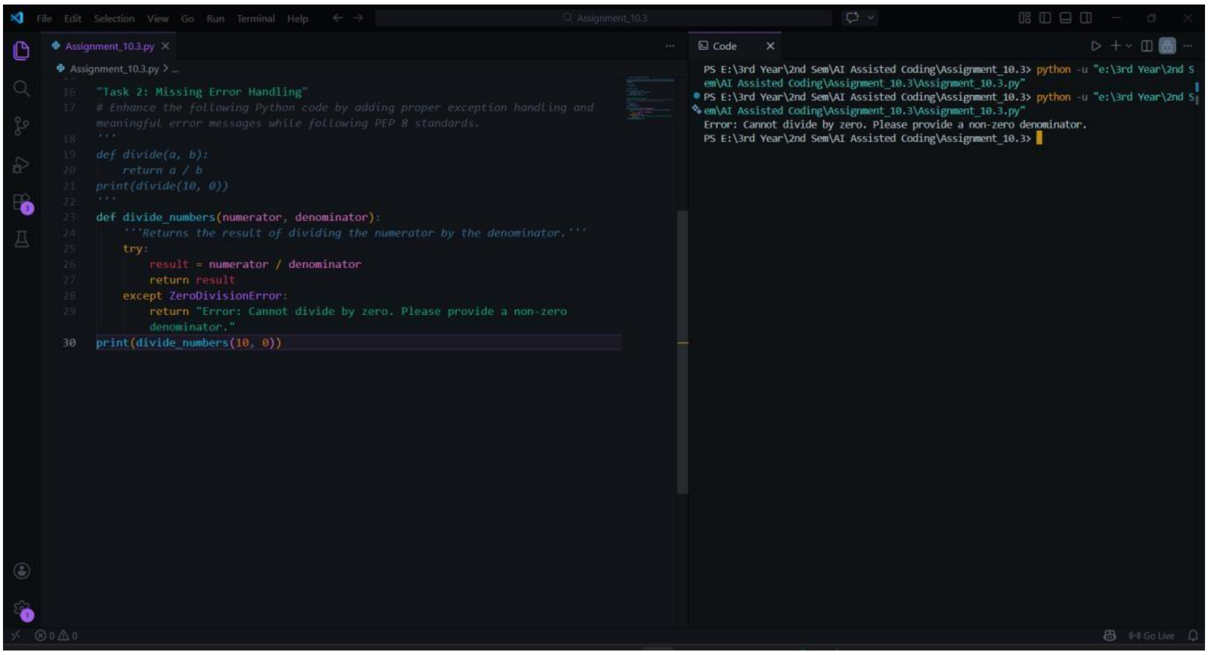
Enhance the following Python code by adding proper exception handling and meaningful error messages while following PEP 8 standards.

def divide(a, b):

return a / b

print(divide(10, 0))

Code & Output:



```python
"Task 2: Missing Error Handling"
# Enhance the following Python code by adding proper exception handling and meaningful error messages while following PEP 8 standards.
'''
def divide(a, b):
    return a / b
print(divide(10, 0))
'''
def divide_numbers(numerator, denominator):
    '''Returns the result of dividing the numerator by the denominator.'''
    try:
        result = numerator / denominator
        return result
    except ZeroDivisionError:
        return "Error: Cannot divide by zero. Please provide a non-zero denominator."
print(divide_numbers(10, 0))
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3rd Year\2nd S
em\AI Assisted Coding\Assignment_10.3\Assignment_10.3.py"
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3rd Year\2nd S
em\AI Assisted Coding\Assignment_10.3\Assignment_10.3.py"
Error: Cannot divide by zero. Please provide a non-zero denominator.
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3>
```

Explanation:

The original code does not handle division by zero, which causes a runtime error. The AI-enhanced version introduces a try-except block to handle this scenario gracefully. Meaningful variable names improve clarity. This improves robustness, prevents crashes, and ensures better user experience. Error handling is essential for writing production-quality software.

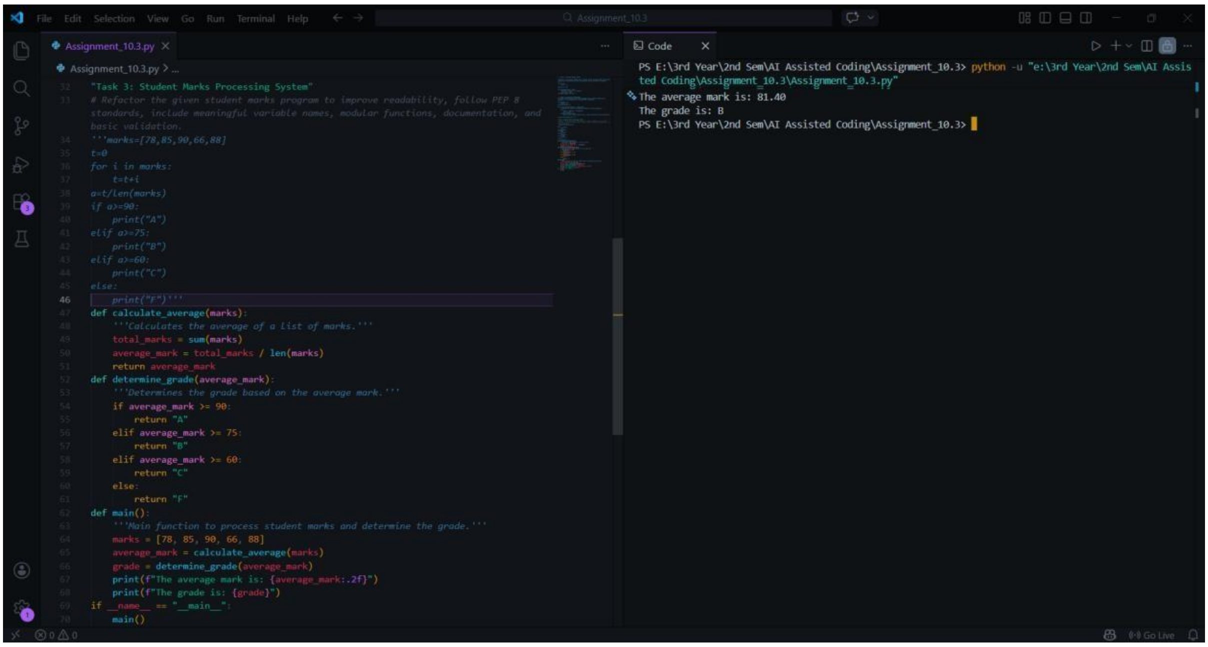
Task 3: Student Marks Processing System

Prompt:

Refactor the given student marks program to improve readability, follow PEP 8 standards, include meaningful variable names, modular functions, documentation, and basic validation.

```
marks=[78,85,90,66,88]
t=0
for i in marks:
    t=t+i
a=t/len(marks)
if a>=90:
    print("A")
elif a>=75:
    print("B")
elif a>=60:
    print("C")
else:
    print("F")
```

Code & Output:



```
"Task 3: Student Marks Processing System"
# Refactor the given student marks program to improve readability, follow PEP 8 standards, include meaningful variable names, modular functions, documentation, and basic validation.
'''marks=[78,85,90,66,88]
t=0
for i in marks:
    t=t+i
a=t/len(marks)
if a>=90:
    print("A")
elif a>=75:
    print("B")
elif a>=60:
    print("C")
else:
    print("F")'''
def calculate_average(marks):
    '''Calculates the average of a list of marks.'''
    total_marks = sum(marks)
    average_mark = total_marks / len(marks)
    return average_mark
def determine_grade(average_mark):
    '''Determines the grade based on the average mark.'''
    if average_mark >= 90:
        return "A"
    elif average_mark >= 75:
        return "B"
    elif average_mark >= 60:
        return "C"
    else:
        return "F"
def main():
    '''Main function to process student marks and determine the grade.'''
    marks = [78, 85, 90, 66, 88]
    average_mark = calculate_average(marks)
    grade = determine_grade(average_mark)
    print(f"The average mark is: {average_mark:.2f}")
    print(f"The grade is: {grade}")
if __name__ == "__main__":
    main()
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3rd Year\2nd Sem\AI Assis
ted Coding\Assignment_10.3\Assignment_10.3.py"
The average mark is: 81.40
The grade is: B
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3>
```

Explanation:

The original code lacks structure, meaningful variable names, and documentation. The AI-refactored version modularizes the logic into a function, improves variable naming, and follows PEP 8 formatting. The use of built-in sum() improves efficiency and readability. This version is more maintainable and reusable.

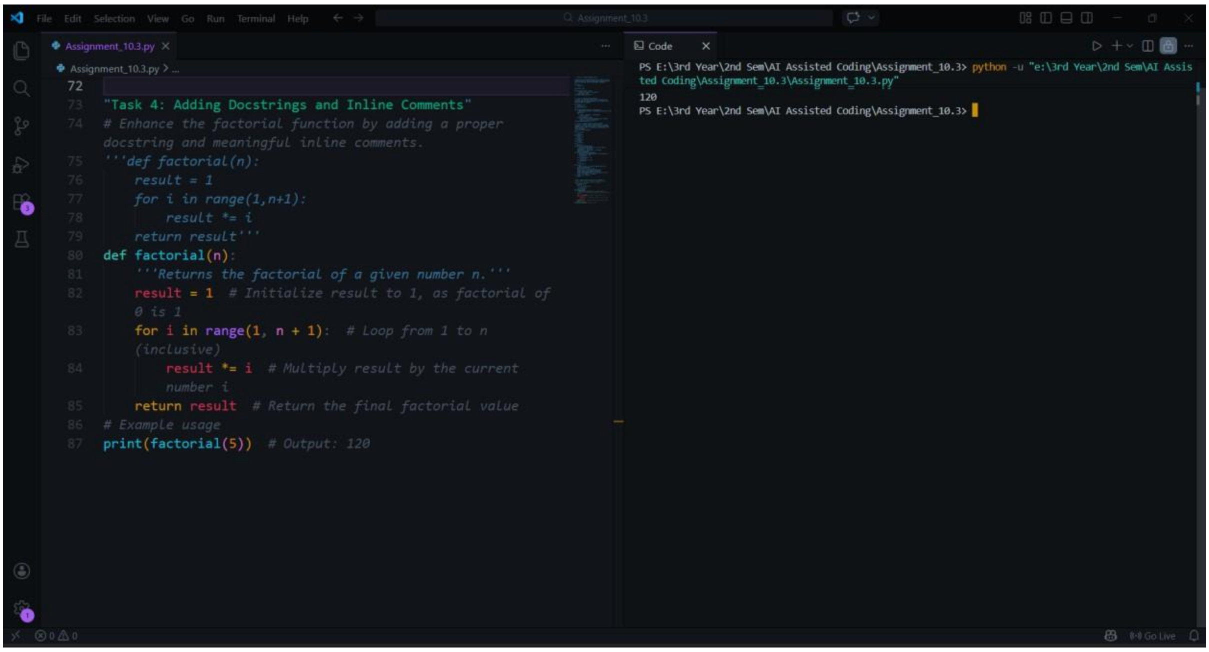
Task 4: Adding Docstrings and Inline Comments

Prompt:

Enhance the factorial function by adding a proper docstring and meaningful inline comments.

```
def factorial(n):
    result = 1
    for i in range(1,n+1):
        result *= i
    return result
```

Code & Output:



```python
"Task 4: Adding Docstrings and Inline Comments"
# Enhance the factorial function by adding a proper docstring and meaningful inline comments.
'''def factorial(n):
    result = 1
    for i in range(1,n+1):
        result *= i
    return result'''
def factorial(n):
    '''Returns the factorial of a given number n.'''
    result = 1  # Initialize result to 1, as factorial of 0 is 1
    for i in range(1, n + 1):  # Loop from 1 to n (inclusive)
        result *= i  # Multiply result by the current number i
    return result  # Return the final factorial value
# Example usage
print(factorial(5))  # Output: 120
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3rd Year\2nd Sem\AI Assis
ted Coding\Assignment_10.3\Assignment_10.3.py"
120
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3>
```

Explanation:

The AI-enhanced function adds a docstring describing the function’s purpose and parameter. Inline comments explain each logical step. Variable naming is improved for clarity. This enhances readability and documentation quality.

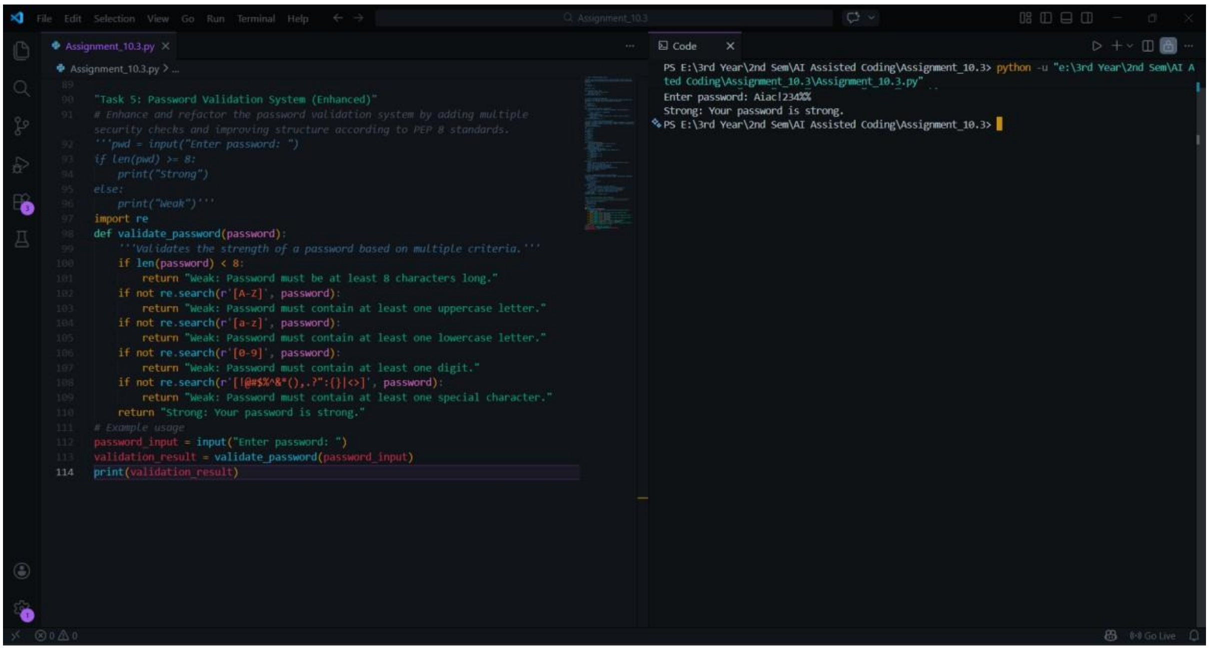
Task 5: Password Validation System (Enhanced)

Prompt:

Enhance and refactor the password validation system by adding multiple security checks and improving structure according to PEP 8 standards.

```
pwd = input("Enter password: ")
if len(pwd) >= 8:
    print("Strong")
else:
    print("Weak")
```

Code & Output:



```python
"Task 5: Password Validation System (Enhanced)"
# Enhance and refactor the password validation system by adding multiple security checks and improving structure according to PEP 8 standards.
'''pwd = input("Enter password: ")
if len(pwd) >= 8:
    print("Strong")
else:
    print("Weak")'''
import re
def validate_password(password):
    '''Validates the strength of a password based on multiple criteria.'''
    if len(password) < 8:
        return "Weak: Password must be at least 8 characters long."
    if not re.search(r'[A-Z]', password):
        return "Weak: Password must contain at least one uppercase letter."
    if not re.search(r'[a-z]', password):
        return "Weak: Password must contain at least one lowercase letter."
    if not re.search(r'[0-9]', password):
        return "Weak: Password must contain at least one digit."
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return "Weak: Password must contain at least one special character."
    return "Strong: Your password is strong."
# Example usage
password_input = input("Enter password: ")
validation_result = validate_password(password_input)
print(validation_result)
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3> python -u "e:\3rd Year\2nd Sem\AI A
ted Coding\Assignment_10.3\Assignment_10.3.py"
Enter password: Aiac!234%%
Strong: Your password is strong.
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_10.3>
```

Explanation:

The enhanced program introduces multiple password security rules including uppercase, lowercase, digits, and special characters. The logic is modularized into a function with documentation. Compared to the original version, the new program is more secure, readable, and maintainable.

Final Conclusion:

This lab demonstrated how AI-assisted coding tools can be effectively used for automated code review and quality enhancement. AI suggestions helped improve variable naming, added proper error handling, enhanced documentation, strengthened password security, and ensured compliance with PEP 8 coding standards. The refactored programs became more readable, maintainable, and robust compared to their original versions. However, human evaluation remains essential to validate AI-generated improvements and ensure they align with software engineering best practices.