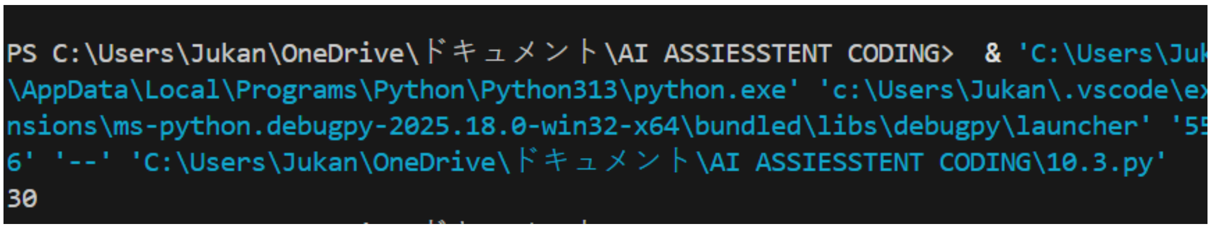
ASSIGNMENT-10.5

2303A51528

TASK-1: Variable Naming Issues

Task: Use AI to improve unclear variable names

```
def add_numbers(num1, num2):
    return num1 + num2


print(add_numbers(10, 20))
```



```
PS C:\Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING>  & 'C:\Users\Juk
\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Jukan\.vscode\ex
nsions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '55
6' '--' 'C:\Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING\10.3.py'
30
```

Observation:

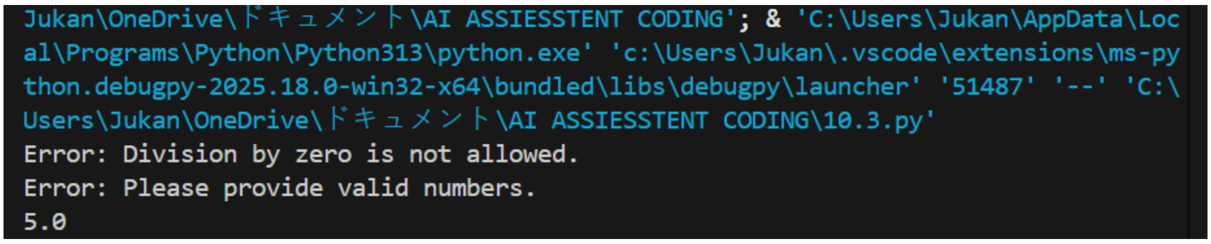
The original program used very short and unclear variable names such as f, a, and b, making the code difficult to understand. After AI enhancement, these names were replaced with meaningful identifiers like add_numbers, num1, and num2. This improved the code's readability and made its purpose clear. Following PEP 8 naming conventions, the function now clearly shows that it adds two numbers. This change makes the code easier to maintain, reuse, and understand for other programmers.

TASK-2: Missing Error Handling

Task: Use AI to add proper error handling.

ASSIGNMENT-10.5

```
def divide(dividend, divisor):
    try:
        result = dividend / divisor
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Please provide valid numbers."


print(divide(10, 0))
print(divide(10, 'a'))
print(divide(20, 4))
```



```
Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING'; & 'C:\Users\Jukan\AppData\Loc
al\Programs\Python\Python313\python.exe' 'c:\Users\Jukan\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51487' '--' 'C:\
Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING\10.3.py'
Error: Division by zero is not allowed.
Error: Please provide valid numbers.
5.0
```

Observation:

In the original code, there was no error handling when performing division. Dividing by zero caused the program to crash, and there were no checks for invalid inputs. The AI-enhanced version introduced a try-except block that handles both ZeroDivisionError and TypeError. It provides clear error messages like “Division by zero is not allowed” or “Please provide valid numbers.” This improvement prevents runtime crashes, increases robustness, and enhances the user experience by providing meaningful feedback. The variable names were also made descriptive (dividend, divisor) for better clarity.

ASSIGNMENT-10.5

TASK-3: Student Marks Processing System

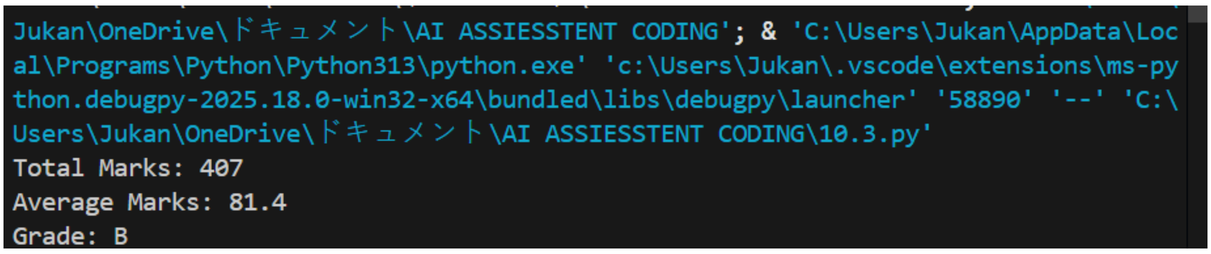
```
def calculate_grade(marks):
    """
    Calculate and display the total, average, and grade of a student.

    Parameters
    ----------
    marks : list of int or float
        A list containing marks for each subject.

    Returns
    -------
    dict
        A dictionary containing the total marks, average marks, and
grade.
    """
    # Input validation
    if not marks or not isinstance(marks, list):
        return {"error": "Invalid input: marks must be a non-empty list."}
    if not all(isinstance(m, (int, float)) for m in marks):
        return {"error": "Invalid input: all marks must be numbers."}
```

ASSIGNMENT-10.5

```
    # Calculate total and average
    total_marks = sum(marks)
    average_marks = total_marks / len(marks)

    # Determine grade based on average
    if average_marks >= 90:
        grade = "A"
    elif average_marks >= 75:
        grade = "B"
    elif average_marks >= 60:
        grade = "C"
    else:
        grade = "F"

    # Return results in a structured format
    return {
        "Total Marks": total_marks,
        "Average Marks": average_marks,
        "Grade": grade
    }

# Example usage
student_marks = [78, 85, 90, 66, 88]
result = calculate_grade(student_marks)
```

ASSIGNMENT-10.5

```
if "error" in result:
    print(result["error"])
else:
    print("Total Marks:", result["Total Marks"])
    print("Average Marks:", result["Average Marks"])
    print("Grade:", result["Grade"])
```



```
Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING'; & 'C:\Users\Jukan\AppData\Loc
al\Programs\Python\Python313\python.exe' 'c:\Users\Jukan\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58890' '--' 'C:\
Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING\10.3.py'
Total Marks: 407
Average Marks: 81.4
Grade: B
```

Observation:

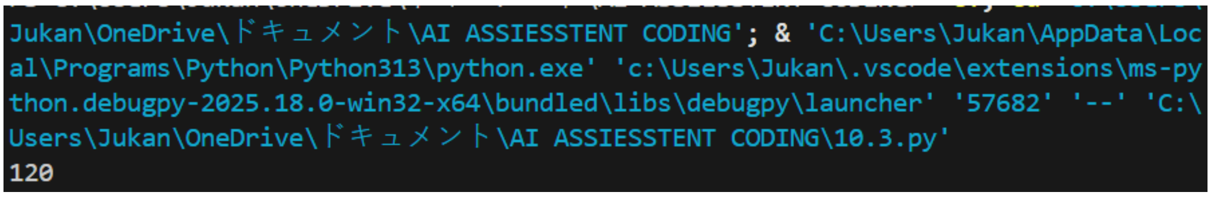
The original student marks program lacked readability, structure, and error handling. Variables like t and a were not meaningful, and there were no comments or documentation. The AI-enhanced version refactored the code into a function named calculate_grade(), introduced proper variable names such as total_marks and average_marks, and added a NumPy-style docstring. Input validation was included to ensure marks are numeric and non-empty. The program now clearly prints total, average, and grade with well-organized output. These improvements make the code PEP 8 compliant, easier to maintain, and much more professional.

TASK-4: Use AI to add docstrings and inline comments

ASSIGNMENT-10.5

to the following function.

```
def factorial(n):
    """
    Calculate the factorial of a given non-negative integer.

    Parameters
    ----------
    n : int
        The number for which the factorial is to be calculated.
        Must be a non-negative integer.

    Returns
    -------
    int
        The factorial of the given number `n`.
        Returns 1 if `n` is 0 or 1.

    Raises
    ------
    ValueError
        If `n` is negative.
    """
    # Validate input
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers.")

    result = 1  # Initialize result variable

    # Loop from 1 to n (inclusive) and multiply result by each number
    for i in range(1, n + 1):
        result *= i  # Multiply result by current number

    return result  # Return the computed factorial

# Example usage
print(factorial(5))  # Output: 120'''
```



```
Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING'; & 'C:\Users\Jukan\AppData\Loc
al\Programs\Python\Python313\python.exe' 'c:\Users\Jukan\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57682' '--' 'C:\
Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING\10.3.py'
120
```

Observation:

The original factorial function was functional but lacked documentation and inline comments, making it unclear to other readers. The AI-enhanced version added a detailed NumPy-style docstring explaining the purpose, parameters, return values, and possible errors. Inline comments were added to describe each logical step, and a validation check was included to prevent negative inputs. These additions greatly improved the readability, reliability, and maintainability of the code while following PEP 8 guidelines.

ASSIGNMENT-10.5

TASK-5: Password Validation System (Enhanced)

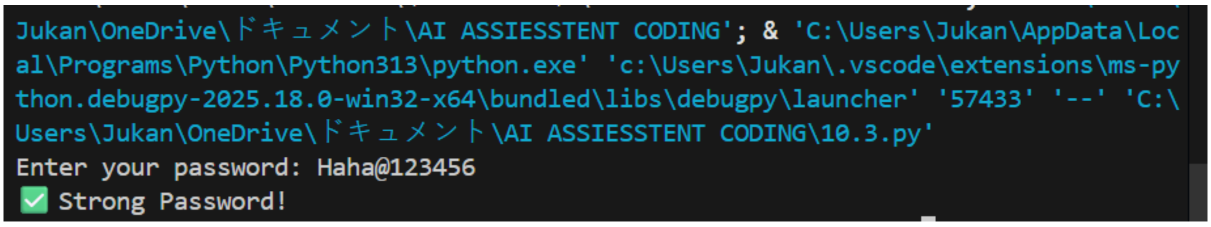
```
import re


def is_strong_password(password):
    """
    Validate the strength of a given password based on security rules.

    Parameters
    ----------
    password : str
        The password string to validate.

    Returns
    -------
    bool
        True if the password meets all security requirements, False otherwise.
    """
    # Rule 1: Minimum length of 8 characters
    if len(password) < 8:
        print("❌ Password must be at least 8 characters long.")
        return False
```

ASSIGNMENT-10.5

```
    # Rule 2: Must contain at least one uppercase letter
    if not re.search(r"[A-Z]", password):
        print("❌ Password must include at least one uppercase letter.")
        return False

    # Rule 3: Must contain at least one lowercase letter
    if not re.search(r"[a-z]", password):
        print("❌ Password must include at least one lowercase letter.")
        return False

    # Rule 4: Must contain at least one digit
    if not re.search(r"[0-9]", password):
        print("❌ Password must include at least one digit.")
        return False

    # Rule 5: Must contain at least one special character
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        print("❌ Password must include at least one special
character.")
        return False

    # If all checks pass
    print("✅ Strong Password!")
    return True
```

ASSIGNMENT-10.5

```
# Example usage
user_password = input("Enter your password: ")
is_strong_password(user_password)
```



```
Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING'; & 'C:\Users\Jukan\AppData\Loc
al\Programs\Python\Python313\python.exe' 'c:\Users\Jukan\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57433' '--' 'C:\
Users\Jukan\OneDrive\ドキュメント\AI ASSIESSTENT CODING\10.3.py'
Enter your password: Haha@123456
✅ Strong Password!
```

Observation:

The original password program only checked for a minimum length of eight characters, which was insufficient for real-world use. The AI-enhanced version introduced multiple security checks: minimum length, presence of uppercase and lowercase letters, digits, and special characters. It used descriptive variable names, added inline comments, and included a docstring for clarity. Specific error messages were also implemented to guide the user. The enhanced code follows PEP 8 standards, is modular, reusable, and far more secure. Overall, AI improvements strengthened code quality, readability, and security robustness.