

# AI ASSISTED CODING

## ASSIGNMENT-10.5

NAME: S.Ravikiran

HT.NO: 2303A52261

BATCH: 36

Task Description #1 – Variable Naming Issues

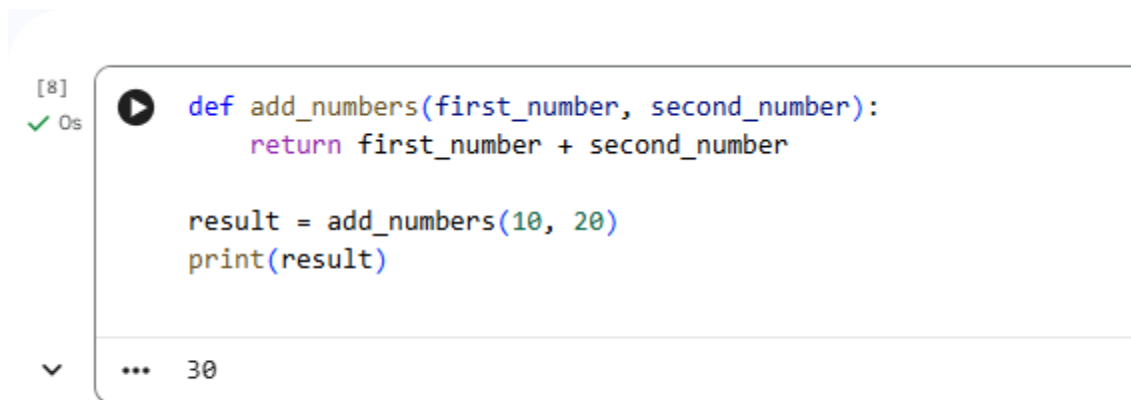
Task: Use AI to improve unclear variable names.

Sample Input Code:

```
def f(a, b):  
  
    return a + b  
  
print(f(10, 20))
```

Expected Output:

- Code rewritten with meaningful function and variable names.

A screenshot of a code editor interface. On the left, there is a sidebar with a file explorer icon, a search icon, and a list of files. The main area shows a Python code snippet. The code is as follows:

```
[8]  
✓ 0s  
def add_numbers(first_number, second_number):  
    return first_number + second_number  
  
result = add_numbers(10, 20)  
print(result)
```

At the bottom of the code editor, there is a status bar showing a dropdown arrow, three dots, and the number 30.

Task Description #2 – Missing Error Handling

Task: Use AI to add proper error handling.

Sample Input Code:

```
def divide(a, b):  
  
    return a / b
```

```
print(divide(10, 0))
```

Expected Output:

- Code with exception handling and clear error messages

```
[9]
✓ Os
def divide_numbers(numerator, denominator):
    try:
        return numerator / denominator
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Both inputs must be numbers."

    result = divide_numbers(10, 0)
    print(result)

... Error: Division by zero is not allowed.
```

### Task Description #3: Student Marks Processing System

The following program calculates total, average, and grade of a student, but it has poor readability, style issues, and no error handling.

```
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")
```

## Task:

- Use AI to refactor the code to follow PEP 8 standards.
- Add meaningful variable names, functions, and comments.
- Add basic input validation and documentation.

```
from typing import List

def calculate_average(marks: List[int]) -> float:
    """
    Calculate the average score from a list of marks.
    :param marks: List of integer marks
    :return: Average of the marks
    :raises ValueError: If marks list is empty or contains invalid values
    """
    if not marks:
        raise ValueError("Marks list cannot be empty.")

    for mark in marks:
        if not isinstance(mark, (int, float)) or mark < 0:
            raise ValueError("All marks must be non-negative numbers.")

    total_marks = sum(marks)
    return total_marks / len(marks)

def determine_grade(average: float) -> str:
    """
    Determine the grade based on average marks.
    :param average: Average marks
    :return: Grade as a string
    """
    if average >= 90:
        return "A"
    elif average >= 75:
        return "B"
    elif average >= 60:
        return "C"
    else:
        return "F"

def main():
    """
    Main function to process student marks and display grade.
    """
    student_marks = [78, 85, 90, 66, 88]
    try:
        average_marks = calculate_average(student_marks)
        grade = determine_grade(average_marks)
        print(f"Average Marks: {average_marks:.2f}")
        print(f"Grade: {grade}")
    except ValueError as error:
        print(f"Input Error: {error}")

if __name__ == "__main__":
    main()
```

✓ Average Marks: 81.40  
Grade: B

## Task Description #4: Use AI to add docstrings and inline comments

to the following function.

```
def factorial(n):
```

```
    result = 1
```

```
    for i in range(1,n+1):
```

```
        result *= i
```

```
    return result
```

```
1  def factorial(n):  
    """  
    Calculate the factorial of a non-negative integer.  
  
    :param n: A non-negative integer  
    :return: Factorial of n  
    :raises ValueError: If n is negative  
    :raises TypeError: If n is not an integer  
    """  
    if not isinstance(n, int):  
        raise TypeError("Input must be an integer.")  
  
    if n < 0:  
        raise ValueError("Input must be a non-negative integer.")  
  
    result = 1  
    for i in range(1, n + 1):  
        result *= i  
  
    return result  
  
print(factorial(5))  
  
... 120
```

### Task Description #5: Password Validation System (Enhanced)

The following Python program validates a password using only a minimum length check, which is insufficient for real-world security requirements.

```
pwd = input("Enter password: ")
```

```
if len(pwd) >= 8:
```

```
    print("Strong")
```

```
else:
```

```
    print("Weak")
```

Task:

1. Enhance password validation using AI assistance to include multiple security rules such as:

- o Minimum length requirement
- o Presence of at least one uppercase letter
- o Presence of at least one lowercase letter
- o Presence of at least one digit
- o Presence of at least one special character

2. Refactor the program to:

- o Use meaningful variable and function names

- o Follow PEP 8 coding standards

- o Include inline comments and a docstring

3. Analyze the improvements by comparing the original and AI-enhanced versions in terms of:

- o Code readability and structure

- o Maintainability and reusability

- o Security strength and robustness

4. Justify the AI-generated changes, explaining why each added rule and refactoring decision improves the overall quality of the program.

```
import re
def is_strong_password(password: str) -> bool:
    """
    Validate a password against common security rules.
    Rules:
    - Minimum length of 8 characters
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one digit
    - At least one special character
    :param password: Password string entered by the user
    :return: True if password meets all rules, otherwise False
    """
    if len(password) < 8:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"\d", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True
def main():
    """
    Accept user input and display password strength result.
    """
    user_password = input("Enter password: ")
    if is_strong_password(user_password):
        print("Strong password")
    else:
        print("Weak password")
if __name__ == "__main__":
    main()
```

---

```
*** Enter password: AI
Weak password
```