ASSIGNMENT-10.5

Name: R.Akshitha

Roll.No:2303A52464

Batch:36

Task Description #1 – Variable Naming Issues

Task: Use AI to improve unclear variable names.

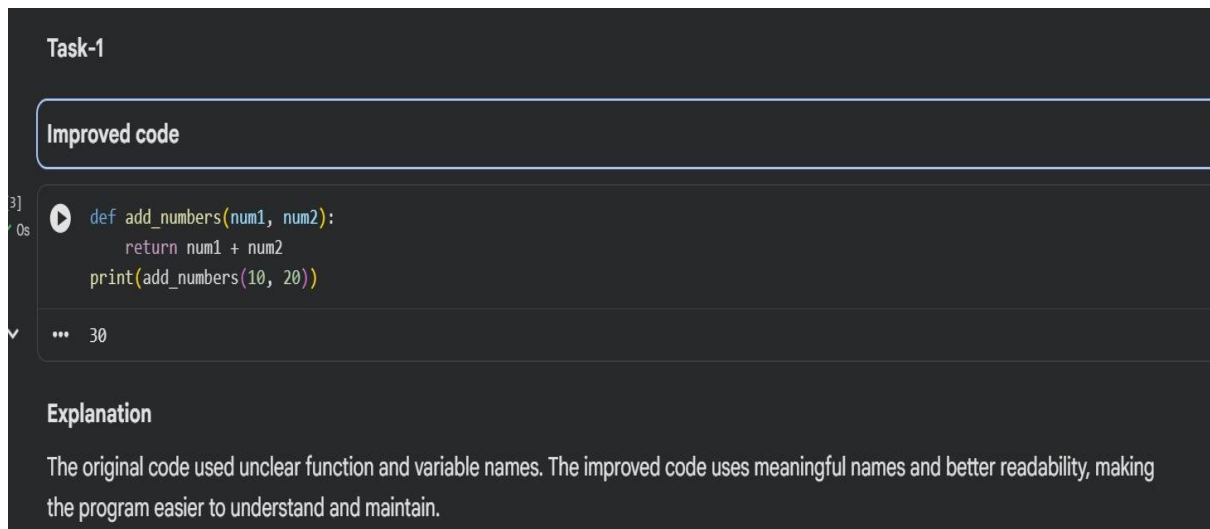Sample Input Code:

def f(a, b):

return a + b

print(f(10, 20))

Expected Output:

• Code rewritten with meaningful function and variable names.

Output:

```
Task-1

Improved code

[3]    def add_numbers(num1, num2):
            return num1 + num2
        print(add_numbers(10, 20))

    ...  30
```

Explanation

The original code used unclear function and variable names. The improved code uses meaningful names and better readability, making the program easier to understand and maintain.

Task Description #2 – Missing Error Handling

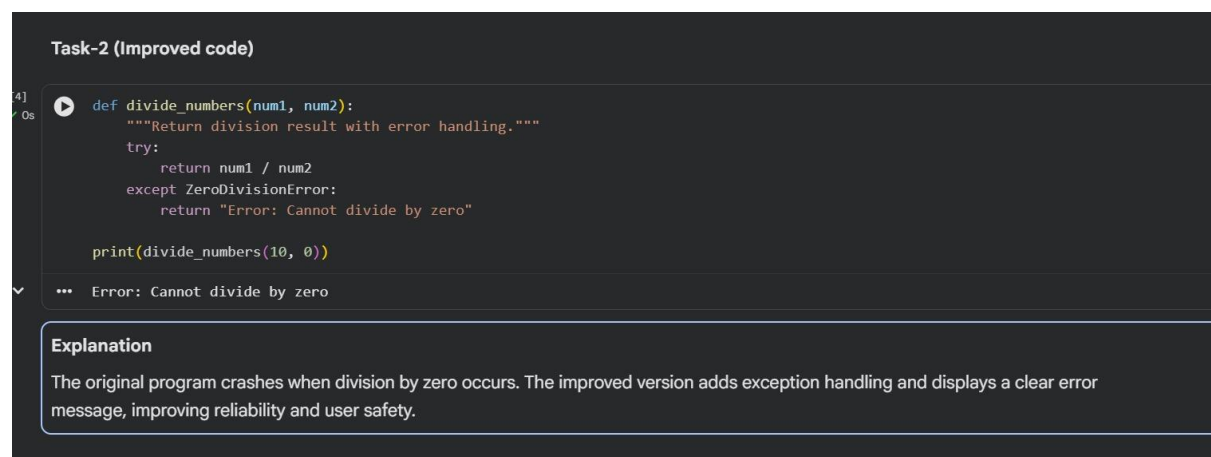**Task:** Use AI to add proper error handling.

Sample Input Code:

```python
def divide(a, b):

return a / b

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

Expected Output:

• Code with exception handling and clear error messages

Output:

Task-2 (Improved code)

```python
def divide_numbers(num1, num2):
    """Return division result with error handling."""
    try:
        return num1 / num2
    except ZeroDivisionError:
        return "Error: Cannot divide by zero"

print(divide_numbers(10, 0))
```

```
Error: Cannot divide by zero
```

Explanation

The original program crashes when division by zero occurs. The improved version adds exception handling and displays a clear error message, improving reliability and user safety.

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.

```python
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.

Output:


```python
Task-3 (Improved Code)

[5]
✓ 0s
        def calculate_grade(marks):
            """Calculate total, average, and grade from marks list."""

            if not marks:
                return "No marks provided"

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

            if average_marks >= 90:
                grade = "A"
            elif average_marks >= 75:
                grade = "B"
            elif average_marks >= 60:
                grade = "C"
            else:
                grade = "F"

            return total_marks, average_marks, grade


    marks_list = [78, 85, 90, 66, 88]
    total, average, grade = calculate_grade(marks_list)
```

```
            grade = "B"
        elif average_marks >= 60:
            grade = "C"
        else:
            grade = "F"

        return total_marks, average_marks, grade


    marks_list = [78, 85, 90, 66, 88]
    total, average, grade = calculate_grade(marks_list)

    print("Total:", total)
    print("Average:", average)
    print("Grade:", grade)

    Total: 407
    Average: 81.4
    Grade: B
```

Explanation The original code had poor readability, no function structure, and no validation. The improved version follows PEP-8 standards, uses meaningful names, introduces a reusable function, and improves clarity and maintainability.

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

Output:

Task-4 (Improved Code)

```
[6]
✓ 0s    def factorial(n):
            """
            Calculate factorial of a non-negative integer.
            """

            if n < 0:
                return "Factorial not defined for negative numbers"

            result = 1

            for i in range(1, n + 1):
                result *= i

            return result
```

Explanation

The original function lacked documentation and input validation. The improved version adds a docstring, clearer formatting, and validation for negative numbers, making the function safer and easier to understand.

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.

Output:

Task-5 (Improved Code)

```
[9]
✓ 23s
        import re

        def is_strong_password(password):
            """Validate password strength using multiple security rules."""

            if len(password) < 8:
                return "Weak: Minimum 8 characters required"

            if not re.search(r"[A-Z]", password):
                return "Weak: Must include an uppercase letter"

            if not re.search(r"[a-z]", password):
                return "Weak: Must include a lowercase letter"

            if not re.search(r"\d", password):
                return "Weak: Must include a digit"

            if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
                return "Weak: Must include a special character"

            return "Strong Password"
```

```
        if not re.search(r"[a-z]", password):
            return "Weak: Must include a lowercase letter"

        if not re.search(r"\d", password):
            return "Weak: Must include a digit"

        if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
            return "Weak: Must include a special character"

        return "Strong Password"


    user_password = input("Enter password: ")
    print(is_strong_password(user_password))
```

```
Enter password: Student@2023
Strong Password
```

Explanation

The original program checked only password length, which is not secure. The improved version adds multiple validation rules, meaningful function structure, and documentation, making the password system more secure and reusable.