AI ASSISTED CODING

Sai Thrishool **2303A51127**

BATCH – 03 **20 – 02 – 2026**

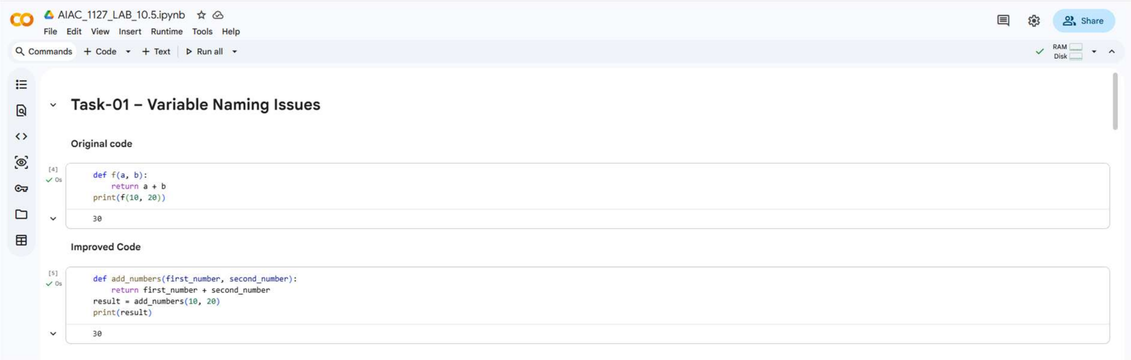
ASSIGNMENT – 10.5

LAB – 10.5 : Code Review and Quality : Using AI to improve code Quality and Readability.

Task – 01: Variable Naming Issues.

Prompt: Review the following Python code and improve it by replacing unclear function and variable names with meaningful and descriptive names. Refactor the code to follow PEP 8 standards and improve readability and maintainability without changing its functionality.

Code & Output:


AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk

Task-01 – Variable Naming Issues

Original code

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

```
30
```

Improved Code

```
def add_numbers(first_number, second_number):
    return first_number + second_number
result = add_numbers(10, 20)
print(result)
```

```
30
```

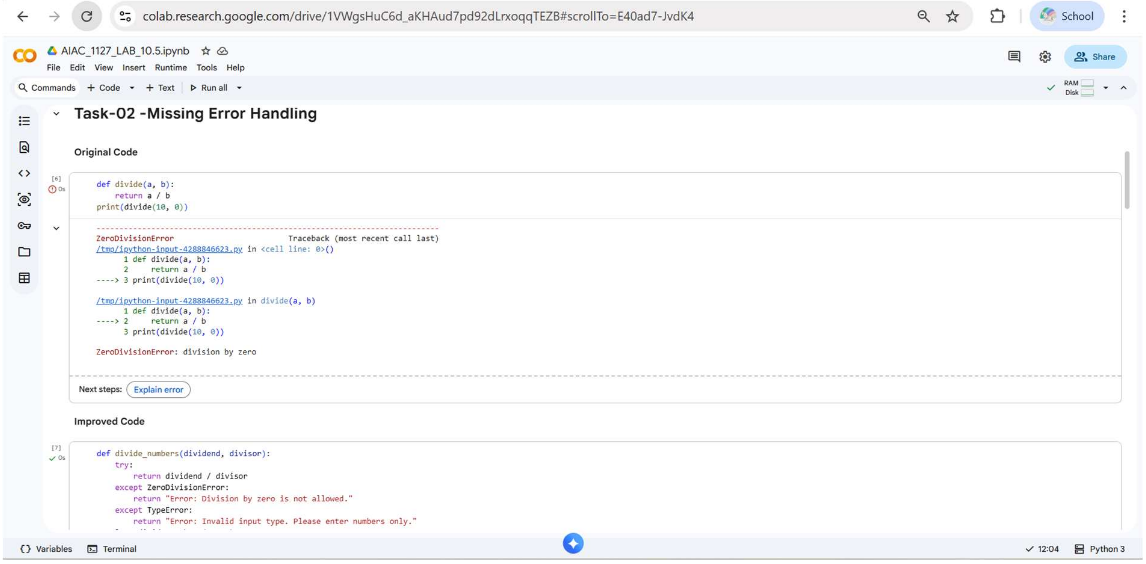
Explanation :

The original code used unclear function and variable names, making it difficult to understand its purpose. AI improved the code by using meaningful names and adding structure, which enhances readability and maintainability.

Task – 02 : Missing Error Handling.

Prompt: Review the following Python code and improve it by adding proper error handling. Handle possible exceptions such as division by zero and invalid input types. Refactor the code to follow PEP 8 standards, use meaningful variable names, and provide clear, user-friendly error messages without changing the core functionality.

Code & Output :


colab.research.google.com/drive/1VWgsHuC6d_aKHAud7pd92dLrxoqqTEZB#scrollTo=E40ad7-JvdK4
School
AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
RAM
Disk
Variables
Terminal
12:04
Python 3

Task-02 -Missing Error Handling

Original Code

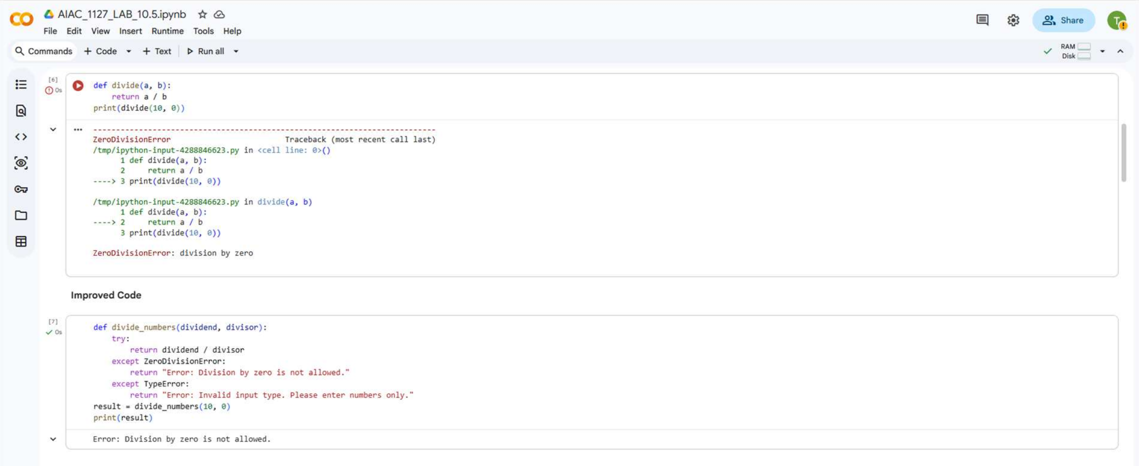
```
[6]
def divide(a, b):
    return a / b
print(divide(10, 0))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
/tmp/ipython-input-4288846623.py in <cell line: 0>()
      1 def divide(a, b):
      2     return a / b
----> 3 print(divide(10, 0))

/tmp/ipython-input-4288846623.py in divide(a, b)
      1 def divide(a, b):
----> 2     return a / b
      3 print(divide(10, 0))

ZeroDivisionError: division by zero
```

Next steps: Explain error

Improved Code

```
[7]
def divide_numbers(dividend, divisor):
    try:
        return dividend / divisor
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Invalid input type. Please enter numbers only."
```


AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
RAM
Disk

```
[6]
def divide(a, b):
    return a / b
print(divide(10, 0))
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
/tmp/ipython-input-4288846623.py in <cell line: 0>()
      1 def divide(a, b):
      2     return a / b
----> 3 print(divide(10, 0))

/tmp/ipython-input-4288846623.py in divide(a, b)
      1 def divide(a, b):
----> 2     return a / b
      3 print(divide(10, 0))

ZeroDivisionError: division by zero
```

Improved Code

```
[7]
def divide_numbers(dividend, divisor):
    try:
        return dividend / divisor
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Invalid input type. Please enter numbers only."
result = divide_numbers(10, 0)
print(result)
```

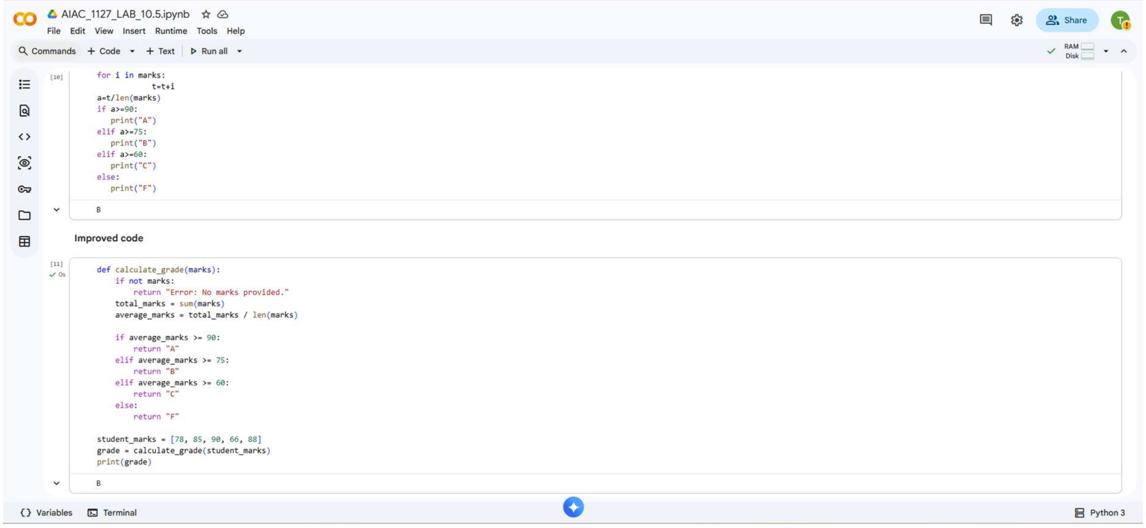
```
Error: Division by zero is not allowed.
```

Explanation : The original code does not handle runtime errors like division by zero, which can cause the program to crash. The improved version adds exception handling to manage errors gracefully and display clear, user-friendly messages. This enhances program reliability, robustness, and overall code quality.

Task – 03 : Student Marks Processing System.

Prompt : Review the following Python program and refactor it to improve readability, structure, and code quality. Follow PEP 8 standards, use meaningful variable and function names, and convert the logic into reusable functions. Add proper input validation, error handling, comments, and a clear docstring. Do not change the core functionality.

Code & Output :


AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Variables Terminal
Python 3

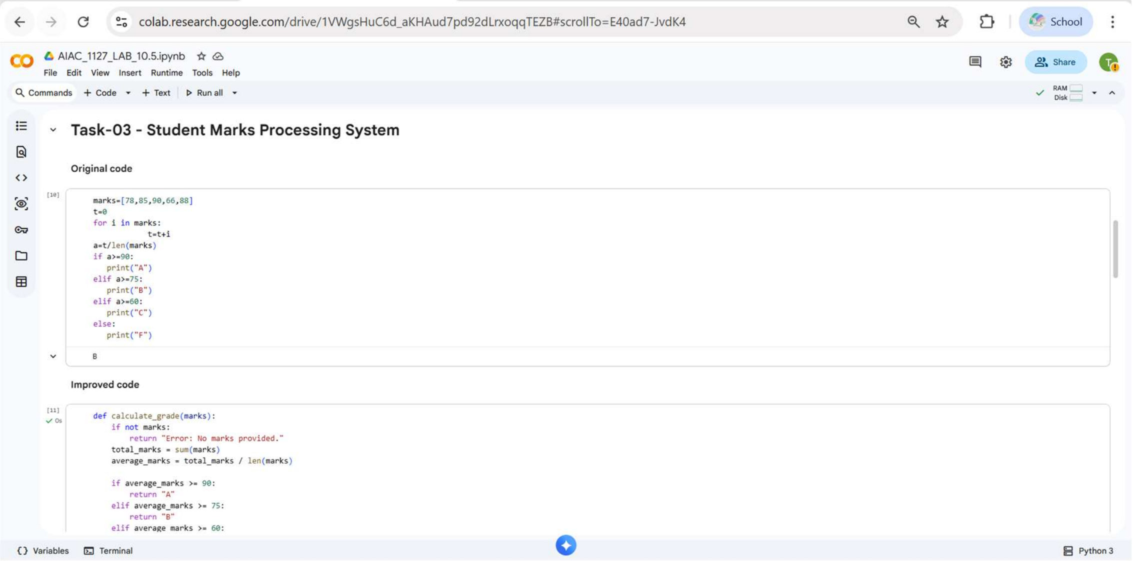
```
for i in marks:
            t=t+i
a=t/len(marks)
if a>=90:
   print("A")
elif a>=75:
   print("B")
elif a>=60:
   print("C")
else:
   print("F")
```

```
B
```

Improved code

```
def calculate_grade(marks):
    if not marks:
        return "Error: No marks provided."
    total_marks = sum(marks)
    average_marks = total_marks / len(marks)

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

student_marks = [78, 85, 90, 66, 88]
grade = calculate_grade(student_marks)
print(grade)
```

```
B
```


colab.research.google.com/drive/1VWgsHuC6d_aKHAud7pd92dLrxoqqTEZB#scrollTo=E40ad7-JvdK4
School
AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Variables Terminal
Python 3

Task-03 - Student Marks Processing System

Original code

```
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")
```

```
B
```

Improved code

```
def calculate_grade(marks):
    if not marks:
        return "Error: No marks provided."
    total_marks = sum(marks)
    average_marks = total_marks / len(marks)

    if average_marks >= 90:
        return "A"
    elif average_marks >= 75:
        return "B"
    elif average_marks >= 60:
```

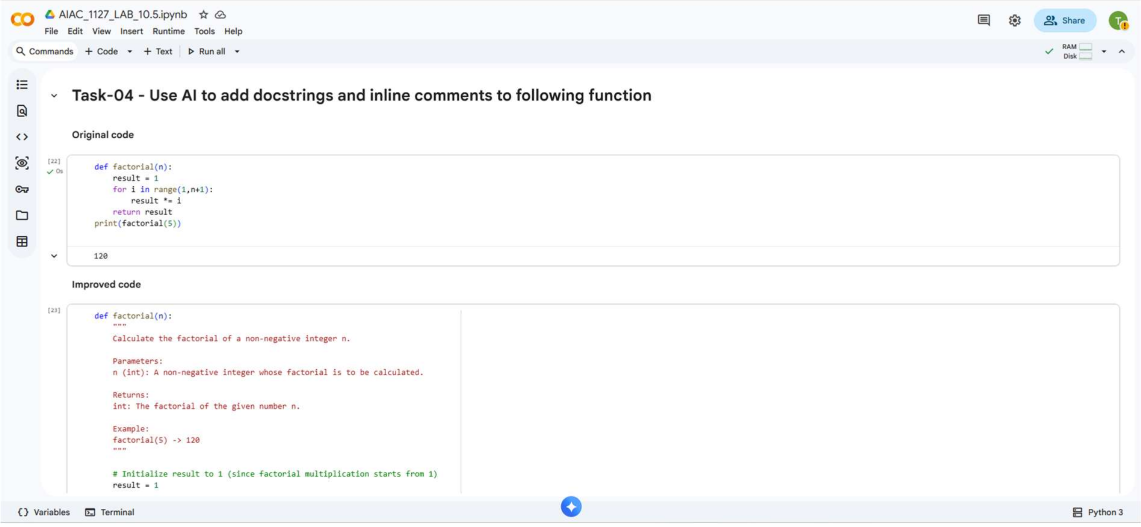
Explanation :

The original program had poor variable naming, no function structure, and lacked input validation, making it difficult to maintain and understand. The refactored version follows PEP 8 standards, uses meaningful names, and organizes the logic into reusable functions with proper validation. This improves readability, maintainability, and overall code quality.

Task – 04: Use AI to add docstrings and inline comments to the following Function.

Prompt : Review the following Python function and enhance it by adding a proper docstring and meaningful inline comments. Ensure the documentation explains the purpose, parameters, return value, and possible exceptions. Follow PEP 8 standards and improve readability without changing the core functionality.

Code & Output :


AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Variables Terminal Python 3

Task-04 - Use AI to add docstrings and inline comments to following function

Original code

[22]

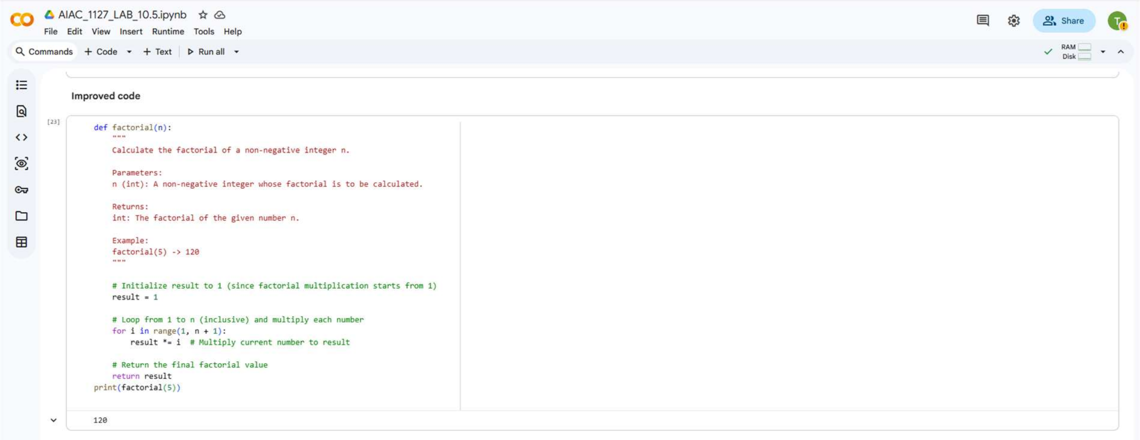
```
def factorial(n):
    result = 1
    for i in range(1,n+1):
        result *= i
    return result
print(factorial(5))
```

```
120
```

Improved code

[23]

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

    Parameters:
    n (int): A non-negative integer whose factorial is to be calculated.

    Returns:
    int: The factorial of the given number n.

    Example:
    factorial(5) -> 120
    """

    # Initialize result to 1 (since factorial multiplication starts from 1)
    result = 1
```


AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share

Improved code

[23]

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

    Parameters:
    n (int): A non-negative integer whose factorial is to be calculated.

    Returns:
    int: The factorial of the given number n.

    Example:
    factorial(5) -> 120
    """

    # Initialize result to 1 (since factorial multiplication starts from 1)
    result = 1

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

    # Return the final factorial value
    return result
print(factorial(5))
```

```
120
```

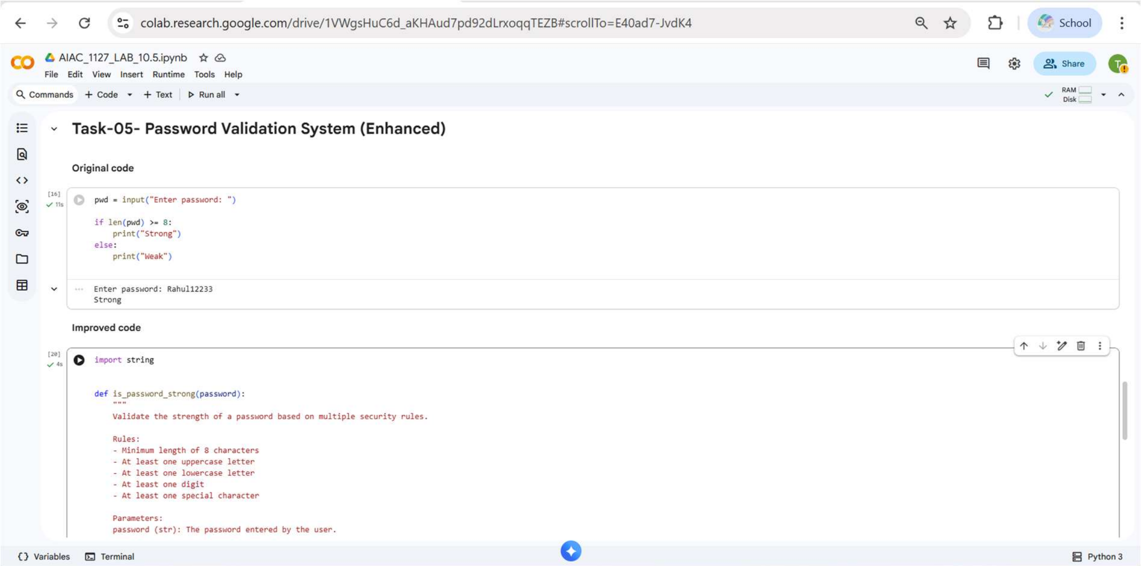
Explanation :

The original function lacked documentation and comments, making it harder to understand its purpose and logic. The improved version adds a clear docstring and inline comments, explaining the functionality, parameters, and return value. This enhances readability, maintainability, and adherence to coding best practices.

Task – 05 : Password Validation System.

Prompt : Refactor the code using meaningful function names, PEP 8 standards, and include a proper docstring with inline comments.

Code & Output :



colab.research.google.com/drive/1VWgsHuC6d_aKHAud7pd92dLrxoqqTEZB#scrollTo=E40ad7-JvdK4
School
AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
Variables
Terminal
Python 3

Task-05- Password Validation System (Enhanced)

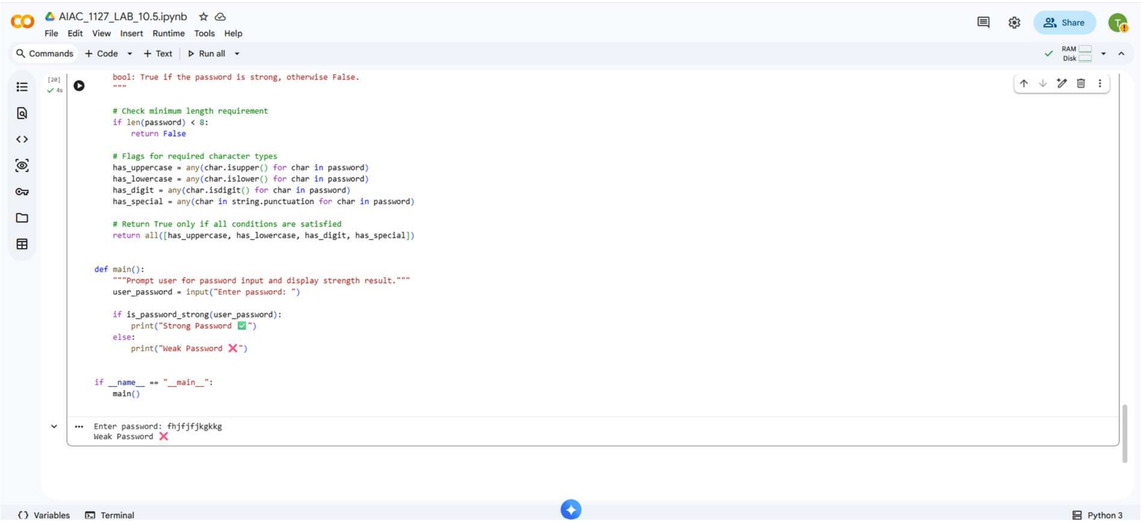
Original code

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

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

```
Enter password: Rahul12233
Strong
```

Improved code

```
import string


def is_password_strong(password):
    """
    Validate the strength of a password based on multiple 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

    Parameters:
    password (str): The password entered by the user.
```



AIAC_1127_LAB_10.5.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
Variables
Terminal
Python 3

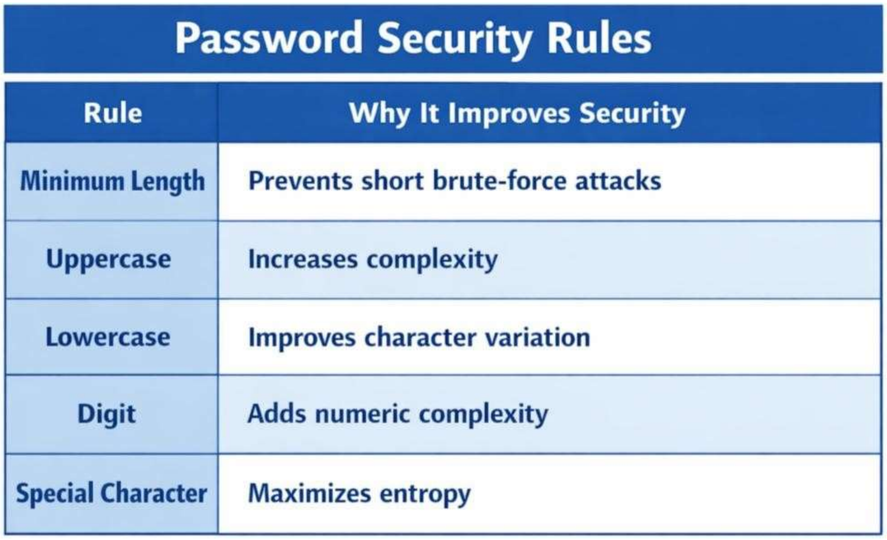
```
    bool: True if the password is strong, otherwise False.
    """

    # Check minimum length requirement
    if len(password) < 8:
        return False

    # Flags for required character types
    has_uppercase = any(char.isupper() for char in password)
    has_lowercase = any(char.islower() for char in password)
    has_digit = any(char.isdigit() for char in password)
    has_special = any(char in string.punctuation for char in password)

    # Return True only if all conditions are satisfied
    return all([has_uppercase, has_lowercase, has_digit, has_special])


def main():
    """Prompt user for password input and display strength result."""
    user_password = input("Enter password: ")

    if is_password_strong(user_password):
        print("Strong Password ✅")
    else:
        print("Weak Password ❌")


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

```
Enter password: fhjfjfjkgkkg
Weak Password ❌
```

Explanation :

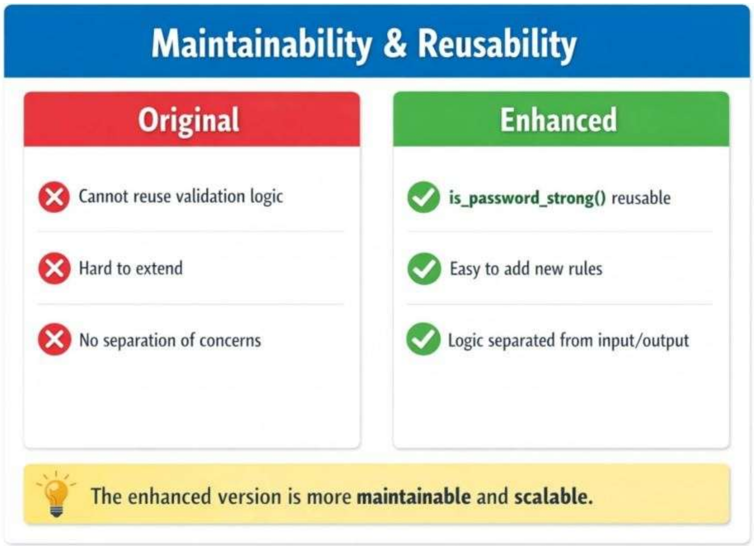
Maintainability & Reusability

- Password logic inside a function
- Can reuse in web apps, login systems
- Easy to update rules



Password Security Rules

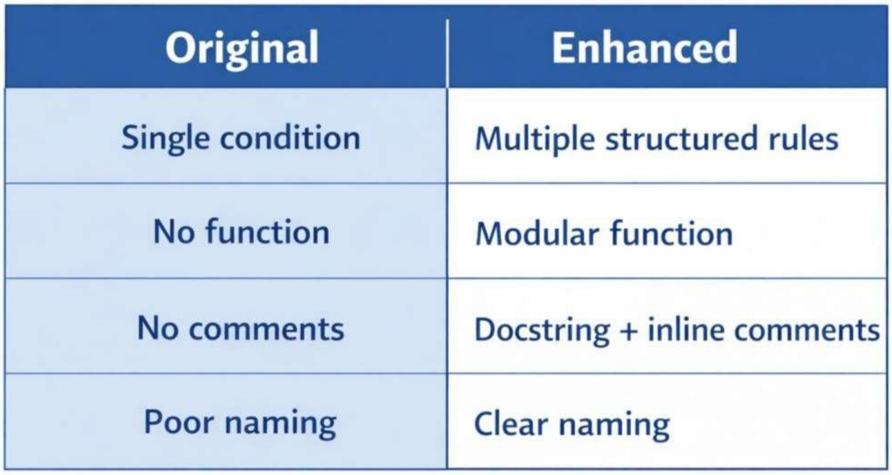
Rule	Why It Improves Security
Minimum Length	Prevents short brute-force attacks
Uppercase	Increases complexity
Lowercase	Improves character variation
Digit	Adds numeric complexity
Special Character	Maximizes entropy



Maintainability & Reusability

Original	Enhanced
✗ Cannot reuse validation logic	✓ **is_password_strong()** reusable
✗ Hard to extend	✓ Easy to add new rules
✗ No separation of concerns	✓ Logic separated from input/output

💡 The enhanced version is more **maintainable** and **scalable.**



Original	Enhanced
Single condition	Multiple structured rules
No function	Modular function
No comments	Docstring + inline comments
Poor naming	Clear naming