

# ASSIGNMENT – 10.5

## AI- ASSISTED-CODING

NAME: DUGYALA NITHYA

HT.NO: 2303A52457

BATCH: 35

### **Task Description -1 : Variable Naming Issues**

Task: Use AI to improve unclear variable names.

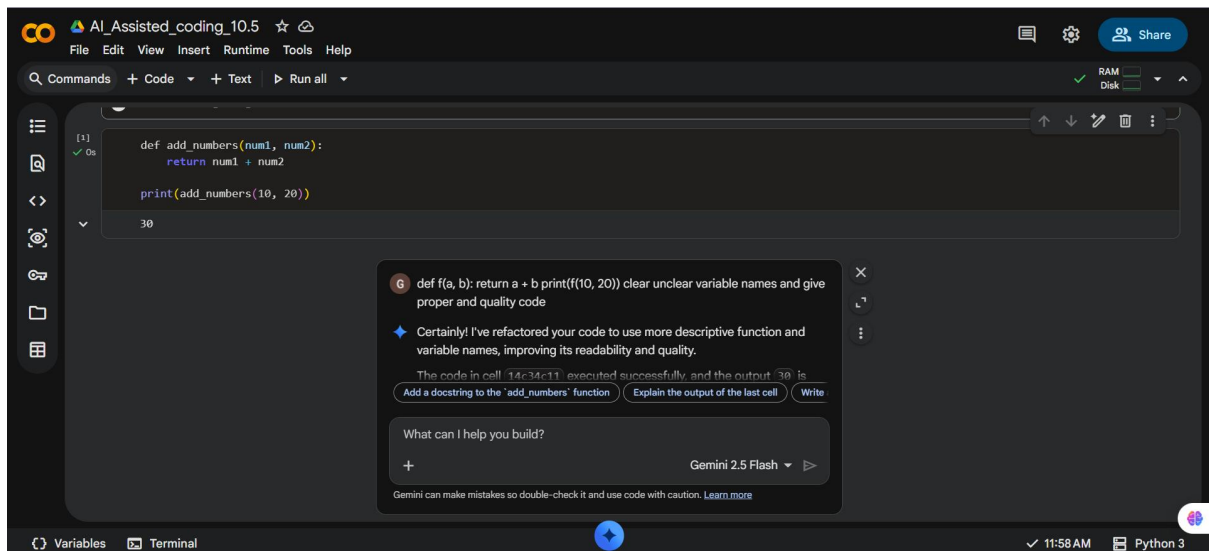
Sample Input Code:

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

Expected Output:

- Code rewritten with meaningful function and variable names.

OUTPUT:



## Task Description -2 : Missing Error Handling

Task: Use AI to add proper error handling.

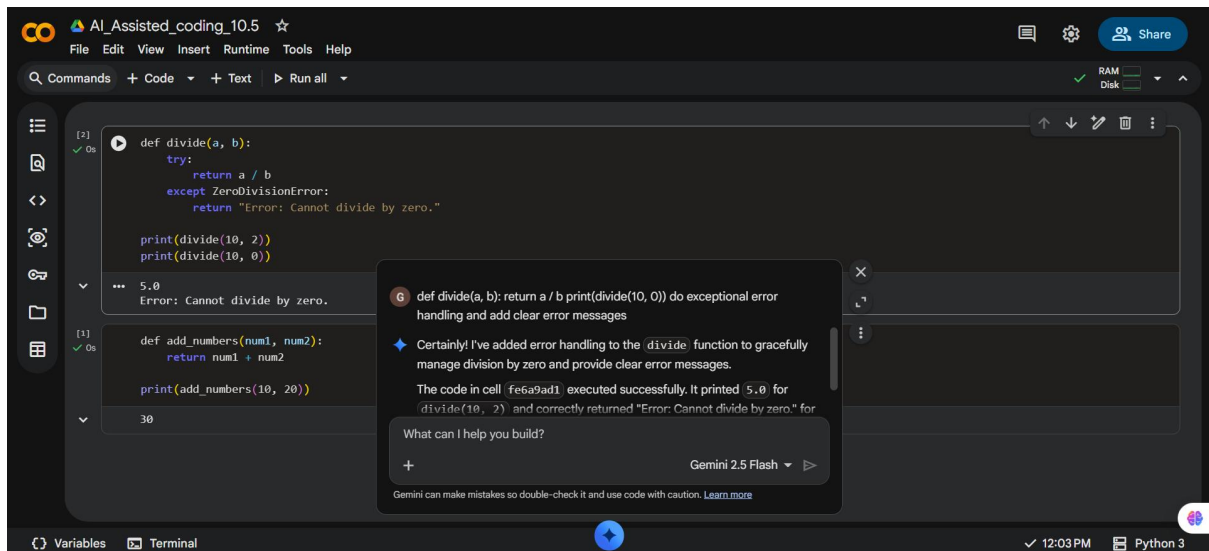
Sample Input Code:

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

Expected Output:

- Code with exception handling and clear error messages

OUTPUT:



## 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")
```

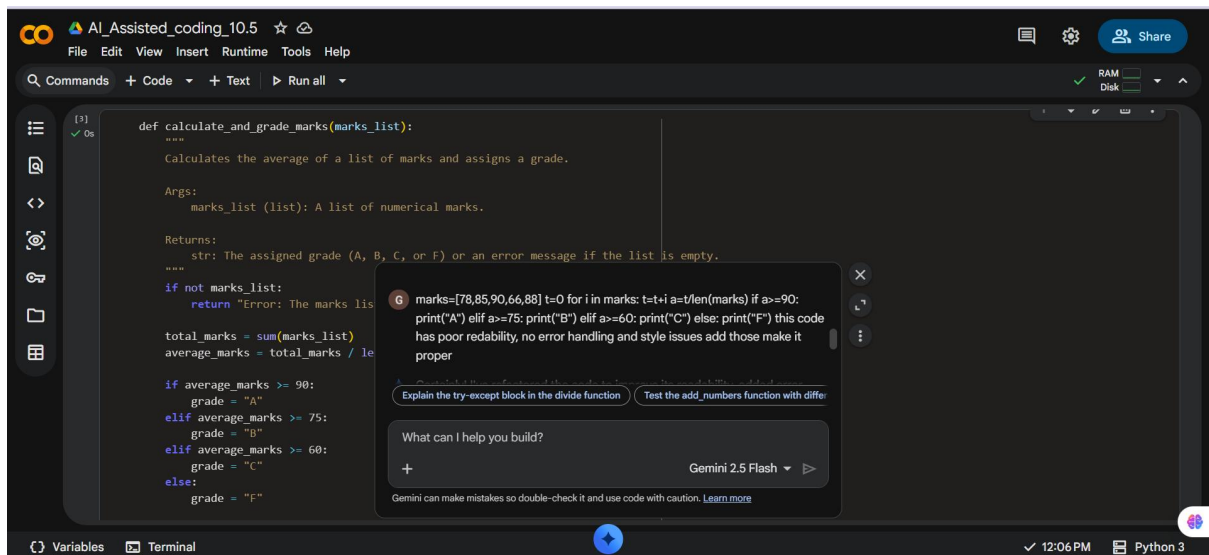
else:

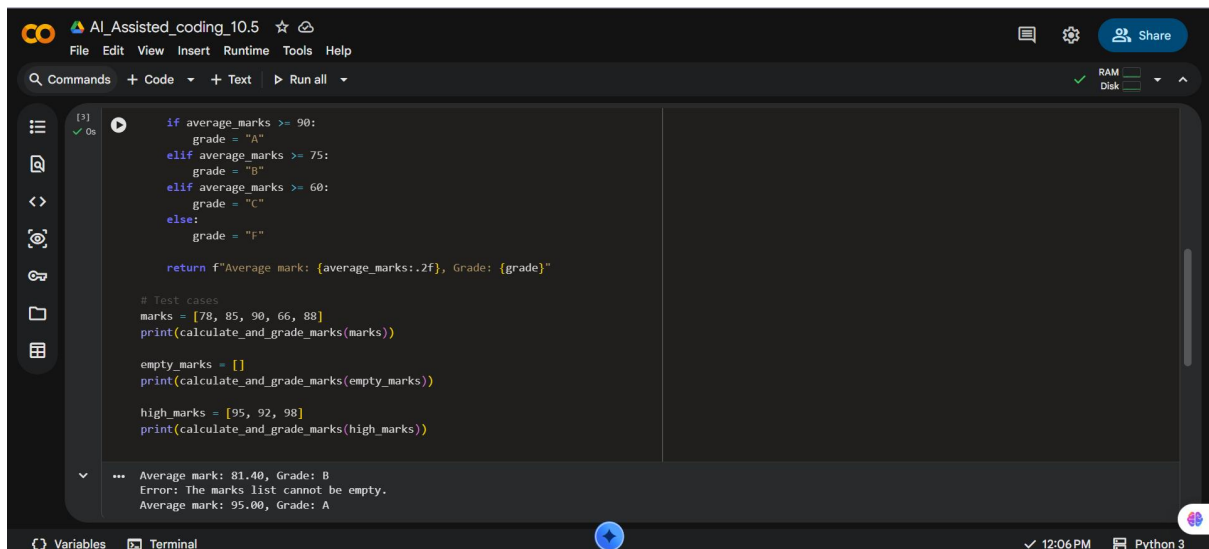
```
    print("F")
```

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.

OUTPUT:





```
[3] 0s
if average_marks >= 90:
    grade = "A"
elif average_marks >= 75:
    grade = "B"
elif average_marks >= 60:
    grade = "C"
else:
    grade = "F"

return f"Average mark: {average_marks:.2f}, Grade: {grade}"

# Test cases
marks = [78, 85, 90, 66, 88]
print(calculate_and_grade_marks(marks))

empty_marks = []
print(calculate_and_grade_marks(empty_marks))

high_marks = [95, 92, 98]
print(calculate_and_grade_marks(high_marks))

... Average mark: 81.40, Grade: B
Error: The marks list cannot be empty.
Average mark: 95.00, Grade: A
```

## 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:

The screenshot shows the 'AI Assisted Coding' IDE interface. The main editor displays a Python function `def factorial(n):` with docstrings and comments. A Gemini chat window is open, showing a prompt to add inline comments and docstrings, and a response confirming the changes. The chat window also includes a 'What can I help you build?' prompt and a 'Gemini 2.5 Flash' model selector.

```
def factorial(n):  
    """  
    Calculates the factorial of a non-negative integer.  
    Args:  
        n (int): The non-negative integer for which to calculate the factorial.  
    Returns:  
        int: The factorial of n.  
    Raises:  
        ValueError: If n is a negative integer.  
    """  
    # Check if the input is a non-negative integer  
    if not isinstance(n, int) or n < 0:  
        raise ValueError("Factorial is defined only for non-negative integers.")  
    # Initialize the result to 1, as 0! = 1  
    result = 1  
    # Iterate from 1 to n (inclusive) to multiply numbers  
    for i in range(1, n + 1):  
        result *= i # Multiply current result by the iterator  
    return result  
  
# Test cases  
print(factorial(5)) # Expected: 120  
print(factorial(0)) # Expected: 1  
# print(factorial(-1)) # This will raise a ValueError  
# print(factorial(3.5)) # This will raise a ValueError
```

The screenshot shows the same IDE interface, but the Gemini chat window is closed. The main editor now displays the completed Python function `def factorial(n):` with test cases. The output of the test cases is shown in the terminal: 120 and 1.

```
def factorial(n):  
    """  
    Calculates the factorial of a non-negative integer.  
    Args:  
        n (int): The non-negative integer for which to calculate the factorial.  
    Returns:  
        int: The factorial of n.  
    Raises:  
        ValueError: If n is a negative integer.  
    """  
    # Check if the input is a non-negative integer  
    if not isinstance(n, int) or n < 0:  
        raise ValueError("Factorial is defined only for non-negative integers.")  
    # Initialize the result to 1, as 0! = 1  
    result = 1  
    # Iterate from 1 to n (inclusive) to multiply numbers  
    for i in range(1, n + 1):  
        result *= i # Multiply current result by the iterator  
    return result  
  
# Test cases  
print(factorial(5)) # Expected: 120  
print(factorial(0)) # Expected: 1  
# print(factorial(-1)) # This will raise a ValueError  
# print(factorial(3.5)) # This will raise a ValueError
```

## 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:

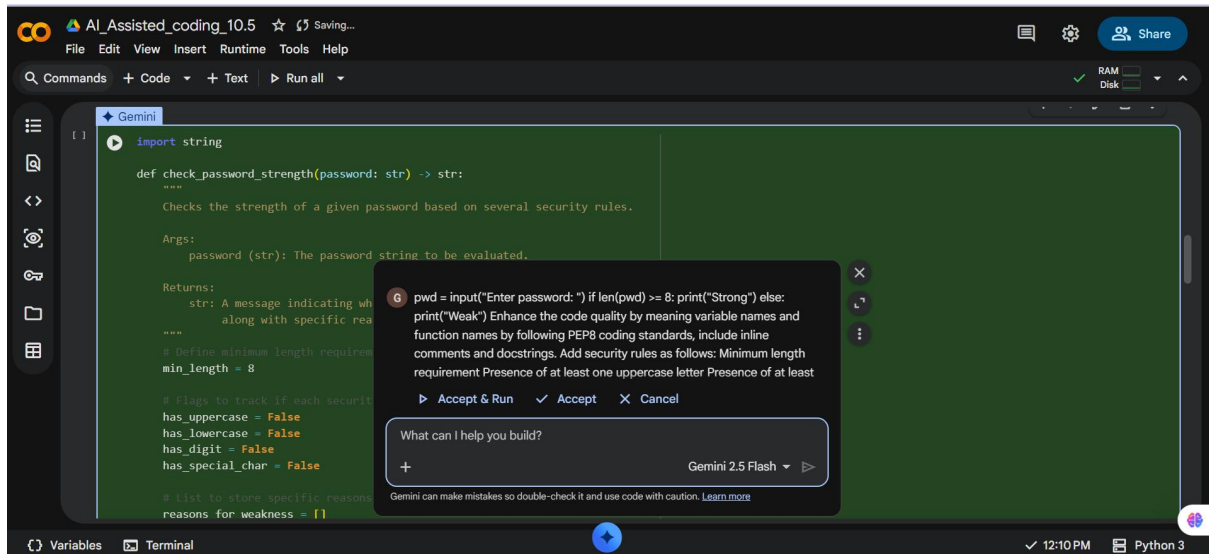
- o Use meaningful variable and function names
- o Follow PEP 8 coding standards
- o Include inline comments and a docstring

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:



The screenshot shows the AI Assisted coding interface with the following code in the editor:

```
import string

def check_password_strength(password: str) -> str:
    """
    Checks the strength of a given password based on several security rules.

    Args:
        password (str): The password string to be evaluated.

    Returns:
        str: A message indicating whether the password is strong or weak,
        along with specific reasons for weakness if it is weak.
    """
    # Define minimum length requirement
    min_length = 8

    # Flags to track if each security criterion is met
    has_uppercase = False
    has_lowercase = False
    has_digit = False
    has_special_char = False

    # List to store specific reasons for weakness
    reasons_for_weakness = []
```

A Gemini chat window is open, displaying the following prompt and response:

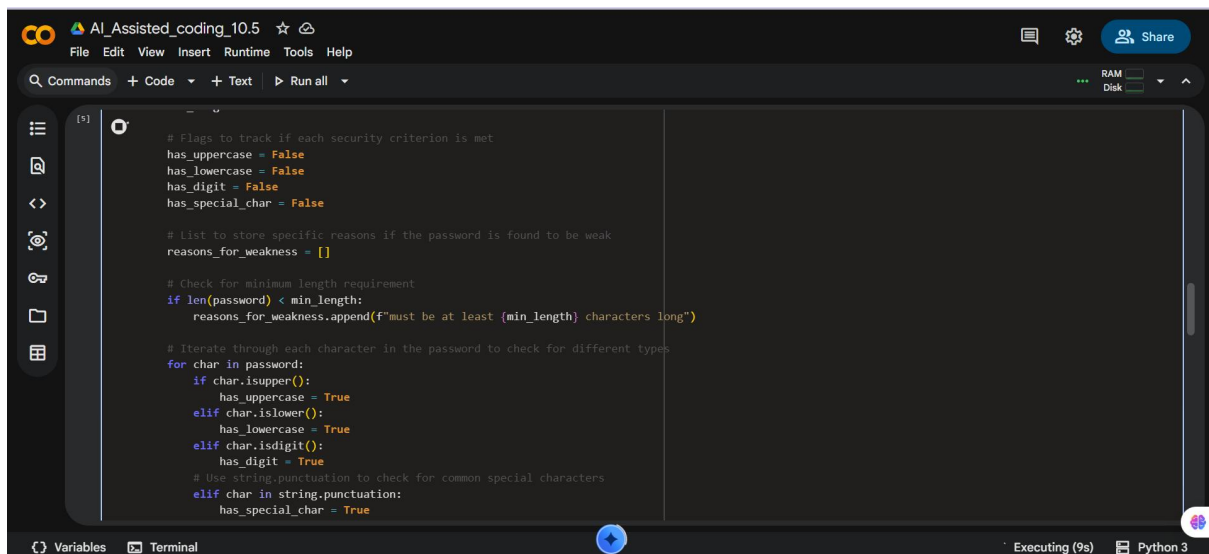
**G** pwd = input("Enter password: ") if len(pwd) >= 8: print("Strong") else: print("Weak") Enhance the code quality by following PEP8 coding standards, include inline comments and docstrings. Add security rules as follows: Minimum length requirement Presence of at least one uppercase letter Presence of at least one lowercase letter Presence of at least one digit Presence of at least one special character

**A** Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

+ Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)



The screenshot shows the AI Assisted coding interface with the following code in the editor:

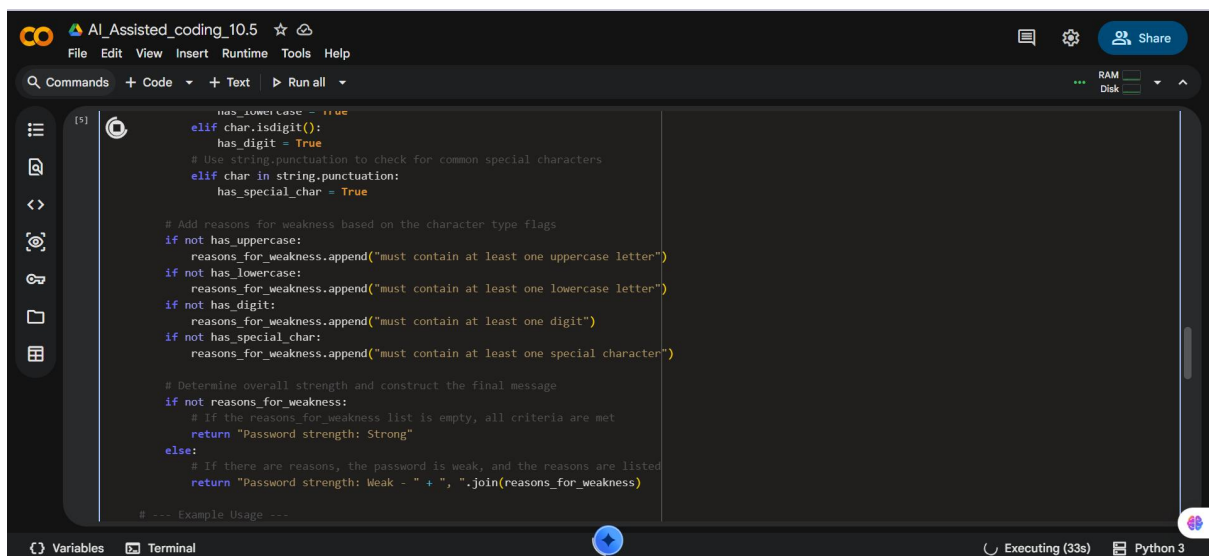
```
# Flags to track if each security criterion is met
has_uppercase = False
has_lowercase = False
has_digit = False
has_special_char = False

# List to store specific reasons if the password is found to be weak
reasons_for_weakness = []

# Check for minimum length requirement
if len(password) < min_length:
    reasons_for_weakness.append(f"must be at least {min_length} characters long")

# Iterate through each character in the password to check for different types
for char in password:
    if char.isupper():
        has_uppercase = True
    elif char.islower():
        has_lowercase = True
    elif char.isdigit():
        has_digit = True
    # Use string.punctuation to check for common special characters
    elif char in string.punctuation:
        has_special_char = True
```

The status bar at the bottom indicates "Executing (9s)" and "Python 3".



The screenshot shows the AI Assisted coding interface with the following code in the editor:

```
has_uppercase = True
elif char.isdigit():
    has_digit = True
# Use string.punctuation to check for common special characters
elif char in string.punctuation:
    has_special_char = True

# Add reasons for weakness based on the character type flags
if not has_uppercase:
    reasons_for_weakness.append("must contain at least one uppercase letter")
if not has_lowercase:
    reasons_for_weakness.append("must contain at least one lowercase letter")
if not has_digit:
    reasons_for_weakness.append("must contain at least one digit")
if not has_special_char:
    reasons_for_weakness.append("must contain at least one special character")

# Determine overall strength and construct the final message
if not reasons_for_weakness:
    # If the reasons_for_weakness list is empty, all criteria are met
    return "Password strength: Strong"
else:
    # If there are reasons, the password is weak, and the reasons are listed
    return "Password strength: Weak - " + ", ".join(reasons_for_weakness)

# --- Example Usage ---
```

The status bar at the bottom indicates "Executing (33s)" and "Python 3".



AI\_Assisted\_coding\_10.5

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

[5] 1m

not has uppercase:  
reasons\_for\_weakness.append("must contain at least one uppercase letter")  
if not has lowercase:  
reasons\_for\_weakness.append("must contain at least one lowercase letter")  
if not has digit:  
reasons\_for\_weakness.append("must contain at least one digit")  
if not has special char:  
reasons\_for\_weakness.append("must contain at least one special character")  
  
# Determine overall strength and construct the final message  
if not reasons\_for\_weakness:  
# If the reasons for weakness list is empty, all criteria are met  
return "Password strength: Strong"  
else:  
# If there are reasons, the password is weak, and the reasons are listed  
return "Password strength: Weak - " + ", ".join(reasons\_for\_weakness)  
  
# --- Example Usage ---  
# Prompt the user to enter a password  
user\_password = input("Enter password: ")  
  
# Call the function to check the password strength and print the result  
print(check\_password\_strength(user\_password))

Enter password: adR234  
Password strength: Weak - must be at least 8 characters long, must contain at least one special character

Variables Terminal

12:22 PM Python 3