

# 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:

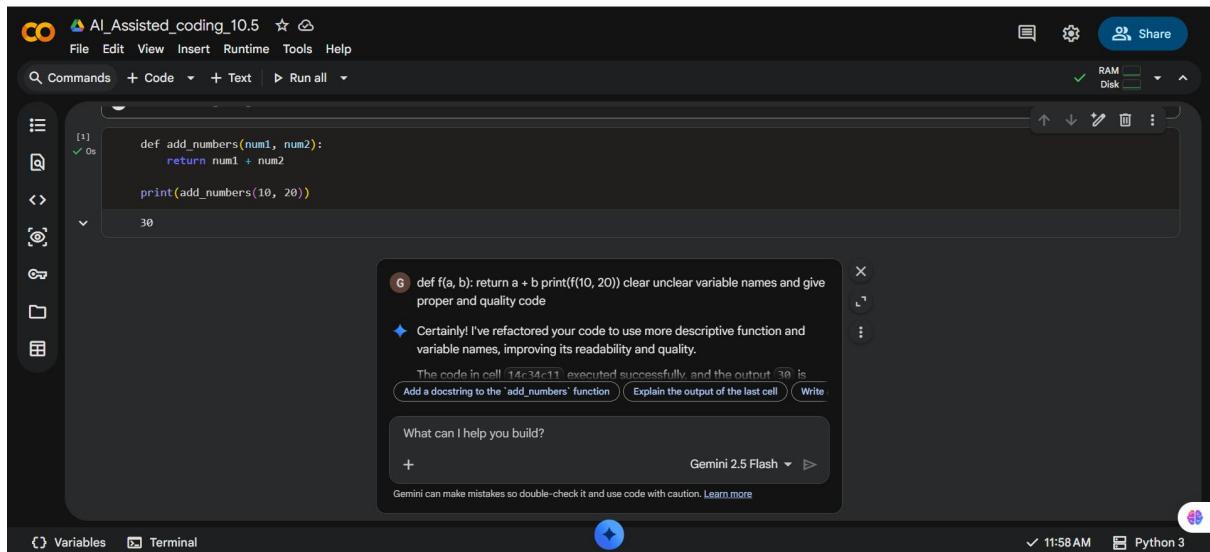
```
def f(a, b):  
    return a + b
```

```
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:

The screenshot shows a Jupyter Notebook interface with two code cells. Cell [2] contains a function `divide` that handles division by zero. Cell [1] contains a function `add_numbers`. A sidebar on the right displays AI-generated feedback from Gemini 2.5 Flash, suggesting improvements for error handling and providing context about the execution of the code.

```

[2]: 
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    
print(divide(10, 2))
print(divide(10, 0))

... 5.0
Error: Cannot divide by zero.

[1]: 
def add_numbers(num1, num2):
    return num1 + num2

print(add_numbers(10, 20))

30

```

**G** def divide(a, b): return a / b print(divide(10, 0)) do exceptional error handling and add clear error messages  
**◆** Certainly! I've added error handling to the `divide` function to gracefully manage division by zero and provide clear error messages.  
The code in cell `fe6a9ad1` executed successfully. It printed `5.0` for `divide(10, 2)` and correctly returned "Error: Cannot divide by zero." for `divide(10, 0)`.

What can I help you build?

## 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:

The screenshot shows the AI-Assisted Coding interface with the following details:

- Title:** AI\_Assisted\_coding\_10.5
- Code Editor Content:**

```
def calculate_and_grade_marks(marks_list):
    """
    Calculates the average of a list of marks and assigns a grade.

    Args:
        marks_list (list): A list of numerical marks.

    Returns:
        str: The assigned grade (A, B, C, or F) or an error message if the list is empty.

    if not marks_list:
        return "Error: The marks list is empty."
    total_marks = sum(marks_list)
    average_marks = total_marks / len(marks_list)

    if average_marks >= 90:
        grade = "A"
    elif average_marks >= 75:
        grade = "B"
    elif average_marks >= 60:
        grade = "C"
    else:
        grade = "F"
```
- AI Feedback (Sidebar):**

G 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") this code has poor readability, no error handling and style issues add those make it proper
- Bottom Status:** Gemini 2.5 Flash
- Bottom Icons:** Variables, Terminal, 12:06 PM, Python 3

The screenshot shows the AI-Assisted Coding interface. The code in the editor is:

```
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))
```

The output window shows:

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

At the bottom, there are tabs for Variables and Terminal.

## 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 interface with a Python file named 'factorial.py'. The code defines a factorial function with a comprehensive docstring and inline comments. A sidebar on the right shows AI-generated feedback and a prompt for further assistance.

```
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
```

The screenshot shows the AI-Assisted Coding interface with the same Python file. A 'Table of contents' sidebar is visible on the left, listing the function's name and its docstring. The main area displays the original code.

```
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
```

## 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 a dark theme. A sidebar on the left contains icons for file operations like Open, Save, and Run. The main area displays Python code for checking password strength. A tooltip from the AI tool 'Gemini' provides feedback on the code, suggesting improvements like using PEP8 coding standards and adding security rules. A modal window from Gemini asks, "What can I help you build?", with options to accept or cancel. The status bar at the bottom shows the time as 12:10 PM and the Python version as Python 3.

```
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.
    """
    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 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
```

This screenshot shows the continuation of the password strength checking code. The AI tool 'Gemini' has provided further suggestions and corrections, such as adding a docstring and improving variable names. The status bar indicates the code is executing. The Python version is Python 3.

```
# 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
```

The final screenshot shows the completed password strength checking code. The AI tool 'Gemini' has provided the final touches, including the addition of a docstring and the final return statement. The status bar shows the code is executing for 33 seconds. The Python version is Python 3.

```
"""
    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.
    """
min_length = 8

def check_password_strength(password: str) -> str:
    has_uppercase = False
    has_lowercase = False
    has_digit = False
    has_special_char = False

    reasons_for_weakness = []

    if len(password) < min_length:
        reasons_for_weakness.append(f"must be at least {min_length} characters long")

    for char in password:
        if char.isupper():
            has_uppercase = True
        elif char.islower():
            has_lowercase = True
        elif char.isdigit():
            has_digit = True
        elif char in string.punctuation:
            has_special_char = True

    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")

    if not reasons_for_weakness:
        return "Password strength: Strong"
    else:
        return "Password strength: Weak - " + ", ".join(reasons_for_weakness)

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

The screenshot shows a code editor interface with the following details:

- Title Bar:** AI Assisted Coding 10.5
- Menu Bar:** File, Edit, View, Insert, Runtime, Tools, Help
- Toolbar:** Commands, Code, Text, Run all
- Code Area:** Displays Python code for checking password strength. The code includes logic to check for uppercase, lowercase, digits, and special characters, and constructs a message indicating the strength based on these findings.
- Output Area:** Shows the result of running the code with the input "adR234", which outputs "Password strength: Weak - must be at least 8 characters long, must contain at least one special character".
- Bottom Status Bar:** Variables, Terminal, Python 3, 12:22PM

```
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 ---
# 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))
```