AI Assisted Coding

Assignment-10.5

Name: Bethi Mounika

Roll No: 2303A52196

Batch: 35

Task Description #1 – Variable Naming Issues

Task: Use AI to improve unclear variable names.

Sample Input Code:

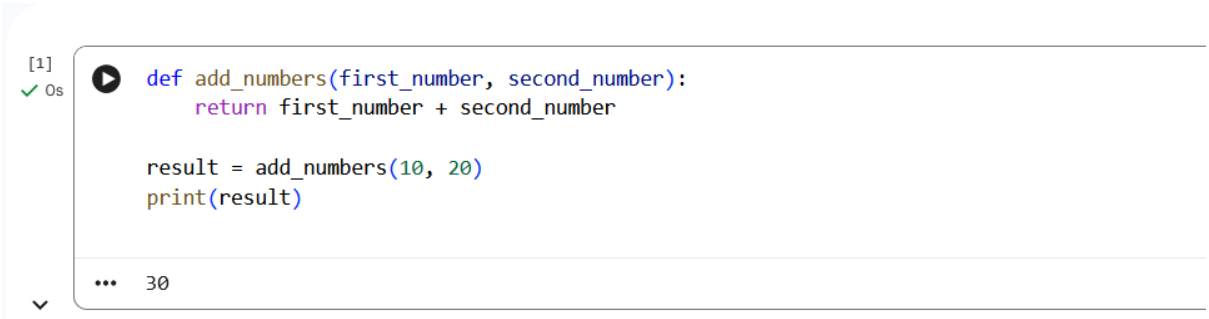
def f(a, b):

return a + b

print(f(10, 20))

Expected Output:

• Code rewritten with meaningful function and variable names.



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

```
def divide_numbers(numerator, denominator):
    try:
        return numerator / denominator
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Please provide valid numeric inputs."

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

```
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.

Code:

```
"""
Student Marks Processing System
This program takes student marks as input,
calculates the average, and assigns a grade.
"""

def calculate_grade(marks_list):
    """
    Calculates and returns the grade based on average marks.
    """
    if not marks_list:
        return "Error: No marks entered."

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

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


# Taking input from the user
try:
    marks_input = input("Enter student marks
    marks = [float(mark) for mark in marks_
    grade = calculate_grade(marks)
```

How can I install Python libraries? | Load data from Google Drive | Show an example of training

What can I help you build?

Gemini 2.5 Flash

```
# Taking input from the user
try:
    marks_input = input("Enter student marks separated by spaces: ")
    marks = [float(mark) for mark in marks_input.split()]

    grade = calculate_grade(marks)
    print("Grade:", grade)

except ValueError:
    print("Error: Please enter only numeric values.")
```

```
Enter student marks separated by spaces: 78 85 90 66 88
Grade: B
```

Task Description #4: Use AI to add docstrings and inline comments to the following function.

def factorial(n):

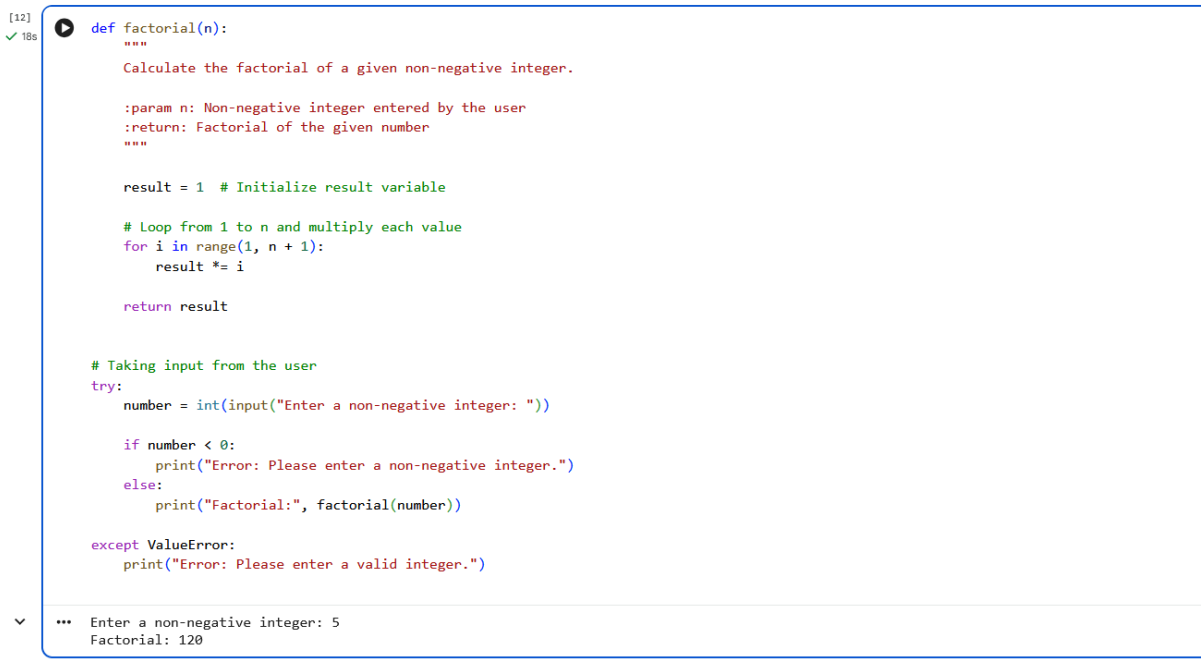
result = 1

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

result *= i

return result

Code:



```
[12]
✓ 18s
def factorial(n):
    """
    Calculate the factorial of a given non-negative integer.

    :param n: Non-negative integer entered by the user
    :return: Factorial of the given number
    """

    result = 1  # Initialize result variable

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

    return result


# Taking input from the user
try:
    number = int(input("Enter a non-negative integer: "))

    if number < 0:
        print("Error: Please enter a non-negative integer.")
    else:
        print("Factorial:", factorial(number))

except ValueError:
    print("Error: Please enter a valid integer.")
```

```
Enter a non-negative integer: 5
Factorial: 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:

Code:

```
[9]
✓ 7s
import re

def validate_password(password):
    """
    Validates 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

    :param password: Password string entered by the user
    :return: Validation result message
    """

    # Check minimum length
    if len(password) < 8:
        return "Weak Password: Must be at least 8 characters long."

    # Check for uppercase letter
    if not re.search(r"[A-Z]", password):
        return "Weak Password: Must include at least one uppercase letter."

    # Check for lowercase letter
    if not re.search(r"[a-z]", password):
        return "Weak Password: Must include at least one lowercase letter."

    # Check for digit
    if not re.search(r"[0-9]", password):
        return "Weak Password: Must include at least one digit."
```

```
    :return: Validation result message
    """

    # Check minimum length
    if len(password) < 8:
        return "Weak Password: Must be at least 8 characters long."

    # Check for uppercase letter
    if not re.search(r"[A-Z]", password):
        return "Weak Password: Must include at least one uppercase letter."

    # Check for lowercase letter
    if not re.search(r"[a-z]", password):
        return "Weak Password: Must include at least one lowercase letter."

    # Check for digit
    if not re.search(r"[0-9]", password):
        return "Weak Password: Must include at least one digit."

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

    return "Strong Password"


# User input
user_password = input("Enter password: ")

# Validate password
result = validate_password(user_password)
print(result)
```

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