AI ASSISTED CODING

LAB-10.5

Adavelli Harika

2303A51376

Batch-11

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.**

Prompt:

#Rewrite the below code to optimise it and improve the readability.

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

Given Code and Output:

```
#Rewrite the below code to optimise it and improve the readability.
def add_numbers(a, b):
    """
    Docstring for add_numbers

    :param a: Description
    :param b: Description
    :return: Description"""
    return a + b  # This line adds two numbers and returns the result
result = add_numbers(10, 20) # This line calls the add_numbers function with 10 and 20 as arguments
print(result) # This line prints the result to the console
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE        Python

PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/Desktop/A
lab10.py
30
```

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))

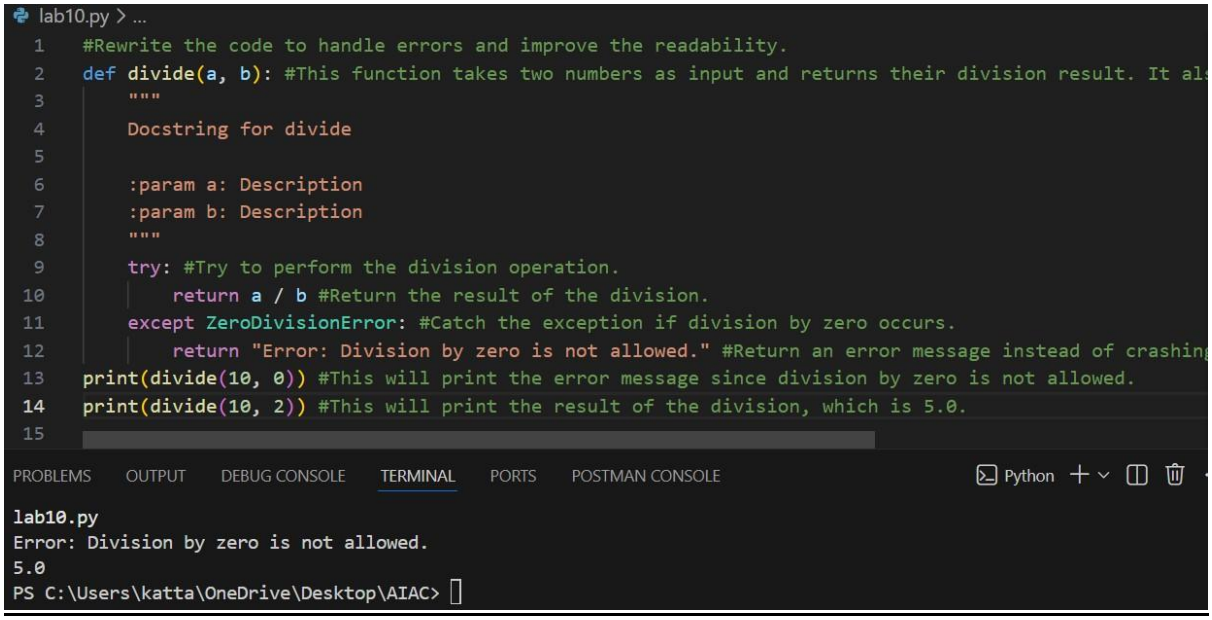
Expected Output:

• Code with exception handling and clear error messages

Prompt:

#Rewrite the code to handle errors and improve the readability.

Given Code and 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.**

Prompt:

#Rewrite the below code to follow PEP 8 standards, add meaningful variable names, functions, and comments and add basic input validation and documentation.

Given Code and Output:

```
#Rewrite the below code to follow PEP 8 standards, add meaningful variable names, functions, and comments and add basic input validation and docume
def calculate_average(marks_list):
    """
    Calculate the average of a list of marks.

    :param marks_list: A list of numerical marks
    :return: The average mark
    """
    if not marks_list: # Check if the list is empty
        raise ValueError("The list of marks is empty.") # Raise an error if there are no marks to calculate the average

    total = sum(marks_list) # Calculate the total sum of the marks
    average = total / len(marks_list) # Calculate the average by dividing the total by the number of marks
    return average # Return the calculated average

def main():
    """Main function to execute the program.
    """
    try: # Try block to catch any potential errors during input and calculation
        marks_input = input("Enter the marks separated by commas: ") # Prompt the user to input marks separated by commas
        marks_list = [float(mark.strip()) for mark in marks_input.split(",")] # Convert the input string into a list of floats, stripping any extra
        average_mark = calculate_average(marks_list) # Calculate the average mark using the calculate_average function
        print(f"The average mark is: {average_mark:.2f}") # Print the average mark formatted to 2 decimal places

        if average_mark >= 90: # Determine the grade based on the average mark and print it
            print("Grade: A")
        elif average_mark >= 75: # Check if the average mark is between 75 and 89.99 and print grade B
            print("Grade: B")
        elif average_mark >= 60: # Check if the average mark is between 60 and 74.99 and print grade C
            print("Grade: C")
        else: # If the average mark is below 60, print grade F
            print("Grade: F")
    except ValueError as e: # Catch any ValueError exceptions that may occur during input conversion or average calculation
        print(f"Input error: {e}")
if __name__ == "__main__":   # Check if the script is being run directly (instead of imported as a module) and call the main function
    main()
```

```
Enter the marks separated by commas: 70,60,85,40
The average mark is: 63.75
Grade: C
```

Task Description #4: Use AI to add docstrings and inline comments to the following function. def factorial(n):

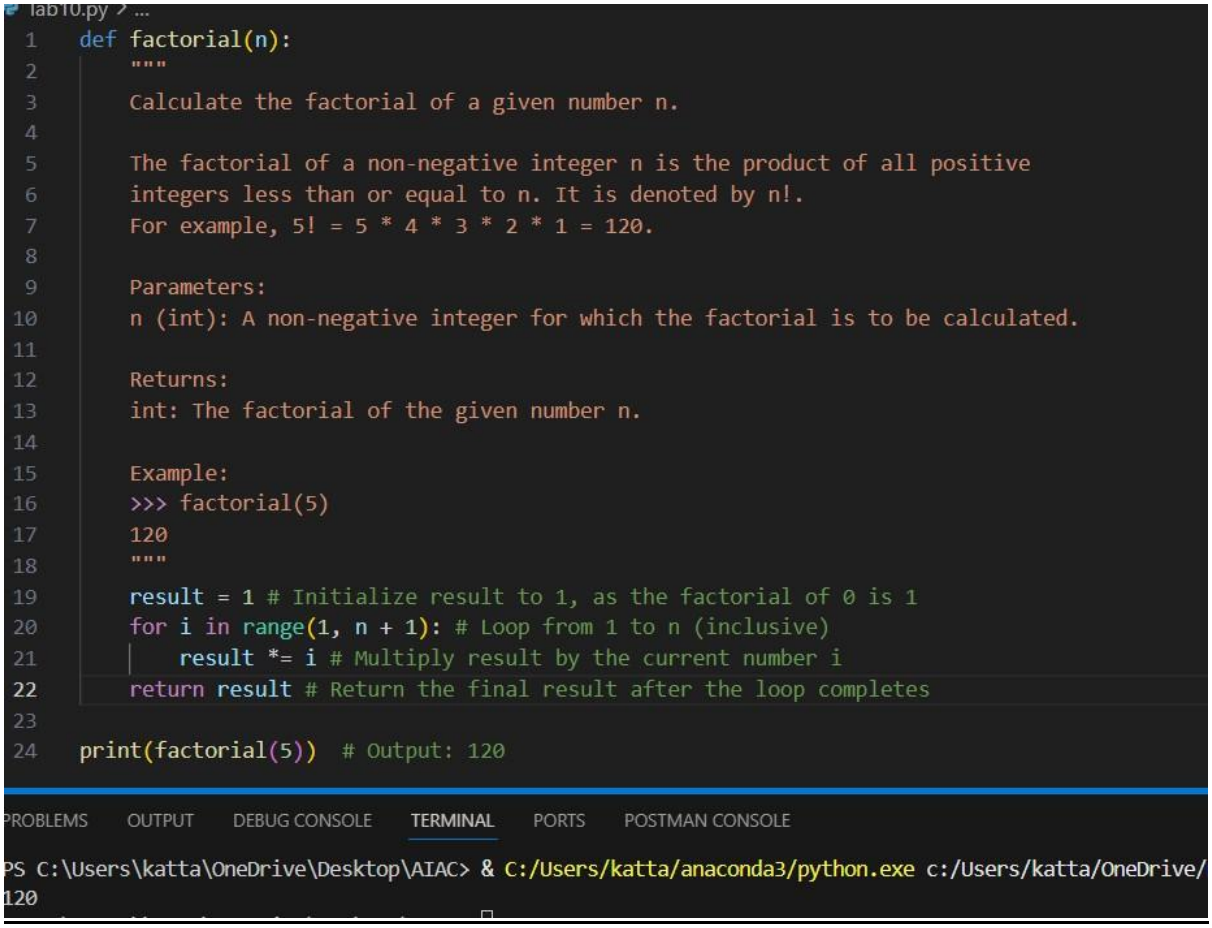
result = 1 for i in

range(1,n+1):

result *= i return

result

Given Code and Output:



```
lab10.py > ...
def factorial(n):
    """
    Calculate the factorial of a given number n.

    The factorial of a non-negative integer n is the product of all positive
    integers less than or equal to n. It is denoted by n!.
    For example, 5! = 5 * 4 * 3 * 2 * 1 = 120.

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

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

    Example:
    >>> factorial(5)
    120
    """
    result = 1 # Initialize result to 1, as the factorial of 0 is 1
    for i in range(1, n + 1): # Loop from 1 to n (inclusive)
        result *= i # Multiply result by the current number i
    return result # Return the final result after the loop completes

print(factorial(5))  # Output: 120
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/
120
```

<u>Task Description #5: Password Validation System (Enhanced)</u> 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

Prompt:

"""Enhance the below password validation code to check for the following criteria:

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

Use meaningful variable and function names

Follow PEP 8 coding standards

Include inline comments and a docstring

Imrove code readability and maintainability"""

Given Code and Output:

```
"""Enhance the below password validation code to check for the following criteria:
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
Use meaningful variable and function names
Follow PEP 8 coding standards
Include inline comments and a docstring
Imrove code readability and maintainability"""
pwd = input("Enter password: ")
"""This code checks the strength of a password based on specific criteria:
1. Minimum length requirement (at least 8 characters)
2. Presence of at least one uppercase letter
3. Presence of at least one lowercase letter
4. Presence of at least one digit
5. Presence of at least one special character from the set !@#$%^&*()-+
The code evaluates the password against these criteria and prints "Strong" if all conditions are met, otherwise it prints "Weak".
The code follows PEP 8 coding standards and includes meaningful variable names and inline comments for better readability and maintainability.
"""
if len(pwd) >= 8:
    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False
    special_characters = "!@#$%^&*()-+"
    for char in pwd:
        if char.isupper():
            has_upper = True
        elif char.islower():
            has_lower = True
        elif char.isdigit():
            has_digit = True
        elif char in special_characters:
            has_special = True
    if all([has_upper, has_lower, has_digit, has_special]):
        print("Strong")
    else:
        print("Weak")
else:
    print("Weak")
```

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/
Enter password: asdf@efg
Weak
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/
Enter password: A1@dfgh%dc
Strong
```