

2303A52416

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.

## Code

```
def add_two_numbers(first_number, second_number):  
    """
```

This function takes two numbers as input  
and returns their sum.

```
"""
```

```
    return first_number + second_number
```

```
# Example usage
```

```
result = add_two_numbers(10, 20)
```

```
print("The sum is:", result)
```

The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. A central window displays Python code in 'main.py':

```

1 def add_two_numbers(first_number, second_number):
2     """
3         This function takes two numbers as input
4         and returns their sum.
5     """
6     return first_number + second_number
7
8
9 # Example usage
10 result = add_two_numbers(10, 20)
11
12 print("The sum is:", result)
13

```

Below the code editor is a terminal window showing the output of the program:

```

input
The sum is: 30

...Program finished with exit code 0
Press ENTER to exit console.

```

A sidebar advertisement for auth0 is visible on the left.

## 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

Code

```
def divide_numbers(numerator, denominator):
```

=====

This function divides two numbers safely.

It handles division errors like dividing by zero.

=====

try:

```
    result = numerator / denominator
    return result
```

except ZeroDivisionError:

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

except TypeError:

```
    return "Error: Please enter valid numeric values."
```

```
# Example usage
output = divide_numbers(10, 0)

print("Result:", output)
```

The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. A banner for 'auth0' is visible. The main area has tabs for 'Run', 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and language selection ('Python 3'). The code editor contains 'main.py' with the provided Python script. The terminal window below shows the output of running the program, which includes a note about division by zero and the resulting error message.

```
main.py
1 def divide_numbers(numerator, denominator):
2     """
3         This function divides two numbers safely.
4         It handles division errors like dividing by zero.
5     """
6     try:
7         result = numerator / denominator
8         return result
9     except ZeroDivisionError:
10        return "Error: Division by zero is not allowed."
11    except TypeError:
12        return "Error: Please enter valid numeric values."
13
14
15
16 # Example usage
17 output = divide_numbers(10, 0)
18
19
20 print("Result:", output)
21
```

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

...Program finished with exit code 0
Press ENTER to exit console.
```

### 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.

```
def calculate_total_and_average(marks_list):
```

```
    """
```

Calculate the total and average marks of a student.

Parameters:

marks\_list (list): List of student marks

Returns:

tuple: Total marks and average marks

```
    """
```

```
total_marks = sum(marks_list)
```

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

```
return total_marks, average_marks
```

```
def determine_grade(average_marks):
```

```
    """
```

Determine the grade based on average marks.

Parameters:

average\_marks (float): Average marks of the student

Returns:

str: Grade (A, B, C, or F)

```
    """
```

```
if average_marks >= 90:
```

```
    return "A"
```

```
elif average_marks >= 75:
```

```
    return "B"
```

```
elif average_marks >= 60:
```

```
    return "C"
```

```
else:
```

```
    return "F"
```

```
def student_marks_processing_system(marks_list):
```

```
    """
```

Process student marks and display total, average, and grade.

Parameters:  
marks\_list (list): List of marks entered for the student  
=====

```
# Input validation
if not marks_list:
    print("Error: Marks list cannot be empty.")
    return

if not all(isinstance(mark, (int, float)) for mark in marks_list):
    print("Error: All marks must be numeric values.")
    return

# Calculate total and average
total, average = calculate_total_and_average(marks_list)

# Determine grade
grade = determine_grade(average)

# Display results
print("Total Marks:", total)
print("Average Marks:", average)
print("Grade:", grade)

# Example usage
student_marks = [78, 85, 90, 66, 88]

student_marks_processing_system(student_marks)
```

The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. A banner from auth0 says: 'You asked, we delivered! Our Free Plan now includes a Custom Domain, 5 Actions, and 25,000 MAUs.' Below the sidebar are links for 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', and 'Privacy', followed by the copyright notice '© 2016 - 2026 GDB Online'.

The main area shows a Python script named 'main.py' with the following code:

```
1 def calculate_total_and_average(marks_list):
2     """
3         Calculate the total and average marks of a student.
4     """
5     Parameters:
6     marks_list (list): List of student marks
7
8     Returns:
9     tuple: Total marks and average marks
10    """
11    total_marks = sum(marks_list)
12    average_marks = total_marks / len(marks_list)
13    return total_marks, average_marks
14
15
16 def determine_grade(average_marks):
17     """
18         Determine the grade based on average marks.
19     """
20     Parameters:
21     average_marks (float): Average marks of the student
22
23     Returns:
24     str: Grade (A, B, C, or F)
```

The terminal window below shows the output of running the program with input 'Total Marks: 407' and 'Average Marks: 81.4'. It prints 'Grade: B' and ends with '...Program finished with exit code 0'.

This screenshot is identical to the one above, showing the same Python script 'main.py' and its execution results. The code defines a function 'determine\_grade' that takes an average mark as input and returns a grade ('A', 'B', 'C', or 'F') based on a range check. The terminal output shows the grade 'B' for an average mark of 81.4.

```

main.py
46     print("Error: Marks list cannot be empty.")
47     return
48
49     if not all(isinstance(mark, (int, float)) for mark in marks_list):
50         print("Error: All marks must be numeric values.")
51         return
52
53     # Calculate total and average
54     total, average = calculate_total_and_average(marks_list)
55
56     # Determine grade
57     grade = determine_grade(average)
58
59     # Display results
60     print("Total Marks:", total)
61     print("Average Marks:", average)
62     print("Grade:", grade)
63
64
65 # Example usage
66 student_marks = [78, 85, 90, 66, 88]
67
68 student_marks_processing_system(student_marks)
69

```

input

```

Total Marks: 407
Average Marks: 81.4
Grade: B

...Program finished with exit code 0
Press ENTER to exit console.

```

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

```

```

def factorial(n):
    """

```

Calculate the factorial of a given number.

Factorial of a number n is the product of all positive integers from 1 to n.

Parameters:

n (int): A non-negative integer

Returns:

int: Factorial value of the number n

"""

```

# Initialize result as 1 (factorial of 0 is 1)
result = 1

```

```

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

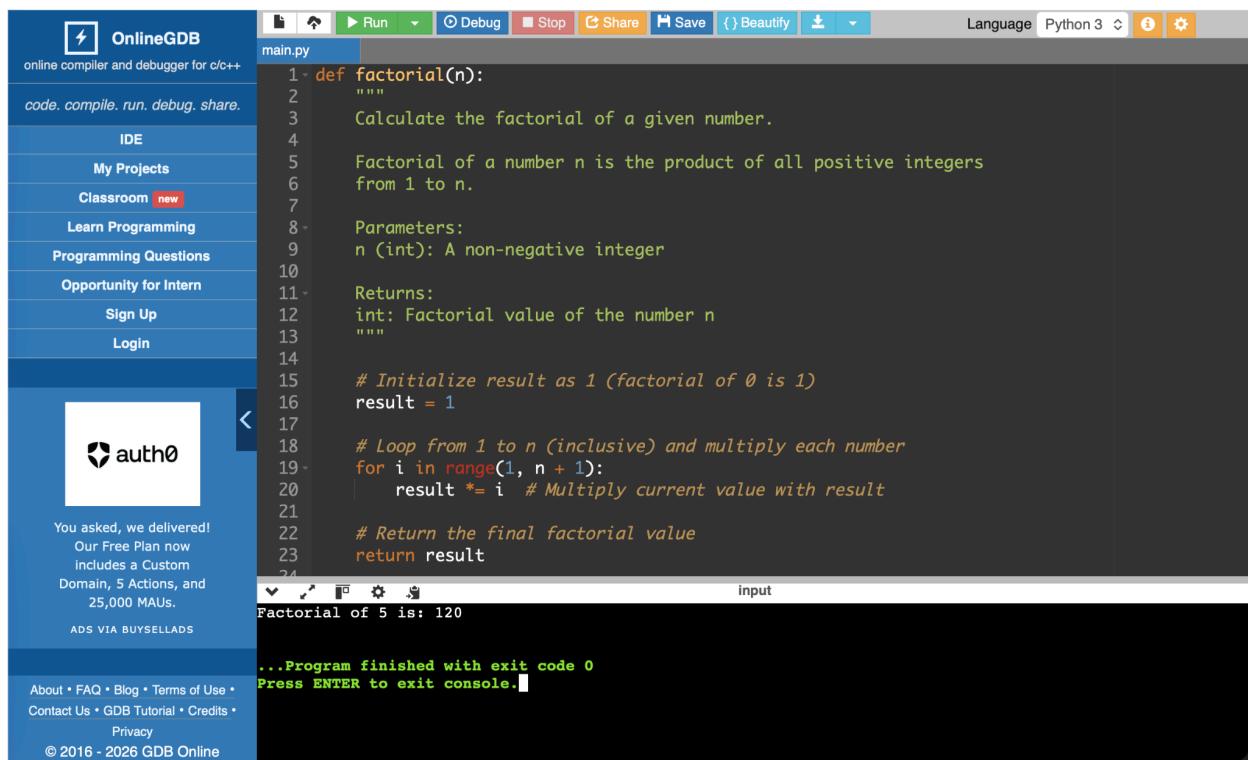
# Return the final factorial value
return result

```

```

# Example usage
print("Factorial of 5 is:", factorial(5))

```



The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links for 'My Projects', 'Classroom' (which is highlighted in red), 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. Below that is an 'auth0' advertisement. At the bottom of the sidebar are links for 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', 'Privacy', and '© 2016 - 2026 GDB Online'.

The main workspace has a toolbar at the top with icons for file operations, run, debug, stop, share, save, and beautify. The language is set to 'Python 3'. The code editor window contains the following Python script:

```

1 def factorial(n):
2     """
3         Calculate the factorial of a given number.
4
5     Factorial of a number n is the product of all positive integers
6     from 1 to n.
7
8     Parameters:
9     n (int): A non-negative integer
10
11    Returns:
12    int: Factorial value of the number n
13    """
14
15    # Initialize result as 1 (factorial of 0 is 1)
16    result = 1
17
18    # Loop from 1 to n (inclusive) and multiply each number
19    for i in range(1, n + 1):
20        result *= i # Multiply current value with result
21
22    # Return the final factorial value
23    return result
24

```

The output window below shows the results of running the script:

```

Factorial of 5 is: 120
...Program finished with exit code 0
Press ENTER to exit console.

```

The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. A banner from 'auth0' is visible, stating 'You asked, we delivered! Our Free Plan now includes a Custom Domain, 5 Actions, and 25,000 MAUs.' Below the sidebar are links for 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', and 'Privacy', along with the copyright notice '© 2016 - 2026 GDB Online'.

The main area shows a Python script named 'main.py' with the following code:

```

1 Factorial of a number n is the product of all positive integers
2 from 1 to n.
3
4 Parameters:
5     n (int): A non-negative integer
6
7 Returns:
8     int: Factorial value of the number n
9     """
10
11     # Initialize result as 1 (factorial of 0 is 1)
12     result = 1
13
14     # Loop from 1 to n (inclusive) and multiply each number
15     for i in range(1, n + 1):
16         result *= i # Multiply current value with result
17
18     # Return the final factorial value
19     return result
20
21
22 # Example usage
23 print("Factorial of 5 is:", factorial(5))
24
25
26
27
28

```

The 'input' field contains 'Factorial of 5 is: 120'. The output window shows '...Program finished with exit code 0' and 'Press ENTER to exit console.'

### 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.

```
import re
```

#### 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.

```
import re
```

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

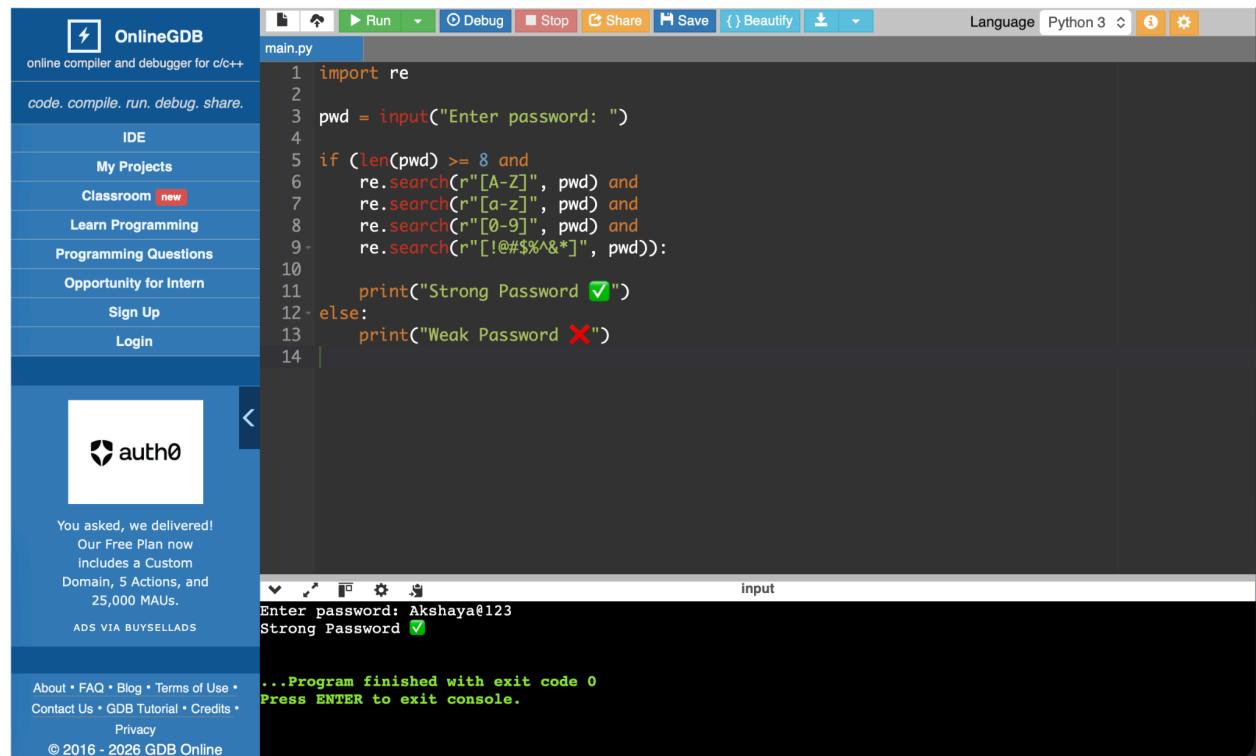
```
if (len(pwd) >= 8 and
```

```

re.search(r"[A-Z]", pwd) and
re.search(r"[a-z]", pwd) and
re.search(r"[0-9]", pwd) and
re.search(r"[@#$%^&*]", pwd)):

    print("Strong Password ✓")
else:
    print("Weak Password ✗")

```



The screenshot shows the OnlineGDB IDE interface. On the left, there's a sidebar with various links like 'My Projects', 'Classroom', 'Learn Programming', etc. The main workspace contains a Python script named 'main.py' with the following code:

```

1 import re
2
3 pwd = input("Enter password: ")
4
5 if (len(pwd) >= 8 and
6     re.search(r"[A-Z]", pwd) and
7     re.search(r"[a-z]", pwd) and
8     re.search(r"[0-9]", pwd) and
9     re.search(r"[@#$%^&*]", pwd)):
10
11     print("Strong Password ✓")
12 else:
13     print("Weak Password ✗")
14

```

Below the code editor, there's a terminal window showing the output of running the script. It prompts for a password ('Enter password: Akshaya@123') and then prints 'Strong Password ✓'. At the bottom of the terminal, it says '...Program finished with exit code 0' and 'Press ENTER to exit console.'