

AI Assisted Coding Assignment 10.5

Name: V.Harivamsh

Hall ticket no: 2303A51266

Batch no: 19

Task 1: Variable Naming Issues

Prompt:

Refactor the following Python code to improve unclear variable names and function naming while maintaining the same functionality and following PEP 8 standards.

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

Code & Output:

The screenshot shows the VS Code interface. The left pane displays the code editor with `assignment_10.5.py` open. The code contains a function `f` and a call to it. The right pane shows the terminal window with the output of running the script.

```
assignment_10.5.py X kk.py
AI_Assistant_coding > assignment_10.5.py > ...
1  # Task 1: Variable Naming Issues
2
3  # Refactor the following Python code to improve unclear variable names and function
4  # naming while maintaining the same functionality and following PEP 8 standards.
5 ...
6  def f(a, b):
7      return a + b
8
9  print(f(10, 20))
10 ...
11 ...
12
13 def add_numbers(num1, num2):
14     """Returns the sum of two numbers."""
15     return num1 + num2
16
17 print(add_numbers(10, 20))

PROBLEMS OUTPUT TERMINAL PORTS
> < TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_10.5.py"
● 30
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The original code uses short and unclear names like **f**, **a**, and **b**, which makes it hard to understand the purpose of the function. In the improved version, these are replaced with clear names like `add_numbers`, `first_number`, and `second_number`. This makes the code easier to read and understand at first glance. It also follows proper naming conventions, making the code cleaner and easier to maintain without changing how it works.

Task 2: Missing Error Handling

Prompt:

Enhance the following Python code by adding proper exception handling and meaningful error messages while following PEP 8 standards. `def divide(a, b): return a / b` `print(divide(10, 0))`

Code & Output:

The screenshot shows a code editor interface with several tabs at the top: `assignment_10.5.py`, `assignment_11.3.py`, and `kk.py`. The active tab is `assignment_10.5.py`. The code in the editor is as follows:

```
1  # Task 2: Missing Error Handling
2
3  # Enhance the following Python code by adding proper exception handling and
4  # meaningful error messages while following PEP 8 standards.
5  """
6  def divide(a, b):
7      return a / b
8
9  print(divide(10, 0))
10 """
11
12
13 def divide_numbers(numerator, denominator):
14     """Returns the result of dividing the numerator by the denominator."""
15     try:
16         result = numerator / denominator
17         return result
18     except ZeroDivisionError:
19         return "Error: Cannot divide by zero. Please provide a non-zero denominator."
20
21 print(divide_numbers(10, 0))
```

Below the code editor is a terminal window titled `TERMINAL`. The terminal output is:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_10.5.py"
● Error: Cannot divide by zero. Please provide a non-zero denominator.
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The original code does not check for division by zero, which can cause the program to crash at runtime. The improved version adds a `try-except` block to handle this situation safely. It also uses clear variable names, making the code easier to understand. This makes the

program more reliable, avoids unexpected crashes, and provides a better user experience. Proper error handling is important for writing real-world, production-quality software.

Task 3: Student Marks Processing System

Prompt:

Refactor the given student marks program to improve readability, follow PEP 8 standards, include meaningful variable names, modular functions, documentation, and basic validation.

```
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")
```

Code & Output:

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with file icons for 'assignment_10.5.py', 'assignment_11.3.py', and 'kk.py'. The main area displays the following Python code:

```
1 # Task 3: Student Marks Processing System
2 # Refactor the given student marks program to improve readability, follow PEP 8
3 # standards, include meaningful variable names, modular functions, documentation,
4 # and basic validation.
5 """
6 marks = [78, 85, 90, 66, 88]
7 t = 0
8 for i in marks:
9     t += i
10 avg = t / len(marks)
11 if avg >= 90:
12     print("A")
13 elif avg >= 75:
14     print("B")
15 elif avg >= 60:
16     print("C")
17 else:
18     print("F")
19 """
20 def calculate_average(marks):
21     """Calculate the average of a list of marks."""
22     total_marks = sum(marks)
23     average_mark = total_marks / len(marks)
24     return average_mark
25 def determine_grade(average_mark):
26     """Determine the grade based on the average mark."""
27     if average_mark >= 90:
28         return "A"
29     elif average_mark >= 75:
30         return "B"
31     elif average_mark >= 60:
32         return "C"
33     else:
34         return "F"
35 def main():
36     """Main function to process student marks and determine the grade."""
37     marks = [78, 85, 90, 66, 88]
38     average_mark = calculate_average(marks)
39     grade = determine_grade(average_mark)
40     print(f"The average mark is: {average_mark:.2f}")
41     print(f"The grade is: {grade}")
42 if __name__ == "__main__":
43     main()
```

Below the code editor is a terminal window tab labeled 'TERMINAL'. The terminal output shows the execution of the script and its results:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_10.5.py"
● The average mark is: 81.40
The grade is: B
○ PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The original code is not well structured and uses unclear variable names, and it also does not include proper documentation. The improved version organizes the logic into a separate function and uses meaningful variable names. It also follows PEP 8 formatting rules, making the code cleaner. Using the built-in `sum()` function makes the code shorter and easier to read. Overall, this version is more efficient, maintainable, and reusable.

Task 4: Adding Docstrings and Inline Comments

Prompt:

Enhance the factorial function by adding a proper docstring and meaningful inline comments.

```
def factorial(n):
```

```
    result = 1
```

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

```
        result *= i
```

```
    return result
```

Code & Output:

The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'assignment_10.5.py' (which is currently active), 'assignment_11.3.py', and 'kk.py'. Below the tabs, the code for 'assignment_10.5.py' is displayed. The code defines a 'factorial' function with a docstring explaining it returns the factorial of a given number n. It uses a loop to calculate the result by multiplying it with each number from 1 to n. An example usage is shown at the bottom, followed by a call to print(factorial(5)) which outputs 120. Below the code editor is a terminal window titled 'TERMINAL'. The terminal shows the command 'python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_10.5.py"' being run, and the output '120' is displayed.

```
AI_Assistant_coding > assignment_10.5.py > ...
1 # Task 4: Adding Docstrings and Inline Comments
2 # Enhance the factorial function by adding a proper
3 # docstring and meaningful inline comments.
4 def factorial(n):
5     result = 1
6     for i in range(1, n+1):
7         result *= i
8     return result
9 def factorial(n):
10     """Returns the factorial of a given number n."""
11     result = 1 # Initialize result to 1, as factorial of 0 is 1
12     for i in range(1, n + 1): # Loop from 1 to n (inclusive)
13         result *= i # Multiply result by the current number
14     return result # Return the final factorial value
15 # Example usage
16 print(factorial(5)) # Output: 120

PROBLEMS OUTPUT TERMINAL PORTS ... | ⚡ X
> ✎ TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_10.5.py"
● 120
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The AI-enhanced function includes a clear docstring that explains the purpose of the function and its parameters. It also adds simple inline comments to describe each step in the logic. The variable names are improved to make their meaning clear. All these changes make the code easier to read and better documented.

Task 5: Password Validation System (Enhanced)

Prompt:

Enhance and refactor the password validation system by adding multiple security checks and improving structure according to PEP 8 standards.

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

if len(pwd) >= 8:
    print("Strong")

else:
    print("Weak")
```

Code & Output:

The screenshot shows the Visual Studio Code interface. The left pane displays the Python file `assignment_10.5.py` with the following code:

```
1 # Task 5: Password Validation System (Enhanced)
2 # Enhance and refactor the password validation system by adding multiple
3 # security checks and improving structure according to PEP 8 standards.
4 ...
5 pwd = input("Enter password: ")
6 if len(pwd) >= 8:
7     print("Strong")
8 else:
9     print("Weak")
10 ...
11 import re
12 def validate_password(password):
13     """Validates the strength of a password based on multiple criteria."""
14     if len(password) < 8:
15         return "Weak: Password must be at least 8 characters long."
16     if not re.search("[A-Z]", password):
17         return "Weak: Password must contain at least one uppercase letter."
18     if not re.search("[a-z]", password):
19         return "Weak: Password must contain at least one lowercase letter."
20     if not re.search("[0-9]", password):
21         return "Weak: Password must contain at least one digit."
22     if not re.search("[!@#$%^&*(),.?':{}|<>]", password):
23         return "Weak: Password must contain at least one special character."
24     return "Strong: Your password is strong."
25 # Example usage
26 password_input = input("Enter password: ")
27 validation_result = validate_password(password_input)
28 print(validation_result)
```

The right pane shows the terminal window with the output of running the script:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "C:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_10.5.py"
Enter password: N@ni123
Weak: Password must be at least 8 characters long.
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

Explanation:

The improved program adds several password security rules, such as checking for uppercase letters, lowercase letters, numbers, and special characters. The logic is organized into a separate function with proper documentation. Compared to the original version, this one is more secure and easier to understand. It is also better structured, making it easier to maintain and update in the future.