Lab Assignment-10.5

Mohammed Awez Ali

2303A51767
Batch-12

Lab 10 – Code Review and Quality: Using AI to Improve Code
Quality and Readability

Task Description #1 – Variable Naming Issues

Task: Use AI to improve unclear variable names.

Prompt: rewrite the above code with meaningful function and
variable names, with documentation and inline suggestions, use pydoc
to generate documentation

Code:

```python
# Lab 10: Functions
# def f(a, b):
#     return a + b
# print(f(10, 20))
#rewrite the above code with meaningful function and variable names, with documentation and inline suggestions, use pydoc to generate documentation
def add_numbers(num1, num2):
    """
    This function takes two numbers as input and returns their sum.

    Parameters:
    num1 (int or float): The first number to be added.
    num2 (int or float): The second number to be added.

    Returns:
    int or float: The sum of the two numbers.

    Example:
    >>> add_numbers(10, 20)
    30
    >>> add_numbers(5.5, 4.5)
    10.0
    >>> add_numbers(-1, 1)
    0
    """
    # Calculate the sum of the two numbers
    result = num1 + num2 # Add num1 and num2 to get the result
    return result # Return the calculated result
print(add_numbers(10, 20)) # Example usage of the add_numbers function to compute the sum of 10 and 20
```

Output:

```
eDrive/Desktop/AI Assistant coding/lab10.py"
The sum of 10 and 20 is: 30
```

Explaination: The function `add_numbers` takes two numbers as input and returns their sum. It includes a docstring that explains the purpose of the function, its parameters, return value, and provides example usage. Inline comments are added to clarify the steps involved in calculating the sum.

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))

Prompt: Handle the above code with exception handling, with documentation and inline suggestions, use pydoc to generate documentation

Code:

```
31  #def divide(a, b):
32  # return a / b
33  # print(divide(10, 0))
34
35  #handle the above code with exception handling, with documentation and inline suggestions, use pydoc to generate documentation
36  def divide_numbers(num1, num2):
37
38      """
39      This function takes two numbers as input and returns the result of dividing the first number by the second number.
40
41      Parameters:
42      num1 (int or float): The numerator.
43      num2 (int or float): The denominator.
44
45      Returns:
46      int or float: The result of the division if successful, or a string message if division by zero is attempted.
47
48      Example:
49      >>> divide_numbers(10, 2)
50      5.0
51      >>> divide_numbers(10, 0)
52      'Error: Division by zero is not allowed.'
53      >>> divide_numbers(5.5, 2.5)
54      2.2
55      """
56      try:
57          # Attempt to perform the division
58          result = num1 / num2 # Divide num1 by num2 to get the result
59          return result # Return the calculated result
60      except ZeroDivisionError: # Catch the exception if division by zero is attempted
61          return 'Error: Division by zero is not allowed.' # Return an error message if division by zero occurs
62  print(divide_numbers(10, 0)) # Example usage of the divide_numbers function to demonstrate handling division by zero
63
```

Output:

```
The result of dividing 10 by 0 is: Error: Division by zero is not allowed.
```

Explaination: Thefunction `divide_numbers` performs division of two numbers while handling the potential error of division by zero. It uses a try-except block to catch the `ZeroDivisionError` and returns an appropriate error message if such an exception occurs. The function is also documented with a docstring that describes its functionality, parameters, return value, and includes examples.

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

```python
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 code to follow PEP 8 style with meaningful variable names and comments, with documentation and inline suggestions, use pydoc to generate documentation

Code:

```
81    #Rewrite the code to follow PEP 8 style with meaningful variable names and comments, with documentation and inline suggestions, use pydoc to generate documentation
82    def calculate_grade(marks):
83        """
84        This function calculates the average grade based on a list of marks and returns the corresponding letter grade.
85
86        Parameters:
87        marks (list): A list of integers representing the marks obtained by a student.
88
89        Returns:
90        str: The letter grade corresponding to the average marks.
91
92        Example:
93        >>> calculate_grade([78, 85, 90, 66, 88])
94        'B'
95        >>> calculate_grade([95, 92, 98, 94, 96])
96        'A'
97        >>> calculate_grade([60, 62, 58, 65, 61])
98        'C'
99        >>> calculate_grade([50, 55, 45, 52, 48])
100       'F'
101       """
102       total_marks = sum(marks) # Calculate the total marks by summing the list of marks
103       average_marks = total_marks / len(marks) # Calculate the average marks by dividing total marks by the number of marks
104
105       # Determine the letter grade based on the average marks
106       if average_marks >= 90:
107           return "A" # Return 'A' for average marks greater than or equal to 90
108       elif average_marks >= 75:
109           return "B" # Return 'B' for average marks greater than or equal to 75 but less than 90
110       elif average_marks >= 60:
111           return "C" # Return 'C' for average marks greater than or equal to 60 but less than 75
112       else:
113           return "F" # Return 'F' for average marks less than 60
114   print(calculate_grade([78, 85, 90, 66, 88]) )# Example usage of the calculate_grade function to compute the letter grade based on a list of marks
115
```

Output:

```
The letter grade for marks [78, 85, 90, 66, 88] is: B
```

Explaination: The `calculate_grade` function computes the average of a list of marks and determines the corresponding letter grade based on predefined thresholds. It uses conditional statements to evaluate the average marks and return the appropriate grade. The function is well-documented with a docstring that outlines its purpose, parameters, return value, and example usage.

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
```

Prompt: Rewrite the code to add documentation and inline suggestions, use pydoc to generate documentation

Code:

```python
124  #Rewrite the code to add documentation and inline suggestions, use pydoc to generate documentation
125  def calculate_factorial(n):
126      """
127      This function calculates the factorial of a given non-negative integer n.
128
129      The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.
130
131      Parameters:
132      n (int): A non-negative integer for which the factorial is to be calculated.
133
134      Returns:
135      int: The factorial of the given number n.
136
137      Example:
138      >>> calculate_factorial(5)
139      120
140      >>> calculate_factorial(0)
141      1
142      >>> calculate_factorial(1)
143      1
144      >>> calculate_factorial(10)
145      3628800
146      """
147      result = 1 # Initialize result to 1, as the factorial of 0 and 1 is 1
148      for i in range(1, n + 1): # Loop through numbers from 1 to n (inclusive)
149          result *= i # Multiply result by the current number i to calculate the factorial
150      return result # Return the calculated factorial
151  print(calculate_factorial(5)) # Example usage of the calculate_factorial function to compute the factorial of 5
152
```

Output:

```
The factorial of 5 is: 120
```

Explaination: The `calculate_factorial` function calculates the factorial of a non-negative integer using a loop. It initializes a result variable to 1 and multiplies it by each integer from 1 to n to compute the factorial. The function includes a docstring that explains its purpose, parameters, return value, and provides examples for clarity.

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.

```python
pwd = input("Enter password: ")
if len(pwd) >= 8:
print("Strong")
else:
print("Weak")
```

Prompt:

Enhance the code with 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

Code:

```python
170  import re
171  def check_password_strength(password):
172      """
173      This function checks the strength of a given password based on specific criteria.
174
175      The password is considered strong if it meets the following requirements:
176      - Minimum length of 8 characters
177      - Contains at least one uppercase letter
178      - Contains at least one lowercase letter
179      - Contains at least one digit
180      - Contains at least one special character
181
182      Parameters:
183      password (str): The password string to be evaluated.
184
185      Returns:
186      str: 'Strong' if the password meets all criteria, 'Weak' otherwise.
187
188      Example:
189      >>> check_password_strength("Password123!")
190      'Strong'
191      >>> check_password_strength("weakpass")
192      'Weak'
193      >>> check_password_strength("Short1!")
194      'Weak'
195      >>> check_password_strength("NoSpecialChar1")
196      'Weak'
197      """
```

```python
198      # Check for minimum length
199      if len(password) < 8:
200          return "Weak" # Return 'Weak' if password is shorter than 8 characters
201
202      # Check for uppercase letter
203      if not re.search(r'[A-Z]', password):
204          return "Weak" # Return 'Weak' if password does not contain an uppercase letter
205
206      # Check for lowercase letter
207      if not re.search(r'[a-z]', password):
208          return "Weak" # Return 'Weak' if password does not contain a lowercase letter
209
210      # Check for digit
211      if not re.search(r'[0-9]', password):
212          return "Weak" # Return 'Weak' if password does not contain a digit
213
214      # Check for special character
215      if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
216          return "Weak" # Return 'Weak' if password does not contain a special character
217
218      return "Strong" # Return 'Strong' if all criteria are met
219  print(check_password_strength("Password123!")) # Example usage of the check_password_strength function to evaluate a password's strength
220
```

Output:

```
The password strength of 'Password123!' is: Strong
```

Explaination: The `check_password_strength` function evaluates the strength of a password based on specific criteria such as length, presence of uppercase and lowercase letters, digits, and special characters. It uses regular expressions to check for these criteria and returns 'Strong' or 'Weak' accordingly. The function is documented with a comprehensive docstring that describes its functionality,

parameters, return value, and includes examples for better understanding.