AI ASSISTED CODING

LAB 10 CODE AND REVIEW AND QUALITY: USING AI TO IMPROVE CODE AND READABILITY

SR UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

2303A51209

BATCH 04

TASK 1 — Variable Naming Issues

PROBLEM

Unclear variable and function names reduce readability and

maintainability.

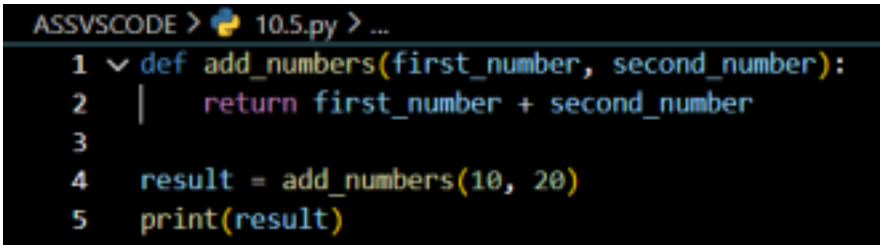
Question

Improve unclear variable names in the given code.

Prompt

Refactor the Python function to use meaningful function and variable names following PEP 8 standards.

Code




```
def add_numbers(first_number, second_number):
    return first_number + second_number
result = add_numbers(10, 20)
print(result)
```

Output

```
PS D:\3-2 SEM\AI ASSISTED> python -u
30
PS D:\3-2 SEM\AI ASSISTED> python -u
30
PS D:\3-2 SEM\AI ASSISTED>
```

Explanation

Function name changed from f → add_numbers for clarity.

Variables a and b renamed to first_number and second_number.

Added docstring to explain function purpose.

Improves readability and maintainability.

TASK 2 — Missing Error Handling

Problem

Division by zero causes runtime errors due to missing exception handling.

Question

Add proper error handling to division function.

Prompt

Modify the function to handle division errors using try-except and display clear messages.

Code

```
def divide_numbers(numerator, denominator):
    try:
        return numerator / denominator
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."
    except TypeError:
        return "Error: Invalid input type."
print(divide_numbers(10, 2))
```

def divide_numbers(numerator, denominator):

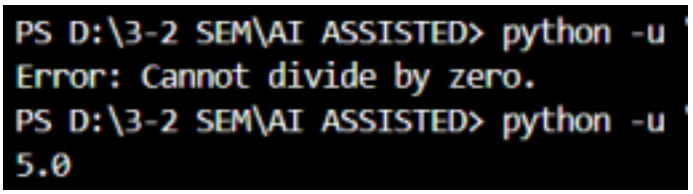
    try:

        return numerator / denominator

    except ZeroDivisionError:

```
        return "Error: Cannot divide by zero."
    except TypeError:
        return "Error: Invalid input type."
print(divide_numbers(10, 2))
```

Output



Before I gave the value zero so it thrown a result of cannot divide by zero

And next I have kept 2 and immediately we got the output and result of

5.0

Explanation

- Added try-except block.
- Handles ZeroDivisionError and TypeError.
- Prevents program crash.
- Provides user-friendly error message.

TASK 3 — Student Marks Processing System

Problem

Poor readability, no functions, no validation, not following PEP8.

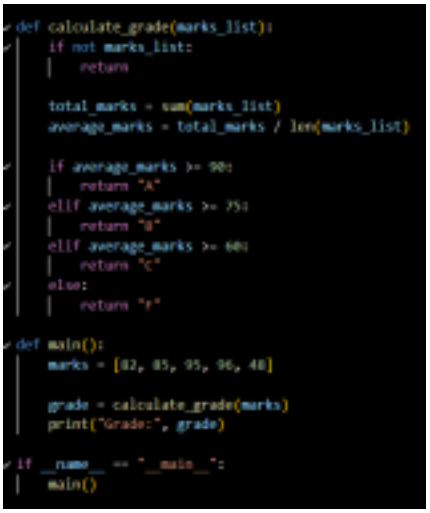
Question

Refactor code with meaningful names, functions, validation, and comments.

Prompt

Rewrite program following PEP 8 with functions, comments, and input validation.

Code



```
def calculate_grade(marks_list):
    if not marks_list:
        return

    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"

def main():
    marks = [82, 85, 95, 96, 48]

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

if __name__ == "__main__":
    main()
```

Output

```
PS D:\3-2 SEM\AI ASSISTED> python
Grade: C
PS D:\3-2 SEM\AI ASSISTED> python
Grade: B
PS D:\3-2 SEM\AI ASSISTED>
```

We get our grades according to our marks

Explanation

- Used functions for modularity.
- Used built-in sum() for efficiency.
- Added validation for empty list.
- Added docstrings and meaningful names.
- Follows PEP 8 formatting.

TASK 4 — Add Docstrings and Comments

Problem

Function lacks documentation and comments.

Question

Add docstrings and inline comments.

Prompt

Enhance function by adding docstring explaining parameters and

logic.

Code

```
def factorial(number):
    result = 1

    # Multiply numbers from 1 to n
    for i in range(1, number + 1):
        result *= i

    return result
num = int(input("Enter a number: "))

# Calling function and printing result
print("Factorial of", num, "is:", factorial(num))
```

def factorial(number):

result = 1
# Multiply numbers from 1 to n

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

result *= i

return result

num = int(input("Enter a number: "))

Calling function and printing result

print("Factorial of", num, "is:", factorial(num))

Output

```
Enter a number: 12
Factorial of 12 is: 479001600
PS D:\3-2 SEM\AI ASSISTED> python -u
Enter a number: 5
Factorial of 5 is: 120
PS D:\3-2 SEM\AI ASSISTED>
```

Explanation

- Added detailed docstring with parameters and return type.
- Added inline comment explaining loop logic.
- Improves documentation and usability.

TASK 5 — Enhanced Password Validation

Problem

Password validation checks only length → weak security.

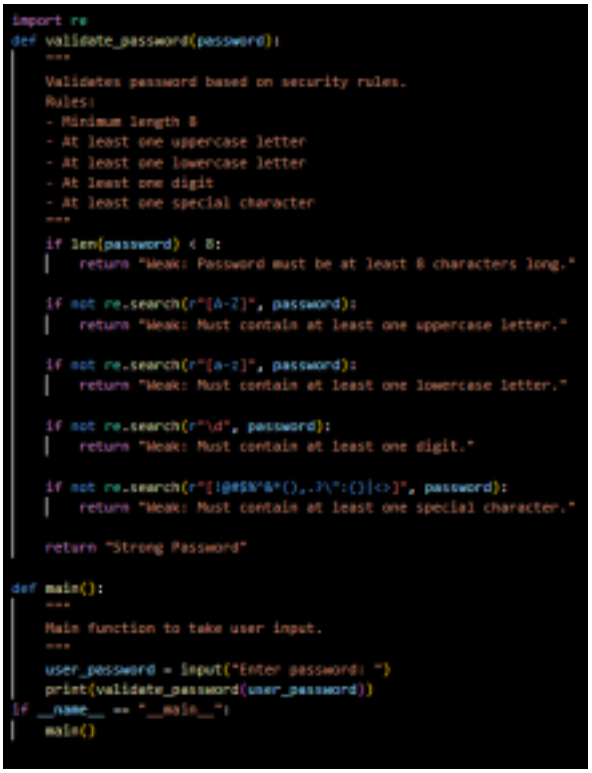
Question

Enhance validation with multiple security rules and analysis.

Prompt

Create a password validation system with multiple rules, comments, docstring, and PEP 8 compliance.

Code



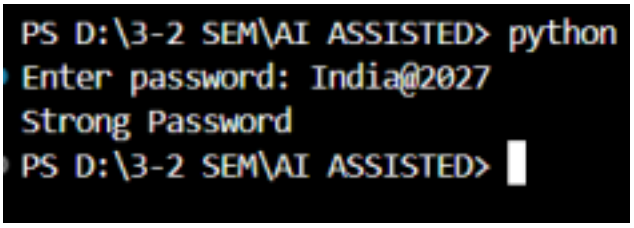
```
import re
def validate_password(password):
    """
    Validates password based on security rules.
    Rules:
    - Minimum length 8
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one digit
    - At least one special character
    """
    if len(password) < 8:
        return "Weak: Password must be at least 8 characters long."

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

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

    if not re.search(r"\d", password):
        return "Weak: Must contain at least one digit."

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

    return "Strong Password"

def main():
    """
    Main function to take user input.
    """
    user_password = input("Enter password: ")
    print(validate_password(user_password))
if __name__ == "__main__":
    main()
```

import re

def validate_password(password):
"""
Validates password based on security rules.
Rules:
- Minimum length 8
- At least one uppercase letter
- At least one lowercase letter

```
- At least one digit
- At least one special character
"""
if len(password) < 8:
return "Weak: Password must be at least 8 characters long."

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

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

if not re.search(r"\d", password):
return "Weak: Must contain at least one digit."

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

return "Strong Password"


def main():
"""
Main function to take user input.
"""
user_password = input("Enter password: ")
print(validate_password(user_password))


if __name__ == "__main__":
main()
```

Output



Explanation

�� Readability Improvements

- **Meaningful function names.**
- **Structured logic.**
- **Clean formatting.**

�� Maintainability

- **Modular function design.**

- **Easy to modify rules.**

Security Improvements

Added checks for:

- **Uppercase → prevents simple passwords**
- **Lowercase → improves complexity**
- **Digit → increases entropy**
- **Special character → prevents brute force**
- **Length → baseline security**

Justification

Each rule increases password entropy and reduces vulnerability to brute-force and dictionary attacks.

Refactoring improves scalability and professional coding standards.