

2303A51449

BATCH :03

Task 1 – Variable Naming Issues

Improved Code:

```
def add_numbers(first_number: int, second_number: int) -> int:  
    """Returns the sum of two integers."""  
    return first_number + second_number  
  
print(add_numbers(10, 20))
```

Task 2 – Missing Error Handling

Improved Code:

```
def divide_numbers(dividend: float, divisor: float) -> float:  
    """Divides two numbers and handles division errors."""  
    try:  
        return dividend / divisor  
    except ZeroDivisionError:  
        print("Error: Division by zero is not allowed.")  
        return None  
    except TypeError:  
        print("Error: Invalid input type. Please provide numeric values.")  
        return None  
  
result = divide_numbers(10, 0)  
if result is not None:  
    print(result)
```

Task 3 – Student Marks Processing System

Refactored Code:

```
from typing import List  
  
def calculate_total(marks: List[int]) -> int:  
    return sum(marks)
```

```

def calculate_average(marks: List[int]) -> float:
    if not marks:
        raise ValueError("Marks list cannot be empty.")
    return sum(marks) / len(marks)

def determine_grade(average_score: float) -> str:
    if average_score >= 90:
        return "A"
    elif average_score >= 75:
        return "B"
    elif average_score >= 60:
        return "C"
    else:
        return "F"

def main():
    marks = [78, 85, 90, 66, 88]
    total_marks = calculate_total(marks)
    average_marks = calculate_average(marks)
    grade = determine_grade(average_marks)

    print(f"Total Marks: {total_marks}")
    print(f"Average Marks: {average_marks:.2f}")
    print(f"Grade: {grade}")

if __name__ == "__main__":
    main()

```

Task 4 – Factorial Function with Documentation

```

def calculate_factorial(number: int) -> int:
    """
    Calculates the factorial of a non-negative integer.
    Raises TypeError if input is not integer.
    Raises ValueError if number is negative.
    """

    if not isinstance(number, int):
        raise TypeError("Input must be an integer.")
    if number < 0:
        raise ValueError("Factorial is not defined for negative numbers.")

    result = 1
    for value in range(1, number + 1):

```

```
    result *= value
```

```
return result
```

Task 5 – Enhanced Password Validation System

```
import re
```

```
def is_strong_password(password: str) -> bool:
```

```
    """
```

```
        Validates a password based on multiple security rules.
```

- Minimum length of 8 characters
- At least one uppercase letter
- At least one lowercase letter
- At least one digit
- At least one special character

```
    """
```

```
    if len(password) < 8:
```

```
        return False
```

```
    if not re.search(r"[A-Z]", password):
```

```
        return False
```

```
    if not re.search(r"[a-z]", password):
```

```
        return False
```

```
    if not re.search(r"[0-9]", password):
```

```
        return False
```

```
    if not re.search(r"[@#$%^&*(),.?':{}|<>]", password):
```

```
        return False
```

```
    return True
```

```
def main():
```

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

```
    if is_strong_password(user_password):
```

```
        print("Strong Password")
```

```
    else:
```

```
        print("Weak Password")
```

```
if __name__ == "__main__":
```

```
    main()
```

Justification

The enhanced versions improve readability through meaningful variable names, structured

functions, and proper formatting according to PEP 8 standards.

Maintainability is improved through modular functions and reusable logic.

Security is strengthened by adding multiple password validation rules and proper exception handling.

AI-assisted refactoring transforms basic scripts into structured, professional-quality Python programs.