

Refactoring & Error Handling Summary Report

1. What Was Refactored

Path 1 (My Code): Improved validation logic, structured error handling, and separated responsibilities between validation and API processing.

Path 2 (Starter Code): Reorganized monolithic logic into modular functions and replaced generic exceptions with custom structured errors.

2. Support Functions Created

- validate_product_data()
- generate_description()
- test_invalid_product_data()
- test_api_error()
- Custom exception classes: ValidationError, APIError

3. Code Modularization

Separated concerns into validation layer, API layer, and testing layer. Each function now has a single responsibility, improving readability, maintainability, and scalability.

4. Error Handling Examples (Before / After)

Before: Generic error such as "Exception occurred" or unhandled crash.

After (Validation): "ERROR in validate_product_data(): Invalid fields: name is required, price must be greater than 0"

After (API): "ERROR in generate_description(): APIError | Product: Test Product | Reason: API timeout | Suggestion: Try again later."

5. Challenges Faced

- Ensuring all invalid fields are collected before raising errors.
- Designing consistent and readable error messages.
- Making tests meaningful and reliable.

6. What I Learned

- Importance of custom exception classes.
- Value of modular architecture and separation of concerns.
- How structured error messages improve debugging and user experience.
- Writing clearer and more maintainable test cases.