Brief Report: Refactoring Product Description Generator

1. What was refactored:
The original Product Description Generator was a monolithic script that combined multiple responsibilities — loading JSON, validating data, generating prompts, calling the OpenAI API, and saving results — in a single function. The code also silently failed on errors, lacked proper error handling, and used deprecated Pydantic validators. The code was refactored to separate concerns, improve readability, and provide clear, informative error messages.

2. Helper functions created:
To improve reusability and maintainability, the following helper functions were introduced:

- load_json_file(file_path: str): Loads JSON from file with FileNotFoundError and JSONDecodeError handling.
- validate_product_data(product_dict: dict): Validates product data using Pydantic, handling ValidationError.
- create_product_prompt(product: Product): Generates OpenAI prompt for a given product.
- parse_api_response(response): Parses the response from the OpenAI API.
- format_output(product: Product, description: str): Formats the final product output.
- generate_description(product: Product, api_client): Handles API calls and network errors.
- call_api_with_network_error(url: str): Simulates network errors for demonstration.

3. How the code was modularized:
Each task in the original monolithic function was moved into a focused module (helper function) with a single responsibility. The main orchestration now lives in main(), which sequentially:

1. Loads and validates JSON products.
2. Processes each valid product via API.
3. Formats and saves the results.
   This modular design allows each function to be tested independently and makes maintenance and debugging easier.

4. Examples of error handling:

- **Before:** Errors were often silent; invalid JSON or missing files caused crashes without explanation.
- **After:**
    - FileNotFoundError:

- ERROR in load_json_file(): FileNotFoundError
- Location: File 'products.json' not found
- Current directory: C:\Users\kupit\week 2\d31
- Suggestion: Check that the file path is correct
- JSONDecodeError:
- ERROR in load_json_file(): JSONDecodeError
- Location: File 'invalid_products.json', line 10, column 1
- Message: Expecting ',' delimiter
- Suggestion: Check JSON syntax at line 10
- ValidationError:
- ERROR in validate_product_data(): ValidationError
- Product ID: P004
- Invalid fields:
- - ('price',): Price must be positive
- Suggestion: Fix the invalid fields above
- OpenAI APIError:
- ERROR in generate_description(): APIError
- Product: Widget X (ID: P001)
- Error type: InvalidRequestError
- Status code: 401
- Message: Incorrect API key provided
- Suggestion: Check API key, rate limits, or try again later
- NetworkError:
- ERROR in generate_description(): NetworkError
- Product: Widget Y (ID: P002)
- Message: Failed to establish a new connection
- Suggestion: Check your internet connection or server status

5. Challenges faced:

- Migrating from Pydantic v1 validators to v2 @field_validator.

- Ensuring all types of errors were explicitly caught and clearly reported.
- Refactoring a monolithic function into modular components while maintaining correct workflow and OpenAI API calls.

6. What was learned:

- Modular code and helper functions greatly improve readability and maintainability.
- Clear and contextual error messages make debugging much faster and safer.
- Handling external API errors and network issues is critical for production-ready code.
- Testing all error scenarios systematically ensures robust and reliable software.