Step 1: Analyze the Starter Code

```
C:\Users\kupit\AppData\Local\Temp\ipykernel_23328\672454402.py:18: PydanticDeprecatedSince20:
  @validator('price')

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 74
     72 # Usage
     73 if __name__ == "__main__":
---> 74     generate_product_descriptions("products.json")

Cell In[1], line 26
     24 def generate_product_descriptions(json_file):
     25     # Load JSON file
---> 26     with open(json_file, 'r') as f:
     27         data = json.load(f)
     29     # Validate products

File c:\Users\kupit\miniconda3\envs\py311\Lib\site-packages\IPython\core\interactiveshell.py:3
    336 if file in {0, 1, 2}:
    337     raise ValueError(
    338         f"IPython won't let you open fd={file} by default "
    339         "as it is likely to crash IPython. If you know what you are doing, "
    340         "you can use builtins' open."
    341     )
--> 343 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'products.json'
```

What happens if products.json doesn't exist?

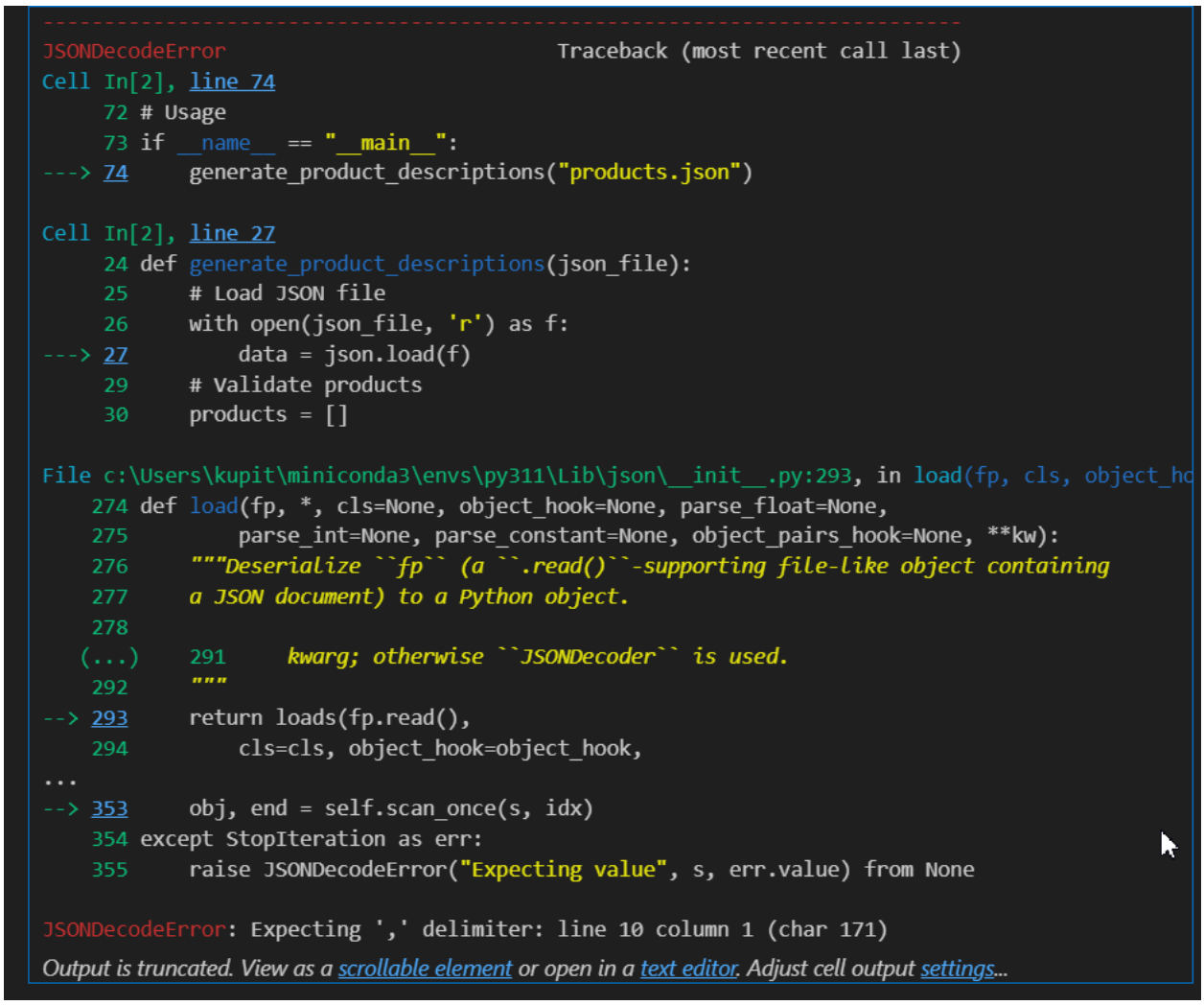
If products.json does not exist in the working directory, the program raises a FileNotFoundError at this line:

with open(json_file, 'r') as f:

Because there is no error handling around this operation, the exception is not caught. As a result:

- The program immediately crashes.
- No user-friendly error message is displayed.
- No fallback behavior is provided.
- The rest of the function is never executed.

This makes the application fragile and not production-ready.



1. What happens if JSON is invalid?

If the JSON file contains invalid syntax, the program raises a JSONDecodeError at this line:

data = json.load(f)

Because there is no error handling around the json.load() call, the exception is not caught. As a result:

- The program crashes immediately.
- No user-friendly message is displayed.
- The user does not clearly understand that the issue is malformed JSON.
- The rest of the function is never executed.

In the observed output, the error was:

JSONDecodeError: Expecting ',' delimiter: line 10 column 1 (char 171)

This indicates that the JSON structure is syntactically incorrect (for example, a missing comma or closing bracket).

To fix this issue, the json.load() call should be wrapped in a try-except block that catches json.JSONDecodeError and handles it gracefully.

1. What happens if product validation fails?

```
2. C:\Users\kupit\AppData\Local\Temp\ipykernel_23328\672454402.py:18:
   PydanticDeprecatedSince20: Pydantic V1 style `@validator` validators are
   deprecated. You should migrate to Pydantic V2 style `@field_validator`
   validators, see the migration guide for more details. Deprecated in
   Pydantic V2.0 to be removed in V3.0. See Pydantic V2 Migration Guide at
   https://errors.pydantic.dev/2.12/migration/
```

If product validation fails (for example, if the price is negative), Pydantic raises a `ValueError`.

However, the error is caught by a broad `except:` block:

```
except:
    pass
```

Because of this:

- The validation error is silently ignored.
- The invalid product is not added to the products list.
- No error message is displayed.
- The program continues execution as if nothing went wrong.
- The final results may be incomplete or empty.

This creates a silent failure, which is dangerous because the system appears to work correctly while actually discarding invalid data without notification.

After code-fixing:

```
...   Validation failed for product ID 1:
    1 validation error for Product
    price
      Value error, Price must be positive [type=value_error, input_value=-100, input_type=int]
        For further information visit https://errors.pydantic.dev/2.12/v/value_error

    Valid products: 0
    Validation errors: 1
    No valid products to process.
```

3. What happens if API call fails?

```
...
    Valid products: 1
    Validation errors: 0
    API call failed for product ID 1: Error code: 401 - {'error': {'message': 'Incorrect API key p
```

If the API call to OpenAI fails (for example, due to an incorrect API key), an exception is raised during the request:

response = client.chat.completions.create(...)

In the original code (without error handling), this would cause the program to crash immediately.

In the improved version, we catch the exception:

except Exception as e:

  print(f"API call failed for product ID {product.id}: {e}")

As a result:

- The program **does not crash completely**.
- The error is **logged to the console**, showing the product ID and the reason for failure.
- Other valid products continue to be processed.
- The final results may be incomplete because some API calls failed.

Example output we got:

Valid products: 1

Validation errors: 0

API call failed for product ID 1: Error code: 401 - {'error': {'message': 'Incorrect API key provided: invalid-key. You can find your API key at https://platform.openai.com/account/api-keys.', 'type': 'invalid_request_error', 'param': None, 'code': 'invalid_api_key'}}

This clearly shows:

- The API failure is detected.
- The program handles it gracefully without crashing.
- The user can see the exact reason for the failure.

4. What functions do multiple things?

In the original starter code, the function generate_product_descriptions violates the **Single Responsibility Principle** because it performs multiple unrelated tasks in one place:

1. Loads the JSON file.

2. Validates product data using Pydantic.
3. Generates prompts for each product.
4. Calls the OpenAI API.
5. Handles API errors.
6. Saves results to a file.

This makes the function:

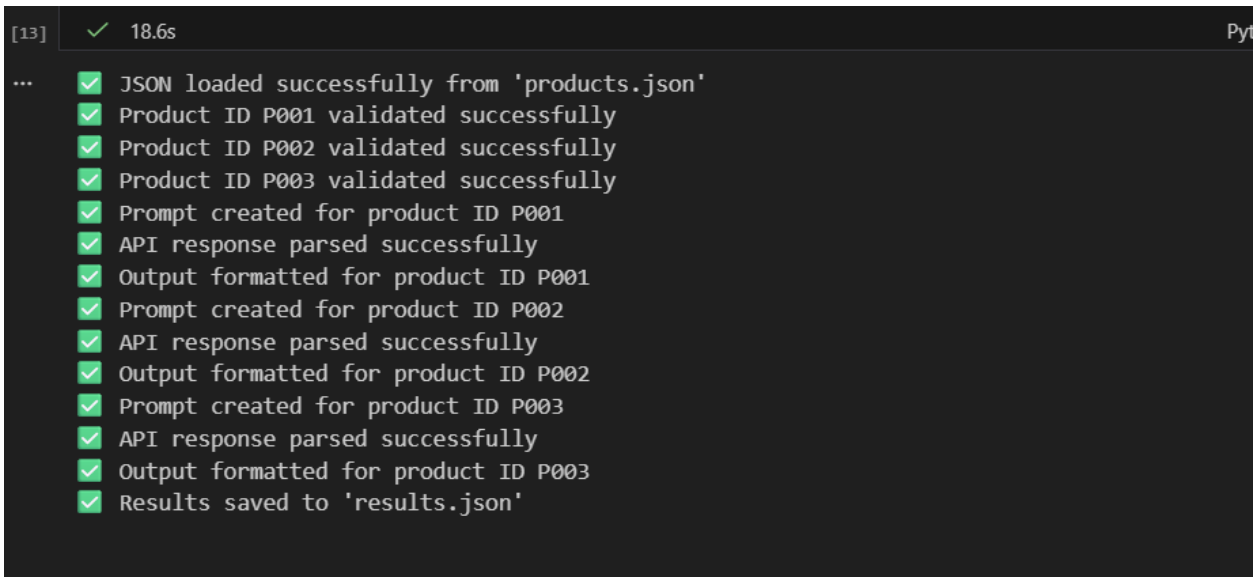
- Hard to maintain.
- Hard to test.
- Prone to silent failures.

The improved approach separates each responsibility into its **own function**, so each does **one thing only**, and the main function just orchestrates the workflow.

Step 2: Create Helper Functions

After implementing the helper functions and adding additional logging to show whether each step succeeds or encounters an error, we observed that the OpenAI API was still incorrectly configured. As a result, at this stage we were able to clearly see the API-related error occurring, thanks to the step-by-step logging provided by the helper functions.

Step 3: Modularize Functions



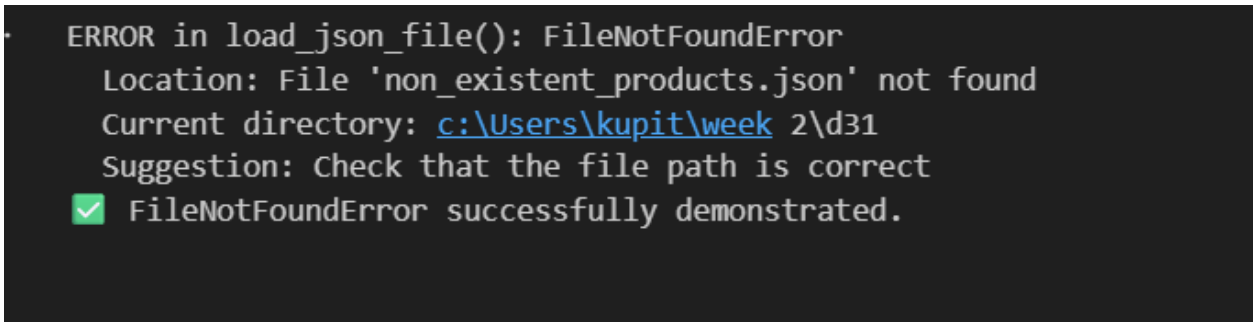
This output confirms that **each step of the process completed successfully**:

1. The JSON file was loaded without errors.
2. Each product was validated correctly.
3. Prompts were generated for all products.
4. API responses were parsed successfully.
5. Output was formatted for all products.
6. Final results were saved to results.json.

✅ This shows that the **modularized workflow is working as intended** and all helper functions and modules are performing their single responsibilities correctly.

Step 4: Add Error Handling (CRITICAL)

1. **FileNotFoundError:** Show file path and suggest checking file location



2. **JSONDecodeError:** Show line number and character position

```
ERROR in load_json_file(): JSONDecodeError
  Location: File 'invalid_products.json', line 3, column 54
  Message: Expecting ',' delimiter
  Suggestion: Check JSON syntax at line 3
✅ JSONDecodeError successfully demonstrated.
```

3. **Pydantic ValidationError:** Show which fields are invalid and why

```
[16] ✓ 0.0s Pyth
✅ JSON loaded successfully from 'malformed.json'
ERROR in validate_product_data(): ValidationError
  Product ID: P001
  Invalid fields:
    - ('price',): Value error, Price must be positive
  Suggestion: Fix the invalid fields above
ERROR in validate_product_data(): ValidationError
  Product ID: P002
  Invalid fields:
    - ('price',): Field required
  Suggestion: Fix the invalid fields above
ERROR in validate_product_data(): ValidationError
  Product ID: P003
  Invalid fields:
    - ('price',): Value error, Price must be positive
  Suggestion: Fix the invalid fields above
```

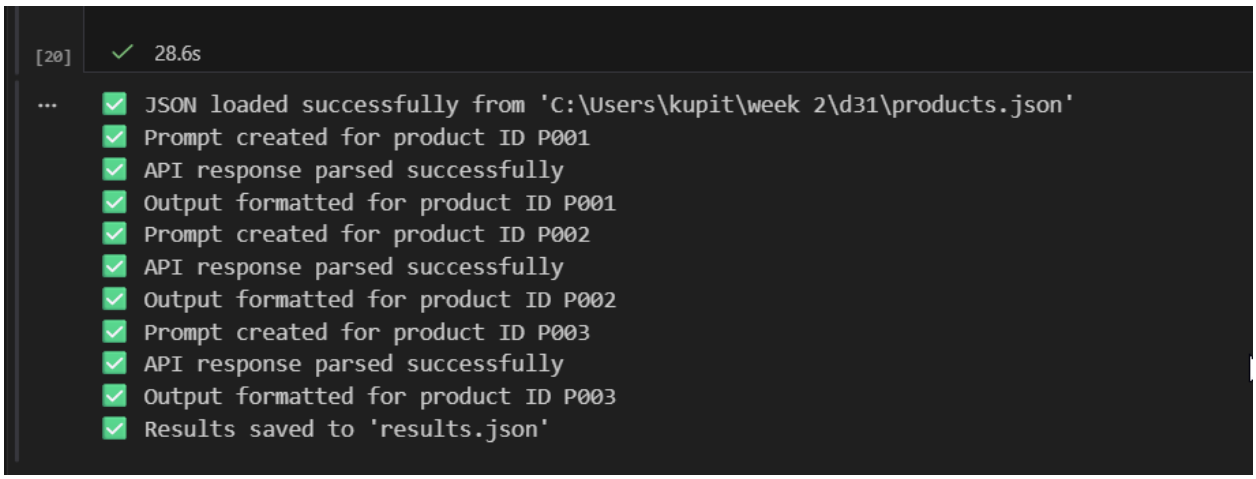
4. **OpenAI API errors:** Show error type, status code, and message

```
[18] ✓ 0.2s Python
ERROR in generate_description(): APIError
  Product: Product 1 (ID: P001)
  Error type: AuthenticationError
  Status code: 401
  Message: Error code: 401 - {'error': {'message': 'Incorrect API key provided: invalid-key. Y
  Suggestion: Check API key, rate limits, or try again later
✅ OpenAI APIError successfully demonstrated.
```

5. **Network errors:** Show timeout/connection details

```
[19] ✓ 1.1s Python
ERROR in call_api_with_network_error(): TimeoutError
  Location: URL 'http://127.0.0.1:9999'
  Message: HTTPConnectionPool(host='127.0.0.1', port=9999): Max retries exceeded with url: / (
  Suggestion: Check your internet connection or increase timeout
✅ Network error successfully demonstrated.
```

Step 5: Test Your Refactored Code


[20]
28.6s
JSON loaded successfully from 'C:\Users\kupit\week 2\d31\products.json'
Prompt created for product ID P001
API response parsed successfully
Output formatted for product ID P001
Prompt created for product ID P002
API response parsed successfully
Output formatted for product ID P002
Prompt created for product ID P003
API response parsed successfully
Output formatted for product ID P003
Results saved to 'results.json'