IDEA

A professional approach to reduce technical debt and handle carry-overs without confusion. A technical debt management formalizing our code into a unified and standardized approach, ensuring every member/developer can trace the logic without needing a walkthrough.

A Custom Documentation Schema is introduced to you to formalize coding. Please follow these naming conventions and documentation tags for every module you contribute.

NAMING CONVENTIONS

To maintain a "single-author" feel, we will use the following identifier styles. Note: Since Python does not support dashes in variable names, we use underscores to separate words.

Component	Style	Example
Global Variables	Leading underscore + snake_case	`_api_endpoint`
Normal Variables	Standard snake_case	`processed_data_list`
Functions	Wrapped in underscores	`_calculate_score_()`
Constants	Leading underscore + ALL_CAPS	`_MAX_TIMEOUT`
Classes	PascalCase (No underscores)	`DataNormalizer`

PROJECT-SPECIFIC ANNOTATION STANDARD (PSAS) TAGS

Core Tags
`@func_ [name]`: Defined function at the start of the docstring.
`@params [names]`: List input arguments and their expected types.
`@var_ [name]`: Used for significant local variables (skip for trivial counters).
`@iter_ [target]`: Explain the purpose of a loop or data traversal.
`@return_ [type]`: Describe the output of the function.

Advanced Logic Tags

`@logic_`: Explain the "why" behind a specific mathematical formula or complex conditional.
`@err_`: Document how potential exceptions (try-except blocks) are handled.
`@dep_`: List any external libraries required for that specific file or block.

For Inline Tags, use double hashtags `##` to avoid Python documentation conflicts

PROJECT HEADERS

```python
"""
Saint Louis University : Team 404FoundUs
@file_ [Filename.py]
@project_ [Thesis Title/Project Name]
@desc_ A brief overview of what this specific script contributes to the system.
@deps_ List of external libraries required (e.g., requests, json, csv).
"""
```

Establish Proper Headers To Give Acknowledgement to the team and authors of the whole code

GOLD STANDARD IMPLEMENTATION

```python
"""
Saint Louis University : Team404FoundUs
@file_ thesis_evaluator.py
@project_ CS Thesis: LLM Normalization Pipeline
@desc_ A comprehensive class for loading test cases, normalizing text via AI,
       and grading the output using an external Judge LLM (OpenRouter).
@deps_ requests, csv, json, os
"""

import csv import json import os import requests

## @const_ Global configuration constants
_OPENROUTER_URL = "https://openrouter.ai/api/v1/chat/completions"
_JUDGE_MODEL = "google/gemini-2.0-flash-001"
```

```python
class ThesisEvaluator:
    """
    @class_ ThesisEvaluator
    @desc_ Manages the end-to-end lifecycle of the testing pipeline.
    @attr_ _api_key : (str) Credential for the OpenRouter API.
    @attr_ _source_file : (str) Path to the CSV input file.
    """

    def __init__(self, _key_input, _file_path):
        self._api_key = _key_input
        self._source_file = _file_path
```

Python

```python
def _process_test_results_(_raw_input, _limit):
    """
    @func_ _process_test_results_ (@params _raw_input, _limit)
    @var_ filtered_data : temporary list to hold threshold results.
    @iter_ _raw_input : filtering the input list based on validity.
    @return_ list : returns a cleaned list of results.
    """

    filtered_data = []

    ## @logic_ Using a list comprehension for efficiency in large datasets
    for _item in _raw_input[:_limit]:
        if _item is not None:
            filtered_data.append(_item)

    return filtered_data
```