Task 1: AI-Powered Code Completion — Documentation

# Goal

To write a Python function that sorts a list of dictionaries by a specified key, using both a manual implementation and an AI-suggested version, then compare their efficiency and robustness.

# Step 2: Comparing the AI-Suggested Code with the Manual Implementation

## Manual Implementation

def sort\_dicts\_by\_key(dict\_list, sort\_key, reverse=False):  
 """  
 Sorts a list of dictionaries by a specified key.  
  
 Args:  
 dict\_list (list): List of dictionaries to sort.  
 sort\_key (str): The key to sort the dictionaries by.  
 reverse (bool): Sort in descending order if True.  
  
 Returns:  
 list: A new list of dictionaries sorted by the specified key.  
 """  
 return sorted(dict\_list, key=lambda x: x.get(sort\_key, None), reverse=reverse)

Key Features:

* - Uses `.get(sort\_key, None)` to safely access the key.
* - Handles cases where some dictionaries might not contain the key.
* - Supports optional descending sort with the `reverse` parameter.

## AI-Suggested Implementation

def sort\_dicts\_by\_key(dict\_list, sort\_key, reverse=False):  
 return sorted(dict\_list, key=lambda d: d[sort\_key], reverse=reverse)

Key Features:

* - Uses direct key access with `d[sort\_key]`, which is shorter.
* - Assumes all dictionaries contain the key.
* - Also supports reverse sorting.

# Step 3: Efficiency and Robustness Comparison

Comparison:

- Manual Implementation:

* - Safe: uses `.get()` to avoid errors
* - Works even if some keys are missing
* - Slightly slower due to `.get()`
* - Slightly more verbose, but clear
* - Suitable for real-world messy data

- AI-Suggested Implementation:

* - Unsafe: may raise `KeyError`
* - Breaks if any dictionary lacks the key
* - Slightly faster due to direct indexing
* - More concise, but assumes consistent data
* - Suitable for clean, well-structured data

## Which Version is More Efficient?

Raw performance: The AI-suggested version is slightly faster, because direct indexing (d[sort\_key]) is more efficient than using `.get()`.  
  
Practical efficiency: The manual implementation is better in real-world scenarios because it is fault-tolerant, making it more reliable for diverse data.  
  
Efficiency is not only about speed, but also about how gracefully the function behaves with real data.

# Conclusion

- Use the manual implementation if:

* - You are dealing with real-world or inconsistent data.
* - You want to avoid unexpected `KeyError`s.

- Use the AI-suggested version if:

* - All dictionaries guarantee the presence of the sort key.
* - You need to optimize performance in a controlled environment.