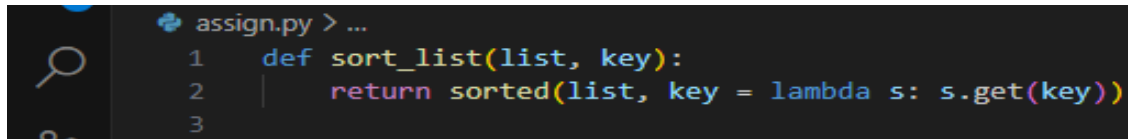


Part 2: Practical implementation

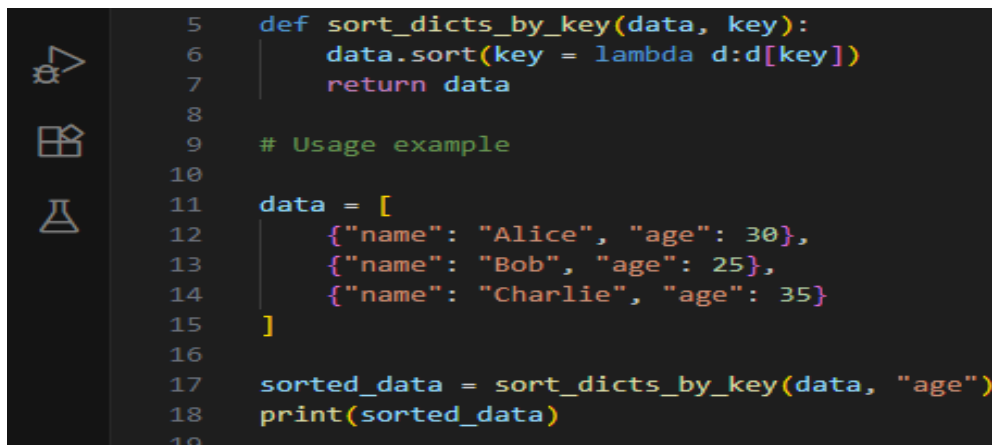
Task 1: Ai-powered Code Completion

1. Python function to sort a list of dictionaries by a specific key



```
assign.py > ...
1 def sort_list(list, key):
2     return sorted(list, key = lambda s: s.get(key))
3
```

2. Ai-suggested Code



```

5 def sort_dicts_by_key(data, key):
6     data.sort(key = lambda d:d[key])
7     return data
8
9 # Usage example
10
11 data = [
12     {"name": "Alice", "age": 30},
13     {"name": "Bob", "age": 25},
14     {"name": "Charlie", "age": 35}
15 ]
16
17 sorted_data = sort_dicts_by_key(data, "age")
18 print(sorted_data)
19
```

3. Comparison between the manual written code and the Ai suggested code

The above pictures indicate how humans and Ai construct code differently. Both functions, the `sort_list()` and the `sort_dicts_by_key()` aim to sort a list of dictionaries by a specified key. In the first function (code by a human), the list is sorted by using the `‘.get()’` method which is considered more safe, stable and more flexible as it sorts the list of dictionaries even when some dictionaries don't have the specified key without crashing or running into an error. It also ensures that the new sorted list is returned without changing the original list.

On the other hand, the Ai suggested function sorts the list by using a direct key access `‘d[key]’`. The suggested code is more efficient. However it is risky because if there are dictionaries with missing keys, this will cause an error. It works better with well recorded and structured data. It also changes the original data which is not very ideal if you would like to keep the original data. The Ai suggested code provides examples which makes it easier for one to understand the code better.

Both the manual and the Ai versions have theirs pros and cons but in our case the manual version is better because of the clean logic, error tolerance and a functional behaviour that does not change the original list.