1. mapping( **base:** List[string], **target:** List[string] ) -> List[str]:
2. --------------------------------
3. // assuming len(base) == n, len(target) == m
4. // there are ((n choose 2) \* (m choose 2) \* 2) pairs
5. possible\_pairs = get\_all\_possible\_pairs(base, target)
7. // here we going to store the entities that already mapped.
8. // the value in index ***i*** in both lists will be the map between them.
9. // it is clear that both must be in the same length.
10. base\_already\_map, target\_already\_map = [], []
12. while len(base\_already\_map) < min(len(base), len(target)):
13. // updating the possible pairs according to the entities that already mapped
14. // the idea is to not break the entities that already mapped.
15. update\_possible\_pairs(possible\_pairs, base\_already\_map, target\_already\_map)
17. // we want the pair with the best score.
18. // the meaning of pair is for example: earth🡪electrons **AND** sun🡪nucleus.
19. res = get\_best\_pair\_mapping(possible\_pairs)
21. if res["score"] > 0:
22. // updating the already mapped lists.
23. // res["base"][0] 🡪 res["target"][0], res["base"][1] 🡪 res["target"][1]
24. update\_list(base\_already\_map, res["base"])
25. update\_list(target\_already\_map, res["target"])
26. else:
27. // no map found at all.
28. break
29. --------------------------------
30. return [f"{b} 🡪 {t}" for b, t in zip(base\_already\_map, target\_already\_map)]
31. get\_best\_pair( **pairs:** List[List[Tuple[string]]] ) -> List[Tuple[string]]:
32. mapping = []
33. --------------------------------
34. for pair in pairs:
35. // pair is something like: [(earth, sun), (electrons, nucleus)]
36. base\_edge, target\_edge = pair
37. mapping.append(pair, get\_score(base\_edge, target\_edge))
38. --------------------------------
39. return sorted(mapping, key=lambda x: [1], reverse=True)[0]