CS562
Jingyuan Xu
Part 3: The progress report

In this parse, I use more test cases to test functions in examine.py than before. For example, we have the data structures like [],[None],['coding', 'snowboarding'] and {{'key1': 'val1'}, {'key2': 'val2'}},{'key1': 'val1'} and ‘val2’. In order to show the probably of different combinations, and avoid only print out single functions by manual, I will put some of them into a container called “exams”, and the container like [‘['coding', 'snowboarding']’,’{'key2': 'val2'}’,’val1’,’[{'key1': 'val1'}, {'key2': 'val2'}]’] , then if I need to add new test cases into this container.

I will use .append function in TSTL which is a new part in this parse. Later I can use test cases in the container to test each functions. Then I use <exams,1> this kind of structure for random testing. The test case looks like this:

print examine.Structure(<exams>);examine.Structure(<exams,1>)

In here, I need first initialize structure <exams>, then I can use <exams,1> to random pick the elements in “exams” and pass for testing functions.

Overall, I improve my test system and make a test checklist. There are these kind of test structures I’m going to test: list (e.g :[]), dictionary (e.g. {}) and tuple (e.g. ()), and the test cases will be randoms conjoin them together. Here is a list that I will test combinations for each type:

1. non structure
2. simple list
3. none list
4. mixed type list
5. simple dict
6. dictionaries with lists
7. blank list in dict
8. blank list
9. blank dict
10. defer list type if empty
11. empty list merged with non list
12. nested list
13. merging none list with int list
14. nested dict
15. add empty list to list of dicts
16. none in list
17. none as val of key in list of dicts
18. tuple
19. deep list nest

20. list of tuples
21. empty tuple merging

In this phase, I finished until No.9, nearly half of the list. So I finished testing basic data structures. Next step, I'm going to test more complex data structure such like "nest list", "merging list" which may need to call __add__method, and examine tuple structure. I have great confidence to finish this checklist at the end of term.

It's very unlucky that I didn't find the bug through my test checklist, I hope later when I test complex structure, there will appear some bugs.

The nice feature of examine.py is it can analysis the datature and show to users in a clear way. Because the main class in this library is Structure(), and it contains some important methods like: __add__(), copy(), type_string(), __str__(), these methods can all be tested by calling Structure class in the test cases. This will improve the code readability, and make a larger coverage. In the simple data structure, based on the test result, this examine library is hardly to find bugs. However, some mix structures may cause problems is the code is not well written. I hope in the next phase, the more complex test cases be used, the more disadvantage will be found in this examine library.