Lab 4: Testing Load Data Module

Objectives:

Practice implementing unit testing.
Test the *load data* module of your software project.

Tasks:

1. Complete Task 1: Develop Unit Tests for Load Data Module.
2. Complete Task 2: Complete the *test_load_data* Module.

Task 1: Develop Unit Tests for Load Data Module

As a team, you will write four different test cases for your *load_data* module. Remember to use automated testing, as you learned during the lectures. You need to import the `check` module explained during the lectures. Use the functions in the `check` module in your implementation.
You are not allowed to modify the `check` module.
Docstring and type annotations are **not required** for this lab.
For your tests, use the `characters-test.csv` file provided.

STEP 1

1. Read the descriptions of the tests below.
2. Decide who is writing which test.

Test 1: Functions Return a List

In a file named *test1.py*, write a test function named *test_return_list* to test that all six functions in the load data module return a list.

For each function except *load_data* you need to provide at least 4 test cases:
5 functions * 4 test/function = 20 test cases
For function *load_data*, you need to provide 6 test cases.
That's 26 test cases in total.

Hint: For this test, you need to check that the return type of the function is a list. You must use the Python function *isinstance.* Look online at how to use this function (e.g., https://www.w3schools.com/python/ref_func_isinstance.asp).

Test 2: Functions Return a List with the Correct Length

In a file named *test2.py*, write a test function named *test_return_list_correct_length* to test that all six functions in the load data module return a list of the correct length.

Example:

`calculate_health([])` should return a list of length zero.

`load_data("characters-test.csv", ("Occupation", "F"))` should return a list of length zero.

`character_strength_list("characters-test.csv", (3,10))` should return a list of length four.

For each function except *load_data* you need to provide at least 4 test cases:
5 functions * 4 test/function = 20 test cases
For function *load_data*, you need to provide 6 test cases.
That's 26 test cases in total.

Test 3: Correct Dictionary Inside the List

In a file named *test3.py*, write a test function named *test_return_correct_dict_inside_list* to test that all six functions in the load data module, the dictionaries inside the list have the correct data.
A general character's data is stored in the following format (although some keys may or may not be present depending on which function is being tested)

```
{'Occupation': 'WA', 'Agility': 11, 'Stamina': 9, 'Personality': 10, 'Intelligence': 8, 'Luck': 0.61, 'Armor': 11, 'Weapon': 'Spear'}.
```

We recommend testing the first and the last element in the list.

For each function except *load_data* you need to provide at least 4 test cases:
5 functions * 4 test/function = 20 test cases
For function *load_data*, you need to provide 6 test cases.
That's 26 test cases in total.

Test 4: Calculate Health

Test 1, 2, and 3 already covered some tests for *calculate_health*, but some extra checks are needed:

(1) *calculate_health* should not change the size of the list.
(2) The number of keys in each dictionary should increase only by one
(3) The value for *Health* should be properly calculated.

In a file named *test4.py*, write a test function named *test_calculate_health* to cover the above-mentioned test cases.

You must conduct these Four tests for all five possible cases:

- ‘Occupation’ is not a key of the dictionary.
- ‘Luck’ is not a key of the dictionary.
- ‘Strength’ is not a key of the dictionary.
- ‘Weapon’ is not a key of the dictionary.
- The case where all keys are present.

That’s 20 test cases.
For (1), you should also consider the case where the empty list is provided as an input parameter. Therefore, you need to provide at least 21 test cases.

STEP 2

1. Implement the test function you have been assigned.
2. Shell test the function you just implemented. Correct any mistakes you find.
3. Submit your test file on Brightspace (individual submission of your assigned test). **Do not submit the *check* module and *load_data.py*.**
4. After submitting, ensure that the submission is on the system. You can submit as many times as you need before the deadline. Remember to share your code with your teammates.

Task 2: Complete the `test_load_data` Module

STEP 1

1. Download the template ***test_load_data.py***
2. Complete the information for __authors__ and __team__ at the top of the file.
3. Copy the four test functions developed in Task 1 into this file. Follow the instructions on the template. If a team member did not participate, complete their assigned task, but not give them credit. Remember that the whole team is responsible for writing the missing functions.
4. As a team, verify that all the test functions are correct. Ensure that you have covered all boundary cases. TAs will check for this! Give feedback to your peers on their mistakes.

STEP 2

1. Submit your test module (*test_load_data.py*) on Brightspace (One submission per team). **Do not submit the *check* module and *load_data.py*.**
2. Before you submit, ensure:
    a. You have completed the information for __authors__ and __team__ at the top of the file. If a name is missing, that person will receive zero. We will assume that the person did not participate in the task.
    b. Test functions have the proper names.
    c. At the end of the file, you do not have a main script with function calls.
    d. The file name is correct.

It is the responsibility of the whole team to ensure that the final version of the module is uploaded on time. All team members, not just the team leader, can upload the team submission on behalf of the team.