Lab 3: Make it better

Objectives

- Writing flexible and configurable code
- Modular programming
- Exploring local and global namespace via functions
- Understanding the differences between simple and complex data types
- Observing the details of pass by value and pass by reference
- Implementing recursive functions
- Handling errors by unittest
- Analyzing a problem

Preliminary Setup

You made a folder, i.e., your **workspace,** on your computer where you work on both projects and labs for this course. Each of the projects and labs will be **folders inside the workspace folder**.

PyCharm

Launch PyCharm. Create a New Project.

Implement/Try/Discuss

1. **Implement/Discuss:** If you are asked to reimplement the ***tictactoe.py*** **in Lab 1** can it be easily modified for the following requirements:
    a. The size of the board is not 3 by 3. It will be given as an input **n.**
    b. The markers will not be X/O but will be given as inputs for player 1 and player 2.
    c. Try to modify your code and add it to your project if it is easy to do so. If not discuss the reasons.
2. **Try/Discuss:** Download ***namespaces.py*** file from Nexus. Add it to your project and run. id(x) returns the address of x. Discuss the followings by considering the program and the output:
    a. Which variables are in the global namespace?
    b. How many local namespace do we have?
    c. Does the n variable have the same address in the global namespace and local namespace?
    d. Are the changes in the functions reflected outside for n?
    e. Does the a_list variable have the same address in the global namespace and local namespace?
    f. Are the changes in the functions reflected outside for a_list?
    g. What is the difference between n and a_list?
3. **Implement/Discuss:**Download the ***recursive_count.py*** and ***test_recursive_count.py*** and add them to your project. Run both files and observe the results. The first one has functions for counting a target in a given list. One function considers the nested lists but not the other. It has manual tests as well in the main part. The second file is a unittest and provides a couple of the tests by using unittest. Now it is your turn. First write two functions ***recursive_len*** and ***recursive_len_nested*** in a file ***recursive_length.py.*** Both functions

will have a list parameter. They will return the number of items in a given list. Nested version will count the items in the nested list as well. In your main test your functions. Then add ***unittest*** in PyCharm. The file name will be ***test_recursive_length.py***. Test your functions with simple unittests.

4. **Discuss:** You are assigned a project to implement a game. The definition is as follows:
    a. ***Write a board game where we have a farmer, a rabbit, a dog, and many carrots in a farm. The farmer would like to harvest his carrots before the rabbit eats them. The rabbit would like to eat carrots and save its life, since the farmer and dog can catch it. The dog would like to catch the rabbit.***
    b. What do you think about this definition?
    c. What is missing?
    d. Ask questions to clarify the problem.
    e. Think about the design steps. Divide and conquer the problem.

How to turn in this lab

Before turning in any program in this class, remember this mantra:

Just because it works does not mean it's good.

Your grade will also come from the following aspects of your code:

- Submission
- Accuracy/correctness
- Readability
- Neatness
- Presentation
- Style
- Testing
- Commenting

For all labs, turn in only an **electronic** version.

Please submit the followings after all labs:

- zip file of your project (the project folder, not just the .py file(s)): zip file name will be your **YourFirstNameLastName.zip**
- a single pdf file of all your codes (.py), screenshots of your output for each step, answers of the discussion question: pdf file name will be **YourFirstNameLastName.pdf**

Submit

- **the zip and pdf files**
- after Monday Lab session until Tuesday 8 AM EST
- from the Nexus Lab submission link that will be accessible in Week 2-10

Ask for help if you are having problems!

namespaces.py

```
HOW_MANY = 72

def simple_data(n):
    print("\nInside simple_data function")
    print("=" * HOW_MANY)

    print("Before the assignment")
    print("Variable: {} and value: {} address: {}".format("n", n, id(n)))

    n = 6

    print("After the assignment")
    print("Variable: {} and value: {} address: {}".format("n", n, id(n)))

    return n + 1


def complex_data(n, a_list):
    print("\nInside complex_data function")
    print("=" * HOW_MANY)

    print("Before the assignments")
    print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
    print("Variable: {} and value: {} address: {}".format("a_list", a_list, id(a_list)))

    n = 2
    a_list[n] += 3

    print("After the assignments")
    print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
    print("Variable: {} and value: {} address: {}".format("a_list", a_list, id(a_list)))


# test data
n = 1
a_list = [5, 10, 15, 20]

print("\nBefore simple_data function is called")
print("="*HOW_MANY)
print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
print("Variable: {} and value: {} address: {}".format("a_list", a_list, id(a_list)))

n_new = simple_data(n)

print("\nAfter simple_data function is called")
print("="*HOW_MANY)
print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
print("Variable: {} and value: {} address: {}".format("n_new", n_new, id(n_new)))

print("\nBefore complex_data function is called")
print("="*HOW_MANY)
print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
print("Variable: {} and value: {} address: {}".format("a_list", a_list, id(a_list)))

complex_data(n, a_list)

print("\nAfter complex_data function is called")
print("="*HOW_MANY)
print("Variable: {} and value: {} address: {}".format("n", n, id(n)))
print("Variable: {} and value: {} address: {}".format("a_list", a_list, id(a_list)))
```

recursive_count.py

```
def recursive_count(my_list, my_target):
    """ Finds the frequency of target in a given list
        without considering nested lists

    :param my_list: The list where target will be searched and counted
    :type my_list: list
    :param my_target: the item to be counted inside my_list
    :type my_target: anything
    :returns: the count of the target in the list my_list
    :rtype: int

    """

    # Empty list result is zero: Base case
    if not my_list:
        return 0
    else:
        # If the first is the target
        # result is the count in the rest plus 1
        if my_list[0] == my_target:
            return 1 + recursive_count(my_list[1:], my_target)
        # else skip the first one
        else:
            return recursive_count(my_list[1:], my_target)


def recursive_count_nested(my_list, my_target):
    """ Finds the frequency of target in a given list
        considering nested lists

    :param my_list: The list where target will be searched and counted
    :type my_list: list
    :param my_target: the item to be counted inside my_list
    :type my_target: anything
    :returns: the count of the target in the list my_list
    :rtype: int

    """

    # Empty list result is zero: Base case
    if not my_list:
        return 0
    else:
        # if first is not a list
        if type(my_list[0]) is not list:
            # If the first is the target
            # result is the count in the rest plus 1
            if my_list[0] == my_target:
                return 1 + recursive_count_nested(my_list[1:], my_target)
            # else skip the first one
            else:
                return recursive_count_nested(my_list[1:], my_target)
        # First is a list: Nested
        # Count the target in the first and in the rest, then add them
        else:
            return recursive_count_nested(my_list[0], my_target) \
                + recursive_count_nested(my_list[1:], my_target)
```

```
if __name__ == '__main__':
    lists = [[], [1], [1, 1], [1, 2, 1],
             [1, [1, 2, 3]], [1, [1, 2, 3, 2, 2]]]
    targets = [1, 2]

    for target in targets:
        for list_test in lists:
            print("Count of {} in {} without nested: "
                  "{}".format(target, list_test,
                              recursive_count(list_test, target)))

    print("=" * 72)

    for target in targets:
        for list_test in lists:
            print("Count of {} in {} with nested: "
                  "{}".format(target, list_test,
                              recursive_count_nested(list_test, target)))
```

test_recursive_count.py

```
import unittest

from recursive_count import recursive_count

from recursive_count import recursive_count_nested


class MyTestCase(unittest.TestCase):
    def test_empty_list(self):
        my_list = []
        target = 1
        self.assertEqual(0, recursive_count(my_list, target),
                         str(my_list) + " list has zero target")

    def test_one_list(self):
        my_list = [1]
        target = 1
        self.assertEqual(1, recursive_count(my_list, target),
                         str(my_list) + " list has 1 target")

    def test_two_list(self):
        my_list = [1, 1]
        target = 1
        self.assertEqual(2, recursive_count(my_list, target),
                         str(my_list) + " list has 2 target")

    def test_nested_not_counted(self):
        my_list = [1, [1, 2, 3]]
        target = 1
        self.assertEqual(2, recursive_count(my_list, target),
                         str(my_list) + " list has 2 target")


    def test_empty_list_nested(self):
        my_list = []
        target = 1
        self.assertEqual(0, recursive_count_nested(my_list, target),
                         str(my_list) + " list has zero target")

    def test_one_list_nested(self):
        my_list = [1]
        target = 1
        self.assertEqual(1, recursive_count_nested(my_list, target),
                         str(my_list) + " list has 1 target")

    def test_two_list_nested(self):
        my_list = [1, 1]
        target = 1
        self.assertEqual(2, recursive_count_nested(my_list, target),
                         str(my_list) + " list has 2 target")

    def test_two_list_nested_counted(self):
        my_list = [1, [1, 2, 3]]
        target = 1
        self.assertEqual(2, recursive_count_nested(my_list, target),
                         str(my_list) + " list has 2 target")


if __name__ == '__main__':
    unittest.main()
```