420-PRO-LCU Programming in Python - Lab Exercise 7

March 14, 2022

Goals for this lab:

- Practice creating simple functions
- Using for loops over iterables

General Lab Instructions

- In each of these examples you will write your own version of some Python built-in functions. Needless to say, you should *not* use the built-in version in your own code - strictly use variables, `for` loops, `while` loops, `if` statements, etc.
- These functions should *not* change the input argument, which will include a mutable list.
- For this lab, hand in a **single** Python file `Lab7.py` with all of your code, consisting of your functions and at least the test cases shown in the lab handout. Run your code and check that it produces the expected results.
- Make sure to write a meaningful **docstring** for each function.
- For each of the functions below, insert the test cases in the same python file at the bottom and add a comment before the test cases of each function as indicated.

Functions and tests

1. Write a function named `my_len(iterable)` that mimics the function of the Python `len()` function. Note that iterable is a sequence of any type. Your function should handle the test cases listed below.

   ```
   # Test cases for Function my_len
   print(my_len([]))
   print(my_len("Marianopolis College"))
   print(my_len((1,5,9,2,1)))
   ```

   You can also run and test your function with different inputs directly from the python shell.

   ```
   >>> print(my_len([])
   0
   >>> print(my_len("Marianopolis College"))
   20
   >>> print(my_len((1,5,9,2,1)))
   5
   ```

2. Write a function named `my_product(my_list)` that returns the product of all elements of the list `my_list` . It should be defined as follows:

   ```
   def my_product(my_list):
   ```

   It should return the product of the elements of the list. Your function need not handle types that are not numbers (e.g. strings).

   ```
   # Test cases for Function my_product
   print(my_product([1, 5, 9]))
   print(my_product([10, 20, 30.0]))
   print(my_product([])) #prints 0
   ```

   You can also run and test your function with different inputs directly from the python shell.

   ```
   >>> print(my_product([1, 5, 9]))
   45
   >>> print(my_product([10, 20, 30.0]))
   6000.0
   >>> print(my_product([]))
   0
   ```

3. Write a function named `my_count()` that mimics the behavior of the `count()` method. It should be defined as follows:

   ```
   def my_count(my_list, value):
   ```

   It should return the number of times `value` appears in `my_list`.

   ```
   # Test cases for Function my_count
   x = [1, 2, 3, 3]
   print(my_count(x, 1))
   print(my_count(x, 3))
   print(my_count(x, 4))
   ```

   You can also run and test your function with different inputs directly from the python shell.

   ```
   >>> x = [1, 2, 3, 3]
   >>> print(my_count(x, 1))
   1
   >>> print(my_count(x, 3))
   2
   >>> print(my_count(x, 4))
   0
   ```

4. Write a function named `my_find()` that mimics the behavior of the `find()` method, in that it searches a list for a value, returning the *index* of that value if present, or -1 if not.

   ```
   def my_find(my_list, value):
   ```

   It should return the *index* of the first instance of value in the list. If the value is not found, it should return -1.

   ```
   # Test cases for Function my_find
   print(my_find([1, 5, 9, 1], 0)) #returns -1
   print(my_find([1, 5, 9, 1], 9)) #returns 2
   print(my_find([1, 5, 9, 1], 1)) #returns 0
   ```

5. Write a recursive Python function that implements the sum_cubes function as seen in Lab 3.

   ```
   def sum_cubes_r(n):
   # Test cases for Function sum_cubes_r
   print(sum_cubes_r(4))
   print(sum_cubes_r(5))
   ```

   You can also run and test your function with different inputs directly from the python shell.

   ```
   >>> print(sum_cubes_r(4))
   100
   >>> print(sum_cubes_r(5))
   225
   ```

6. Write a a **recursive** Python function named `my_find_r()` that mimics the behavior of the `find()` method, in that it searches a list for a value, returning the *index* of that value if present, or -1 if not.

   ```
   def my_find_r(my_list, value):
   ```

   It should return the *index* of the first instance of value in the list. If the value is not found, it should return -1.

   ```
   # Test cases for Function my_find
   print(my_find_r([1, 5, 9, 1], 0)) #returns -1
   print(my_find_r([1, 5, 9, 1], 9)) #returns 2
   print(my_find_r([1, 5, 9, 1], 1)) #returns 0
   ```