COMP 1510 Programming Methods Lab 06

Christopher Thompson
cthompson98@bcit.ca

BCIT CST — October 2019

Welcome!

This lab is due at the end of your lab period.

Welcome back. For lab six, let's do something fun. Let's use the Python dictionary to store information efficiently. Let's explore sparse vectors.

A sparse vector is a vector whose entries are usually zero. For example, a sparse vector might look like this: [1, 0, 0, 0, 1, 3, 0, 0, 0, 1, 0, 0, 1]. This looks like a good way to use the list data structure. But there's a problem. Storing all those zeros in a list wastes memory.

Instead, we can use a dictionary to keep track of the non-zero entries. We can store the same sparse vector in a dictionary like this: { 0:1, 4:1, 5:3, 9:1, 12:1 }. We record the fact that the only non-zero entries are at indices 0, 4, 5, 9, and 12, and we record the entries at each index. The index is the key, and the entry is the value.

1 Grading

Figure 1: This lab is graded out of 5

This lab will be marked out of 5. For full marks this week, you must:

1. (3 points) Correctly implement the coding requirements in this lab.
2. (1 point) Correctly format and comment your code.
3. (1 point) Correctly generate unit tests for your code.

2 Project Setup

Please complete the following:

1. For your sixth lab, all files must go into the Lab06 folder.
2. After you complete each task, commit and push your change to version control. In order to earn full marks, you must commit and push after each function is complete. Your commit message must be a short message to me that describes what you completed for this commit, i.e., "deconstructed base_conversion", or "debugged function_name", etc.
3. Since I am already a collaborator I will pull your future work automatically.

3 Requirements

Please complete the following:

1. Implement the following function. The name you must use is below. Do not modify the name I have provided.
2. Include correctly formatted docstrings that include comprehensive doctests (where possible) for each function. These docstrings must explain to the user how the function is meant to be used, and what the expected output will be when input is provided.
3. Each function must also be tested in a separate unit test file by a separate collection of unit tests. Ensure your test cases and functions have descriptive, meaningful names, and ensure that every possible valid input is tested for each function.
4. Ensure that the docstring for each function you write has the following components (in this order):
   (a) Short one-sentence description that begins with a verb in imperative tense
   (b) One blank line
   (c) Additional comments if the single line description is not sufficient (this is a good place to describe in detail how the function is meant to be used)
   (d) PARAM statement for each parameter that describes what it should be, and what it is used for
   (e) PRECONDITION statement for each precondition which the user promises to meet before using the function
   (f) POSTCONDITION statement for each postcondition which the function promises to meet if the precondition is met
   (g) RETURN statement that describes what the function returns
   (h) One blank line
   (i) And finally, the doctests. Here is an example:

```
def my_factorial(number):
    """Calculate factorial.

    A simple function that demonstrates good comment construction.
    PARAM: number, a positive integer
    PRECONDITION: number must be a positive integer
    POSTCONDITION: calculates the correct factorial
    RETURN: correctly calculated factorial as an int
    >>> my_factorial(0)
    1
    >>> my_factorial(1)
    1
    >>> my_factorial(5)
    120
    """
    return math.factorial(number)
```

5. The sum of two vectors is just the element-wise sum of their elements. For example, the sum of [1, 2, 3] and [4, 5, 6] is [1+4, 2+5, 3+6] or [5, 7, 9]. **Implement a function called sparse_add in a file called sparsevector.py. The sparse_add function accepts two sparse vectors stored as dictionaries and returns a new dictionary that represents their sum**.
6. Your team lead has asked you to implement a function called sparse_length that will return the length of a sparse vector (in the same manner than Python's len( ) function returns the length of a list). The problem is you can't really do that with the current eway we are storing sparse vectors. Why not? **What do you need to ask your team lead before you can begin?**. Answer these questions in a comment at the top of your source file.
7. That's it. Not so hard, was it?