CS 4710: Artificial Intelligence

P0: Python Review

- Spring 2020
- Due: 11:59pm Tuesday, January 21st, 2020

Submission Information

This assignment should be completed individually. You are encouraged to consult online sources and post questions on Piazza. You are allowed, and encouraged to, discuss problems with your peers; however, you may not share code. Submit your code to UVACollab by the due date. If you still have problems after consulting Python resources, you are encouraged to consult your peers, come to office hours, and use the Piazza discussion forum. Submissions not following the requirements *will not be graded* so make sure to ask quesitons before the deadline.

Good luck!

Written Assignment (5 points)

Please go to the PDF in the written folder for the written instructions. You can use the provided latex template or other tools to write up your solutions and *submit the pdf file to UVACollab*.

Problem 1 (2 points)

Problem 2 (1 point)

Problem 3 (1 point)

Problem 4 (1 point)

Coding Assignment (15 points)

Files you'll edit (and thus submit to UVACollab for grading):

`util.py` Where all of your code for today will reside.

Welcome (back) to Python! The purpose of this warm-up assignment is to refresh your programming skills and help you recall some of the important concepts, such as:

- recursion vs. iteration,
- classes and instances,
- anonymous lambda functions,
- function mapping,
- destructive vs. nondestructive functions, and
- data structures

It is very important to complete this assignment on time because Python programming ability will be required for all subsequent assignments.

If you wish to start working on this assignment before the Python review session, go ahead and start planning out the code. Here are some resources you may find helpful:

http://wiki.python.org/moin/BeginnersGuide

Complete the definitions of the required classes and functions by editing `util.py`. Do all six problems. Unless otherwise noted, your functions should be non-destructive, meaning it should not change the object passed as arguments to functions. Always aim to make your programs elegant and include comments for all of your functions. Your solutions must adhere to the function specifications as they appear below. Your functions need not perform error checking, but must work for all reasonable input.

Problem 1 (2 points)

Write a function called `matrix_multiply` which multiplies two 2-dimensional lists of real numbers. For instance,

```
from util import Util
ut = Util()
mm = ut.matrix_multiply([[1, 2], [3, 4]], [[5, 6], [7, 8]])
# should evaluate to
assert(mm == [[ 19, 22],[43, 50]])
```

Your function should return the result matrix as a 2-dimensional list

- *Note 1*: Your method should support general NxM matrices ("2-dimensional" above refers to the fact that they are matrices, not vectors or tensors).
- *Note 2*: You may assume that the dimensions of the input arrays are compatible.
- *Note 3*: In real life you would use numpy or a similar library to handle matrix math, but for this exercise, please implement the method yourself.

If you are not familiar with lists in Python, you may want to read about them here:

http://www.linuxtopia.org/online_books/programming_books/introduction_to_python/python_tut_18.html

http://docs.python.org/library/functions.html#list

Problem 2 (2.5 points)

Complete the definitions of the methods `init`, `push` and `pop` in the Python classes `MyQueue` and `MyStack`. The operation pop should return, not print, the appropriate object in the structure. If empty, it should return None instead of throwing an error. On the other hand, operation push does not have to return anything. The underlying data structure you use to implement the functions are up to you! An example behavior is as follows:

```
q = ut.MyQueue()
q.push(1); q.push(2)
# should evaluate to
assert(q.pop() == 1)
```

If you are unfamiliar with class declarations in Python, read about them here:

http://docs.python.org/tutorial/classes.html

Problem 3 (1.5 points)

For the two classes written in Problem 2, override **eq**, **ne**, and **str** methods. You can read about these methods in detail here:

http://docs.python.org/reference/datamodel.html#basic-customization

Simply put, the above methods do the following:

- **eq**(self, other) returns True if self and other are `equal`.
- **ne**(self, other) returns True if self and other are `not equal`.
- **str**(self) **returns** a string representation of self.

We will call two stacks or two queues *equal* if and only if they contain the same elements and in the same order. You may assume that only elements they contain are integers.

Problem 4 (4 points)

Write three functions called `add_position_iter`, `add_position_recur`, and `add_position_map`, using iteration, recursion, and the built-in map function, respectively. All the versions should take a list

of numbers and return a new list containing, in order, each of the original numbers incremented by the position of that number in the list. Positions in lists are numbered starting with 0, so:

```
ret = ut.add_position_iter([7, 5, 1, 4])
assert(ret == [7, 6, 3, 7])
```

Remember that this function should not be destructive i.e.,

```
a = [7, 5, 1, 4]
ret = ut.add_position_iter(a)
assert(a != [7, 6, 3, 7])
```

Furthermore, your function should also take an optional argument `number_from` (default value = 0) that can be used to specify a different value from which to start numbering positions. In other words, instead of incrementing the first element by 0, the second by 1, etc., the function will increment the first element by the value of `number_from` , the second element by `number_from+1` ,etc. For example:

```
ret = ut.add_position_iter([0, 0, 3, 1], number_from=3)
assert(ret == [3, 4, 8, 7])
```

Here are some useful tutorials on recursion and map-reduce.

Problem 5 (3 points)

Write a function called `remove_course` which takes in `roster` , `student` , `course` as arguments and returns a modified roster. This function, unlike ones you've just written, should be **destructive**, meaning that roster should be modified directly.

- `roster` will be of type dictionary in Python. It will map a string (i.e., a student name) to a set. Each set will contain strings, representing courses the corresponding student is currently taking.
- `student` will be of type string; it will represent the name of the student.
- `course` will also be of type string.

You can read more about set and dict below:

http://docs.python.org/tutorial/datastructures.html#sets

http://docs.python.org/tutorial/datastructures.html#dictionaries

An example behavior is as follows:

```
roster = {'kyu': set(['cs182']), 'david': set(['cs182'])}
ut.remove_course(roster, 'kyu', 'cs182')
assert(roster == {'kyu': set([]), 'david': set(['cs182'])})
```

Problem 6 (2 points)

Now write a function called `copy_remove_course`. The specifications are the same as above except the function should now be **non-destructive**. An example behavior is as follows:

```
roster = {'kyu': set(['cs182']), 'david': set(['cs182'])}
new_roster = ut.copy_remove_course(roster, 'kyu', 'cs182')
assert(roster == {'kyu': set(['cs182']), 'david': set(['cs182'])})
assert(new_roster == {'kyu': set([]), 'david': set(['cs182'])})
```

Hint: *copy* the original roster and apply remove_course from Problem 5.

You can read more about the copy module below: http://docs.python.org/2/library/copy.html

REMEMBER TO SUBMIT TO UVACollab!!!!!