CS 182: Intelligent Machines: Reasoning, Actions and Plans
Assignment 0: Python Review
Fall 2013

Due: September 10, 5pm

Introduction

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 presumed in all subsequent assignments.

Getting started

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:

- https://www.dropbox.com/s/9eamgwnohs82e8c/tutorial.html
- http://wiki.python.org/moin/BeginnersGuide

To get started: Create a directory called asst0. Copy the util.py file provided on the website into your directory.

Open a command terminal and cd into your asst0 directory. You may then start a new Python interpreter session by typing python in the terminal. Once you have started a Python session within your asst0 directory, complete the definitions of the classes and functions by editing util.py. Alternatively, you can use any other editor and only use the command line to run the files. To test your code, add a main function to util.py, and add to it a call the method you want to test. Then, run util.py (either through an IDE or through the command line or python interpreter).

Problems

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

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

```
>>> util.matrix_multiply([[1, 2], [3, 4]], [[4, 3], [2, 1]])
```

should evaluate to:

```
>>> [[8, 5], [20, 13]]
```

You may assume that the dimensions of the input arrays are compatible. Your function should return the resultant matrix. 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

Complete the definitions of the methods init, push and pop in the Python classes MyQueue and MyStack, using a deque data structure in Python. 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. An example behavior is as follows:

```
>>> q = util.MyQueue()
>>> q.push(1); q.push(2)
>>> q.pop()
1
```

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

- http://docs.python.org/library/collections.html#collections.deque
- http://docs.python.org/tutorial/classes.html

Problem 3

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. You may decide on whatever representation you want, but ideally the method should at least print the elements.

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

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:

```
>>> add_position_iter([7, 5, 1, 4])
[7, 6, 3, 7]
```

Remember that this function should not be destructive i.e., if a = [7, 5, 1, 4], then a should not equal [7, 6, 3, 7] after the function is called on a. Furthermore, your function should also take an optional argument number_from which, although its default value should be 0, can be used to specify a different value from which to start numbering positions, such that 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:

```
>>> add_position_iter([0, 0, 3, 1], number_from=3)
[3, 4, 8, 7]
```

If you have not encountered optional arguments before, you can read about them here:

- http://docs.python.org/tutorial/controlflow.html#default-argument-values

More information about map function and lambda forms can be found here:

- http://docs.python.org/library/functions.html#map
- http://docs.python.org/tutorial/controlflow.html#lambda-forms

Problem 5

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'])}
>>> util.remove_course(roster, 'kyu', 'cs182')
{'kyu': set([]), 'david': set(['cs182'])}
>>> roster
{'kyu': set([]), 'david': set(['cs182'])}
```

Problem 6

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 = util.copy_remove_course(roster, 'kyu', 'cs182')
>>> roster
{'kyu': set(['cs182']), 'david': set(['cs182'])}
>>> 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

Submission

The assignment should be completed individually. You are encouraged to consult online sources and post questions on the i-site forum. You are allowed, and encouraged to, discuss problems with your peers; however you may **not** share code. Submit your modified util.py file to the dropbox folder on the course i-site. If you still have problems after consulting Python resources, you are encouraged to consult your peers and use the i-sites discussion forum.

Good luck!