Introduction to Artificial Intelligence – EECS 348
Programming Assignment 1 – Intro to Python
Due Wednesday April 8th at 11:59pm
(to be completed SOLO)

In this course you will complete your programming assignments in Python, a language that I think you'll find quite useful to know. The purpose of this assignment is to help you get comfortable programming in Python, but more importantly, get you started using the Python Documentation (found here: http://www.python.org/doc/).

Getting Started (this section reviews the python basics covered on day1 of the class)

Python is installed on all of the lab machines in EECS (http://www.eecs.northwestern.edu/student-lab-hostnames2) in /home/software/python-2.7.9, but I recommend that you also install it on your own computer. You can obtain Python at www.python.org/download/. I will be running python 2.7.9 to grade your work and it is critical that you use python 2.7.9 also.

After installing Python you can start python by opening a command window and typing "python" at the prompt. Of course you need to add Python to your path if you are not running from within Python's directory.

Try the following, pressing enter after each one. Notice that you do NOT need semi-colons at the end of each line, nor do you need to declare variables before you use them (Python has implicit typing).

```
>> 10**4

>> a = 'ai is cool'

>> len(a)

>> a.split()

>> myL = [1, 2, 3, 4]

>> myL[1:3]

>> myL[1:]
```

Can you understand what each of the above lines of code does? You'll get more practice with lists and strings later, but you may also want to check out the documentation.

Of course, you will usually want to save the programs you write. Like in almost any other language, you can write programs in a file and then load them into the Python interpreter. In the labs, you can use "Aquamacs."

Open your editor of choice and type in (or copy) the following code that defines a simple list search function:

```
def listSearch( L, x ):
    """ Searches through a list L for the element x"""
    for item in L:
        if item == x:
            return True    # We found it, so return True
```

```
    return False        # Item not found
```

Save your file as "search.py".

Notice a few things about the syntax of this function:

- Python uses colons and indentation to indicate blocks. Indentation is critical in Python. The colon at the end of a line indicates the start of a block of code, all of which must be indented to the same degree. For example, the "return False" line is outside the for-loop because it is indented at the level of the function body, not the for-loop body.

- Variables are not declared in Python. The L is understood to be a list and the x an element in the list, but it's up to the user to enforce this. Return values are also not specified. If a function includes a return statement, that's what it returns. If it does not, then it returns nothing (specifically, the None value).

- The for-loop in Python is a little different from what you might be used to. It specifies that the variable “item” should take on each value in the list L. The "range" function is very useful in getting for-loops in Python to behave like "normal" for-loops. (See the documentation on for-loops).

- The line in quotes at the top of the function is a special comment called a "docstring". It is displayed when the user asks for documentation about this function by typing help(listSearch) at the Python prompt. (Try this below). The parts of the line starting with the symbol # are regular comments.

To run this function, you first need to load the code into the interpreter. In a command prompt navigate to the directory containing the file you just created and run Python (by typing "python" and pressing <enter>). At the interpreter (the >>) type:

```
>> execfile(“search.py”)
```

Now that the program is loaded you can run any functions defined in the file (in this case just one). Try it out, e.g.:

```
>> listSearch([2, 5, 1, 6, 10], 1)
True
```

The Assignment! Programming in Python

You're now set up and ready to go. Complete each of the following problems. Complete the following problems in a file called *your-netid*-pa1.py. For example, my submission would be called *sho533-pa1.py*.

IMPORTANT:

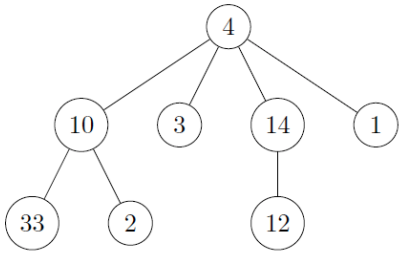
I. Be sure to include **docstrings** and **comments** for every function and class you write (you will lose points on this and all future assignments if you do not include these).
II. You **must** name your functions and classes exactly as defined in this assignment, with the same number of parameters, in the same order as described (you will lose points on this and all future assignments if you fail to do so).
III. Your code MUST compile. We will not give partial credit for code that doesn’t compile.
IV. Do NOT include any additional print statements, etc (for example, for debugging) when you submit your code EXCEPT WHEN SPECIFIED in the assignment (for example, as bfs and dfs specify to do so below).
V. Include your name and netid in a comment at the top of every file.

VI. Notice that I have provided a file called pa1tests.py and another called pa1tests-output.txt. These files are intended to show you how I will call your code, and what I expect your code to output. They are not intended to be exhaustive, but rather just to provide you with some testing. Please design more tests for your code as you see fit.

Problem 1: Write a function called twoToTheN(n) that calculates 2^n in log(n) time. You can expect that n is 0 or greater. You may NOT use math.pow for this problem.

Problem 2: Write two functions, one called mean(L) and the other called median(L) that return the average and the median of a list of numbers, respectively. You may implement this functions recursively or iteratively. (Note – for a list with an even number of elements, let the median be the average of the two central items.)

Problem 3: Assume a tree is represented as a nested list of lists. E.g., the tree



Would be represented as the list [4, [10, [33], [2]], [3], [14, [12]], [1]].
Notice that this tree is neither binary nor ordered in any way and that the leaves are all lists of length 1. Write two functions: bfs(tree, elem) and dfs(tree, elem) that perform a breadth first search and depth first search, respectively, of the tree and return whether or not 'elem' is in 'tree.' You will probably want to read about how lists can be used as Stacks and Queues in the Python documentation (or it may be intuitive to you to just create Stack and Queue classes).
IMPORTANT: to demonstrate that your functions actually perform breadth first, and depth first search, you must print each element in the tree just before it has been examined (on a line by itself). See an example of what this looks like as output in the file pa1tests-output.txt.

Problem 4: In this problem you will develop a class for managing a game of Tic Tac Toe. While we covered the basics of creating classes in Python in class, you will probably want to refer to Python's documentation on creating classes. Create a class called TTTBoard that defines the following functions:

- __init__(self): Initialize a 3x3 tic tac toe board. The board should contain a single attribute, a list, that initially contains nine '*' characters. This list contains the contents of the board. An '*' denotes that this position has not yet been claimed by 'X' or 'O'. Again, this is simply a flat list of 9 items, not a list of lists.
- __str__(self): Returns a string representation of the board. Instead of just displaying a flat list of the 9 items, I want you to display this list like a tic tac toe board, with 3 rows of 3 items each. See the test file output for what this looks like.
- makeMove(self, player, pos): Places a move for player in the position pos (where the board squares a numbered from left to right, starting in the top left square with 0, and beginning at the left in each new row), if possible. 'player' is a character ("X" or "O") and pos is an integer. Returns True if the move was made and False if not (because the spot was full, or outside the boundaries of the board).
- hasWon(self,player): Returns True if player has won the game, and False if not
- gameOver(self): Returns True if someone has won or if the board is full, False otherwise
- clear(self): Clears the board to reset the game

You may define other functions as well. You may wish also to define a function that allows two players to play the game to make sure your above functions are working correctly.

SUBMIT

Remember to name your file *your-netid*-pa1.py. For example, my submission would be called *sho533-pa1.py*. Upload it to canvas before the deadline listed.