University of Pennsylvania
CIT 592: Fall 2011
Python Project: Hanoi Puzzle
Instructor: Donna Dietz

You should first read in your text/texts and/or on Wikipedia for relevant background material. Your classnotes should also prove useful. The code for this project should be roughly the same length as your RSA project was. The play() method will not be tested using the tester, but it will be tested by hand, to see if the play is smooth. You may take liberties with this method to add your own personal touch!

Your project file will be named **Hanoi.py**. The name of the tester file is **testHanoi.py**. You will want to import random, copy, and math.

For this project create these functions:

def int2baseTwo(x): x is a positive integer. Convert it to base two as a list of integers in reverse order. For example, int2baseTwo(6) = [0, 1, 1]. This is the same as the int2baseTwo from your RSA project.

def randLocationR(n): returns a random location (in rollcall format) within a size n Hanoi puzzle. This is returned as a list of length n containing random digits between 0 and 2 inclusive.

def rollcall2List(R): this returns a list of 3 lists, representing the physical Hanoi puzzle. The integers 1 through n are distributed throughout the three sublists, and they are stored in increasing order within each sublist. For exapmle, $L = [[1,3,5],[2,4],[6]]$ represents discs 1, 3, and 5 on the first post, discs 2 and 4 on the second post, and disc 6 (the largest) on the last post. Note that disc one is always on the top of some post.

def isLegalMove(L,a,b): returns True if it is legal to move a disc from post a to post b. L is the list of lists as described in the rollcall2List method. Returns False if move is illegal. If a==b, this may return either True or False.(This design decision will be yours to make.)

def makeMove(L,a,b): if it is legal to move the top disc from post a to post b, this method makes that move happen, otherwise it does nothing. The actual list structure is changed. Nothing is explictly returned.

def printList(L): this prints the list of lists to the screen in a useful way for game play. The posts are labeled "1", "2", and "3", and the display should be easy to understand. This method should be able to handle single and double digit values without making a mess of the display. (So, test it with a size 20 Hanoi game,for example.)

def list2Rollcall(L): this method takes in a list of lists as described earlier, and converts that list into rollcall format. Rollcall always starts with the largest disc and ends with the smallest disc. For example, $L = [[1, 3, 5], [2, 4], [6]]$ as input would return $[2, 0, 1, 0, 1, 0]$

def rollcall2SA(R): this method takes in a Hanoi puzzle configuration in rollcall format and converts it to Sierpinski Address format. See class notes.

def SA2rollcall(SA): this method is the inverse function of rollcall2SA. I leave it as a problem for the student to figure out how to accomplish this task!

def SA2TA(SA): this method takes in a Hanoi puzzle configuration as a Sierpinski Address and returns a Ternary Address for it. See class notes.

def TA2SA(TA): this method is the inverse function of SA2TA. I leave this as a problem for the student, with the hint that int2baseTwo will be useful!

def reduceTA(A,B): this method takes in two Hanoi puzzle configurations in Ternary Address format and reduces it by "removing" any large dics which are in common. For example, if the 4 largest discs in both configurations are in the same locations, they are irrelevant to the problem. The two inputs are directly changed, and nothing is explicitly returned by this method. Thus, the resulting two configurations will have their largest discs on two different posts.

def distTA(A_in,B_in): this method first makes deepcopies of the inputs, because you do not want a distance measure to change the actual input data. If the two inputs are identical, 0 is immediately returned. Otherwise, the problem is reduced using the reduceTA method, and distance is calculated by the technique shown in the class notes.

def play(): allows the player to mess around with these fun puzzles! Rather than the usual game people typically play (which is to start with all the discs on one post and move them to another post), we will start with a random configuration. So, for small values of n, you may even get to start with a solved puzzle! The user should have a mechanism for quitting early, and of course, the fun part is getting the distance measure to work correctly! You should report the minimal distance to the solution after each move. Once your distance measure is working, it adds more fun and reduces stress in this game! You are encouraged to start writing this well before you have the distance measure finished, because you can use this method to help you debug earlier methods! If an illegal move is attempted, the player should be notified, but the configuration should not change. The user should be prompted for which post the disc should be taken from and which post the disc should be moved to. (As an optional variation, you may program the input pattern to be something else, such as "Which disc do you want to move?" and "Which disc should it be placed on top of?" However, this should be named something different such as play2().) Play should terminate when a puzzle is solved or when the user asks to quit. You may heckle the quitter as desired or congratulate the winner.