CS 110 Computational Thinking I *Assigned: Friday, 24 Feb.*

Programming Assignment 2: Logic Dots Lab

Spring 2017 *Due: 11:00 am Friday, 10 Mar.*

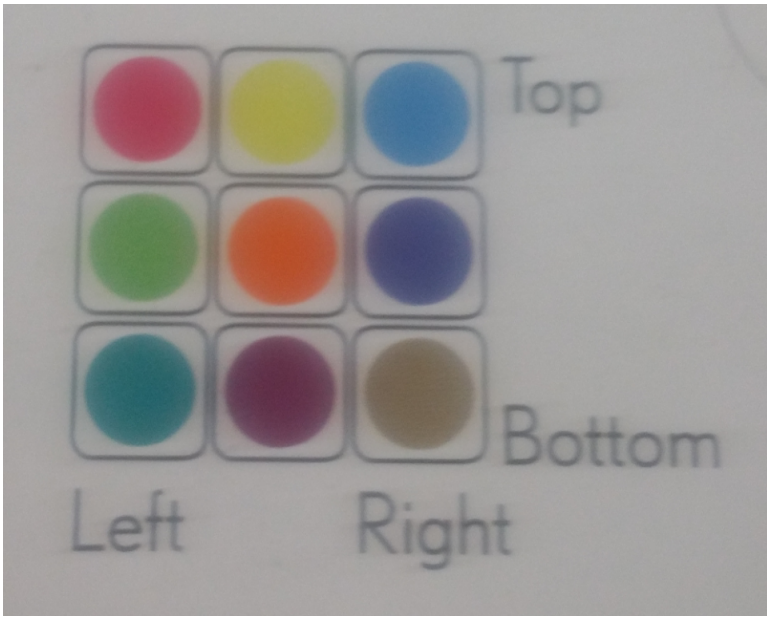
To facilitate grading and timely feedback
please note that all submissions are through Gradescope.
The filename of the python file should be: ***submission.py***

Disclaimer: The Gradescope Autograder script uses the python 2.7 compiler.
This shouldn't be much of a problem since there are very minor changes in syntax. However, in case you have any problems with the file working fine on your pc and giving syntax error on Gradescope. Contact your instructor or Syed Shariq Ali (sa00509@st.habib.edu.pk)

Introduction

Having already been introduced to the game of LogicDots in your lab session, you must have enjoyed solving the logic puzzle. Given a list of rules, place tiles on the board in such a way which satisfies all the rules and in doing so, find the position of the golden tile. Now imagine that if instead of a 3x3 board this were a 5x5 board and you were given 20 rules to satisfy. Would it still be easy for you to solve? What about a 10x10 board? This is where computers come in!

Over the next 2 weeks you'll be building a program which when given a list of tiles (colors) and a list of rules, will output the solution for a 3x3 board, which can easily be extended to an NxN board.



A board like this will be represented using the variable **board** in your program, (a nested 3x3 list) for instance the board in the picture will be represented in your code at the very top using:

board=[[“Pink”,“Yellow”,“LightBlue”],[“Green”,“Orange”,“DarkBlue”],[“Teal”,“Purple”,“Gold”]]

The above board will be used as an example in all the given questions.

Now you’ll be implementing functions which will test whether a given board satisfies a certain rule or not.
Some definition of the terms used in these rules (using the above board) are:

1. **Top,bottom, left, right:**
   The pink tile is on the left and at the top.
   The golden tile is on the right and on the bottom.

2. **Rows and Columns.**
   The pink and teal tiles are in the same column.
   The yellow and the light blue are in the same row.

3. **Between**
   The orange tile is between the purple and yellow tiles. The orange tile is NOT between the teal and light blue tiles.

4. **Touching**
   The green tile is touching the pink, orange and the teal tiles.
   The green tile is NOT touching the purple or yellow tiles.

1 PART 1: Basic Functions. Should ideally be completed by Wednesday, 1 Mar.

1. **(3 marks)** Define a function **getPositions** which takes the **board** and **color** as arguments and returns a list of all positions (tuple containing the row,col) at which this color is on in the board. `https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences` `https://en.wikibooks.org/wiki/Python_Programming/Tuples`

   getPositions(board,"Yellow") on above board should return [(0,1)]
   getPositions(board,"Red") on above board should return []

2. **(4 marks)** Define a function **isTouching** which takes the **board**, **color1**,**color2** as argument and returns whether any tile colored "color1" is touching any other tile colored "colored2"

   On the example board, Orange is touching Yellow, however Pink is not touching Teal so:
   isTouching(board,"Orange","Yellow") should return True
   while isTouching(board,"Pink","Teal") should return False

3. **(3 marks)** Define a function **sameRow** which takes the **board** ,**color1** and **color2** as argument and returns whether there exists any row in which the two colored tiles are in the same row.

   The yellow and light blue tiles are in the same row while the pink and purple tiles aren't.
   sameRow(board,"Yellow","Light Blue") should return True
   sameRow(board,"Pink","Purple") should return False

4. **(3 marks)** Define a function **sameColumn** which takes the **board**, **color1** and **color2** as argument and returns whether the two colored tiles are in the same column.

   The pink and teal tiles are in the same column while yellow and gold aren't.
   sameRow(board,"Pink","Teal") should return True
   sameRow(board,"Yellow","Gold") should return False

5. **(3 marks)** Define a function **inRow** which takes the **board**, **pos** (top,middle,bottom), **color** and **N** as arguments and checks whether at least N number of "color" tiles are there in top/middle/bottom row or not.

   inRow(board,"top","Yellow",1) should return True as there is at least one yellow tile in the top row.

6. **(3 marks)** Define a function **inColumn** which takes the **board**, **pos** (left,middle,right), **color** and **N** as arguments and checks whether at least N number of "color" tiles are there in column "pos" or not.
   There is at least one gold tile in the right column so:
   inCol(board,"right","Gold",1) should return True

7. **(4 marks)** Define a function **isBetween** which takes the **board**, **color**, **betweencol1** and **betweencol2** as arguments and checks whether "color" tile is between tiles colored "betweencol1" and "betweencol2".

   The Orange tile is between the Yellow and Purple tiles while the Green tile is not between the Purple and Pink tiles.
   isBetween(board,"Orange","Yellow","Purple") should return True
   isBetween(board,"Green","Purple","Pink") should return False

8. **(3 marks)** Define a function **atPlace** which takes the **board**, **color**, **row** and **col** as arguments and checks whether the tile colored "color" is given position or not.

   "The Orange tile is in the middle" should return True.
   "The Purple tile is at the top left" should return False.
   "Dark Blue tile is at the middle left" should return True.
   atPlace(board,"Dark Blue","middle","left") should return True.

9. **(5 marks)** Define a function **isTowards** which takes the **board**, **color**,**direction**,**color2** as arguments and checks whether "color2" is in a specific direction (above,under,left,right) of the tile colored "color" or not.
   Teal tile is towards the **left** of the Gold tile.
   isTowards(board,"Teal","left","Gold") should return True

   Purple tile is **under** the Yellow tile.
   isTowards(board,"Purple","under","Yellow") should return True

   Orange tile is not towards the **right** of the Yellow tile so
   isTowards(board,"Orange","right","Yellow") should return False

   Orange is not **above** the Gold tile so
   isTowards(board,"Orange","right","Gold") should return False

10. **(10 marks) (bonus)** Implement the functionality of "not" in the above functions by modifying them and adding an extra flag parameter, **isTrue** into the above functions. Which should be True by default and when set to False should result in the negation of the actual output.
For instance:

    if someone calls the isTouching function like isTouching(board,Orange,Yellow) this is equal to calling isTouching(board,Orange,Yellow,True) and it should return True. The isTrue argument should be optional, if it isn't given the value should be assumed to be True.

    "The teal tiles does not touch the gold tile"
    isTouching(board,"Teal","Gold",False) should return True

    "The Orange tile is not between the Yellow and Purple tiles." is false
    isBetween(board,Orange,Yellow,Purple,False) should return False

    "The Teal tile is not between the Purple and Gold tile" is true
    isBetween(board,Teal,Purple,Gold,False) should return True
    isTouching(board,Orange,Gold,True) should return False

2 PART 2 Board and Rules. Should ideally be completed by Monday, 6 Mar.

11. **(2 marks)** Implement a function, **getColors**, which takes as argument, **colors**, a string of colors separated by space as argument and returns the list of colors.

12. **(3 marks)** Implement a function, **getRules**, which takes as argument, **rules**, and returns the list of rules.

13. **(5 marks)** Implement a function **inputFromFile** which takes a filename(of a simple file) as argument, reads it and then returns the list of colors and list of rules using the above functions in the form of a tuple (colors,rules).

14. **(4 marks)** Implement a function, **isValidBoard**, which takes the **board** and **colors**, a list of colors, as argument and checks whether the given board is a valid board or not. (ie the list is 3x3 and the board contains only the tiles given in colors"

15. **(10 marks)** Implement a function, **checkRules**, which takes the **board** and **rules** as arguments and checks whether the board satisfies all the rules or not.

3 PART 3 Game Solver Should ideally be completed by Thursday, 9 Mar.

16. **(Optional)** Implement a function "makePermutations" which takes "colors", a list of colors, as input and generates permutations. (You will be using this function to implement the next function. So if you can implement the next function without this function or in some other way. Feel free to do so, we won't be checking this function)

17. **(15 marks)** Implement a function **GameSolver** which takes **filename** of a simple file ("N_simple.txt") as argument, and extracts the colors and rules from the file using previous functions (inputFromFile). Then uses these colors to generate all possible permutations of the board (brute force) and check whether each possible board fulfills all the rules or not (checkRules), when you find a solution, a board which fulfills all the rules, return the solution board, if no solution exists, return an empty list ([]).

18. **(10 marks) (Bonus*)** Implement the function **GameSolver_efficient** which does the same as above but instead of using brute-force, uses some form of reasoning to generate the board. The strategy which is used should be well-documented using comments. The marks for this question will entirely be on instructor's discretion based on how good the strategy is. (You'll only get marks in this question if **GameSolver** has been implemented correctly.)

19. **(20 marks) (Bonus*)** Implement a function **GameSolver_bonus** which does the same as above but instead of taking the **filename** of a simple file as argument, takes the bonus file as argument. (You might have to implement several other helper functions to achieve this! You'll only get marks in this question if **GameSolver** and all previous functions have been implemented correctly.)

***In case you are attempting bonus, submit the above two bonus questions on lms with the filename being "submission_bonus.py".**

Good Luck!