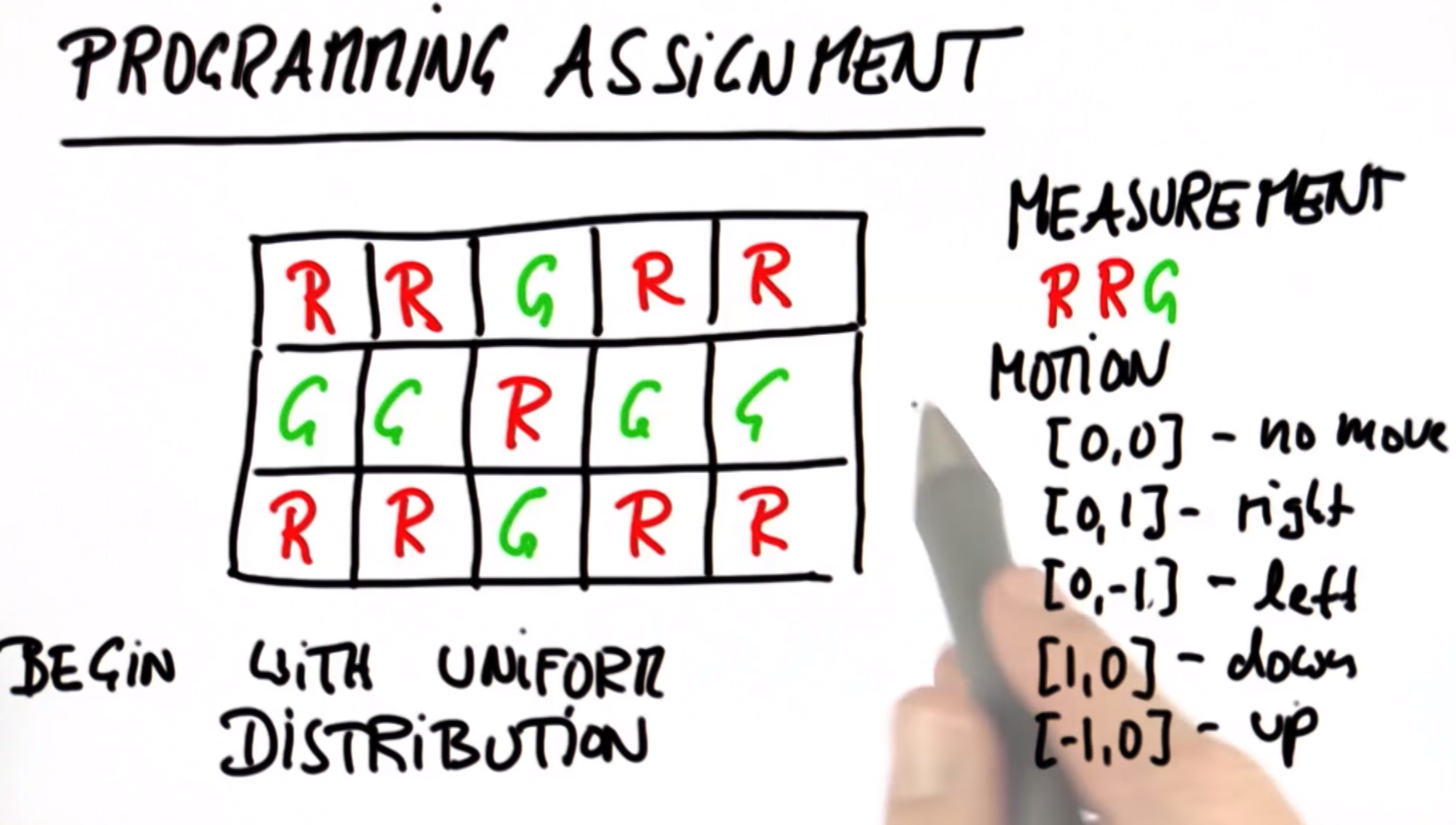
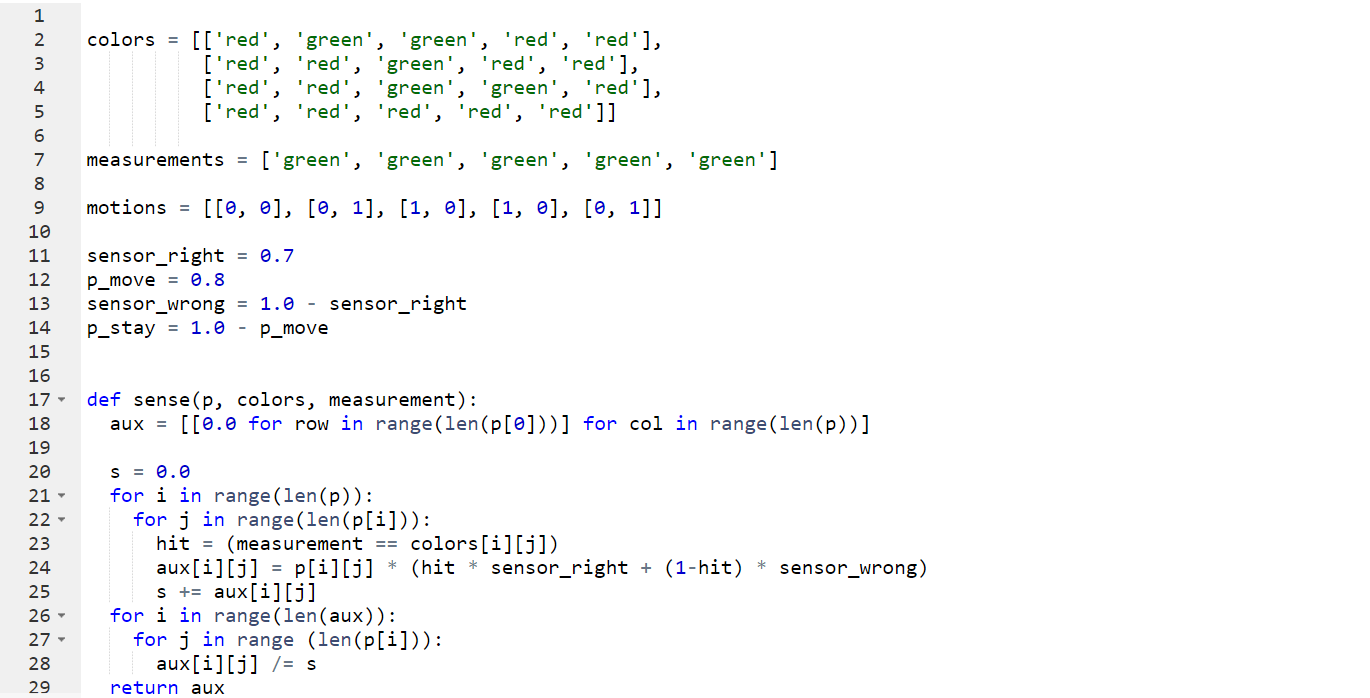
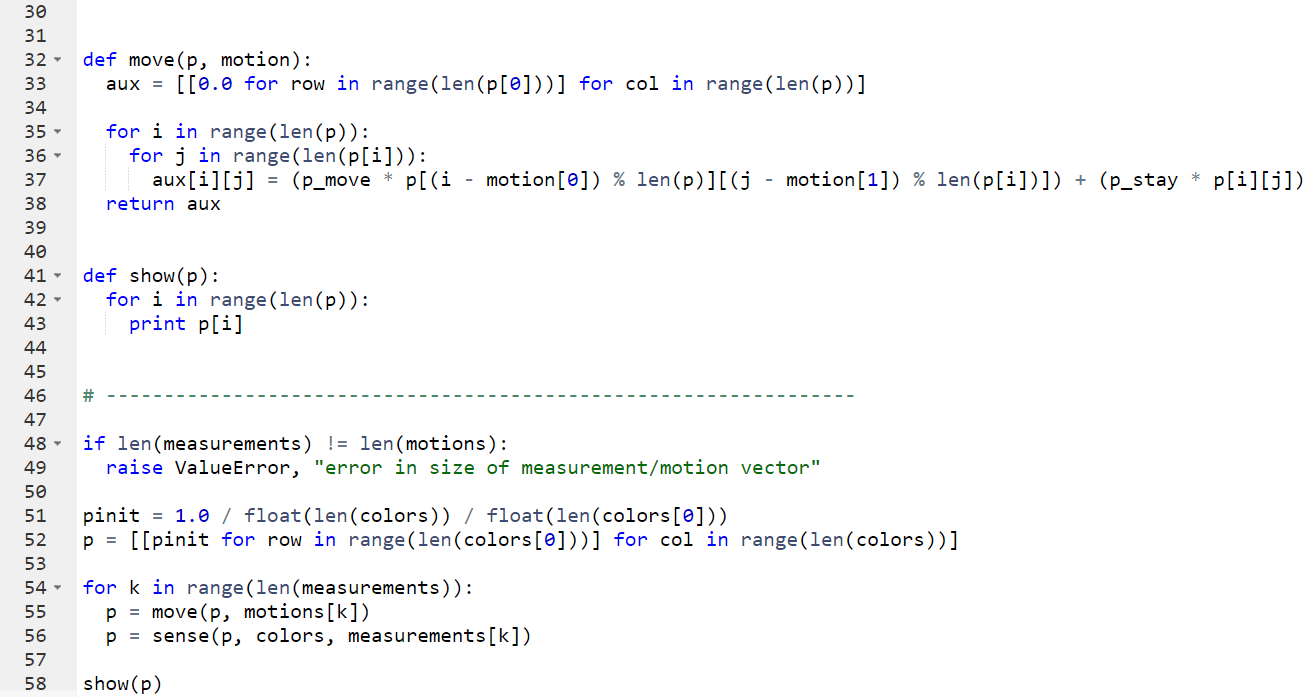
## 4. Quiz: Localization Problem

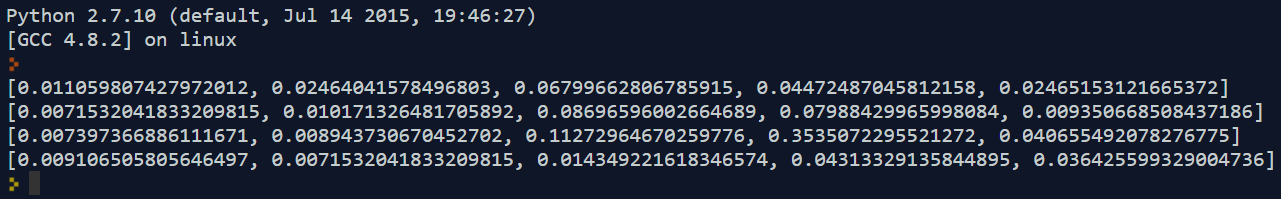


Python Code: (Python 2 online complier <https://repl.it/LNBY>)

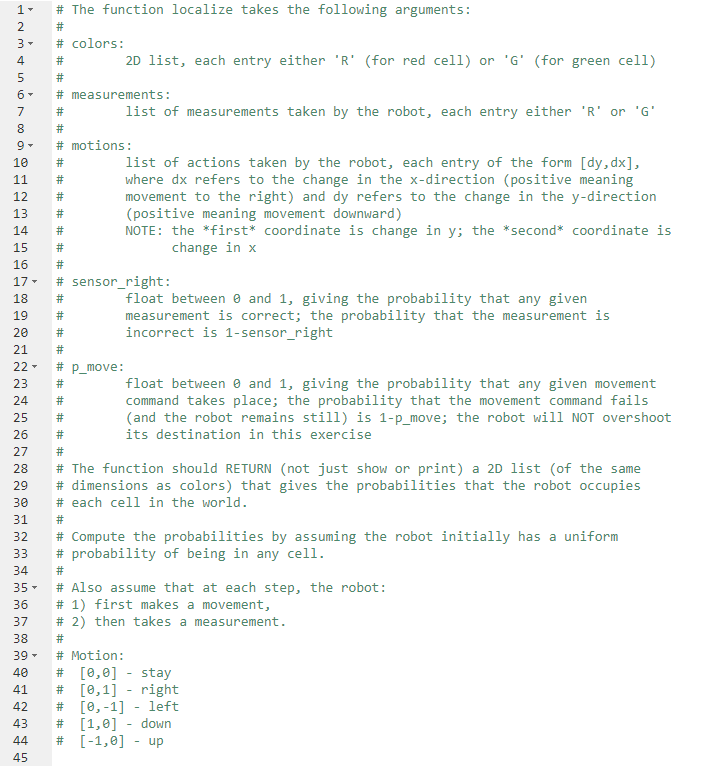
### Solution:

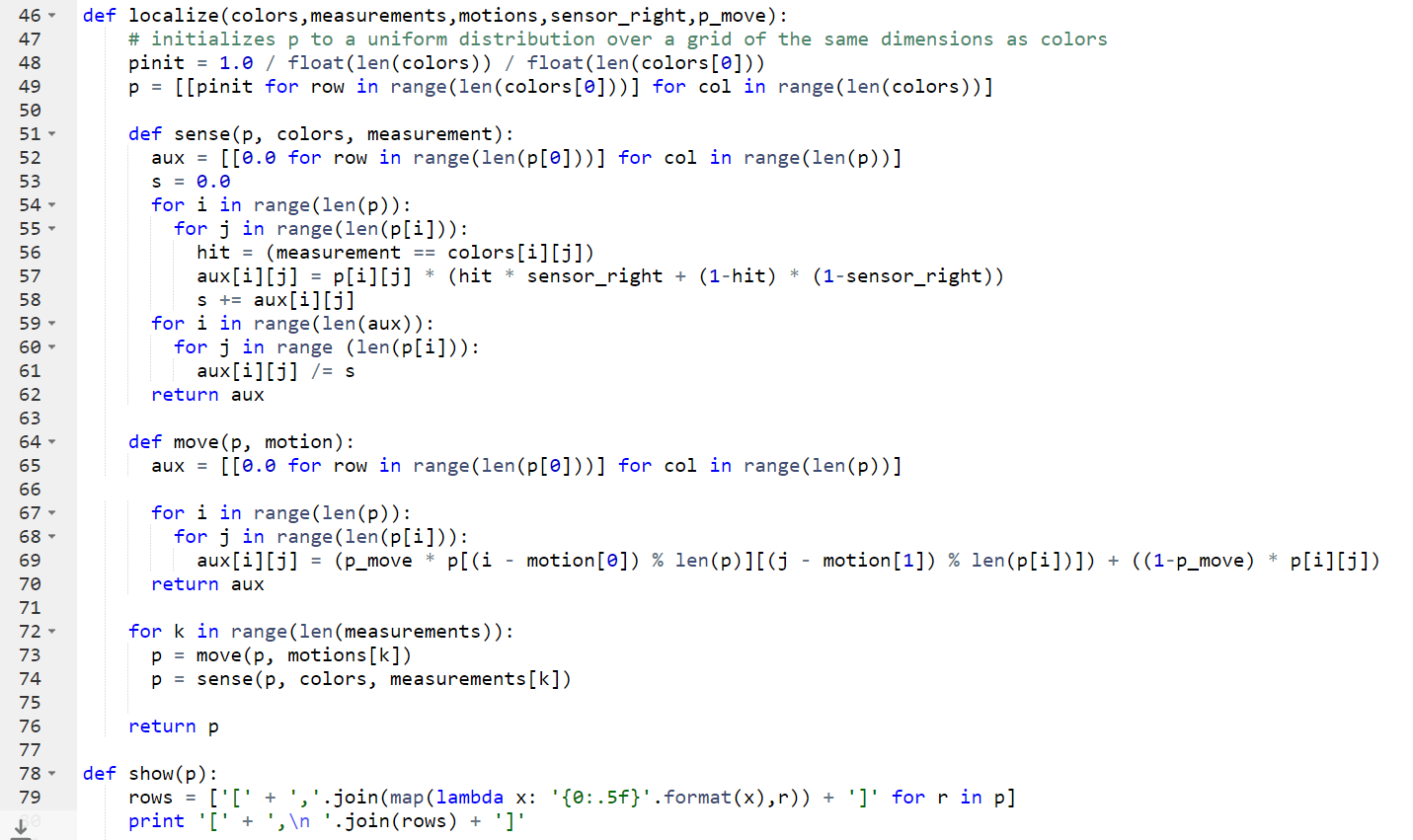


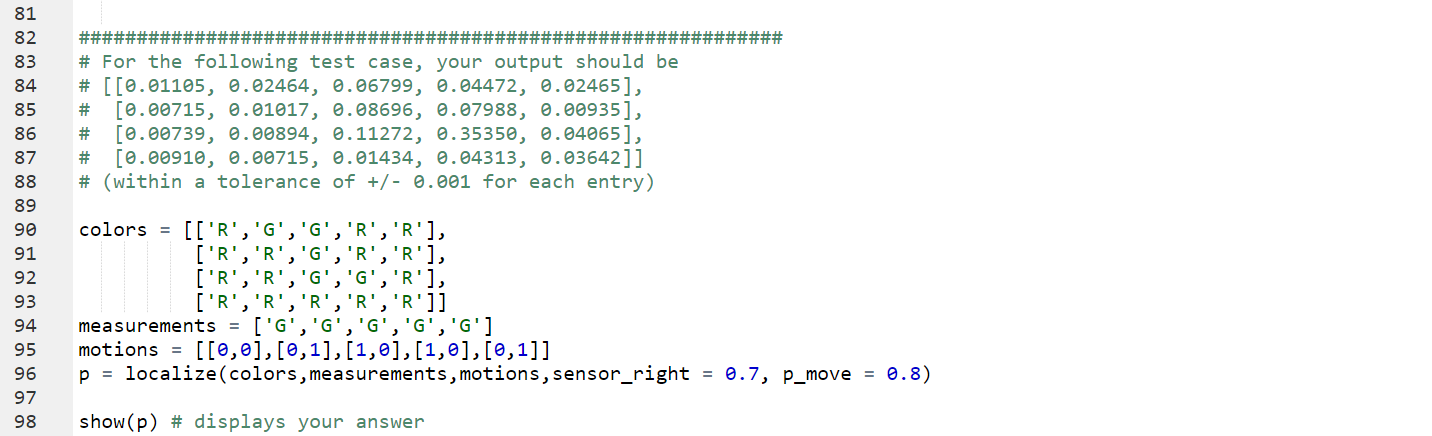




### More generic solution:









### Code for solution:

colors = [['red', 'green', 'green', 'red', 'red'],

['red', 'red', 'green', 'red', 'red'],

['red', 'red', 'green', 'green', 'red'],

['red', 'red', 'red', 'red', 'red']]

measurements = ['green', 'green', 'green', 'green', 'green']

motions = [[0, 0], [0, 1], [1, 0], [1, 0], [0, 1]]

sensor\_right = 0.7

p\_move = 0.8

sensor\_wrong = 1.0 - sensor\_right

p\_stay = 1.0 - p\_move

def sense(p, colors, measurement):

aux = [[0.0 for row in range(len(p[0]))] for col in range(len(p))]

s = 0.0

for i in range(len(p)):

for j in range(len(p[i])):

hit = (measurement == colors[i][j])

aux[i][j] = p[i][j] \* (hit \* sensor\_right + (1-hit) \* sensor\_wrong)

s += aux[i][j]

for i in range(len(aux)):

for j in range (len(p[i])):

aux[i][j] /= s

return aux

def move(p, motion):

aux = [[0.0 for row in range(len(p[0]))] for col in range(len(p))]

for i in range(len(p)):

for j in range(len(p[i])):

aux[i][j] = (p\_move \* p[(i - motion[0]) % len(p)][(j - motion[1]) % len(p[i])]) + (p\_stay \* p[i][j])

return aux

def show(p):

for i in range(len(p)):

print p[i]

# -----------------------------------------------------------------

if len(measurements) != len(motions):

raise ValueError, "error in size of measurement/motion vector"

pinit = 1.0 / float(len(colors)) / float(len(colors[0]))

p = [[pinit for row in range(len(colors[0]))] for col in range(len(colors))]

for k in range(len(measurements)):

p = move(p, motions[k])

p = sense(p, colors, measurements[k])

show(p)

### Code for more generic solution:

# The function localize takes the following arguments:

#

# colors:

# 2D list, each entry either 'R' (for red cell) or 'G' (for green cell)

#

# measurements:

# list of measurements taken by the robot, each entry either 'R' or 'G'

#

# motions:

# list of actions taken by the robot, each entry of the form [dy,dx],

# where dx refers to the change in the x-direction (positive meaning

# movement to the right) and dy refers to the change in the y-direction

# (positive meaning movement downward)

# NOTE: the \*first\* coordinate is change in y; the \*second\* coordinate is

# change in x

#

# sensor\_right:

# float between 0 and 1, giving the probability that any given

# measurement is correct; the probability that the measurement is

# incorrect is 1-sensor\_right

#

# p\_move:

# float between 0 and 1, giving the probability that any given movement

# command takes place; the probability that the movement command fails

# (and the robot remains still) is 1-p\_move; the robot will NOT overshoot

# its destination in this exercise

#

# The function should RETURN (not just show or print) a 2D list (of the same

# dimensions as colors) that gives the probabilities that the robot occupies

# each cell in the world.

#

# Compute the probabilities by assuming the robot initially has a uniform

# probability of being in any cell.

#

# Also assume that at each step, the robot:

# 1) first makes a movement,

# 2) then takes a measurement.

#

# Motion:

# [0,0] - stay

# [0,1] - right

# [0,-1] - left

# [1,0] - down

# [-1,0] - up

def localize(colors,measurements,motions,sensor\_right,p\_move):

# initializes p to a uniform distribution over a grid of the same dimensions as colors

pinit = 1.0 / float(len(colors)) / float(len(colors[0]))

p = [[pinit for row in range(len(colors[0]))] for col in range(len(colors))]

def sense(p, colors, measurement):

aux = [[0.0 for row in range(len(p[0]))] for col in range(len(p))]

s = 0.0

for i in range(len(p)):

for j in range(len(p[i])):

hit = (measurement == colors[i][j])

aux[i][j] = p[i][j] \* (hit \* sensor\_right + (1-hit) \* (1-sensor\_right))

s += aux[i][j]

for i in range(len(aux)):

for j in range (len(p[i])):

aux[i][j] /= s

return aux

def move(p, motion):

aux = [[0.0 for row in range(len(p[0]))] for col in range(len(p))]

for i in range(len(p)):

for j in range(len(p[i])):

aux[i][j] = (p\_move \* p[(i - motion[0]) % len(p)][(j - motion[1]) % len(p[i])]) + ((1-p\_move) \* p[i][j])

return aux

for k in range(len(measurements)):

p = move(p, motions[k])

p = sense(p, colors, measurements[k])

return p

def show(p):

rows = ['[' + ','.join(map(lambda x: '{0:.5f}'.format(x),r)) + ']' for r in p]

print '[' + ',\n '.join(rows) + ']'

#############################################################

# For the following test case, your output should be

# [[0.01105, 0.02464, 0.06799, 0.04472, 0.02465],

# [0.00715, 0.01017, 0.08696, 0.07988, 0.00935],

# [0.00739, 0.00894, 0.11272, 0.35350, 0.04065],

# [0.00910, 0.00715, 0.01434, 0.04313, 0.03642]]

# (within a tolerance of +/- 0.001 for each entry)

colors = [['R','G','G','R','R'],

['R','R','G','R','R'],

['R','R','G','G','R'],

['R','R','R','R','R']]

measurements = ['G','G','G','G','G']

motions = [[0,0],[0,1],[1,0],[1,0],[0,1]]

p = localize(colors,measurements,motions,sensor\_right = 0.7, p\_move = 0.8)

show(p) # displays your answer