

To print higher-resolution math symbols, click the
Hi-Res Fonts for Printing button on the jsMath control panel.

Final Project

class Crowd():

"""

Creates a class which consists of an matrix of people, doors, and bomb.

INPUT:

c - input of people, doors, and bomb

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>print c
```

Bus terminal arranged [[1, 0, 1], [0, 0, 3], [2, 0, 2]]

AUTHORS:

Bill Francis and Thomas Paul

TODO:

Implement probabilities of an individual choosing a particular door based on the distance.

Include families or groups of people moving together within the matrix.

Return matrix plot, instead of just a matrix, for each individual state.

Add objects within the matrix for people to move around.

REFERENCES:

<http://www.usna.edu/Users/math/wdj/>

"""

```
def __init__(self, c):
```

"""

Initializes the input values as the dimensions of the matrix.

"""

```
    self.state = c
```

```
    self.row = len(c)
```

```
    self.column = len(c[0])
```

```
def __repr__(self):
```

"""

Computes the official string representation of a matrix and can be used to

recreate a matrix with the same values.

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c
```

```
Crowd([[1, 0, 1], [0, 0, 3], [2, 0, 2]])
```

```
"""
```

```
return "Crowd(%s)"%self.state
```

```
def __str__(self):
```

```
"""
```

Computes the informal string representation of the matrix.

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>print c
```

```
Bus terminal arranged [[1, 0, 1], [0, 0, 3], [2, 0, 2]]
```

```
"""
```

```
return "Bus terminal arranged %s"%self.state
```

```
def countdoors(self):
```

```
"""
```

Counts the number of doors entered by the user into the matrix

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.countdoors()
```

```
2
```

```
"""
```

```
c = self.state
```

```
numdoors = flatten(c).count(2)
```

```
return numdoors
```

```
def countpeople(self):
```

```
"""
```

Counts the number of people entered by the user into the matrix

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.countpeople()
```

```
2
```

```
"""
```

```
c = self.state
```

```
numpeople = flatten(c).count(1)
return numpeople
```

```
def view(self):
```

```
"""

```

```
    Displays a matrix plot of the matrix entered by the user
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.view()
```

```
    Cannot copy/paste graph into code
```

```
"""

```

```
c = self.state
```

```
return matrix_plot(matrix(c))
```

```
def distance(self,(i,j),(m,n)):
```

```
"""

```

```
    Computes distances from people to doors
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.distance((0,0),(2,2))
```

```
4
```

```
"""

```

```
x = abs(i-m)
```

```
y = abs(j-n)
```

```
d = x + y
```

```
return d
```

```
def peoplelist(self):
```

```
"""

```

```
    Computes the list of people in the matrix
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.peoplelist()
```

```
[[0, 0], [0, 2]]
```

```
"""

```

```
c = self.state
```

```
m = matrix(c)
```

```
v = []
```

```
for i in range(self.row):
```

```
    for j in range(self.column):
```

```
        if m[i,j] == 1:
```

```
v.append([i,j])
```

```
return v
```

```
def doorlist(self):
```

```
"""
```

```
Computes the list of doors in the matrix
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.doorlist()
```

```
[[2, 0], [2, 2]]
```

```
"""
```

```
c = self.state
```

```
m = matrix(c)
```

```
l = []
```

```
for i in range(self.row):
```

```
    for j in range(self.column):
```

```
        if m[i,j] == 2:
```

```
            l.append([i,j])
```

```
return l
```

```
def doordictionary(self):
```

```
"""
```

```
Creates a dictionary of the doors within the matrix
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.doordictionary()
```

```
{0: [2, 0], 1: [2, 2]}
```

```
"""
```

```
d = {}
```

```
for j in range(len(self.doorlist())):
```

```
    a = self.doorlist()[j]
```

```
    d[j] = a
```

```
return d
```

```
def peopledictionary(self):
```

```
"""
```

```
Creates a dictionary of the people within the matrix, and also creates a  
dictionary of the distances to the doors within that dictionary.
```

```
EXAMPLES:
```

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.peopledictionary()
```

```
{0: ([0, 0], {0: 2, 1: 4}), 1: ([0, 2], {0: 4, 1: 2})}
```

```
"""

```

```
d = {}
```

```
b = self.doordictionary()
```

```
e = {}
```

```
for i in range(len(self.peoplelist())):
```

```
    del e
```

```
    e = {}
```

```
    for j in range(len(b.keys())):
```

```
        a = self.peoplelist()[i]
```

```
        c = self.distance(a,b.values())[j]
```

```
        e[j] = c
```

```
    d[i] = a,e
```

```
return d
```

```
def mindistance(self,a):
```

```
"""

```

Calculates the minimum distances from a person to each door

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.mindistance(0)
```

```
2
```

```
"""

```

```
d = self.peopledictionary()
```

```
v = d[a]
```

```
r = v[1]
```

```
e = r.values()
```

```
k = min(e)
```

```
for l in range(len(r.keys())):
```

```
    if r[l] == k:
```

```
        return l
```

```
def movepeople(self):
```

```
"""

```

Creates an output of each individual person's movements throughout the matrix to exit to the doors.

EXAMPLES:

```
>>>c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

```
>>>c.movepeople()
```

```
[0 0 1]
```

```
[1 0 3]
```

```
[2 0 2]
```

```
[0 0 1]
[0 0 3]
[2 0 2]

[0 1 0]
[0 0 3]
[2 0 2]

[0 0 0]
[0 1 3]
[2 0 2]

[0 0 0]
[0 0 3]
[2 1 2]

[0 0 0]
[0 0 3]
[2 0 2]
"""
c = self.state
p = self.peopledictionary()
e = self.doordictionary()
g = matrix(c)
for a in range(len(self.peoplelist())):
    d = self.mindistance(a)
    t = e[d]
    q = p[a]
    y = q[0]
    u = q[1]
    [i,j] = t
    [m,n] = y
    while i != m:
        g[m,n] = 0
        h = m
        if i > m:
            m = m + 1
        elif i < m:
            m = m - 1
        if g[m,n] == 3:
            m = h
            if n == 0:
                n = n + 1
```

```

else:
    n = n - 1
if (g[m,n] != 2) and (g[m,n] != 3):
    g[m,n] = 1
print g
print ''
while j != n:
    g[m,n] = 0
    o = n
    if j > n:
        n = n + 1
    elif j < n:
        n = n - 1
    if g[m,n] == 3:
        n = o
        if m == 0:
            m = m + 1
        else:
            m = m - 1
    if (g[m,n] != 2) and (g[m,n] != 3):
        g[m,n] = 1
    print g
    print ''

```

```
d = {0: ([0, 0], {0: 2, 1: 4})}
```

```
v = d[0]
```

```
r = v[1]
```

```
e = r.values()
```

```
k = min(e)
```

```
for l in range(len(r.keys())):
```

```
    if r[l] == k:
```

```
        print l
```

```
0
```

```
c.movepeople()
```

```
[0 0 1]
```

```
[1 0 3]
```

```
[2 0 2]
```

```
[0 0 1]
```

```
[0 0 3]
```

```
[2 0 2]
```

```
[0 1 0]
```

```
[0 0 3]  
[2 0 2]
```

```
[0 0 0]  
[0 1 3]  
[2 0 2]
```

```
[0 0 0]  
[0 0 3]  
[2 1 2]
```

```
[0 0 0]  
[0 0 3]  
[2 0 2]
```

c.peoplelist()

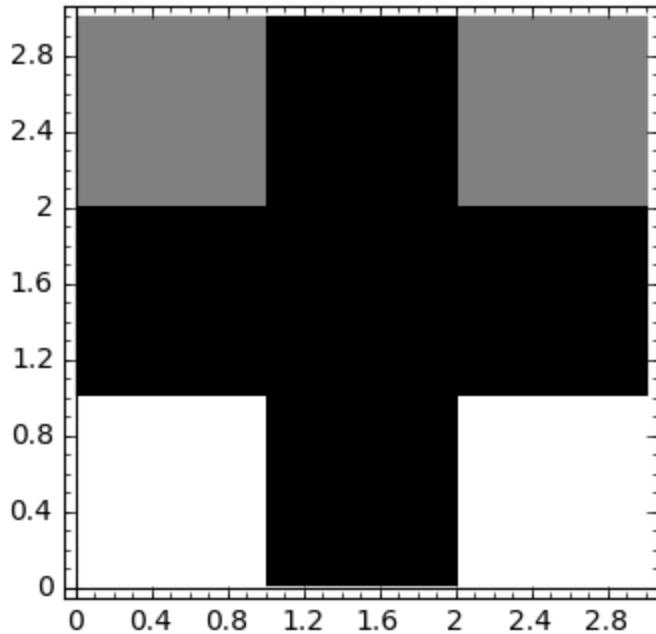
```
[[0, 0], [0, 2]]
```

c

```
Crowd([[1, 0, 1], [0, 0, 3], [2, 0, 2]])
```

```
c = Crowd([[1,0,1],[0,0,3],[2,0,2]])
```

c.view()



c.countdoors()

2

```
c.countpeople()
```

```
1
```

```
c.distance((0,0),(2,2))
```

```
4
```

```
c.peoplelist()
```

```
[[0, 0]]
```

```
c.distance((0,0),(2,2))
```

```
4
```

```
c.doorlist()
```

```
[[2, 0], [2, 2]]
```

```
c.choosedoors()
```

```
4
```

```
c.distance([0,0],[2,2])
```

```
4
```

```
c.peopledictionary()
```

```
{0: ([0, 0], {0: 2, 1: 4}), 1: ([0, 2], {0: 4, 1: 2})}
```

```
c.mindistance(0)
```

```
The closest door to person 0 is door 0.
```

```
c.movepeople()
```

```
[0 0 1]  
[1 0 3]  
[2 0 2]
```

```
[0 0 1]  
[0 0 3]  
[2 0 2]
```

```
[0 1 0]  
[0 0 3]  
[2 0 2]
```

```
[0 0 0]  
[0 1 3]  
[2 0 2]
```

```
[0 0 0]
[0 0 3]
[2 1 2]
```

```
[0 0 0]
[0 0 3]
[2 0 2]
```

```
c.peopledictionary()
```

```
{0: ([0, 0], {0: 2, 1: 4}), 1: ([0, 2], {0: 4, 1: 2})}
```

```
c.choosedoor()
```

```
2
```

```
d = {1:{1:5}}
```

```
{1: 5}
```

```
d
```

```
2
```

```
v[]
```

```
line 4
    v[]
      ^
SyntaxError: invalid syntax
```

```
v = [1]
```

```
b = []
h = []
d = {}
for j in range(len(c.doorlist())):
    for i in range(len(c.peoplelist())):
        a = c.peoplelist()[i]
        b.append(c.doorlist()[j])
        h.append(c.distance(a,b[j]))
        d[i] = [a,h]
```

```
Traceback (click to the left for traceback)
...
```

```
AttributeError: 'list' object has no attribute 'doorlist'
```

```
v.append([1,1])
```

```
v
```

```
[1, [1, 1]]
```

```
v
```

```
_sage_const_0 |--> 1
```

```
m = [[1,0,0],[0,0,0],[0,0,2]]
```

```
m
```

```
[[1, 0, 0], [0, 0, 0], [0, 0, 2]]
```

```
m(1,1)
```

```
Traceback (click to the left for traceback)
```

```
...
```

```
TypeError: 'list' object is not callable
```

```
m[1,1]
```

```
Traceback (click to the left for traceback)
```

```
...
```

```
TypeError: list indices must be integers, not tuple
```

```
m = matrix(m)
```

```
m[1,1]
```

```
0
```

```
m[2,2]
```

```
2
```

```
***
```

```
0= empty space
```

```
1= person
```

```
2= door
```

```
3= bomb
```

```
-1= object
```

```
***
```

```
d = c.peoplelist()
```

d

min(1,2,3)

1

max(1,2,3)

3

v = (1,2,3)

v.append(4)

Traceback (click to the left for traceback)

...

AttributeError: 'tuple' object has no attribute 'append'

len(v)

3

range(v)

Traceback (click to the left for traceback)

...

TypeError: range() integer end argument expected, got tuple.

i in range(v)

Traceback (click to the left for traceback)

...

TypeError: range() integer end argument expected, got tuple.

range(3)

[0, 1, 2]

c.peoplelist()

[[0, 0], [0, 2]]

len(c.peoplelist())

2

v = dict([(1,[1,2,3])])

v

c.peopledictionary()

```
{0: ([0, 0], {0: 2, 1: 4}), 1: ([0, 2], {0: 4, 1: 2})}
```

```
e.values()
```

```
[[[0, 0], [2, 4]], [[0, 2], [4, 2]]]
```

```
e.values()[0]
```

```
[[0, 0], [2, 4]]
```

```
r = e[0]
```

```
r
```

```
[[0, 0], [2, 4]]
```

```
a = r[1]
```

```
a
```

```
[2, 4]
```

```
min(a)
```

```
2
```

```
m = [[1,0,1],[0,0,0],[2,0,2]]
```

```
m(1,1)
```

```
Traceback (click to the left for traceback)
```

```
...
```

```
TypeError: 'list' object is not callable
```

```
m[1,1]
```

```
Traceback (click to the left for traceback)
```

```
...
```

```
TypeError: list indices must be integers, not tuple
```

```
mat(m)
```

```
Traceback (click to the left for traceback)
```

```
...
```

```
NameError: name 'mat' is not defined
```

```
m = matrix(m)
```

```
m(1,1)
```

Traceback (click to the left for traceback)

...

TypeError: 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'
object is not callable

m[1,1]

0

m

[1 0 1]
[0 0 0]
[2 0 2]

m[2,1]

0