In [69]:
```
import getopt
import sys

import numpy as np
import pandas as pd
import query

database = dict()


def read_table(table):
    file = open(table, "r")
    lines = file.readlines()
    lines = [line.replace("\n", "") for line in lines]
    lines = [line.strip() for line in lines]
    lines = [line.split(',') for line in lines]
    table_name = lines[0][0]
    data = np.array(lines[1:])
    print('table data', data)
    columns = []
    for i in xrange(data.shape[1] - 1):
        columns.append('Var' + str(i+1))
    columns.append('Prob')
    df = pd.DataFrame(data = data, columns = columns)
    database[table_name] = df
    print('dataframe',df)


def parse_query(query):
    file = open(query, "r")
    lines = file.readlines()
    sentence = lines[0]
    sentence.strip().split("||")

    print sentence.strip().split("||")
    queryline=sentence.strip().split("||")
```

In [70]:
```
# Here we are loading the tables
tables = ["t1.txt","t2.txt","t3.txt"]
for table in tables:
    read_table(table)
```

```
('table data', array([['0', '0.7'],
       ['1', '0.8'],
       ['2', '0.6']], dtype='|S3'))
('dataframe',   Var1 Prob
0    0  0.7
1    1  0.8
2    2  0.6)
('table data', array([['0', '0.7'],
       ['1', '0.3'],
       ['2', '0.5']], dtype='|S3'))
('dataframe',   Var1 Prob
0    0  0.7
1    1  0.3
2    2  0.5)
('table data', array([['0', '0', '0.8'],
       ['0', '1', '0.4'],
       ['0', '2', '0.5'],
       ['1', '2', '0.6'],
       ['2', '2', '0.9']], dtype='|S3'))
('dataframe',   Var1 Var2 Prob
0    0    0  0.8
1    0    1  0.4
2    0    2  0.5
3    1    2  0.6
4    2    2  0.9)
```

In [71]:
```
# This part is not automated, in the final solution,
#the query parser would give us a list of strings so we know which dataf
rames we need to evaluate.
queries = "query1.txt"
parse_query(queries)
```

```
['R(x1,y1),Q(x1)']
```

In [72]:
```
    file = open("t3.txt" ,"r")
    lines = file.readlines()
    lines = [line.replace("\n", "") for line in lines]
    lines = [line.strip() for line in lines]
    lines = [line.split(',') for line in lines]
    table_name = lines[0][0]
    data = np.array(lines[1:])
    print('table data', data)
```

```
('table data', array([['0', '0', '0.8'],
       ['0', '1', '0.4'],
       ['0', '2', '0.5'],
       ['1', '2', '0.6'],
       ['2', '2', '0.9']], dtype='|S3'))
```

In [73]:
```
print(database)
```

```
{'Q':   Var1 Prob
0     0  0.7
1     1  0.3
2     2  0.5, 'P':   Var1 Prob
0     0  0.7
1     1  0.8
2     2  0.6, 'R':   Var1 Var2 Prob
0     0     0  0.8
1     0     1  0.4
2     0     2  0.5
3     1     2  0.6
4     2     2  0.9}
```

In [74]:
```
print(database['Q'])
```

```
  Var1 Prob
0     0  0.7
1     1  0.3
2     2  0.5
```

In [75]:
```
print(database['P'])
```

```
  Var1 Prob
0     0  0.7
1     1  0.8
2     2  0.6
```

In [76]:
```
database['R']["Prob"]= pd.to_numeric(database['R']["Prob"])
database['Q']["Prob"]= pd.to_numeric(database['Q']["Prob"])
database['P']["Prob"]= pd.to_numeric(database['P']["Prob"])
#'Here I needed a data type conversion to numeric as in our parser, we s
et all elements in the df to string')
#'This step can be moved back into the parser, maybe?')
```

In [77]:
```
#'in the r xy table, need the negative of the possibilities'
database['R']["NegProb"]= (1-database['R']["Prob"])
```

In [78]:
```
print(database['R'])
```

```
  Var1 Var2  Prob  NegProb
0     0     0   0.8      0.2
1     0     1   0.4      0.6
2     0     2   0.5      0.5
3     1     2   0.6      0.4
4     2     2   0.9      0.1
```

In [79]:
```
#THE KEY TO THE WHOLE ALGORITHM IS THE FOLLOWING FEW LINES.
print('The key is to find the neg probability in each (x,y) pairing, gro
uping on x')
print(database['R'].groupby('Var1').prod())
df = pd.DataFrame(database['R'].groupby('Var1').prod())
database['Rprod']=database['R'].groupby('Var1').prod()

print('we are taking the R x,y table and grouping on x, essentially elim
inating y, so we can do an inner join with the dataframes in a following
 step')
```

```
The key is to find the neg probability in each (x,y) pairing, grouping
on x
      Prob  NegProb
Var1
0     0.16     0.06
1     0.60     0.40
2     0.90     0.10
we are taking the R x,y table and grouping on x, essentially eliminatin
g y, so we can do an inner join with the dataframes in a following step
```

In [80]:
```
print(database['Q'])
```

```
  Var1  Prob
0    0   0.7
1    1   0.3
2    2   0.5
```

In [81]:
```
print(database['Rprod'])
```

```
      Prob  NegProb
Var1
0     0.16     0.06
1     0.60     0.40
2     0.90     0.10
```

In [82]:
```
result = pd.merge(database['Rprod'][['NegProb']],database['Q'][['Var1',
'Prob']],how='inner', on = 'Var1')
print('here we inner join on Qx dataframe and our newly simplified R(x,
y) database')
# note the syntax here is quite strange, as the double bracket [[]]in th
e dataframe is a select specific columns
```

```
here we inner join on Qx dataframe and our newly simplified R(x,y) data
base
```

In [83]:
```
print(result)
```

```
  Var1  NegProb  Prob
0    0     0.06   0.7
1    1     0.40   0.3
2    2     0.10   0.5
```

In [84]:
```
answer = 1-((1-(result["Prob"]*(1-result["NegProb"])))).prod()
# this is the final equation to give us our answer , it is:
#
# 1. Prob, the probability that each variable is true,
# 2. and (1-Negprob) which is the case that given x, the probability tha
t x,y is true.

# this probability is essentially the prob that each (x), (x,y) paring i
s true.
# we do 1- to find the prob that each paring is false.
# we do 1- the product of all paring is false, which equals at lease one
 pariing is true.
#  Notes
# (1 - negprob) takes into account the fact that in the x,y table, one v
ariable x can have many y mappings.
# hence the group by product eariler
```

In [85]:
```
answer
```

Out[85]:
```
0.845758
```

In [88]:
```
'yay able to get the 8458 from parsing the files'
```

Out[88]:
```
'yay able to get the 8458 from parsing the files'
```

In [89]:
```
#For the many inputs in the query case, we will need to take each answer
 prob, store them in a list and then do
#1- the product of (1-Prod) again
#In essence another layer of 1- on the existing algorithm
```

In [ ]:
```
To do list:
Remaining functions which need work.
parse_query()       -- parse query . -- input . query string -- output l
ist of (x),(x,y) pairings in an array
dfmani(['P','R'])   -- dataframe manipulation  --input letters of the co
rresponding tables, --output append pair prob to answer array
Final(Answer Array) -- final solution calculation  -- input answer array
 .  -- output final solution

1. In the query parser, we should get an array with a list of x , (x,y)
parings so like  ['P','R'],['Q','R'],['T','R'] so we can run the steps o
n multiple pairs
    my thought for this is the first element of each tuple is the x pair
, and the second is the x,y pair.
    that would make things super easy for a for loop


2. We need to put the dataframe manipulation into a def function so we c
an call it for each pair.
    This function should store each answer in an answer array. for each
x, (x,y) pair
    goal implementation dfmani(['Q','R'])  would give us the 0.8458

3. We need a final output function which takes each answer in the answer
 array and do the 1- prod manipulation.''''''
```