**Constraints**

# [Constraint programming in Python](http://agiliq.com/blog/2009/03/constraint-programming-in-python/)

###### *By :*[Shabda Raaj](http://agiliq.com/blog/author/shabda/)

Imagine this, you want to solve a problem, the algorithm for which you do not know. You just know the problem.

From wikipedia,  
Constraints differ from the common primitives of imperative programming languages in that they do not specify a step or sequence of steps to execute, but rather the properties of a solution to be found.

Assume that there exists an alternate world where you only need to specify the problem, the computer will find out an algorithm to find it, even better if could you write it in Python.

Stop assuming it hapens every day, and this is the magic of constraint programming.

Example to solve a + b = 5, a\*b=6, don't explain algebra, just tell the relations between variables. Here is the complete program.

from constraint import \*  
problem = Problem()  
problem.addVariable('a', range(5))  
problem.addVariable('b', range(5))  
problem.addConstraint(lambda a, b: a + b == 5)  
problem.addConstraint(lambda a, b: a \* b == 6)  
problem.getSolutions()

Don't worry if this doesn't make sense yet, we will explain it in a lot of detail. Also we cheated a little, by assuming that the solutions are positive integers.

## Files required

You need a constraint library, which allows ypu to do constraint programming. [**python-constraints**](http://labix.org/python-constraint) from [**Gustavo Niemeyer**](http://labix.org/) is an excellent library to do constraint programming, which we will use here. Download and install python-constraint from [**here**](http://labix.org/python-constraint).

After you setup, you should be able to do import constraint on a python shell

## On to our program

Ok lets see the same program with cpomments.

from constraint import \* #imports  
problem = Problem()  #Create a blank problem where we can add solutions.  
problem.addVariable('a', range(5+1)) #Add a variable named a, and specify its domain. Since a+b=5, a is a positive integer less than 5  
problem.addVariable('b', range(5+1)) #Same thing with b  
problem.addConstraint(lambda a, b: a + b == 5) # Tell the computer that a+b=5  
problem.addConstraint(lambda a, b: a \* b == 6) #tell the computer that a\*b=6  
print problem.getSolutions() #We are done, get the solutions.

Not much to explain after you see it with comments. We created two variables, and relation between them and got the answer.

## Ok something better

Ok so the last example was cute, but can constraint programming can solve any harder problems? Lets try something harder, like umm, magic square. (How, original!)

So lets think, how can we represent our problem.

1. We have a n\*n board. Start with 3, So our board is [(0,0), (0,1) ...., (2,1), (2,2)
2. Rows have a sum of sum(1 .... n\*n+1)/size
3. Cols have a sum of sum(1 .... n\*n+1)/size
4. Diagonals have a sum of sum(1 .... n\*n+1)/size
5. Thats all,

So without further ado, lets see the code.

from constraint import \*  
  
def solve\_magic\_square(size = 3):  
            "Get the magic square solution for any numbered square."  
    magic = Problem()#create a blank problem  
    rows = range(size)#Rows indices eg[0, 1, 2]  
    cols = range(size)#cols indices eg [0, 1, 2]  
    board\_line\_sum = sum(range(1, size\*size+1))/size # What does each row, col and diag sum up to? Eg 15, for size 3  
    board = [(row, col) for row in rows for col in cols] # Cartesan of rows and col, eg [(0, 0), (0, 1), (1, 0), (1, 1)] for size = 2  
    row\_set = [zip([el]\*len(cols), cols) for el in rows]#A list of all the rows, eg [[(0, 0), (0, 1)], [(1, 0), (1, 1)]]  
    col\_set = [zip(rows, [el]\*len(rows)) for el in cols]#A list of all the columns, eg [[(0, 0), (1, 0)], [(0, 1), (1, 1)]]  
    diag1 = zip(rows, cols)#One of the diagonals, eg [(0,0), (0,1)]  
    diag2 = zip(rows, cols[::-1])#Other diagonal, eg [(0,1), (1,0)]  
  
    magic.addVariables(board, range(1, size\*size+1))#add Each block of square as a variable. There range is between [1..n\*n+1]  
    magic.addConstraint(ExactSumConstraint(board\_line\_sum), diag1)#Add diagonals as a constraint, they must sum to board\_line\_sum  
    magic.addConstraint(ExactSumConstraint(board\_line\_sum), diag2)#Add other diagonal as constraint.  
    for row in row\_set:  
        magic.addConstraint(ExactSumConstraint(board\_line\_sum), row)#Add each row as constraint, they must sum to board\_line\_sum  
    for col in col\_set:  
        magic.addConstraint(ExactSumConstraint(board\_line\_sum), col)#Similarly add each column as constraint.  
    magic.addConstraint(AllDifferentConstraint(), board)#Every block has a different number.  
    return magic.getSolution()Retutn the solution.  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    print solve\_magic\_square()

Not much to explain again after you see the commented code. We created a board, added constraints and got solution.

[**Labix**](http://labix.org/python-constraint) has a different implementation if how to solve the magic square.

## What more can I do?

Constraint programming has a wide application, wherever you can specify the problem as relation between variables you can use constraint programming.

Link bij bovenstaande tekst: <http://agiliq.com/blog/2009/03/constraint-programming-in-python/>

Python constraint module om te downloaden en gebruiken in python, met wat voorbeelden om een probleem op te lossen (bijv. maqic square problem): <https://labix.org/python-constraint>

Wiki over constraint programmeren: <https://en.wikipedia.org/wiki/Constraint_programming>

Algoritme van de sheet, die mogelijk aansluit bij ons problem: <https://en.wikipedia.org/wiki/Constraint_satisfaction_problem>