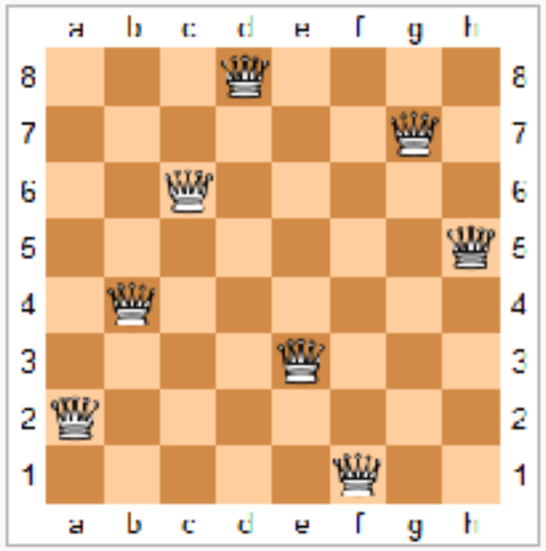
15-112 Fundamentals of Programming

Week 5 - Lecture 3: More Advanced Recursion

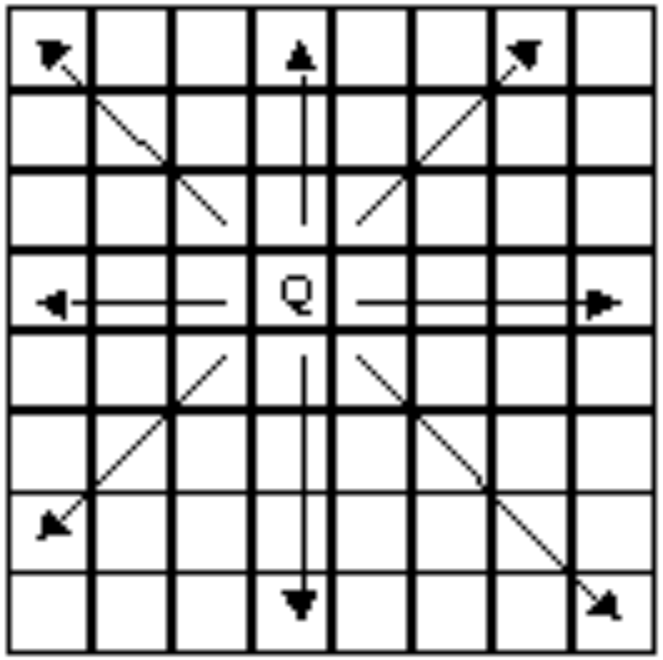


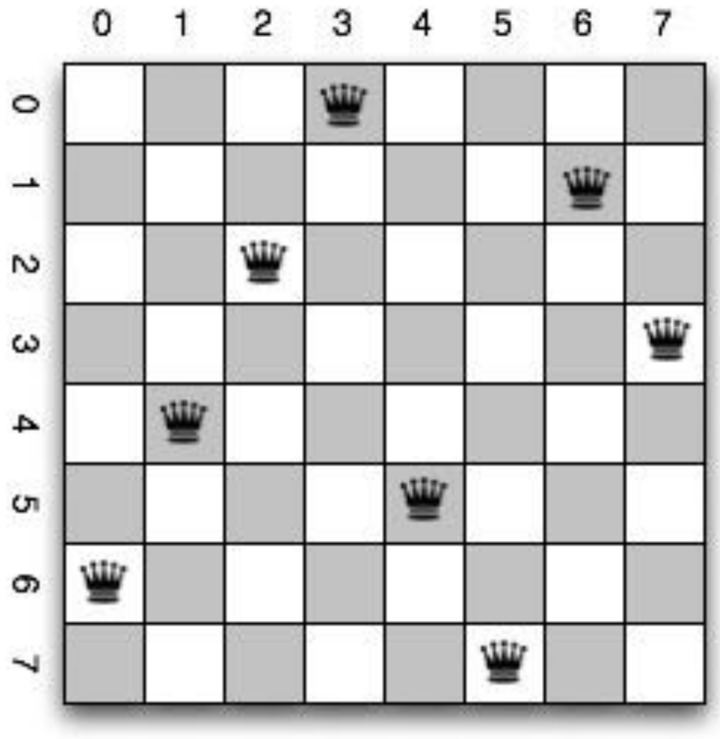
One solution to the eight queens puzzle

June 23, 2017

nQueens Problem

Place n queens on a n by n board so that no queen is attacking another queen.



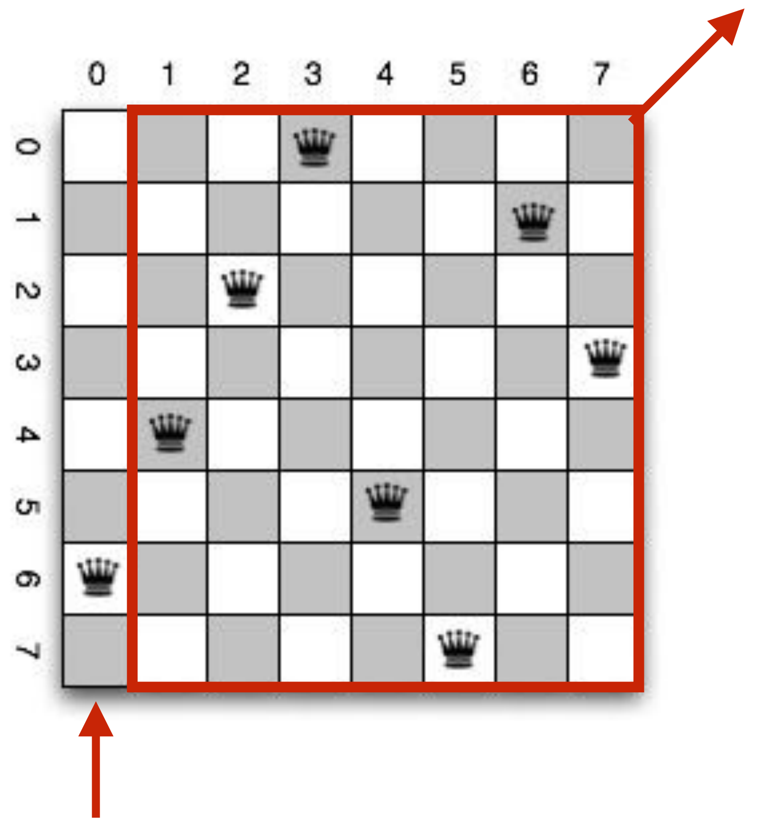


def solve(n): —> [6, 4, 2, 0, 5, 7, 1, 3]

list of rows

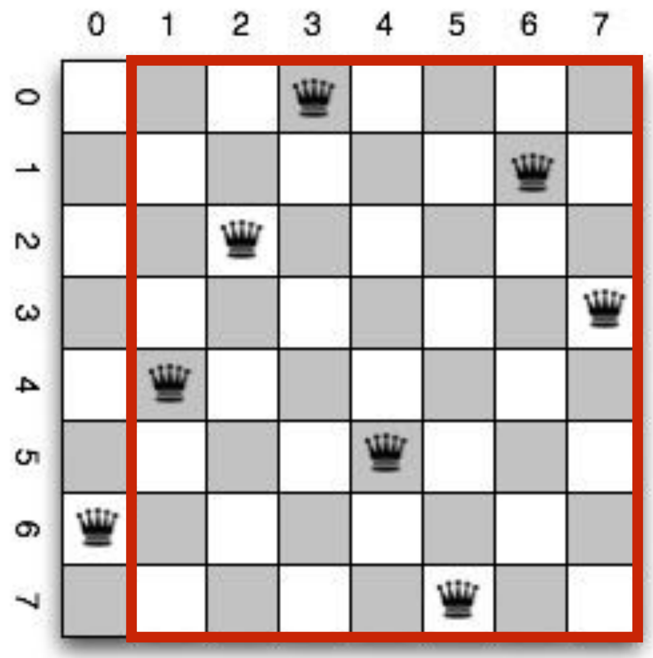
nQueens Problem

Place n queens on a n by n board so that no queen is attacking another queen.



one queen has to be on first column

nQueens Problem



<u>First attempt:</u>

- try rows 0 to 7 for first queen
- for each try, recursively solve the red part

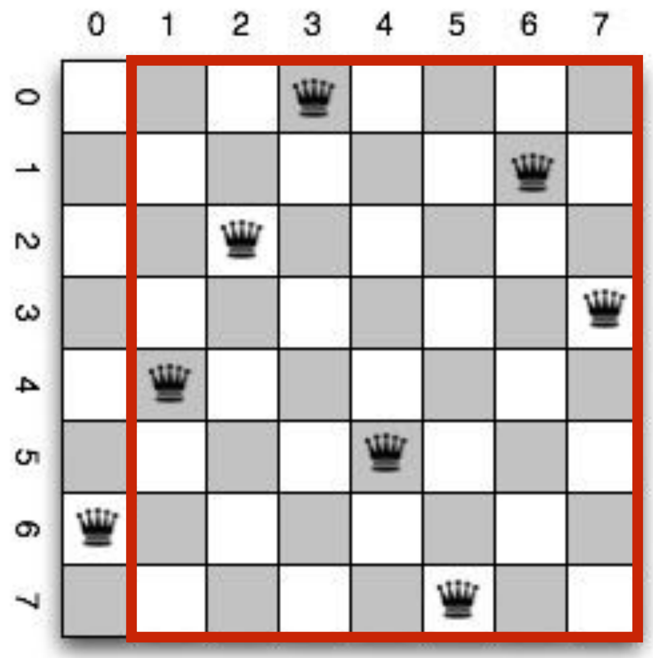
<u>Problem:</u>

Can't solve red part without taking into account first queen
First queen puts constraints on the solution to the red part

Need to be able to solve nQueens with added constraints.
Need to generalize our function:

```
def solve(n, m, constraints):
```

nQueens Problem



def solve(n, m, constraints):

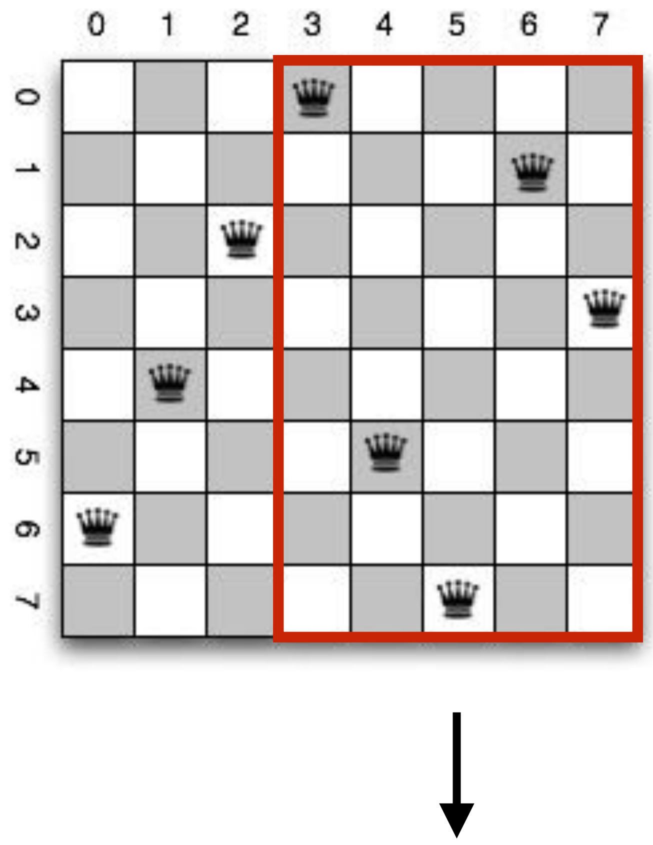
n = number or rows

m = number or columns

constraints (in what form?)
list of rows

For the red part, we have the constraint [6]

nQueens Problem



def solve(n, m, constraints):

n = number or rows

m = number or columns

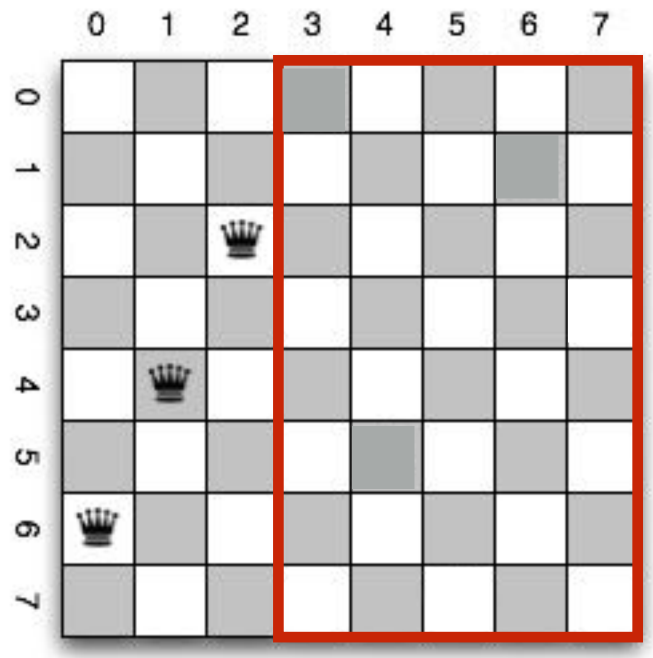
constraints (in what form?)
list of rows

For the red part, we have the constraint [6,4,2]

The constraint tells us which cells are <u>unusable</u> for the red part.

To solve original nQueens problem, call: solve(n, n, [])

nQueens Problem



[?,?,?,?,?]

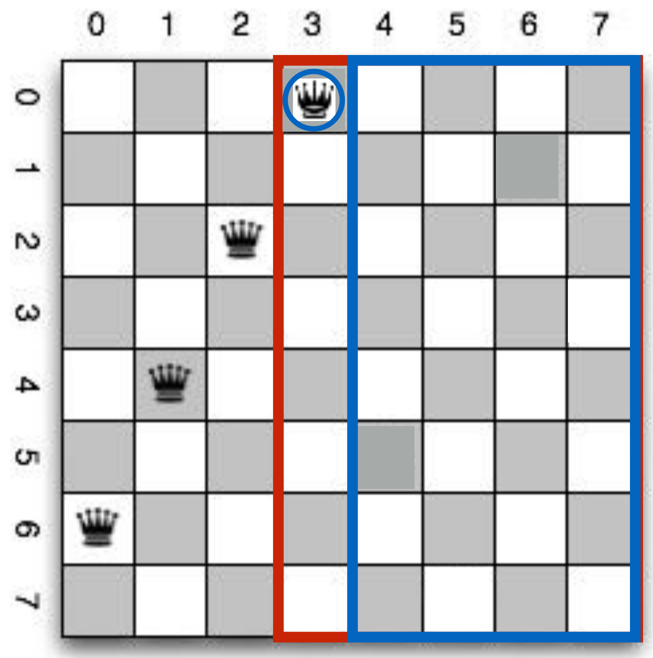
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

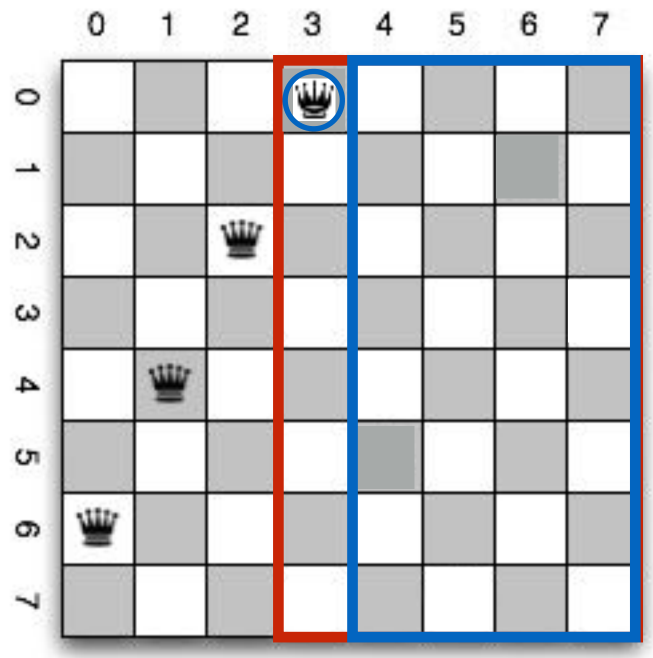
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

[5,7,1,3]

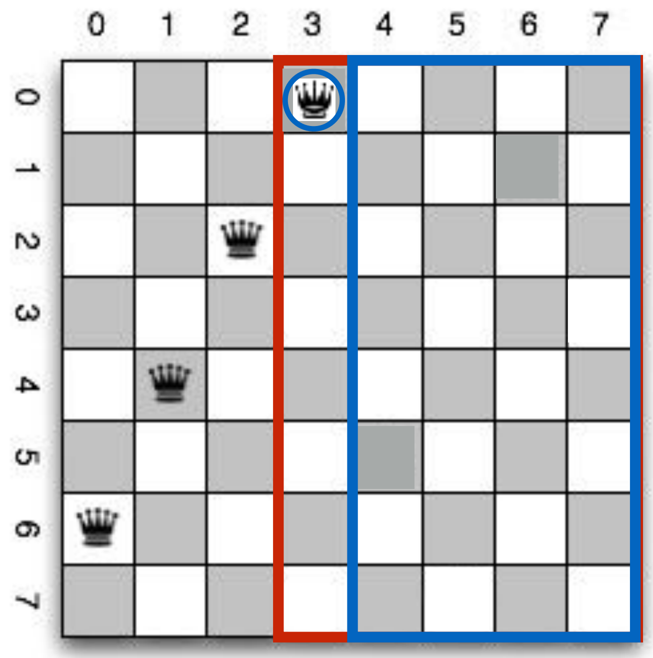
n = 8

m = 5

constraints = [6,4,2]

```
def solve(n, m, constraints):
```

nQueens Problem



```
def solve(n, m, constraints):
```

[0,?,?,?,?]

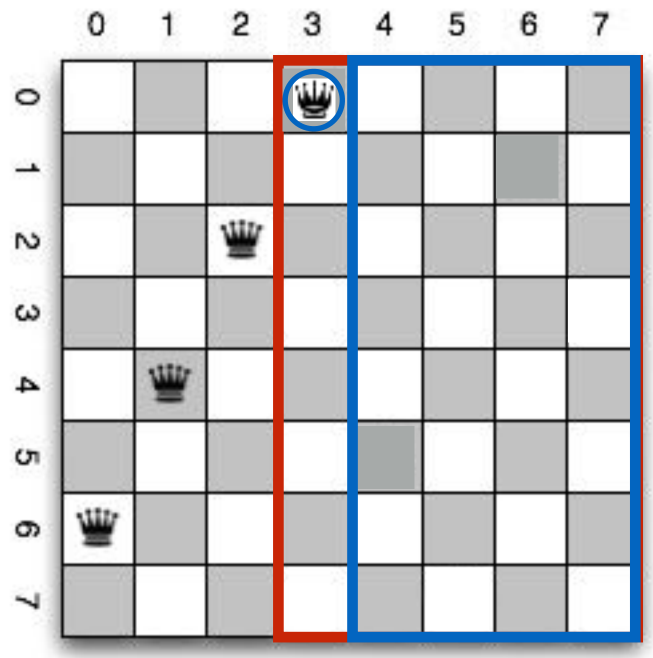
[5,7,1,3] —> [0,5,7,1,3]

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

Suppose no solution

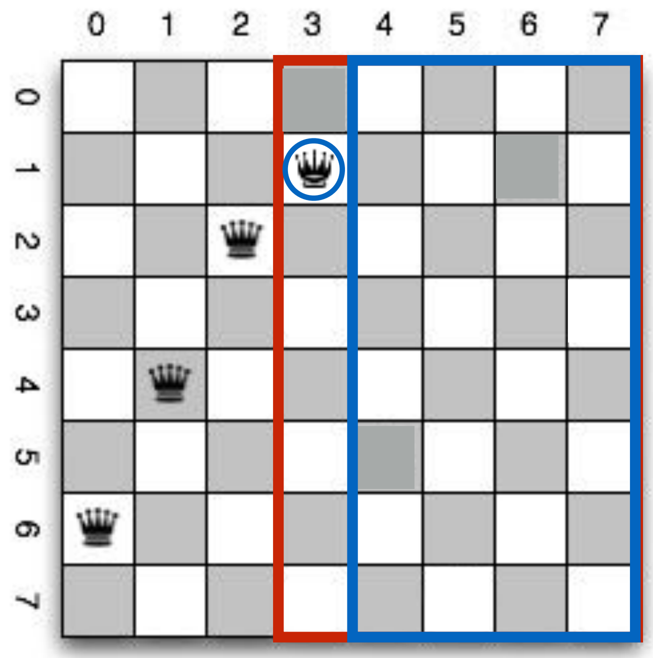
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

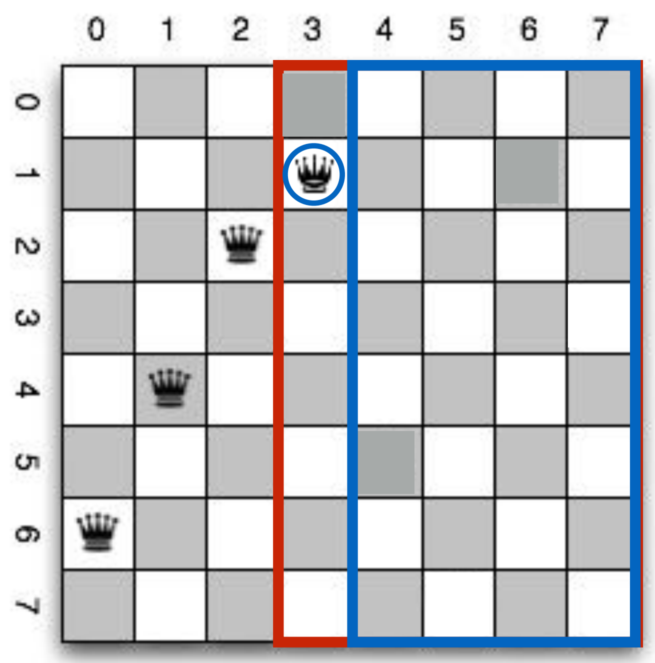
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

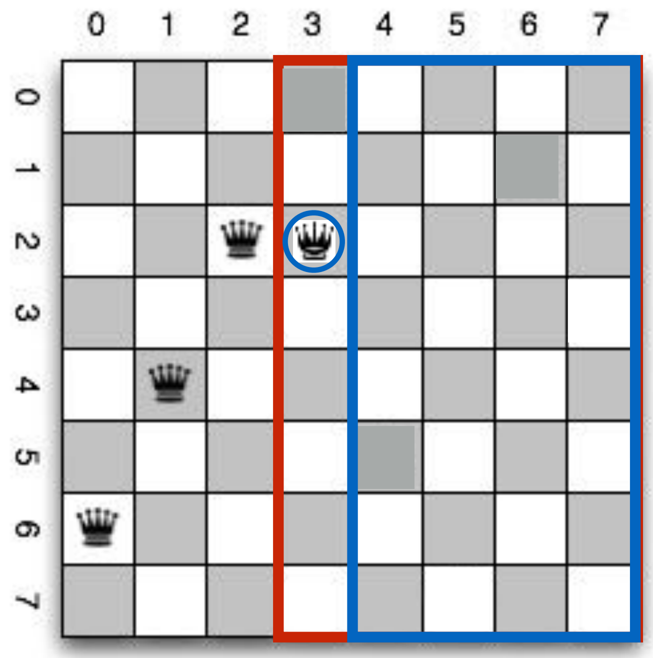
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

def solve(n, m, constraints):

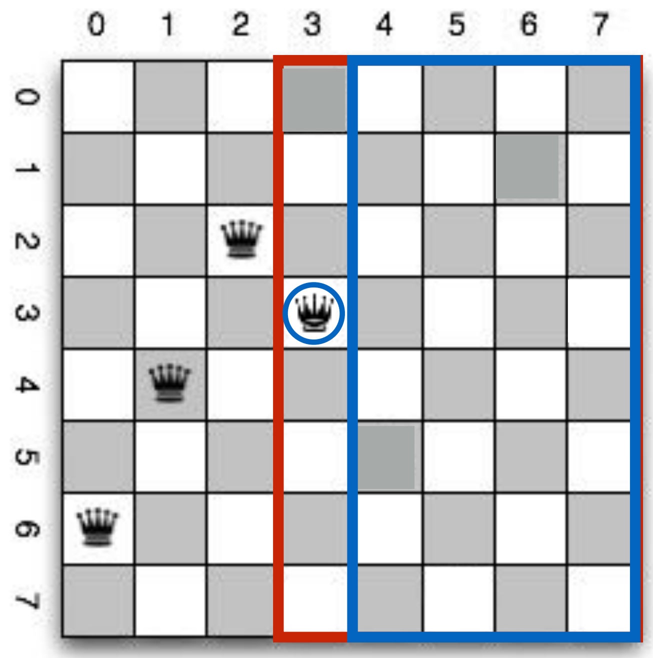
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

```
def solve(n, m, constraints):
```

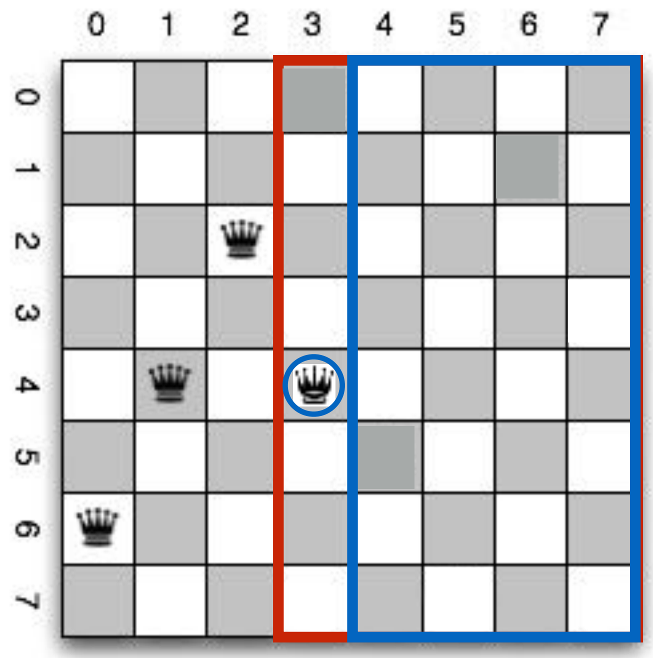
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

```
def solve(n, m, constraints):
```

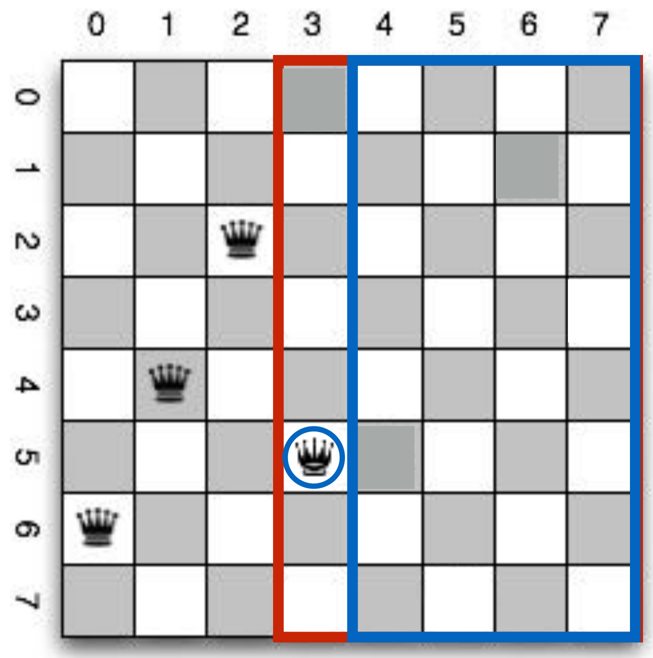
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

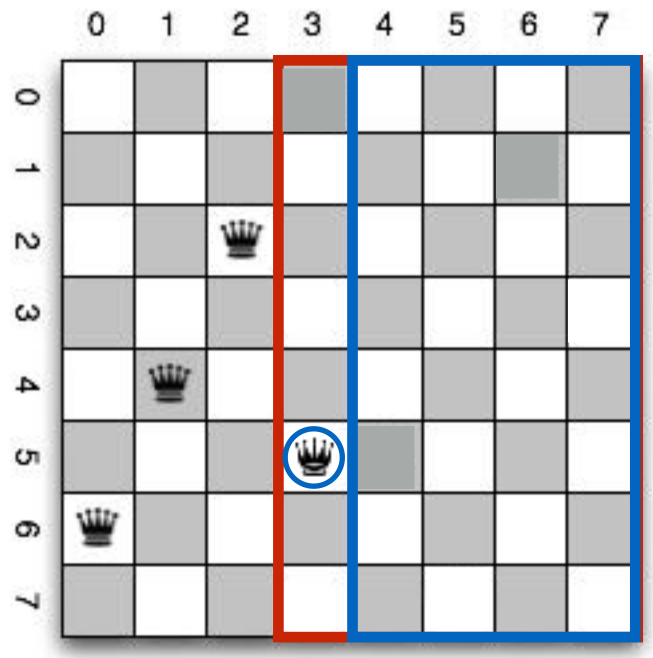
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

no solution

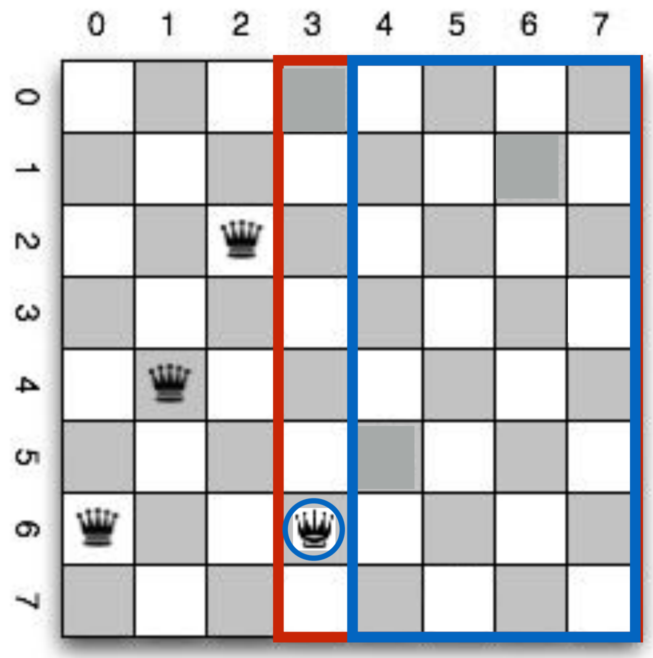
n = 8

m = 5

constraints = [6,4,2]

```
def solve(n, m, constraints):
```

nQueens Problem



[0,?,?,?,?]

```
def solve(n, m, constraints):
```

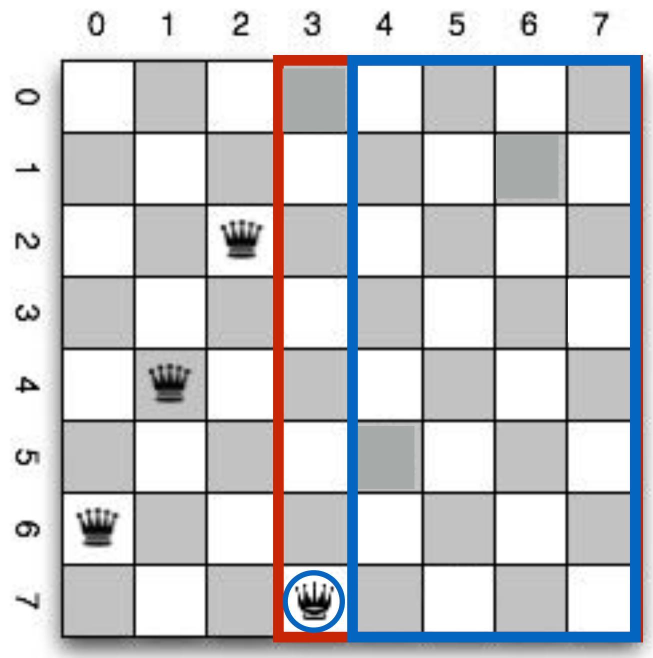
NOT LEGAL

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

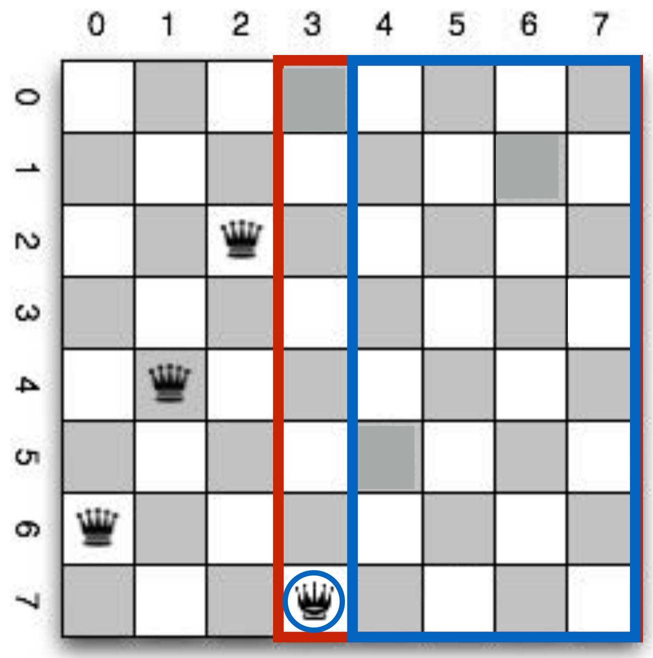
```
def solve(n, m, constraints):
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



[0,?,?,?,?]

no solution

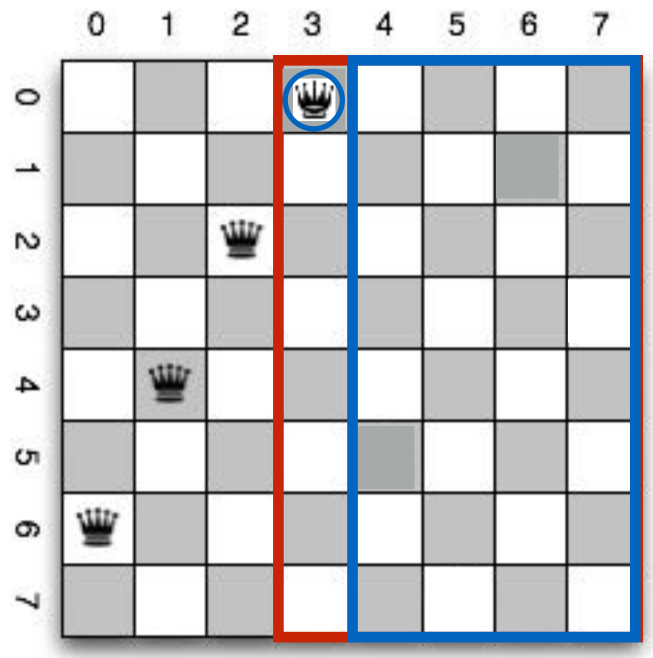
n = 8

m = 5

constraints = [6,4,2]

```
def solve(n, m, constraints):
```

nQueens Problem



[0,?,?,?,?]

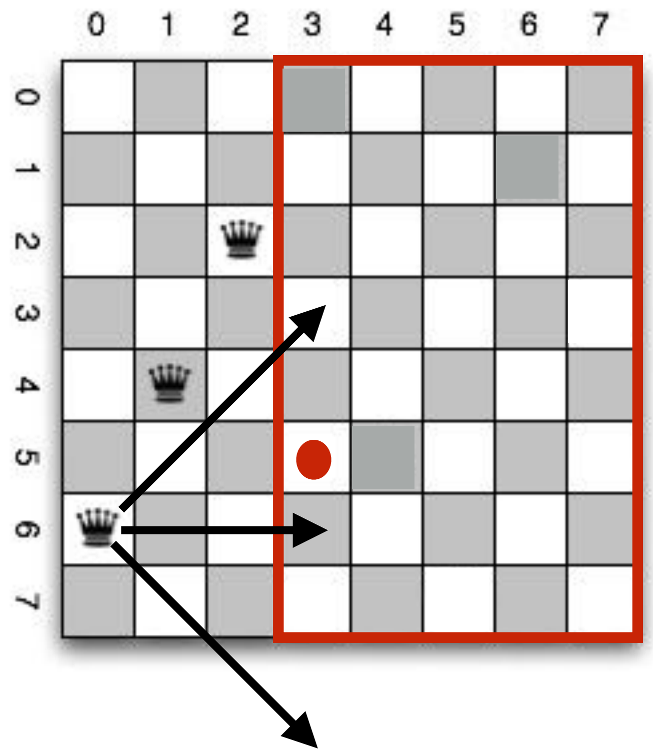
```
def solve(n, m, constraints):
    if(m == 0):
        return []
    for row in range(n):
        if (isLegal(row, constraints)):
            newConstraints = constraints + [row]
            result = solve(n, m-1, newConstraints)
            if (result != False):
                return [row] + result
    return False
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



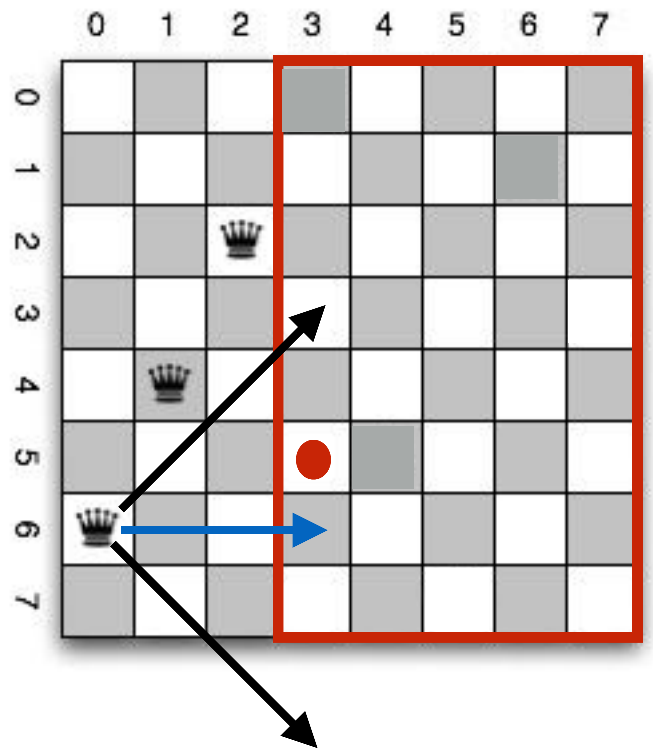
```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



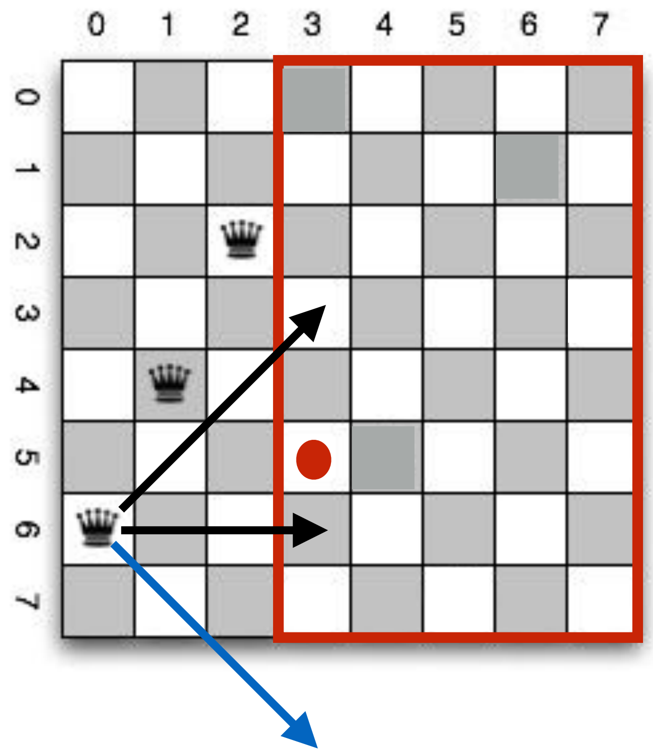
```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
             (row == crow + shift) or
             (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



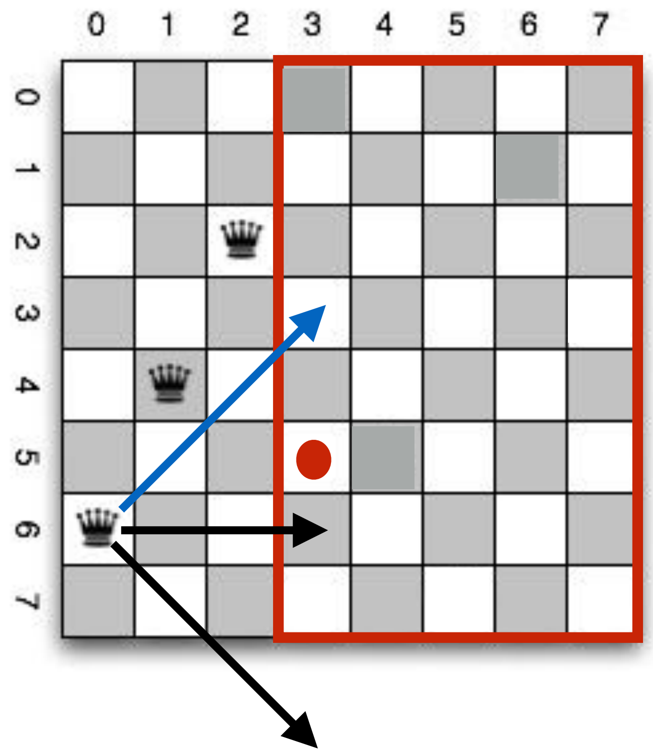
```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
          return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

nQueens Problem



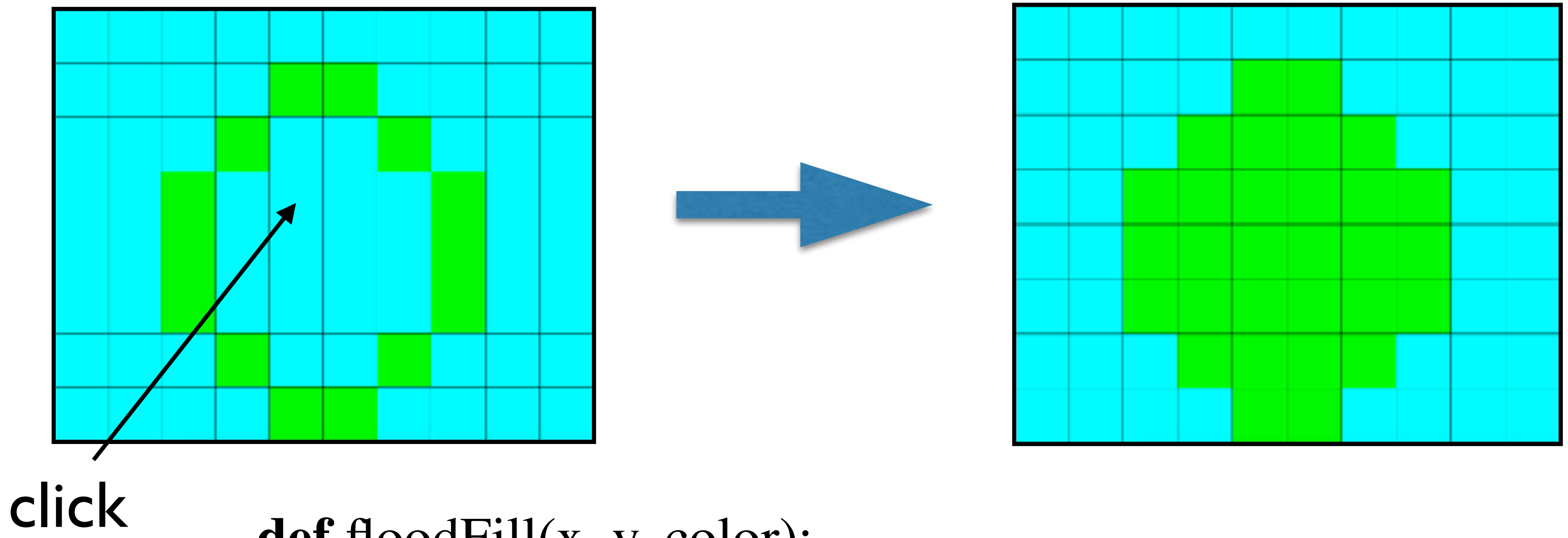
```
def isLegal(row, constraints):
    for ccol in range(len(constraints)):
        crow = constraints[ccol]
        shift = len(constraints) - ccol
        if ((row == crow) or
            (row == crow + shift) or
            (row == crow - shift)):
            return False
    return True
```

n = 8

m = 5

constraints = [6,4,2]

Flood fill



```
def floodFill(x, y, color):
    if ((not inImage(x,y)) or (getColor(img, x, y) == color)):
        return
    img.put(color, to=(x, y))
    floodFill(x-1, y, color)    U
    floodFill(x+1, y, color)    D
    floodFill(x, y-1, color)    L
    floodFill(x, y+1, color)    R
```

Term Project

Some general rules

- SOLO: must do your own independent project.

- Can use any external materials
e.g. code, designs, images, text, sounds, …

These must be very clearly cited!

This includes citing yourself!

You'll be graded on your original contributions.

Some general rules

- Must use Python

- You will be assigned a "Mentor CA":

  Provides most of the support and guidance.

  Will grade your TP.

The overall process

Sun	Mon	Tue	Wed	Thu	Fri	Sat
19	26 **Meet**	21	28 **Meet**	23	30 **Meet**	1 **Meet**
2 **DEADLINE**	3	4				

Meeting 1

- **Project proposal**

  > Define the problem

  > Description on how you intend to solve it

  > List all modules/technologies you plan to use

- **Competitive analysis**

  > Find existing products similar to what you propose

  > List features you plan to include

  > List features you plan to change

Meeting 1

- **Storyboard**

  > Hand-drawn pictures showing how app will run from the perspective of the user.

- **Technology demonstrations**

  > Demonstration of competency

- **Code artifacts**

  > If you have any

- **Timesheet**

  > timesheet.txt

  > Keep track of the time you spend on the project.

Meeting 2

- **Progress**

  > A good amount of code

  > Basic features implemented and functional

- **Timesheet**

Meeting 3

- **Working demo**

  > A working B-level final project

  > May miss some features, contain some bugs, etc…

- **Timesheet**

Submission

- **Project source files and support files**

    > Python files + others (.jpg, midi, …)

- **Readme file (readme.txt)**

    > What is your project?

    > How to install and run it

    > How to download/install 3rd party libraries

Submission

- **Design documents**

  > Explain the problem, and how you solve it.

  > Why you chose the particular functions, data structures, algorithms that you used.

  > Discuss the user interface choices.

- **Project video**

  > 1-3 minutes long

  > Show the most important features, highlights

- **Timesheet**

Submission

Submission will be made to Autolab.

Single zip file.

Cannot exceed 10MB.

- Submit complete version to your mentor.
- You can run complete version in grading session.

Grading

Important Factors

- Complexity and sophistication
- Robust operational program
- User interface
- Effort
- Design
- Style
- Presentation

A+
A
A-
B+
B
B-
C+
C
C-
D+
D
D-
R

Grading

- Grading meeting:
  - 2 TAs
  - Demo your term project
  - Be ready to walk the TAs through any part of the code

HAVE FUN!