## 15-112 View

<table>
<thead>
<tr>
<th></th>
<th>Recursion</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elegance</strong></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td><strong>Debugability</strong></td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>
def fib(n):
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    return result

print(fib(6))

How many times is fib(2) computed? 5
fibResults = dict()

def fib(n):
    if (n in fibResults):
        return fibResults[n]
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    fibResults[n] = result
    return result
def rangeSum(lo, hi):
    if (lo > hi):
        return 0
    else:
        return lo + rangeSum(lo+1, hi)

print(rangeSum(1, 1234))
# RuntimeError: maximum recursion depth exceeded

print(callWithLargeStack(rangeSum(1, 123456)))
# Works
More Examples
Given a list, return a list of all the subsets of the list.

\[
[1,2,3] \rightarrow \text{[[]], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]}\]
Power set

Given a list, return a list of all the subsets of the list.

\[ 1,2,3 \] -> \[ [], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3] \]
Power set

Given a list, return a list of all the subsets of the list.

\[1,2,3\] -> \[
[[]], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]\]

All subsets = All subsets that do not contain 1 +
Power set

Given a list, return a list of all the subsets of the list.


All subsets = All subsets that do not contain 1 +
Power set

Given a list, return a list of all the subsets of the list.

\[1,2,3\] \rightarrow [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

All subsets = All subsets that do not contain 1 +

All subsets that contain 1
Power set

Given a list, return a list of all the subsets of the list.

\[1,2,3\] \rightarrow \text{[[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]}

\text{[1]} + \text{subset that doesn’t contain a 1}

\text{All subsets} = \text{All subsets that do not contain 1} + \text{All subsets that contain 1}
Power set

Given a list, return a list of all the subsets of the list.

\[1,2,3\] \rightarrow \emptyset, [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]

```python
def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = []
        for subset in powerset(a[1:]):
            allSubsets += [subset]
            allSubsets += [[a[0]] + subset]
        return allSubsets
```
Power set

Given a list, return a list of all the subsets of the list.

\[1,2,3\] -> \[
\[
\], \[1\], \[2\], \[3\], \[1,2\], \[2,3\], \[1,3\], \[1,2,3\]\]

def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = [ ]
        for subset in powerset(a[1:]):
            allSubsets += [subset]
        allSubsets += [[a[0]] + subset]
        return allSubsets
**Power set**

Given a list, return a list of all the subsets of the list.

\[1,2,3\] -> \[
[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]\]

```python
def powerset(a):
    if (len(a) == 0):
        return [[]]
    else:
        allSubsets = []
        for subset in powerset(a[1:]):
            allSubsets += [subset]
            allSubsets += [[a[0]] + subset]
        return allSubsets
```
Permutations

Given a list, return all permutations of the list.

\[1,2,3\] \rightarrow \text{[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]}
Given a list, return all permutations of the list.

\[[1,2,3] \rightarrow \[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1] \]

\[[1,2,3], [2,1,3], [2,3,1]]
Permutations

Given a list, return all permutations of the list.

\[1,2,3\] -> \[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]\]
\[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]\]
Permutations of \([2,3,4]\)
Permutations

Given a list, return all permutations of the list.

\[
[1,2,3] \rightarrow [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]
\]

def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = []
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
Given a list, return all permutations of the list.

\[
[1,2,3] \rightarrow [[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]]
\]

def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = []
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
Given a list, return all permutations of the list.

\[1,2,3\] -> \[[1,2,3], [2,1,3], [2,3,1], [1,3,2], [3,1,2], [3,2,1]\]

def permutations(a):
    if (len(a) == 0):
        return [[]]
    else:
        allPerms = [
        for subPermutation in permutations(a[1:]):
            for i in range(len(subPermutation)+1):
                allPerms += [subPermutation[:i] + [a[0]] + subPermutation[i:]]
        return allPerms
### Print files in a directory

<table>
<thead>
<tr>
<th>Name</th>
<th>Date Modified</th>
<th>Size</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder1</td>
<td>Today, 10:11 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>Folder2</td>
<td>Today, 10:12 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>helloworld.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>todo</td>
<td>Oct 3, 2014, 1:04 PM</td>
<td>1 KB</td>
<td>rich text</td>
</tr>
<tr>
<td>Name</td>
<td>Date Modified</td>
<td>Size</td>
<td>Kind</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------</td>
<td>-----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Folder1</td>
<td>Today, 10:11 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>foo.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>fooo.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>SubFolder1</td>
<td>Today, 10:11 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>fooooo.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>SubFolder2</td>
<td>Today, 10:12 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>foooooo.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>fooooooo.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>SubSubFolder1</td>
<td>Today, 10:13 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>somePic</td>
<td>Today, 9:32 PM</td>
<td>56 KB</td>
<td>PNG picture</td>
</tr>
<tr>
<td>Folder2</td>
<td>Today, 10:12 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>haha</td>
<td>Oct 3, 2014, 1:04 PM</td>
<td>1 KB</td>
<td>rich text</td>
</tr>
<tr>
<td>helloworld.py</td>
<td>Oct 7, 2014, 1:10 PM</td>
<td>812 bytes</td>
<td>Python</td>
</tr>
<tr>
<td>todo</td>
<td>Oct 3, 2014, 1:04 PM</td>
<td>1 KB</td>
<td>rich text</td>
</tr>
</tbody>
</table>
import os
def printFiles(path):
    if (os.path.isdir(path) == False):
        # base case: not a folder, but a file, so print its path
        print(path)
    else:
        # recursive case: it's a folder
        for filename in os.listdir(path):
            printFiles(path + "/" + filename)
Look at all these countries I used to own
nQueens Problem

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

```python
def solve(n):
    return [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.

$n$ rows and $n-1$ columns

one queen has to be on first column
nQueens Problem

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

Problem:
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:
```python
def solve(n, m, constraints):
```
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]
def solve(n, m, constraints):
    n = number of rows
    m = number of 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 unusable 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

def solve(n, m, constraints):

[0,?,?,?,?,?]

n = 8
m = 5
constraints = [6,4,2]
def solve(n, m, constraints):

[0,?,?,?,?,?]
[5,7,1,3]

n = 8
m = 5
constraints = [6,4,2]
def solve(n, m, constraints):

n = 8
m = 5
constraints = [6, 4, 2]
n Queens Problem

Suppose no solution

n = 8
m = 5
constraints = [6, 4, 2]
nQueens Problem

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

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

```
n = 8
m = 5
constraints = [6,4,2]
```
nQueens Problem

n = 8
m = 5
constraints = [6, 4, 2]

def solve(n, m, constraints):
    NOT LEGAL

[0,?,?,?,?,?]
nQueens Problem

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

NOT LEGAL
```

\[ [0, ?, ?, ?, ?] \]

\[ n = 8 \]

\[ m = 5 \]

\[ constraints = \{6, 4, 2\} \]
def solve(n, m, constraints):

NOT LEGAL

n = 8
m = 5
constraints = [6, 4, 2]
nQueens Problem

def solve(n, m, constraints):

NOT LEGAL

[0,?,?,?,?]

n = 8
m = 5
constraints = [6,4,2]
nQueens Problem

def solve(n, m, constraints):

[0, ?, ?, ?, ?, ?]

n = 8
m = 5
constraints = [6, 4, 2]
def solve(n, m, constraints):

n = 8
m = 5
constraints = [6, 4, 2]
nQueens Problem

def solve(n, m, constraints):

NOT LEGAL

[0,?,?,?,?,?]

n = 8
m = 5
constraints = [6,4,2]
nQueens Problem

def solve(n, m, constraints):

n = 8
m = 5
constraints = [6,4,2]
nQueens Problem

def solve(n, m, constraints):

[0,?,?,?,?]

no solution

n = 8
m = 5
constraints = [6,4,2]
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]
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]
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]
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]
Solving a maze puzzle
Solving a maze puzzle
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    —> True or False

Main Idea:
if isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)),
then for some neighbor (rowN, colN) of (rowStart, colStart),
isSolvable(maze, (rowN, colN), (rowEnd, colEnd))
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True

    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and isSolvable(maze, newCell, (rowEnd, colEnd))):
            return True

    return False

Where is the bug?
Solving a maze puzzle

visited = set()

def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True
    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and isSolvable(maze, newCell, (rowEnd, colEnd))):
            return True
    if ((rowStart, colStart) in visited):
        return False
    visited.add((rowStart, colStart))
    return False
def isSolvable(maze, (rowStart, colStart), (rowEnd, colEnd)):
    if ((rowStart, colStart) == (rowEnd, colEnd)):
        return True
    for dir in [(-1,0), (1,0), (0,1), (0,-1)]:
        newCell = (rowStart, colStart) + dir
        if (isLegal(maze, newCell) and isSolvable(maze, newCell, (rowEnd, colEnd))):
            return True
    if ((rowStart, colStart) in visited):
        return False
    visited.add((rowStart, colStart))
    solution.add((rowStart, colStart))
    solution.remove((rowStart, colStart))
    return False
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
A change rule:

\[
\text{length} \rightarrow \text{length}/3
\]
```
def kochSide(length, n):
    if (n == 1):
        turtle.forward(length)
    else:
        kochSide(length/3, n-1)
        turtle.left(60)
        kochSide(length/3, n-1)
        turtle.right(120)
        kochSide(length/3, n-1)
```
def kochSnowflake(length, n):
    # just call kochSide 3 times
    for step in range(3):
        kochSide(length, n)
    turtle.right(120)
def drawST(x, y, size, level):
    # (x, y) is the bottom-left corner of the triangle
    if (level == 0):
        canvas.create_polygon((x, y),
                              (x+size, y),
                              (x+size/2, y-size*(3**0.5)/2),
                              fill="black")
    else:
        drawST(x, y, size/2, level-1)
        drawST(x+size/2, y, size/2, level-1)
        drawST(x+size/4, y-size*(3**0.5)/4, size/2, level-1)