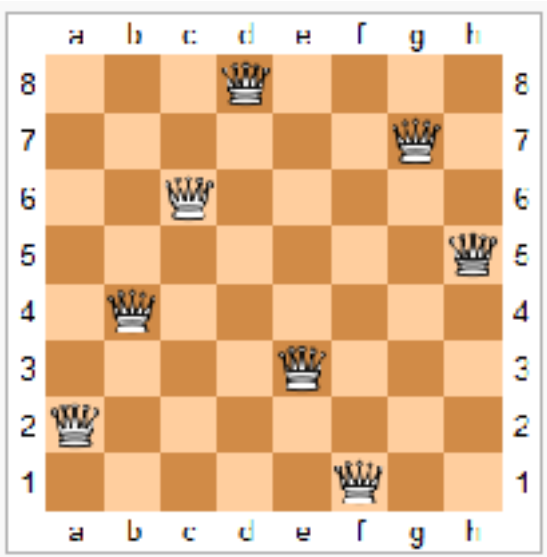
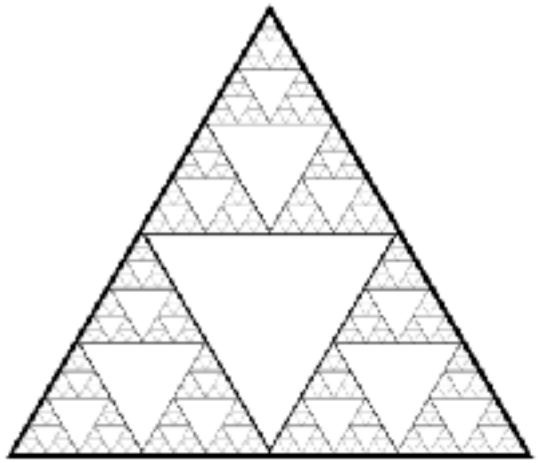
15-112
Fundamentals of Programming

Week 5 - Lecture 3:
More Advanced Recursion



One solution to the eight queens puzzle



June 22, 2017

Memoization

```
def fib(n):
    if (n < 2):
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
    return result

print(fib(4))
```

How many times is fib(2) computed? **2**

Unwinding the code

fib(4)

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

↓

fib(2) + fib(1)

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

↓

fib(2) + fib(1)

↓

fib(1) + fib(0)

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

↓

fib(2) + fib(1)

↓

1 + 1

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

↓

2 + fib(1)

Unwinding the code

fib(4)

↓

fib(3) + fib(2)

↓

2 + 1

Unwinding the code

fib(4)

↓

3 + fib(2)

Unwinding the code

fib(4)

↓

3 + fib(2)

↓

fib(1) + fib(0)

Unwinding the code

fib(4)

↓

3 + fib(2)

↓

1 + fib(0)

Unwinding the code

fib(4)

↓

3 + fib(2)

↓

1 + 1

Unwinding the code

fib(4)

↓

3 + 2

Unwinding the code

5

Memoization

```
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
```

Expanding the stack size and recursion limit

```
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

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]]

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.

[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.

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

[1] + subset that doesn't contain a 1

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] -> [[], [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]]

```
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
```

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]]

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]

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

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
```

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]]

```
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
```

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]]

```
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

Name	Date Modified	Size	Kind
Folder1	Today, 10:11 PM	--	Folder
Folder2	Today, 10:12 PM	--	Folder
helloworld.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
todo	Oct 3, 2014, 1:04 PM	1 KB	rich tex

Print files in a directory

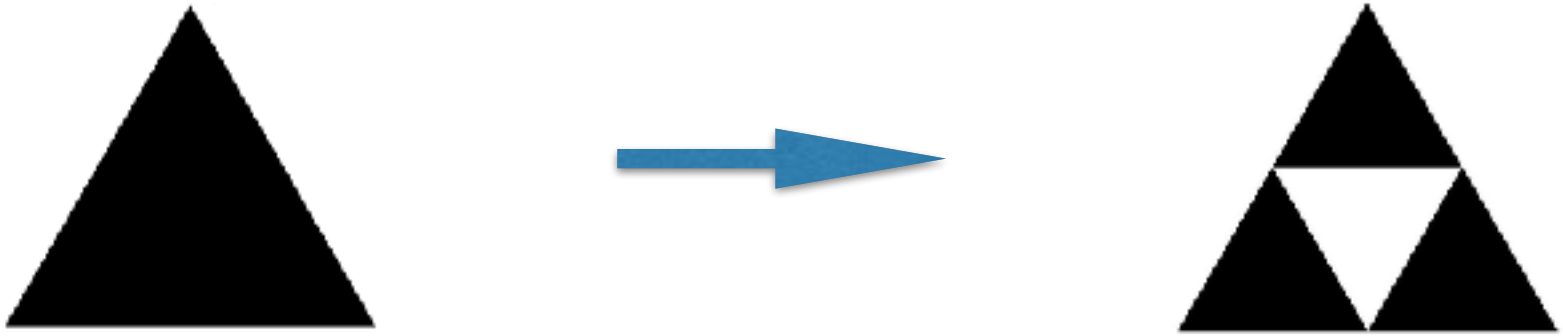
Name	Date Modified	Size	Kind
Folder1	Today, 10:11 PM	--	Folder
foo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
fooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
SubFolder1	Today, 10:11 PM	--	Folder
foooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
SubFolder2	Today, 10:12 PM	--	Folder
fooooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
foooooo.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
SubSubFolder1	Today, 10:13 PM	--	Folder
somePic	Today, 9:32 PM	56 KB	PNG ir
Folder2	Today, 10:12 PM	--	Folder
haha	Oct 3, 2014, 1:04 PM	1 KB	rich tex
helloworld.py	Oct 7, 2014, 1:10 PM	812 bytes	Python
todo	Oct 3, 2014, 1:04 PM	1 KB	rich tex

Print files in a directory

```
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)
```

Fractals: Sierpinski Triangle

level 0



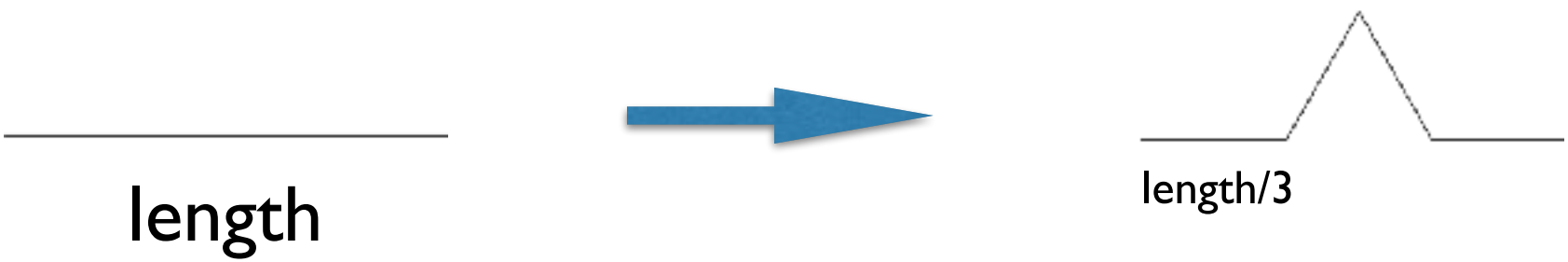
level 1

level 2

```
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)
```

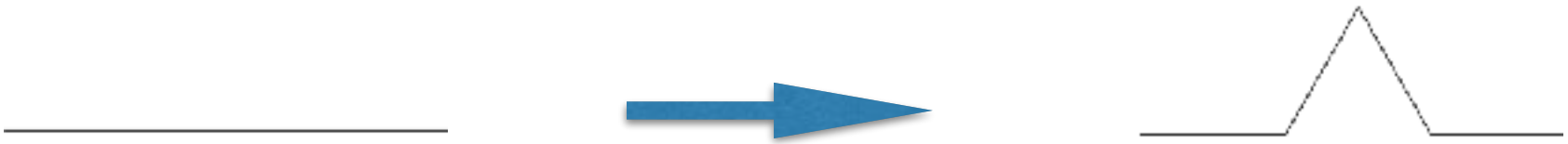
Fractals

A change rule:



Fractals: kochSnowflake

n = 1



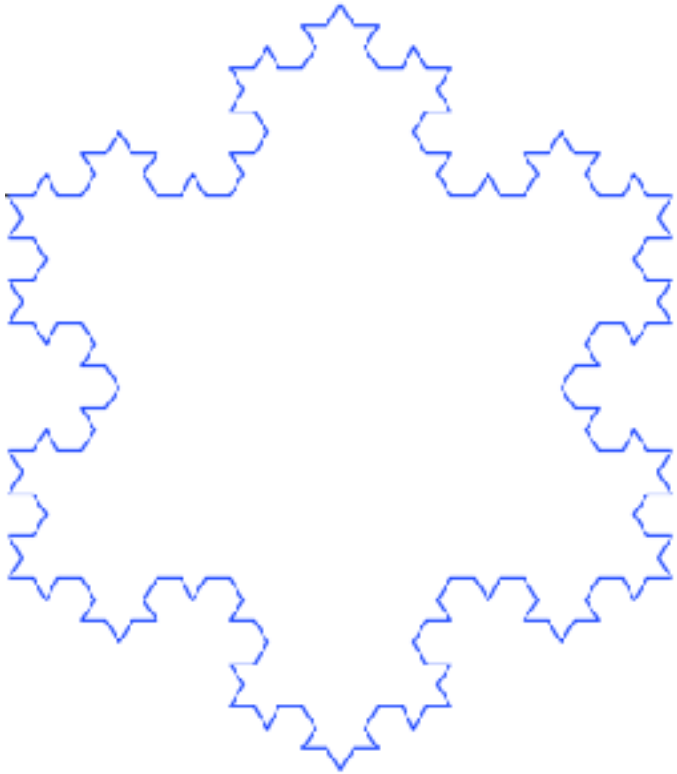
n = 2

n = 3

n = 4

```
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)
        turtle.left(60)
        kochSide(length/3, n-1)
```

Fractals: kochSnowflake



```
def kochSnowflake(length, n):
    # just call kochSide 3 times
    for step in range(3):
        kochSide(length, n)
        turtle.right(120)
```