Lesson 28:

1. Quiz: Stirling and Bell

# kind of easy when they give you the formulas

def stirling(num\_items, num\_sets):

# S(n, k) = k\*S(n-1, k) + S(n-1, k-1)

if num\_items < num\_sets:

return 0

elif num\_items == 1:

return 1

elif num\_sets == 1:

return 1

elif num\_items == num\_sets:

return 1

else:

return num\_sets \* stirling(num\_items - 1, num\_sets) + stirling(num\_items - 1, num\_sets - 1)

def bell(integer\_input):

# B(n) is the sum of S(n,k) for k = 1, 2, ... , n

summation = 0

num\_sets = 1

while integer\_input >= num\_sets:

summation += stirling(integer\_input, num\_sets)

num\_sets += 1

return summation

1. Quiz: Combating Link Spam

def compute\_ranks(graph, k):

d = 0.8 # damping factor

numloops = 10

ranks = {}

npages = len(graph)

for page in graph:

ranks[page] = 1.0 / npages

for i in range(0, numloops):

newranks = {}

for page in graph:

newrank = (1 - d) / npages

for node in graph:

if page in graph[node]:

if not is\_reciprocal\_link(graph, node, page, k):

newrank = newrank + d \* (ranks[node] / len(graph[node]))

newranks[page] = newrank

ranks = newranks

return ranks

def is\_reciprocal\_link(graph, node, page, k):

if k == 0:

if node == page:

return True

else:

return False

if node in graph[page]:

return True

for item in graph[page]:

if is\_reciprocal\_link(graph, node, item, k - 1):

return True

return False

1. Quiz: Elementary Cellular Automaton

def cellular\_automaton(non\_empty\_string, pattern\_number, n):

patterns\_dictionary = divisors(pattern\_number)

return\_string = non\_empty\_string

for x in range(0, n):

previous\_string = return\_string

return\_string = ''

for y in range(0, len(non\_empty\_string)):

last = y + 1

if last == len(non\_empty\_string):

last = 0

sub\_string = previous\_string[y - 1] + previous\_string[y] + previous\_string[last]

return\_string += patterns\_dictionary[sub\_string]

return return\_string

def divisors(pattern\_number):

divisors\_list = list()

for x in range(7, -1, -1):

contribution = 2 \*\* x

if pattern\_number % contribution < pattern\_number:

divisors\_list.append(contribution)

pattern\_number -= contribution

return pattern\_dict\_generator(divisors\_list)

def pattern\_dict\_generator(divisors\_list):

pattern\_list = dict()

for x in range(0, 8):

value = 2 \*\* x

if value in divisors\_list:

pattern\_list[pattern\_matcher(x)] = 'x'

else:

pattern\_list[pattern\_matcher(x)] = '.'

return pattern\_list

def pattern\_matcher(integer):

binary\_string = str(bin(integer))

binary\_string = binary\_string[2:]

while len(binary\_string) < 3:

binary\_string = '.' + binary\_string

binary\_string = binary\_string.replace('0', '.')

binary\_string = binary\_string.replace('1', 'x')

return binary\_string