Smith College

Computer Science

Week 13

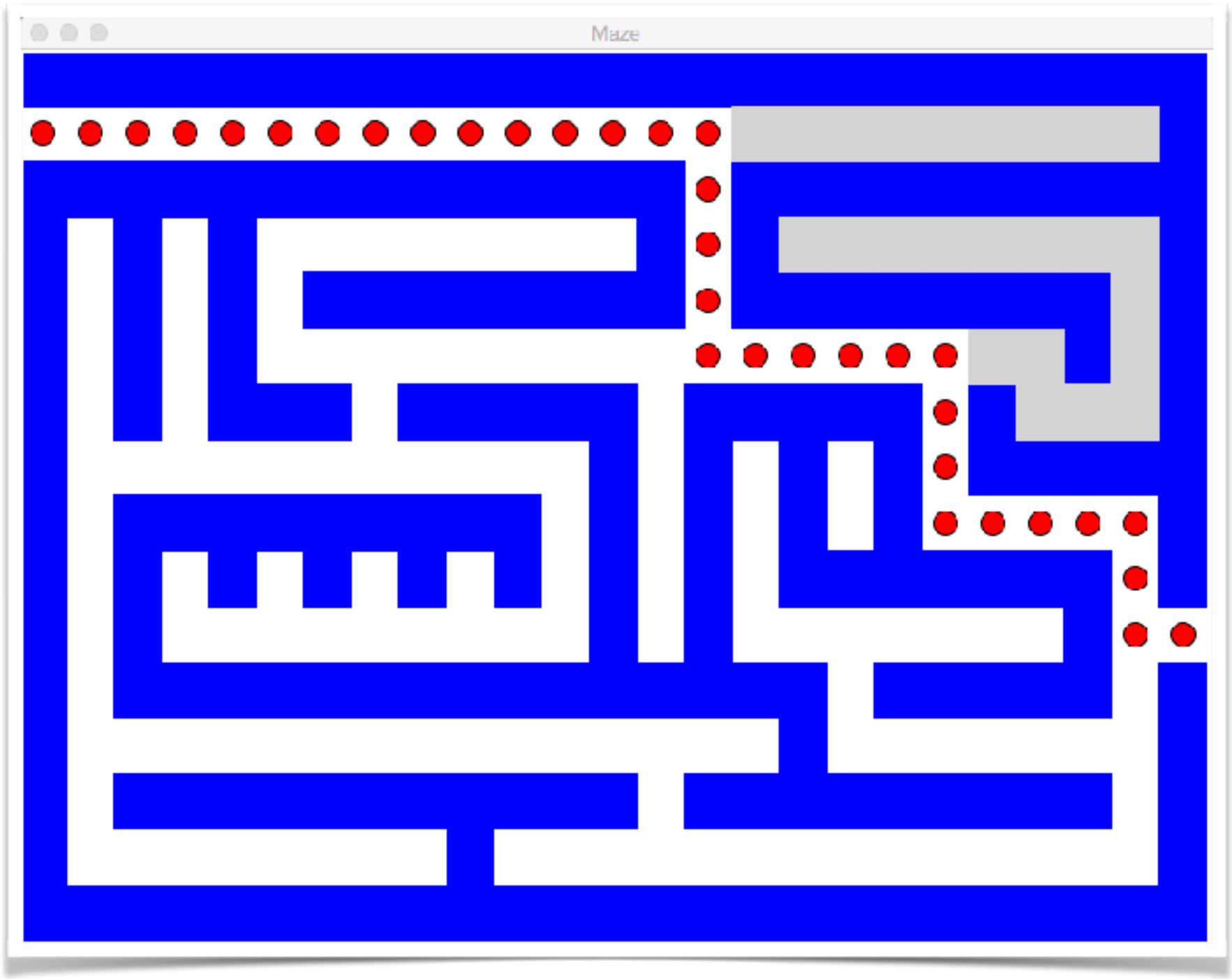
CSC111 — Spring 2018

Dominique Thiébaut
dthiebaut@smith.edu

Recursion Continued

Visiting a Maze

Start



Exit


Maze

How do we represent a maze in Python?

```
mazeText = """
##########################
.........................#
##############.##########
#.#.#........#.#........#
#.#.#.########.########.#
#.#.#..................#.#
#.#.###.#####.#####.#...#
#..........#.#.#.#.#####
#.#########.#.#.#.....#
#.#.#.#.#.#.#.#######.#
#.#........#.#.......#..
#.##############.#####.#
#..............#......#
#.##########.#########.#
#........#..............#
##########################
"""
```

Step 1: String Definition

```
mazeText = """
##############################
                              #
############## ##########
# # #          # #          #
# # # ######## ######## #
# # #                    # #
# # ### ##### ##### #    #
#             # # # # #####
# ######### # # # #      #
# # # # # # # ####### #
# #           # #          #
# ############ ######### #
#            #                #
##############################
"""
```

Step 2: Replace dots by spaces

```
maze = [
'############################',
'                           #',
'############ ##########',
'#  #  #          #  #        #',
'#  #  # ######## ######## #',
'#  #  #                  #  #',
'#  #  ### #####  #####  #    #',
'#           #  #  #  #  #####',
'#  ######### #  #  #  #      #',
'#  #  #  #  #  #  #  #######  #',
'#  #           #  #        #   ',
'#  ############## #####  #',
'#                    #        #',
'#  ##########  ########  #',
'#          #                  #',
'############################'  ]
```

Step 3: Split into List of Strings

```
maze = [
'############################',
'                          #',
'############ ##########',
'#  #  #          #  #          #',
'#  #  #  ########  ########  #',
'#  #  #                    #  #',
'#  #  ###  #####  #####  #    #',
'#              #  #  #  #  #####',
'#  #########  #  #  #  #       #',
'#  #  #  #  #  #  #  #######  #',
'#  #            #  #          #   ',
'#  ##############  #####  #',
'#                      #          #',
'#  ###########  #########  #',
'#            #                    #',
'############################'  ]
```

```
maze = [
  ———,
  ——-,
  ——-,
  ——-  ]
```

```
maze = [———, ——-,… ]
```

Step 3: Split into List of Strings

```
maze = [
'##########################',
'                          #',
'############ ##########',
'#  #  #          #  #          #',
'#  #  #  ########  ########  #',
'#  #  #                      #  #',
'#  #  ###  #####  #####  #    #',
'#              #  #  #  #  #####',
'#  #########  #  #  #  #      #',
'#  #  #  #  #  #  #  #######  #',
'#  #          #  #          #    ',
'#  ##########  ########  #',
'#          #                    #',
'##########################'  ]
```

maze[0]

Result: Each row can be accessed with an index

```
maze = [
'############################',
'                           #',
'############ ##########',
'#  #  #          #  #          #',
'#  #  #  ########  ########  #',
'#  #  #                      #  #',
'#  #  ###  #####  #####  #     #',
'#              #  #  #  #  #####',
'#  #########  #  #  #  #        #',
'#  #  #  #  #  #  #  #  #######  #',
'#  #            #  #          #    ',
'#  ############  #####  #',
'#                #                  #',
'#  ##########  ########  #',
'#           #                      #',
'############################'  ]
```

maze[3]

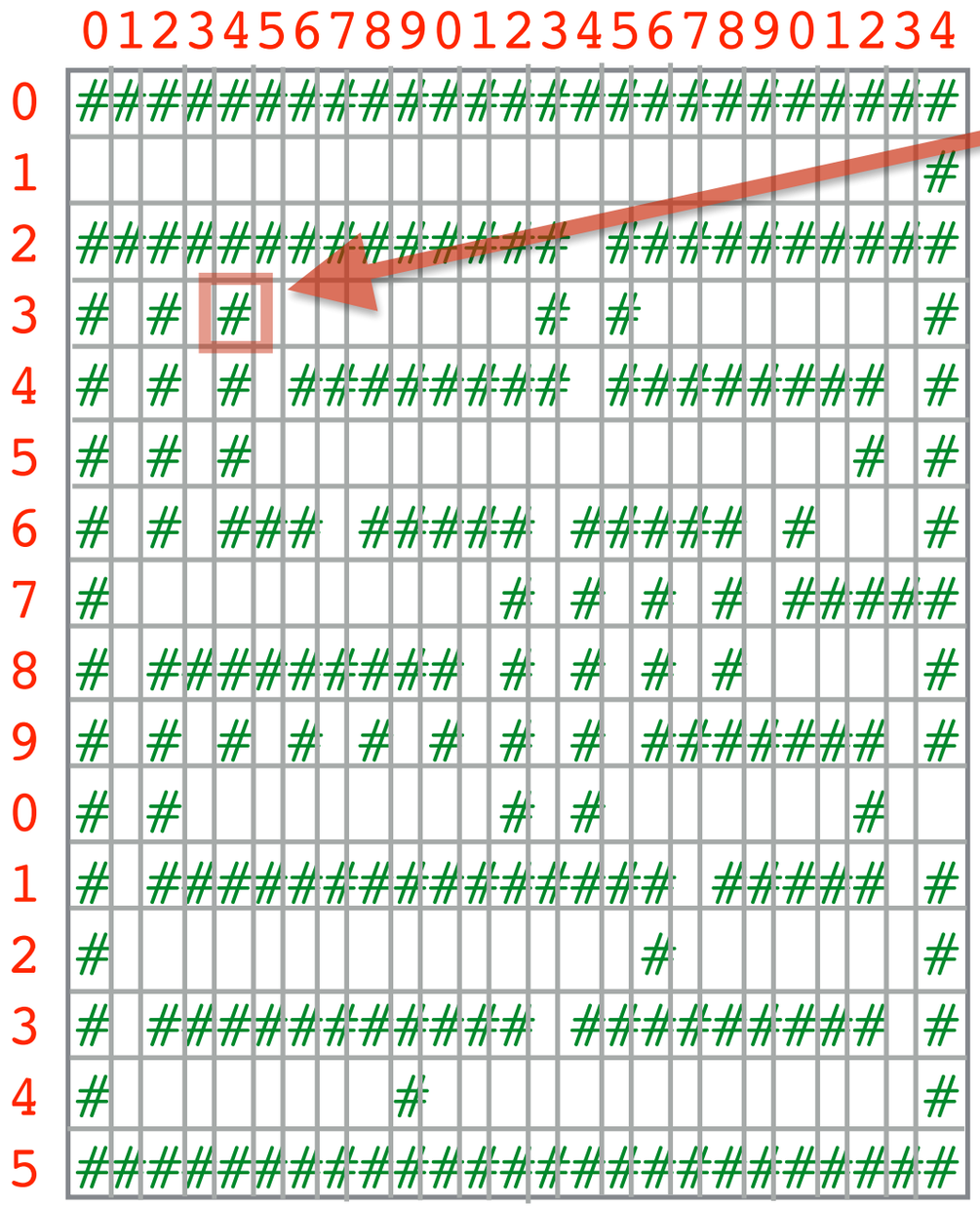
Result: Each row can be accessed with an index

maze[3][4]

```
maze = [
'############################',
'                           #',
'########### ##########',
'#  #  #          # #          #',
'# # # ######## ######## #',
'# # #                    # #',
'# # ### ##### ##### #    #',
'#              # # # # #####',
'# ######### # # # #      #',
'# # # # # # # # ####### #',
'# #           # #        #  ',
'# ############## ##### #',
'#                  #       #',
'# ########## ######### #',
'#          #                #',
'############################' ]
```

maze[3]

Result: a 2-Dimensional Structure!



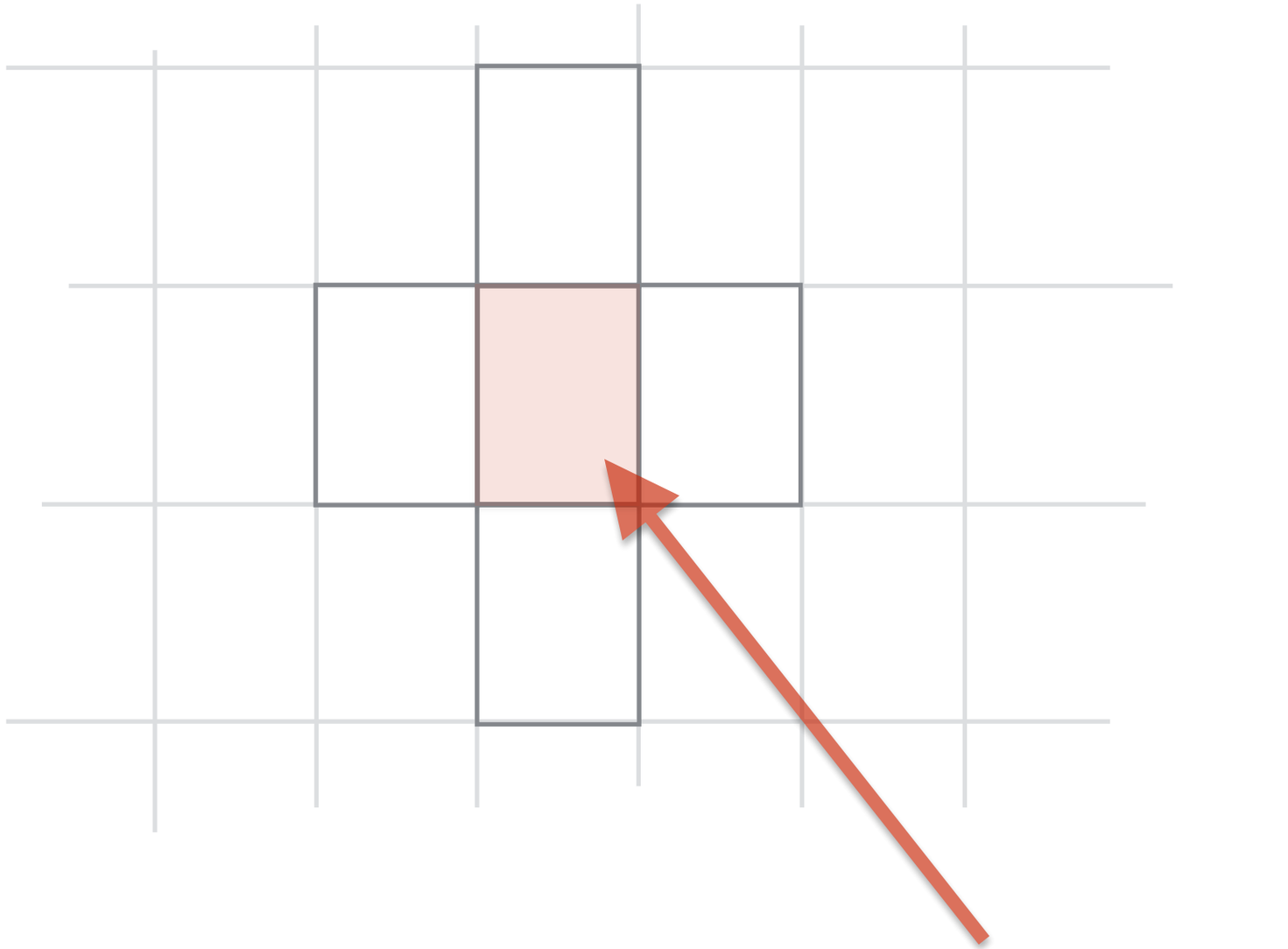
maze

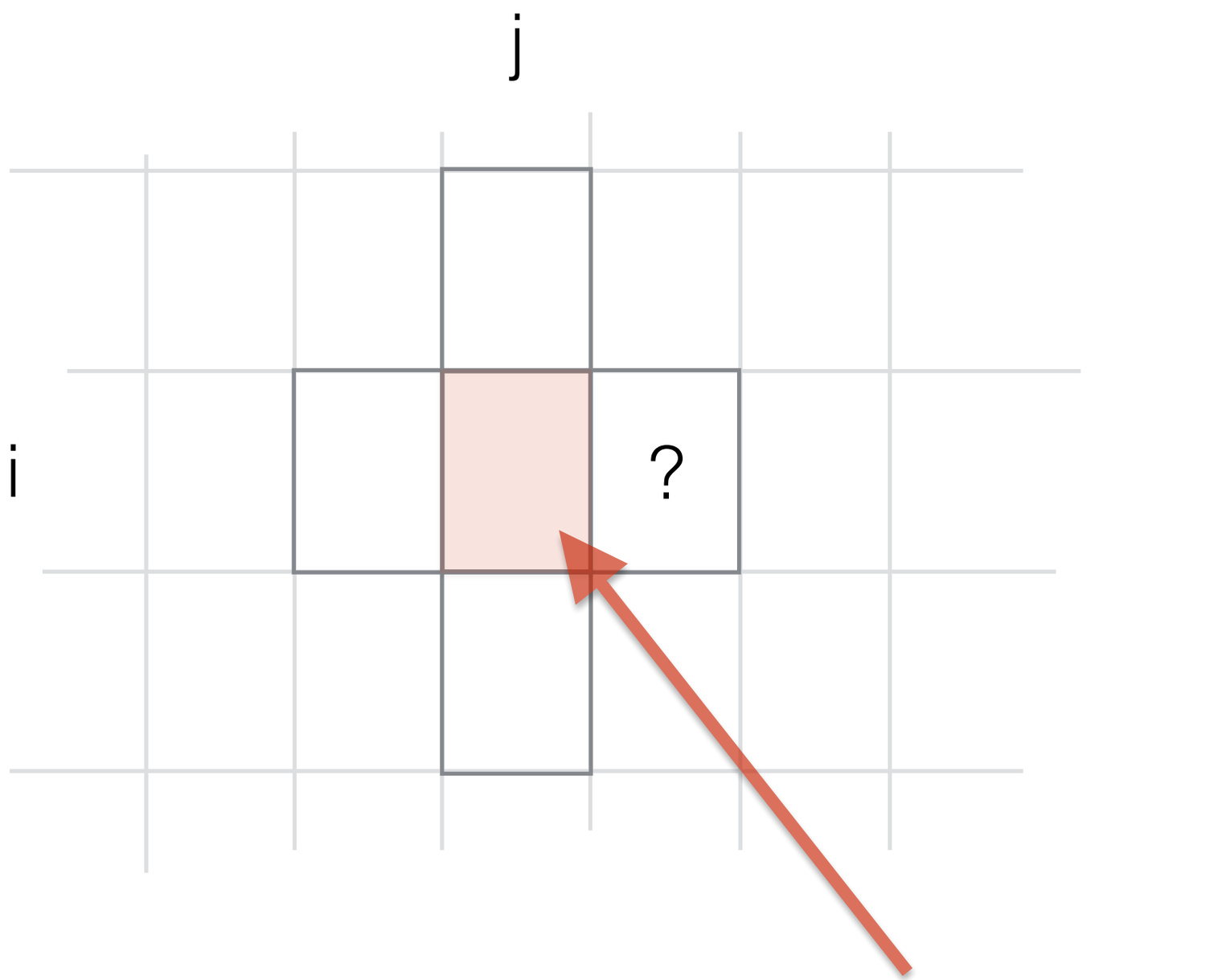
maze[3][4]

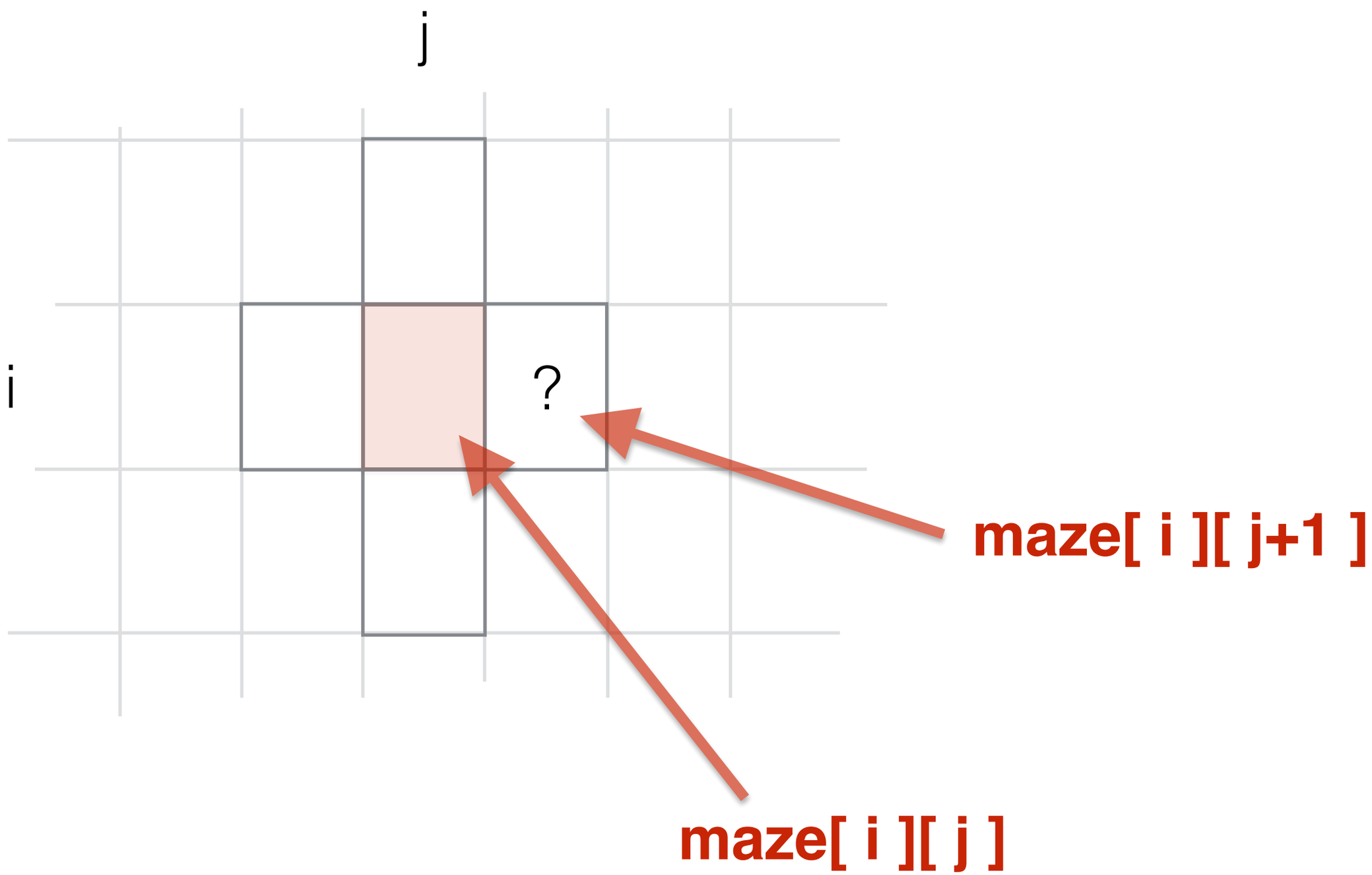
each cell is defined by a row index, and a column index: maze[row][col]

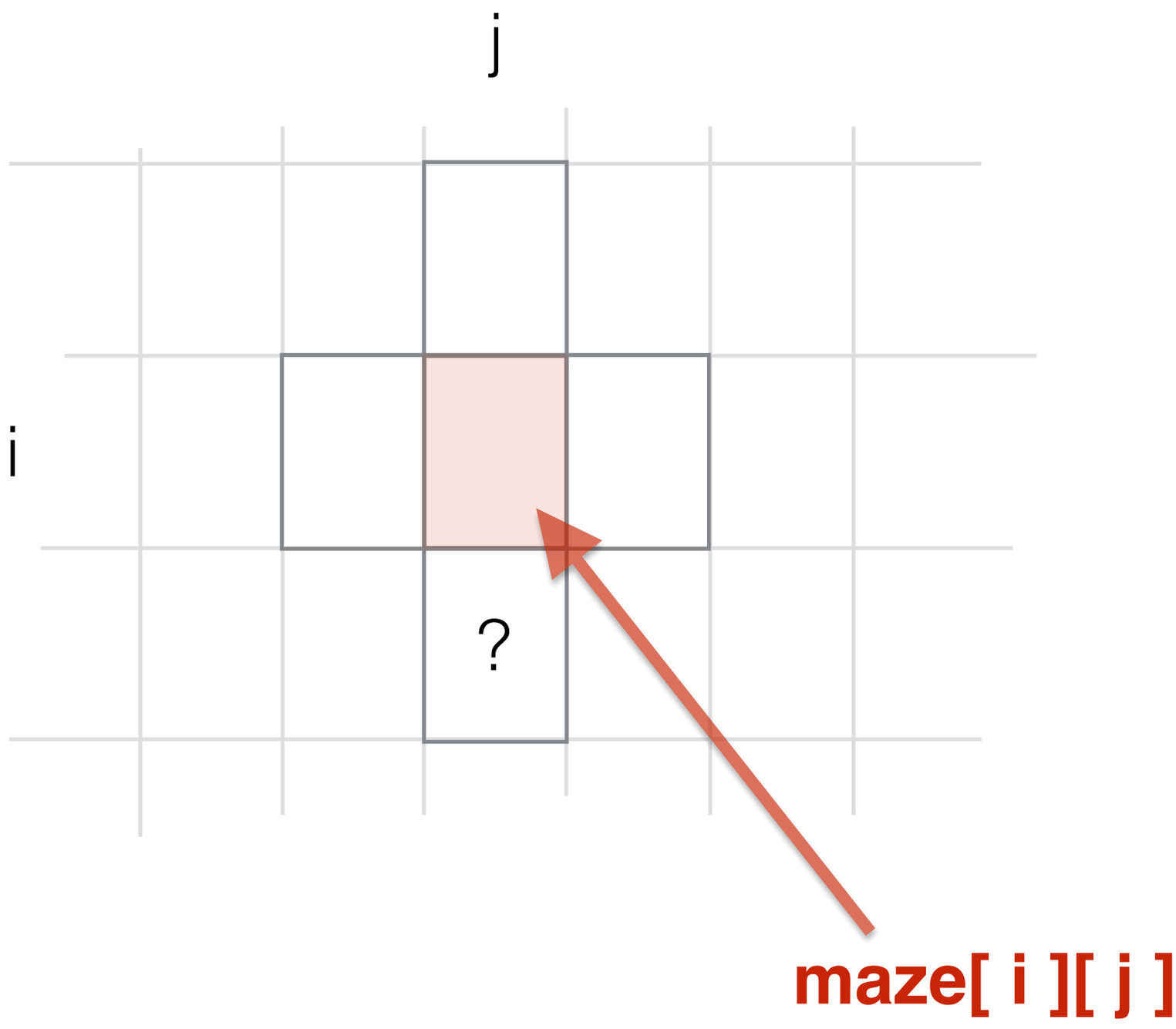
Result: a 2-Dimensional Array

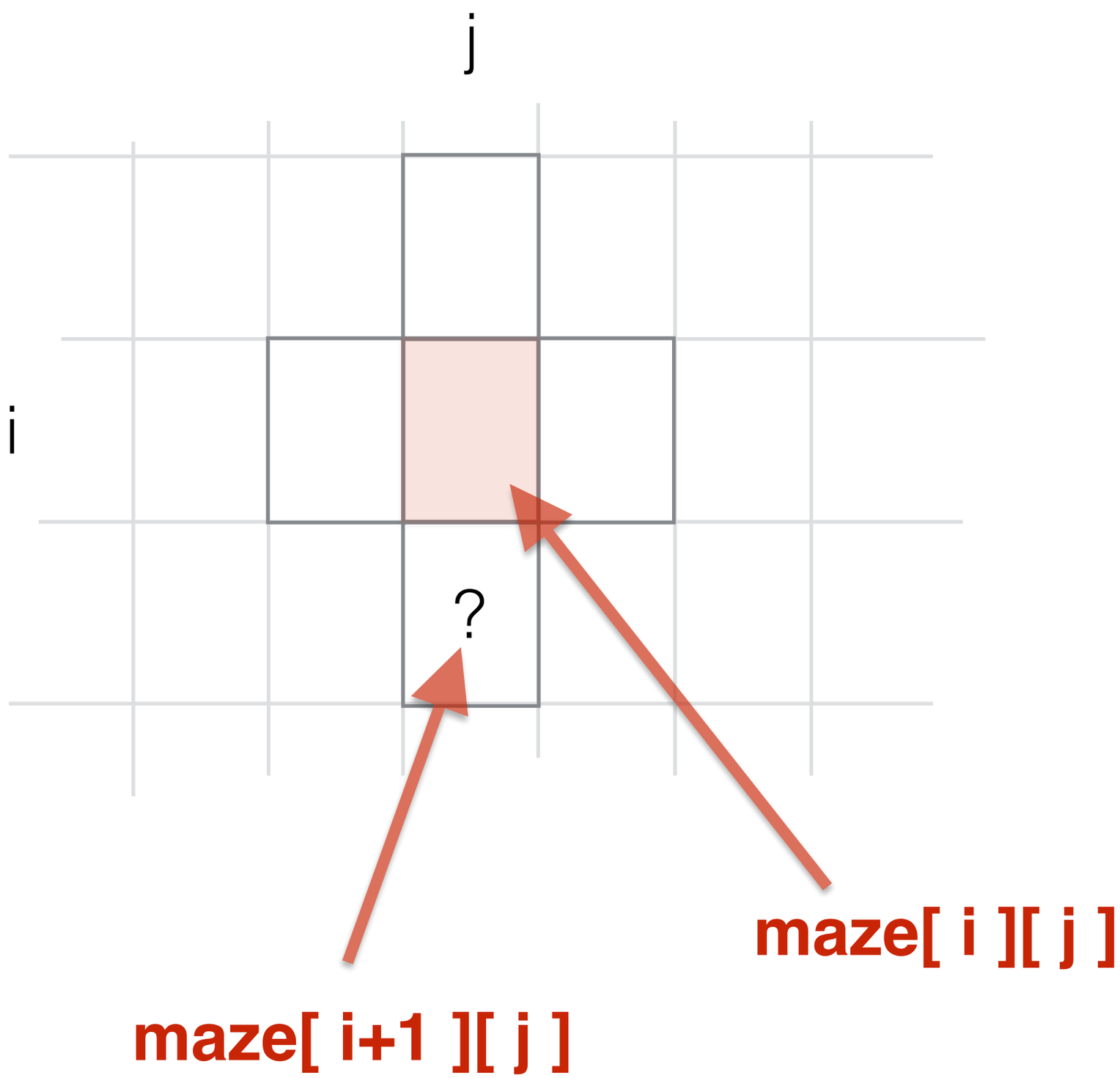
Algorithm

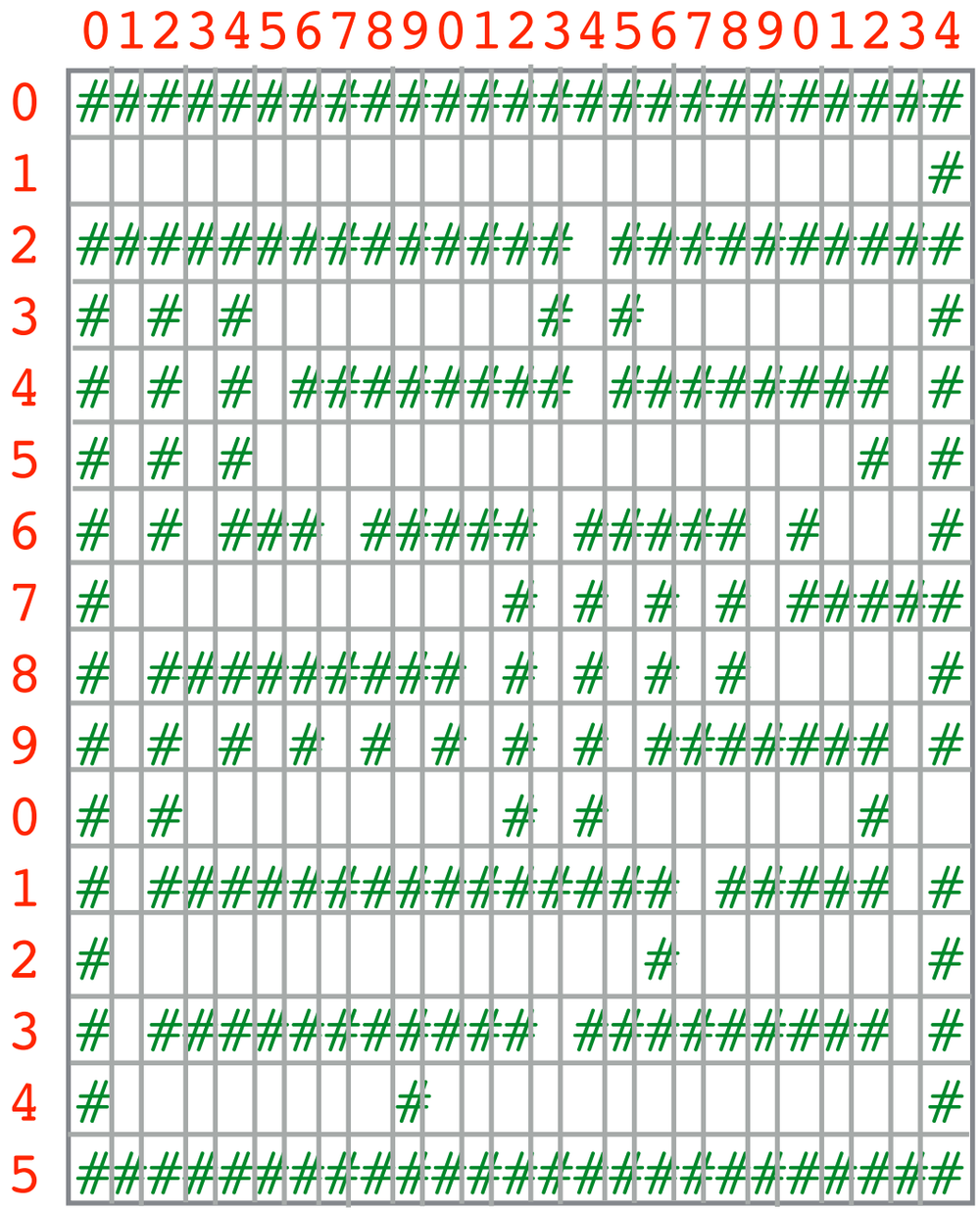



j
i
?
maze[i][j]

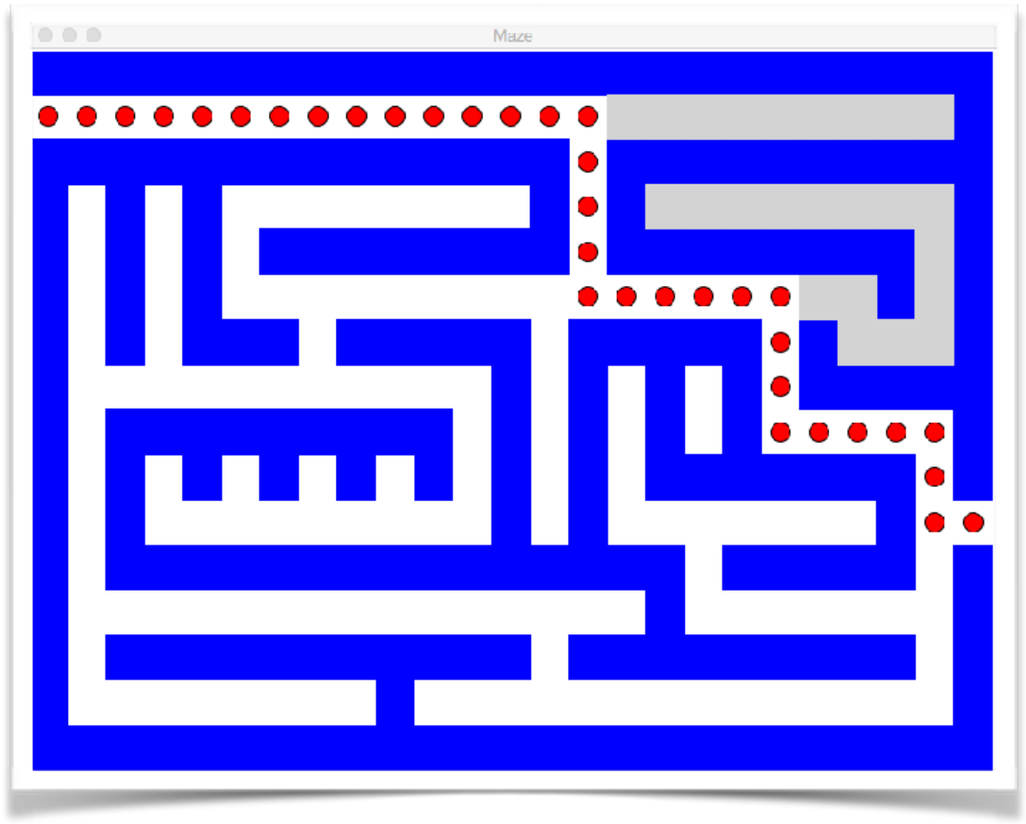

j
i
?
maze[i][j+1]
maze[i][j]

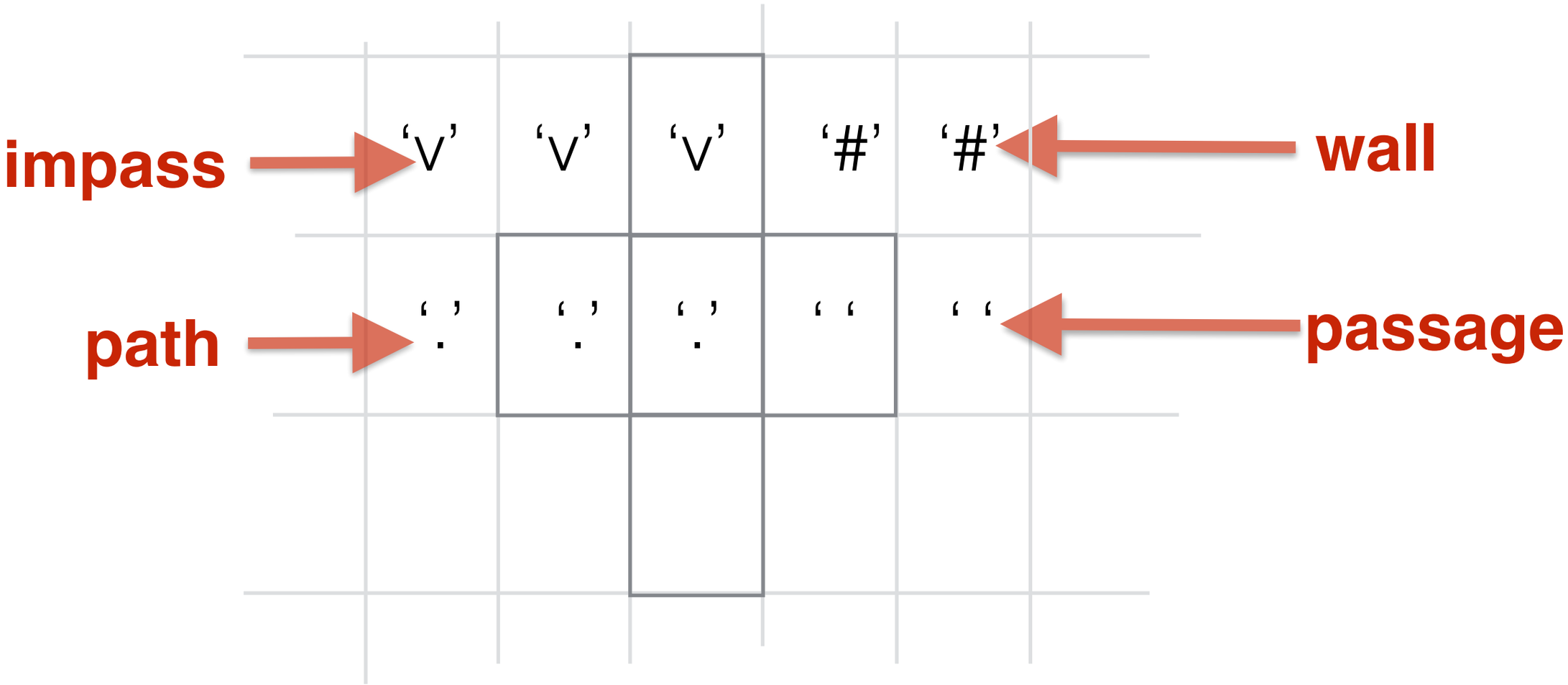

j
i
?
maze[i][j]

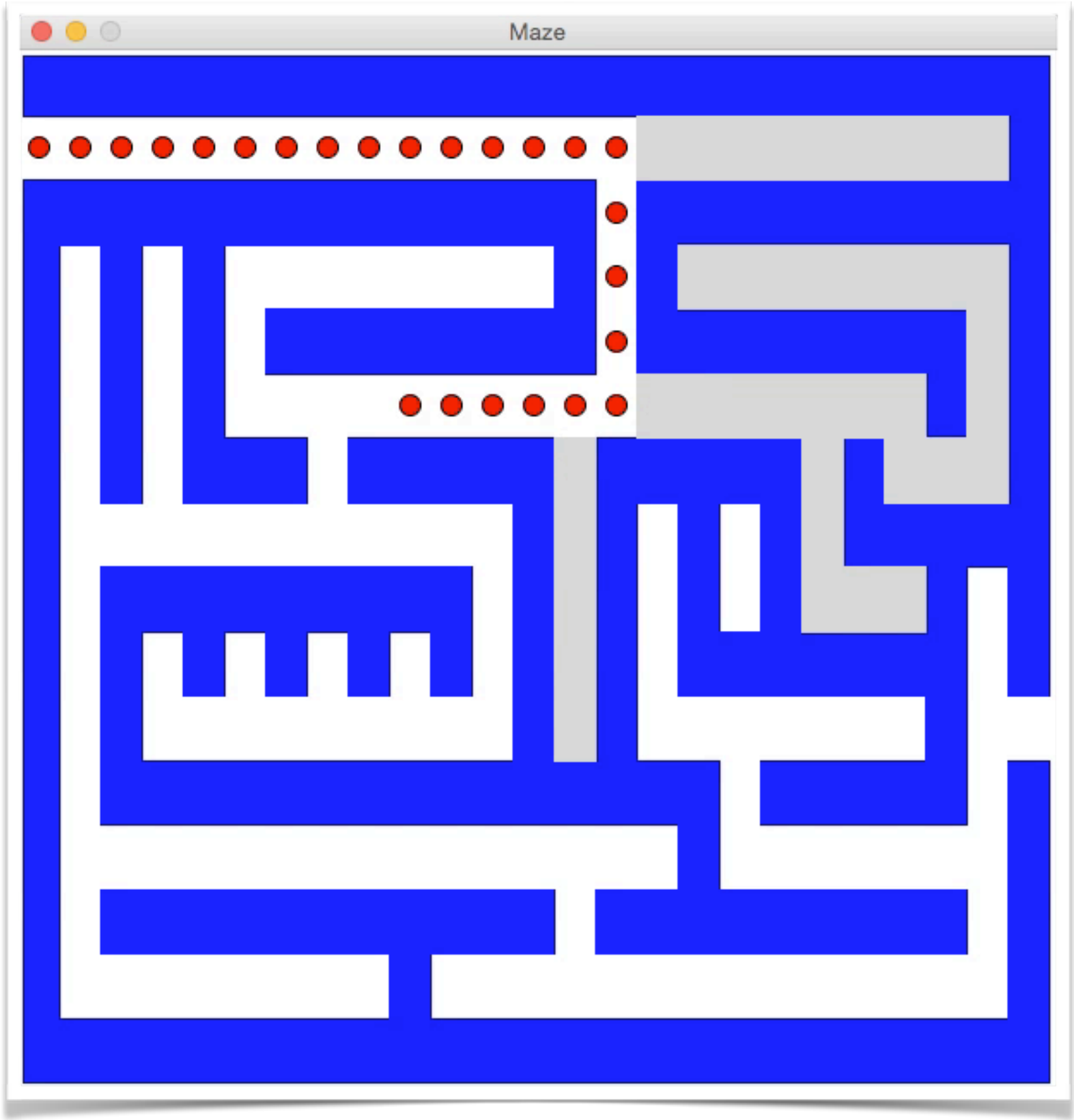

j
i
?
maze[i][j]
maze[i+1][j]



maze




impass
'v'
'v'
'v'
'#'
'#'
wall
path
'.'
'.'
'.'
' '
' '
passage


Maze

Exploring the Code

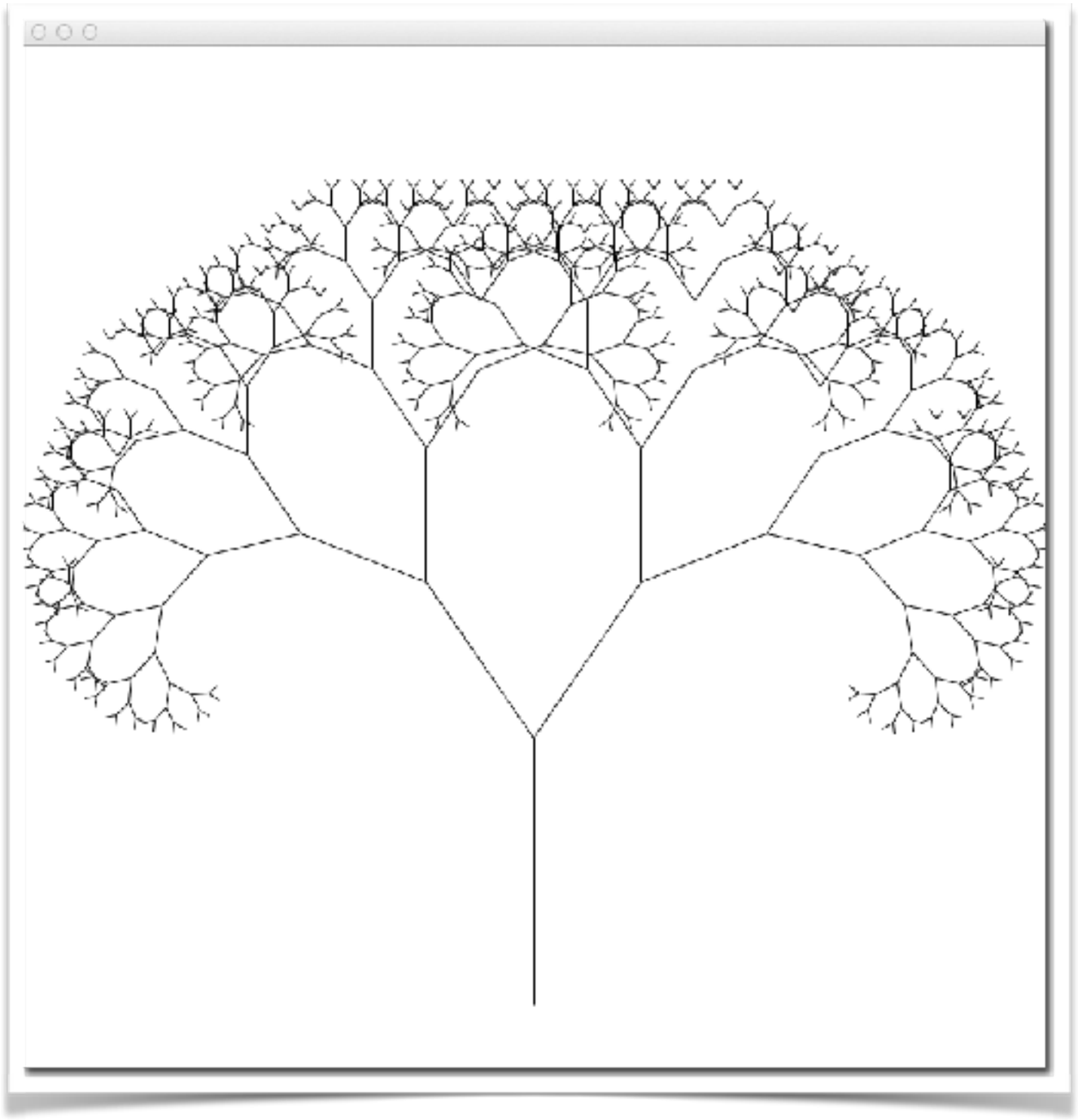
Observing the Recursive Nature of visitMaze()

mazeTreasures.py - /Users/thiebaut/Desktop/Dropbox/111/Week12/mazeTreasures.py

```
    for letter in line:
        letters.append( letter )
    letters[j] = char
    maze[i] = ''.join( letters )

def visitMaze( maze, i, j, win ):
    """recursive visit of the maze.  Returns True when it
    has found the exit, False otherwise"""

    setBreakCrumb( i, j, win )
    setChar( maze, i, j, '.' )

    # wait for user to click before recursing
    win.getMouse()

    #printMaze( maze )
    if ( i != STARTi or j != STARTj ) \
        and ( (i==0 or i==len( maze )-1 ) or (j==0 or j==len( maze[0] )-1 ) ):
        return True

    #--- try the four directions around where we are ---
    #--- to the right? ---
    if j+1< len( maze[0] ) and maze[i][j+1]==' ':
        if visitMaze( maze, i, j+1, win ) == True:
                return True # found an exit by going right!

    #--- down? ---
    if i+1< len( maze ) and maze[i+1][j]==' ':
        if visitMaze( maze, i+1, j, win ) == True:
                return True # found an exit by going down!
```

Ln: 239 Col: 0

Fractal Trees

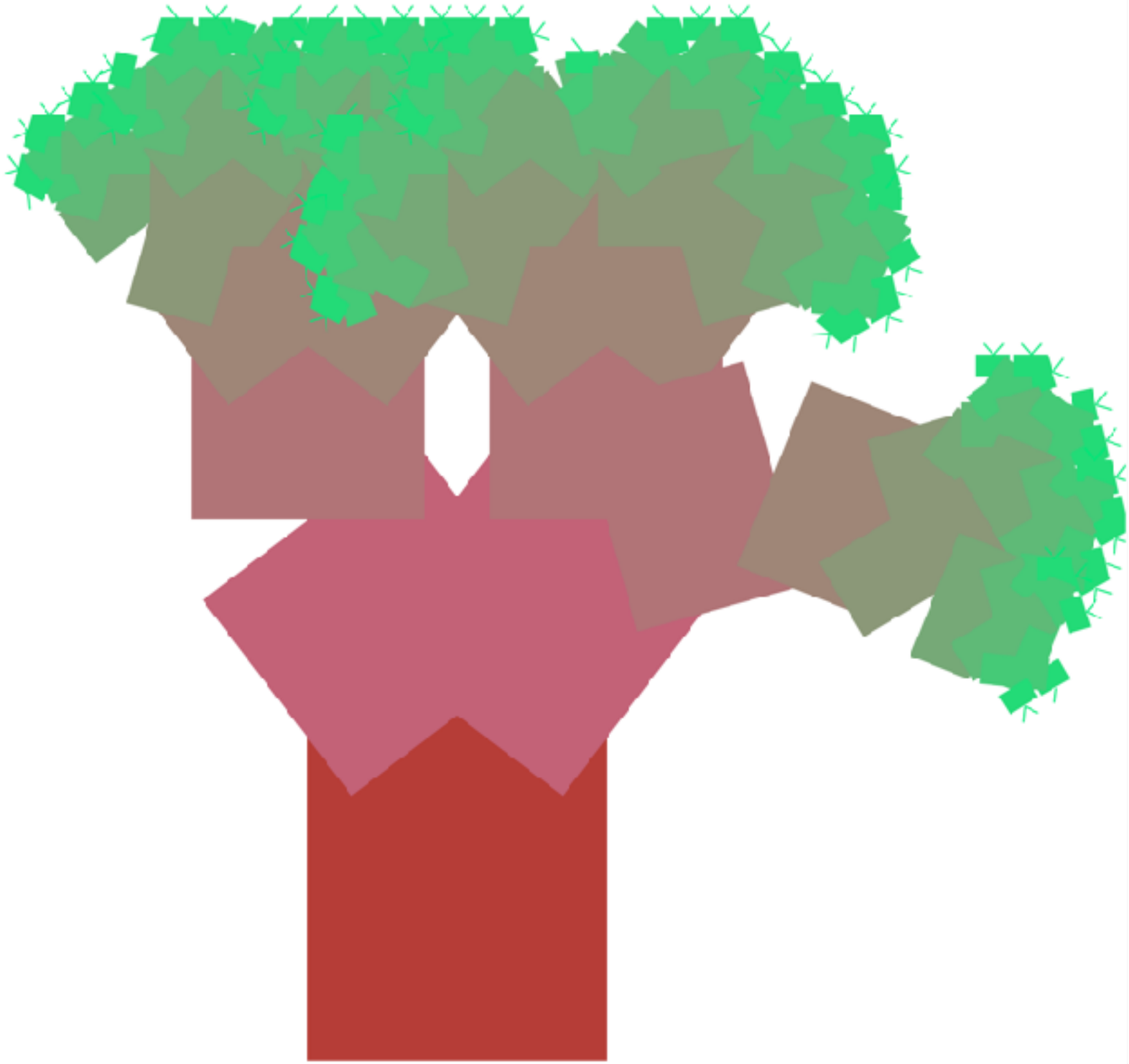


Let's Play with the Fractal Tree

- Make the order larger (e.g. 12)
- Make it draw the right side first
- Change the angle *theta*
- Make the drawing of a branch random
- Make the size of a branch very large
- Make the color proportional to the order of recursion

Fractal Tree

color_rgb(order*20, 255-order*20, 100)



Examples

https://www.youtube.com/watch?v=yWRFCSIzej0

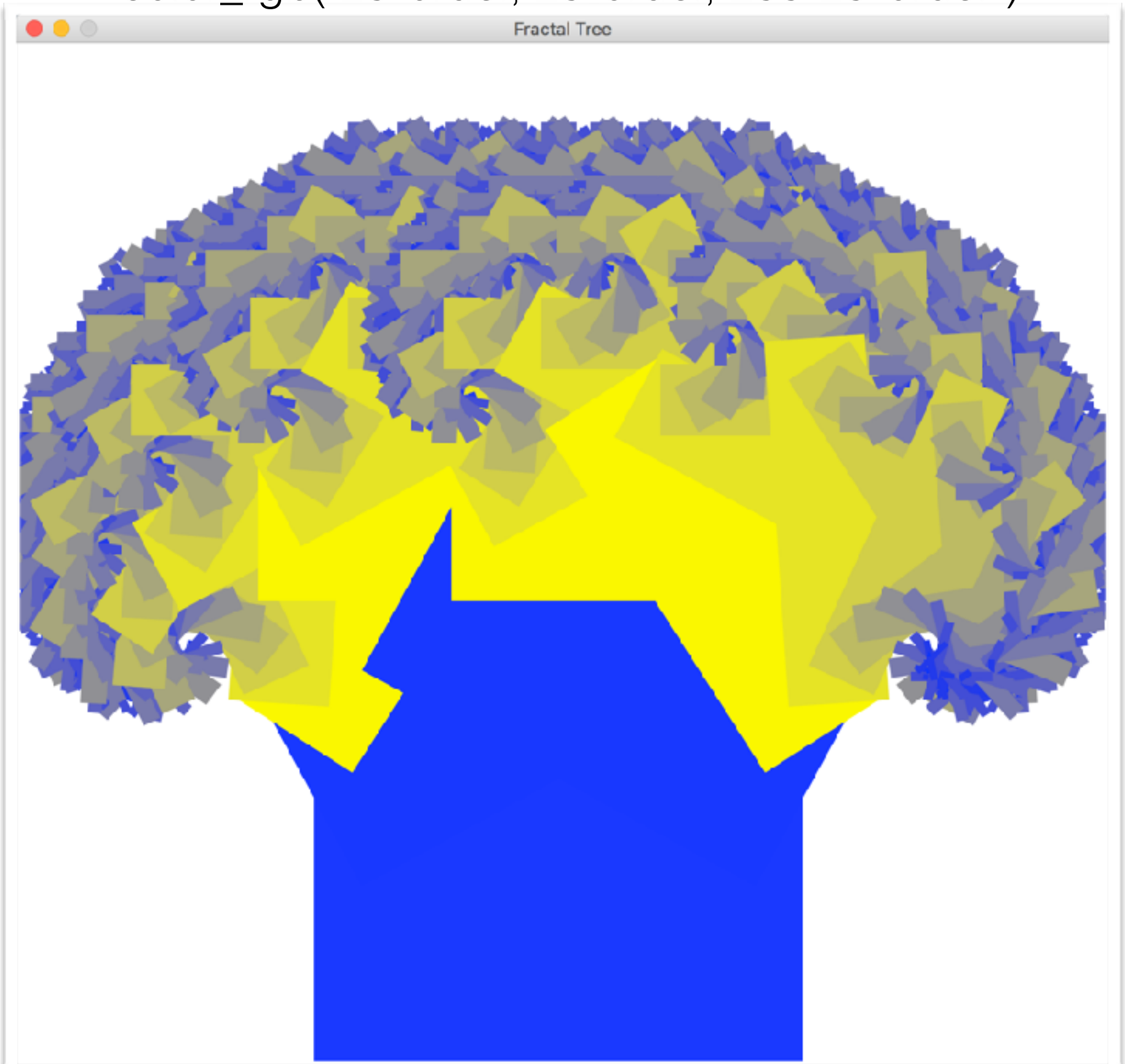
https://www.youtube.com/watch?v=v5qvePPHM6E

https://www.youtube.com/watch?v=s0Nu1OaBXa8

The result…

https://www.youtube.com/watch?v=tiqS6J_A6Jc

color_rgb(25*order, 25*order, 255-25*order)



Tough problems, simple solutions

Recursive Functions

- Finding the Largest in a List
- Finding the Smallest in a List
- Factorial
- Traversing a Maze
- Recursive Trees
- **Towers of Hanoi**

We stopped here last time…

Solving Problems Recursively:

Towers of Hanoi

A B C

http://en.wikipedia.org

The Legend of the Towers of Hanoi

The puzzle was invented by the **French mathematician Édouard Lucas** in 1883.
There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it surrounded by **64 golden disks**. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the immutable rules of the Brahma, since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. **According to the legend, when the last move of the puzzle will be completed, the world will end**.
It is not clear whether Lucas invented this legend or was inspired by it.

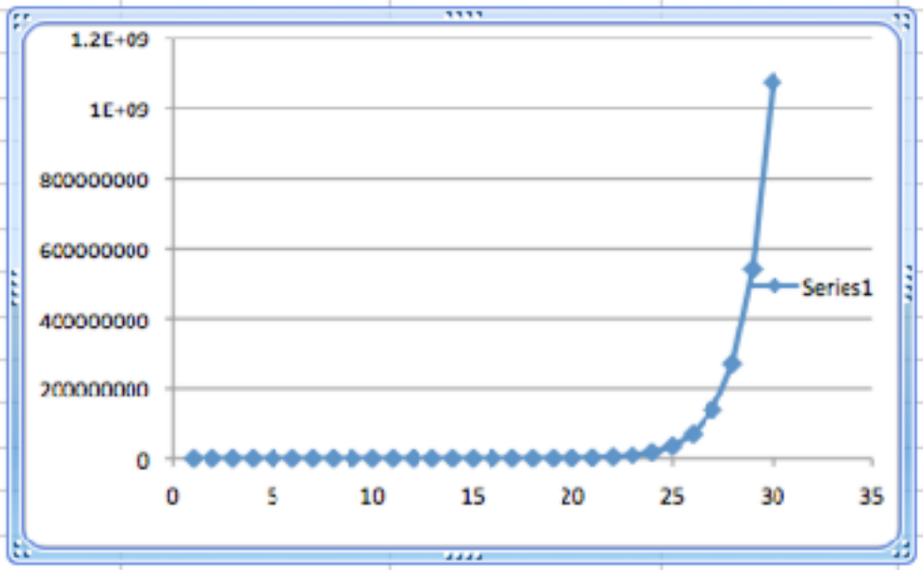
From Wikipedia, https://en.wikipedia.org/wiki/%C3%89douard_Lucas

Coding
The Towers of Hanoi

How long would it take a good PC to move 64 disks on 3 pegs?

Timing Analysis

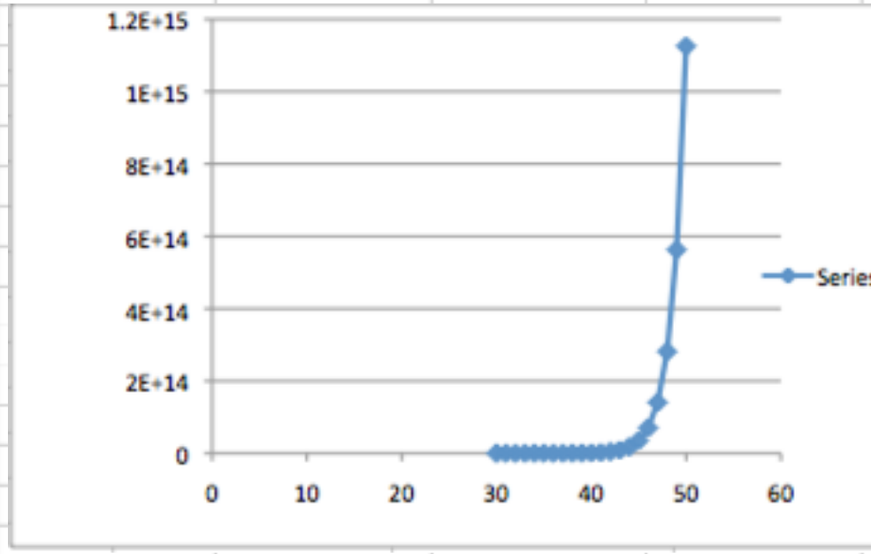
	TOWERS OF HANOI								
2			Number of	Number of	Number of	Number of		Laptop	
3	Number of Disks	Number of Moves	Sec required	Min required	Hours require	Days required		3GHz	Speed
4	1	1	3.33E-10					3.00E+09	moves/second
5	2	3	1.00E-09						
6	3	7	2.33E-09						
7	4	15	5.00E-09						
8	5	31	1.03E-08						
9	6	63	2.10E-08						
10	7	127	4.23E-08						
11	8	255	8.50E-08						
12	9	511	1.70E-07						
13	10	1023	3.41E-07						
14	11	2047	6.82E-07						
15	12	4095	1.37E-06						
16	13	8191	2.73E-06						
17	14	16383	5.46E-06						
18	15	32767	1.09E-05						
19	16	65535	2.18E-05						
20	17	131071	4.37E-05						
21	18	262143	8.74E-05						
22	19	524287	1.75E-04						
23	20	1048575	3.50E-04						
24	21	2097151	6.99E-04						
25	22	4194303	1.40E-03						



http://cs.smith.edu/dftwiki/index.php/CSC231_Analysis_of_the_Towers_of_Hanoi

TOWERS OF HANOI

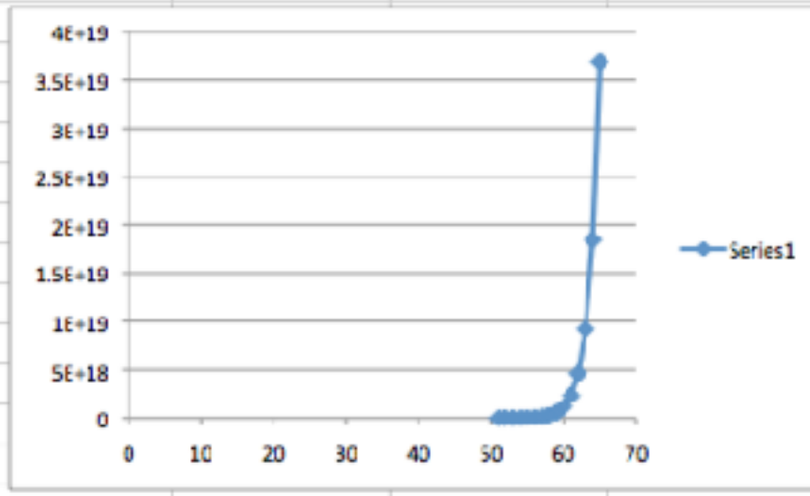
Number of Disks	Number of Moves	Number of Sec required	Number of Min required	Number of Hours require	Number of Days required		Laptop 3GHz	Speed
1	1	3.33E-10					3.00E+09	moves/second
31	2147483647	7.16E-01						
32	4294967295	1.43E+00	<-- 1 Second					
33	8589934591	2.86E+00						
34	17179869183	5.73E+00						
35	34359738367	1.15E+01						
36	68719476735	2.29E+01						
37	1.37439E+11	4.58E+01						
38	2.74878E+11	9.16E+01	1.527099483	<-- 1 minute				
39	5.49756E+11	1.83E+02	3.054198966					
40	1.09951E+12	3.67E+02	6.108397932					
41	2.19902E+12	7.33E+02	12.21679586					
42	4.39805E+12	1.47E+03	24.43359173					
43	8.79609E+12	2.93E+03	48.86718346					
44	1.75922E+13	5.86E+03	97.73436691	<-- 1 hour				
45	3.51844E+13	1.17E+04	195.4687338	3.25781223				
46	7.03687E+13	2.35E+04	390.9374677	6.515624461				
47	1.40737E+14	4.69E+04	781.8749353	13.03124892				
48	2.81475E+14	9.38E+04	1563.749871	26.06249784	<-- 1 day			
49	5.6295E+14	1.88E+05	3127.499741	52.12499569	2.17187482			
50	1.1259E+15	3.75E+05	6254.999482	104.2499914	4.343749641			
51	2.2518E+15	7.51E+05	12509.99896	208.4999827	8.687499281			
52	4.5036E+15	1.50E+06	25019.99793	416.9999655	17.37499856			
53	9.0072E+15	3.00E+06	50039.99586	833.999931	34.74999712			
54	1.80144E+16	6.00E+06	100079.9917	1667.999862	69.49999425			
55	3.60288E+16	1.20E+07	200159.9834	3335.999724	138.9999885			
56	7.20576E+16	2.40E+07	400319.9669	6671.999448	277.999977			
57	1.44115E+17	4.80E+07	800639.9338	13343.9989	555.999954	<-- 1 year		



http://cs.smith.edu/dftwiki/index.php/CSC231_Analysis_of_the_Towers_of_Hanoi

TOWERS OF HANOI

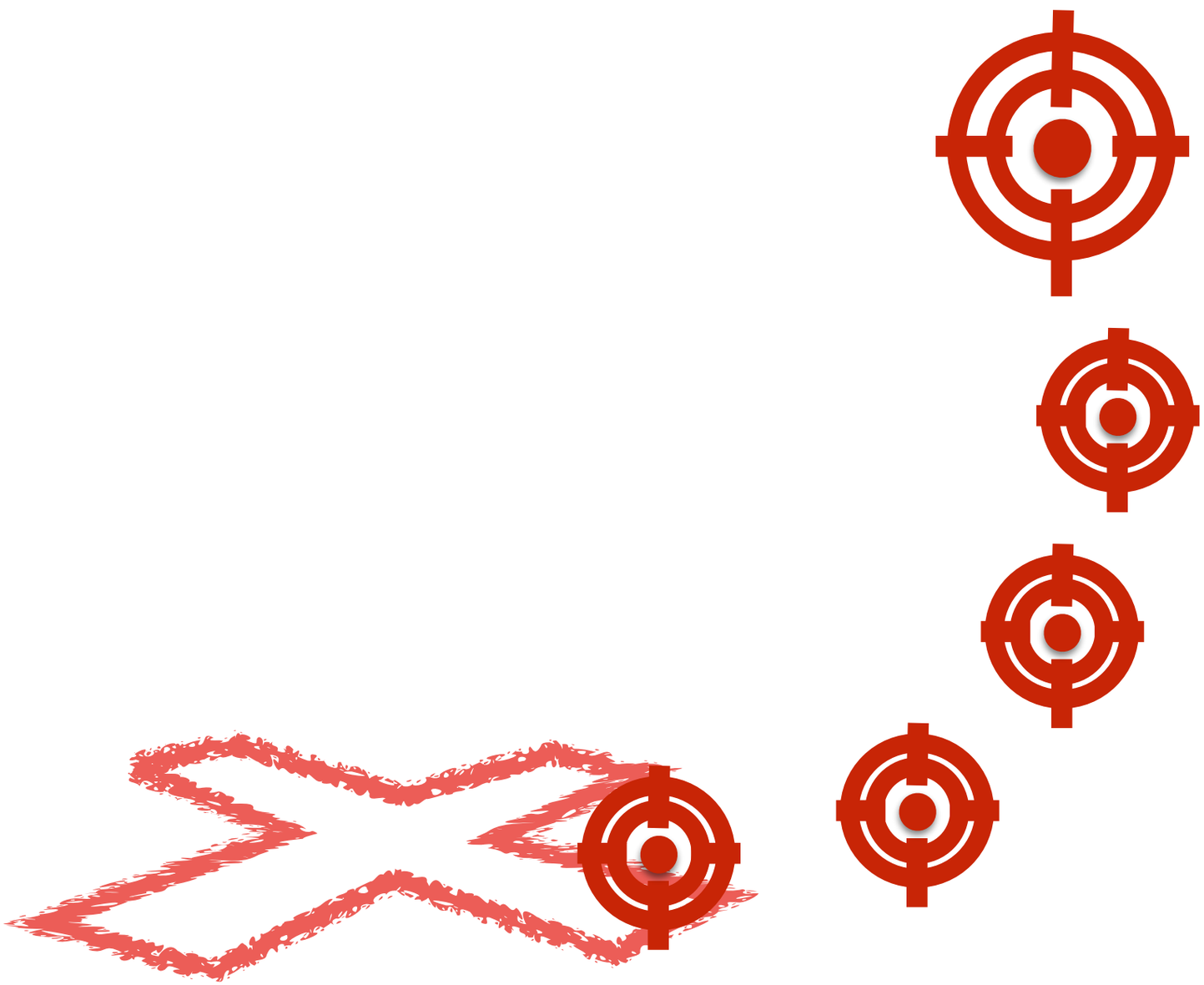
Number of Disks	Number of Moves	Number of Sec required	Number of Min required	Number of Hours require	Number of Days required		Laptop 3GHz	Speed
1	1	3.33E-10					3.00E+09	moves/second
2	3	1.00E-09						
39	5.49756E+11	1.83E+02	3.054198966					
40	1.09951E+12	3.67E+02	6.108397932					
41	2.19902E+12	7.33E+02	12.21679586					
42	4.39805E+12	1.47E+03	24.43359173					
43	8.79609E+12	2.93E+03	48.86718346					
44	1.75922E+13	5.86E+03	97.73436691	<-- 1 hour				
45	3.51844E+13	1.17E+04	195.4687338	3.25781223				
46	7.03687E+13	2.35E+04	390.9374677	6.515624461				
47	1.40737E+14	4.69E+04	781.8749353	13.03124892				
48	2.81475E+14	9.38E+04	1563.749871	26.06249784	<-- 1 day			
49	5.6295E+14	1.88E+05	3127.499741	52.12499569	2.17187482			
50	1.1259E+15	3.75E+05	6254.999482	104.2499914	4.343749641			
51	2.2518E+15	7.51E+05	12509.99896	208.4999827	8.687499281			
52	4.5036E+15	1.50E+06	25019.99793	416.9999655	17.37499856			
53	9.0072E+15	3.00E+06	50039.99586	833.999931	34.74999712			
54	1.80144E+16	6.00E+06	100079.9917	1667.999862	69.49999425			
55	3.60288E+16	1.20E+07	200159.9834	3335.999724	138.9999885			
56	7.20576E+16	2.40E+07	400319.9669	6671.999448	277.999977			
57	1.44115E+17	4.80E+07	800639.9338	13343.9989	555.999954	<-- 1 year		
58	2.8823E+17	9.61E+07	1601279.868	26687.99779	1111.999908	3.0549448		
59	5.76461E+17	1.92E+08	3202559.735	53375.99558	2223.999816	6.1098896		
60	1.15292E+18	3.84E+08	6405119.47	106751.9912	4447.999632	12.2197792		
61	2.30584E+18	7.69E+08	12810238.94	213503.9823	8895.999264	24.4395584		
62	4.61169E+18	1.54E+09	25620477.88	427007.9647	17791.99853	48.8791168		
63	9.22337E+18	3.07E+09	51240955.76	854015.9293	35583.99706	97.7582337		
64	1.84467E+19	6.15E+09	102481911.5	1708031.859	71167.99411	195.516467		
65	3.68935E+19	1.23E+10	204963823	3416063.717	142335.9882	391.032935	<-- 400 years	



Final Exam

From Here to There…

Most Important Tips



Most Important Tips

- Small, start small
- Don't work with the original data
- Take a small sample that is representative
- Take all the shortcuts possible

Most Important Tips

- Create small test data set with all possible cases for exceptions
- Make sure your program works with
  - empty lists
  - empty strings
  - 0, 1 or ***n*** results

Most Important Tips

- Reuse code we have created before (solution programs, programs in the book)
- Solve only one problem at a time
- Do not be afraid of exceptions!
- "*It would be nice if I could…*" —> search python.org