Review

• Recursion
• Factorial (Iterative and Recursive versions)
• Call Stack (Last-in, first-out Queue)
• Tracing recursive functions
• Fibonacci Sequence – Recursive Implementation
• Recursive Maze Generation
One can declare an array of any type

```java
int myInt;
float myFloat;
String myStr;
```

... just add `[]`

To create and size the array, use the `new` keyword

```java
int[] myInts = new int[10];
float[] myFloats = new float[20];
String[] myStrs = new String[30];
```
One can declare an array of custom classes

```java
Mammoth[] mammoths;    // declare array variable

void setup() {
    mammoths = new Mammoth[30]; // create + size array
}

class Mammoth {
    String name;
    String sound;

    Mammoth( String name, String sound ) {
        this.name = name;
        this.sound = sound;
    }
}
```
If this is a float...

    float myFloat;

and this is an array of floats...

    float[] myFloats;

what is this?

    float[][] myFloats2;
# Declare, size, and fill a 2D array

```java
void setup() {

    float[][] myFloats2 = new float[10][10];

    for (int i=0; i<10; i++)
    {
        for (int j=0; j<10; j++)
        {
            myFloats2[i][j] = random(100);
        }
    }
}
```

![Diagram of a 2D array](ex1.pde)
float[][] vals;

void setup() {
    vals = new float[20][300];

    for (int i=0; i<20; i++) {
        println(vals[i].length); // What is going on here?
    }
}
“Ragged” Arrays

float[][] ragged;

void setup() {
    ragged = new float[5][];
    for (int i=0; i<5; i++) {
        int n = int(random(10));
        ragged[i] = new float[n];
    }
    for (int i=0; i<5; i++) {
        println(ragged[i].length);
    }
}
Cellular Automata

- **A regular grid of Cells**
- **Cell**
- **Two States**
  1. On
  2. Off
- **Neighborhood**
- Cell states evolve over time according to a predefined set of rules.
Sample Set of Rules – Conway's Game of Life

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.

2. Any live cell with two or three live neighbors lives on to the next generation.

3. Any live cell with more than three live neighbors dies, as if by overcrowding.

4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

An example of "Emergence"

Interesting Patterns – Conway's Game of Life

<table>
<thead>
<tr>
<th>Still lives</th>
<th>Oscillators</th>
<th>Spaceships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Blinker (period 2)</td>
<td>Glider</td>
</tr>
<tr>
<td>Beehive</td>
<td>Toad (period 2)</td>
<td>Lightweight spaceship (LWSS)</td>
</tr>
<tr>
<td>Loaf</td>
<td>Beacon (period 2)</td>
<td></td>
</tr>
<tr>
<td>Boat</td>
<td>Pulsar (period 3)</td>
<td></td>
</tr>
</tbody>
</table>

Top-level procedure

1. Draw the current grid
2. Advance game by applying rules to all cells of current and filling next
3. Swap current and next grid
int N = 5;
boolean[] cell = new boolean[N];

← One-dimensional array
int \ N = 5;
boolean[][] cell = new boolean[N][N];

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
int N = 5;
boolean[][] cell = new boolean[N][N];

cell[1][2] = true;

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
// 3-Dimensional Array

int N = 50;
boolean[][][] cell = new boolean[N][N][2];

cell[1][2][0] = true;