Recursion I

CSE 120 Spring 2017

Instructor: Justin Hsia

Teaching Assistants: Anupam Gupta, Braydon Hall, Eugene Oh, Savanna Yee

Administrivia

- Assignments:
  - Controlling Elli *due tonight* (5/1)
  - Mid-Quarter Survey due Wednesday (5/3)
  - Living Computers Museum Report (5/14)

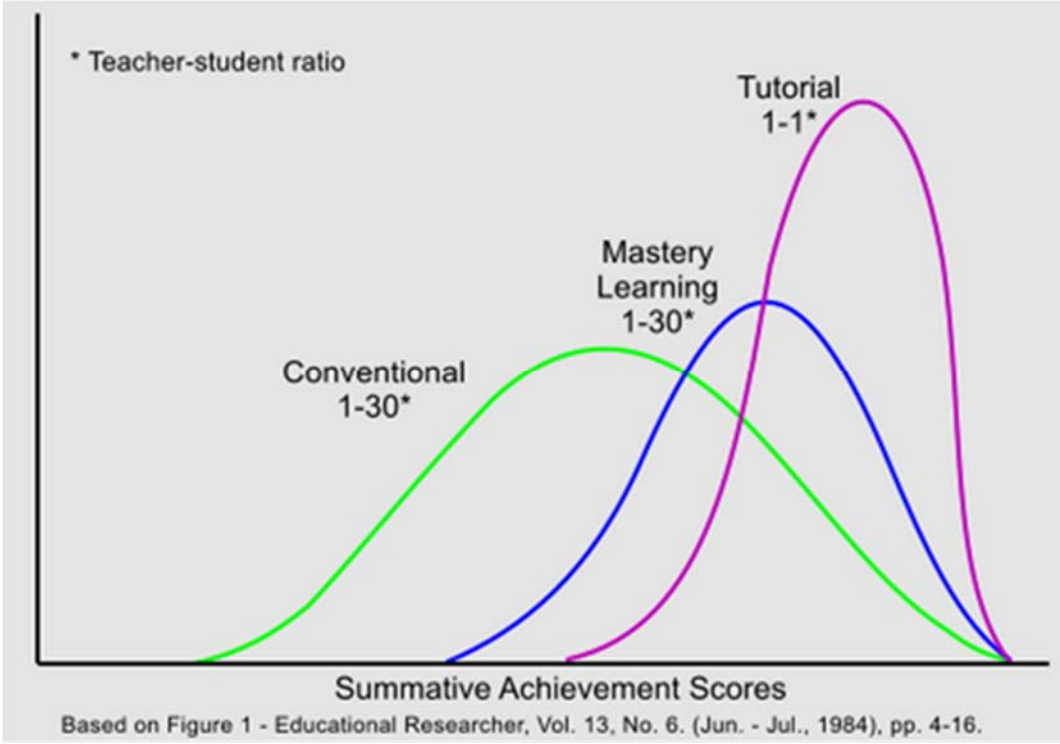
- Midterm grades released on Gradescope
  - Average: 21.81/30, Std Dev: 5.31
  - Regrade requests via Gradescope due Wednesday (5/3)
    - Make sure to submit separate requests for *each* subproblem
  - Make sure you understand your mistakes

Growth vs. Fixed Mindset

- Students can be thought of as having either a "growth" mindset or a "fixed" mindset (based on research by Carol Dweck)
  - "In a **fixed mindset** students believe their basic abilities, their intelligence, their talents, are just fixed traits. They have a certain amount and that's that, and then their goal becomes to look smart all the time and never look dumb."
  - "In a **growth mindset** students understand that their talents and abilities can be developed through effort, good teaching and persistence. They don't necessarily think everyone's the same or anyone can be Einstein, but they believe everyone can get smarter if they work at it."

Bloom's Two Sigma Problem

- 1984 *Educational Researcher* paper
  - http://web.mit.edu/5.95/readings/bloom-two-sigma.pdf
  - Core observation: An "average" student learning in a 1 on 1 format achieves results similar to the top 2% of a lecture based class (two standard deviations above the mean).
  - Very strongly implies that the "fixed" mindset is wrong!



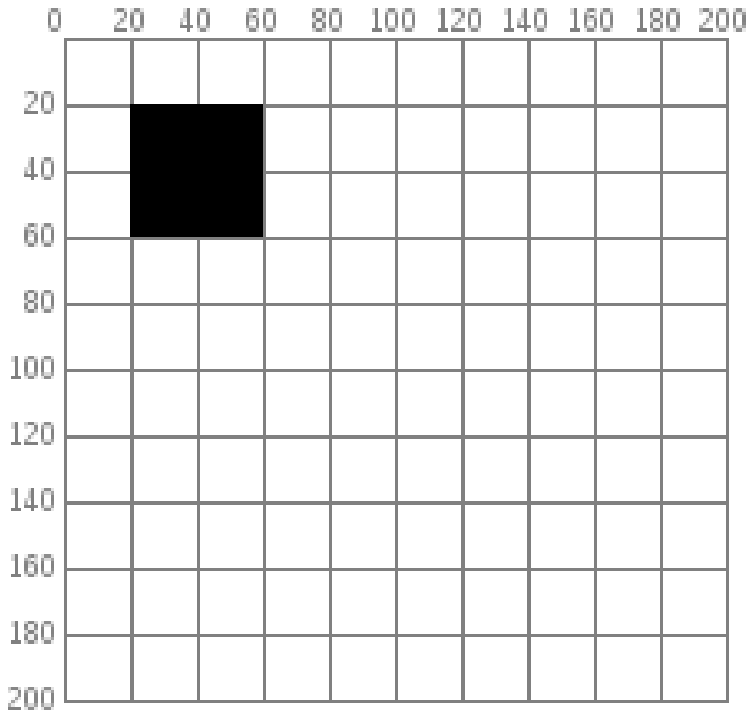
Outline

- **Processing: `translate()` and `rotate()`**
- Recursion
- Solving Problems Using Recursion
- Variable Scope Revisited

Manipulating the Coordinate Grid

- We have always used the default coordinate grid:
  - (0,0) is the upper left corner
  - Increasing x-position moves towards the right
  - Increasing y-position moves towards the bottom

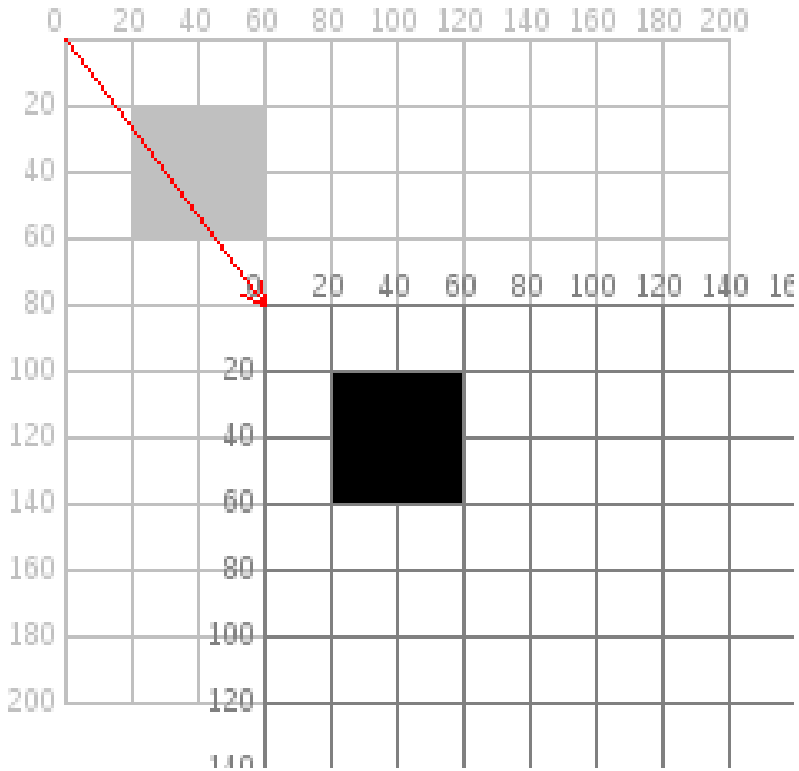
```
rect(20,20,40,40);
```



Manipulating the Coordinate Grid

- `translate(xShift,yShift)` moves the origin
  - (0,0) now refers to the pixel (`xShift`,`yShift`) on the drawing canvas
  - Can use negative coordinates

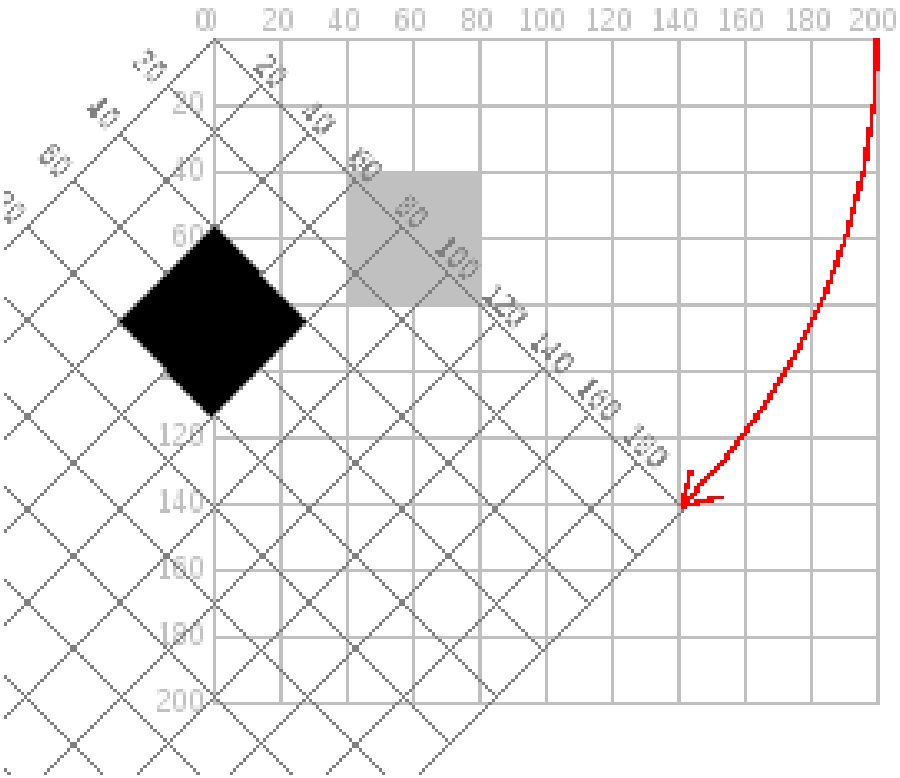
```
translate(60,80);
rect(20,20,40,40);
```



Manipulating the Coordinate Grid

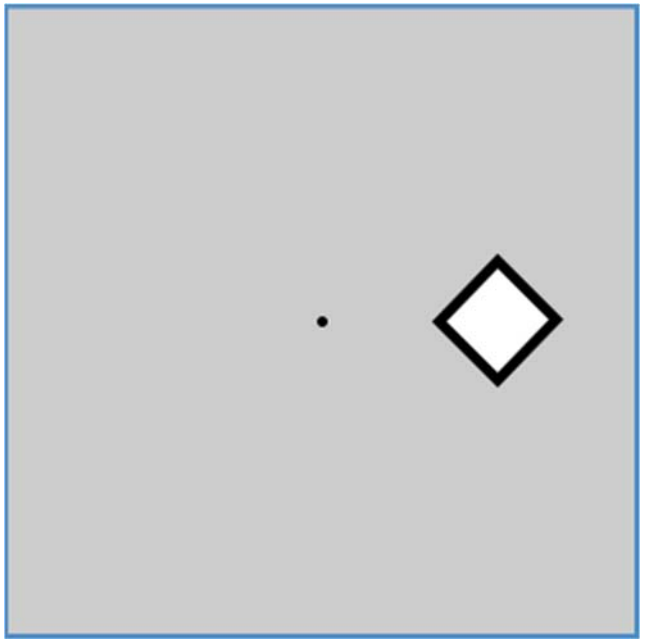
- `rotate(angle)` rotates the grid *clockwise* around the origin
  - Increasing x-position now moves at `angle` from horizontal
  - Increasing y-position now moves at `angle` from vertical

```
rotate(radians(45));
rect(40,40,40,40);
```



Peer Instruction Question

- Which combination of commands, when followed by `rect(20,20,20,20)` produces the picture shown below?
  - Vote at http://PollEv.com/justinh



A. rotate(radians(45));
translate(width/2,height/2);

B. translate(width/2,height/2);
rotate(radians(45));

C. rotate(radians(-45));
translate(width/2,height/2);

D. translate(width/2,height/2);
rotate(radians(-45));

E. We're lost…

Outline

- Processing: `translate()` and `rotate()`
- **Recursion**
- Solving Problems Using Recursion
- Variable Scope Revisited

Recursion

- Recursion is an algorithmic technique where a function, in order to accomplish a task, calls itself with some *part* of the task
  - A function is *recursive* if the body of the function calls the function itself

- We've seen this before!
  - Lightbot recursion
  - `add(x,y)` function from "Beauty in Computer Science"
  - Algorithm for Selection Sort

Building Blocks of Algorithms

- **Sequencing**
  - The application/execution of each step of an algorithm in the order given

```
fill(255);
rectMode(CORNERS);
rect(-r, -r, 0, r);
ellipse(0, -r/2, r, r);
```

- **Selection**
  - Use of conditional to select which instruction to execute next

```
if(mousePressed) {
  fill(0,0,255);
}
```

- **Iteration**
  - Repeat part of algorithm a specified number of times

```
for(int i=20; i<400; i=i+60) {
  line(i,40,i+60,80);
}
```

- **Recursion**
  - Algorithm calls itself to help solve the problem on smaller parts

Why Recurse?

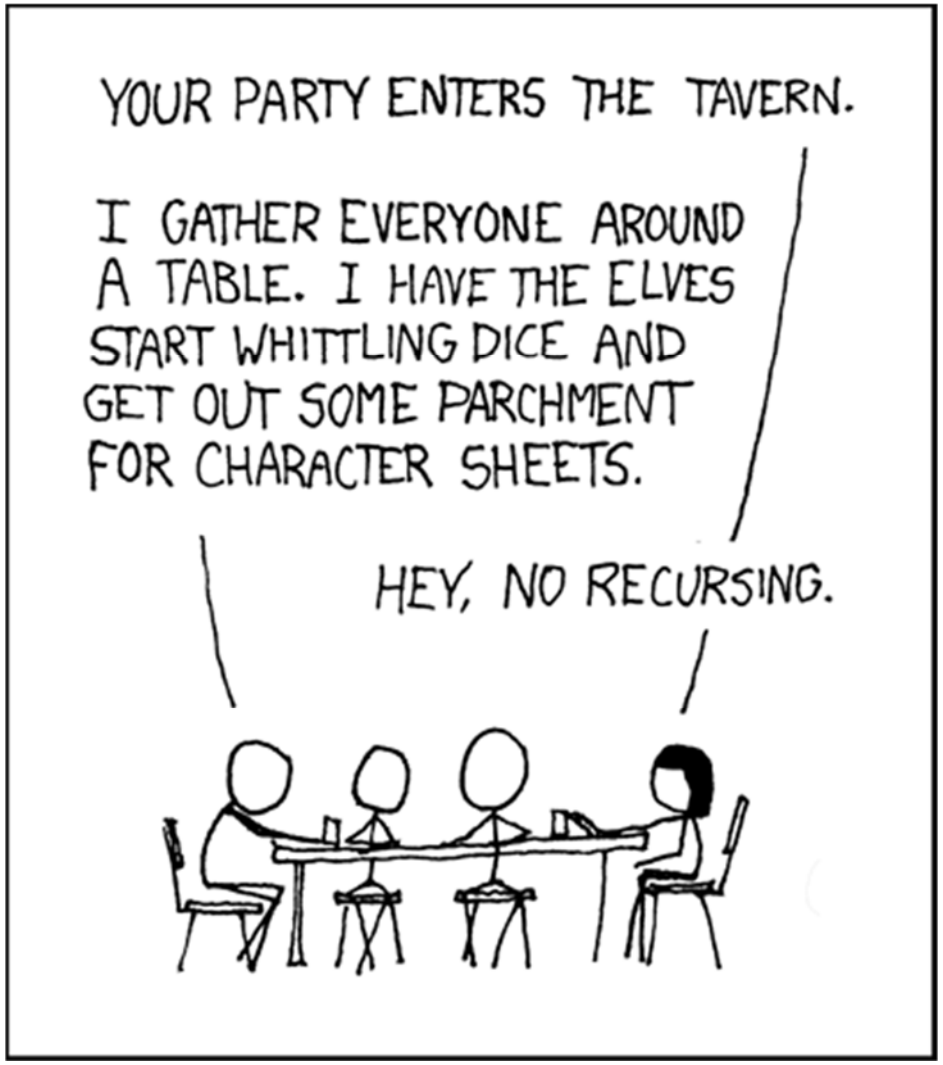
- In its most boring form, recursion is simply an *alternative* to iteration/looping
  - Anything you can do with iteration, you can do with recursion (an vice versa)

- However, it is a profoundly powerful way of thinking!
  - Tremendously useful when the problem is *self-similar*
  - No more powerful than iteration, but often leads to more concise and "better" code

Examples of Recursion

Examples of Recursion

Examples of Recursion



https://xkcd.com/244/

Peer Instruction Question

- What will happen when we try to run the following code in Processing?
  - Vote at: http://PollEv.com/justinh

```
void draw() {
  countdown(3);
  noLoop();
}
void countdown(int n) {
  println(n);
  countdown(n-1);
}
```

A. It prints out 3, 2, 1, 0
B. It runs forever
C. Error occurs *before* execution
D. Error occurs *during* execution
E. We're lost…

Fixing the Countdown

- Without using loops, how would you modify `countdown( )` to stop at 0?
  - A *conditional* is needed to stop the infinite loop!

```
void countdown(int n) {
  println(n);
  countdown(n-1);
}
```

```
void countdown(int n) {
  if(n<0) {
    // do nothing
  } else {
    println(n);
    countdown(n-1);
  }
}
```

Outline

- Processing: `translate()` and `rotate()`
- Recursion
- **Solving Problems Using Recursion**
- Variable Scope Revisited

Recursive Solutions

- Base case(s)
  - When the problem is simple enough to be solved directly
  - Needed to prevent infinite recursion
- Recursive case(s)
  - Function calls itself one or more times on "smaller" problems
    - *Divide* the problem into one or more simpler/smaller parts
    - *Invoke* the function (recursively) on each part
    - *Combine* the solutions of the parts into a solution for the problem
- Depending on the problem, any of these may be trivial or complex

Add

```
int add(int x, int y) {
  if(y==0) {
    return x;
  } else {
    return add(x+1,y-1);
  }
}
```

- Divide: y is reduced by 1
- Invoke: call to add(x+1,y-1)
- Combine: none

- Base: y==0 (nothing left to add)

Add (alternate)

```
int add(int x, int y) {
  if(y==0) {
    return x;
  } else {
    return 1 + add(x,y-1);
  }
}
```

- Divide: y is reduced by 1
- Invoke: call to `add(x,y-1)`
- Combine: add 1 to result

- Base: `y==0` (nothing left to add)

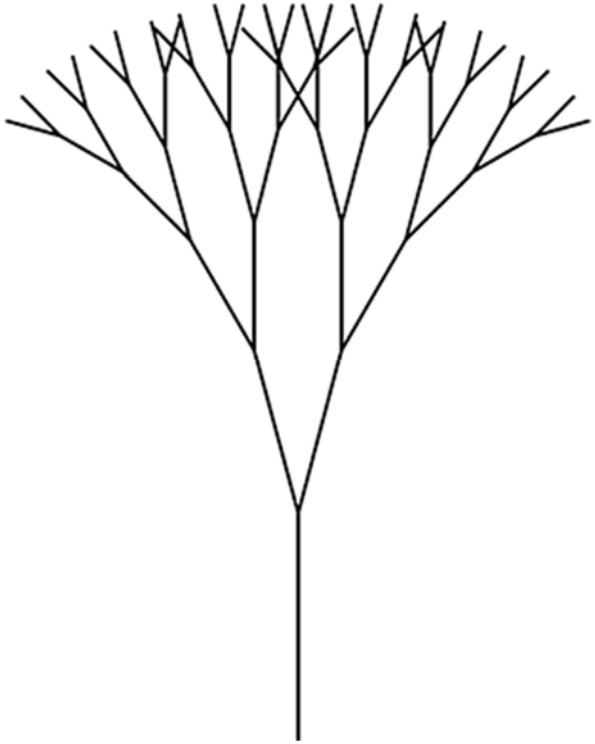
Selection Sort

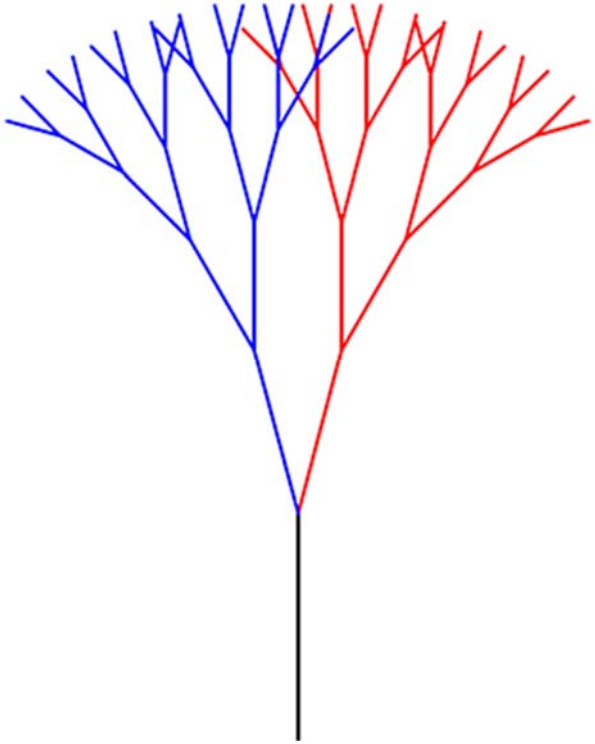
- Algorithm:
  - Find smallest number in an array and move that to the front
  - Repeat this entire procedure, but for the *rest of the array*

To selection sort this:	7	3	1	8	4
Move 1 to the front:	**1**	3	**7**	8	4
Then selection sort this:	1	3	7	8	4

  - Divide: array "size" reduced by 1
  - Invoke: call selection sort on smaller array
  - Combine: smallest number in front of sorted array
  - Base: array of size 1

Drawing a Recursive Tree

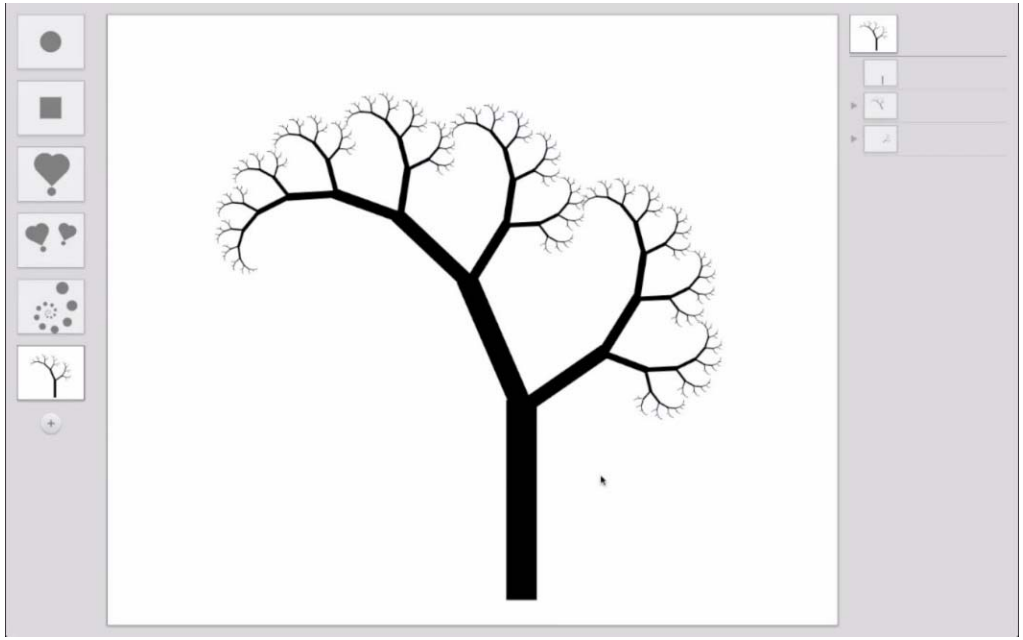




- Divide: draw smaller tree (fewer levels, shorter branches)
- Invoke: draw tree as "right branch" and "left branch"
- Combine: draw branch rotated from end of "trunk"
- Base: "leaf branches" at end

Recursive Drawing (video)

- "Recursive Drawing is an exploration of user interface ideas towards the development of a spatially-oriented programming environment."
  - Create drawings without any lines of code!
  - Created by Toby Schachman
  - http://recursivedrawing.com



Summary

- A recursive function calls itself to solve a problem
  - Always have a **base case** and **recursive case**

- Recursion adds no "power" to programming
  - Anything that can be done with iteration can be done with recursion and vice versa
  - But it makes some things much easier and more elegant
    - Particularly problems with "self-similarity"