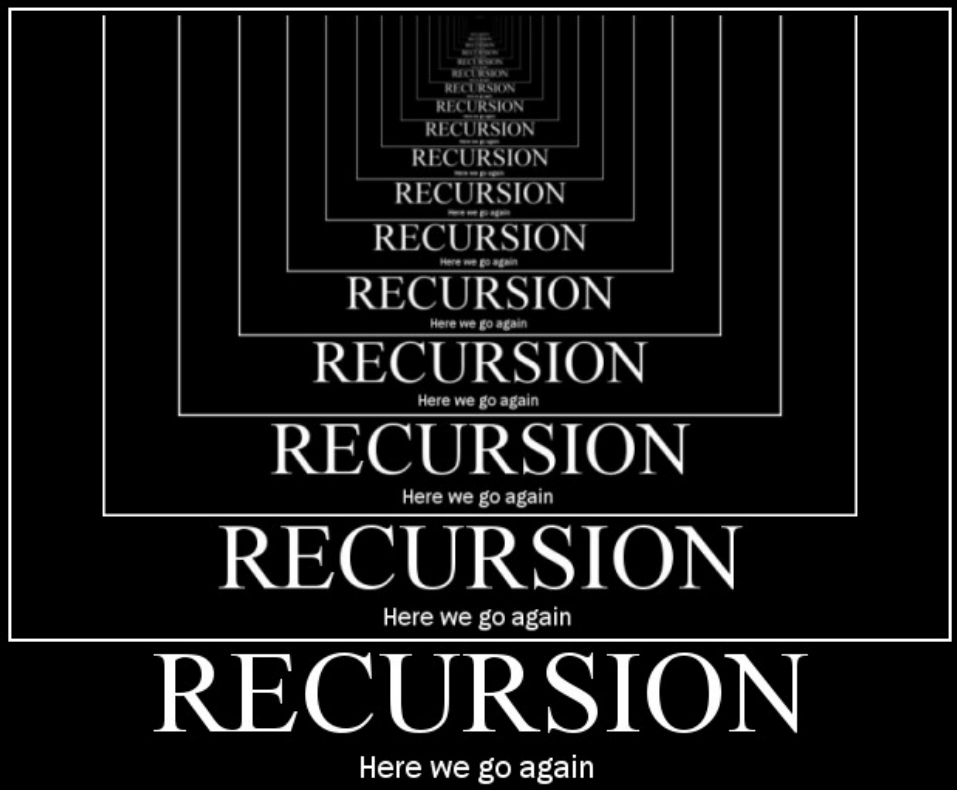
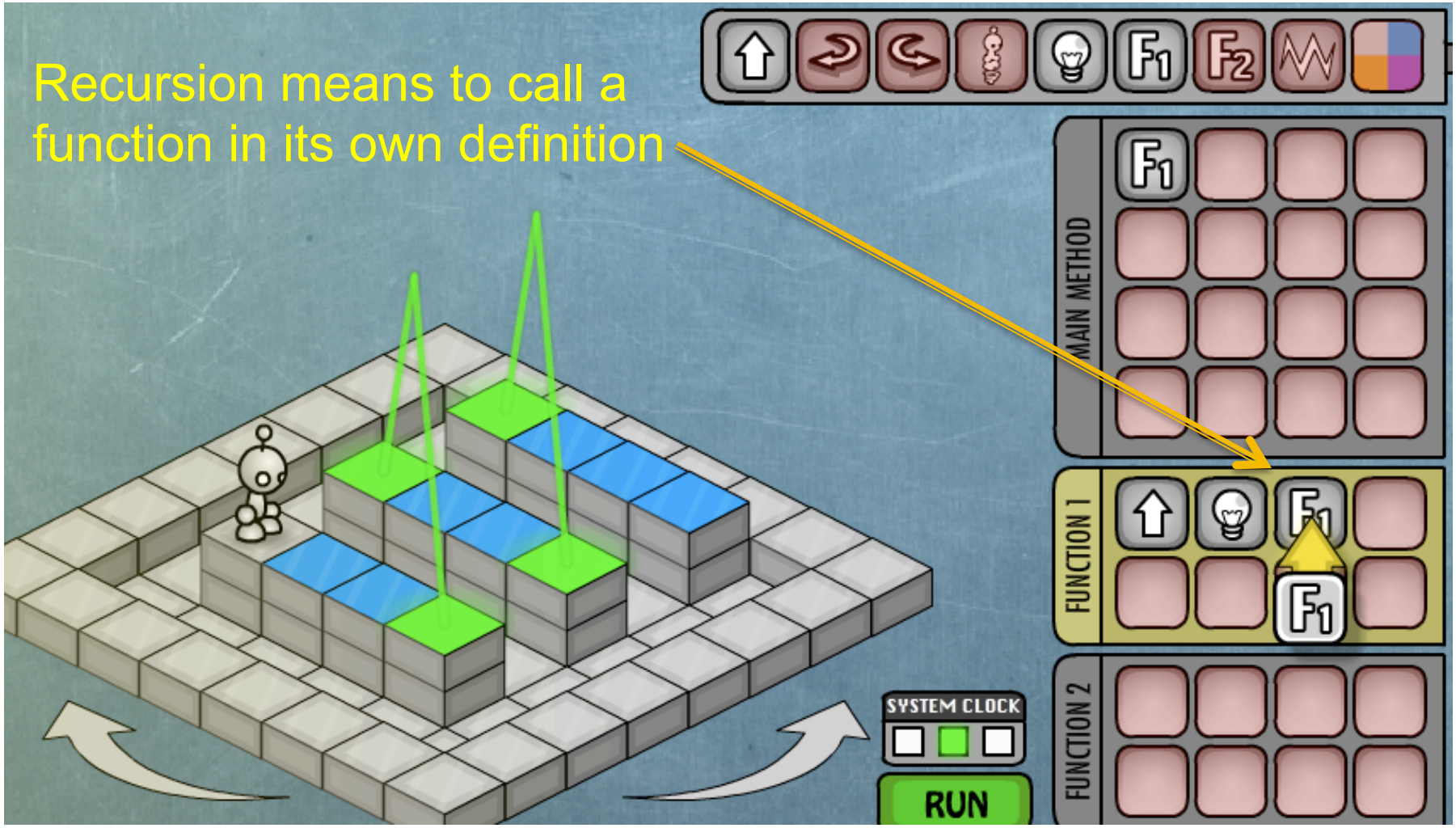
Use what you've got

Recursion



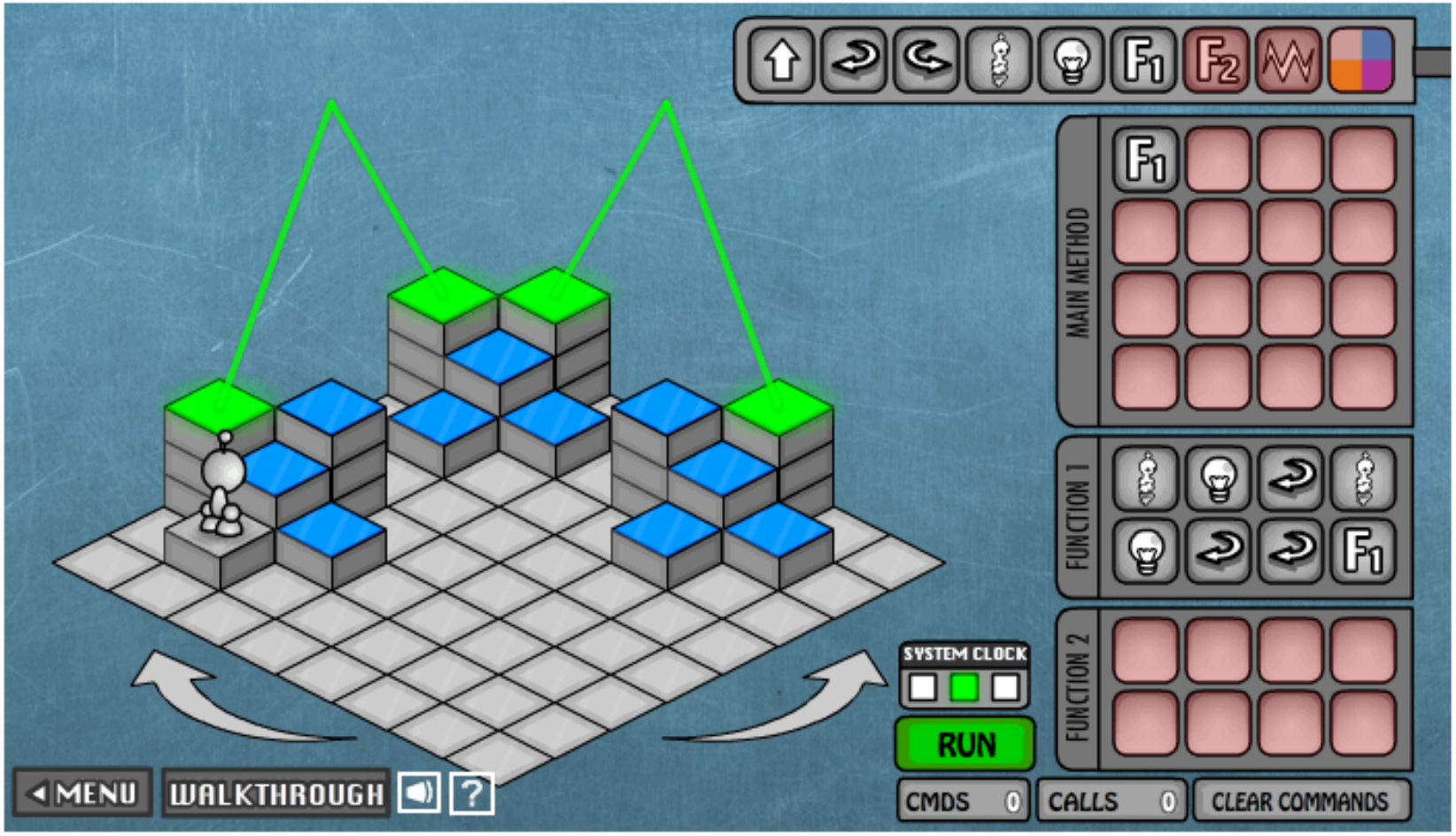
Lawrence Snyder
University of Washington

Recall Recursion In Lightbot 2.0



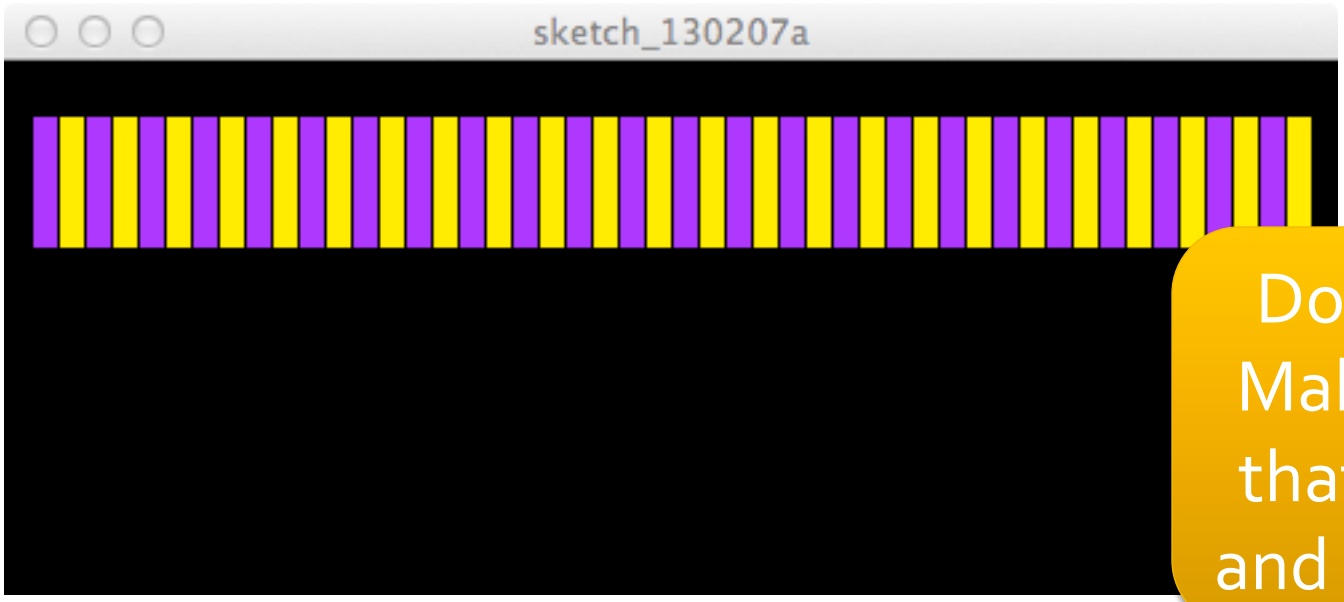
Recursion

- If the "concept applies," use it



Often, Recursive Is Easiest Solution

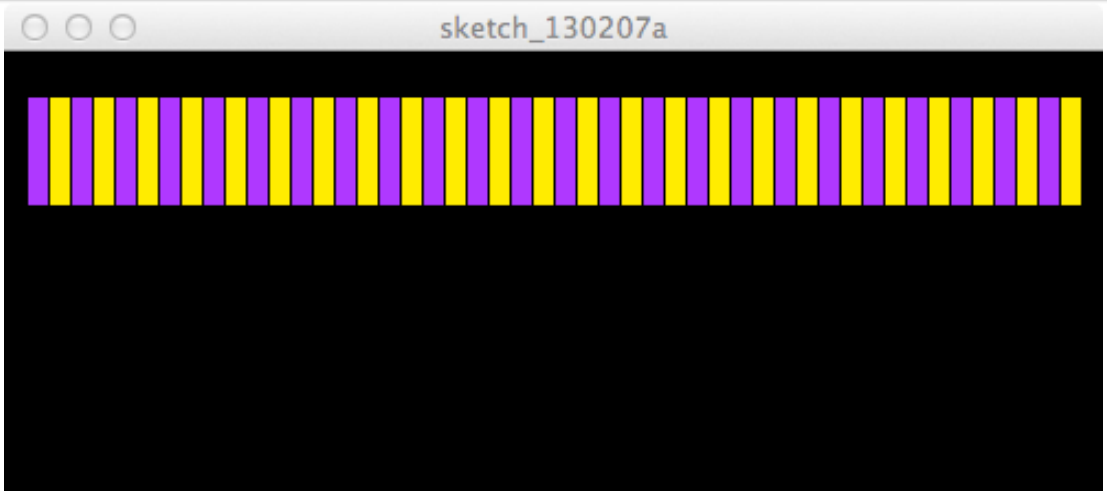
- Specification: Draw pairs of purple & gold bars across the top, leaving 10 px at each end
  - How large is canvas?
  - How many bars are needed?
  - What color to start/end with?



Don't Know? Make solution that is flexible and adjust later

Recursion: Ask "Space?" If so, do it

```
void setup( ) {
  size(500,500);
  background(0);
}
void draw( ) {
  huskyPairs(10, 50);
}
void huskyPairs(int xpos, int ypos) {
  if (xpos + 20 <= width - 10) {
    fill(157, 0, 255);
    rect(xpos, ypos, 10, 50);
    fill(255, 235, 0);
    rect(xpos+10, ypos, 10, 50);
    huskyPairs(xpos + 20, ypos);
  }
}
```

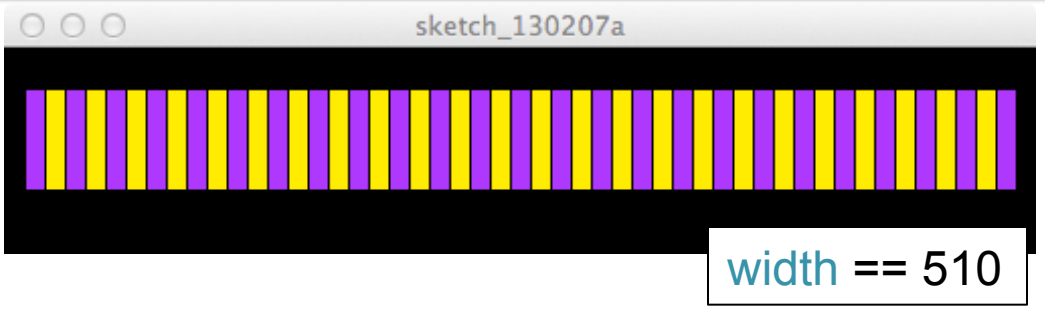


Is There Space For One More Pair Before End?

Try Again In Case There's Space

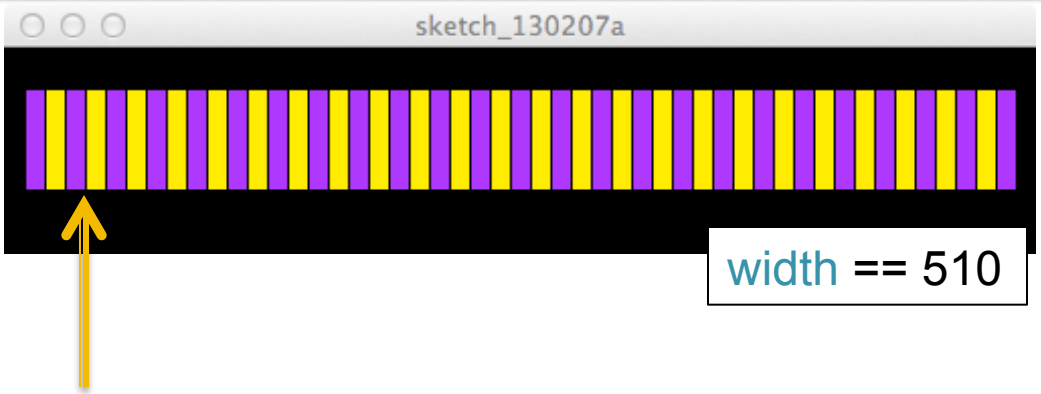
Another: Draw 1 Bar, Flip Colors

```
void setup( ) {
  size(500,500);
  background(0);
}
void draw( ) {
  husky(10, 50, true);
}
void husky(int xpos, int ypos, Boolean pORg) {
  if (xpos + 10 <= width - 10) {
    if (pORg) {
      fill(157, 0, 255);
    } else {
      fill(255, 235, 0);
    }
    rect(xpos, ypos, 10, 50);
    husky(xpos + 10, ypos, !pORg);
  }
}
```



Keeping Track of Calls, Params

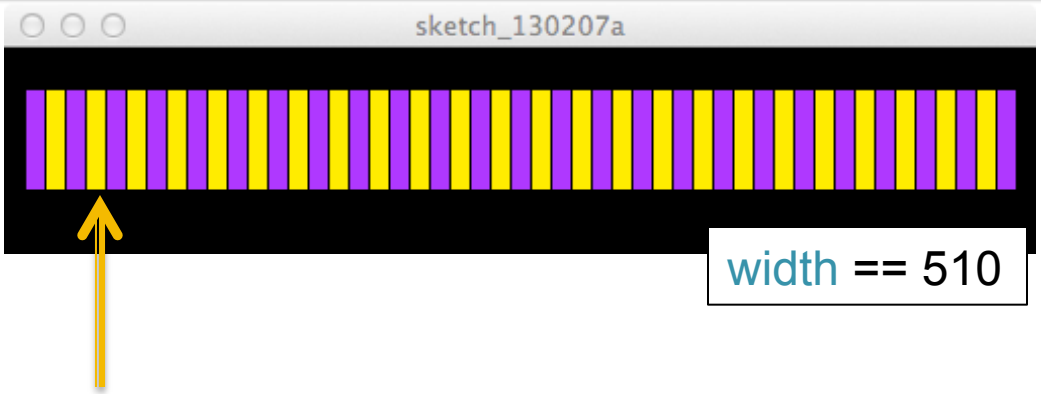
```
void setup( ) {
  size(500,500);
  background(0);
}
void draw( ) {
  husky(10, 50, true);
}
void husky(int xpos, int ypos, Boolean pORg) {
  if (xpos + 10 <= width - 10) {
    if (pORg) {
      fill(157, 0, 255);
    } else {
      fill(255, 235, 0);
    }
    rect(xpos, ypos, 10, 50);
    husky(xpos + 10, ypos, !pORg);
  }
}
```



H: 40,50,F
H: 30,50,T
H: 20,50,F
H: 10,50,T

Keeping Track of Calls, Params

```
void setup( ) {
  size(500,500);
  background(0);
}
void draw( ) {
  husky(10, 50, true);
}
void husky(int xpos, int ypos, Boolean pORg) {
  if (xpos + 10 <= width - 10) {
    if (pORg) {
      fill(157, 0, 255);
    } else {
      fill(255, 235, 0);
    }
    rect(xpos, ypos, 10, 50);
    husky(xpos + 10, ypos, !pORg);
  }
}
```



H: 50,50,T
H: 40,50,F
H: 30,50,T
H: 20,50,F
H: 10,50,T

One More Example: Factorial

- Math people say n! = n * (n-1)*(n-2)* … * 2 * 1
- CS people say n! = if n == 1, then 1, else n*(n-1)!

```
int fact( int n) {
  int soFar = 1;
  for(int i = 2; i <= n; i = i +1) {
  soFar= soFar* i;
  }
}
```

```
int fact( int n ) {
    if (n <= 1)  {
        return 1;
    } else {
        return n*fact(n-1);
    }
```

Most Recursions Have 2 Cases

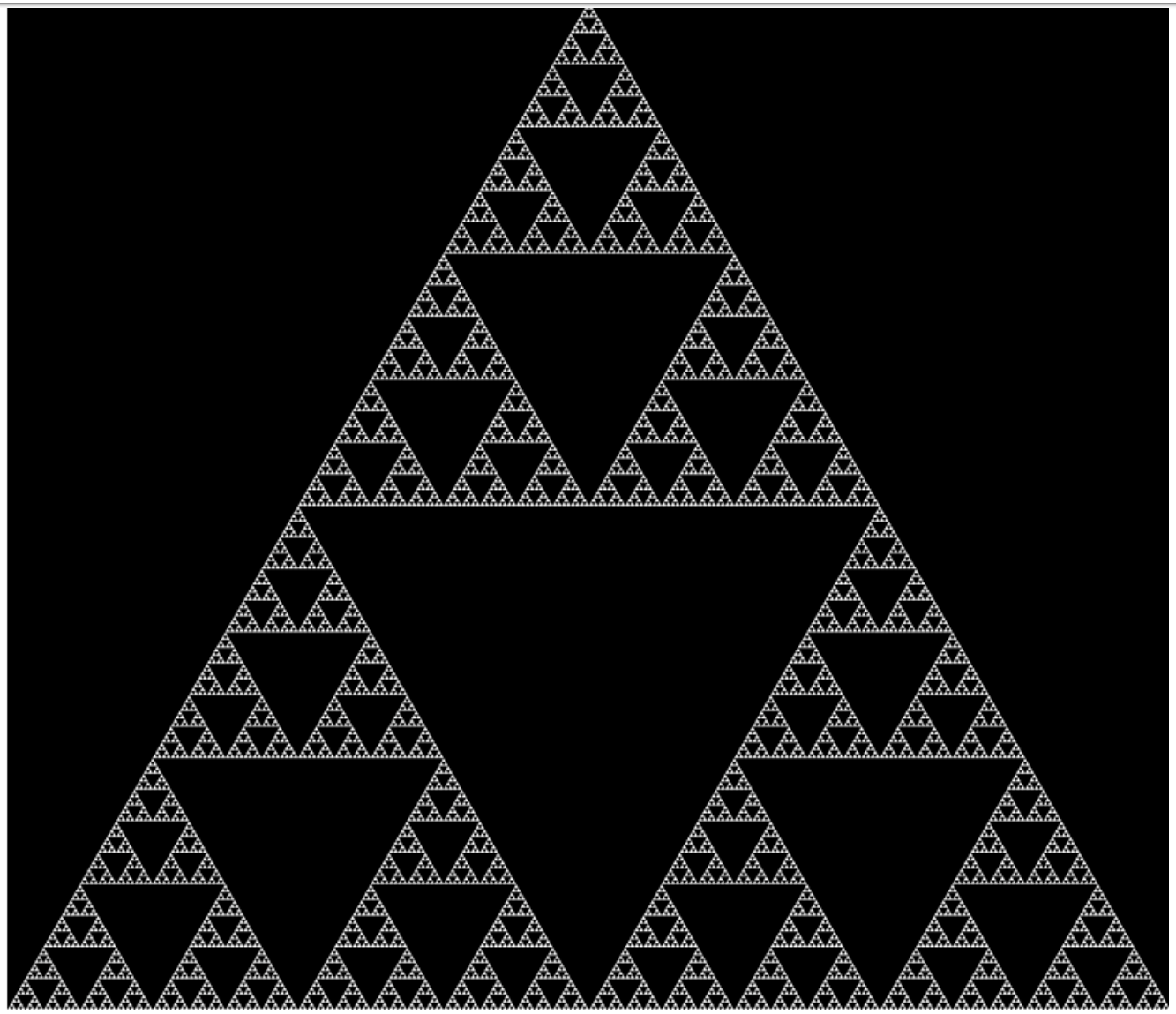
- Generally, in recursion, the program will show two cases: Base and Recursive … you need both

```
int fact( int n ) {
    if (n <= 1)  {
        return 1;
    } else {
        return n*fact(n-1);
}
```

Basis Case – It Stops The Recursion

Recursive Case – It Keeps The Recursion Going

Check it Out: Sierpinski Triangle



Sierpinski Triangle

- What is it?
- Abstracting, we have
  - "A Sierpinski Triangle is an equilateral triangle"
  - "A Sierpinski Triangle can also be three copies of a Sierpinski Triangle, touching at their corners"

Sierpinski Triangle

- What is it?
- Abstracting, we have
  - "A Sierpinski Triangle is an equilateral triangle"
  - "A Sierpinski Triangle can also be three copies of the next smaller Sierpinski Triangle, touching at their corners"

  - What's the base case? What's the recursive case?

Sierpinski Triangle Code (tiny)

```
// Sierpinski.pde by Martin Prout
float T_HEIGHT = sqrt(3)/2;
float TOP_Y = 1/sqrt(3);
float BOT_Y = sqrt(3)/6;
float triangleSize = 800;

void setup(){
  size(int(triangleSize),int(T_HEIGHT*triangleSize));
  smooth();
  fill(255);
  background(0);
  noStroke();
  drawSierpinski(width/2, height * (TOP_Y/T_HEIGHT), triangleSize);
}

void drawSierpinski(float cx, float cy, float sz){
  if (sz < 5){  // Limit no of recursions on size
    drawTriangle(cx, cy, sz); // Only draw terminals
    noLoop();
  }
  else{
    float cx0 = cx;
    float cy0 = cy - BOT_Y * sz;
    float cx1 = cx - sz/4;
    float cy1 = cy + (BOT_Y/2) * sz;
    float cx2 = cx + sz/4;
    float cy2 = cy + (BOT_Y/2) * sz;
    drawSierpinski(cx0, cy0, sz/2);
    drawSierpinski(cx1, cy1, sz/2);
    drawSierpinski(cx2, cy2, sz/2);
  }
}

void drawTriangle(float cx, float cy, float sz){
  float cx0 = cx;
  float cy0 = cy - TOP_Y * sz;
  float cx1 = cx - sz/2;
  float cy1 = cy + BOT_Y * sz;
  float cx2 = cx + sz/2;
  float cy2 = cy + BOT_Y * sz;
  triangle(cx0, cy0, cx1, cy1, cx2, cy2);
}
```

Sierpinski Triangle Code

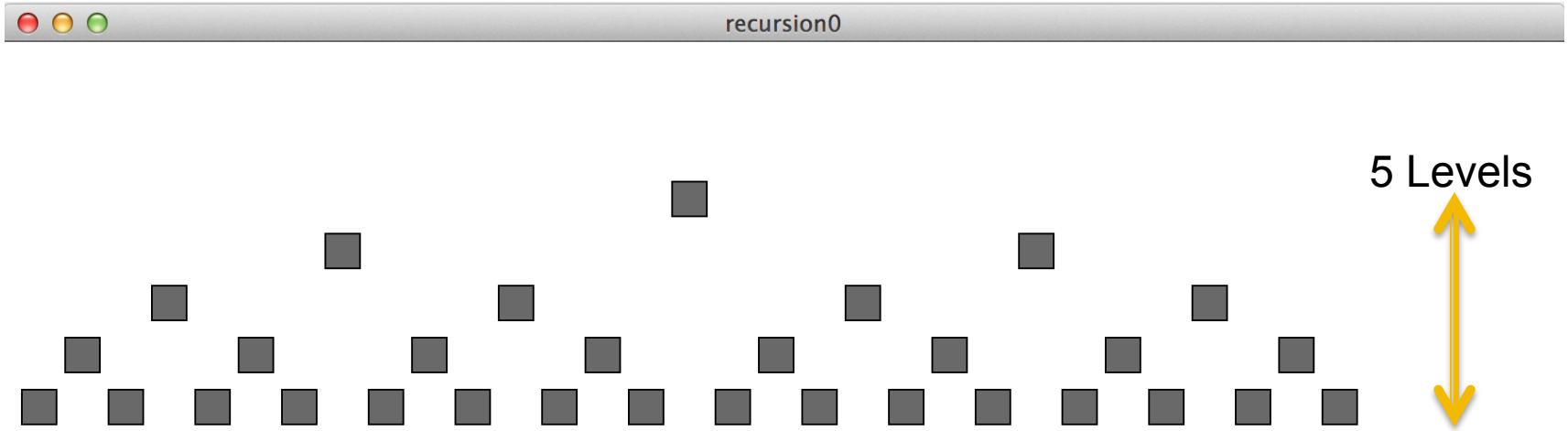
```
1  // Sierpinski.pde by Martin Prout
2  float T_HEIGHT = sqrt(3)/2;
```

```
7  void setup(){
8    size(int(triangleSize),int(T_HEIGHT*triangleSize));
9    smooth();
10   fill(255);
11   background(0);
12   noStroke();
13   drawSierpinski(width/2, height * (TOP_Y/T_HEIGHT), triangleSize);
14 }
15
16 void drawSierpinski(float cx, float cy, float sz){
17   if (sz < 5){  // Limit no of recursions on size
18     drawTriangle(cx, cy, sz); // Only draw terminals
19     noLoop();
20   }
21   else{
22     float cx0 = cx;
23     float cy0 = cy - BOT_Y * sz;
24     float cx1 = cx - sz/4;
25     float cy1 = cy + (BOT_Y/2) * sz;
26     float cx2 = cx + sz/4;
27     float cy2 = cy + (BOT_Y/2) * sz;
28     drawSierpinski(cx0, cy0, sz/2);
29     drawSierpinski(cx1, cy1, sz/2);
30     drawSierpinski(cx2, cy2, sz/2);
31   }
32 }
```

```
41   triangle(cx0, cy0, cx1, cy1, cx2, cy2);
42 }
```

Define A Tree

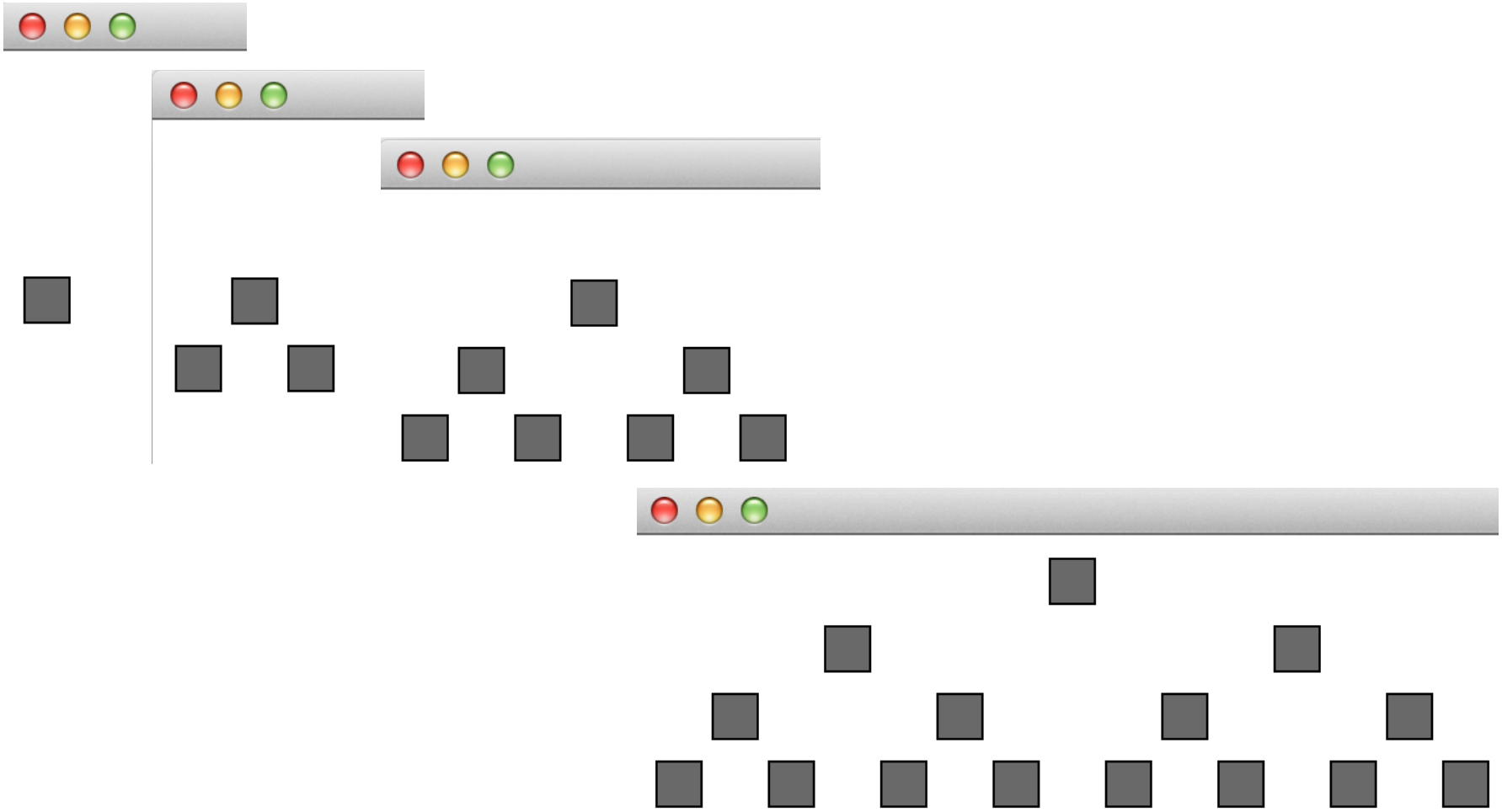
- Suppose we define a tree as root, or a root with a left subtree (that's also a tree) and a right subtree (that's also a tree) …



- *It's CS so trees are drawn in odd ways!*

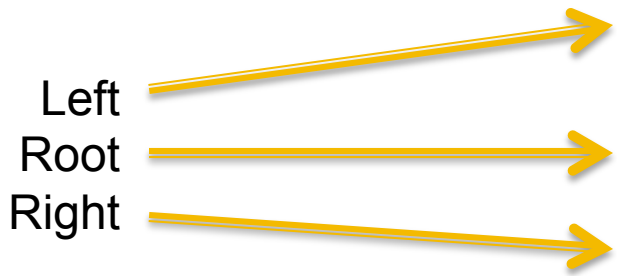
The Definition is Recursive

- What's the basis case? What's the recursion?



A (Slightly) More Complex Recur

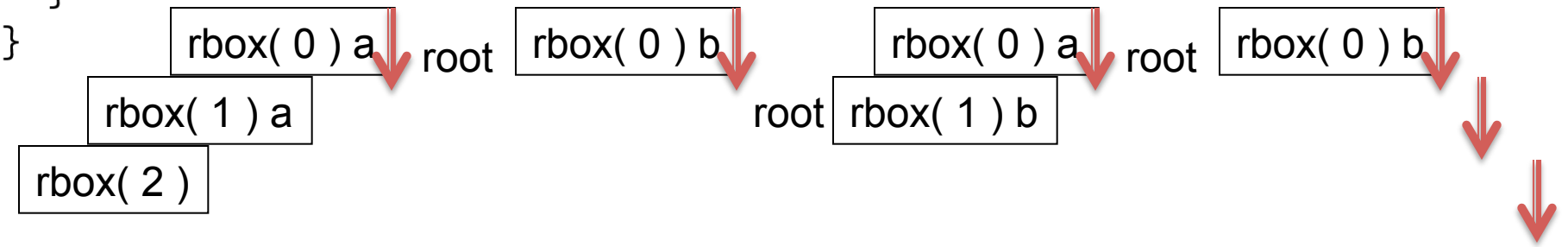
- Like Sierpinski, rbox( ) calls itself more than once
- It defines a tree
  - 1st call is left side
  - 2nd call is right
  - Between is root



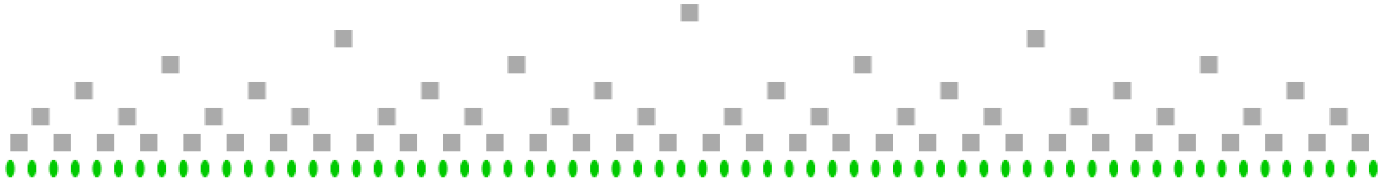

```
int xdist = 10;
void setup( ) {
  size(900,500);
  background(255);
  noLoop( );
}
void draw( ) {
  rbox(4);
}
void rbox(int level) {
  if (level > 0) {
    level = level - 1;
    rbox(level);
    fill(105);
    rect(xdist, 100-level*30, 20, 20);
    xdist = xdist + 25;
    rbox(level);
  }
}
```

Watch The Calls For Level == 2

```
void draw( ) {
  rbox(2);
}
void rbox(int level) {
  if (level > 0) {
    level = level - 1;
    rbox(level);
    fill(105);
    rect(xdist, 200-level*30, 20, 20);
    xdist = xdist + 25;
    rbox(level);
  }
}
```



See The Progression of Calls …

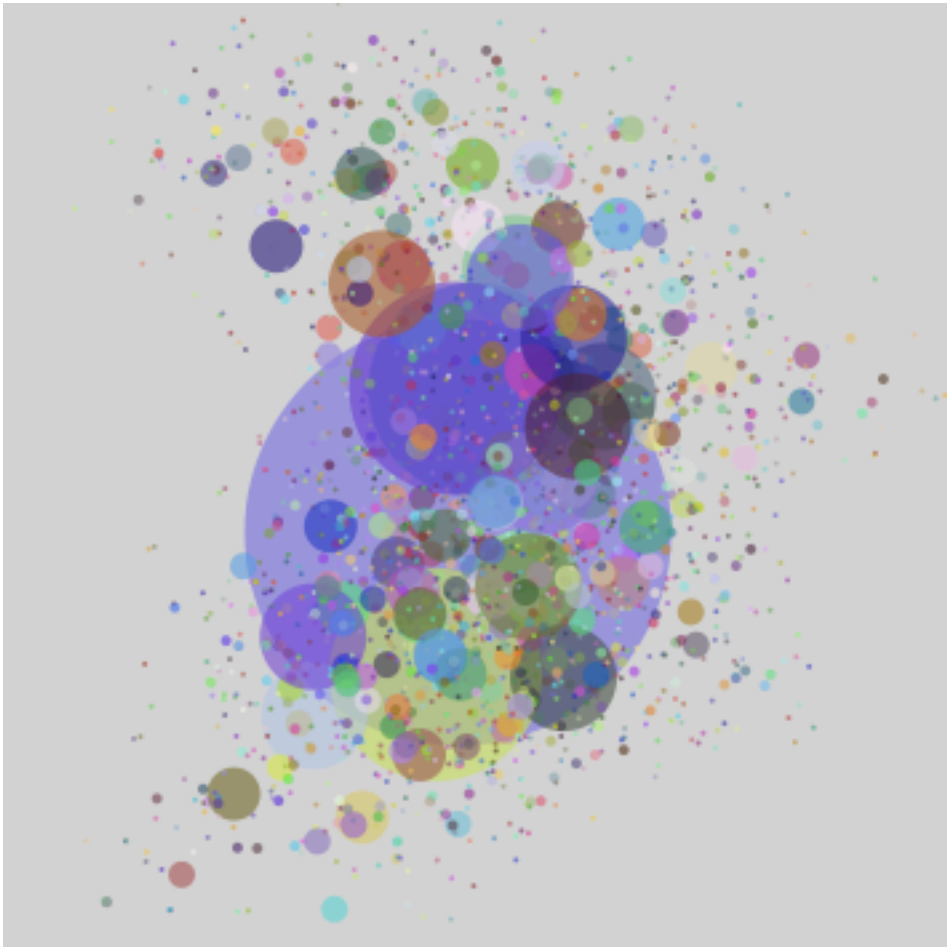


```
void box(int level) {
  if (level > 0) {
    level = level - 1;
    box(level);
    fill(170);
    rect(xdist, 200-level*30, 20, 20);
    xdist = xdist + 25;
    box(level);
  } else {
    fill(0,255,0);
    ellipse(xdist+10, 230, 10, 20);
  }
}
```

Each level 0 call draws a leaf

Wrap Up

- Recursion often simplifies programming
- It's only a big deal to CS people, and for them only because it is so "elegant" (?)
- See Processing Ref for this cute program



All Circles From 1 Call Are 1 Color