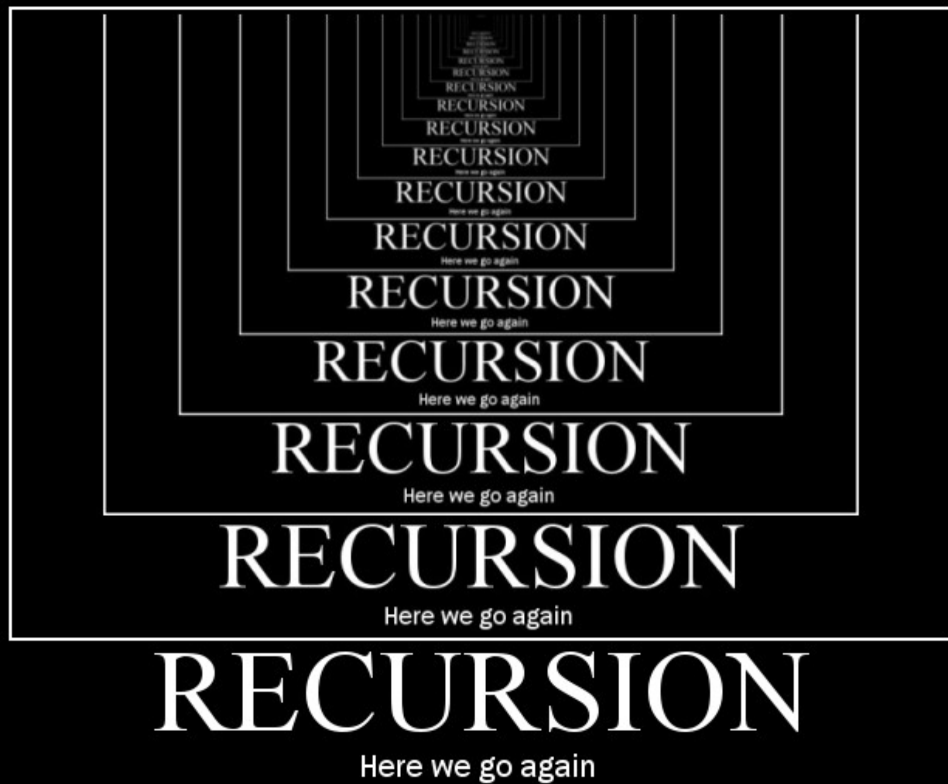
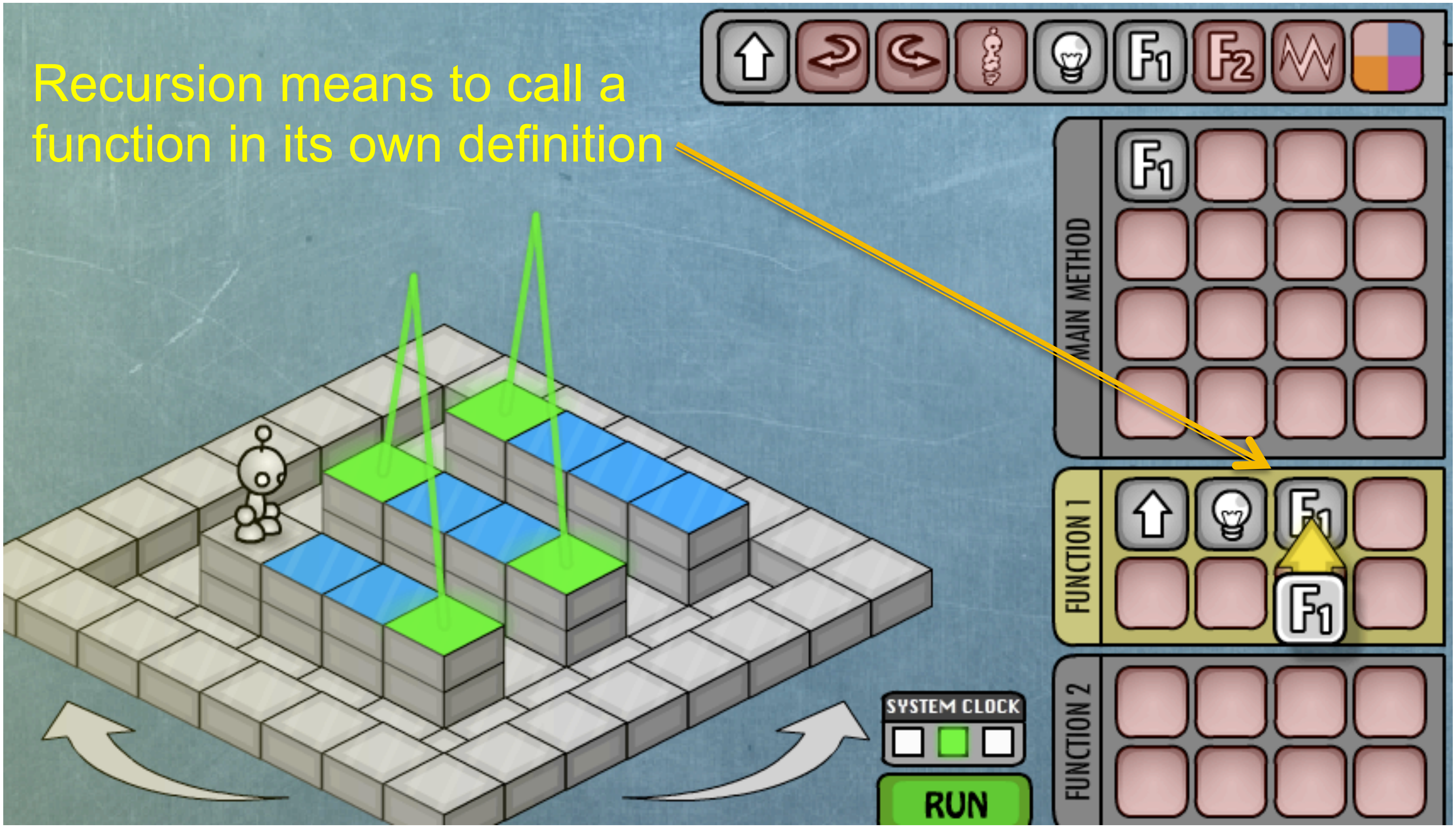
Use what you've got

Recursion



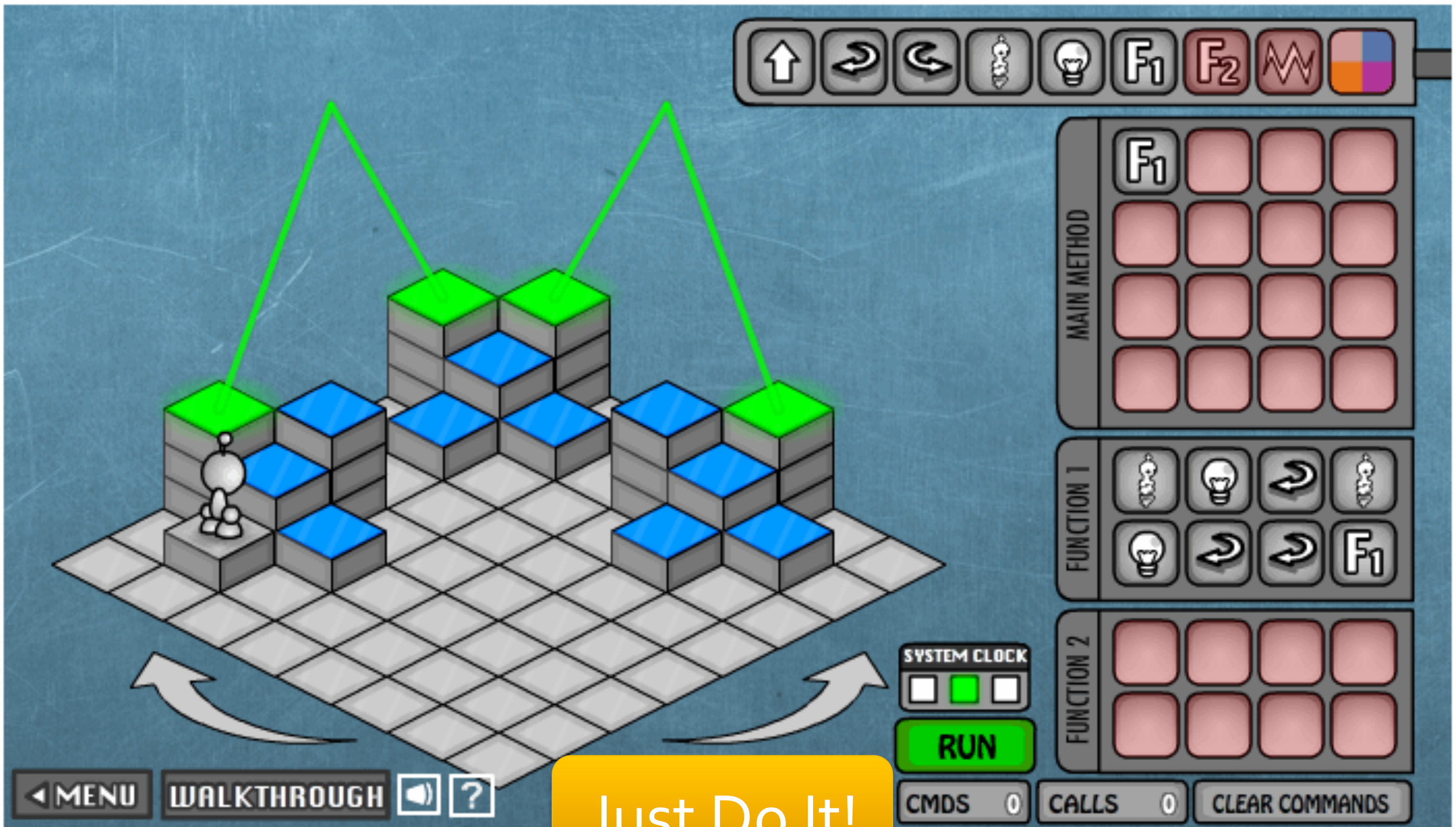
Lawrence Snyder
University of Washington

Recall Recursion In Lightbot 2.0



Recursion

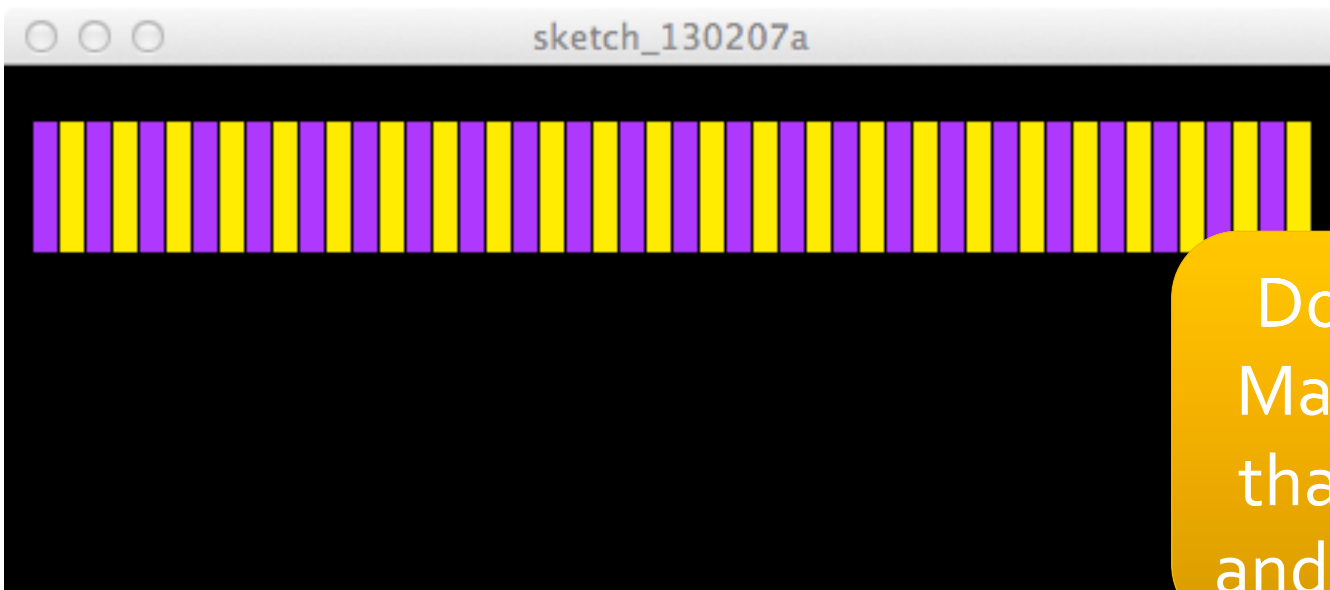
- If the "concept applies," use it



Just Do It!

Often, Recursive Is Easiest Solution

- Specification: Draw alternating 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?

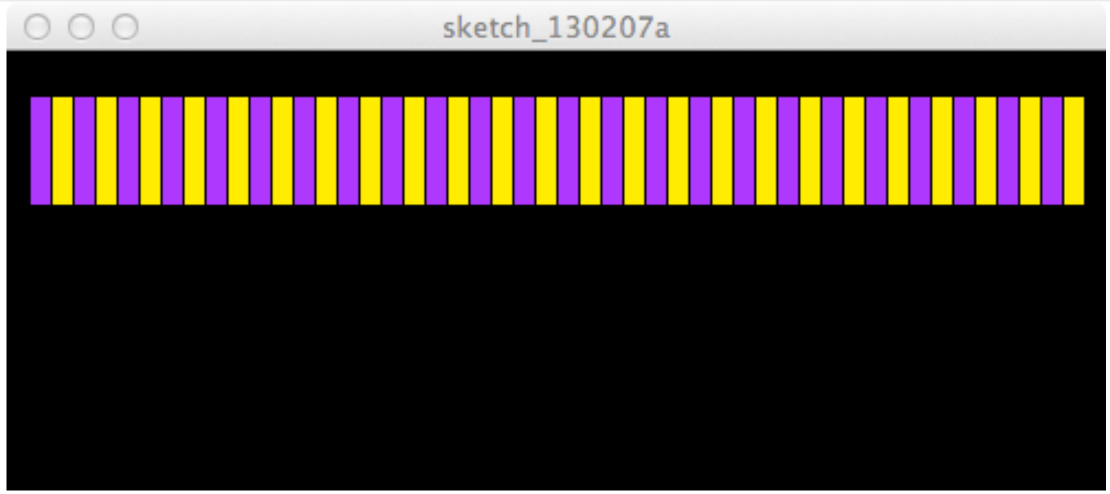


Don't Know? Make solution that is flexible and adjust later

Recursion: Draw A Pair, Ask "More?"

```
void setup( ) {
   size(500,500);
   background(0);
}

void draw( ) {
   husky(10, 100);
}
```

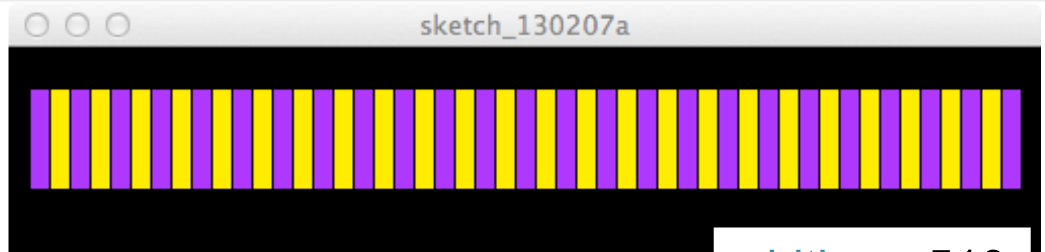



```
void husky(int xpos, int ypos) {
   fill(157,0,255);
   rect(xpos, ypos, 10,50);
   fill(255,235,0);
   rect(xpos+10, ypos, 10, 50);
   if (xpos+20 <= width-30) {
     husky(xpos+20, ypos);
   }
}
```

- Draw A Purple Bar
- Draw A Gold Bar
- Is There Space For One More Pair Before End?
- Yup, Do It Again

Just Do It!

Another: Draw 1 Bar, Flip Colors



width == 510

```
void draw( ) {
  husky(10, 20, true);
}

void husky(int xpos, int ypos, boolean pORg) {
  if (pORg) {
    fill(157,0,255); // Pick Purple
  } else {
    fill(255,235,0); // Pick Gold
  }
  rect(xpos, ypos, 10,50);      // Draw bar
  if (xpos+10 <= width-20) {   // Keep Going?
    husky(xpos+10, ypos, !pORg); // Yes
  }
}
```

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

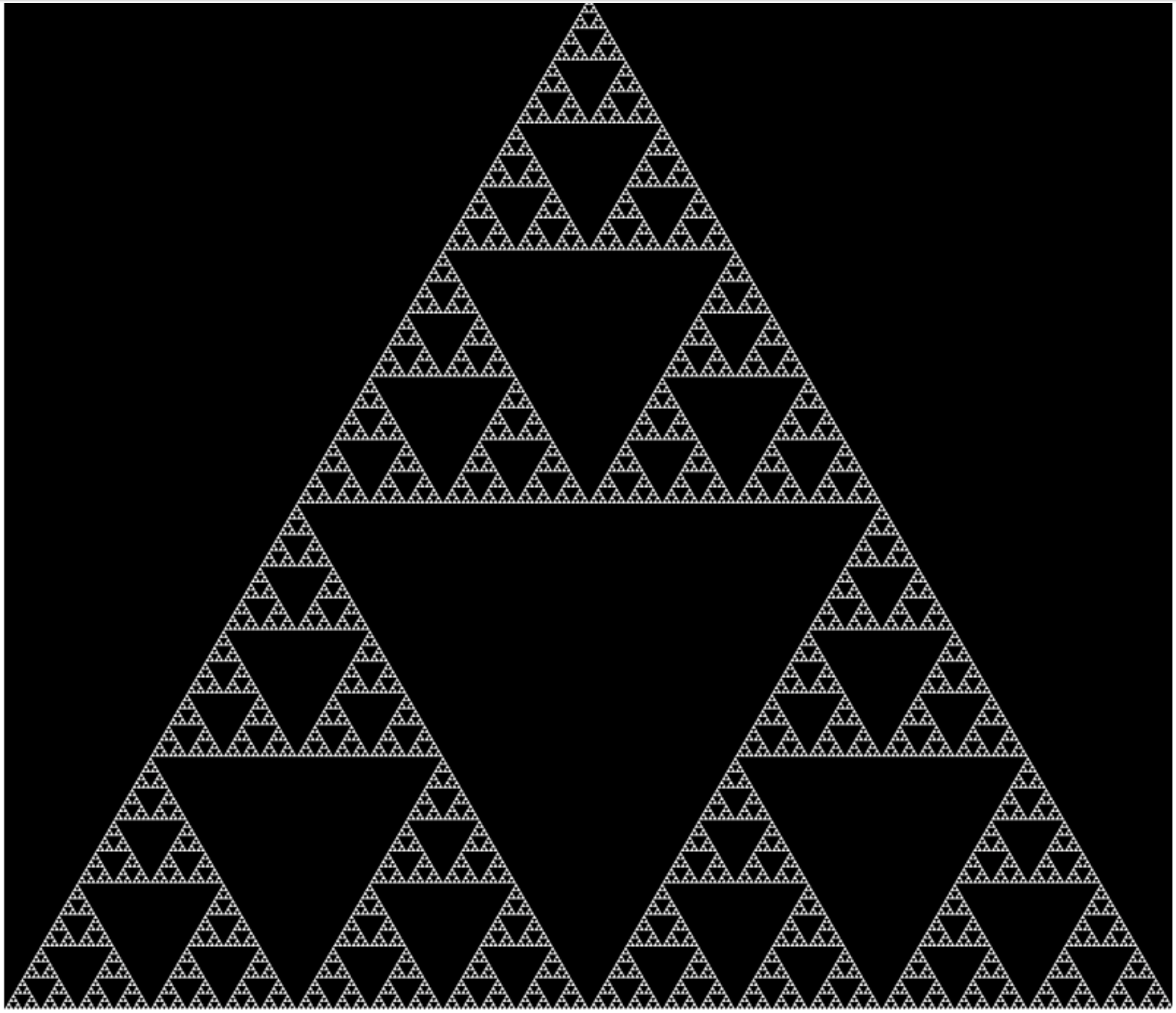
- Generally, in recursion, the program will show two cases: Base and Recursive … you need both

```
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 a 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

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

```
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);
  }
}
```

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

Recursion Often Uses Less Thinking

- Often we can solve a problem "top down"
- Fibonacci numbers –

  1, 1, 2, 3, 5, 8, 13, 21, 34, …

  i^{th} item is i-1^{st} + i-2^{nd} except the first two, both 1

- Translate definition directly:

$$fib(n) = \begin{cases} 1 & \text{if } n<2 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

- It works like all functions work

The Fib Code In Processing …

- If anyone actually cared about Fibonacci numbers, they could be computed …

```
int fib(int n) {
  if (n < 2 ) {
    return 1;
  }else{
    return fib(n-1) + fib(n-2);
  }
}
```

Leave The Thinking To The Agent …

$$fib(n) = \begin{cases} 1 & \text{if } n<2 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

fib(4) = fib(3) + fib(2)

- fib(3) = fib(2) + fib(1)
  - fib(2) = fib(1) + fib(0) = 1 + 1 = 2

  = 2 + 1 = 3

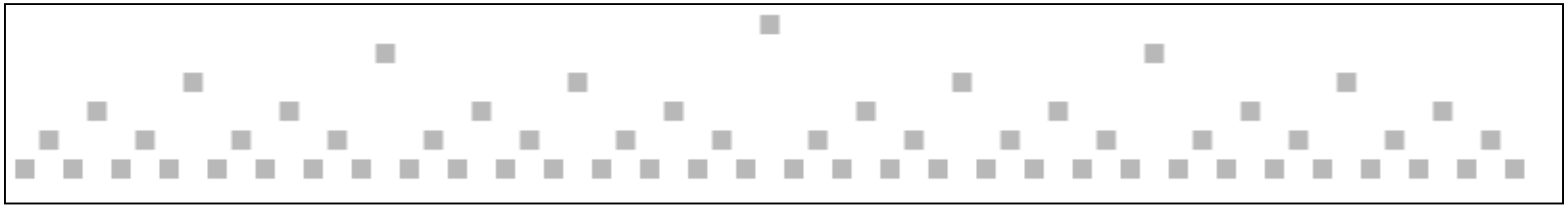
So, fib(4) = 3 + fib(2)

- fib(2) = fib(1) + fib(0) = 1 + 1 = 2

So, fib(4) = 3 + 2 = 5

Programmers don't need to worry about the details if the definition is right and the termination is right; the computer does the rest

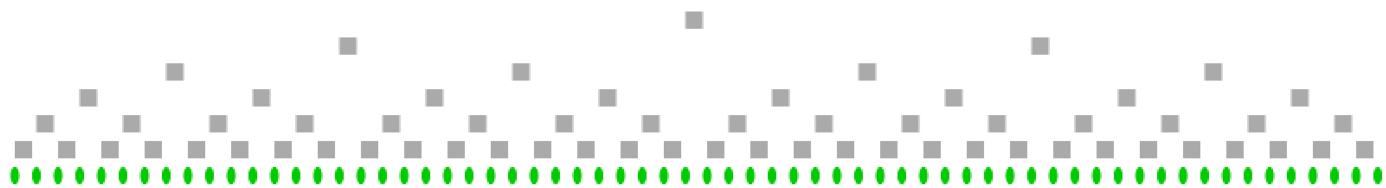
See The Progression of Calls …



```
  box(6);
}
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);
  }
}
```

The boxes are drawn in order, left to right

See The Progression of Calls …

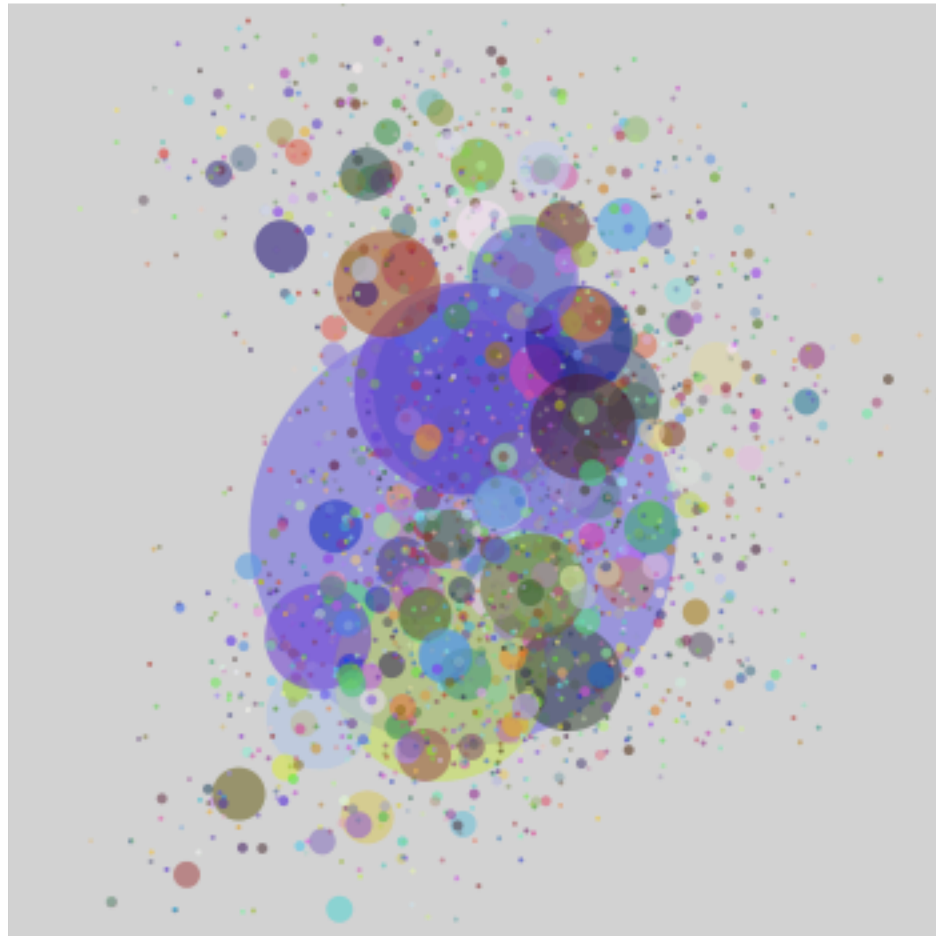


```
void box(int level) {
  if (level > 0) {
    level = level - 1;
    box(level);
    fill(170);
    rect(xdist, 200-level*30, 20, 20);
    xdist = xdist + 25;
    box(level);
  } else {
    fill(0,255,0);
    ellipse(xdist+10, 230, 10, 20);
  }
}
```

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