What do you think of Lightbot?
As seasoned Lightbot hackers...

- What are you doing in Lightbot?
- Commanding a robot through a “blocks world”
- Programming is **commanding** an agent
Other aspects of “commanding”

- The agent is usually a computer, but can be a person or device (like a robot)
- The agent follows the commands, the instructions, without making mistakes, without stopping, doing only what is asked
- The program implements your intentions – it does what you tell it to do
- You are trying to get the robot to the blue tiles and flip them to yellow
- Instructions are *given* in sequence

- They are *executed* in sequence – essential
  - Instructions...
    - From a limited repertoire
    - All are within agent's ability (can't jump across squares, teleport)
    - Executed one at a time
  - A “program counter” keeps track of agent's progress
Instructions formed from simpler actions

- Check out this screenshot
- It is partway through an instruction ... it's beacon is lit, but the tile is still blue
- To the programmer the instruction is one step (monolithic)
- To the agent each instruction is a series of steps
The word “abstraction” is used a lot in computer science.

It's one of the Big 7 Ideas.

Abstraction is a way to understand and solve problems.

Abstraction: eliminates details to focus on essential properties.

The instruction example demonstrates functional abstraction – that means we have given a name to a series of operations that perform a meaningful activity.
Collecting the operations together and giving them a name is **functional abstraction**

- The group of operations performs some function but we ignore the details
- Only need to refer to the name we gave it
- This is **AMAZINGLY** powerful
- Why?
  - We can forget about the operations
  - Only thing about what function they do
  - We'll talk more about this...

Let's do some functional abstraction
Example: Abstracting your morning

Get dressed

Dress bottom half
- Put on socks

Dress top half
- Put on pants
- Put on shirt
Functions abstract by packaging computation

- F1() packages actions (e.g. “process a riser”)
Functions abstract by packaging computation

- Here the “main” method has 9 commands
Because F1() “processes a riser”, you can think of the task as:
- Process a riser  F1
- Move to next riser
- Process a riser  F1
- Move to next riser
- Process a riser  F1

With F1() as a concept, we simplified the programming to 9 steps instead of 21 steps.

Hmm, what about...
- Move_to_next_riser()
Functions abstract by packaging computation
Five Instruction main program

New way of thinking
about the problem
1 idea, 2 applications

- To a programmer, the instruction is monolithic (one thing)
- To an agent each instruction is a series of steps

F1(): Process a riser
F2(): Move to next riser

F1 is eight instructions

F1 is one instruction
Formulating blocks of computation:
- Concept is called **functional abstraction**

What we did is important and we do it all the time in everyday life
- We break down tasks into one (or more) subtasks
- We solve subtasks using sequences of instructions
- We put the solution into a function “package”, giving it a name, “process a riser”, thus creating a new thing, a concept we can talk about and use
- Then we use used it to solve something more complicated ... can keep doing this at next higher level

This lets us do more complicated things
True or False

- Functional abstraction allows an agent (like the robot) to complete a task in fewer steps.
The “basic” Lightbot exercises were about
  – sequencing
  – functional abstraction

Next set of exercises are about...
Recursion

• Recursion: the repeated application of a recursive procedure or definition.
Recursion is everywhere

- Mathematics: 0 through 9 are **digits**. A *string of digits* is a digit followed by a string of digits.
void draw()
{
  background(255);
  drawTris(0, depth, new PVector(border, h-border), new PVector(width/2, border), new PVector(width-border, h-border));
}

void drawTris(int level, int maxLevels, PVector left, PVector top, PVector right)
{
  level++;
  if(level >= maxLevels) {
    triangle(left.x, left.y, top.x, top.y, right.x, right.y);
  } else {
    PVector a = PVector.add(left, PVector.div(PVector.sub(top, left), 2));
    PVector b = PVector.add(right, PVector.div(PVector.sub(top, right), 2));
    PVector c = PVector.add(left, PVector.div(PVector.sub(right, left), 2));
    drawTris(level, maxLevels, a, top, b);
    drawTris(level, maxLevels, left, a, c);
    drawTris(level, maxLevels, c, b, right);
  }
}

void mousePressed()
{
  switch(mouseButton) {
    case LEFT: depth++; break;
    case RIGHT: depth = max(depth-1, 0); break;
  }
What controls when it happens?

```cpp
void drawTris(int level, int maxLevels, PVector left, PVector top, PVector right) {
    level++;
    if (level >= maxLevels) {
        triangle(left.x, left.y, top.x, top.y, right.x, right.y);
    } else {
        PVector a = PVector.add(left, PVector.div(PVector.sub(top, left), 2));
        PVector b = PVector.add(right, PVector.div(PVector.sub(top, right), 2));
        PVector c = PVector.add(left, PVector.div(PVector.sub(right, left), 2));
        drawTris(level, maxLevels, a, top, b);
        drawTris(level, maxLevels, left, a, c);
        drawTris(level, maxLevels, c, b, right);
    }
}

void mousePressed() {
    switch(mouseButton) {
        case LEFT: depth++;
        case RIGHT: depth = max(depth-1, 0);
        break;
    }
    redraw();
}
Why doesn't it go on forever?

drawTris(0, depth, new PVector(border, n-border), new PVector(width/2, border), new PVector(width-border, n-border));

void drawTris(int level, int maxLevels, PVector left, PVector top, PVector right)
{
  if(level >= maxLevels) {
    triangle(left.x, left.y, top.x, top.y, right.x, right.y);
  } else {
    PVector a = PVector.add(left, PVector.div(PVector.sub(top, left), 2));
    PVector b = PVector.add(right, PVector.div(PVector.sub(top, right), 2));
    PVector c = PVector.add(left, PVector.div(PVector.sub(right, left), 2));
    drawTris(level, maxLevels, a, top, b);
    drawTris(level, maxLevels, left, a, c);
    drawTris(level, maxLevels, c, b, right);
  }
}

void mousePressed()
{
  switch(mouseButton) {
    case LEFT: depth++; break;
    case RIGHT: depth = max(depth-1, 0); break;
  }
  redraw();
}
Why does recursion work?

- Size of 'problem' gets smaller on each call.
- Stopping conditions.
First Try: Recursion 2
Second Try: Recursion 2
Third Try: Recursion 2

What should stop condition be?
Programming Rule 1: Keep trying!

The agent does exactly what you tell it to do.
Recursion 3: First try
Recursion 3: First try
Recursion 3: Second try
Recursion 3: Third try
Recursion 4: Whoa!! What is F2 subroutine?
What do you think?
Recursion 4: Whoa!! What is F2 subroutine?
Recursion 4: Whoa!! What is F2 subroutine?
Recursion 4: Keep Trying
The Function is just Packaging

- Another solution to the risers problem
Programming is commanding an agent
- **Agent**: usually a computer, can be person or device
- Agent follows **instructions**, flawlessly
- The program implements human intent

Instructions are *given* in sequence

... and *executed* in sequence
- Limited repertoire, within ability, one at a time
- “Program counter” keeps track of current instruction

Formulating computation as a “concept” is **functional abstraction**
An instruction is **abstracted** into the command name

- functions **abstract** useful and meaningful operations
- functions abstract functions built of functions...

Layer upon layer, we build software solutions
Homework 2: Symbolic Lightbot

Goal: The purpose of this assignment is to look at textual ways of programming the Lightbot and to learn how to express a function symbolically.

When we solved the Lightbot 2.0 exercises last time, we programmed the bot using a tiny list of instructions presented iconographically, that is, as pictures. But, these could just as easily be written symbolically, that is, as text. So, our instruction list

![Lightbot Symbols]

can be expressed symbolically,

```
Step  Right  Left  Jump  Power  F.name
```

as shown. When we program symbolically, we don’t have to name the functions F1 and F2. We can give them actual names like `F.turn_around` for a function that causes the bot to turn around. So, we can give the symbolic solution to the following Lightbot problem as

```
Left, Step, Right, Step, Step, Step, Step, Step, Right, Step, Power.
```
Reading for next time

- Read *Blown to Bits* chapter 2.
Look at processing.org
Break