Instructions Formed of Simpler Instructions

- Check out this screen shot of the Lightbot
- It is partway through an instruction ... its beacon is lit, but not the tile
- To a programmer the instruction is monolithic (one thing)
- To an agent each instruction is a series of steps

An Instruction abstracts those steps

Abstraction

- The word “abstraction” is used a lot in computing
- Remember: it was one of the 7 big ideas
- Abstraction is a way to understand and solve problems
- As a general definition, abstraction eliminates details to focus on essential properties
- The instruction example just given illustrates functional abstraction meaning that we have given a name to a series of operations that perform a coherent (and to us meaningful) activity; the name is the instruction, the series of operations are the bot’s actions to implement it

Abstracting

- Collecting the operations together and giving them a name is functional abstraction
- The group of operations perform some function but we ignore all of the details
- Giving it a name is functional abstraction
- This is AMAZINGLY powerful
- What makes it powerful, is we can forget about the operations and think only about the function they do; more about this later
- Let’s do some functional abstraction

Functions abstract by packaging Computation

- \( F_1() \) packages actions: E.G. “process a riser”

The Function Becomes A Concept

- Because \( F_1() \) "processes a riser," I think of the programming task as

- With \( F_1() \) as a concept, I simplify the programming to just 5 steps rather than 21
- It also suggests another concept:
  - \( \text{Move_to_next_riser}() \)
Layering: Building Functions out of Functions

Functional Abstraction Reduces Complexity

Functions In Processing

- Form of function definition in Processing
  ```java
  <return type> <name> ( <param list> ) { 
  <body>
  }
  as in
  void draw_a_box (int x_pos, int y_pos) {
  rect(x_pos, y_pos, 20, 20);
  color pink();
  or
  return color(255, 200, 200);
  }
  ```

Functions In Processing: Result

- Functions that do something, but do not return a value, have `void` as their `return type`
- Functions that return a value must say its type

```java
void draw_a_box (int x_pos, int y_pos) {
rect(x_pos, y_pos, 20, 20);
}
color pink() {
return color(255, 200, 200);
}
```

Functions In Processing: Params

- Parameters are the values used as input to the function; parameters are not required, but the parentheses are
- The type of each parameter must be given

```java
void draw_a_box (int x_pos, int y_pos) {
rect(x_pos, y_pos, 20, 20);
color pink();
return color(255, 200, 200);
}
```

Functions In Processing: Return

- A function returns its value with the `return` statement ... the stuff following `return` is the result
- The function is done when it reaches `return`

```java
void draw_a_box (int x_pos, int y_pos) {
rect(x_pos, y_pos, 20, 20);
color pink();
return color(255, 200, 200);
}
```

Writing Functions

- Processing function definitions are typically listed after the standard blocks: `setup()`, `draw()`, `mousePressed()`, etc.

```java
void draw() {
fill(255, 0, 0);
rect(20, 20);
color pink();
return color(255, 200, 200);
}
```
Using Functions

- Once defined, functions can be called repeatedly...it’s the point of writing them!

Arguments Become Parameters

- Notice that if the DEFINITION has n parameters, the CALL needs n arguments
- The parameters and arguments correspond

Parameters

- Parameters are automatically declared (and initialized) on a call, and remain in existence as long as the function remains unfinished
- When the function ends, the parameters vanish, only to be recreated on the next call
- It is wise to choose parameter names that help you remember exactly what they mean
  - colorFlag (Chao’s code)
  - dir (for direction)
  - I chose xbase as the orientation point of the figure in the x direction, I use that name a lot and I know what it means

Example: Clock Timer. In sample codes.

- Draw digital timer elements
- Assemble elements into digits
- Light digit segments to create numbers
- Select number based on a digit

Define hexa() and rexa()

- Pattern: Parameterize the functions by a consistent position – upper left corner is good

Use H’gons to Form A Digit: digit re-uses
Let There Be Light (and Dark)

- Define the illumination of the digit
  - Must declare two color variables, initialize to proper colors, use them in fill, and then check them

```c
void setup() {
  size(250, 180);
  background(255);
  stroke(0);
}

void draw() {
  int l = colorMap(0, 255, colorMap(0, 255, 255));
  stroke(l);
  fill(l);
  //triangle
  //rectangle
}
```

Count In Lights: a function for each number

- Light up the digit for each number:

```c
void digit(int n, float base, float phase) {
  //fill color
  //draw shape
}
```

Select A Number To Display

- Given an integer, display it in lights

```c
void digit(int n, float base, float phase) {
  //fill color
  //draw shape
}
```

Create a 3 Digit Display

```c
void three_digit(int n, float base, float phase) {
  //fill color
  //draw shape
}
```

Count up At The Frame Rate

- Review What We Did

```c
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{digit} & \text{one} & \text{two} & \text{three} & \text{four} & \text{five} & \text{six} & \text{seven} \\hline
\text{hexa} & \text{hexa} & \text{hexa} & \text{hexa} & \text{hexa} & \text{hexa} & \text{hexa} & \text{hexa} \\hline
\end{array}
```

- The computation is ONLY drawing triangles and rectangles, but we don’t think of it that way … to us, it’s a timer
Homework 10: Functions

- void cell(int x, int y, int s, color tinto)
- void triple(int x, int y, int s, color tinto)
- Also functions for
  - block()
  - row()
  - cellarray()