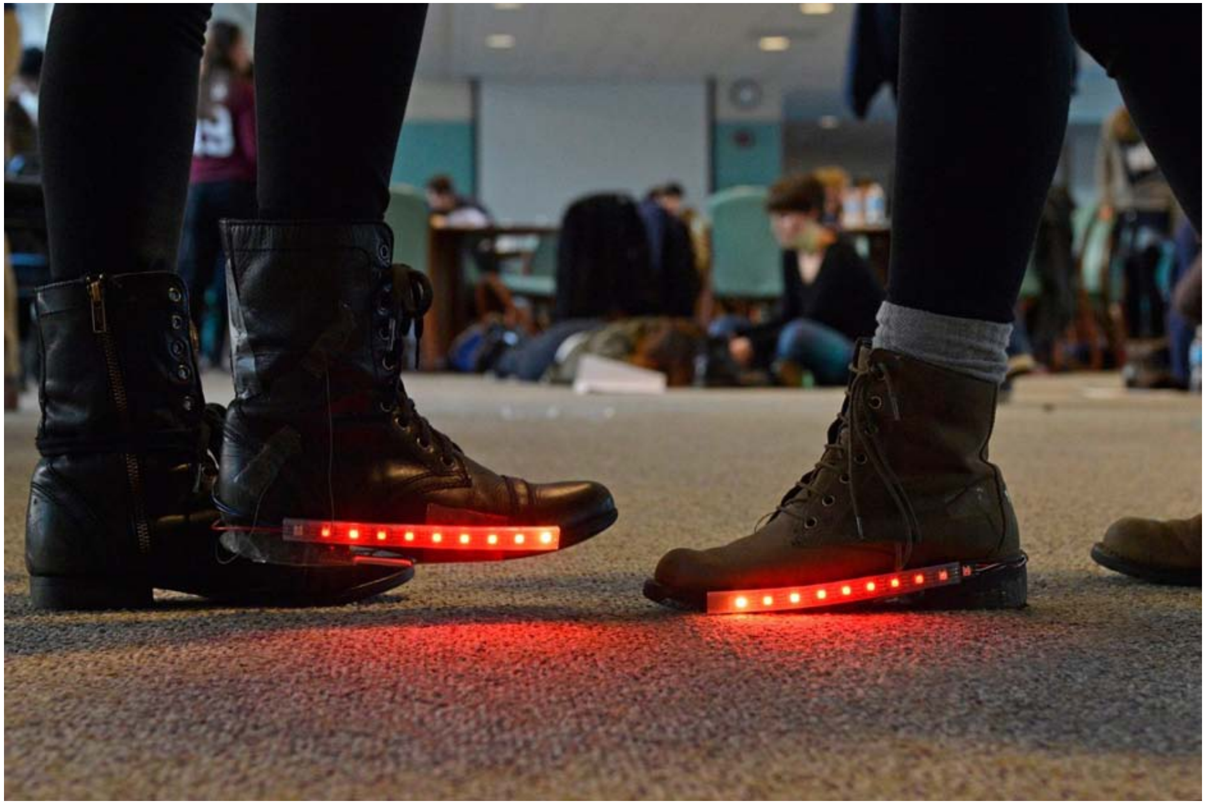
Make Your Own Wearables Workshop

CIRCUITS AND CODE

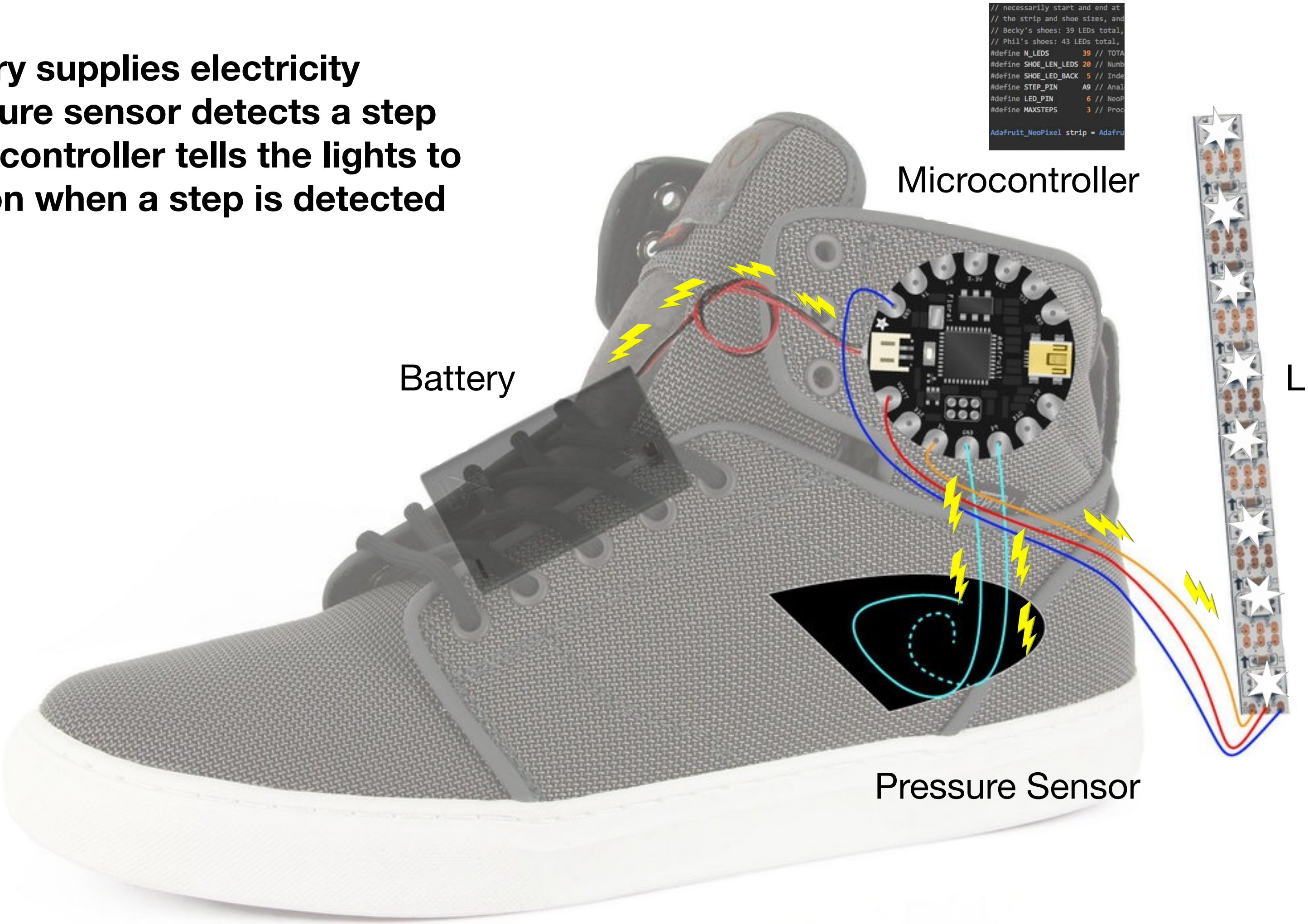
Today’s Project



https://www.youtube.com/watch?v=gWZi71JkPAA

What is in our circuit?

- **Battery supplies electricity**
- **Pressure sensor detects a step**
- **Microcontroller tells the lights to turn on when a step is detected**



```
// necessarily start and end at
// the strip and shoe sizes, and
// Becky's shoes: 39 LEDs total,
// Phil's shoes: 43 LEDs total,
#define N_LEDS        39 // TOTA
#define SHOE_LEN_LEDS 20 // Numb
#define SHOE_LED_BACK  5 // Inde
#define STEP_PIN      A9 // Anal
#define LED_PIN        6 // NeoP
#define MAXSTEPS       3 // Proc

Adafruit_NeoPixel strip = Adafru
```

Courtesy of Adafruit. Used with permission.

Your body is like the Firewalker Circuit

Your brain is like a *microcontroller—that's where all the processing happens!*

Your hands, ears, eyes, nose and tongue are all ***sensors***

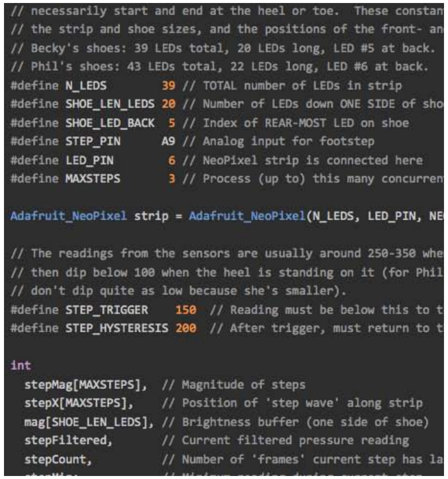
Your nerves are like ***wires*** that send messages from your sensors to your brain

You get energy from food, like the circuit gets energy from a ***battery***

You can move and talk based on your sensory input and what your brain tells you to do!

What is computer programming?

- A program is step-by-step instructions for a computer
- Each instruction is processed one at a time, *exactly* as written.
- For the Firewalker circuit, code instructions are:
  - Light up when a step is detected by the pressure sensor



```
// necessarily start and end at the heel or toe.  These constan
// the strip and shoe sizes, and the positions of the front- an
// Becky's shoes: 39 LEDs total, 20 LEDs long, LED #5 at back.
// Phil's shoes: 43 LEDs total, 22 LEDs long, LED #6 at back.
#define N_LEDS        39 // TOTAL number of LEDs in strip
#define SHOE_LEN_LEDS 20 // Number of LEDs down ONE SIDE of sho
#define SHOE_LED_BACK  5 // Index of REAR-MOST LED on shoe
#define STEP_PIN      A9 // Analog input for footstep
#define LED_PIN        6 // NeoPixel strip is connected here
#define MAXSTEPS       3 // Process (up to) this many concurren

Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, LED_PIN, NE

// The readings from the sensors are usually around 250-350 whe
// then dip below 100 when the heel is standing on it (for Phil
// don't dip quite as low because she's smaller).
#define STEP_TRIGGER    150  // Reading must be below this to t
#define STEP_HYSTERESIS 200  // After trigger, must return to t

int
  stepMag[MAXSTEPS],  // Magnitude of steps
  stepX[MAXSTEPS],    // Position of 'step wave' along strip
  mag[SHOE_LEN_LEDS], // Brightness buffer (one side of shoe)
  stepFiltered,       // Current filtered pressure reading
  stepCount,          // Number of 'frames' current step has la
```

“If” Statements

“If” statements allow your program to ignore lines of a code (or instructions) if a certain *condition* isn’t met.

Your brain uses “if” statements every day!

You wake up and check the temperature outside.

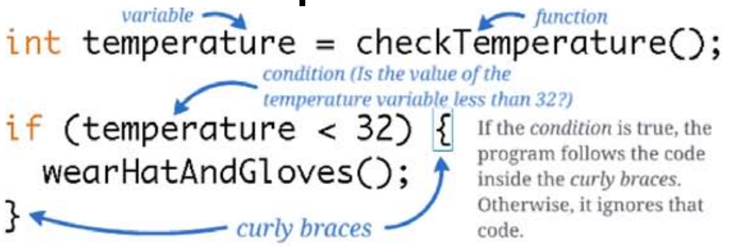
If it’s freezing (less than 32 degrees Fahrenheit)…

Courtesy of Emery Way. License: CC BY.

Then you wear a hat and gloves that day. (Otherwise you skip the hat and gloves)

Courtesy of Linden Down. Used with permission.

Example Code:



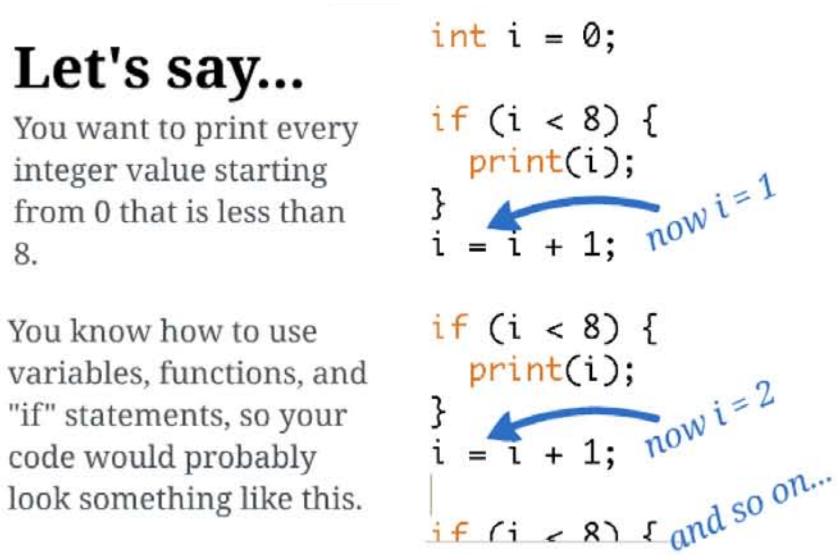
```
int temperature = checkTemperature();
if (temperature < 32) {
  wearHatAndGloves();
}
```

variable
function
condition (Is the value of the temperature variable less than 32?)
curly braces

If the *condition* is true, the program follows the code inside the *curly braces.* Otherwise, it ignores that code.

Loops

A “loop” allows your program (or instructions) to repeat until a certain *condition* is no longer true.



Let's say...

You want to print every integer value starting from 0 that is less than 8.

You know how to use variables, functions, and "if" statements, so your code would probably look something like this.

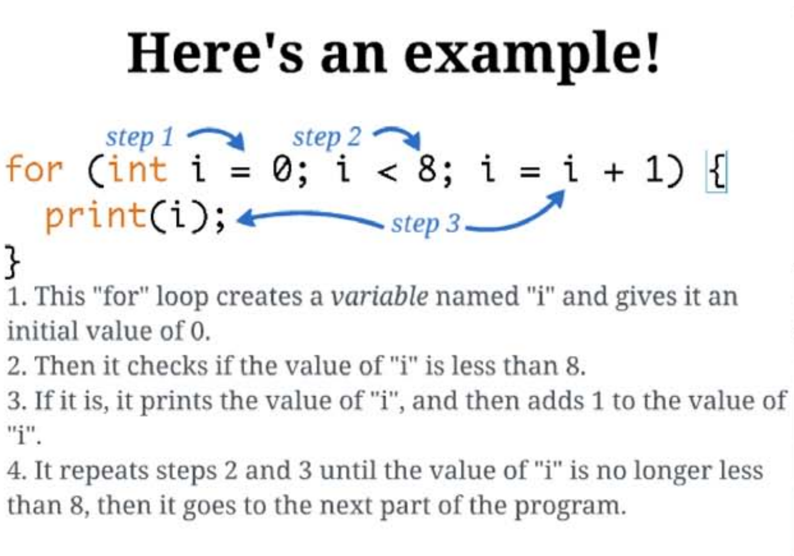
```
int i = 0;

if (i < 8) {
  print(i);
}
i = i + 1;

if (i < 8) {
  print(i);
}
i = i + 1;

if (i < 8) {
```

now i = 1
now i = 2
and so on...



Here's an example!

```
for (int i = 0; i < 8; i = i + 1) {
  print(i);
}
```

step 1
step 2
step 3

1. This "for" loop creates a *variable* named "i" and gives it an initial value of 0.
2. Then it checks if the value of "i" is less than 8.
3. If it is, it prints the value of "i", and then adds 1 to the value of "i".
4. It repeats steps 2 and 3 until the value of "i" is no longer less than 8, then it goes to the next part of the program.

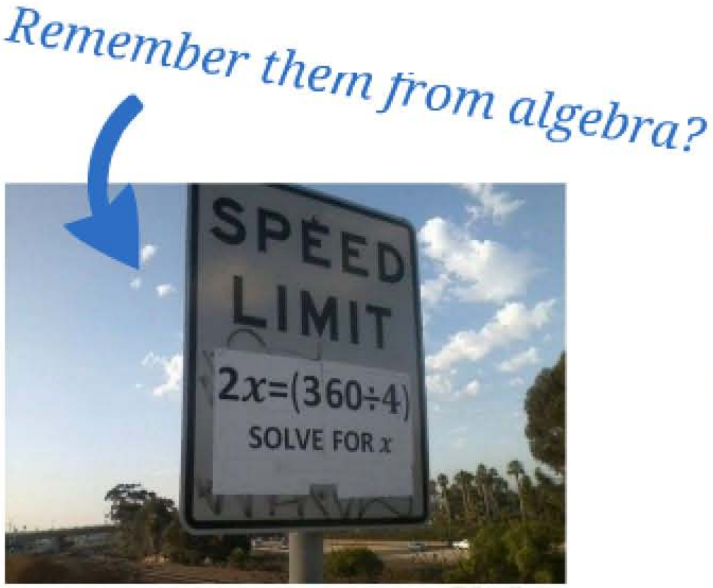
Let’s program a “robot” to build a PBJ Sandwich

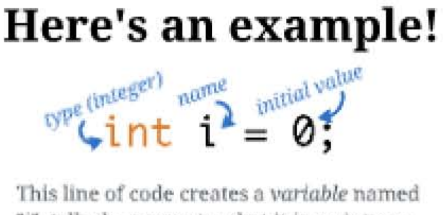
Variables

Use ***variables*** to give names to the values in your program.

The values might change, but the names will stay the same.

You can name variables whatever you want! Give them good, descriptive names and your program will be easy to read and understand.





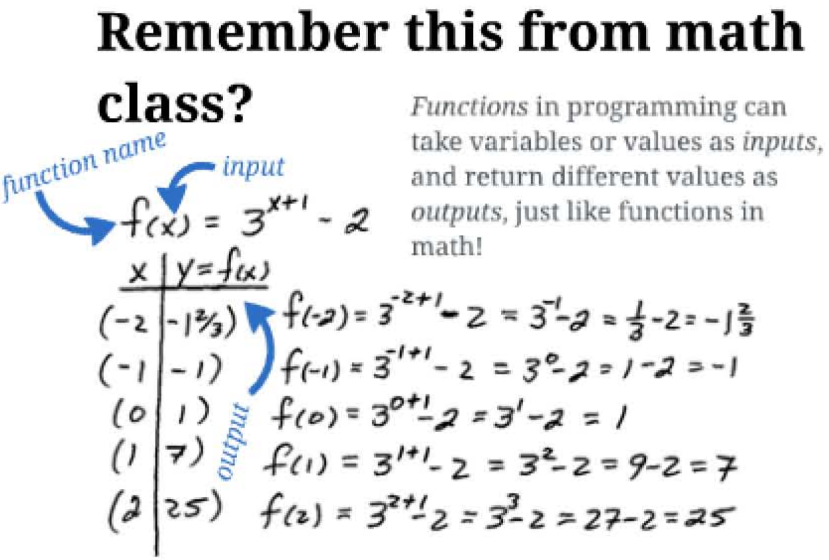
Here's an example!

```
int i = 0;
```

This line of code creates a *variable* named

Functions

Functions are like variables, but they give names to whole sections of code instead of just one value



Remember this from math class?

Functions in programming can take variables or values as *inputs*, and return different values as *outputs*, just like functions in math!

$$f(x) = 3^{x+1} - 2$$

x	y=f(x)
(-2	-1⅔)
(-1	-1)
(0	1)
(1	7)
(2	25)

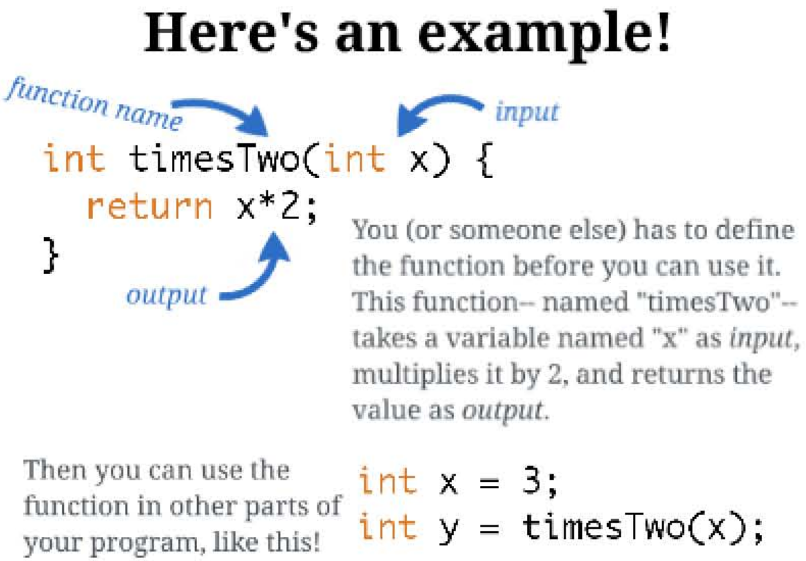
$$f(-2) = 3^{-2+1} - 2 = 3^{-1} - 2 = \frac{1}{3} - 2 = -1\frac{2}{3}$$

$$f(-1) = 3^{-1+1} - 2 = 3^{0} - 2 = 1 - 2 = -1$$

$$f(0) = 3^{0+1} - 2 = 3^{1} - 2 = 1$$

$$f(1) = 3^{1+1} - 2 = 3^{2} - 2 = 9 - 2 = 7$$

$$f(2) = 3^{2+1} - 2 = 3^{3} - 2 = 27 - 2 = 25$$



Here's an example!

```
int timesTwo(int x) {
  return x*2;
}
```

You (or someone else) has to define the function before you can use it. This function-- named "timesTwo"-- takes a variable named "x" as *input*, multiplies it by 2, and returns the value as *output*.

Then you can use the function in other parts of your program, like this!

```
int x = 3;
int y = timesTwo(x);
```