

# IoT Product Design and Coding Bootcamp

Brian Rashap, Ph.D.

05-Oct-2020

# IoT Fun



# Brian Rashap, Ph.D.

- Proud husband of Krista and father of Shelby (22) and Ethan (18)
- Electrical Engineer with 25 years industrial experience
- High School track coach
- Hobbies: running, cycling, reading, spending time with family



# Introductions

## INTRODUCTIONS

# Class Rules

- Respect Each Other, Help Each Other
- Ask Questions
- Be On Time (let us know via Slack if you won't be here)
- Keep Your Workspace and the Classroom Neat and Tidy
- If you are struggling, let me, Susan, or Esteban know. We are here to HELP!

# Grading

Class assignments will total 1000 points. You will need to earn at least 750 total points and at least 225 points on your Capstone to graduate.

- ① IoT assignments + Lab Notebooks: 300 pts
- ② Fusion 360 assignments: 100 pts
- ③ Weekly quizzes: 100 pts
- ④ Midterm Projects: Smart Room Controller/Plant Watering System: 200 pts
- ⑤ Team Capstone Project: 300 pts

More information later today on how assignments are turned in

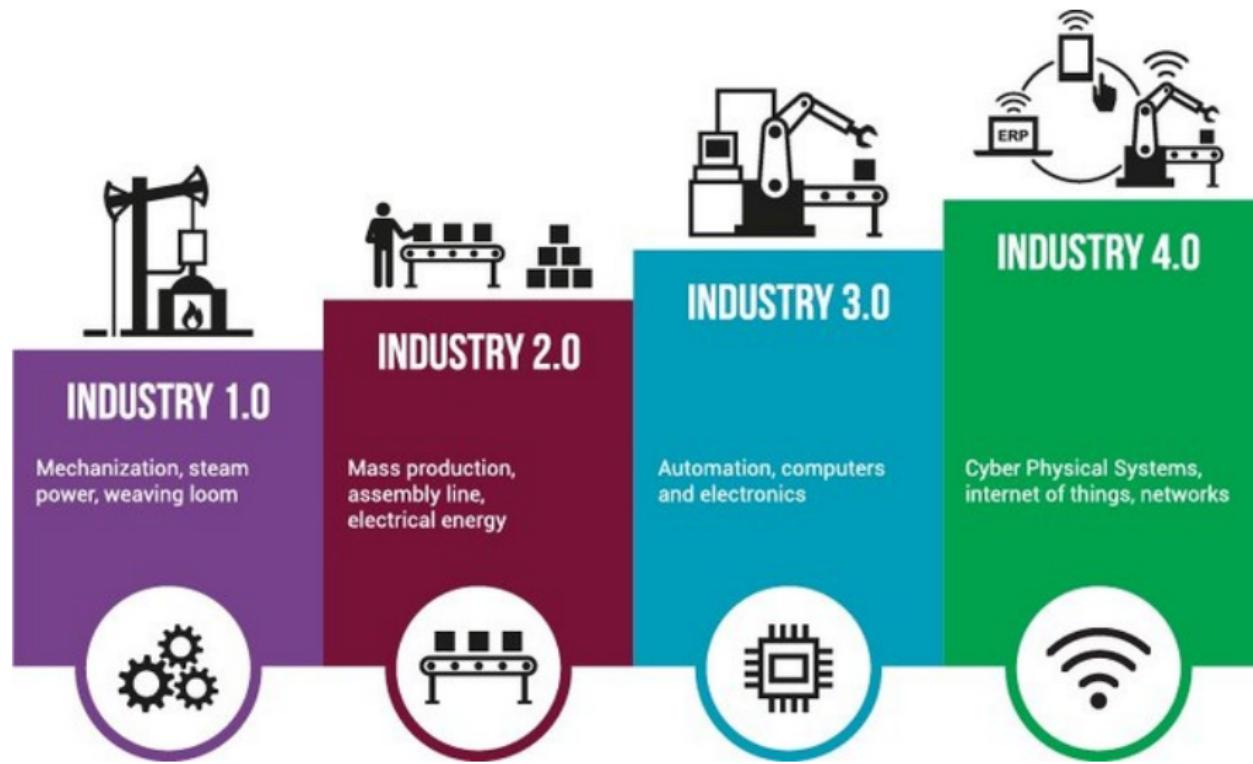
# Credit for Prior Learning (CPL)

Approved for CPL	
CIS 1605	Internet of Things
CIS 1275	Introduction to C++
BCIS 1110	Fundamentals of Information Literacy and Systems
BUSA 1130	Business Professionalism
BUSA 1198	Project Management Fundamentals
CSIS 1151	Intro to Programming for Non-Majors of CS*

\* CSIS 1151 credit requires appropriate math prerequisites

Under Review for CPL	
RPID 1005	3 Dimensional CAD
RPID 1010	Design and Simulation
RPID 1015	Prototype Fabrication I
RPID 1020	Prototype Fabrication II

# Evolution of Industry



# Components of Industry 4.0



# IoT and Data Science



**AI:** Data-based learning

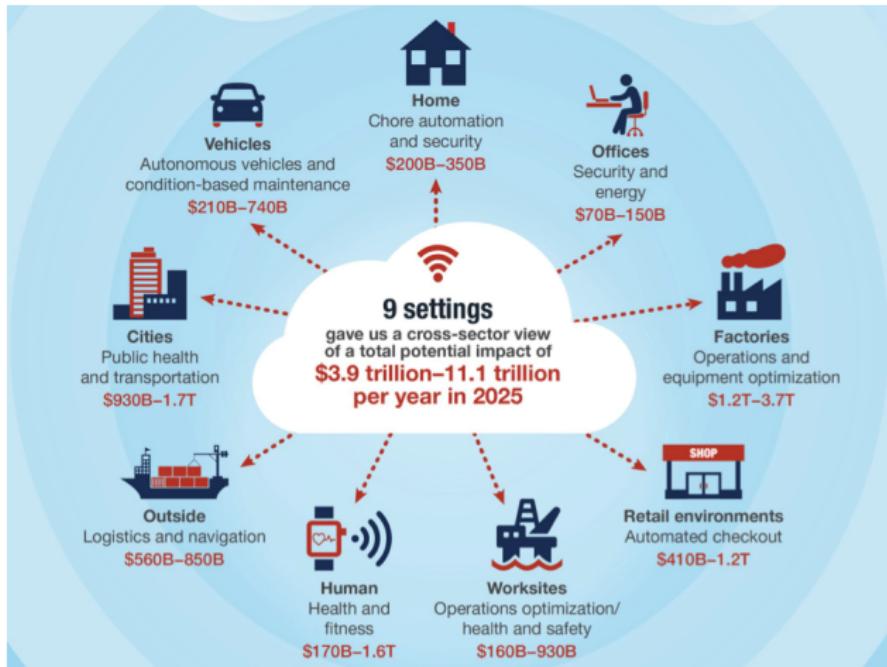


**Big Data:** Capture, storage, analysis of data



**IOT:** Data Collection through IoT

# IoT 2025



# Smart Facilities



# Healthcare 2025



# Smart World

## Libelium Smart World



# And Out of This World



# IoT Growth



# Let's Begin Our Journey



# Computer Languages

## Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,000-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, C#, Cobol, Java and other modern source codes dominate our systems, hundreds of others are running out of time.

An ad hoc collection of ancient and electronic lexicographers, if you will—aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of code still fluent in these nearly forgotten linguistic frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the tools of software practice, the mind-set of programmers, the technical, social, and economic factors that shaped history," he says. Booch explains, "It's important to provide the raw material for software archaeologists, historians, and developers to learn what worked; what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive roundup, check out the Language List at [HTTP://WWW.INFORMATIK.UNI-FREIBURG.DE/JAVA/MSVC/LANG\\_LIST.HTML](http://www.informatik.uni-freiburg.de/Java/msvc/lang_list.html). - Michael Mendano

Key
Year Introduced
Active: thousands of users
Protected: taught at universities; compilers available
Extinct: usage dropping off
Extinct: no known active users or up-to-date compilers
Lineage continues



# Computer Languages



# Why C++

## Features of C++



# CLI vs GUI

[root@localhost ~]# cd /var  
[root@localhost var]# ls -la  
total 72  
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .  
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..  
drwxr-xr-x. 2 root root 4096 May 14 00:15 account  
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache  
drwxr-xr-x. 3 root root 4096 May 18 16:03 db  
drwxr-xr-x. 3 root root 4096 May 18 16:03 empty  
drwxr-xr-x. 2 root root 4096 May 18 16:03 games  
drwxrwx-T. 2 root gdm 4096 Jun 2 18:39 pdfs  
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib  
drwxr-xr-x. 2 root root 4096 May 18 16:03 log  
drwxr-xr-x. 2 root root 4096 May 18 16:03 media  
drwxr-xr-x. 2 root root 4096 May 18 16:03 private  
drwxr-xr-x. 2 root root 4096 Jul 1 22:11 repodata  
lrwxrwxrwx. 1 root root 6 May 14 00:12 run -> /run  
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool  
drwxrwxrwt. 4 root root 4096 Sep 12 23:58 tmp  
drwxr-xr-x. 2 root root 4096 May 18 16:03 yp  
[root@localhost var]# yum search wiki  
No matches for package 'wiki'  
Available plugins: langpacks, presto, refresh-packagekit, reponame, transaction-testing, primary\_db  
Brian Rashap, Ph.D.

Start



VS



# Command Line Interface - Basic Navigation

The Command Line Interface (CLI) will allow us to directly navigate the computers operating system. We will use:

- macOS or Linux: Terminal
- Windows: PowerShell

The following commands will work on all three system, except where noted below. macOS and Linux are case-sensitive, Windows is not.

- `pwd`: Show the present working directory.
- `ls`: To get the list of all the files or folders.
- `cd`: Used to change the directory.
- `du`: Show disk usage. (not available in PowerShell).
- `man`: Used to show the manual of any command.

# Command Line Interface - File and Directory Manipulation

- **mkdir:** Used to create a directory if it does not already exist. It accepts directory name as input parameter.
- **rmdir:** It is used to delete a directory if it is empty.
- **cp:** This command will copy the files and directories from source path to destination path. It can copy a file/directory with new name to the destination path. It accepts source file/directory and destination file/directory.
- **mv:** Used to move the files or directories. This command's working is almost similar to cp command but it deletes copy of file or directory from source path.
- **rm:** Used to remove files or directories.
- **touch:** Used to create or update a file. (PowerShell New-Item).

# Command Line Interface - Displaying the file contents

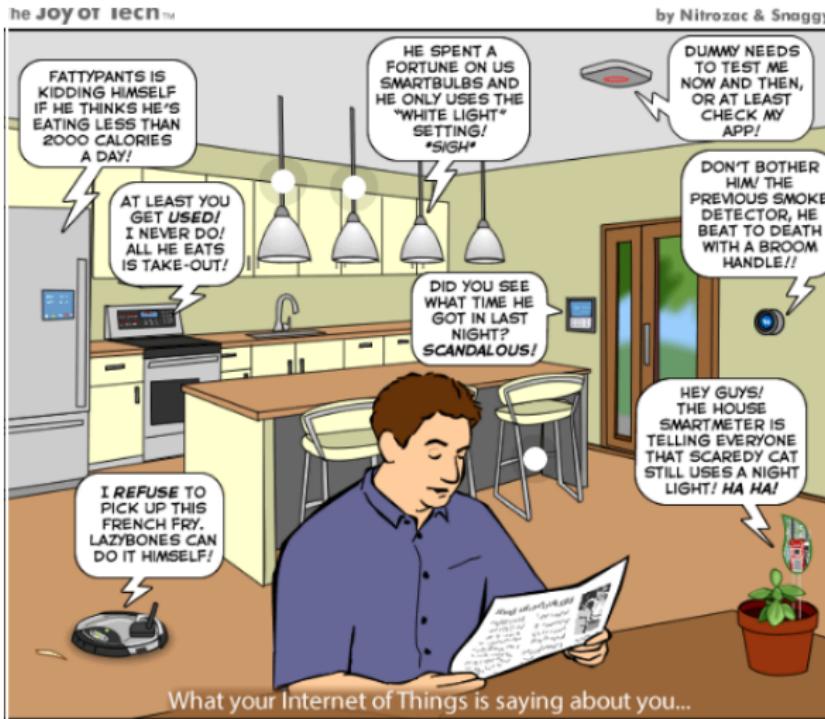
- cat: It is generally used to concatenate the files. It gives the output on the standard output.
- more: It is a filter for paging through text one screenful at a time.

The below commands are not available in PowerShell:

- less: It is used for viewing the files instead of opening the file. Similar to more command but it allows backward as well as forward movement.
- head: Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.
- tail: Used to print the last N-1 lines of a file. It accepts N as input and the default value of N is 10.

On all systems, commands can be "piped" together: ls | more <file>

# IoT Fun



iovtotech.com

# Our First Microcontroller

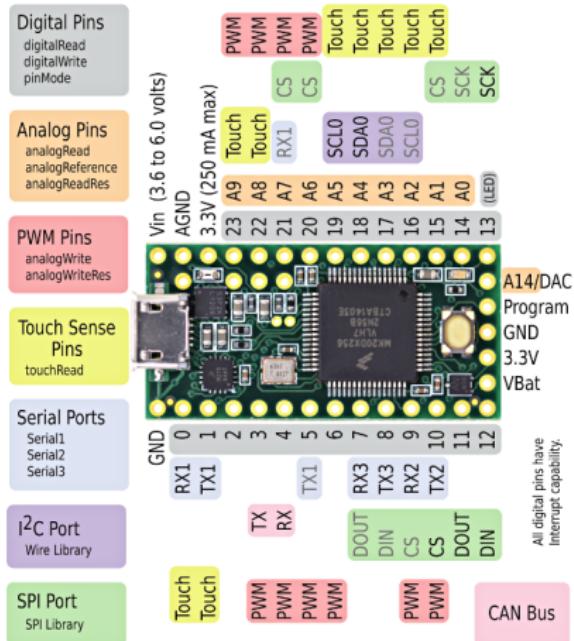


# Smart Room Controller



# Teensy 3.2

- Cortex-M4 72MHz (overclocked to 96 MHz)
- 34 GPIO pins
- 3.3V and 5.0V operating voltages
- 500mA of available power with USB



# Arduino IDE for Teensy

We are going to start off using the Arduino IDE<sup>1</sup>. The Arduino IDE is programmed essentially using C++ code, but makes the compiling and loading onto the microcontroller simpler.

We begin by installing the Arduino IDE:

*<https://www.arduino.cc/en/main/software>*

Then, we install the Teensyduino add-on:

*[https://www.pjrc.com/teensy/td\\_download.html](https://www.pjrc.com/teensy/td_download.html)*

---

<sup>1</sup>An IDE, or Integrated Development Environment, enables programmers to consolidate the different aspects of writing a computer program.

# Other Software

## ① Git

- <https://git-scm.com/downloads>

## ② Fritzing

- IoT Bootcamp Teams Site

## ③ Drawio

- <https://app.diagrams.net/>

## ④ Fusion 360

- Instructional Videos - Teams Site

## ⑤ Formlab's Preform

- <https://formlabs.com/software/>

## ⑥ Ultimaker's Cura

- <https://ultimaker.com/software/ultimaker-cura>

# GITHUB Simplified

```
1 // In Powershell go to ./Documents/<yourname>
2 // Get a repository that already exists and pull
   it into your local machine
3 git clone <URL of repository>
4
5 // From the repository directory, get updates
6 git pull
7
8 // Send your changes up to the repository
9 git add .
10 git commit -m "<comment>"
11 git push
12
13 // You may get asked to enter your GIT username
14 git config --global user.email "you@example.com"
```

# GITHUB First Clone

ddc-iot-classroom-2

Accept the assignment —

L01\_HelloWorld

Once you accept this assignment, you will be granted access to the `l01-helloworld-brashap` repository in the `ddc-iot` organization on GitHub.

Accept this assignment



You're ready to go!

You accepted the assignment, **L01\_HelloWorld**.

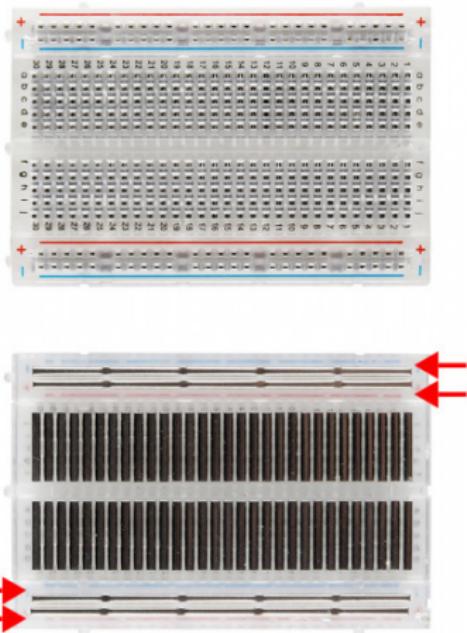
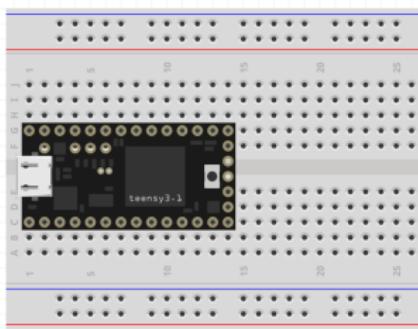
Your assignment repository has been created:

<https://github.com/ddc-iot/l01-helloworld-brashap>

```
1 brian:~$ cd Documents/
2 brian:Documents$ mkdir IoT
3 brian:Documents$ cd IoT
4 brian:IoT$ git clone https://github.com/ddc-iot/L01_helloWorld-brashap
5 Cloning into 'L01_helloWorld'...
6 Username for 'https://github.com': brashap
7 Password for 'https://brashap@github.com':
8 remote: Enumerating objects: 4, done.
9 remote: Counting objects: 100% (4/4), done.
10 remote: Compressing objects: 100% (3/3), done.
11 remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
12 Unpacking objects: 100% (4/4), 321 bytes | 53.00 KiB/s, done.
```

Second Clone: [https://github.com/ddc-iot/class\\_slides](https://github.com/ddc-iot/class_slides)

# Teensy on Breadboard



# Basic Structure of Arduino Sketch

```
1 // the "header" is used for GLOBALS
2
3 void setup() {
4     // code in setup() runs once
5     // it is used to initialize objects,
6     // begin processes, and set variables
7     pinMode(13, OUTPUT);    //set Pin 13 as an Output
8 }
9
10 void loop() {
11     // functionality of your code
12     // this loops indefinitely
13 }
```

# Class Assignments

- ① Lab Notebook - flow chart
- ② Lab Notebook - schematic
- ③ Fritzing breadboard layout
- ④ Arduino code with comments

```
1 /*
2  * Project:      Title of Project
3  * Description: Description of Project
4  * Author:       Your Name
5  * Date:        Today's Date
6 */
7
8 // Single Line Comments
```

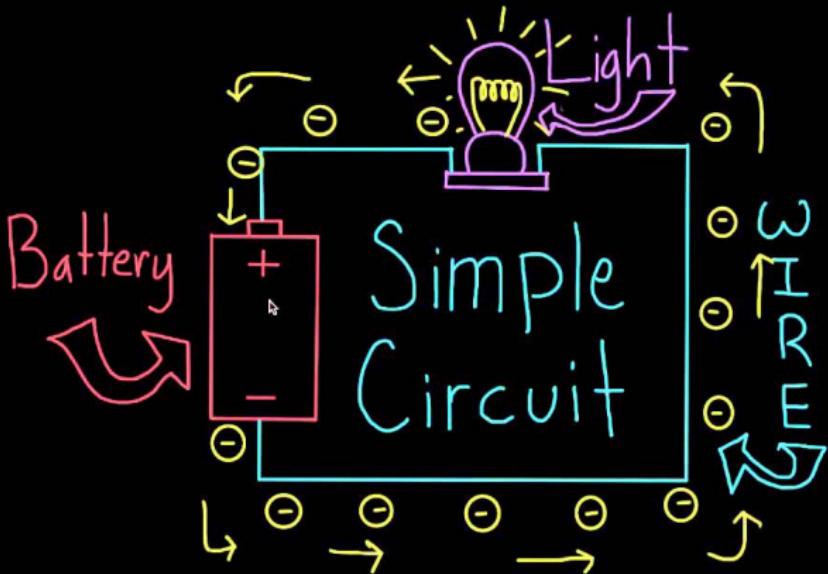
# Assignment L01\_01\_HelloWorld



We will write our first program together as a class, using:

- `pinMode(pin,mode)`
- `digitalWrite(pin,state)`
- `delay(delay_time)`

# Introduction to Electrical Circuits



# Energy

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

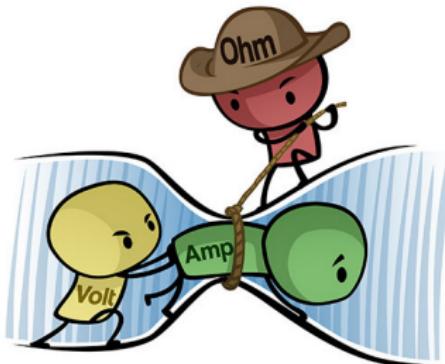
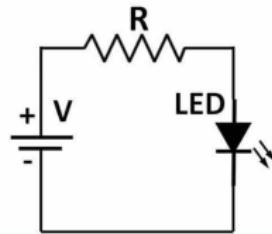


# Ohm's Law

Georg Ohm (16 March 1789 – 6 July 1854) was a German physicist and mathematician. As a school teacher, Ohm began his research with the new electrochemical cell, invented by Italian scientist Alessandro Volta. Ohm found that there is a direct proportionality between the potential difference (voltage) applied across a conductor and the resultant electric current. This relationship is known as Ohm's law:

## Ohm's Law

$$V = I * R$$



# Resistor Color Bands

**4-Band-Code**

2%, 5%, 10%

560k Ω ± 5%

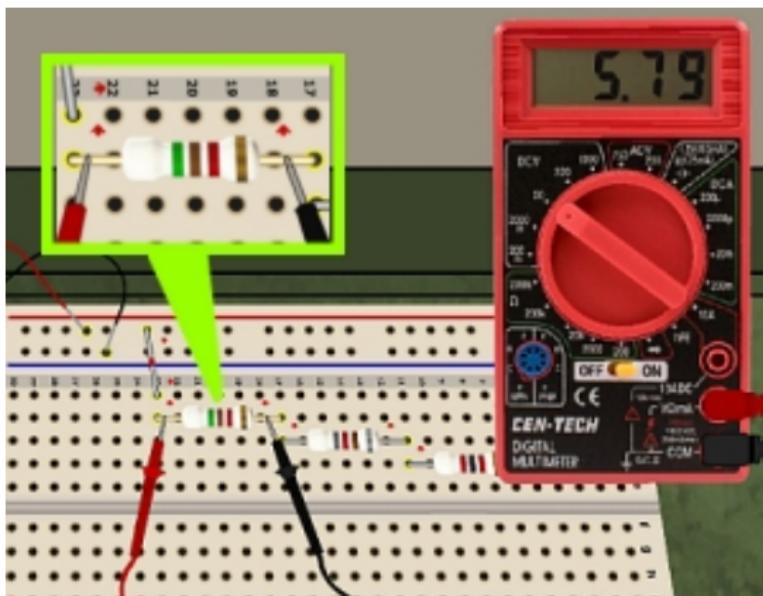
COLOR	1 <sup>ST</sup> BAND	2 <sup>ND</sup> BAND	3 <sup>RD</sup> BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

**5-Band-Code**

0.1%, 0.25%, 0.5%, 1%

237 Ω ± 1%

# Measuring Voltage, Current, and Resistance



# Power from the Teensy 3.2



The Teensy 3.2 has three pins related to power:

- 3.3V: 250mA of power to be used for most hardware
- $V_{in}$ : 5V from the USB cable to power 5V hardware
- GND: The ground pin to close the electrical loop

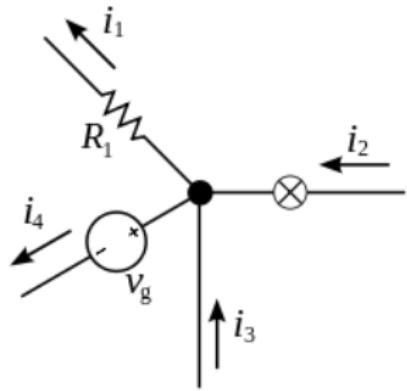
# One Resistor



# Kirchhoff's First Law

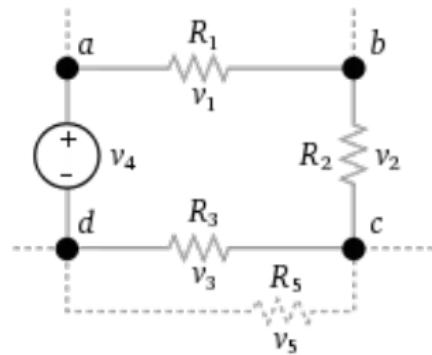
Gustav Robert Kirchhoff (12 March 1824 – 17 October 1887) was a German physicist who contributed to the fundamental understanding of electrical circuits. His first law:

In an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node

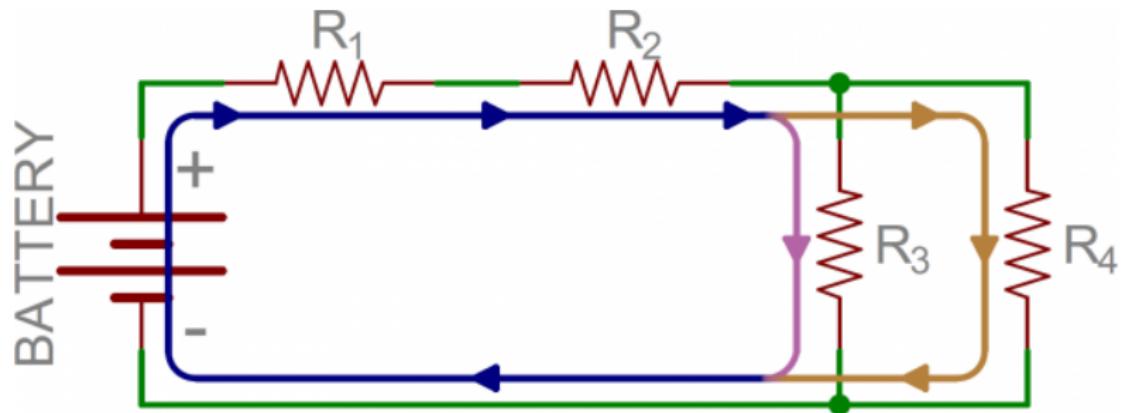


# Kirchhoff's Second Law

The directed sum of the potential differences (voltages) around any closed loop is zero.



# Resistors in Series and Parallel



# Resistors in Series and Parallel



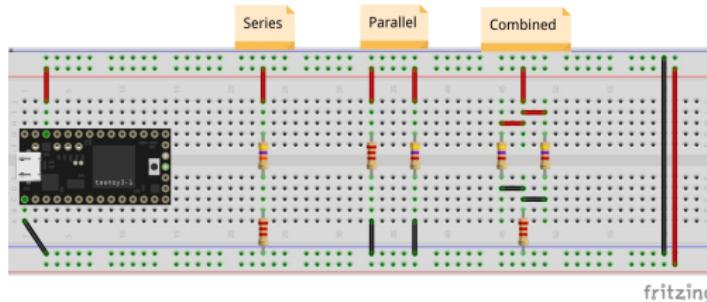
Parallel resistors

$$\frac{1}{R_{\text{equivalent}}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Series resistors

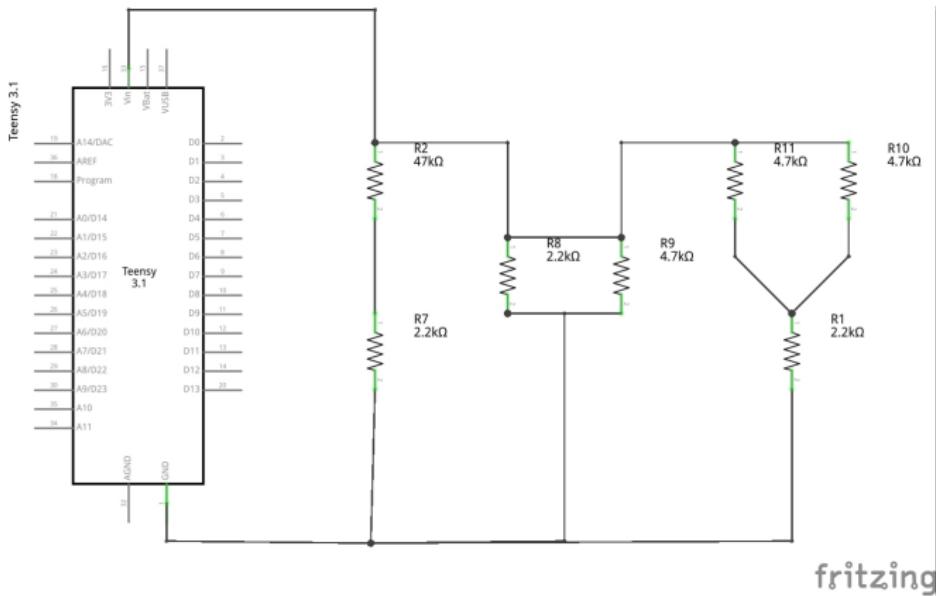
$$R_{\text{equivalent}} = R_1 + R_2$$

# Assignment: L02\_00\_Resistors



- In your lab notebook, draw the circuit diagrams
  - ① Series:  $4.7\text{k}\Omega$  and  $2.2\text{k}\Omega$
  - ② Parallel:  $4.7\text{k}\Omega$  and  $2.2\text{k}\Omega$
  - ③ Combined: Two  $4.7\text{k}\Omega$  in series with  $2.2\text{k}\Omega$
- Calculate the combined resistance and the voltage at each node, as well as the current through each component.
- Create Fritzing diagram
- Build (**one at a time**) on your breadboard and test your calculations with a multimeter.

# Schematics in Fritzing



In Fritzing, go to the Schematic tab and layout your resistors.

**NOTE:** The schematic for the Teensy does not match it's physical layout.

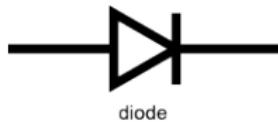
# Diodes

The key function of a diode is to control the direction of current-flow. Current passing through a diode can only go in one direction, called the forward direction. Current trying to flow the reverse direction is blocked.



# Light Emitting Diodes

LEDs (that's "ell-ee-dees") are a particular type of diode that convert electrical energy into light.



diode



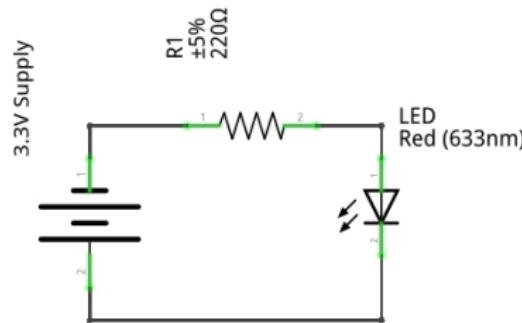
light emitting diode

# Current Limiting Resistors

As a LED has very little resistance, when it is connected directly to a power supply, the current draw will exceed its specs and it will burn out.

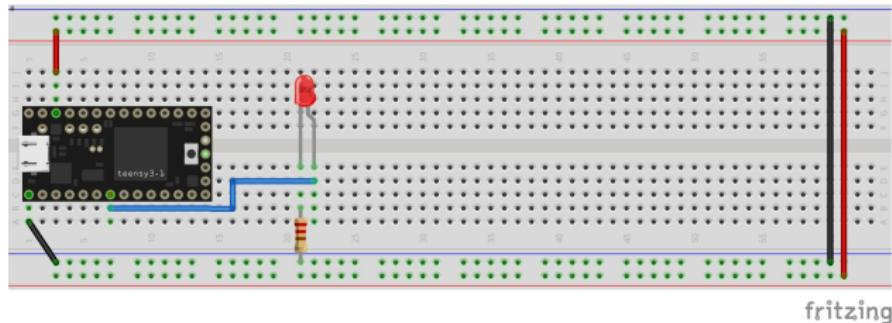
$$V_{pp} - V_{LED} = IR \implies R >= \frac{V_{pp} - V_{LED}}{I_{max}}$$

For a 3.3V power supply, a 0.43V across the LED, and a max current of 100mA, the resistor needs to be greater than 29Ω.



fritzing

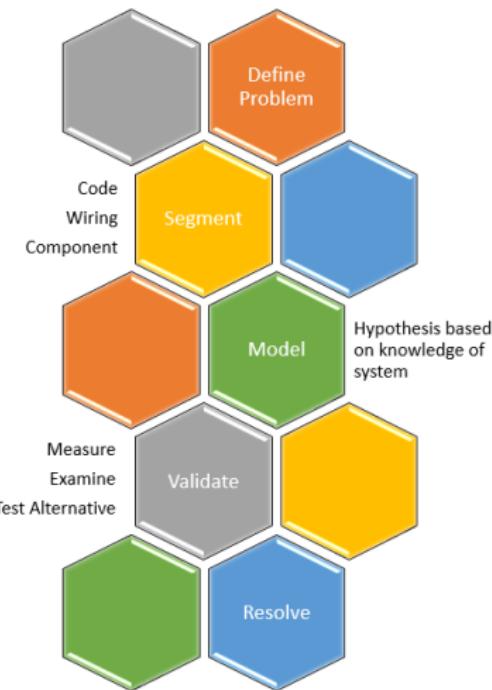
# Assignment L02\_01\_helloLED



- Using Pin 5 as an output and the appropriate current limiting resistor, blink the LED once per second.
- Measure the voltage at both leads of the LED and record in your notebook.
- Change the resistor to  $1k\Omega$  and then  $10k\Omega$ . What happens to the brightness? Measure the voltage and current in each case. Record in your notebook.

**REMEMBER:** Lab notebook, Fritzing, breadboard, then code

# Model Based Troubleshooting



# Constants and Variables

It is often useful to give a name to something that will be used repeatedly in the code. They can be constants or variables:

- Constant is a declaration that does not change throughout the code.  
For example a the pin that an LED is attached to.
- Variable is a declaration that changes as the code processes. For example, a counter or index.

The use of Constants and Variables has several advantages:

- It improves readability by assigning names to items
- Items can be changed by changing a single declaration
- It allows the code to do math

The first two Data Types that we will be using:

- int**: an Integer between  $\pm 2,147,483,648$ .
- float**: a Floating point number with 7-digits precision

# Constants and Variables Example

```
1 const int ledPin = 5;
2 const int ledDelay = 1000;
3 int i;
4
5 void setup() {
6     pinMode(ledPin, OUTPUT); // set ledPin as
    Output
7     i = 100;
8 }
9 void loop() {
10    digitalWrite(ledPin, HIGH);
11    delay(ledDelay);
12    digitalWrite(ledPin, LOW);
13    delay(ledDelay+i);
14    i = i + 100;
15 }
```

# Assignment L02\_02\_helloLEDvar



- Convert L02\_01\_helloLED with constants and/or variables.

# Pulse Width Modulation

Software Configurable:

- Digital Input: High/Low (3.3V/0V)
- Digital Output: High/Low (3.3V/0V)
- Analog Input: 0V to 3.3V
- Analog Output: 0V to 3.3V PWM



# Assignment L02\_03\_helloLEDanalog

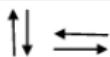


Use `analogWrite` to change the brightness of the LED, using values:

- 255
- 63
- 171
- 16

Measure the voltage with your multimeter at each value.

# Flowcharts

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

# Loops



# FOR Loop syntax

```
1 // FOR loop syntax
2 for (initialization; condition; increment) {
3     // statement(s);
4 }
5
6 // EXAMPLE
7 for (j=0; j <= 255; j++) {
8     analogWrite(ledPin, j);
9 }
```

# WHILE loop syntax

```
1 // WHILE loop syntax
2 while (condition) {
3     // statement(s)
4 }
5
6
7 // EXAMPLE
8 while (button == HIGH) {
9     digitalWrite(ledPin, HIGH);
10 } //continue this loop until button is released
```

# For vs While Loops

## For VS While Loop

Comparison Chart

For Loop	While Loop
The for loop is used for definite loops when the number of iterations is known.	The while loop is used when the number of iterations is not known.
For loops can have their counter variables declared in the declaration itself.	There is no built-in loop control variable with a while loop.
This is preferable when we know exactly how many times the loop will be repeated.	The while loop will continue to run infinite number of times until the condition is met.
The loop iterates infinite number of times if the condition is not specified.	If the condition is not specified, it shows a compilation error.

# Assignment L02\_04\_helloLEDtri



Using a FOR Loop, have the LEDs follow a Triangle Wave function from off to full brightness with a period of 10 seconds.

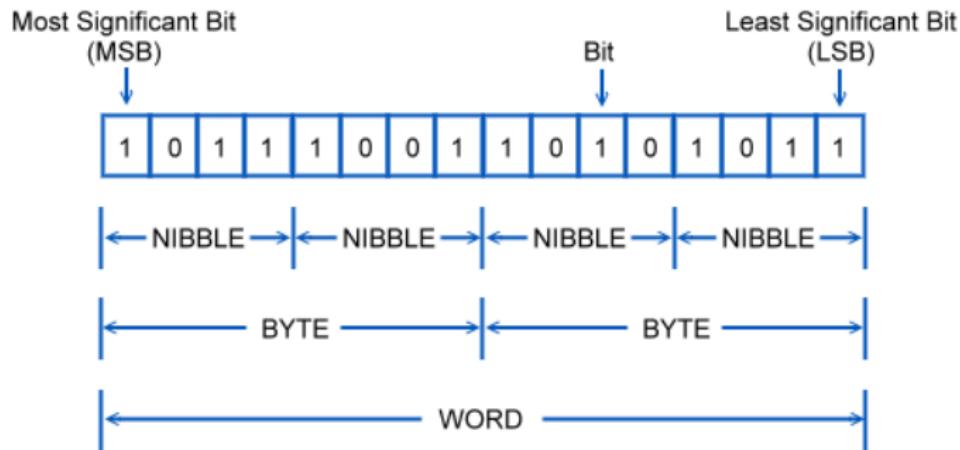
# While or Do While



# Number Systems



# Bits, Nibbles, Bytes, and Words



# Data Types: Numbers

Data Type	8-bit AVR systems (Arduino Uno)				32-bit ARM systems (Teensy 3.2)		
	bytes	range (signed)	range (unsigned)	bytes	range (signed)	range (unsigned)	
char	1	-128 to 127	0 to 255	1	-128 to 127	0 to 255	
short	2	+/- 32,767	0 to 65,353	2	+/- 32,767	0 to 65,353	
int	2	+/- 32,767	0 to 65,353	4	+/- 2,147,483,648	0 - 4,294,967,295	
long	4	+/- 2,147,483,648	0 - 4,294,967,295	4	+/- 2,147,483,648	0 - 4,294,967,295	
long long	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615	8	+/- 9,223,372,036,854,770,000	0 to 18,446,744,073,709,551,615	
float	4	3.4E +/- 38 (7 digits)	n/a	4	3.4E +/- 38 (7 digits)	n/a	
double	4	3.4E +/- 38 (7 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a	
long double	8	1.7E +/- 308 (15 digits)	n/a	8	1.7E +/- 308 (15 digits)	n/a	
Unambiguous							
uint8_t	1	n/a	0 to 255	1	n/a	0 to 255	
int08_t	1	-128 to 127	n/a	1	-128 to 127	n/a	
uint16_t	2	n/a	0 to 65,353	2	n/a	0 to 65,353	
int16_t	2	+/- 32,767	n/a	2	+/- 32,767	n/a	
uint32_t	4	n/a	0 - 4,294,967,295	4	n/a	0 - 4,294,967,295	
int32_t	4	+/- 2,147,483,648	n/a	4	+/- 2,147,483,648	n/a	

There are  $7.5 \times 10^{18}$  grains of sand on Earth. A long long integer and the floating point numbers are larger than this.

# Math Warning and Type Casting

Be cognizant of the data type when performing math operations.

```
1 int x = 3;
2 int y = 2;
3 float yf = 2.0;
4 float z;
5
6 //int divided by an int returns an int
7 z = x/y;                      // z = 1.0
8 z = x/yf;                     // z = 1.5
9 z = x / 2;                     // z = 1.0
10 z = x / 2.0;                  // z = 1.5
11
12 //type casting used to change datatype
13 z = (float) x / (float) y;    // z = 1.5
14 z = x / (float) y;           // z = 1.5
15
16 z = (int) (x / yf);         // z = 1.0
17 y = x / yf;                 // y = 1
```

Type Casting is a way to ensure that you are correctly moving between datatypes.

# Header Files

A header file is a file with the extension .h which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: those that the programmer writes and those that come with the compiler.

Both the user and system header files are included using the preprocessing directive #include. It has the following two forms:

- `#include <file.h>` for system header files.
- `#include "file.h"` for user created header files in the directory that contains the current code.

An example is the math.h header that defines various mathematical functions.

# Basic Structure of Arduino Sketch Revisited

```
1 #include <math.h>           // include header files
2 const int ledPin = 5;       // declare constants
3 float Vout;                // declare variables
4 float n;
5
6 void setup() {              // runs once
7     pinMode(13, OUTPUT);    // system settings
8     n = 0;                  // set variables
9 }
10
11 void loop() {               // loops indefinitely
12     Vout = sin(2*PI*n);
13     n = n+0.01;
14 }
```

# Assignment L02\_05\_helloLEDsin



Use a `sin()` function to vary brightness of your LED

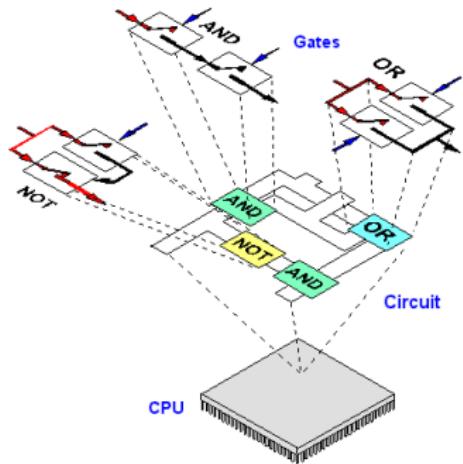
- Use `math.h`
- Function `sin` takes a double as an input and returns a double.
- Set the period to 5 seconds. One cycle of  $\sin(2\pi\nu)$  for each integer of n.

The function `millis()` returns milliseconds since Teensy has been powered on. The frequency ( $\nu$ ) is then defined by `millis() / period` in milliseconds. For example, for a period of 5 seconds, the frequency  $\nu = (\text{millis()}/5000.0)$ .

# IoT Fun



# Data Types: Boolean



Boolean datatype (bool)  
holds either a TRUE or  
FALSE

Boolean Logic Operations (condition statements)

- ① NOT (!): true if operand is false and visa-versa
  - $x = !x$
- ② AND (&&): true if both operands are true
  - $z = x \&\& y$
- ③ OR (||): true if either operand is true
  - $z = x || y$

# Boolean In Action

```
1 x = !x;           // invert x
2
3 if (!x) {        // if x is false
4   // statements
5 }
6
7 // if both pins read HIGH
8 if ((digitalRead(pin1) == HIGH) && (digitalRead(pin2) == HIGH) {
9   //statements
10 }
11
12 // if either value is greater than zero
13 if (x > 0 || y > 0) {
14   // statements
15 }
```

# Displaying to the Screen: The Serial Monitor

```
1 void setup() {  
2  
3 // Enable Serial Monitor  
4 Serial.begin (9600);  
5 while (!Serial); // wait for Serial monitor  
6 Serial.println ("Ready to Go");  
7 }  
8  
9 void loop() {  
10 for (i=0; i <=13; i++){  
11 Serial.print(i);  
12 delay(printDelay);  
13 }  
14 }
```

# Print Statements

- ① Serial.print() prints data to the monitor through the serial port as human-readable text:
  - Serial.print('N') prints: N
  - Serial.print("Hello World") prints: Hello World
  - Serial.print(78) prints: 78
  - Serial.print(3.141592) prints 3.14
  - Serial.print(3.141592,5) prints 3.14159
- ② Serial.println() displays the print() followed by a carriage return (\r) or newline (\n).
- ③ Serial.printf() displays a formatted print.

# Format Specifiers Statements

<b>specifier</b>	<b>Output</b>	<b>Example</b>
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

Serial.printf(" Print an integer %i and a float %0.4f \n", count, value);  
 Where count is an int and value is a float.

# Assignment L03\_00\_SerialMonitor



- ① Print Hello World to your monitor screen.
- ② Next, display to the screen a count from 0 to 13, separated by commas, by using:
  - Serial.print();
  - Serial.println();
  - Serial.printf();

# Hello World - What a microcontroller sees

## HelloWorld.ino

```
1 void setup() {  
2   Serial.begin(9600);  
3   Serial.printf("Hello World! \n");  
4 }  
5  
6 void loop() {}
```

## Hex Code and Assembly Language

```
1 HelloWorld.bin:      file format binary  
2 Disassembly of section .data:  
3 00000000 <.data>:  
4     101c: bd10      pop {r4, pc}  
5     101e: 4402      add r2, r0  
6     1020: 4603      mov r3, r0  
7     1022: 4293      cmp r3, r2  
8     1024: d002      beq.n 0x102c  
9     1026: f803 1b01  strb.w r1, [r3], #1  
10    102a: e7fa      b.n 0x1022  
11    102c: 4770      bx lr  
12    102e: 0000      movs r0, r0  
13    1030: b538      push {r3, r4, r5, lr}
```

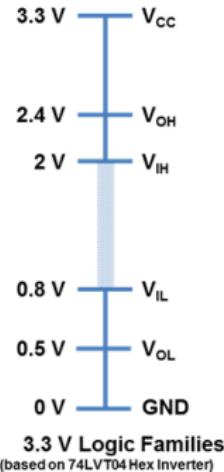
# One Pin - Many Functions



Software Programmable: Input or Output and Digital or Analog.

# Digital Input/Output

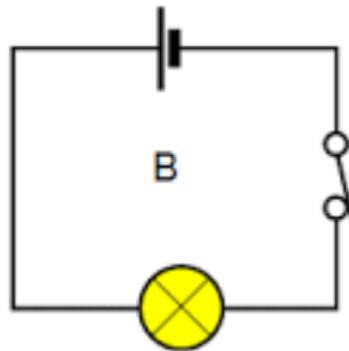
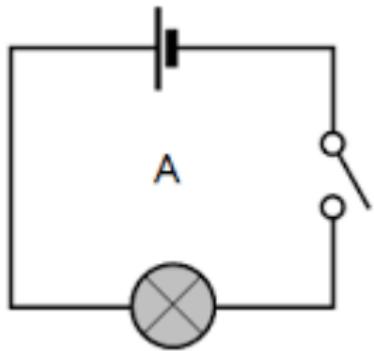
Digital electronics rely on binary logic to store, process, and transmit data or information. Binary Logic refers to one of two states – ON or OFF. This is commonly translated as a binary 1 or binary 0. A binary 1 is also referred to as a HIGH signal and a binary 0 is referred to as a LOW signal.



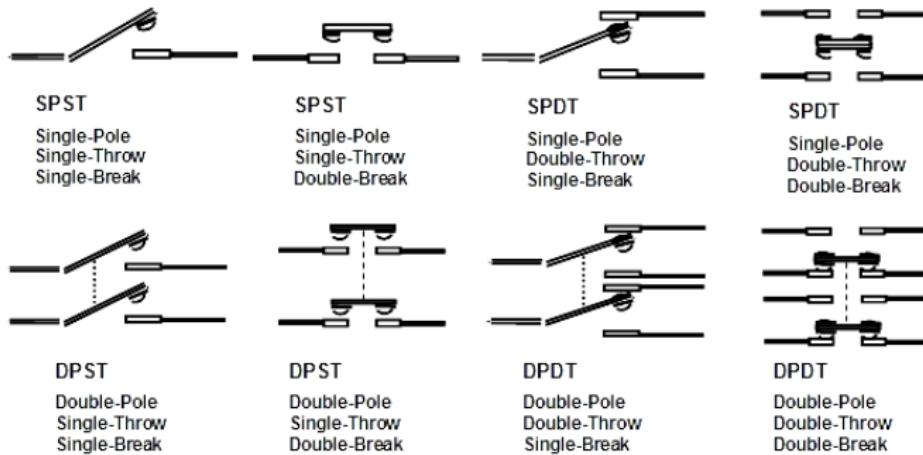
- `digitalWrite(pin,value);`
- `inputValue = digitalRead(pin);`

where, value equals HIGH or LOW.

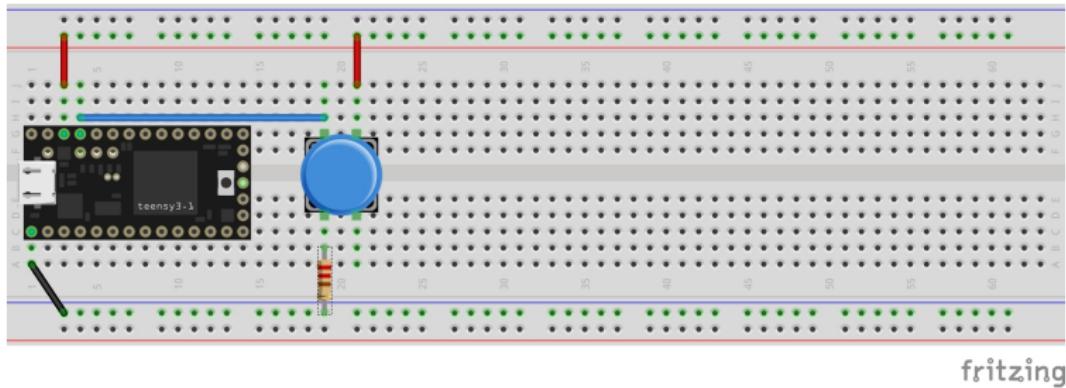
# Switches



# Types of Switches



# Our First Button and Pull Down Resistors



fritzing

Wait - why is there a resistor connected to ground?

# Assignment: Buttons



- Labbook: draw circuit
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L03\_01\_button

- Connect button (Pin 23)
- Print button state to the screen.
- Now, remove the pull-down resistor and restart your code.

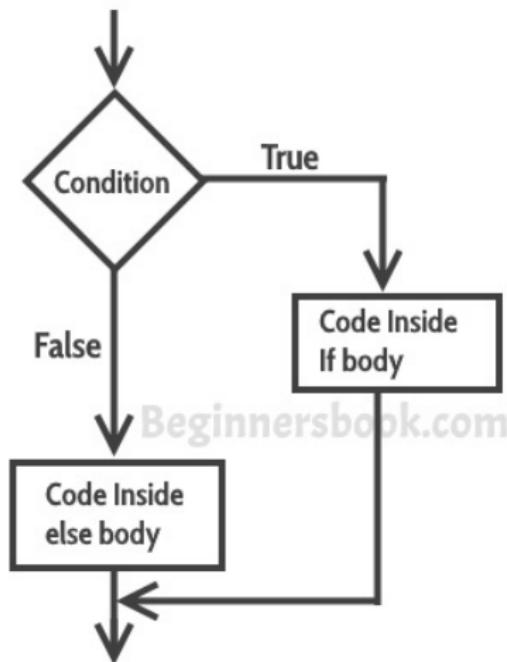
## ② L03\_02\_button\_pullup

- Replace the pull-down resistor with a pull-up resistor.
- Not pressed: 3.3V
- Pressed: GND
- How does the logic change?

## ③ L03\_03\_button\_input\_pullup

- Remove the pull-up resistor
- Implement:  
`pinMode(pin,INPUT_PULLUP);`

# IF-ELSE Statements



Beginnersbook.com

# IF-ELSE Statements

```
1 // IF statement SYNTAX
2 if (condition) {
3     //statement(s)
4 }
5 else {
6     // else statement(s)
7 }
8
9 // EXAMPLE
10 if (button == HIGH) {
11     Serial.printf("Button is not pressed \n");
12 }
13 else {
14     Serial.printf("Button is pressed \n");
15 }
```

# Button and LED



# Assignment: Buttons and LEDs



## ① L03\_04\_buttonLED

- Add an LED to Pin 5 and use the button to turn the LED on and off.
- Also, print button state to the screen

## ② L03\_05\_twobuttonLED

- Add a second button (Pin 16) and LED (Pin 6)
- Have each button control one LED
- Also, print button states to the screen

# Voltage Divider

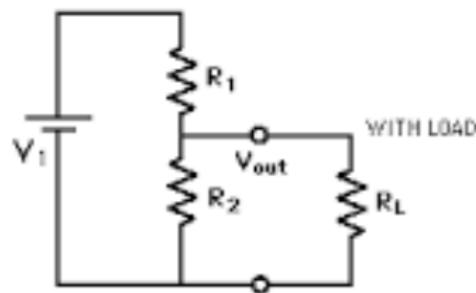
OPEN CIRCUIT BEHAVIOR



$$V_{out} = V_1 \frac{IR_2}{I(R_1 + R_2)} = \frac{V_1 R_2}{(R_1 + R_2)}$$

for open circuit

BEHAVIOR UNDER LOAD



$$V_{out} = \frac{V_1(R_2||R_L)}{(R_1 + R_2||R_L)}$$

for loaded circuit

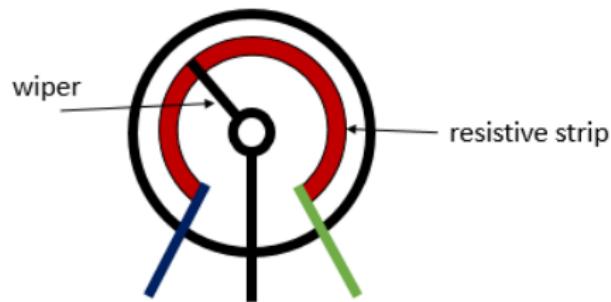
# Voltage Dividing



We are just using the Teensy to provide Power and GND.

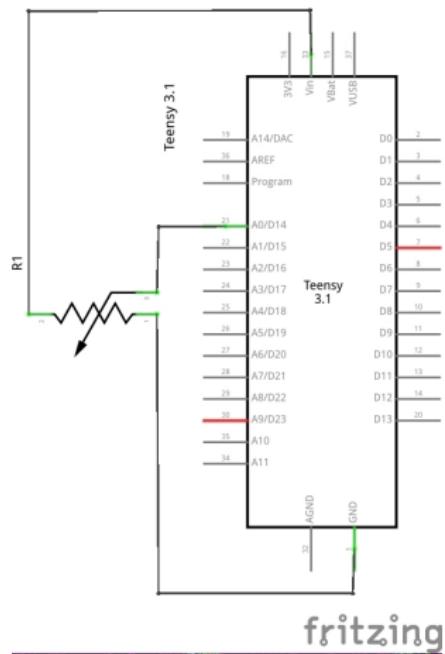
- Use various combinations of resistors between  $1k\Omega$  and  $22k\Omega$ .
- Calculate the Series resistance and the voltage between the two resistors in your Lab Notebook.
- Measure with your multimeter and compare.

# Potentiometer - Variable Resistor



A potentiometer has 3 pins. Two terminals (the blue and green) are connected to a resistive element and the third terminal (the black one) is connected to an adjustable wiper.

# Assignment L03\_06\_AnalogInput



- Utilize `analogRead()` to measure analog input across potentiometer (voltage divider) using Pin 14.
- Determine the range of the `analogRead` across the entire range of the potentiometer.

# Anatomy of a Function

## Anatomy of a C function

Datatype of data returned,  
any C datatype.

"void" if nothing is returned.

Parameters passed to  
function, any C datatype.

```
int myMultiplyFunction(int x, int y){  
    int result;  
    result = x * y;  
    return result;  
}
```

Function name

Return statement,  
datatype matches  
declaration.

Curly braces required.

# Types of Variables

```
1 int toadStools = 20;           // global variable
2
3 int seatsLeft(int toadStools) {
4     int frogs = 8;             // local variable
5     static int toads = 2;      // static variable
6
7     frogs = frogs - 2;
8     toads = toads + 1;
9
10    seatsLeft = toadStools - frogs - toads;
11    return seatsLeft;
12 }
```

## ① Global Variables

- Accessible throughout the program and all functions.

## ② Local Variables

- Accessible only in the function
- Created when function is called, destroyed when function is returned.

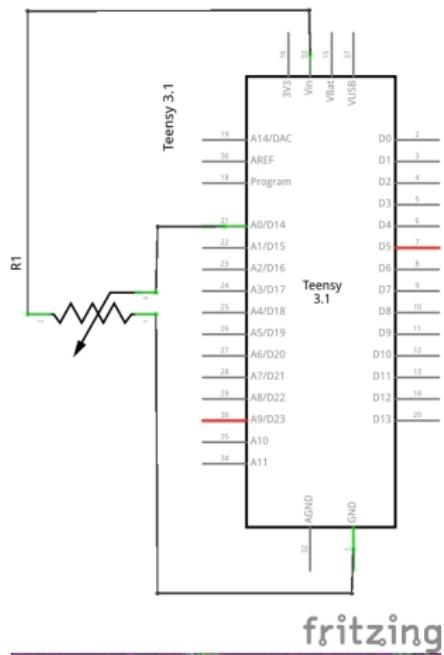
## ③ Static Local Variables

- Accessible only in the function
- Maintains value across multiple calls of a function.
- Destroyed only when program is terminated.

# Basic Structure of Arduino Sketch Revisited

```
1 int Num, doubleNum;
2
3
4 void setup() {
5     Serial.begin(9600);    // Turn on Serial Monitor
6     while(!Serial);        // Wait for Serial Monitor to be running
7     Num = 1;                // Initialize Num;
8 }
9
10 void loop() {
11     doubleNum = twotimes(Num); // call the funciton twotimes
12     Num = doubleNum;
13     Serial.printf("The number is now %i \n", Num);
14     Serial.printf("The number in HEX is %X \n", Num);
15     delay(1000);
16 }
17
18 int twotimes(int number) {
19     int answer;          // declare answer as a local variable
20     answer = 2 * number;
21     return answer;
22 }
```

## Assignment L03\_06\_AnalogInput Revisited



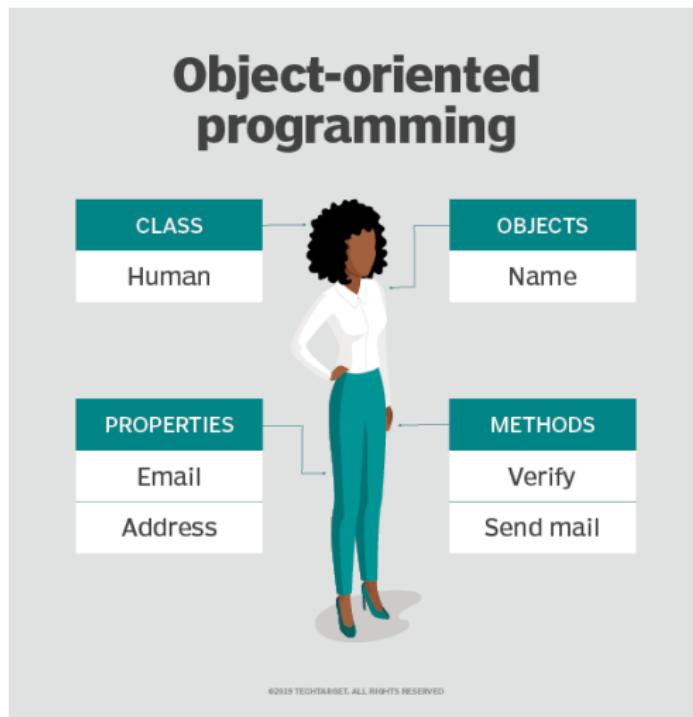
- ① Modify your code by adding a function, `in2volts()`, that converts the analog input value to voltage.
  - ② Print both the raw `analogInput` and the associated voltage to your screen.

# IoT Humor



*"I remember when you could only lose a chess game to a supercomputer."*

# Objects

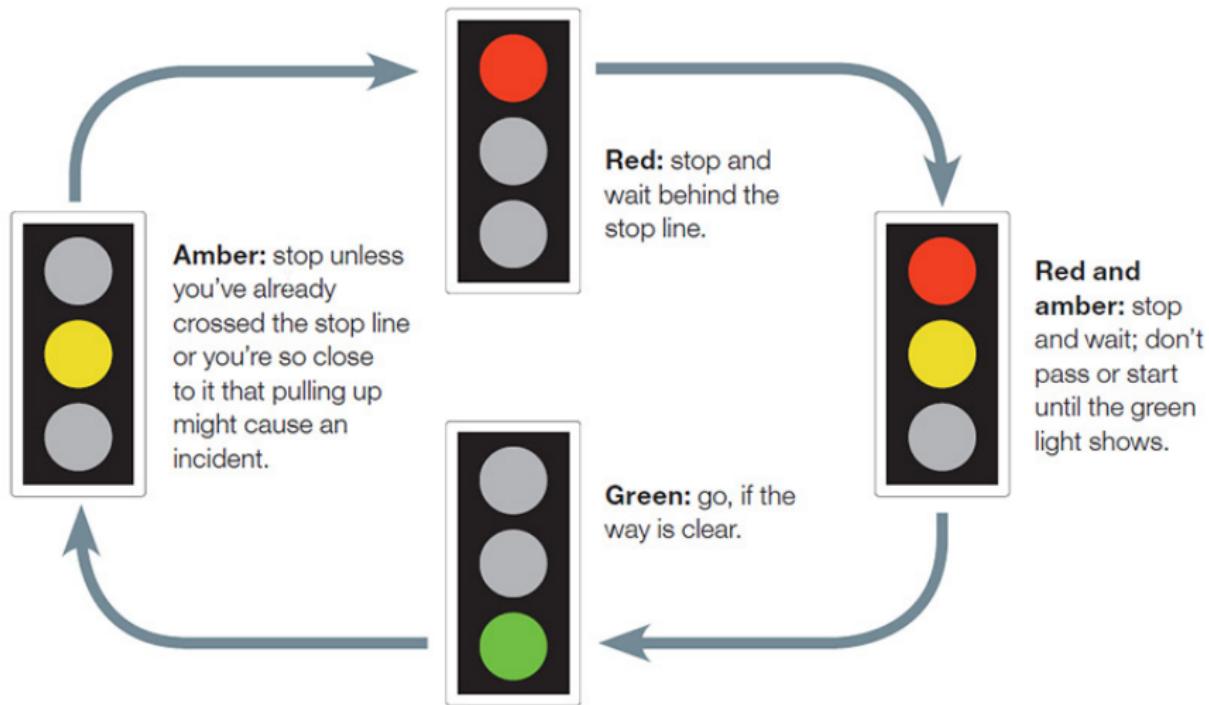


# Traffic Light



Let's use the traffic light to build our own Objects

# State Machine - Traffic Lights, British Style



# SWITCH...CASE syntax - multiple IFs

```
1 // SWITCH...CASE syntax
2 switch (var) {
3     case label1:
4         // statements
5         break;
6     case label2:
7         // statements
8         break;
9     default:
10        // statements
11        break;
12 }
```

Where:

- var: variable whose value is compared to the case values
- label1, label2 are the case values (int or char)

## Enumeration (enum)

The C-language has a declaration type, enum, which allows for multiple states.

- Within the enum declaration descriptive tags are used
- Then the compiler assigns the tags an integer value.

```
1 // ENUM example:  
2 // A variable State with four states  
3 enum State {  
4     GREEN,  
5     YELLOW,  
6     RED,  
7     RED_YELLOW  
8 };
```

The compiler treats enum as your personal variable type. For example, the enum variable (e.g., State) can now be used within switch...case statements.

# OneButton Library



The tick() method checks the input pin for a single click, double click or long press situation.

# Basic Structure of Arduino Sketch - Revisited

```
1 // the "header" is used for GLOBALS
2 #include <bme280.h> // library files
3 #include <Adafruit_SSD1306.h>
4 Adafruit_BME280 bme; // name object in class
5 Adafruit_SSD1306 display(WIDTH, HEIGHT, &Wire);
6 bool onoffState;      // declare global variables
7
8 void setup() {
9     display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
10    bme.begin(0x76);           // begin processes
11    onoffState = false;        // set variables
12 }
13
14 void loop() {
15 }
```

# OneButton Declarations

```
1 #include <OneButton.h>
2 OneButton button1(pin, activeLOW, pullUP);
3 void setup() {
4     button1.attachClick(click1);
5     button1.attachDoubleClick(doubleclick1);
6     button1.attachLongPressStart(longPressStart1);
7     button1.attachLongPressStop(longPressStop1);
8     button1.attachDuringLongPress(longPress1);
9     button1.setClickTicks(250);
10    button1.setPressTicks(2000);
11 }
```

OneButton parameters (not variables):

- pin: the pin the button is connected to.
- activeLOW: "true" means input LOW when button pressed.
- pullUP: "true" is INPUT\_PULLUP pinMode.

# Using OneButton

```
1
2 void loop() {
3     button1.tick();      // check the state of the
4         button
5
6 void click1() {
7     Serial.println("Hi, my name is Brian.");
8 }
9
10 void doubleClick1() {
11     Serial.println("Hope your day is going well.");
12 }
```

# Assignment: OneButton



- Notebook: flowchart
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L04\_01\_oneButton

- Use OneButton libary and button on Pin 23.
- Click() - toggle bool variable buttonState.
- doubleClick() - toggle bool variable blinker.

## ② L04\_02\_oneButtonLED

- Toggle LED on/off with buttonState.

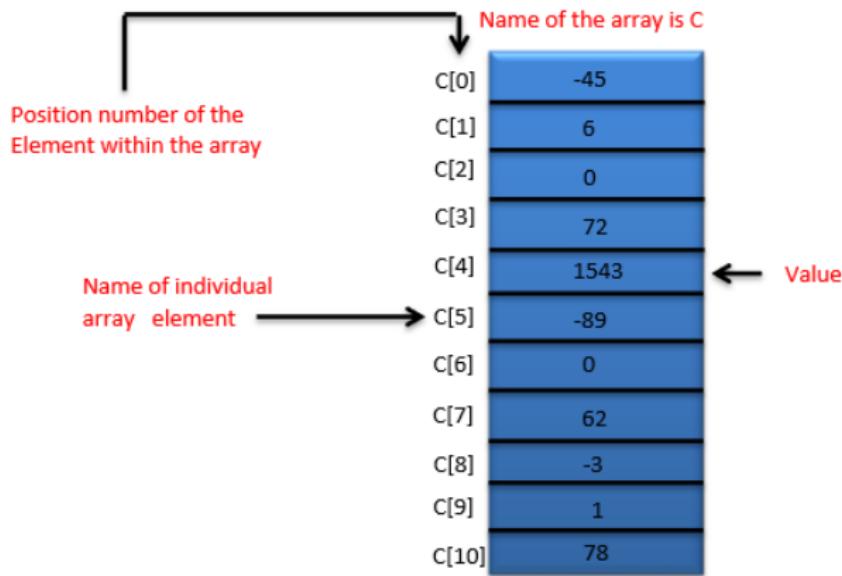
## ③ L04\_03\_oneButtonLEDblink

- When ON, change from solid to blinking when the variable blinker is toggled.

# Avoiding Delays

```
1 void loop() {
2     //run constantly
3     currentTime = millis();
4
5     //run once per second
6     if((currentTime - lastSecond) > 1000) {
7         Serial.print(".");
8         lastSecond = millis();
9     }
10
11    //run once per minute
12    if((currentTime - lastMinute) > 60000) {
13        Serial.println();
14        Serial.println("Minute");
15        lastMinute = millis();
16    }
```

# Arrays



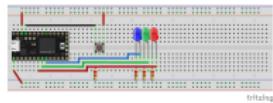
- Syntax: datatype var[ ] = {element 1, element 2, element 3};
- Example: int ledArray[ ] = {greenPin, yellowPin, redPin};

# Using Arrays

```
1 int myInts[6];
2 int myPins[] = {2, 4, 8, 3, 6};
3 int mySensVals[6] = {2, 4, -8, 3, 2};
4 char message[6] = "hello";
5
6 void loop() {
7     mySensVals[0] = 10; //assign value to array
8     x = mySensVals[4]; //retrieve value from array
9     for (i = 0; i < 5; i = i + 1) {
10         Serial.println(myPins[i]);
11     }
12 }
```

**NOTE:** The array index starts at 0 (not 1).

# Assignment: OneButton



- Notebook: flowchart
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L04\_04\_oneButtonArray

- Use 3 LEDs and one Button
- Click() - toggle current LED on/off
- doubleClick() - using an array, select the next LED
- longPressStart() - light up the three LEDs in sequence
- longPressStop() - light up the three LEDs in reverse order

# Assignment: Serial Read

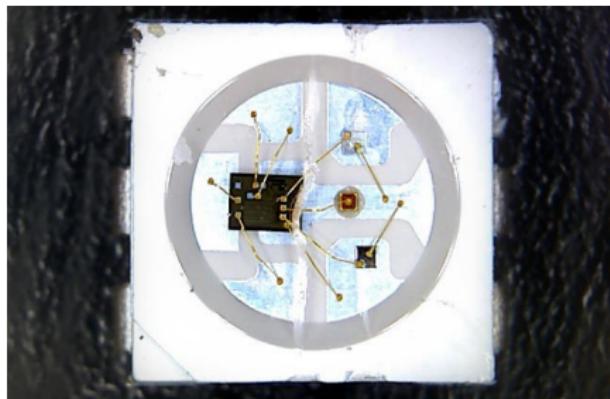


- SerialReadExample.ino

## ① L04\_05\_timer - Create a Stop Watch and Countdown Timer

- Click for start and stop
- Double Click to switch between Stop Watch and Timer. In Timer mode, prompt the user on the Serial Monitor to enter the time to countdown from.
- Long Press for reset

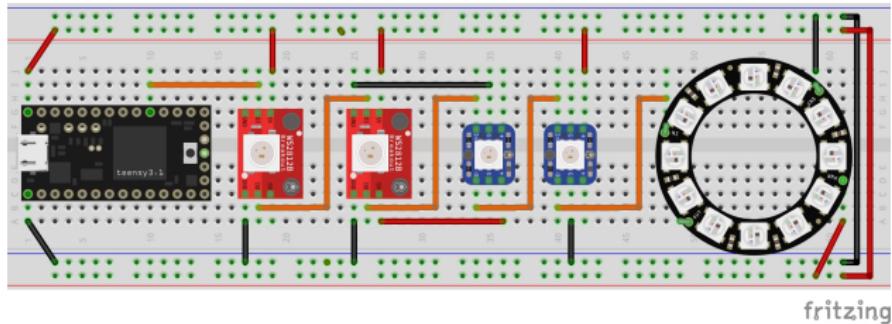
# NeoPixels



NeoPixels are:

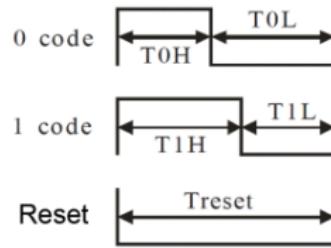
- Addressable RGB LEDs based on the WS2812 (or WS2811) LED/drivers.
- They come as individual pixels, in strips, in matrices, rings, etc.
- They can be programmed via your microcontroller to create a wide array of effects and animations.

# NeoPixel Programming

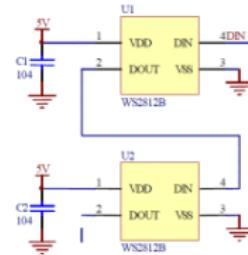


fritzing

WS2812 Protocol



LED-Chain



# NeoPixel Declaration

Including the NeoPixel library and setting up the object: pixel.

```
1 #include <Adafruit_NeoPixel.h>
2
3 const int LED_PIN = 17;      // Pin the NeoPixels are connected to
4 const int LED_COUNT = 16;    // Total number of NeoPixels
5
6 Adafruit_NeoPixel pixel(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);
7 /* Argument 1 = Number of pixels
8 * Argument 2 = GPIO pin number
9 * Argument 3 = Pixel type flags, add together:
10 *   You will use:
11 *     NEO_GRB      Pixels are wired for GRB bitstream (most NeoPixel products)
12 *     NEO_KHZ800   800 KHz bitstream (WS2812)
13 *
14 * Other options for Argument 3:
15 *     NEO_KHZ400   400 KHz (WS2811)
16 *     NEO_RGB      Pixels are wired for RGB bitstream (v1)
17 *     NEO_RGBW     Pixels are wired for RGBW bitstream
18 */
```

# Using NeoPixel Methods

```
1 void setup() {  
2     pixel.begin();  
3     pixel.show(); //initialize all off  
4 }  
5  
6 void loop() {  
7     pixel.setPixelColor(n, red, green, blue);  
8     pixel.setPixelColor(n, color); \\hex code  
9     pixel.fill(color, first, count);  
10    pixel.setBrightness(bri) \\ 0 - 255  
11    pixel.show(); \\nothing changes until show()  
12    pixel.clear();  
13    pixel.show();  
14 }
```

# Assignment: NeoPixels



- Notebook: flowchart
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L05\_01\_neoPixel

- Light up the 4 pixels and the ring using a FOR-loop.

## ② L05\_02\_colorHeader

- Implement a header file that contains the pixel colors.

## ③ L05\_03\_pixelStrip, using FOR-loop, implement functions that:

- Send a pixel of a random color down and back on the strip
- Light the strip up as a rainbow
- Send a pair of Maize and Blue lights down the strip.

## ④ L05\_04\_pixelFill

- Light up 6 segments of different colors using the fill() method

# Where do global header files go?

The screenshot shows a Windows File Explorer window with the following details:

- File Explorer Title Bar:** Libraries
- Menu Bar:** File, Home, Share, View
- Toolbar:** Pin to Quick access, Copy, Paste, Cut, Copy path, Move to, Copy to, Delete, Rename, New folder, New item, Easy access, Properties, Open, Select all, Select none, History, Invert selection, Select.
- Address Bar:** This PC > Local Disk (C:) > Users > IoT\_Instructor > Documents > Arduino > libraries
- Left Sidebar:** Quick access, Desktop, Downloads, Documents, Pictures, IoT, class\_slides, instructor\_guide, Messages, Workflow, Creative Cloud Files, OneDrive, This PC, 3D Objects, Desktop, Documents, Downloads, Music, Pictures, Videos.
- Table View:** A list of folders and files in the "libraries" folder.

Name	Date modified	Type
ACROBOTIC_SSD1306	3/11/2020 1:45 PM	File folder
Adafruit_ADXL343	3/3/2020 9:27 AM	File folder
Adafruit_BME280_Library	3/3/2020 9:27 AM	File folder
Adafruit_BusIO	7/20/2020 2:00 PM	File folder
Adafruit_GFX_Library	7/20/2020 2:00 PM	File folder
Adafruit_NeoPixel	10/12/2020 10:17 AM	File folder
Adafruit_PWM_Servo_Driver_Library	8/3/2020 10:09 AM	File folder
Adafruit_SSD1306	7/20/2020 2:00 PM	File folder
Adafruit_Unified_Sensor	3/3/2020 9:27 AM	File folder
colors	3/5/2020 11:23 AM	File folder
DS1307RTC	8/3/2020 3:12 PM	File folder
Grove_-_Air_quality_sensor	7/31/2020 1:25 PM	File folder
hue	7/22/2020 10:23 AM	File folder
hue2	8/3/2020 2:23 PM	File folder
mac	3/12/2020 12:45 PM	File folder
OLED_SSD1306_Chart	8/3/2020 12:15 PM	File folder
OneButton	3/3/2020 9:26 AM	File folder
RTC	8/3/2020 3:12 PM	File folder
RTClib_by_NeiroN	8/3/2020 9:41 AM	File folder
wemo	7/16/2020 7:48 AM	File folder
wemoObj	7/20/2020 10:21 AM	File folder
readme	2/18/2020 9:32 AM	Text Document
		1 KB

# Generating Random Numbers

```
1  /*
2   * The random function generates pseudo-random
3   * numbers.
4   *     random(min,max)
5   *     random(max)      //assumes min = 0
6   * returns a number between min and max-1
7   */
8
9  // print a random number from 0 to 299
10 randNumber = random(300);
11 Serial.println(randNumber);
12
13 // print a random number from 10 to 19
14 randNumber = random(10, 20);
15 Serial.println(randNumber);
```

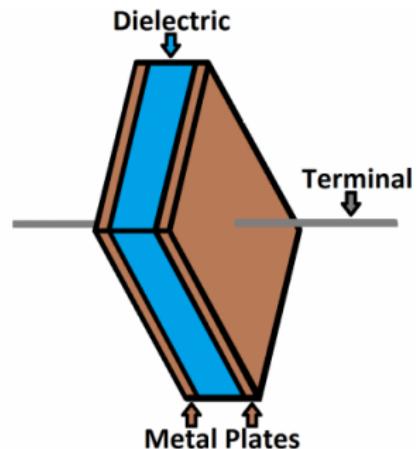
# IoT Humor



# Capacitors

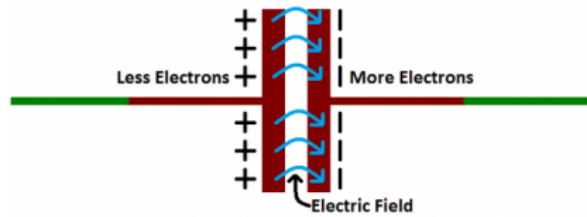
A capacitor is created out of two metal plates and an insulating material called a dielectric. The metal plates are placed very close to each other, in parallel, but the dielectric sits between them to make sure they don't touch.

- The dielectric can be made out of all sorts of insulating materials: paper, glass, rubber, ceramic, plastic, or anything that will impede the flow of current.
- The plates are made of a conductive material: aluminum, tantalum, silver, or other metals.



# Capacitors

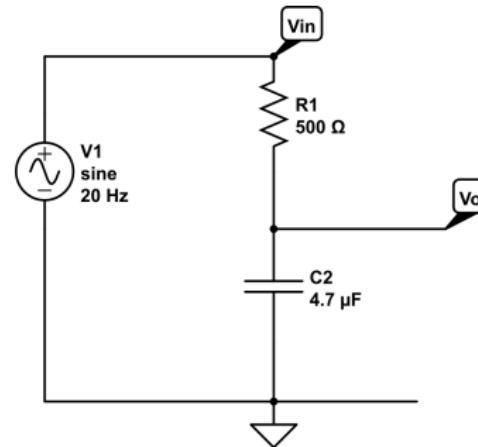
When current flows into a capacitor, the charges get "stuck" on the plates because they can not get past the insulating dielectric. Electrons build up on one of the plates, and it becomes overall negatively charged. The large amount of negative charges pushes away like charges on the other plate, making it positively charged.



The stationary charges on these plates create an electric field, which influences electric potential energy and voltage. When charges group together on a capacitor like this, the cap is storing electric energy just as a battery might store chemical energy.

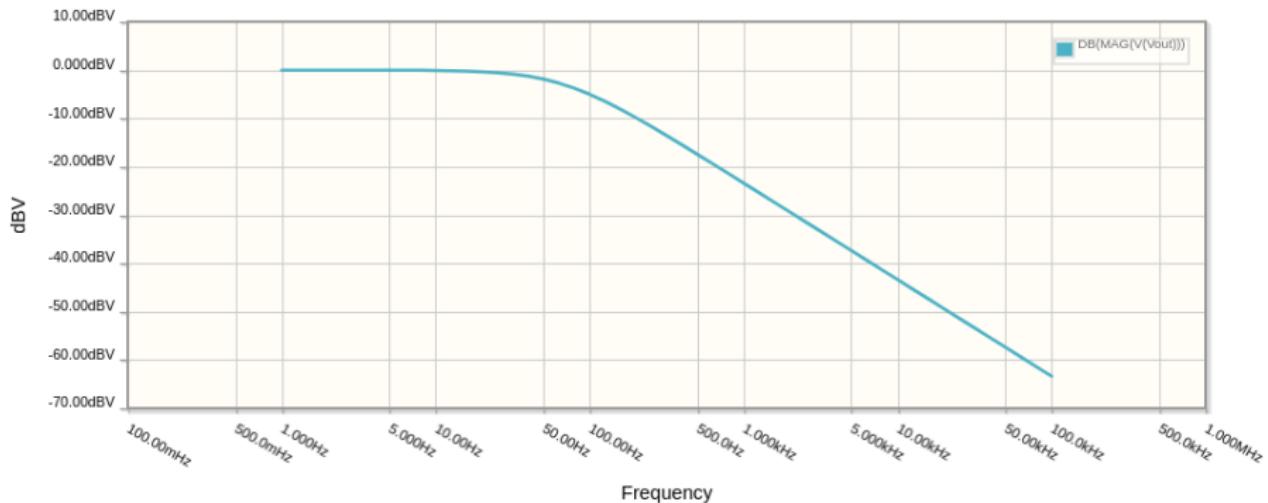
# Low Pass Filter - cutoff frequency $f_c$

- At low frequencies, there is plenty of time for the capacitor to charge up to practically the same voltage as the input voltage.
- At high frequencies, the capacitor only has time to charge up a small amount before the input switches direction. The output goes up and down only a small fraction of the amount the input goes up and down. At double the frequency, there's only time for it to charge up half the amount.



$$f_c = \frac{1}{2\pi\tau} = \frac{1}{2\pi RC}$$

# Low Pass Filter Response

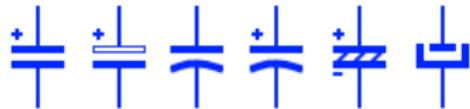


$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi(500)(4.7 \times 10^{-6})} = 67.5678\text{Hz}$$

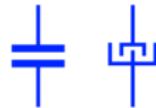
# Capacitors - does it matter how they are placed

- Some types of capacitors (electrolytic and tantalum) are polarized (they have + and - terminals). This is due to how the dielectric film has been deposited, the reverse polarity leads to degradation of the dielectric.
- Other capacitors (ceramic and film) do not have a polarity and can be installed in either direction.

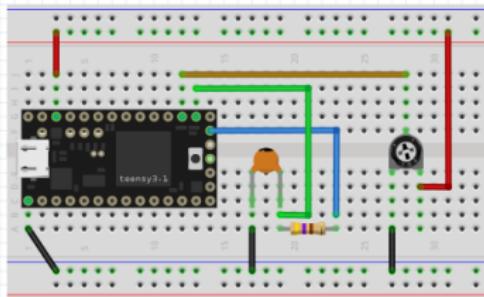
Polarized Electrolytic Capacitor



Generic Capacitor



# Assignment: Low Pass Filters



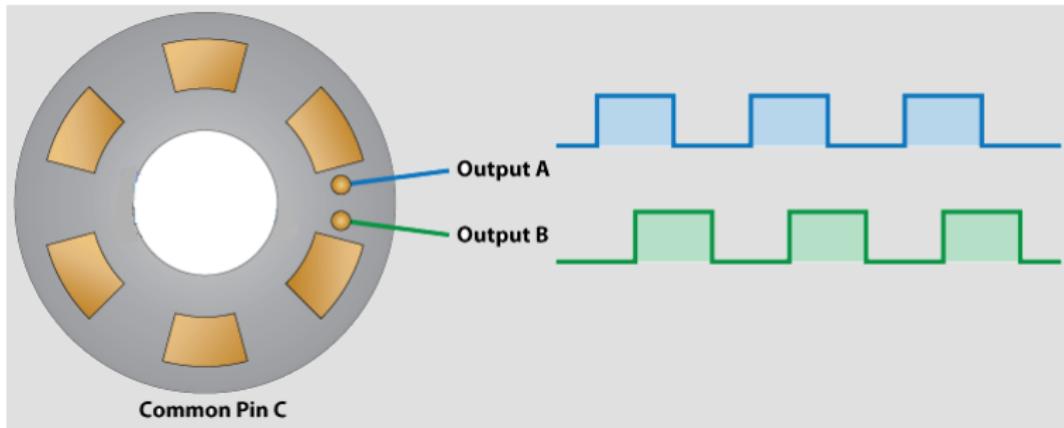
## ① L06\_00\_lowPass

- Create code that generates a sine wave of frequency  $\nu$ :  $\sin(2\pi\nu t)$
- Connect the output to an input
- Using the Serial Plotter, plot both the output and the input.
- Create a low pass filter with  $f_c \approx 67\text{Hz}$
- Pass the output through the low pass filter before inputting back to the Teensy.
- In code, vary the frequency and observe the difference between the two signals.
- Use the potentiometer to now vary the frequency.

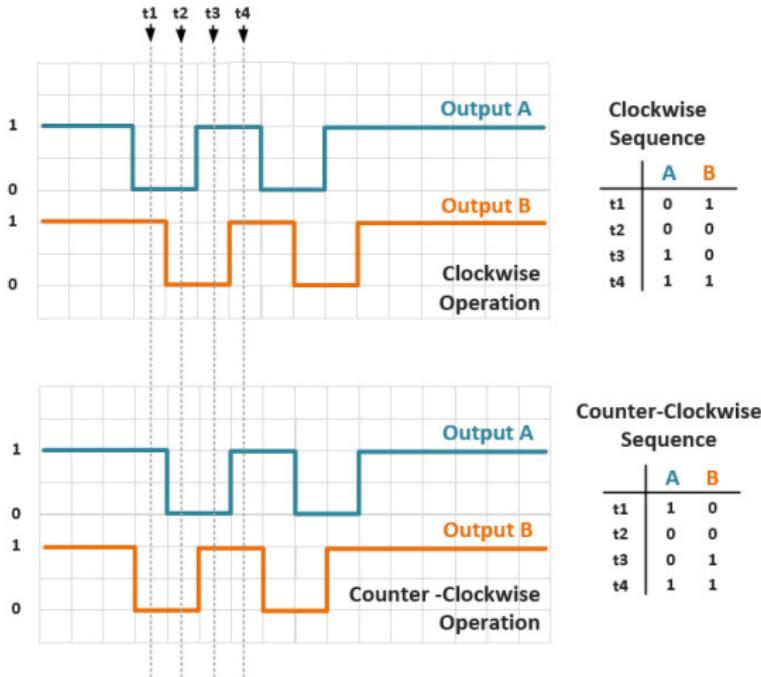
Using Serial Plotter:

- `Serial.begin(9600);`
- `Serial.printf("%i , %i, %0.3f \n", d1,d2,d3);`
- Close Serial Monitor

# Encoders



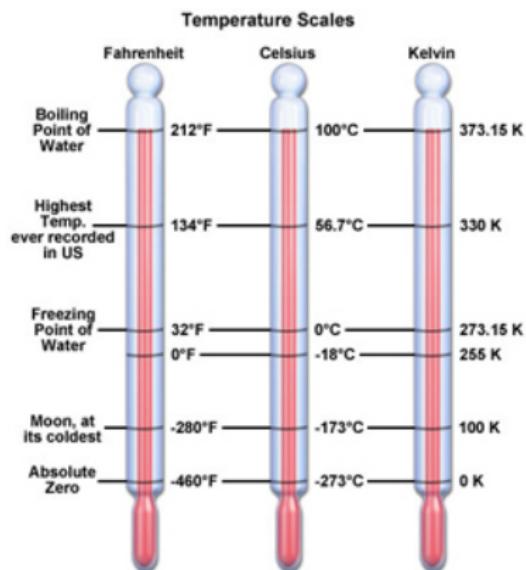
# Encoders



# The Encoder Class

```
1 #include <Encoder.h>
2 Encoder myEnc(pinA, pinB);
3
4 void setup() {
5 }
6
7 void loop() {
8     // read encoder position
9     position = myEnc.read();
10
11    // set encoder to a position
12    myEnc.write(maxPos);
13 }
```

# Mapping (or Converting)



Mapping is the conversion from one set of units to another. For example converting from Celsius to Fahrenheit:

$$Temp(^{\circ}F) = \frac{9}{5} * Temp(^{\circ}C) + 32$$

C++ provides us with a function to do this mapping:

```
newVal = map(value, fromLow, fromHigh, toLow, toHigh);
```

For example:

```
tempF = map(tempC,0,100,32,212);
```

# Assignment: Encoders



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L06\_01\_encoder

- Display the encoder position to the screen

## ② L06\_02\_encoderScaled

- The encoder has 96 positions. Mathematically map (without using the map() function) the encoder to 12 pos ( $0-7 = 0, 8-15 = 1$ , etc.). Show your work to an instructor before moving on to the map() function.
- Next, use the map() function.

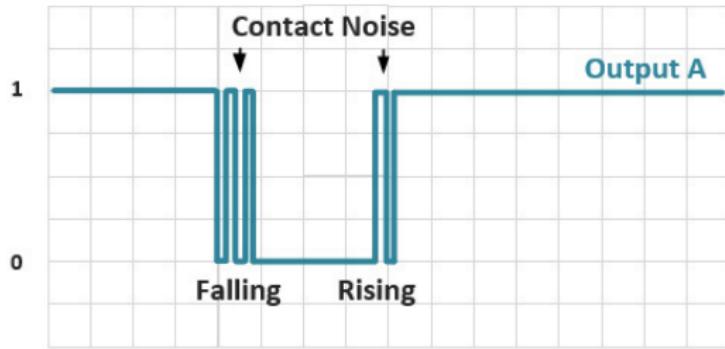
## ③ L06\_03\_encoder\_NeoPixel

- Use the encoder to light up the pixel ring

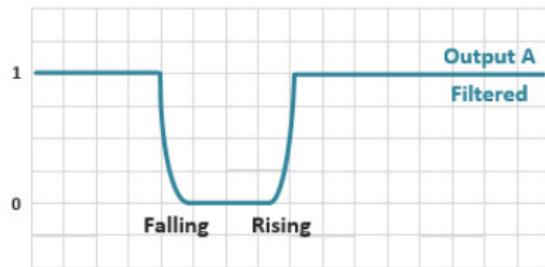
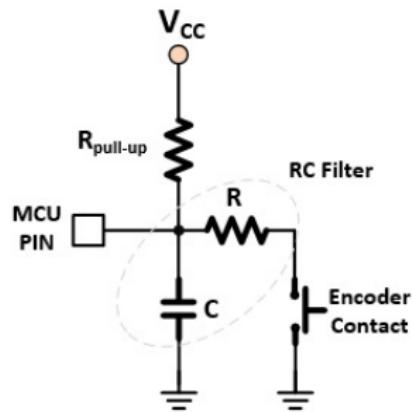
Reminder of the syntax of the map() function:

`newVal = map(value, fromLow, fromHigh, toLow, toHigh)`

# Encoder Jitter



# Encoder - Low Pass Filter



# Assignment: Encoders



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L06\_04\_encoder\_switch

- Connect your microcontroller to the encoder switch and LEDs
- Use the switch to turn on/off the NeoPixels.
- Also, the encoder LED should be red for off and green for on.

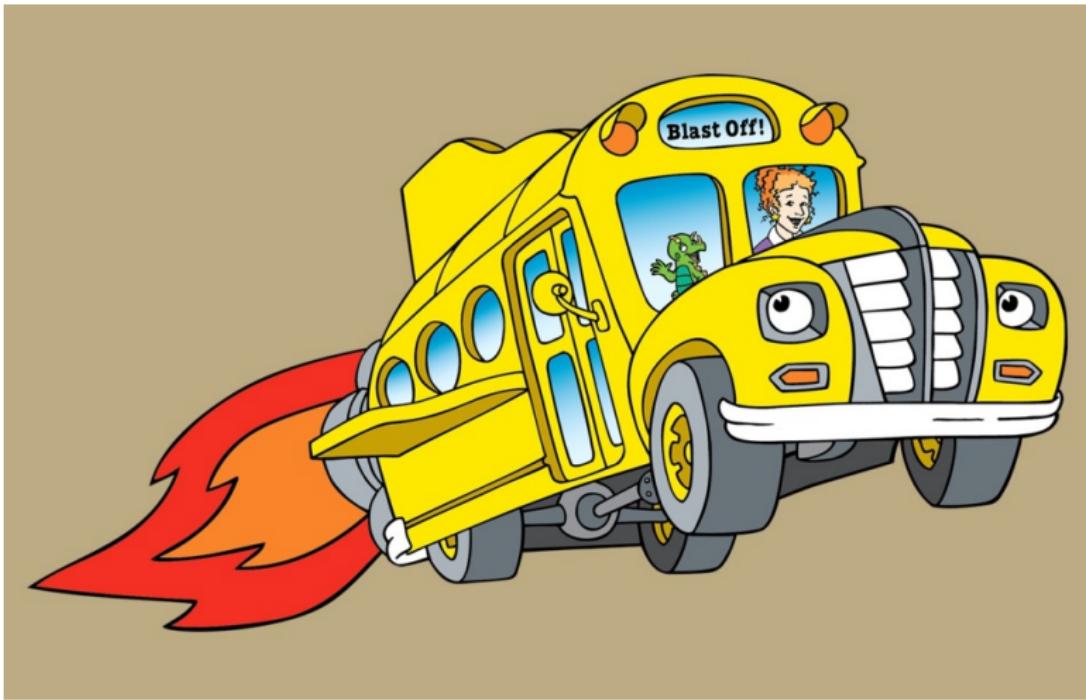
## ② L06\_05\_rainbow1 - extra credit

- Without OneButton: Use a button to cycle the NeoPixel ring colors through the colors of the rainbow. (i.e., one color change each time button is pressed).

## ③ L06\_06\_rainbow2 - extra credit

- With OneButton: Have it continuously cycle (i.e, while pressed colors change every one second).

# Buses and Interfaces

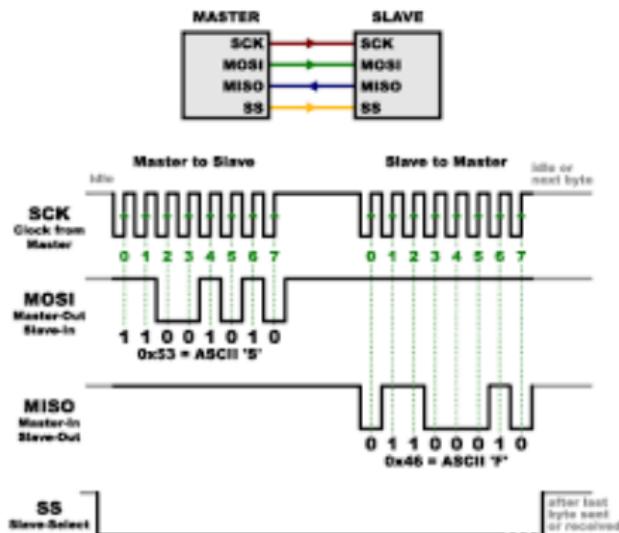


# UART



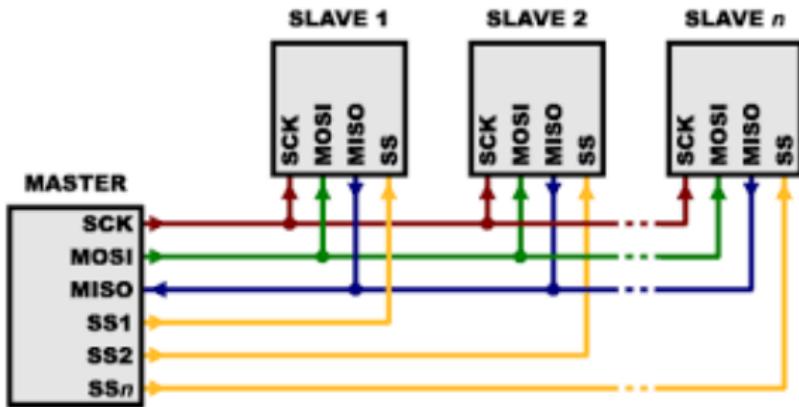
Universal Asynchronous Receiver/Transmitter

# Serial Peripheral Interface



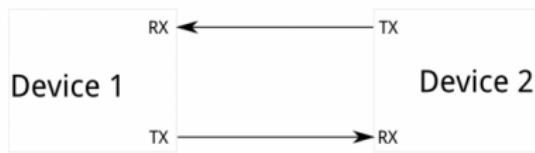
- Master Out, Slave In (MOSI) connects to Data In
- Master In, Slave Out (MISO) connects to Data Out

# Serial Peripheral Interface

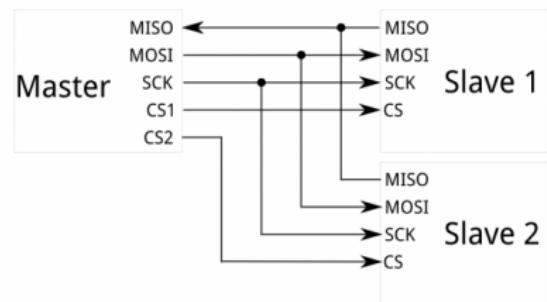


# Serial +/−

## UART



## SPI



# Assignment: L07\_01\_dataLogger



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

- ➊ The starter code details the pin assignments for SPI, use these for your schematic and Fritzing.
- ➋ Modify the starter code to:
  - Read two inputs:
    - ➌ The value of the encoder (unbounded)
    - ➍ The Pin 22, left floating (connected to nothing)
  - Write to  $\mu$ SD Card a timestamp and the two input values every 5 seconds.

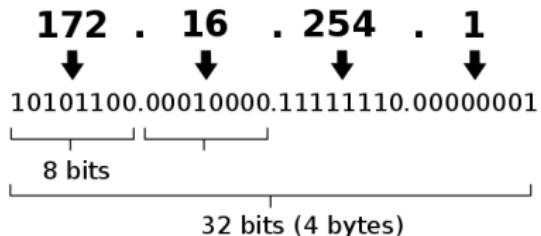
# The Internet



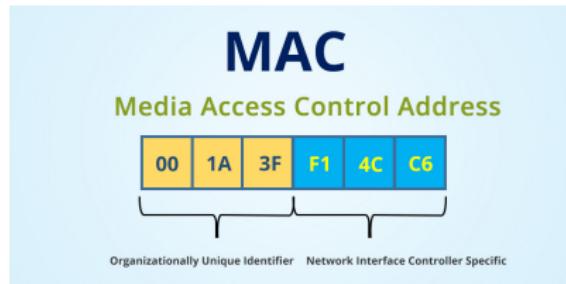
## IP Addresses

- When a device joins the network it is given an internet address.
    - static or dynamic
  - IPv4 (32-bit) - 4.2 billion
  - IPv6 (128-bit) - 340 quadrilliard
  - In Powershell, try:  
*ipconfig /all*
  - In Terminal (MAC), try:  
ipconfig getifaddr en0

## IPv4 address in dotted-decimal notation



# MAC Address



A MAC Address is a unique 6-byte (48-bit) address that is usually permanently burned into a network interface card (NIC) and uniquely identifies the device on an Ethernet-based network. The uniqueness of MAC addresses is ensured by IEEE.

If you are creating your own MAC address, the 2's place bit of the first byte, the "locally administered bit" should be set. The 1's place bit, the "globally administered" bit must be off.

Therefore, `xA-xx-xx-xx-xx-xx` is valid, while `x7-xx-xx-xx-xx-xx` is not.

# Assignment: Wemo



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

Add the Ethernet (CS=10) to your breadboard along with a button in Pin 23.

## ① L08\_00\_EthernetTest

- Create your own mac.h MAC Address.

## ② L08\_01\_Wemo

- From the wemo.h library determine how the functions are called.
- Create code to turn on/off multiple Wemo Outlets in the classroom.

## ③ L08\_02\_Wemo\_Timer

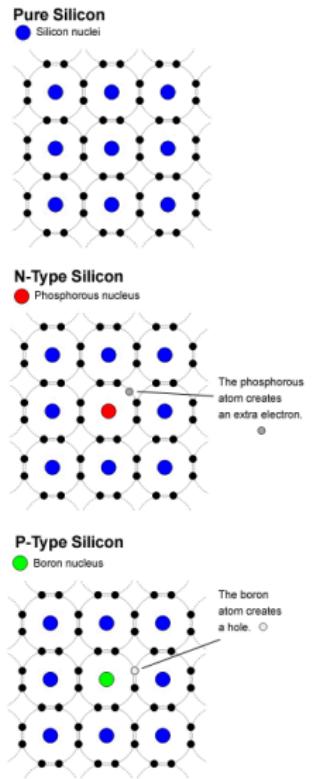
- Create timer that turns off a Wemo 10 secs after you push "off" button without using delay()

## ④ L08\_03\_Wemo\_Object

- Modify the wemo.h library to be a Class and Methods.
- Modify your wemo code to create and use a wemo object.

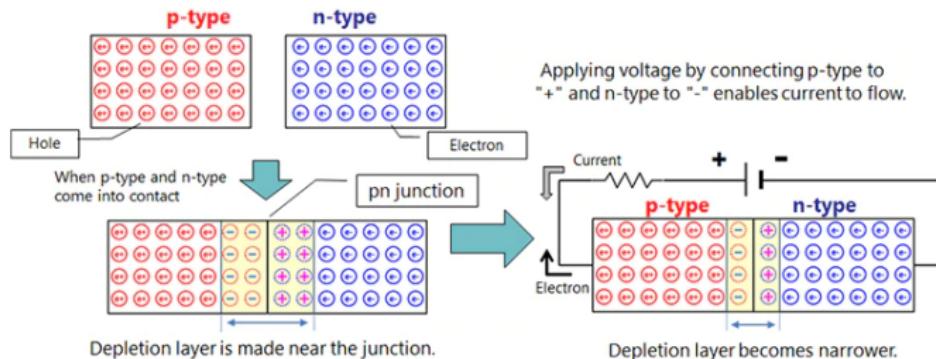
# Semiconductor

- A silicon atom has four electrons in its outer shell and bonds tightly with four surrounding silicon atoms creating a crystal matrix with eight electrons in the outer shells. The tight bonds make pure silicon non-conducting.
- Phosphorus has five electrons, and when combined, the fifth electron becomes a "free" electron that moves easily within the crystal when a voltage is applied.
- Boron has only three electrons in its outer shell and can bond with only three of surrounding silicon atoms. Thus one silicon atom has a vacant location in its outer shell, called a "hole," that readily accepts an electron.



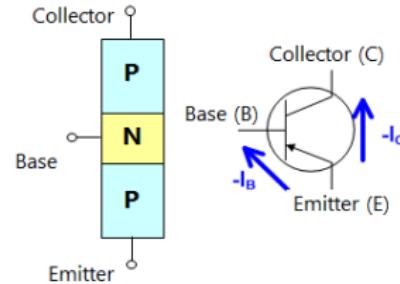
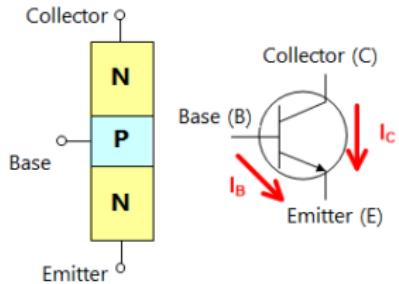
# pn junction diode

- When p-type and n-type semiconductors are bonded, holes and free electrons are attracted, combine, and disappear near the boundary. Since there are no carriers in this area, it is called a depletion layer and it is an insulator.
- A positive voltage applied to the p-type region causes electrons to flow sequentially from the n-type. The electrons will first disappear by combining with holes, but excess electrons move to the positive pole and current will flow.



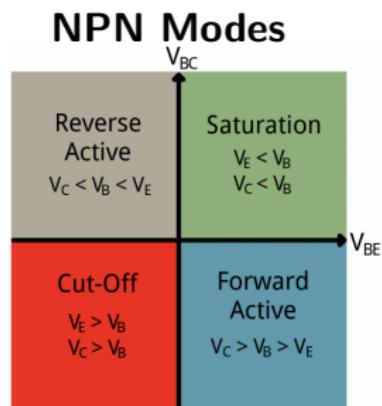
# Bipolar Junction Transistor

The transistor has three regions, namely base, emitter and collector. The emitter is a heavily doped terminal and emits electrons into the base. Base terminal is lightly doped and passes the emitter-injected electrons on to the collector. The collector terminal is intermediately doped and collects electrons from base. This collector is large as compared with other two regions so it dissipates more heat.



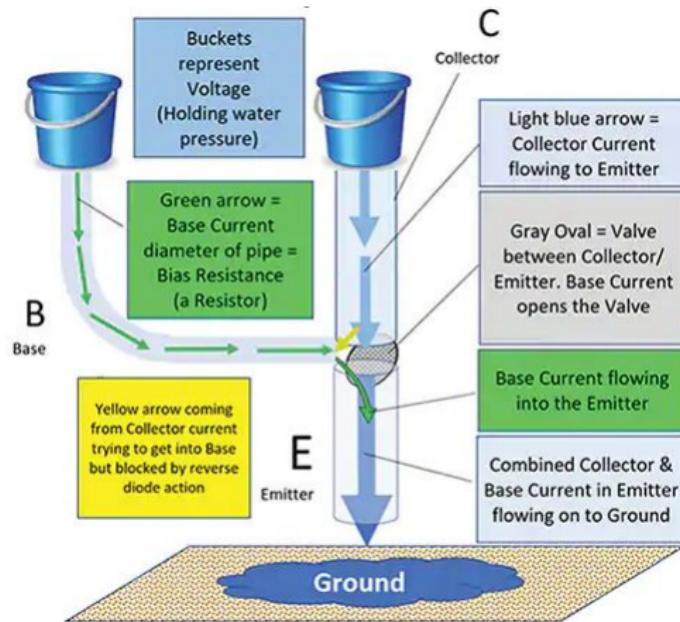
# Bipolar Junction Transistor - Modes of Operation

- **Saturation:** Current freely flows from collector to emitter. (ON Switch)
- **Cut-off:** No current flows from collector to emitter. (OFF Switch)
- **Active:** The current from collector to emitter is proportional to the current flowing into the base. (Amplifier)
- **Reverse-Active:** Like active mode, the current is proportional to the base current, but it flows reverse from emitter to collector (not the purpose transistors were designed for).



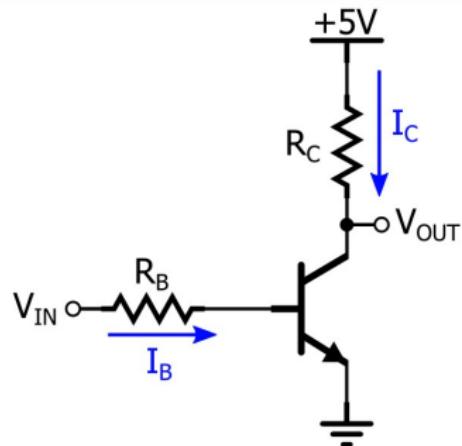
Voltage relations	NPN Mode	PNP Mode
$V_E < V_B < V_C$	Active	Reverse
$V_E < V_B > V_C$	Saturation	Cutoff
$V_E > V_B < V_C$	Cutoff	Saturation
$V_E > V_B > V_C$	Reverse	Active

# Water Analogy



# Active Mode NPN Transistor Circuit

If you apply a voltage  $V_{IN}$  that is high enough to forward-bias the base-to-emitter junction, current ( $I_B$ ) will flow from the input terminal, through  $R_B$ , through the BE junction, to ground. Current ( $I_C$ ) will also flow through  $R_C$  and the collector-to-emitter portion of the transistor.

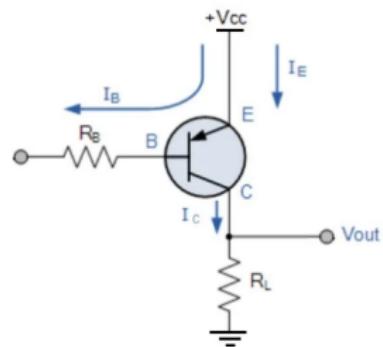


**NOTE:**  $V_{OUT}$  is an amplified but inverted signal of  $V_{IN}$ . This simple circuit will step-up a 0 - 3.3V output from the microcontroller to 0 - 5.0V (inverted). The low impedance of the output will also provide sufficient current to drive a higher current device (e.g., a relay).

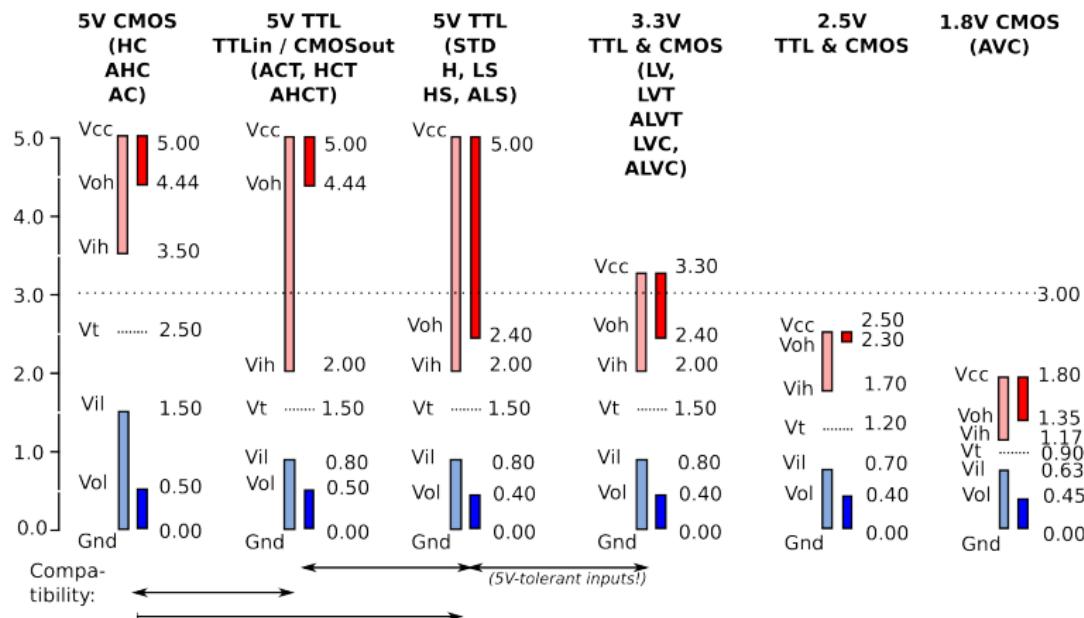
# PNP Transistor

NPN Transistors are more common than PNP for a number of reasons:

- The voltage and current behavior of an NPN transistor is significantly more intuitive.
- When a switch or driver circuit is required, NPNs provide a more straightforward interface to digital output signals (such as a control signal generated by a microcontroller).
- NPNs are higher performance (faster switching speeds) due to higher mobility of electrons vs holes.



# Logic Voltage Level Standards

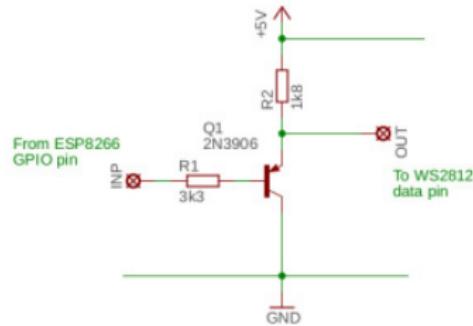


Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).

Or, what is wrong with the NeoPixels.

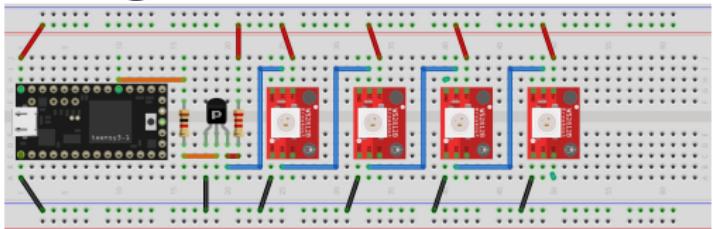
# Emitter Follower

- NeoPixels are designed around 5V CMOS transistors
  - $V_{IH} > 3.5V$
  - 3.3V Microcontroller
  - $V_{OH} = 3.3V$
- An Emitter Follower (i.e., a PNP transistor wired backwards) is a current amplifier, but will also produce a  $V_{OUT} = 3.9V$ .
- Alternatively, the first NeoPixel could be sacrificed by reducing its  $V_{cc}$  to 4.3V with a diode.



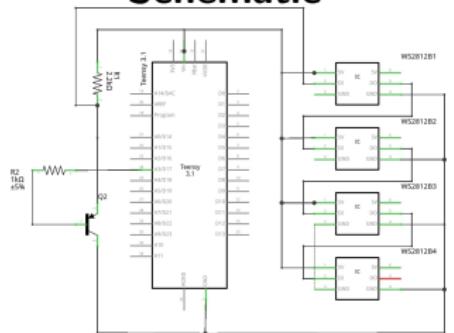
# Emitter Follower Layout

## Fritzing



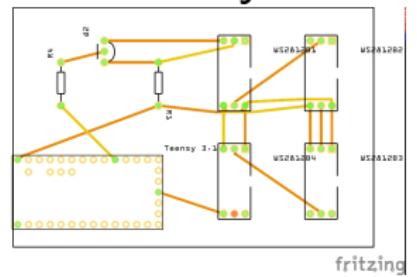
fritzing

## Schematic



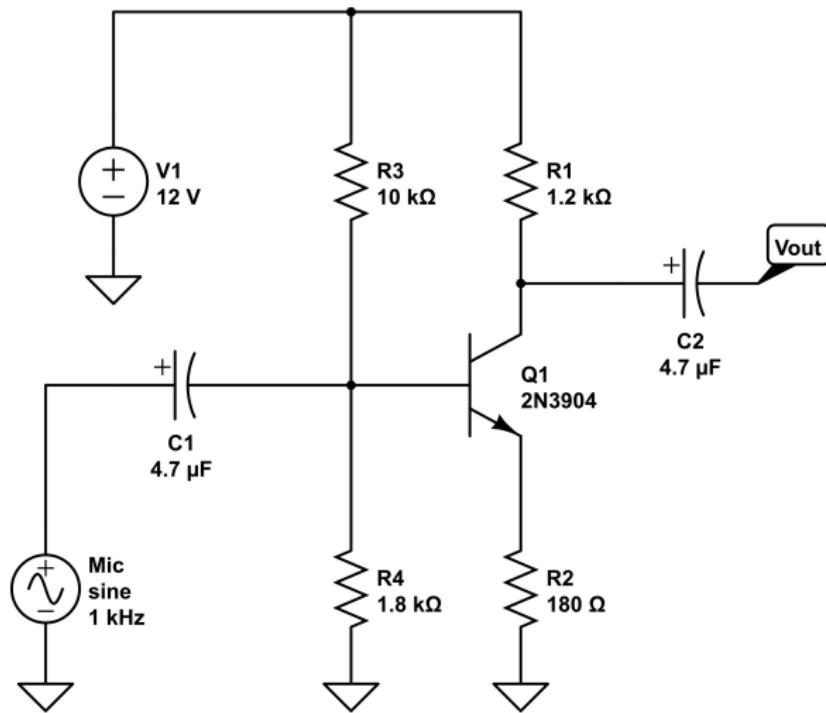
fritzing

## PCB Layout

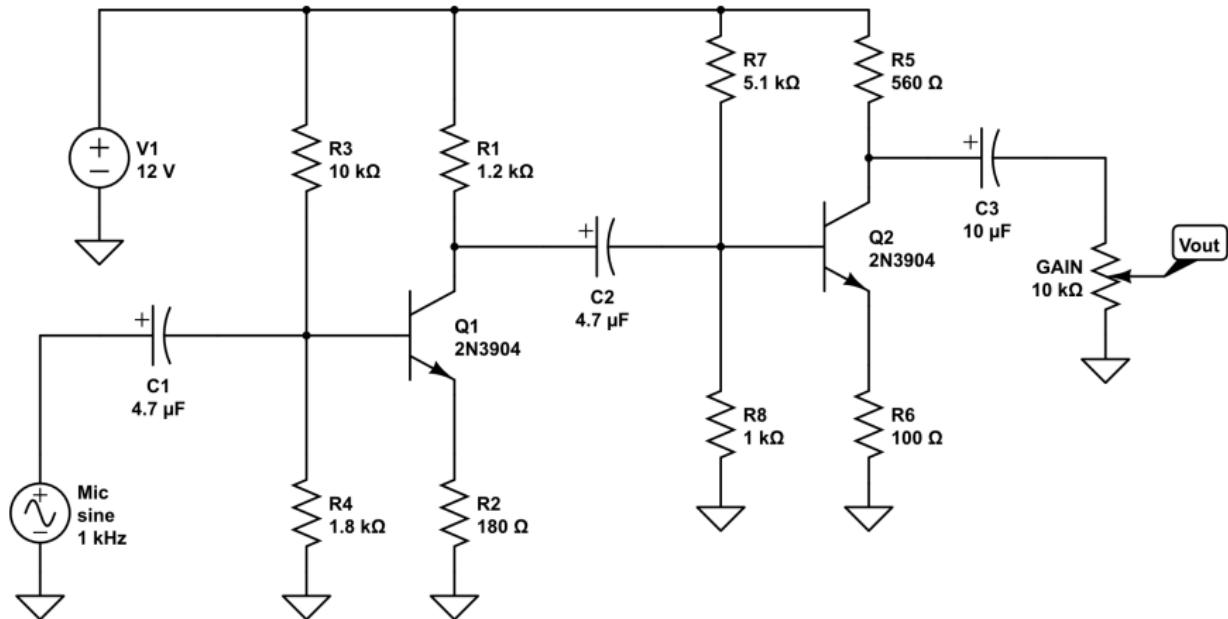


fritzing

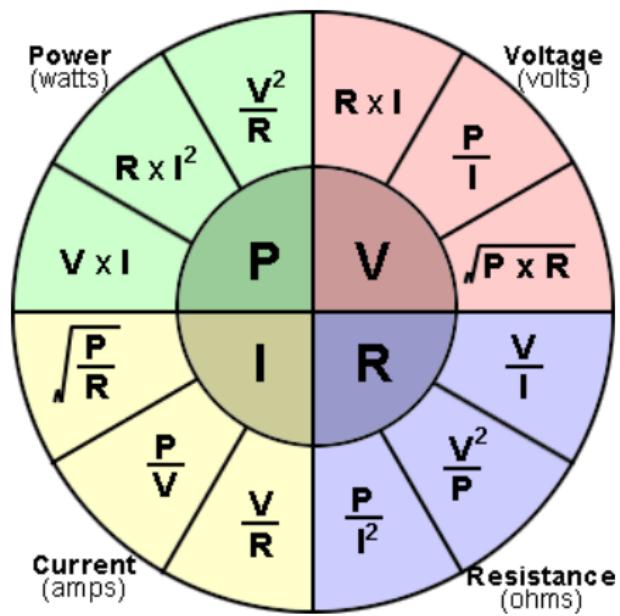
# NPN Pre-Amplifier Circuit



# Two Stage Pre-Amplifier



# Ohm's Law - Revisited



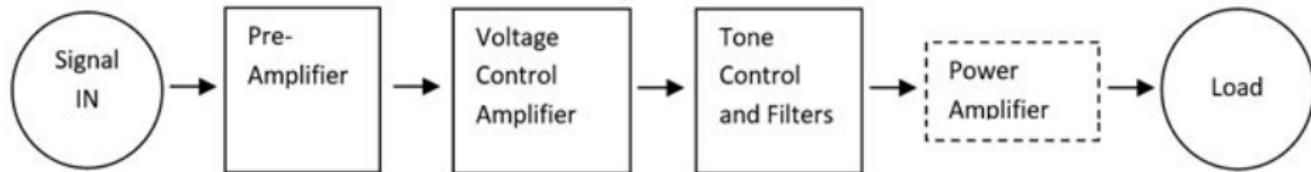
# PreAmp vs PowerAmp

PreAmp:

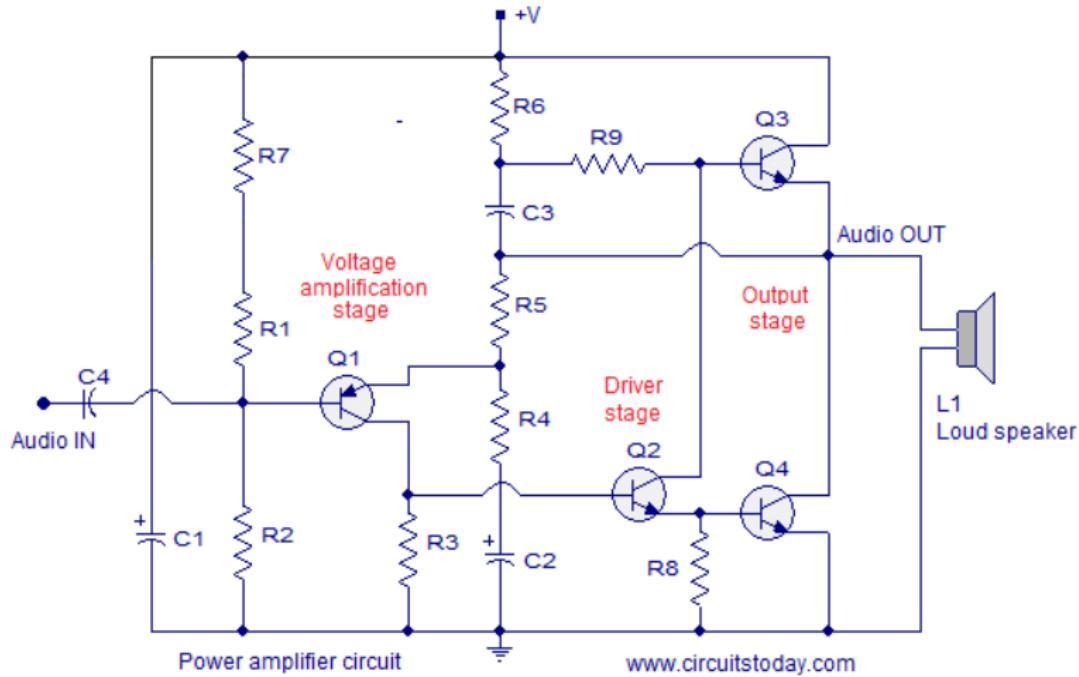
- A preamp boosts the signal up to 'line level'.
- Guitar PreAmp
  - A pure guitar signal typically sounds weak and anaemic, as is seen if a guitar is directly plugged PA system.
  - A preamp is able to raise a guitar's signal up to an audible volume.
  - It can also be used to affect the audio characteristics.

PowerAmp:

- A power amp boosts that line level signal even more – so that it can be projected through speakers.

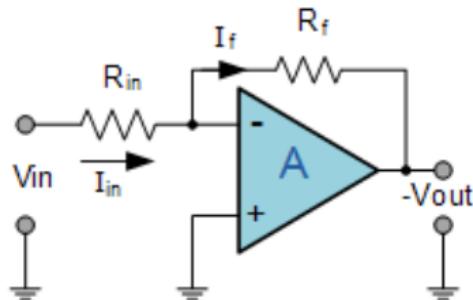


# Power Amplifier



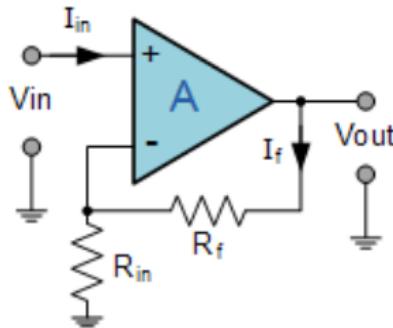
# Op Amp Lesson - Under Construction

## Inverting Op-amp



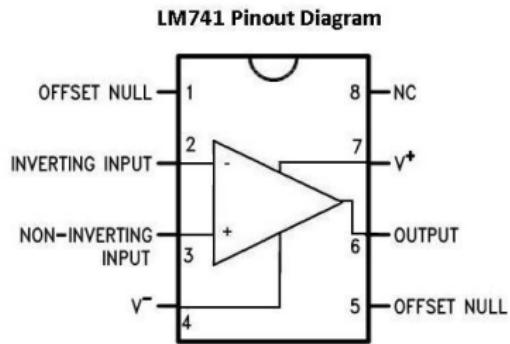
$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

## Non-inverting Op-amp



$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

# Assignment: Amplifiers



## ① L09\_01\_NeoPixel

- Add a Emitter-Follower into your NeoPixel circuit to boost the pixel commands to 5V.

## ② L09\_02\_NPNAMP (no Teensy)

- Make NPN Preamp
- Test your NPN Preamp with the signal generator and oscilloscope.

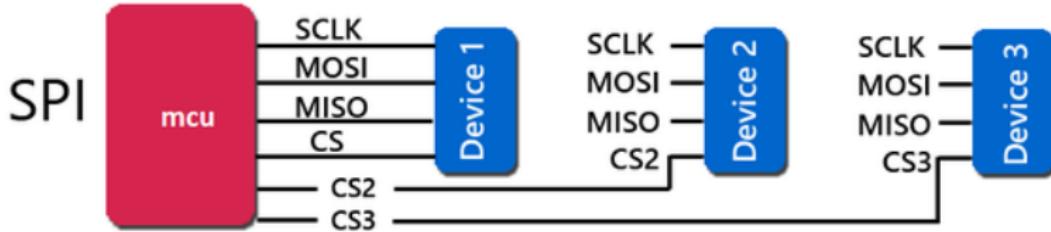
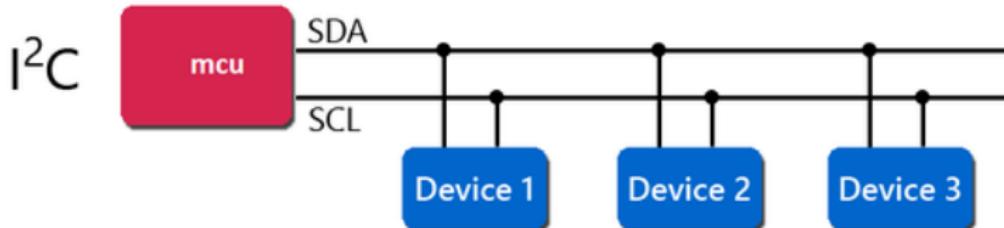
## ③ L09\_03\_OpAmp

- Create an OpAmp circuit to amplify your mic.
- Use a NeoPixel string (either your ring or a strip) to display the audio level.

# IoT Humor



# Inter-integrated Circuit ( $I^2C$ )



# $I^2C$ vs SPI

## I<sup>2</sup>C v/s SPI

I <sup>2</sup> C	SPI
Speed limit varies from 100kbps, 400kbps, 1mbps, 3.4mbps depending on i2c version.	More than 1mbps, 10mbps till 100mbps can be achieved.
Half duplex synchronous protocol	Full Duplex synchronous protocol
Support Multi master configuration	Multi master configuration is not possible
Acknowledgement at each transfer	No Acknowledgement
Require Two Pins only SDA, SCL	Require separate MISO, MOSI, CLK & CS signal for each slave.
Addition of new device on the bus is easy	Addition of new device on the bus is not much easy a I <sup>2</sup> C
More Overhead (due to acknowledgement, start, stop)	Less Overhead
Noise sensitivity is high	Less noise sensitivity

# L10\_00\_I2CScanner

Let's create an I2C scanner

- Create a Fritzing diagram for adding the BME280 and the OLED to your Teensy.
- Follow along to create the I2C code.
  - Use library wire.h
  - Wire.begin();
  - Wire.beginTransmission(i);
  - Wire.endTransmission();
    - 0: Transmission Successful
    - 1: Data too long to fit in transmit buffer
    - 2: Received NACK (Negative Acknowledgment) on transmit of address
    - 3: Received NACK on transmit of data
    - 4: Other error
- Wire up your schematic and determine the I2C addresses of each device.

# Assignment: I<sup>2</sup>C

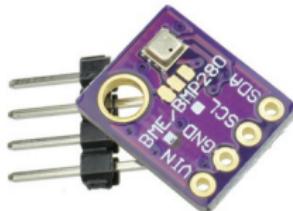


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L10\_01\_OLEDWrite

- Install Adafruit\_SSD1306 library
- Review and run SSD\_1306\_128x32\_i2c example
- Create variables for your birthday. Using the syntax from testdrawstyles() print using printf():
  - Hello World
  - Your Name
  - Your Birthday
  - Your Favorite Color
- Experiment with rotating the screen using the setRotation(int) method.

# Assignment: BME280



- ① Header - define BME280 object
  - I2C: Adafruit\_BME280 bme;
  - SPI: Adafruit\_BME280 bme(BME\_CS);
- ② Setup - start BME280
  - status = bme.begin(hex address);
  - if(status==false) → initialization failed
- ③ Loop - read from sensor
  - tempC = bme.readTemperature();
  - pressPA = bme.readPressure()/100.0;
  - humidRH = bme.readHumidity();

# Assignment: I<sup>2</sup>C



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L10\_02\_BME280

- Read BME280 data
- Convert to tempF and inHg
- Print to Serial.Monitor

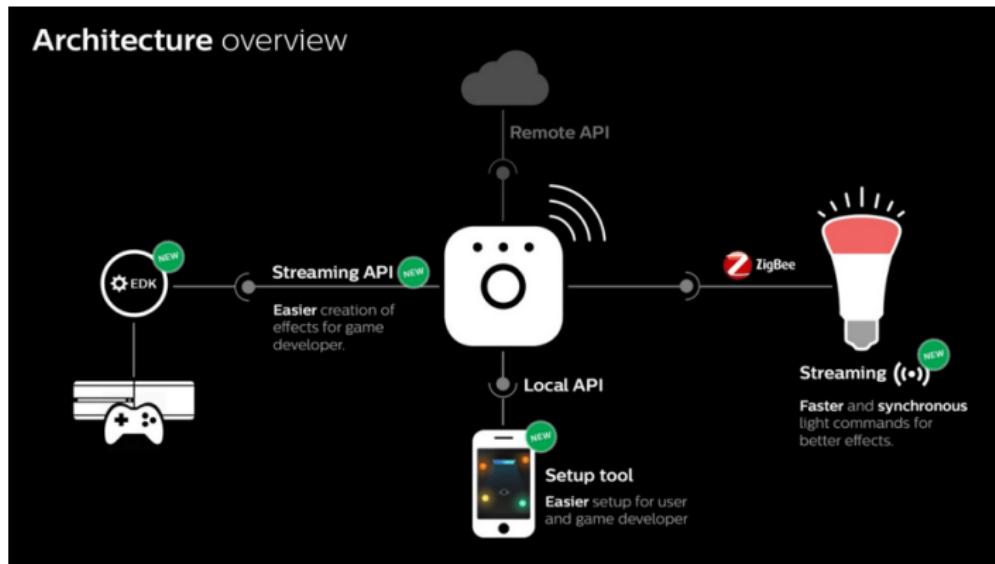
## ② L10\_03\_BME280\_OLED

- Print data to the OLED display

## ③ L10\_04\_BME280\_SDMicro

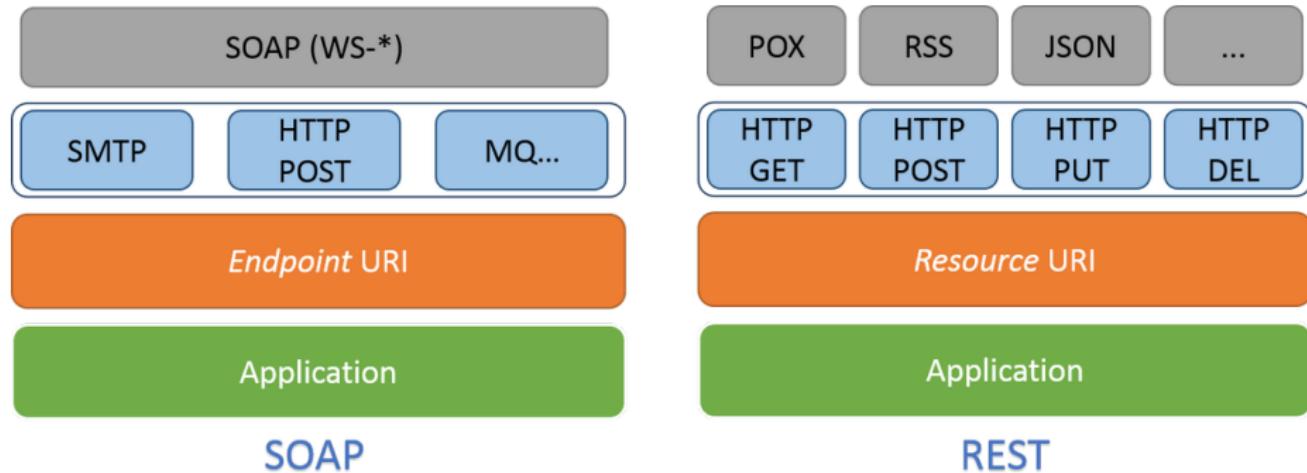
- Add in saving data to the µSD card
- Use your NeoPixels to give a visual indication of room conditions

# Phillips Hue API



Application Programming Interface: a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

# SOAP vs REST



# Assignment: L11\_01\_Hue



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

① Using the hue.h library and HueH\_Example as a template, create code that

- button that turns on and off the Hue light at your pod
- uses the encoder to change the brightness of the Hue bulb
- has a method of cycling the Hue light through the colors of the rainbow.

# Getting with GIT

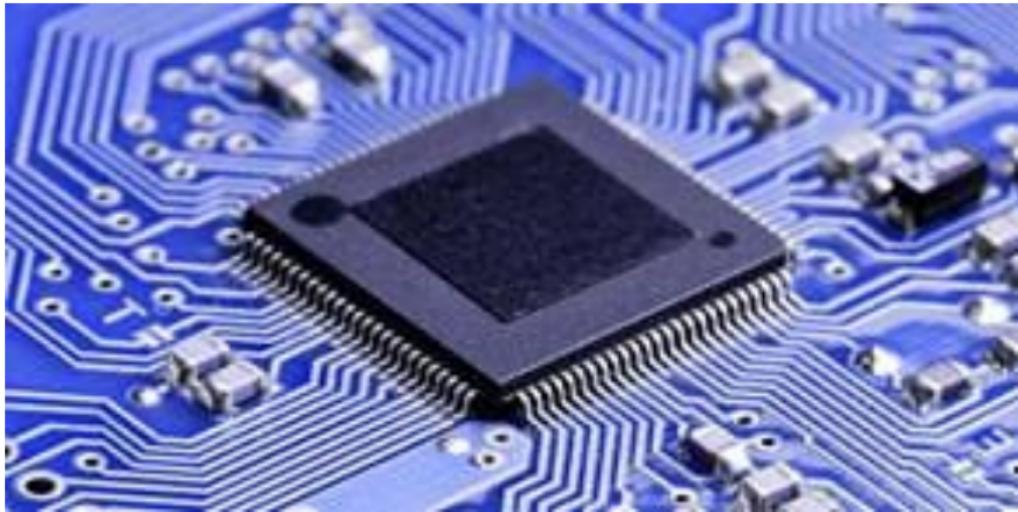


# GitHub

# Midterm Project - Smart Room Controller

- ① Determine functionality of your Smart Room Controller
  - Use the components that we have learned over the last 3 weeks.
  - Get minimum requirements from the Instructor.
  - Sketch out the basic layout of your room controller in your lab notebook.
  - Draw flowcharts of the main functions you plan to implement.
  - Get feedback from at least 3 peers on your planned functionality.
- ② Layout your circuitry in Fritzing along with a legible schematic.
- ③ Wire up your circuitry as if you're going to demo your controller for a perspective customer.
- ④ Code, debug, test.
- ⑤ Documentation and Demonstration:
  - Ensure all files are uploaded to GitHub with an appropriate README.md
  - Upload your project to hackster.io
  - Prepare a presentation/demonstration for the class on your controller
  - Participate in class demonstrations (Friday morning - Week 4)

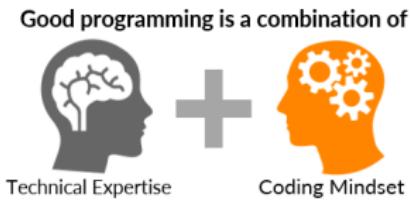
# Our Second Microcontroller



# Expectations for the rest of the course



# Expectations for the rest of the course



A programmer's three high-level goals  
are to write code that...

- ➊ Solves a specific problem
- ➋ Is easy to read
- ➌ Is maintainable and extendable

- ➊ Most important: Be Consistent
- ➋ Proper Indentation
- ➌ Brace placement: K&R or BSD
- ➍ Do not check boolean for equality.
- ➎ A variable's name (noun) should describe its contents.
- ➏ A function's name (verb) should describe the set of actions it performs.
- ➐ Reduce duplication / Modularize
- ➑ So, what about capitalization?
  - ➒ variableNames
  - ➒ nameFunction
  - ➒ CONSTANTS
  - ➒ Classes and Enum

# Particle Argon

## Main processor:

Nordic Semiconductor nRF52840 SoC

- ARM Cortex-M4F 32-bit processor @ 64MHz
- 1MB flash, 256KB RAM
- Bluetooth LE (BLE) central and peripheral support
- 20 mixed signal GPIO (6 x Analog, 8 x PWM), UART, I2C, SPI
- Supports DSP instructions, HW accelerated Floating Point Unit (FPU) calculations
- ARM TrustZone CryptoCell-310 Cryptographic and security module
- Up to +8 dBm TX power (down to -20 dBm in 4 dB steps)
- NFC-A radio

## Argon Wi-Fi network coprocessor:

Espressif ESP32-D0WD 2.4 GHz Wi-Fi coprocessor

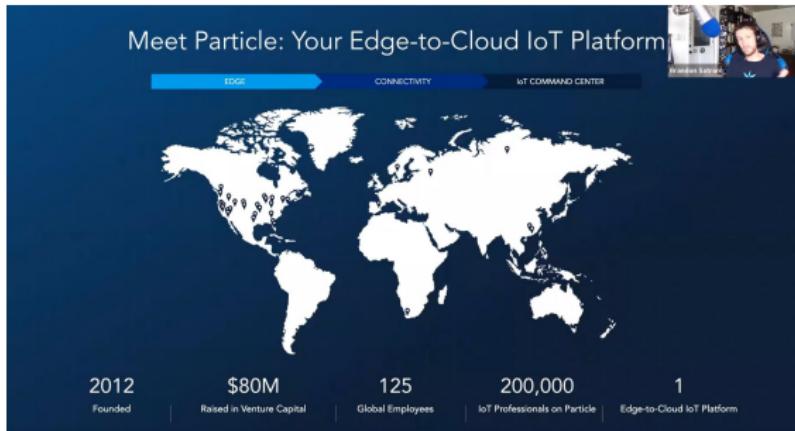
- On-board 4MB flash for the ESP32
- 802.11 b/g/n support
- 802.11 n (2.4 GHz), up to 150 Mbps



## Argon general specifications:

- On-board additional 4MB SPI flash
- Micro USB 2.0 full speed (12 Mbps)
- Integrated Li-Po charging and battery connector
- JTAG (SWD) Connector
- RGB status LED
- Reset and Mode buttons
- On-board 2.4GHz PCB antenna for Bluetooth (does not support Wi-Fi)
- Two U.FL connectors for external antennas (one for Bluetooth, another for Wi-Fi)
- Meets the [Feather specification](#) in dimensions and pinout
- FCC, CE and IC certified
- RoHS compliant (lead-free)

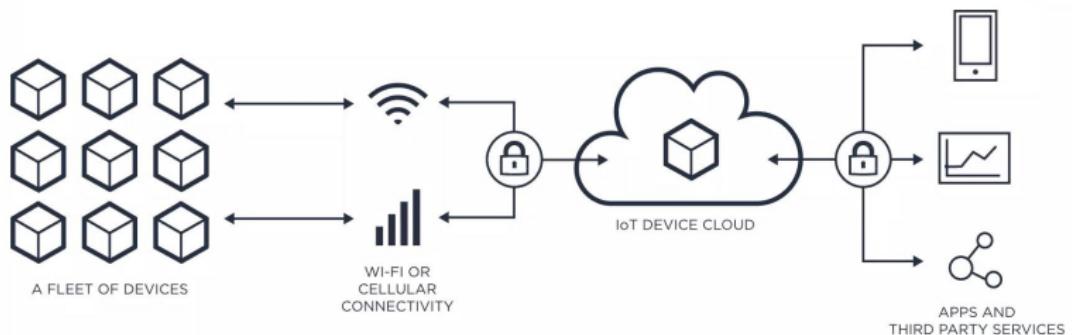
# Why Particle



- Global reach with over 200,000 IoT professionals
- Edge-to-Cloud infrastructure
- Prototyping to Production with same code
- WI-FI, Bluetooth, and Cellular
- Secure Device OS
- Built in cloud communication
- Real-Time OS that works across all products

# Particle: Edge to Cloud

## EDGE-TO-CLOUD IOT PLATFORM



# Particle: Prototyping to Production

## HARDWARE AND CONNECTIVITY



1

HARDWARE FOR  
PROTOTYPING  
& PRODUCTION



2

USE-CASE-SPECIFIC  
MODULES AND PRODUCTS

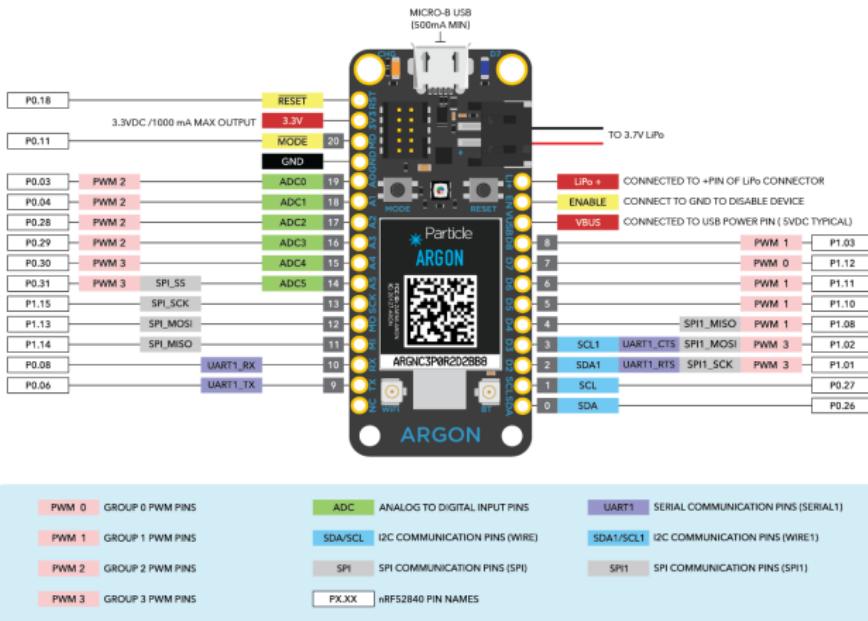


3

CELLULAR, BLE  
& WI-FI CONNECTIVITY

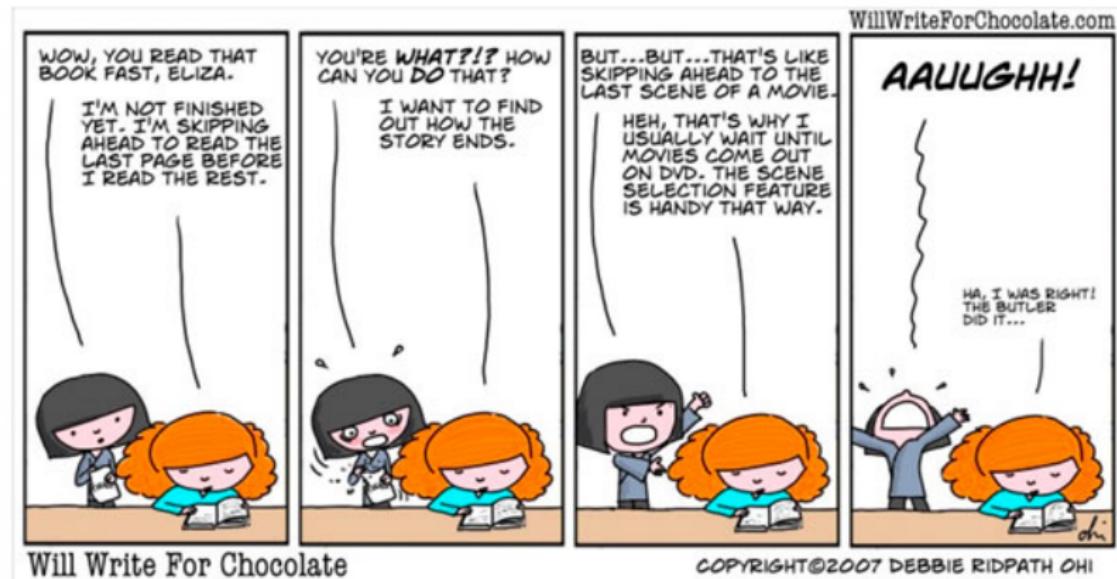


# Particle Argon Pin Layout



v1.0

# Please Do Not Skip Ahead During Setup



# Particle Software - ParticleCLI and Visual Studio Code

- ① Create Particle login: <https://login.particle.io/signup>
- ② Install Particle Command Line Interface. Download from <https://docs.particle.io/tutorials/developer-tools/cli/>.
  - On Windows, you need to run this by right-clicking on it and selecting "Run As Administrator."
  - Test that the Particle CLI installed correctly by going to PowerShell or Terminal and type particle
- ③ Download Particle Workbench / Visual Studio Code <https://docs.particle.io/quickstart/workbench/>.
  - Select all default values during install.
  - **Do NOT install Azure IoT.**
  - After it is installed, when you launch it, it may ask you to Install Dependencies. If so, select yes.

# Particle Setup

- ① Attach the Wi-Fi antenna to your Argon. Use the correct connector, there are 3 U.FL connectors: WiFi, BT, and NFC.
- ② Plug the Argon into a USB port. It should begin blinking blue.
- ③ Open PowerShell or Terminal.
- ④ Login into your Particle Account

```
1 particle login
```

- ⑤ Ensure you have the latest Particle CLI

```
1 particle update-cli
```

- ⑥ Put the Argon in DFU mode (blinking yellow) by holding down MODE. Tap RESET and continue to hold down MODE. The status LED will blink magenta (red and blue at the same time), then yellow. Release when it is blinking yellow.

# Updating your Argon to latest Device OS

- ① Update the device by running the following two commands. If the device goes out of blinking yellow after the first command, put it back into DFU mode. See Note <sup>2</sup>.

```
1 particle update  
2 particle flash --usb tinker
```

- ② When the command reports Flash success!, reset the Argon. It should go back into listening mode (blinking dark blue).
- ③ Verify that the update worked by running the following command:

```
1 particle serial identify  
2  
3 Your device id is e00fce681ffffffffc08949b  
4 Your system firmware version is 1.5.2
```

---

<sup>2</sup>particle flash –usb tinker can be used for device troubleshooting

# Setting Up WiFi

- Set your Argon into Listening Mode by holding the MODE button for three seconds, until the RGB LED begins blinking blue.
- Execute the command: `particle serial wifi`

```
brian:~$ particle serial wifi
? Should I scan for nearby Wi-Fi networks? No
? SSID DDCIOT
? Security Type WPA2
? Cipher Type AES+TKIP
? Wi-Fi Password ddcIOT2020
Done! Your device should now restart.
```

After setting, your Argon should go through the normal sequence of blinking green, blinking cyan (light blue), fast blinking cyan, and breathing cyan.

# Claim Your Device

- ① Claim the device to your account. This can only be done if it's breathing cyan. Replace e00fce681ffffffffc08949b with the device ID you got earlier from particle serial identify. Then, rename it to the name of your choice.

```
1 particle device add e00fce681fffffffffc08949b  
2 particle device rename e00fce681fffffffffc08949b  
    myArgon
```

- ② Ensure that your setup flag is marked as done

```
1 particle usb setup -done
```

- ③ You have successfully set up your Argon!

# Useful Particle CLI Commands

- ① Enter DFU mode from the CLI

```
1 particle usb dfu
```

- ② If Argon won't enter DFU mode, restore the base firmware

```
1 particle flash --usb tinker
```

- ③ Get a list of your Particle devices and their connection status

```
1 particle list
```

- ④ Search for available libraries

```
1 particle library search <search term>
```

# Argon LED Modes

Mode	LED Status
Connected	Breathing Cyan
OTA Firmware Update	Blinking Magenta (red and blue together)
Looking for Internet	Blinking Green
Connecting to Cloud	Rapid Blinking Cyan
Listening Mode	Blinking Blue
Network Reset	Rapid Blinking Blue
WiFi Off	Breathing White
Safe Mode	Breathing Magenta (red and blue together)
DFU (Device Firmware Upgrade)	Blinking Yellow
Restore Factory Firmware	Rapid Blinking Yellow
Factory Reset	Rapid Blinking White
Decryption Error	Blinking Cyan followed by 1 Orange Blink
No Internet	Blinking Cyan followed by 2 Orange Blink
No Particle Cloud	Blinking Cyan followed by 3 Orange Blink
Authentication Error	Blinking Cyan followed by 1 Magenta Blink
Handshake Error	Blinking Cyan followed by 1 Red Blink
Decryption Error	Blinking Cyan followed by 1 Orange Blink
SOS - Firmware Crash	Blinking Red - 3 short, 3 long, 3 short, error code

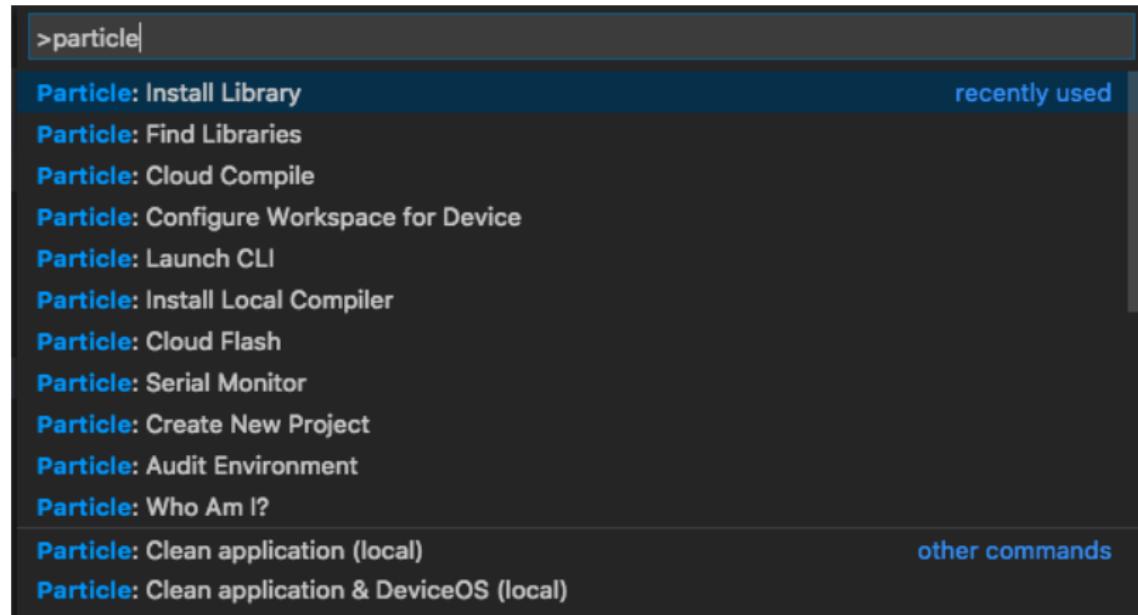
# Directory Structure - VERY IMPORTANT

The screenshot shows the VS Code interface with the following details:

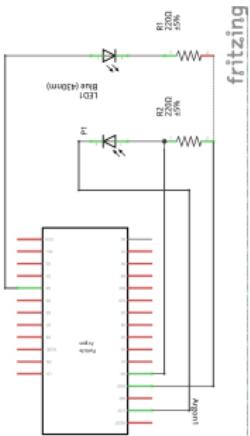
- EXPLORER** view: Shows the project structure.
  - OPEN EDITORS**: 1 UNSAVED
  - PM25\_Testino** (selected)
  - PM25\_TEST**
  - .vscode**: *launch.json*, *settings.json*
  - lib\Seeed\_HM330X**: *examples\basic\_demo*, *basic\_demo.ino*
  - src**: *HM330XErrorCode.h*, *I2COperations.cpp*, *I2COperations.h*, *Seeed\_HM330X.cpp*, *Seeed\_HM330X.h*
  - PM25\_Testino**
  - target\1.5.0\argon**
  - project.properties**
  - README.md**
- CODE** view: Displays the *PM25\_Testino* file content.

```
src > PM25_Testino > ...
1 /*
2  * Project PM25
3  * Description: 2.5um Particle Measurement with H3301 Sensor
4  * Author: Brian Rashap
5  * Date: 17-APR-2020
6 */
7 #include <Particle.h>
8 #include <Seeed_HM330X.h>
9 #include <Wire.h>
10
11 //*****Set Up HM330X*****
12 HM330X sensor;
13 uint8_t buf[30];
14 int PM25;
15
16 const char* str[] = {"sensor num: ", "PM1.0 concentration(CF=1,Standard particulate matter",
17 "PM2.5 concentration(CF=1,Standard particulate matter,unit:ug/m3): ",
18 "PM10 concentration(CF=1,Standard particulate matter,unit:ug/m3): ",
19 "PM1.0 concentration(Atmospheric environment,unit:ug/m3): ",
20 "PM2.5 concentration(Atmospheric environment,unit:ug/m3): ",
21 "PM10 concentration(Atmospheric environment,unit:ug/m3): ",
22 };
23
24 HM330XErrorCode print_result(const char* str, uint16_t value) {
25     if (NULL == str) {
26         return ERROR_PARAM;
27     }
28     Serial.print(str);
29     Serial.println(value);
30     return NO_ERROR;
}
```

# Command Palette - Ctrl-Shift-P



# Assignment: L12\_HelloParticle



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L12\_01\_HelloParticle

- Blink the onboard LED (Pin D7)

## ② L12\_02\_HelloNightLight

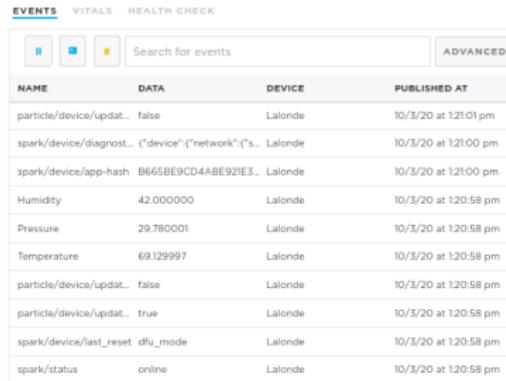
- The anode of the photodiode is connected to Pin A0. Note, unlike an LED, the cathode (short pin) of the photodiode is connected to 3.3V.
- The LED anode to Pin D4.
- Using analogRead/digitalWrite, turn on the LED when photodiode is dark.
- Using digitalWrite, turn on LED slowly as room darkens

## ③ L12\_03\_HelloRes

- Create code to determine the resolution (in bits) of read / write.
- Use Serial Monitor via Command Palette

# Particle Publish

One of the advantages of the Argon is seamless publishing to the Cloud.



The screenshot shows the Particle Cloud interface with the 'EVENTS' tab selected. It displays a table of published events with columns: NAME, DATA, DEVICE, and PUBLISHED AT. The data includes various sensor readings and device status updates from a device named 'Lalonde'.

NAME	DATA	DEVICE	PUBLISHED AT
particle/device/updat...	false	Lalonde	10/3/20 at 1:21:01 pm
spark/device/diagnost...	{"device": {"network": "..."}, "status": "ok", "last_re...	Lalonde	10/3/20 at 1:21:00 pm
spark/device/app-hash	B665BE9CD4ABE921E3...	Lalonde	10/3/20 at 1:21:00 pm
Humidity	42.000000	Lalonde	10/3/20 at 1:20:58 pm
Pressure	29.780001	Lalonde	10/3/20 at 1:20:58 pm
Temperature	69.129997	Lalonde	10/3/20 at 1:20:58 pm
particle/device/updat...	false	Lalonde	10/3/20 at 1:20:58 pm
particle/device/updat...	true	Lalonde	10/3/20 at 1:20:58 pm
spark/device/last_reset	dfu_mode	Lalonde	10/3/20 at 1:20:58 pm
spark/status	online	Lalonde	10/3/20 at 1:20:58 pm

```

1 float temp, prs, hum;      // BME280 variables
2 String Temp, Prs, Hum;    // Strings to hold BME280 values
3
4 void loop() {
5     Temp = String(temp);
6     Prs = String(prs);
7     Hum = String(hum);
8
9     Particle.publish("Temperature", Temp, PRIVATE);
10    Particle.publish("Pressure", Prs, PRIVATE);
11    Particle.publish("Humidity", Hum, PRIVATE);
12 }
```

# JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

# JSON Parser Generator

Creating objects in JSON are straightforward but can be tedious. There is a JSON Parser available to simplify the process.

```
1 #include <JsonParserGeneratorRK.h>
2
3 void createEventPayLoad(float tempValue, float presValue, float humValue) {
4     JsonWriterStatic<256> jw;
5     {
6         JsonWriterAutoObject obj(&jw);
7
8         jw.insertKeyValue("Temperature", tempValue);
9         jw.insertKeyValue("Pressure", presValue);
10        jw.insertKeyValue("Humidity", humValue);
11    }
12    Particle.publish("env-vals", jw.getBuffer(), PRIVATE);
13 }
```

# Assignment: L12\_HelloParticle

The screenshot shows the Particle Cloud interface with two main sections: 'EVENTS' and 'env-vals'.

**EVENTS:** This section displays a table of events. The columns are NAME, DATA, DEVICE, and PUBLISHED AT. The data includes:

NAME	DATA	DEVICE	PUBLISHED AT
particle/device/updat...	false	Lalonde	10/3/20 at 1:09:40 pm
spark/device/diagnost...	{"device": {"network": "..."}, "...	Lalonde	10/3/20 at 1:09:40 pm
spark/device/app-hash	651A9E3A5D0A02A4115...	Lalonde	10/3/20 at 1:09:39 pm
env-vals	{"PhotoDiode": 487, "LED": 487}	Lalonde	10/3/20 at 1:09:38 pm
LED Output	47	Lalonde	10/3/20 at 1:09:38 pm
PhotoDiode	487	Lalonde	10/3/20 at 1:09:38 pm

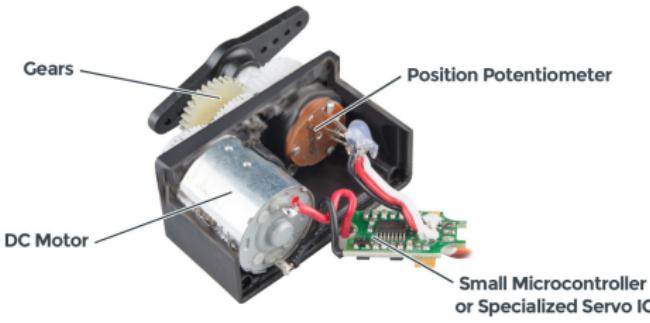
**env-vals:** This section shows environment variable values. It includes a 'PRETTY' button, a 'RAW' button, and a JSON representation of the data:

```
PRETTY
RAW
{
  "PhotoDiode": 487,
  "LED": 487
}
```

## ① L12\_04\_HelloPublish

- Using Particle.publish() send the photodiode and LED values to the Particle Cloud
- Additionally, send the same data using the JSON Generator

# Servo Motors



A servo is any motor-driven system with a feedback element built in.

A servo motor basically has three core components:

- a DC motor,
- a controller circuit,
- a potentiometer or similar feedback mechanism.

# Assignment: Servo.h library

There are a number of libraries that are automatically added into your Particle DeviceOS code, Servo.h is one of them.

## ① Header

- Servo myServo; - create object myServo of class Servo

## ② void setup()

- myServo.attach(pin) - attach the Servo object to a pin

## ③ void loop()

- myServo.write(angle) - move servo to angle (in degrees)
- myServo.read() - returns the current angle of the servo

# Assignment: L13\_Servo



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L13\_01\_Servo

- Have your Servo cycle from 0 to 180 degrees

## ② L13\_02\_ServoSine

- Have the Servo move in a sine wave pattern
- Add a button (using pinMode: INPUT\_PULLDOWN) that starts and stops the motion.

# SparkFun\_Qwiic\_Twist RGB Encoder

```
1 #include <SparkFun_Qwiic_Twist_Arduino_Library.h>
2 TWIST twist; //Create instance of this object
3
4 void setup() {
5     Serial.begin(9600);
6     Serial.println("Qwiic Twist Example");
7
8     if(twist.begin() == false)
9     {
10         Serial.print("Twist does not appear to be
11 connected.");
12         Serial.println("Please check wiring. Freezing
13 ...");
14         while(true);
15     }
16 }
```

# SparkFun\_Qwiic\_Twist Commands

- `bool twist.begin()` - initializes the sensor with basic settings and returns false if sensor is not detected
- `int twist.getCount()` - returns the number of indents the user has twisted the knob
- `int twist.getDiff()` - returns the difference between the last two `getCount()` calls
- `bool twist.isMoved()` - returns true if knob has been twisted
- `int twist.timeSinceLastMovement()` - returns the number of milliseconds since the last encoder movement
- `bool twist.clicked()` - returns true if a click event has occurred
- `int twist.timeSinceLastPress()` - returns the number of milliseconds since the last button event (press and release)
- `bool twist.isPressed()` - returns true if button is currently being pressed
- `void twist.setCount(pos)` - set the encoder count to a specific pos
- `void twist.setColor(R,G,B)` - sets the color of the encoder.

# Assignment: L13\_Servo



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L13\_03\_Qwiic

- Clone and place in <project folder>/<project name>/lib the appropriate library.
- Print encoder position, only if Moved
- Using ENUM, create 3 button pressed states lighting up different colors for each
- Display time since last moved and last pressed

## ② L13\_04\_QwiicServo

- Create two states: OFF and ON. Red for OFF, Green for ON.
- When OFF, encoder does nothing.
- When ON, encoder moves Servo. Half rotation moves encoder from 0 to 180
- When button is pressed and held, turning encoder changes ON color.

# Too Much Time On My Hands

When the Particle Argon connects to the Particle Cloud, it synchronizes its clock to the current time.

```
1 void sync_my_time() {
2     String DateTime, TimeOnly;
3     char currentDateTime[25], currentTime[9];
4
5     Time.zone(-7); // Set Time Zone to MDT(UTC-7)
6     unsigned long cur=millis();
7
8     // Request time synch from Particle Cloud
9     Particle.syncTime();
10    waitUntil(Particle.syncTimeDone);
11 }
```

# Using Particle Time

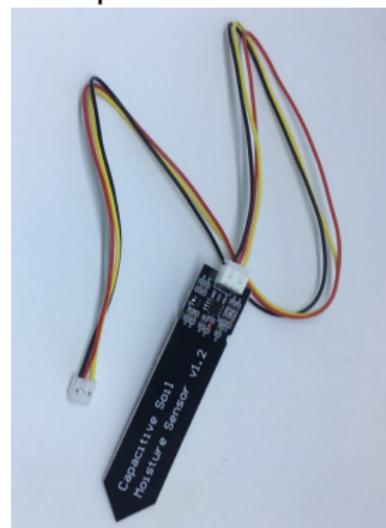
```
1 // Get Current Time
2 DateTime = Time.timeStr();
3 TimeOnly = DateTime.substring(11,19);
4 Serial.println(DateTime);
5 Serial.println(TimeOnly);
6
7 // Using Time with Formatted Print Statements
8 DateTime.toCharArray(currentDateTime,25);
9 TimeOnly.toCharArray(currentTime,9);
10 Serial.printf("Date and time is %s\n",
11     currentDateTime);
11 Serial.printf("Time is %s\n",currentTime);
```

# Soil Moisture Sensors

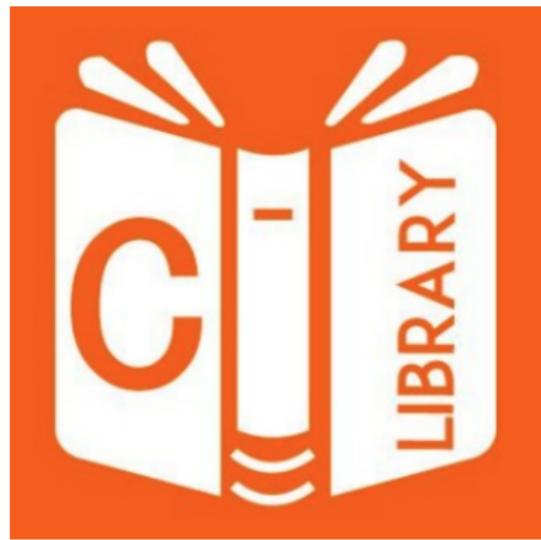
Resistive Sensor



Capacitive Sensor



# Installing Libraries



# Assignment: L14\_SoilMoisture



## ① L14\_01\_OLED

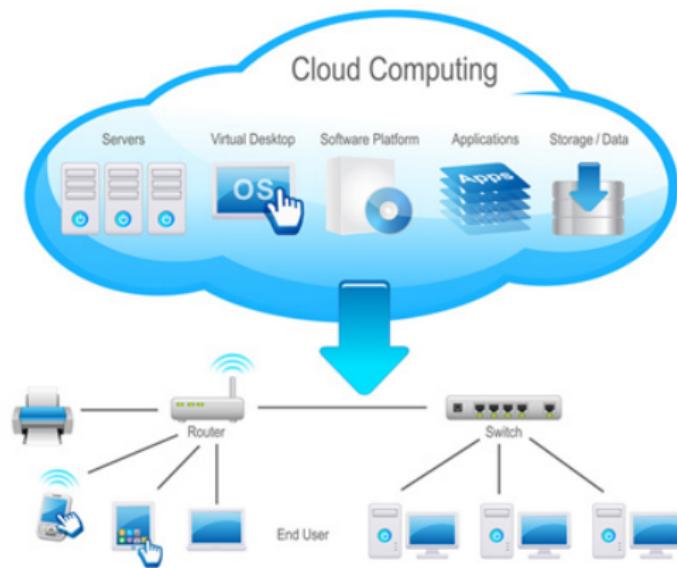
- Install the Adafruit\_SSD1306 library
- NOTE: you have full 128x64 pixel access
- Create sample code displaying your name and time to the OLED

## ② L14\_02\_Moisture

- Using the Capacitive Soil Moisture probe. In your notebook note the moisture readings when:
  - Empty Cup
  - Submerged in water to the notch
  - Dry Soil
  - Soil after watered
- Display the moisture to the OLED with a Time-stamp

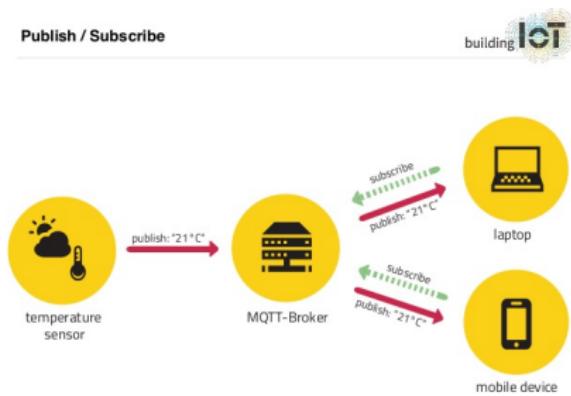
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

# The Cloud

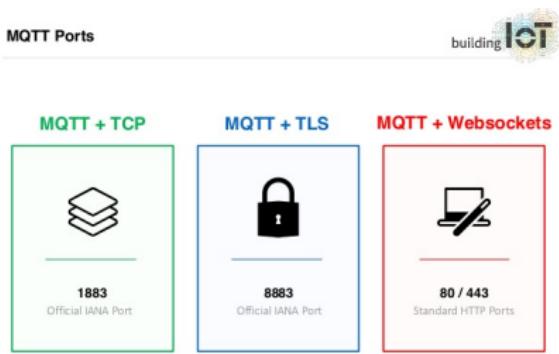


# MQTT: MQ Telemetry Transport

## Publish / Subscribe



## MQTT Ports



# Adafruit.io



## Let's create an Adafruit.io account

**Get Started**

**FREE**  
forever

30 data points per minute  
30 days of data storage  
Triggers every 15 minutes  
5 feed limit

[Sign Up Now](#)

**Power Up**

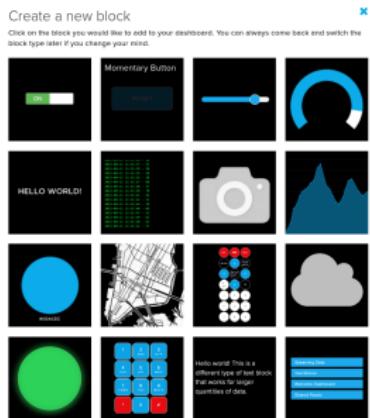
**\$10 or \$99**  
per month      per year

60 data points per minute  
60 days of data storage  
Triggers every 5 seconds  
Unlimited feeds

[Learn more about IO+>](#)

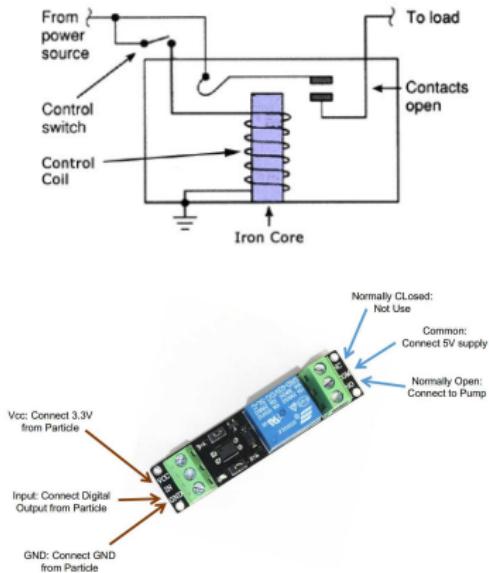
[Sign Up Now](#)

# Assignment: L14\_03\_SubscribePublish



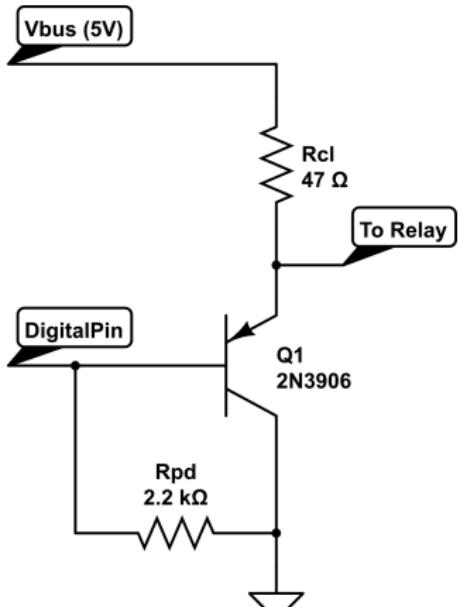
- ① Modify the starter code for your Adafruit.io
- ② Subscribe
  - Add a button to your Adafruit.io dashboard and connect it to a feed called buttonOnOff
  - Subscribe to the buttonOnOff and turn on the on board LED when pressed.
- ③ Publish
  - Publishing a random number to a feed once per minute (do not use a delay).
  - Create a line chart on your dashboard to display the random number.
- ④ Experiment with other blocks
  - Replace the button with a slider.
  - Control the brightness of an LED
  - Display data with other dashboard blocks.

# Relays



- When a device (e.g. a pump) requires higher voltage ( $> 5V$ ) or higher current, then a relay can be used as a switch for the device
- The relay is activated by a digital pin from the microcontroller.
- However, as the relay requires 100mA from the digital pin, we will use a transistor switch to draw power directly from the USB connection.

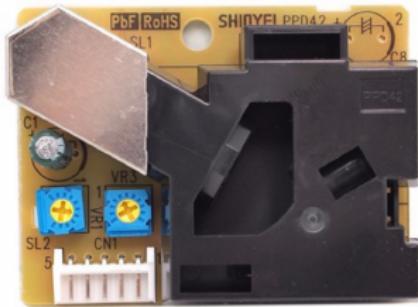
# Assignment: L14\_04\_PlantWater



PNP Emitter Follower

- ① Integrate an 2N3906 Emitter Follower and Relay, as well as a BME280 and Display.
- ② Publish soil moisture and room environmental data to a new dashboard
- ③ Automatically water your plant with the soil is too dry
  - Only turn on the pump for a very short period of time ( $\frac{1}{2}$  sec).
- ④ Integrate a button into your dashboard that manually waters the plant.

# Seeed Sensors



Seeed Grove - Dust Sensor



Seeed Grove - Air Quality  
Sensor v1.3

# Assignment: More L14

## ① L14\_05\_PulseIn

- Look up the `pulseIn()` command in the Arduino reference
- Connect PWM pin to Analog input. Use `pulseIn` to measure length of PWM-pulse while varying voltage output with encoder.

## ② L14\_04\_PlantWater (revisted): Integrate into your Plant Water project both Seeed sensors.

- Look up the Seeed sensors online to see how they work.
- Do not blindly copy the examples, only use the code you need.
- By looking at the .cpp code, determine how to get a quantitative value for air quality, in addition to the qualitative level.

## ③ Include Hackster.io story

# Negative Numbers

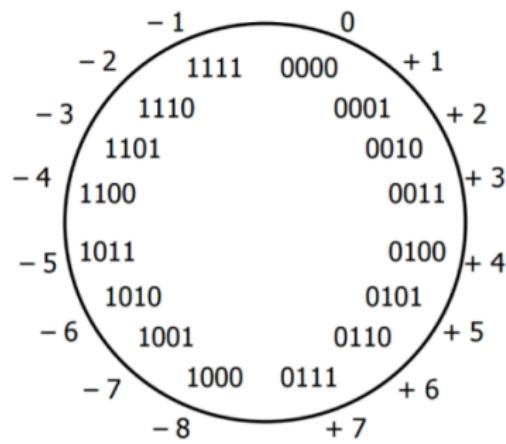
Question: How are negative numbers represented in binary?

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Answer: Left-most bit is 1 to signify negative. But wait...

# 2's Compliment

2's compliment is used as it makes the math consistent.



Integer		2's Complement
Signed	Unsigned	
5	5	0000 0101
4	4	0000 0100
3	3	0000 0011
2	2	0000 0010
1	1	0000 0001
0	0	0000 0000
-1	255	1111 1111
-2	254	1111 1110
-3	253	1111 1101
-4	252	1111 1100
-5	251	1111 1011

The negative plus the positive equals zero.

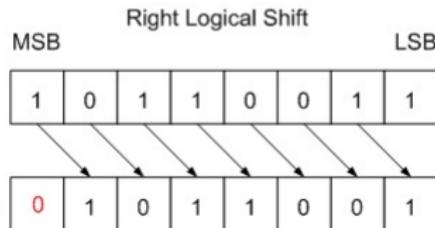
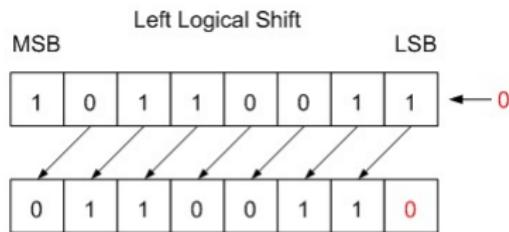
# Bitwise Operations

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

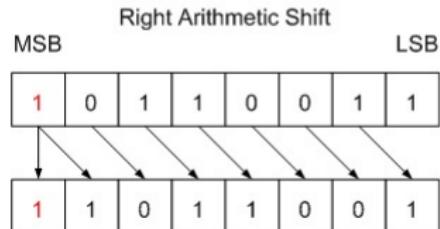
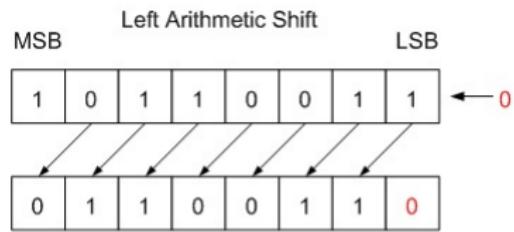
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	$(A \& B) = 12$ , i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	$(A   B) = 61$ , i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A ^ B) = 49$ , i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	$(\sim A) = \sim(60)$ , i.e., 1100 0011
<<	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand.	$A << 2 = 240$ i.e., 1111 0000
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	$A >> 2 = 15$ i.e., 0000 1111

# Bit Shifting

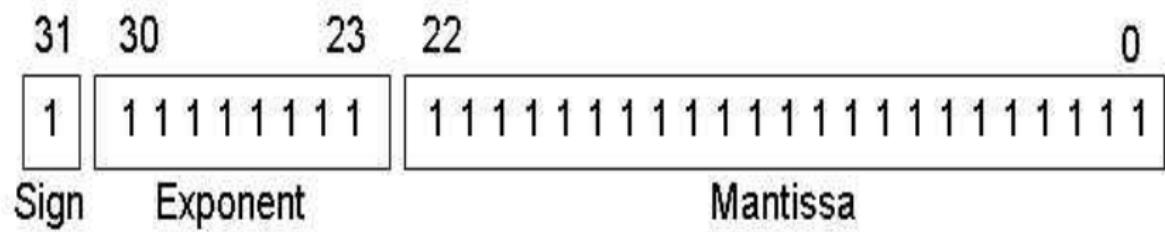
## Logical Bit Shift



## Arithmetic Bit Shift



## BONUS: But what about Floating Point



Example:

$$-36382.366 = -3.6382366 \times 10^4 = 1000001011101111000010001011110$$

# Electrically Erasable Programmable Read Only Memory

EEPROM emulation allows small amounts of data to be stored and persisted even across reset, power down, and user and system firmware flash operations. Since the data is spread across a large number of flash sectors, flash erase-write cycle limits should not be an issue in general.

```
1 len = EEPROM.length(); //available EEPROM bytes
2 // Argons have 4096 bytes of emulated EEPROM.
3 // Addresses 0x0000 through 0xFFFF
4
5 addr = 0x00AE;      //addr between 0 and len-1
6
7 val = 0x45;
8 EEPROM.write(addr, val);
9
10 value = EEPROM.read(addr);
```

# Interrupts

Interrupts are a way to write code that is run when an external event occurs. As a general rule, interrupt code should be very fast, and non-blocking. This means performing transfers, such as I2C, Serial, TCP should not be done as part of the interrupt handler. Rather, the interrupt handler can set a variable which instructs the main loop that the event has occurred.

```
1 attachInterrupt(pin, function, mode);
```

Mode: defines when the interrupt should be triggered. Three constants are predefined as valid values:

- CHANGE to trigger the interrupt whenever the pin changes value,
- RISING to trigger when the pin goes from low to high,
- FALLING for when the pin goes from high to low.

# L15\_BEI Assignments

## ① L15\_01\_ConvertStore

- Convert random hex color into R, G, and B components using Bit Shifting and Bitwise AND.
  - First sketch out how this might be done in your lab notebook. Show to instructor.
  - Then, and only then, write the code.
- Store the components of the color as bytes in the Argon's EEPROM

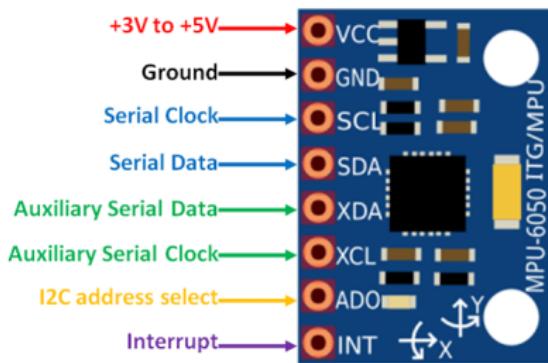
## ② L15\_02\_RetrieveShow

- Retrieve the color from EEPROM memory
- Convert to Hex code
- Display color in NeoPixel.

## ③ L15\_03\_Interrupt Create an interrupt Stop Button

- Wire a button, a GREEN and a RED LED to your Argon
- Using delays, blink your GREEN LED (once per second)
- Using an Interrupt, when the button is pressed, the GREEN turns off and the RED lights up.

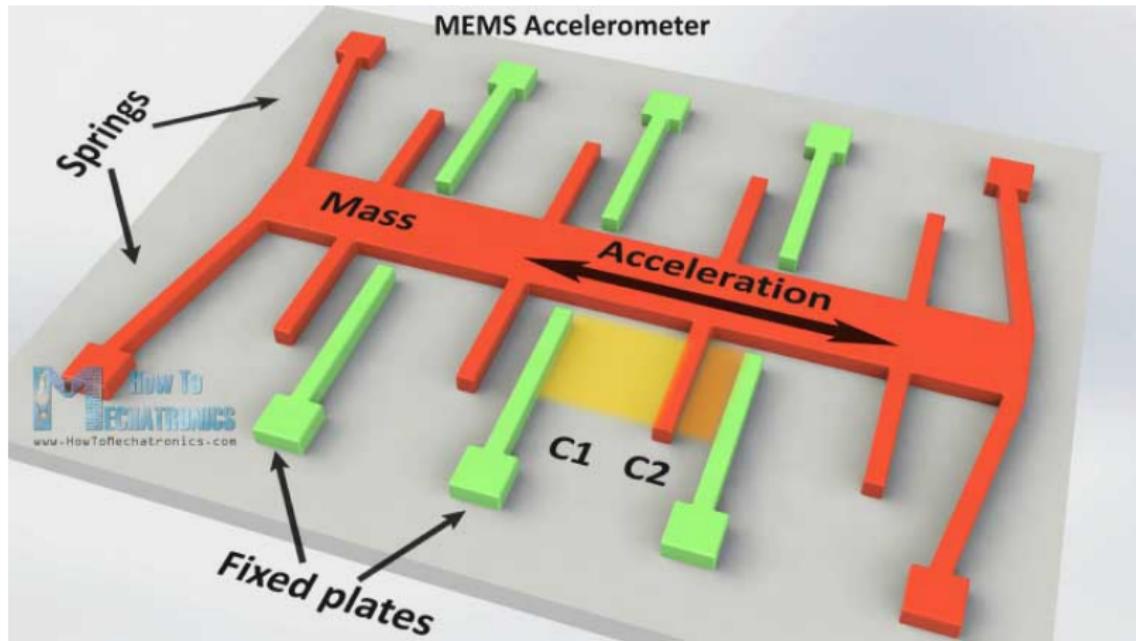
# MPU6050 Accelerometer



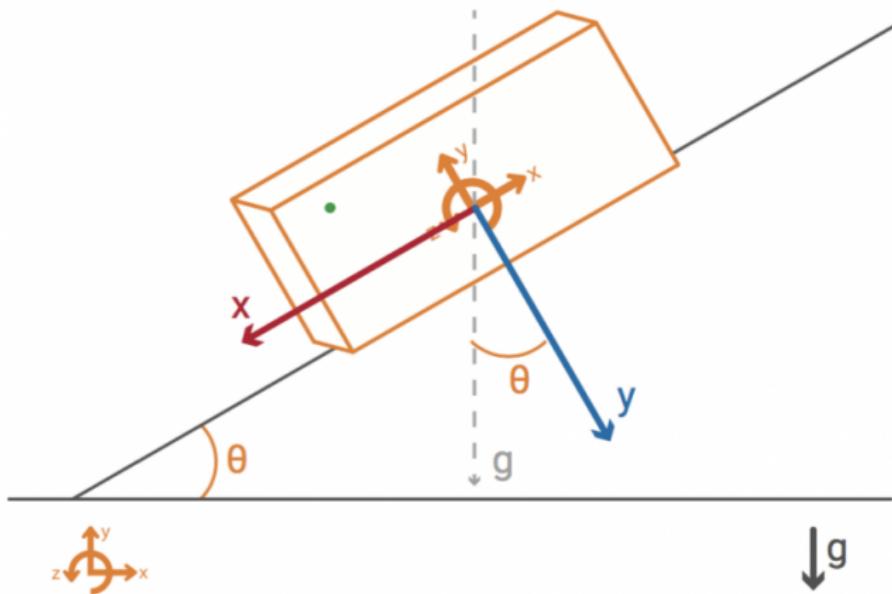
- Data Output - 16 Bit
- Gyros range:  $\pm 250$   $\pm 500$   $\pm 1000$   $\pm 2000$  °/s
- Accel range:  $\pm 2$   $\pm 4$   $\pm 8$   $\pm 16$ g

- The interrupt pin lets the MPU be notified about available data. To reduce power consumption the processor can go into sleep mode, the interrupt can be used as a wake up.
- XDA and XCL refer to the I2C bus that the MPU-6050 controls, so it can read from slave devices such as magnetometers etc.

# Accelerometers



# Gravity and Orientation



$$a_y = g * \cos(\theta)$$

# Initializing MPU6050

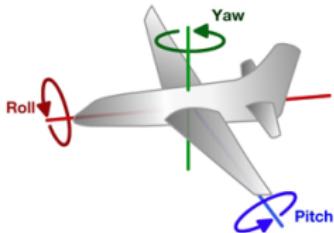
```
1 // Begin I2C communications
2 Wire.begin();
3
4 // begin transmission to GY-521
5 Wire.beginTransmission(MPU_ADDR);
6
7 // Select and write to PWR_MGMT1 register
8 Wire.write(0x6B);
9 Wire.write(0); // set to 0 (wakes up MPU-6050)
10
11 // end transmission
12 Wire.endTransmission(true);
```

# Initializing MPU6050

```
1 // Using Wire.requestFrom()
2 // Wire.requestFrom(address, quantity, stop)
3
4 Wire.beginTransmission(MPU_ADDR);
5 Wire.write(0x3B); // starting with register 0x3B
6 Wire.endTransmission(false); // keep active.
7
8 // Request and then read 2 bytes
9
10 int16_t accel_z, accel_z_h, accel_z_l      // 16-bit int
11 Wire.requestFrom(MPU_ADDR, 2, true);
12 accel_z_h = Wire.read(); // x accel MSB
13 accel_z_l = Wire.read(); // x accel LSB
14
15 accel_x = accel_z_h << 8 | accel_z_l;    // what happens if declared int instead?
```

Note, the data is stored in Big Endian Byte Order. The most significant byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in the next byte.

# Assignment: L16\_Motion



- Notebook:  
schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## ① L16\_01\_MP6050

- Connect the GY-521 and OLED
- Read the memory addresses for X, Y, and Z acceleration
- Convert the returned bytes to acceleration measured in G.

## ② L16\_02\_AutoRotate

- Create a Clock on your OLED
- Use GY-521 to auto-rotate the OLED

## ③ L16\_03\_Airplane

- Calculate pitch  $\theta = \arcsin(a_x)$
- Calculate roll  $\phi = \arctan2(a_y/a_z)$

## ④ L16\_04\_Shock

- Store  $a_{tot}$  in an array every 10ms for 5s
- Find and print the max value
- Repeat

# Pitch and Roll Improved

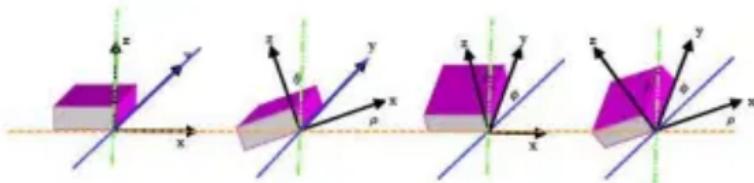


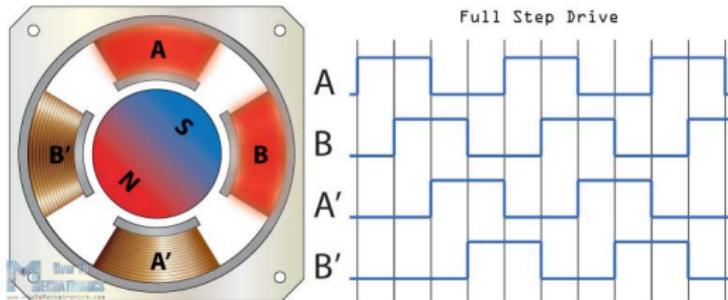
Figure 8. Three Axis for Measuring Tilt

$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}\right)$$

$$\theta = \arctan\left(\frac{\sqrt{A_X^2 + A_Y^2}}{A_Z}\right)$$

# Stepper Motors



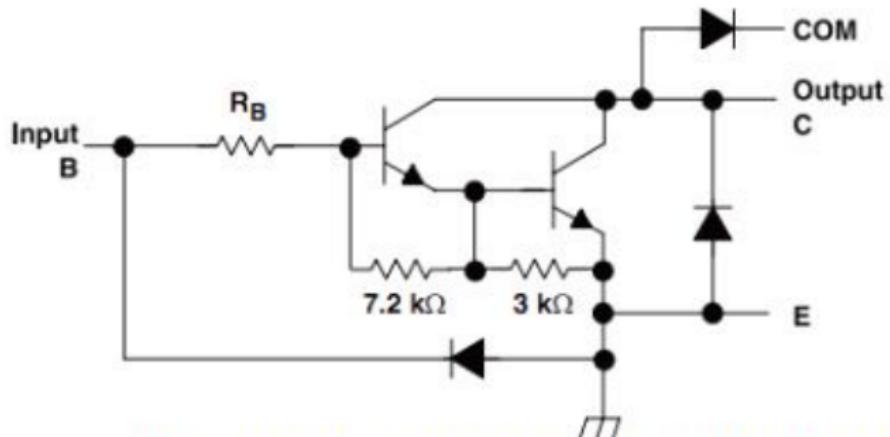
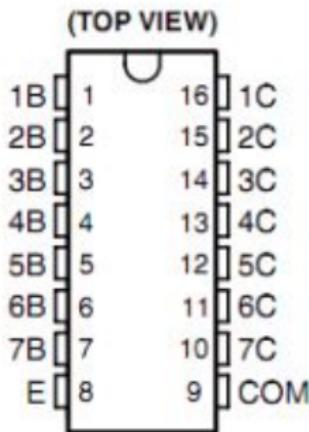
## 28BYJ Stepper Motor

- 2048 steps per revolution
  - 32 steps per rotor revolution
  - Gear ratio 1:64
- Capable of 10-15 RPM (at 5V)
- ULN2003 Darlington Array driver

## Stepper.h

- Stepper myStepper (spr,IN1,IN3,IN2,IN4)
- myStepper.setSpeed(speed)
- myStepper.step(steps)

# ULN2003 Darlington Array



ULN2003A, ULN2003AI, ULN2004A, ULQ2003A, ULQ2004A

# Assignment: L16\_Motion



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

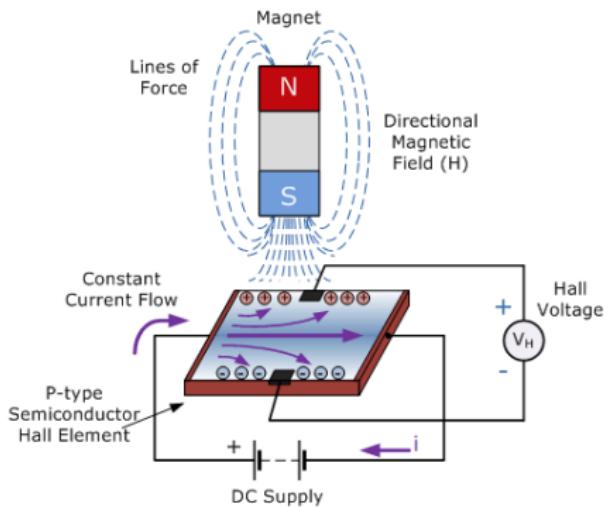
## ① L16\_05\_StepperGyro

- Wire the Stepper Motor noting the order: IN1, IN3, IN2, IN4
- Move the motor 2 rotations clockwise, pause, 1 rotation counter-clockwise, repeat.

## ② L16\_06\_Gyro

- Connect the GY-521 to your system
- Obtain the z-axis rotation from the appropriate register on the GY-521.
- Map the change in gyro position (signed 16-bit) to one revolution of the Stepper (signed 11-bit)
- Move the stepper the amount of steps corresponding to the mapped gyro change.

# Hall Effect Sensor



Discovered by Edwin Hall in 1879, the Hall Effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and to an applied magnetic field perpendicular to the current.

# Assignment: L16\_Motion



## ① L16\_07\_Alarm

- Connect Hall Effect Sensor, button, and Neopixel
- Use the button to enable / disable alarm
- When armed:
  - Green when magnet detected
  - Blinking Red when magnet not detected

## ② L16\_08\_RPM

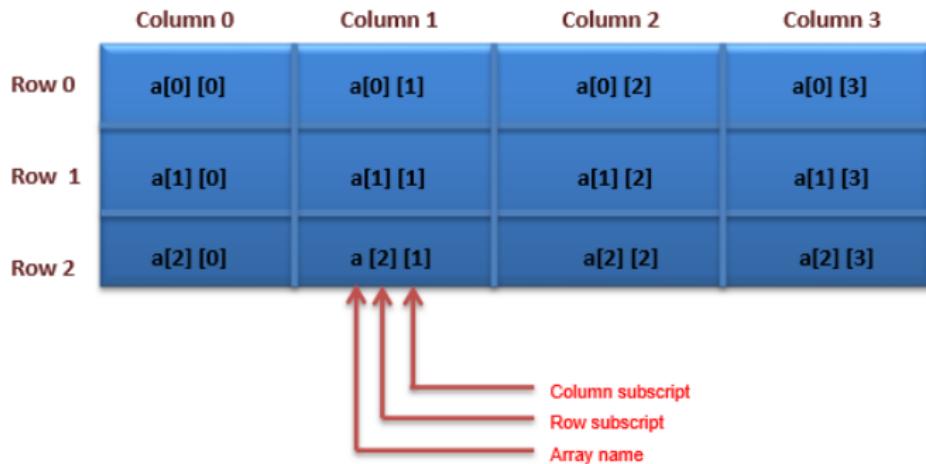
- Notebook:  
schematic
- Fritzing diagram
- Wire your circuit
- Write the code
- Place magnet on shaft of Lathe
- Using Hall Sensor, measure time per rotation
- Convert to rotations per minute
- Display on Adafruit.io databoard
- EXTRA: Create a jig to hold the sensor
- EXTRA: Measure RPM on other equipment at FUSE.

# Piezoelectric Elements



- The piezoelectric effect is the appearance of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an `analogRead()`, the vibration (change in mechanical stress) can be monitored over time.

# 2-dimensional arrays



- Declare Array: `int a[3][4] = {{0,1,2,3},{4,5,6,7},{8,9,10,11}};`
- Set a Cell: `a[1][2] = 7;`
- Access a Cell: `x = a[0][0];`

# Assignment: L16\_Motion

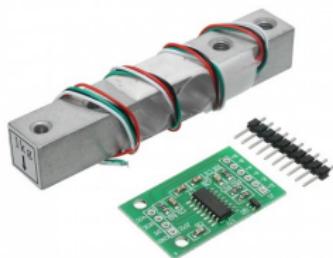


## ① L16\_09\_Vibration

- Connect the Piezo and a Button to your Argon
- Executing a loop 4096 times:
  - Every  $500\mu\text{sec}$ , collect piezoelectric data
  - Save the pizeo data and a time stamp (in seconds) to a 2-dimensional array
- When the loop is complete, time stamp and data to a file.
- Collect vibration data from Lathe, cellphone vibration, other machines at FUSE
- Use Excel and the FFT Tutorial (Class\_Slides) to resample graph data in frequency domain (this process will be reviewed as a class).

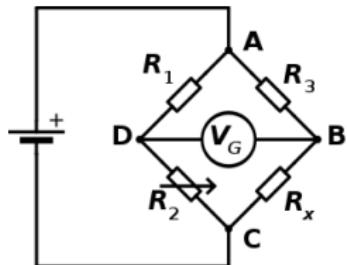
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

# Load Cells



- ① A load cell is a force transducer. It converts a force such as tension, compression, pressure, or torque into an electrical signal that can be measured and standardized. As the force applied to the load cell increases, the electrical signal changes proportionally. The most common types of load cells used are hydraulic, pneumatic, and strain gauge.
- ② HX711 module is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

# Wheatstone Bridge

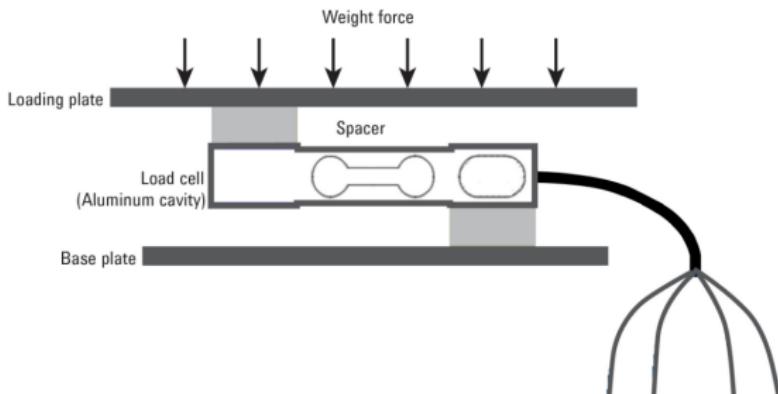


- ① The Wheatstone bridge was invented by Samuel Hunter Christie in 1833 and improved and popularized by Sir Charles Wheatstone in 1843.
- ② A Wheatstone bridge is an electrical circuit used to measure an unknown electrical resistance by balancing two legs of a bridge circuit, one leg of which includes the unknown component.
- ③ The primary benefit of the circuit is its ability to provide extremely accurate measurements (in contrast with something like a simple voltage divider)

# HX711A Library

```
1 #include "HX711.h"
2 HX711 myScale(DOUT,CLK);
3
4 void setup() {
5     myScale.set_scale();
6     delay(5000);          // let scale settle
7     myScale.tare();        // set the tare weight (or zero)
8
9     myScale.set_scale(cal_factor); //adjust calibration
10 }
11
12 void loop() {
13 // Using data from loadcell
14 // samples is the number of samples to average
15
16 offset = myScale.get_offset();
17 calibration = myScale.get_scale();
18
19 // rawData is the raw loadcell reading minus the offset
20 rawData = myScale.get_value(samples);
21
22 // value is the rawData divided by calibration
23 value = myScale.get_units(samples);
24
25 // more stable is short weight between readings
26 delay(5000);
27 }
```

# Assignment: L16\_Motion (Learn to Calibrate)



## ① L16\_10\_Scale

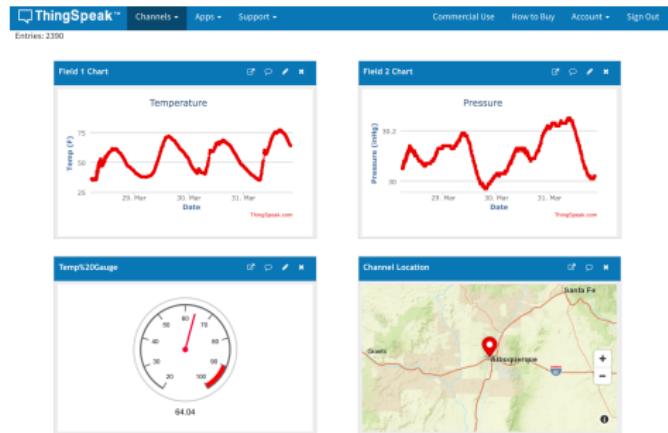
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code
- Set initial cal\_factor to 1000 and measure calibration\_weight
- Revise cal\_factor to get proper measurement in grams
- Post data to Adafruit.io (once / min)
- Send text via IFTTT

# IFTTT



If This Then That, also known as IFTTT, is a freeware web-based service that creates chains of simple conditional statements, called applets. An applet is triggered by changes that occur within other web services such as Gmail, Facebook, Telegram, Instagram, or Pinterest.

# ThingSpeak Follow Along



We will be learning how to use Particle Webhooks to publish between cloud services. For this, you will need:

- 1 Particle Argon
- 2 BME280

# JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

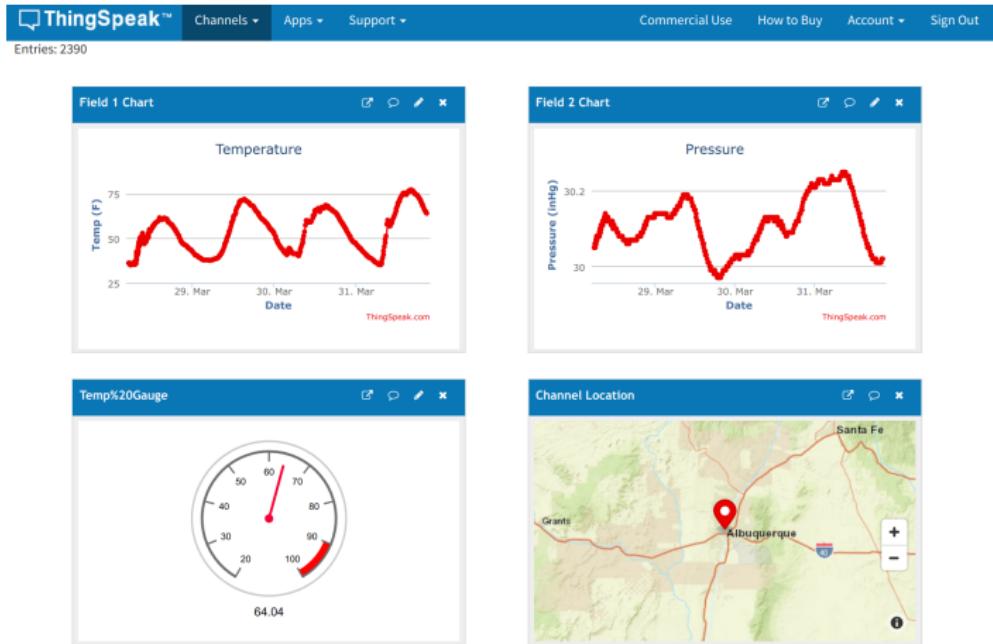
- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

# JSON Parser Generator

Creating objects in JSON are straightforward but can be tedious. There is a JSON Parser available to simplify the process.

```
1 #include <JsonParserGeneratorRK.h>
2
3 void createEventPayLoad(int moistValue, float tempValue, float presValue, float humValue,
4                         int waterED) {
5     JsonWriterStatic<256> jw;
6     {
7         JsonWriterAutoObject obj(&jw);
8
9         jw.insertKeyValue("Moisture", moistValue);
10        jw.insertKeyValue("Temperature", tempValue);
11        jw.insertKeyValue("Pressure", presValue);
12        jw.insertKeyValue("Humidity", humValue);
13        jw.insertKeyValue("Plant Watered", waterED);
14    }
15    Particle.publish("env-vals", jw.getBuffer(), PRIVATE);
}
```

# ThingSpeak Dashboard



# Step 1 - Create ThingSpeak Channel

## My Channels

New Channel						Search by tag		
Name						Created	Updated	
FUSEMakerspace						2020-01-09	2020-03-27 16:04	
Private	Public	Settings	Sharing	API Keys	Data Import / Export			
Home IoT Plant Watering						2020-04-16	2020-04-16 19:56	
Private	Public	Settings	Sharing	API Keys	Data Import / Export			
Dew Point Measurement						2020-04-16	2020-04-19 13:38	
Private	Public	Settings	Sharing	API Keys	Data Import / Export			
Home Weather Station						2020-04-17	2020-04-17 18:52	
Private	Public	Settings	Sharing	API Keys	Data Import / Export			

## Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click [New Channel](#) to create a new ThingSpeak channel.

Click on the column headers of the table to sort by the entries in that column or click on a tag to show channels with that tag.

Learn to [create channels](#), explore and transform data.

Learn more about [ThingSpeak Channels](#).

## Examples

- [Arduino](#)
- [Arduino MKR1000](#)
- [ESP8266](#)
- [Raspberry Pi](#)
- [Netduino Plus](#)

## Upgrade

Need to send more data faster?

Need to use ThingSpeak for a commercial project?

[Upgrade](#)

# Step 2 - Get API Key

**ThingSpeak™** Channels Apps Support Commercial Use How to Buy Account Sign Out

## Dew Point Measurement

Channel ID: 1039626  
Author: mraw0000017234878  
Access: Private

Private View Public View Channel Settings Sharing API Keys Data Import / Export

### Write API Key

Key  Generate New Write API Key

### Read API Keys

Key   
Note   
Save Note Delete API Key Add New Read API Key

### Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

### API Keys Settings

- Write API Key: Use this key to write data to a channel. If you feel your key has been compromised, click [Generate New Write API Key](#).
- Read API Keys: Use this key to allow other people to view your private channel feeds and charts. Click [Generate New Read API Key](#) to generate an additional read key for the channel.
- Note: Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

### API Requests

Write a Channel Feed  
GET [https://api.thingspeak.com/update?api\\_key=BK1I6D1UTGPQ5B5H](https://api.thingspeak.com/update?api_key=BK1I6D1UTGPQ5B5H)

Read a Channel Feed  
GET [https://api.thingspeak.com/channels/1039626\(feeds.json?tag](https://api.thingspeak.com/channels/1039626(feeds.json?tag)

Read a Channel Field  
GET [https://api.thingspeak.com/channels/1039626\(fields/1.json](https://api.thingspeak.com/channels/1039626(fields/1.json)

Read Channel Status Updates  
GET <https://api.thingspeak.com/channels/1039626/status.json?>

[Learn More](#)

# Step 3 - Create Webhook

From `console.particle.io`:

The screenshot shows the Particle Argon web interface with the following details:

- Left Sidebar:** Contains icons for Device, Project, Data, Code, and Help.
- Header:** Personal, Docs, Contact Sales, Support, and barashap@gmail.com.
- Section:** Integrations
- Existing Integrations (Four cards):**
  - Webhook**
    - bme-vals
    - Lalonde
    - thingspeak.com
  - Webhook**
    - temp
    - any device
    - thingspeak.com
  - Webhook**
    - FUSEMakerspa...
    - any device
    - thingspeak.com
  - Webhook**
    - env-vals
    - Herbert
    - thingspeak.com
- New Integration Placeholder:** A dashed box with a plus sign and the text "NEW INTEGRATION".

# Step 4 - Add JSON Data

Personal ☰

Integrations | Edit Integration

WEBHOOK BUILDER CUSTOM TEMPLATE

Read the Particle webhook guide

Event Name: env-vals

URL: https://api.thingspeak.com/update

Request Type: POST

Request Format: Web Form

Device: Herbert

Advanced Settings

For information on dynamic data that can be sent in any of the fields below, please visit [our docs](#).

FORM FIELDS

Custom

api_key	> XXXXXXXXXXXXXXXXXXXX	x
field1	> {{(Moisture)}}	x
field2	> {{(Temperature)}}	x
field3	> {{(Pressure)}}	x
field4	> {{(Humidity)}}	x
field5	> {{(Plant Watered)}}	x

# Step 5 - Particle Cloud Events

Personal ⚙

Docs | Contact Sales | Support | barashap@gmail.com •

## Events

Search for events ADVANCED

NAME	DATA	DEVICE	PUBLISHED AT
spark/status	offline	Herbert	4/20/20 at 10:06:49 am
spark/status	offline	Lalonde	4/20/20 at 10:06:26 am
hook-response/env-vals/0	1236	particle-internal	4/20/20 at 10:06:16 am
hook-sent/env-vals		particle-internal	4/20/20 at 10:06:16 am
env-vals	{"Moisture":2392,"Temperature":66.650000,"Pressure":30.033356,"Humidity":22.477539,"Plant Watered":0}	Herbert	4/20/20 at 10:06:16 am
Plant Watered	0	Herbert	4/20/20 at 10:06:16 am
Temperature	66.650000	Herbert	4/20/20 at 10:06:16 am
Moisture	2392	Herbert	4/20/20 at 10:06:16 am
spark/status	offline	Herbert	4/20/20 at 10:01:48 am

**env-vals**

Published by e00fce6873080a74a8599312 on 4/20/20 at 10:06:16 am

PRETTY RAW

⌘ {  
  "Moisture" : 2392  
  "Temperature" : 66.650000  
  "Pressure" : 30.033356  
  "Humidity" : 22.477539  
  "Plant Watered" : 0  
}

# Step 6 - Create Channel

## Home IoT Plant Watering

Channel ID: 1039355  
Author: mwaw0000017234878  
Access: Public

Plant Watering in Home IoT Classroom

Private View    Public View

Channel Settings

Sharing

API Keys

Data Import

### Channel Settings

Percentage complete 50%

Channel ID 1039355

Name Home IoT Plant Watering

Description Plant Watering in Home IoT Classroom

Field 1 Moisture

Field 2 Temperature

Field 3 Pressure

Field 4 Humidity

Field 5 Watered

Field 6

Field 7

Field 8

Metadata JSON

### Help

Channels  
eight field  
status dat  
visualize i

### Chanr

- Per cha cha
- Chr
- Del
- Fiel cha
- Mel
- Tag
- Lin Thi
- Shc

- Vid infc

# Step 7 - Create Dashboard

You can change the colors of your lines, by editing the graph. The hex codes are found at <https://htmlcolorcodes.com/color-picker/>

## Home IoT Plant Watering

Channel ID: 1039355

Author: mwa0000017234878

Access: Public

Plant Watering in Home IoT Classroom



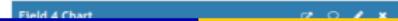
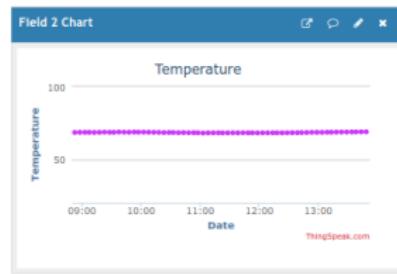
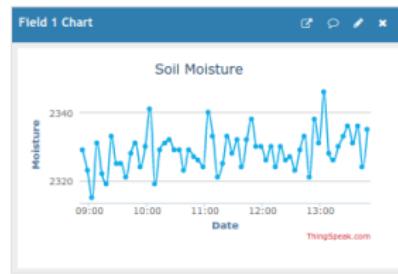
Channel 2 of 4 < >

### Channel Stats

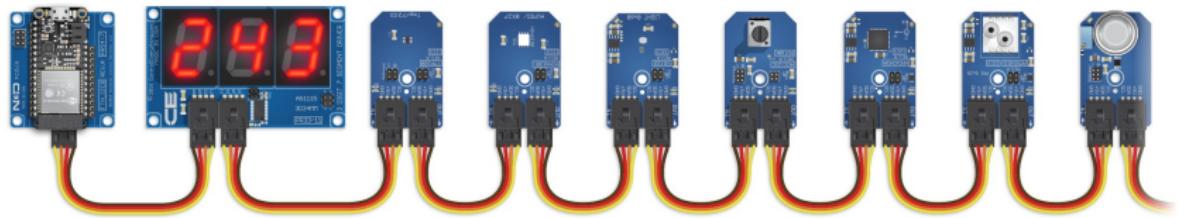
Created: 3 days ago

Last entry: 2 minutes ago

Entries: 994



# NCD.io Control Everywhere



# NCDio TMG3993 Proximity/Color Sensor



- From Address 0x94
- Request 9 bytes
  - Byte 0 / 1 - LSB / MSB of Infrared Luminance
  - Byte 2 / 3 - LSB / MSB of Red Luminance
  - Byte 4 / 5 - LSB / MSB of Green Luminance
  - Byte 6 / 7 - LSB / MSB of Blue Luminance
  - Byte 9 - Proximity

# TMG3993 Initialization

```
1 Serial.println("Initializing TMG3993");
2 Wire.beginTransmission(Addr);
3 // Select Enable register
4 Wire.write(0x80);
5 // Power ON, ALS enable, Proximity enable, Wait enable
6 Wire.write(0x0F);
7 Wire.endTransmission();
8
9 Wire.beginTransmission(Addr);
10 // Select ADC integration time register
11 Wire.write(0x81);
12 // ATIME : 712ms, Max count = 65535 cycles
13 Wire.write(0x00);
14 Wire.endTransmission();
15
16 Wire.beginTransmission(Addr);
17 // Select Wait time register
18 Wire.write(0x83);
19 // WTIME : 2.78ms
20 Wire.write(0xFF);
21 Wire.endTransmission();
22
23 Wire.beginTransmission(Addr);
24 // Select control register
25 Wire.write(0x8F);
26 // AGAIN is 1x
27 Wire.write(0x00);
28 Wire.endTransmission();
29 }
```

# NCDio ACD121C MQ9 CO Sensor



## 12-Bit Analog to Digital Conversion

- From Address 0x00
- Request 2 bytes (raw\_adc)
  - Byte 0 - MSB
  - Byte 1 - LSB
- CO (ppm) =  $\frac{1000}{4096} * raw\_adc + 10$

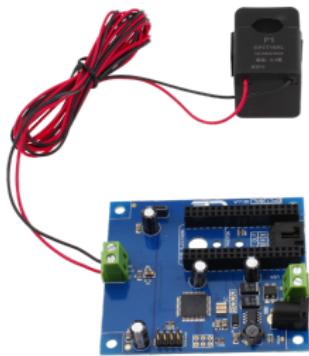
# NCDio ACD121C MQ131 Ozone Sensor



## 12-Bit Analog to Digital Conversion

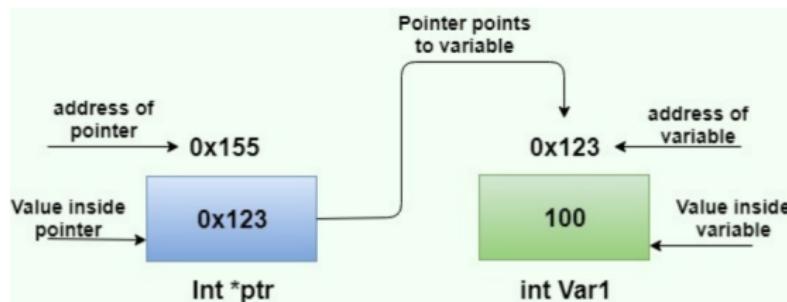
- From Address 0x00
- Request 2 bytes (raw\_adc)
  - Byte 0 - MSB
  - Byte 1 - LSB
- $O_3 \text{ (ppm)} = \frac{1.99}{4095} * raw\_adc + 0.01$

# NCDio PECMAC Current Sensor



- 1 to 8 channels
- Full range between 10 and 100 amps
- Simple to use. Simply run an AC power wire through the opening of the current sensor. This controller will read the magnetic field inducted onto the current sensor and provide you with a real-world current measurement value that is 98 percent accurate (prior to calibration).

# Pointers



- ① A pointer is a variable whose value is the address of another variable.
- ② When you declare a pointer, the `*` symbol denotes that this variable is a pointer variable. For example:
  - Pointer to an Integer: `int *ptr;`
- ③ Reference operator (`&`) gives the address of a variable.
- ④ To get the value stored in the memory address, we use the dereference operator (`*`).

# Pointers

```
1 int data = 13;
2 int data2;
3 int *ptr;
4
5 void setup() {
6   Serial.begin(9600);
7   delay(1000);
8   ptr = &data;           //point ptr to the memory location of data
9   data2 = *ptr;         //set data2 to value of data (13)
10
11 // Print the Value and Address of the Variables
12 Serial.printf("Variable      Value      Address \n");
13 Serial.printf("  data        %i        0x%X  \n",data, &data);
14 Serial.printf("  ptr         0x%X        0x%X  \n",ptr, &ptr);
15 Serial.printf("  data2       %i        0x%X  \n",data2,&data2);
16 }
```

Serial monitor opened successfully:

Variable	Value	Address
data	13	0x2003E380
ptr	0x2003E380	0x2003E3F4
data2	13	0x2003E3F0

# Finding Average of an Array

```
1 // This function finds the average of an array
2 // The array is passed to it as a pointer
3
4 float getAve(int *array,int size) {
5     int j;
6     float total=0;
7     for(j=0;j<size;j++) {
8         total += array[j];
9     }
10    return total/size;
11 }
```

# Finding Average of Arrays in Action

```
1 int xArray[4], yArray[256];
2 int *pointerX, *pointerY;
3 float average;
4 int i, sizeX, sizeY;
5
6 void setup() {
7     pointerX=&xArray[0];
8     sizeX=sizeof(xArray)/4;
9     pointerY=&yArray[0];
10    sizeY=sizeof(yArray)/4;
11
12    for(i=0;i<sizeX;i++) {
13        xArray[i] = random(0,255);
14    }
15    for(i=0;i<sizeY;i++) {
16        yArray[i] = random(256,512);
17    }
18
19    average = getAve(pointerX, sizeX);
20    average = getAve(pointerY, sizeY);
```

```
Array X Average = 162.50
Array Y Average = 388.35
xArray[0] value: 173, *pointerX: 173, pointerX: 0x2003E3DC
xArray[1] value: 179, *(pointerX+1): 179, pointerX+1: 0x2003E3E0
xArray[2] value: 110, *(pointerX+2): 110, pointerX+2: 0x2003E3E4
xArray[3] value: 188, *(pointerX+3): 188, pointerX+3: 0x2003E3E8
```

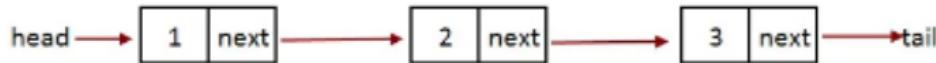
# Struct datatype and Member Operators

```
1 struct geo {
2     float lat;
3     float lon;
4     int alt;
5 };
6 geo myLoc;
7 geo *ptLoc;
8
9 void setup() {
10     Serial.begin(9600);
11     delay(500);
12
13     ptLoc = &myLoc;
14     myLoc.lat = 35.120606;
15     myLoc.lon = -106.65818;
16     myLoc.alt = 1517;
17
18     Serial.printf("Location: lat %f, lon %f, alt %i \n",myLoc.lat,myLoc.lon,myLoc.alt);
19     Serial.printf("Location: lat %f, lon %f, alt %i \n",ptLoc->lat,ptLoc->lon,ptLoc->alt);
20 }
```

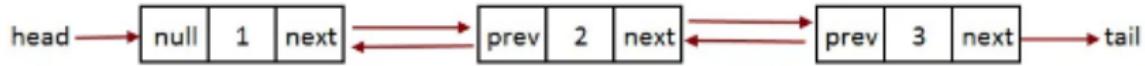
The . (dot) operator and the -> (arrow) operator are used to reference individual members of structures.

- The dot operator is applied to the actual object.
- The arrow operator is used with a pointer to an object.

# Linked Lists and Doubly Linked Lists

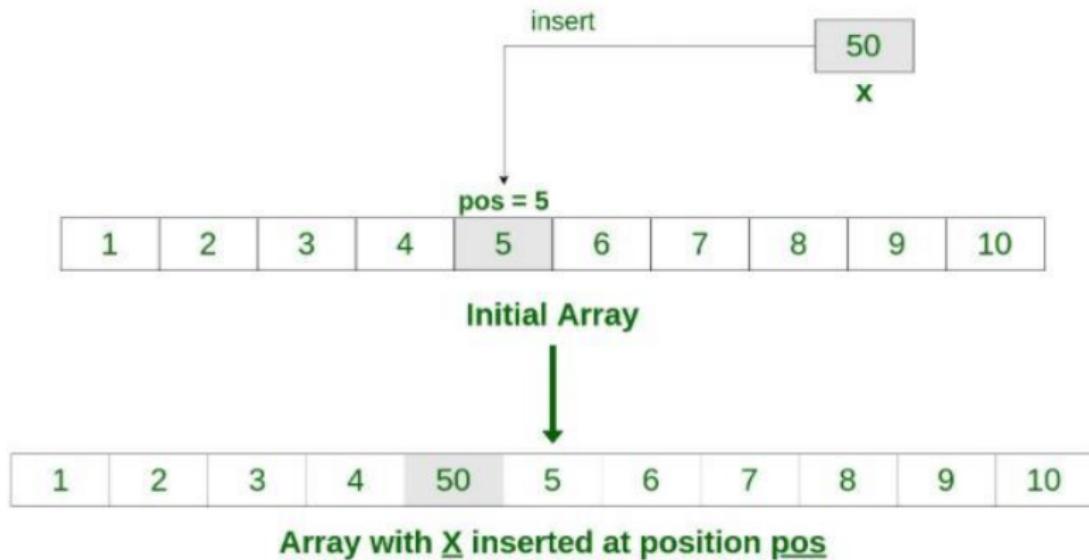


Singly Linked List

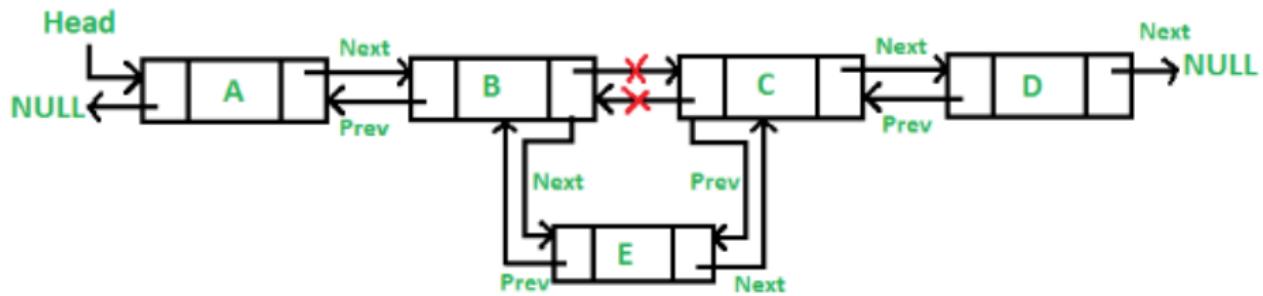


Doubly Linked List

# Inserting a "cell" into an Array



# Inserting a "cell" into a Linked List



# Capstone Projects

Intent:

- Based on a direct observation or a need expressed by one of our guest speakers.
- Original work demonstrating the skills obtained in this class.
- Demonstrate the ability to work as part of a team.
- A pitch to potential employers or investors.

Guidelines:

- Break class into 3 or 4 teams.
- Practical application of smart home, manufacturing, city environment or immersive entertainment.
- Needs to include a Cloud Dashboard component.
- Project will include a video presentation as well as a Hackster.io post.

Previous capstone projects can be found at: <https://www.youtube.com/playlist?list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey>

# Capstone Project Timeline

- ① Select Capstone teams (Aug 12)
- ② Teambuilding exercise with Sue/Esteban (Aug 12)
- ③ Review of Rev 0 ideas with instructors (Aug 20)
- ④ Start work on CapStone (Aug 26)
- ⑤ Videos due to Digital Media (Sept 4)
- ⑥ Employer Roundtable (Sept 11)

# The Big Question: What about main()

```
1 #include <Arduino.h>
2
3 extern "C" int main(void)
4 {
5 #ifdef USING_MAKEFILE
6
7 // To use Teensy 3.0 without Arduino, simply put your code here.
8 // For example:
9
10 pinMode(13, OUTPUT);
11 while (1) {
12     digitalWriteFast(13, HIGH);
13     delay(500);
14     digitalWriteFast(13, LOW);
15     delay(500);
16 }
17
18
19#else
20 // Arduino's main() function just calls setup() and loop()....
21 setup();
22 while (1) {
23     loop();
24     yield();
25 }
26#endif
27 }
```



# Thevenin Equivalent Circuits

Léon Charles Thévenin ( 30 March 1857 – 21 September 1926) was a French telegraph engineer who extended Ohm's law to complex circuits.

Any combination of batteries and resistances with two terminals can be replaced by a single voltage source  $V_{th}$  and a single series resistor  $R_{th}$ .

## Useful relationships

Equivalent Circuit

