

# IoT Product Design and Coding Bootcamp

Brian Rashap, Ph.D.

24-Feb-2020

# Table of contents

- 1 Introduction
- 2 Smart Room Controller
- 3 Communications Buses
- 4 The Internet
- 5 Particle Argon

# IoT Fun



© marketoonist.com

# Brian Rashap, Ph.D.

- Proud husband of Krista and father of Shelby (21) 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

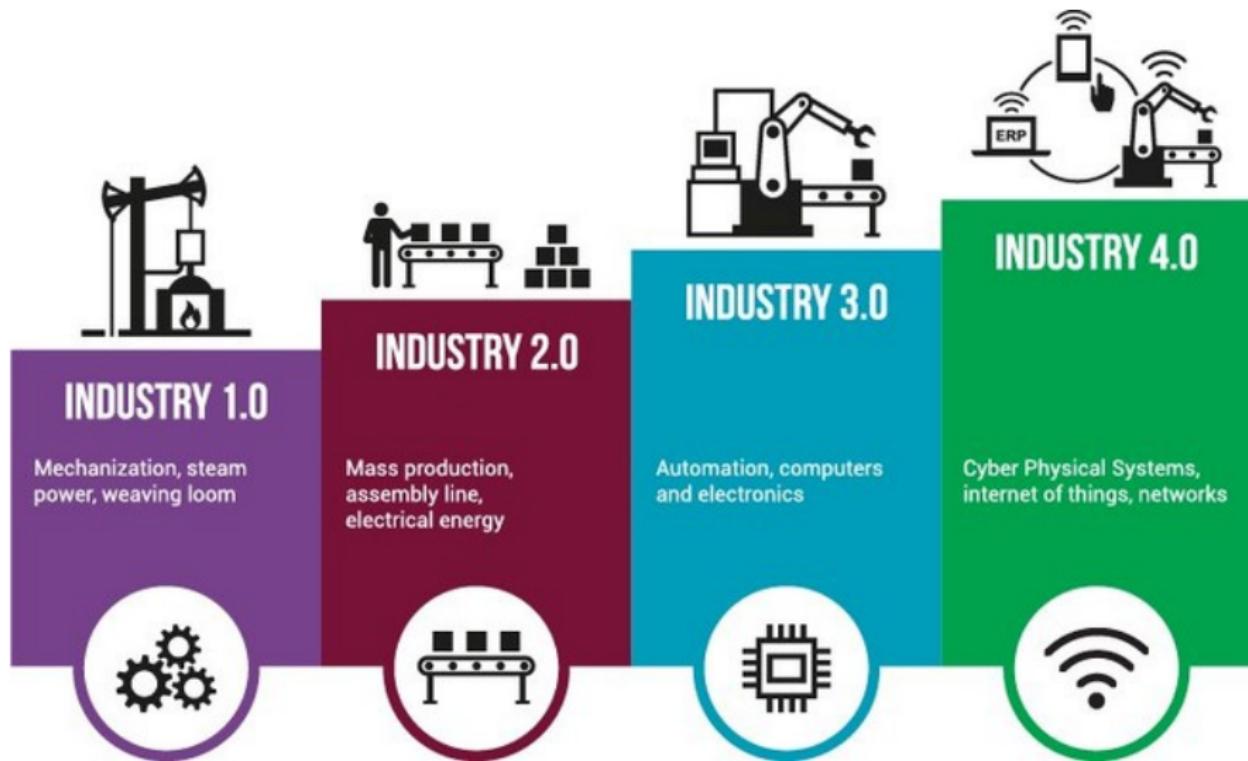
- Respect Each Other, Help Each Other
- Ask Questions
- Be On Time (let's us know via Slack if you won't be here)
- Keep Your Workspace and the Classroom Neat and Tidy
- If you are struggling, let myself, Susan, or Esteban know. We are here to HELP!

# Grading

- ① Weekly IoT assignments: 40
- ② Lab Notebook: 10
- ③ Weekly quizzes: 10
- ④ Individual Midterm Project: Smart Room Controller: 20
- ⑤ Team Capstone Project: 20

More information later today on how assignments are turned in

# Evolution of Industry



# Components of Industry 4.0



# IoT and Data Science



**AI:** Data-based learning

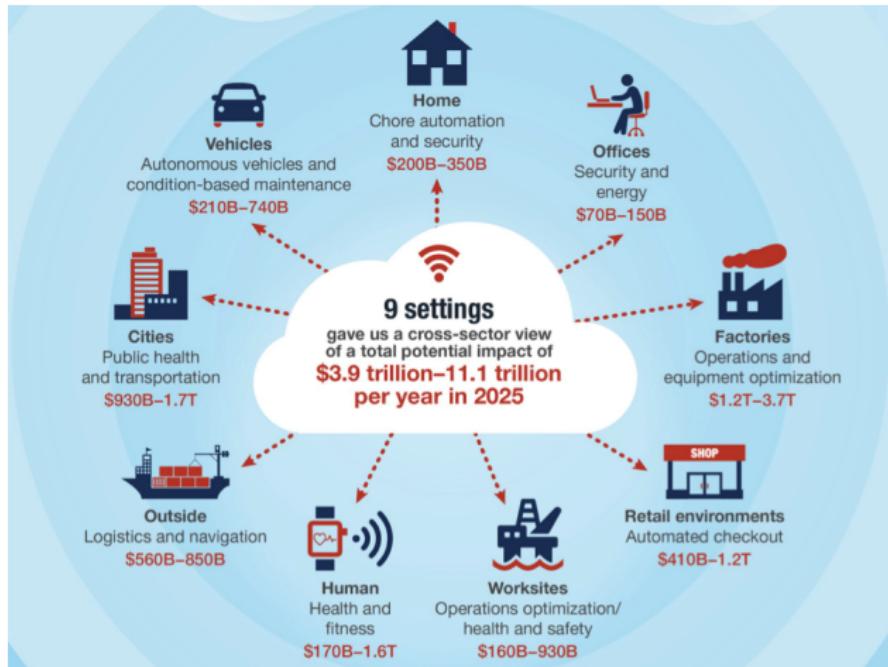


**Big Data:** Capture, storage, analysis of data



**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 2,300-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 endangered 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 coders 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



# 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 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

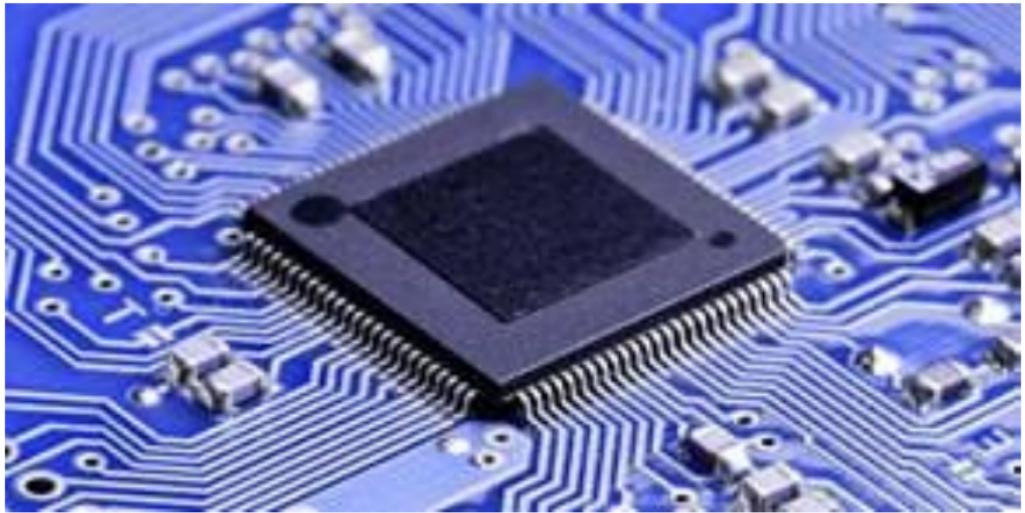
- cat: It is generally used to concatenate the files. It gives the output on the standard output.
- more: It is a filter for paging through text one screenful at a time.

The below commands are not available in PowerShell:

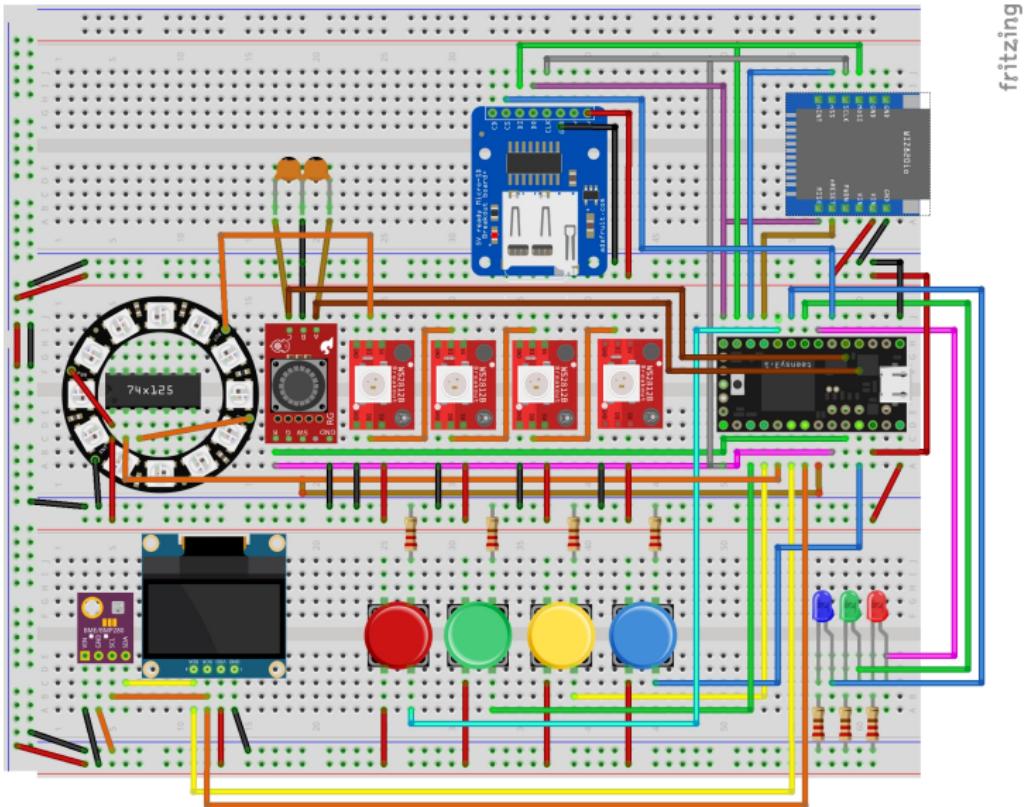
- less: It is used to viewing the files instead of opening the file. Similar to more command but it allows backward as well as forward movement.
- head: Used to print the first N lines of a file. It accepts N as input and the default value of N is 10.
- tail: Used to print the last N-1 lines of a file.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

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

# Our First Microcontroller

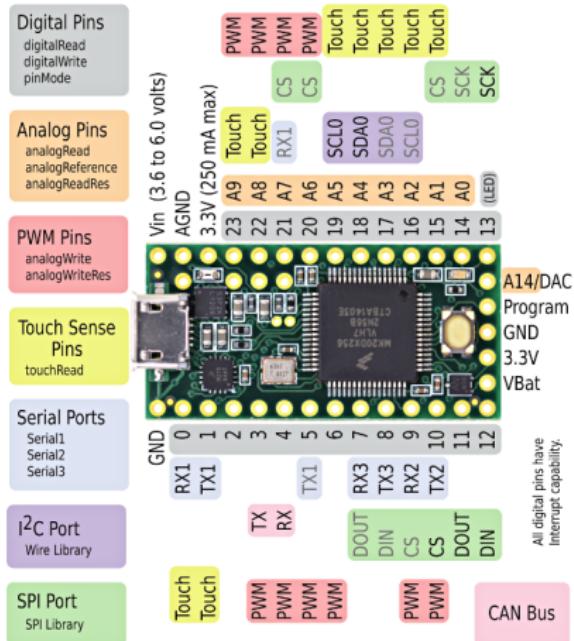


# Smart Room Controller



# 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 make 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 Fritzing

- IoT Bootcamp Teams Site

## ② Drawio

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

## ③ Fusion 360

- Instructional Videos - Teams Site

## ④ Formlab's Preform

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

## ⑤ Ultimaker's Cura

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

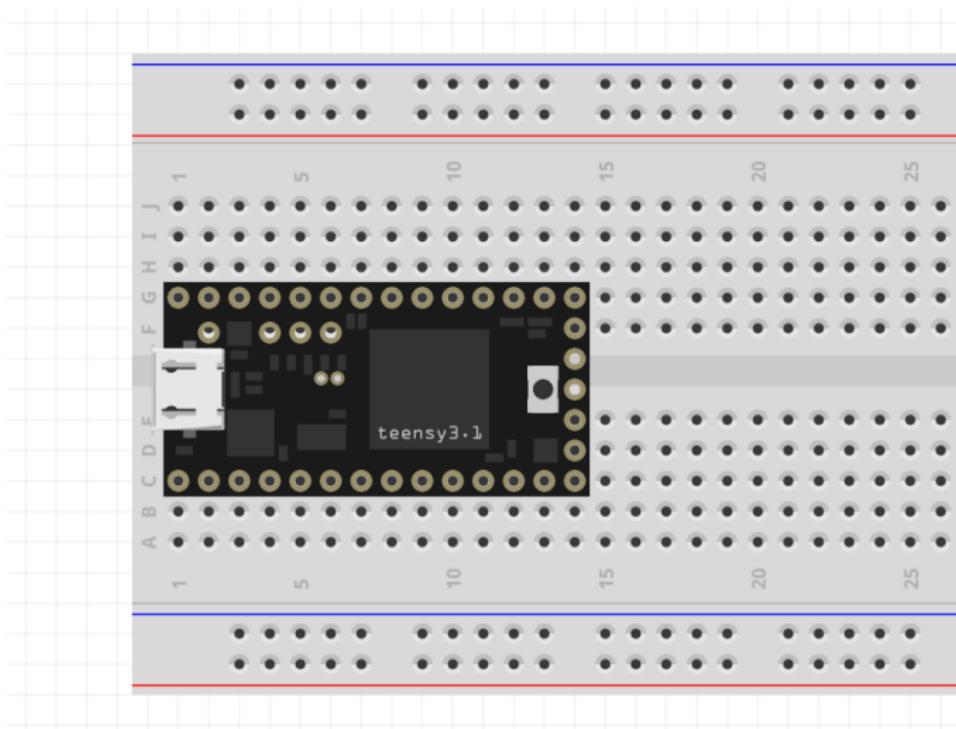
## ⑥ Git

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

# 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"
```

# 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
```

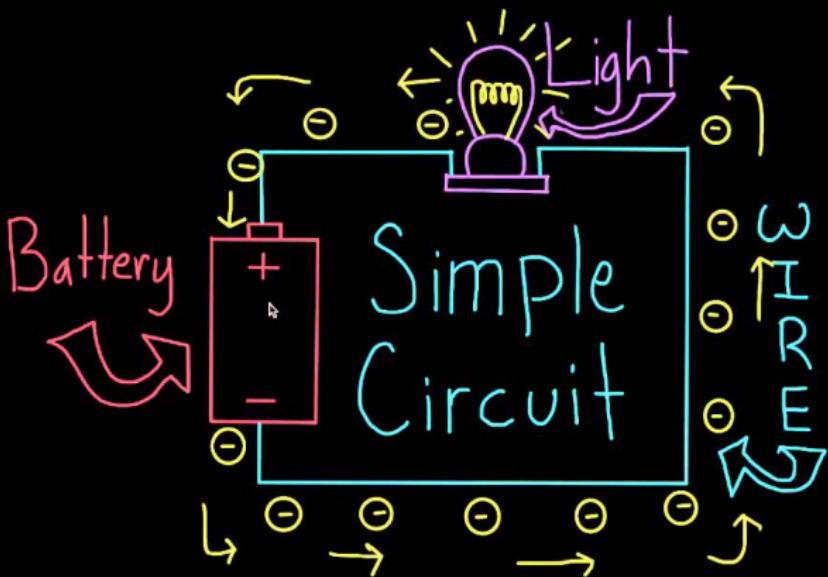
# Assignment L01\_01\_HelloWorld



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

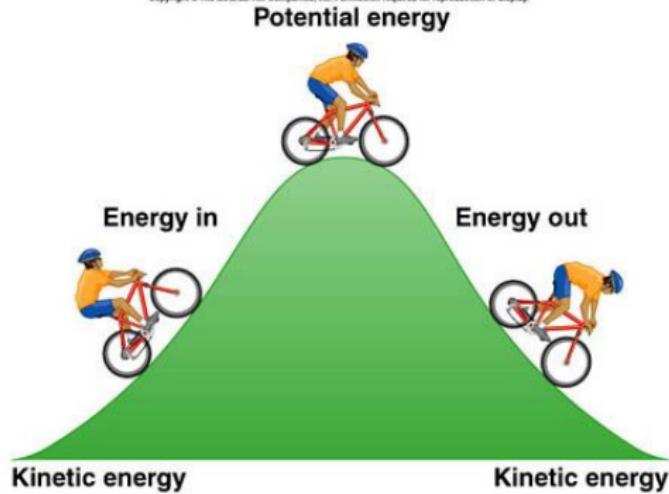
- `pinMode(pin,mode)`
- `digitalWrite(pin,state)`
- `delay(delay_time)`

# 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%

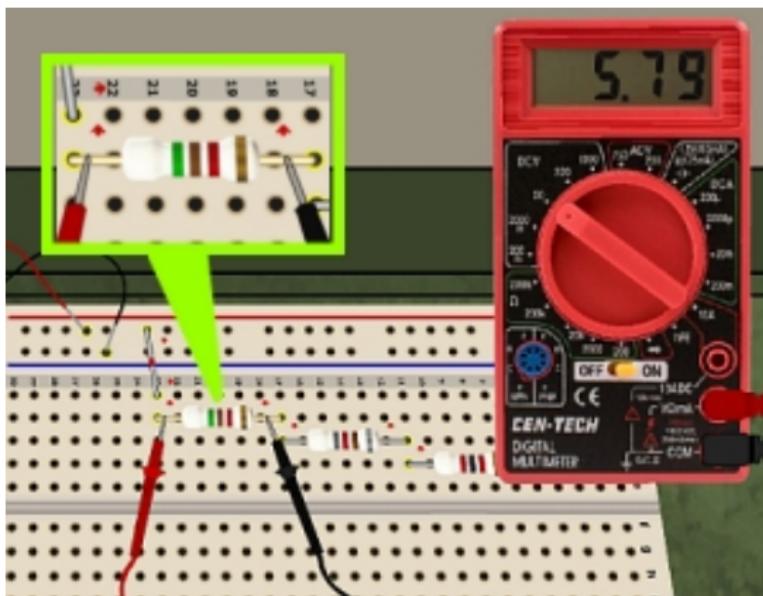
COLOR	1 <sup>ST</sup> BAND	2 <sup>ND</sup> BAND	3 <sup>RD</sup> BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%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



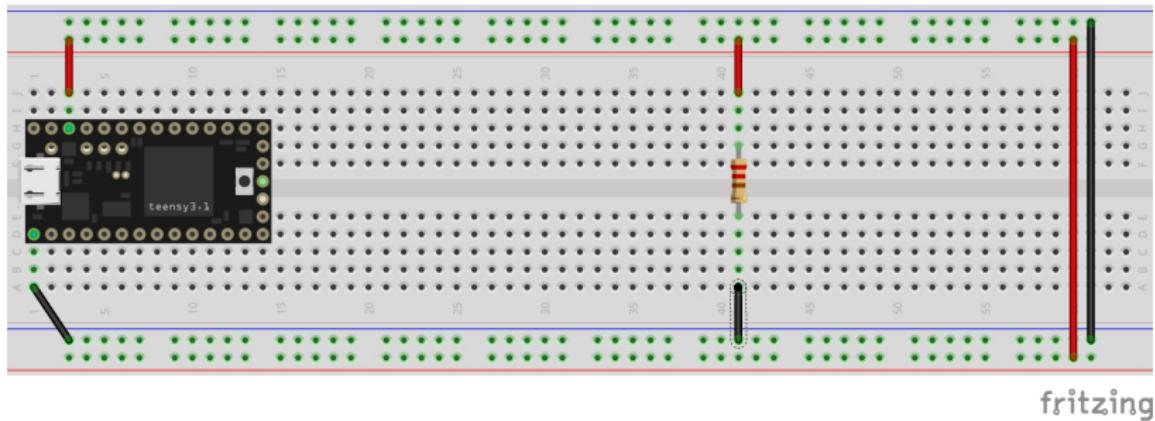
# Power from the Teensy 3.2



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

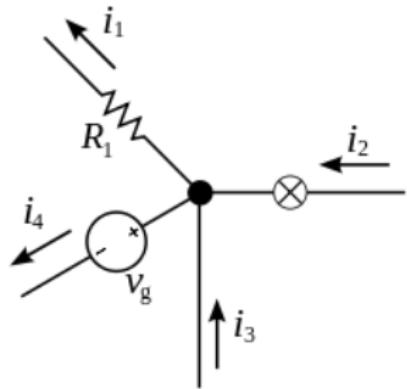


fritzing

# Kirchhoff's First Law

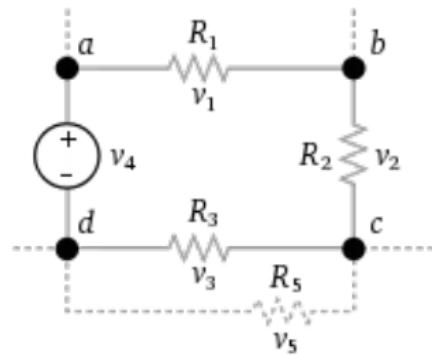
Gustav Robert Kirchhoff (12 March 1824 – 17 October 1887) was a German physicist who contributed to the fundamental understanding of electrical circuits. His first law:

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

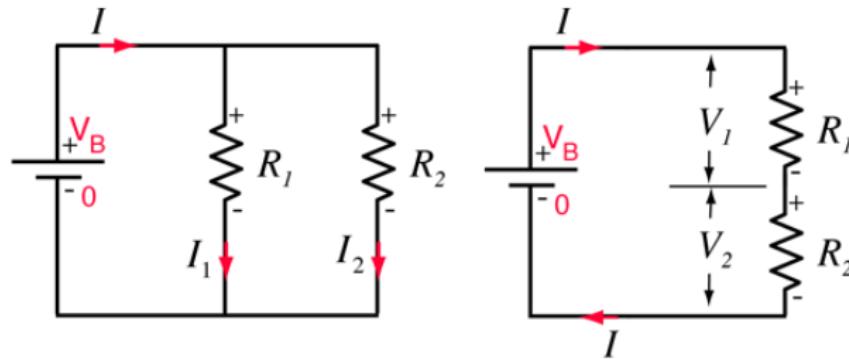
The directed sum of the potential differences (voltages) around any closed loop is zero.



# Resistors in Series and Parallel



# Resistors in Series and Parallel



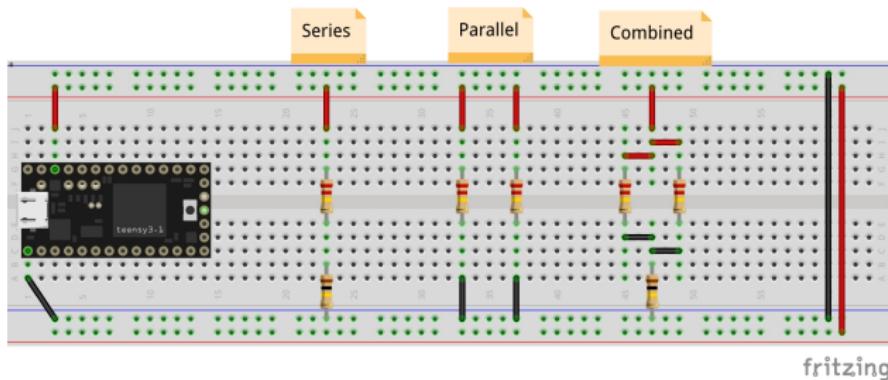
Parallel resistors

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

Series resistors

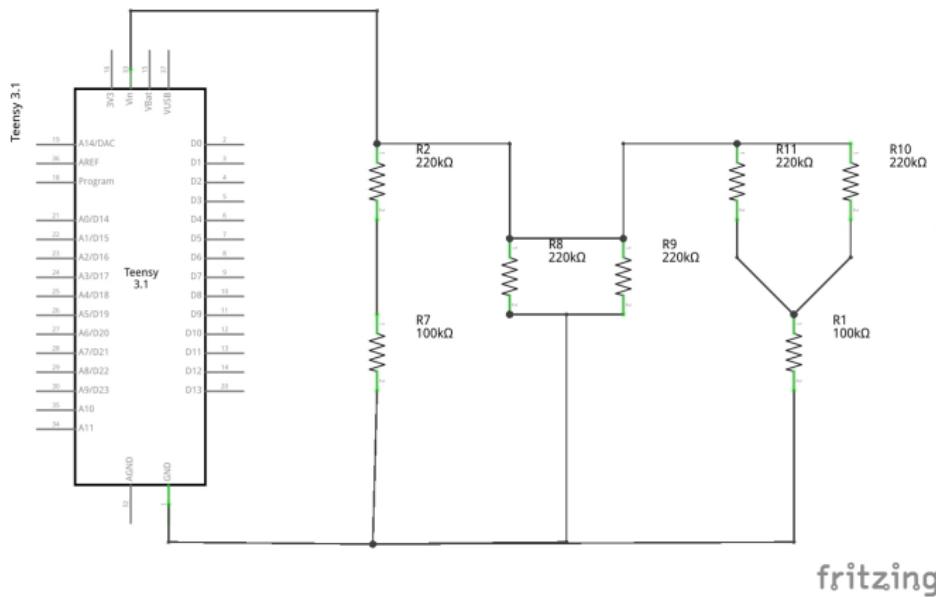
$$R_{\text{equivalent}} = R_1 + R_2$$

# Resistor Lesson



- In your lab notebook, draw the circuit diagrams
  - ① Series:  $220k\Omega$  and  $100k\Omega$
  - ② Parallel:  $220k\Omega$  and  $220k\Omega$
  - ③ Combined: Two  $220k\Omega$  in series with  $100k\Omega$
- Calculate the combined resistance and the voltage at each point
- Create Fritzing diagram
- Build (one at a time\*) on your breadboard and test with 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.

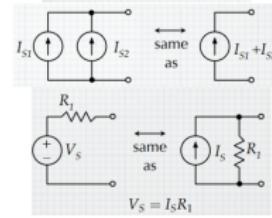
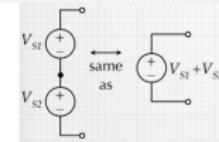
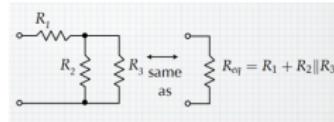
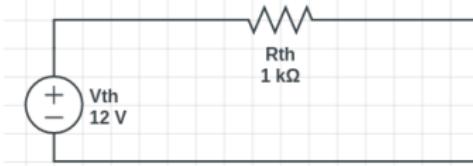
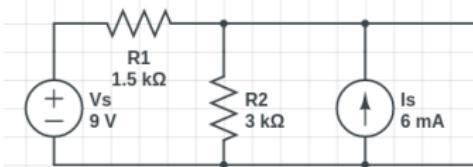
# Thevenin Equivalent Circuits

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  $R_{th}$ .

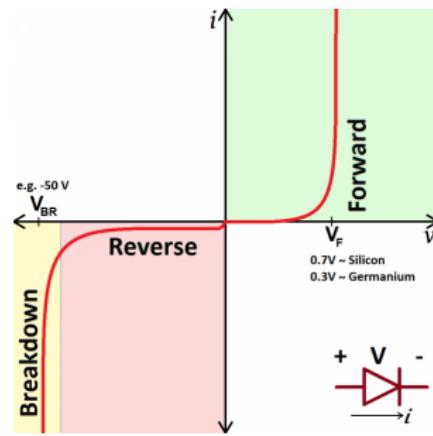
## Useful relationships

Equivalent Circuit



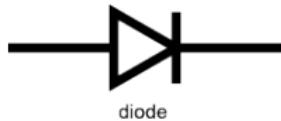
# Diodes

The key function of an 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



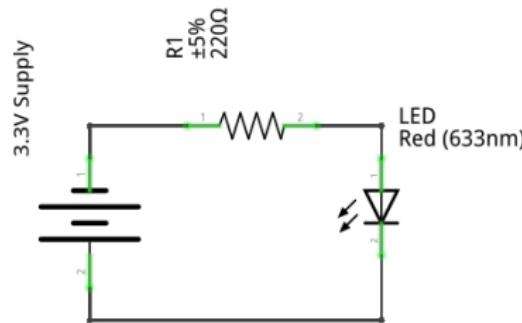
light emitting diode

# 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.
- Change the resistor to  $1\text{k}\Omega$  and then  $10\text{k}\Omega$ , what happens to the brightness?

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

# Constants and Variables

It is often useful to give a name to something that will be used repeatedly in the code. Then 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  $\pm 675,376,778$
- float**: a Floating point number with 7-digits precision

# Constant 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

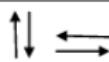


Use `analogWrite` to change the brightness of the LED, using values:

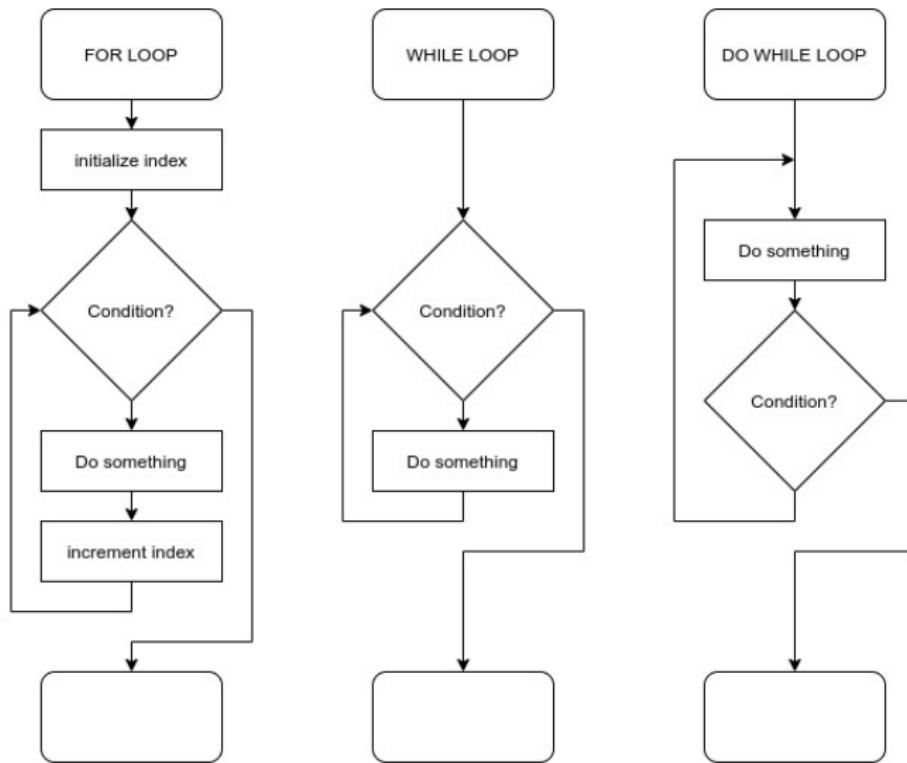
- 255
- 63
- 171
- 16

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
```

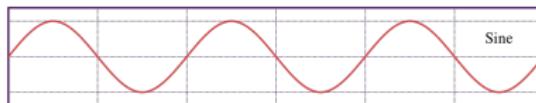
# For vs While Loops

## 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.

# 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.

# 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:

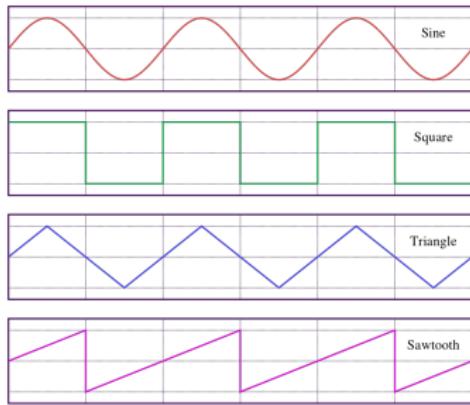
- `#include <file.h>` for system header files.
- `#include "file.h"` for user created header files in the directory that contains the current code.

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

# Basic Structure of Arduino Sketch Revisited

```
1 #include <file.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

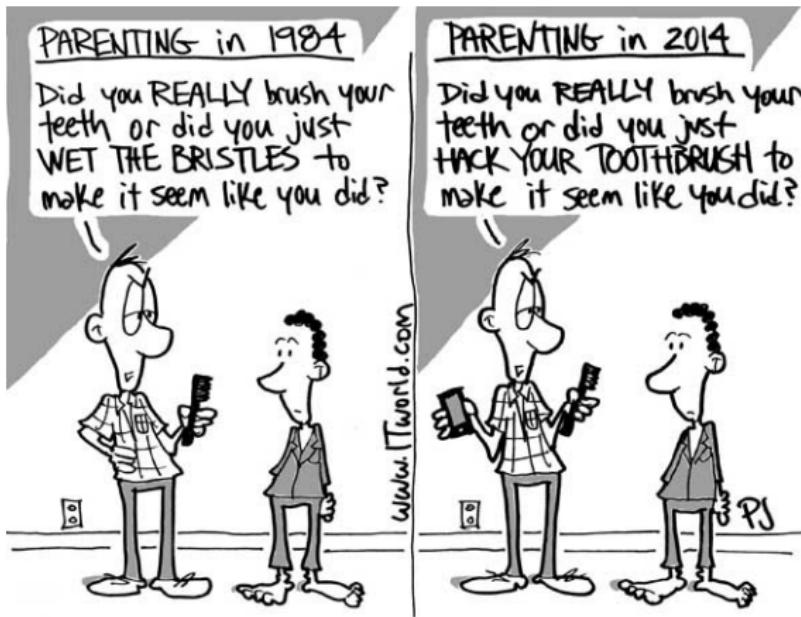


Use a `sin()` function to vary brightness of your LED

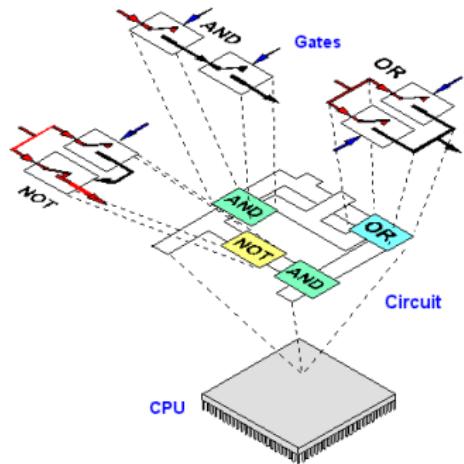
- Use `math.h`
- Function `sin` take a double as an input and returns a double.
- Set the period to 5 seconds. One cycle of  $\sin(2\pi n)$  for each integer value of  $n$ .

The function `millis()` returns milliseconds since Teensy has been powered on.  $n = (\text{millis}() / 5000) / \text{period in seconds}$ .

# 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(
9     pin2) == HIGH) {
10    //statements
11 }
12
13 // if either value is greater than zero
14 if (x > 0 || y > 0) {
15     // statements
16 }
```

# 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 }
```

# 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("Let's print an integer %i and a float %0.4f \n", int, 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();

# One Pin - Many Functions

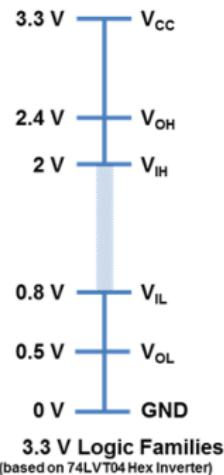


ai15939b

Software Programmable: Input or Output and Digital or Analog.

# Digital Input/Output

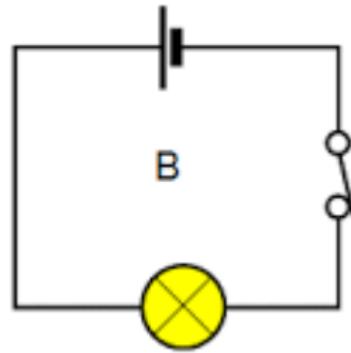
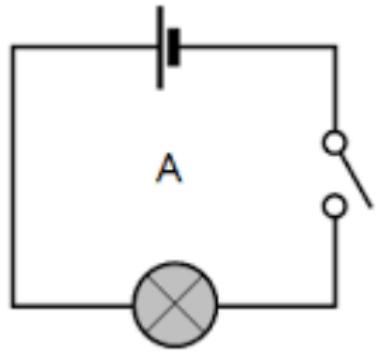
Digital electronics rely on binary logic to store, process, and transmit data or information. Binary Logic refers to one of two states – ON or OFF. This is commonly translated as a binary 1 or binary 0. A binary 1 is also referred to as a HIGH signal and a binary 0 is referred to as a LOW signal.



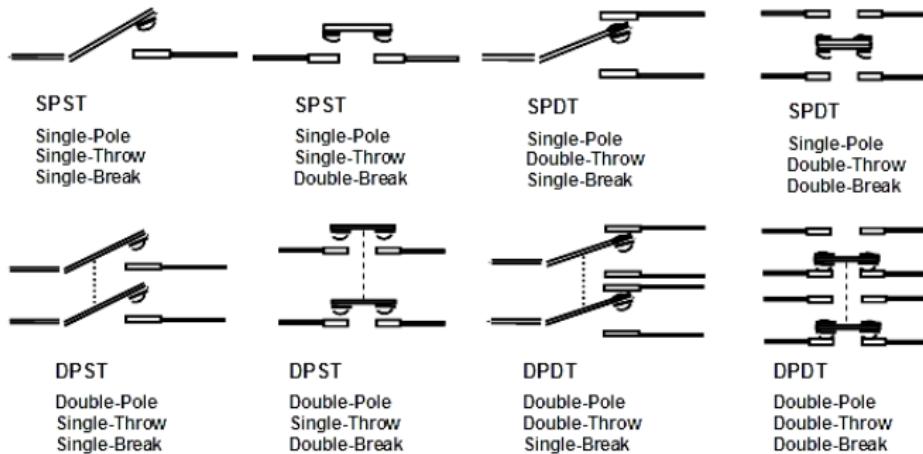
- `digitalWrite(pin,value);`
- `inputValue = digitalRead(pin);`

where, value equals HIGH or LOW.

# Switches



# Types of Switches



# Our First Button and Pull Down Resistors



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.

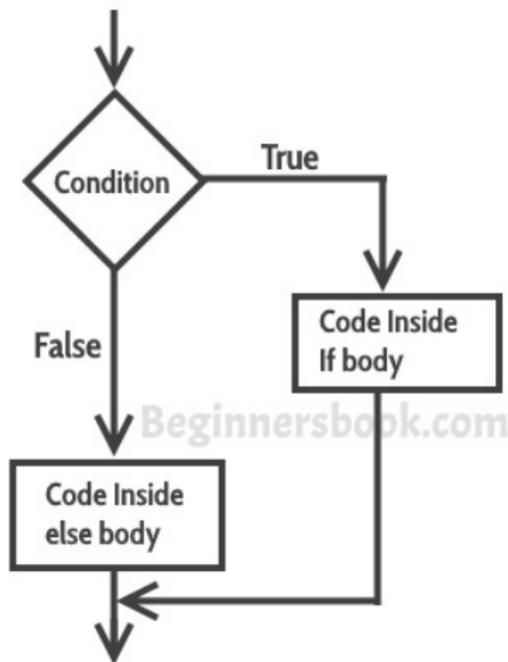
## ② L03\_02\_button\_pullup

- 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

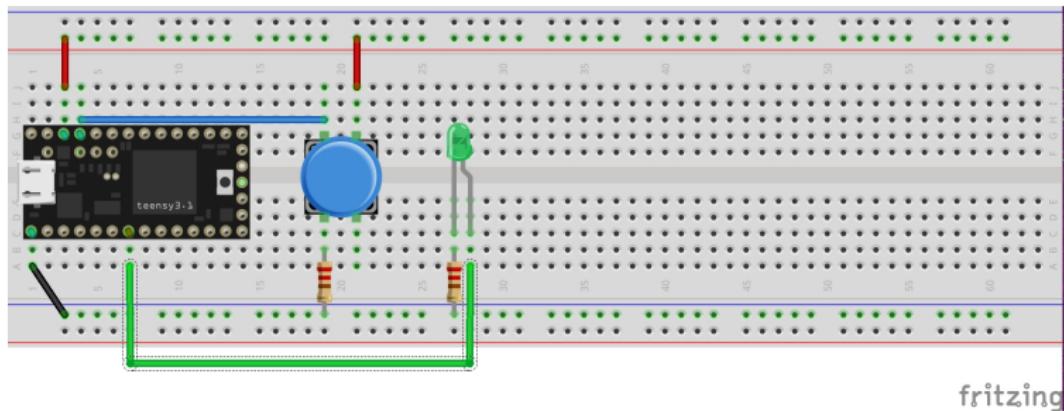


Beginnersbook.com

# 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

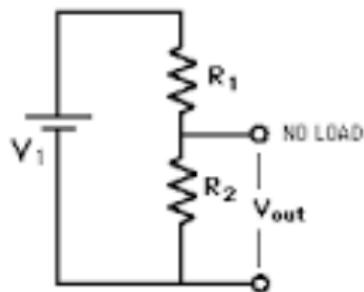
- Add an LED to Pin 5 and use the button to turn the LED on and off.
- Also, print button state to the screen

## 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

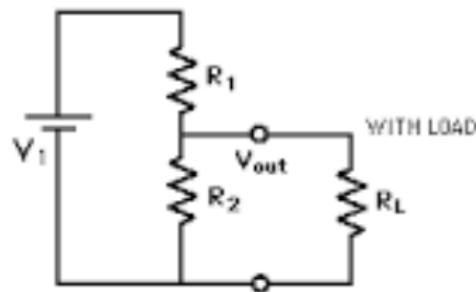
OPEN CIRCUIT BEHAVIOR



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

for open circuit

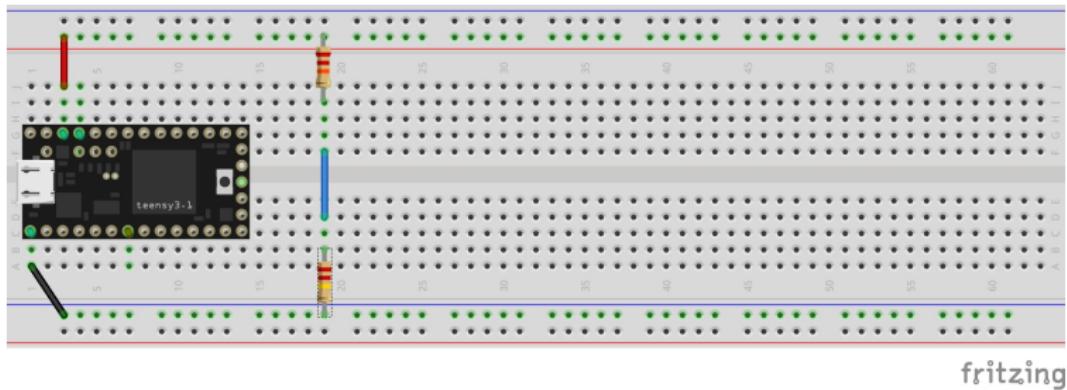
BEHAVIOR UNDER LOAD



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

for loaded circuit

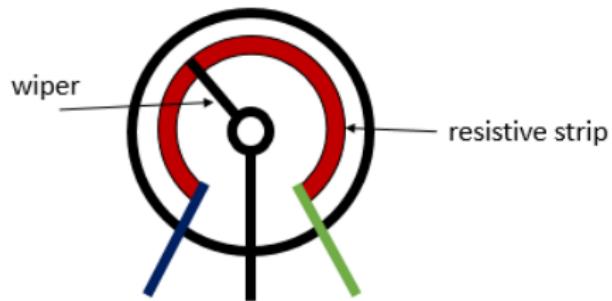
# Voltage Dividing



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

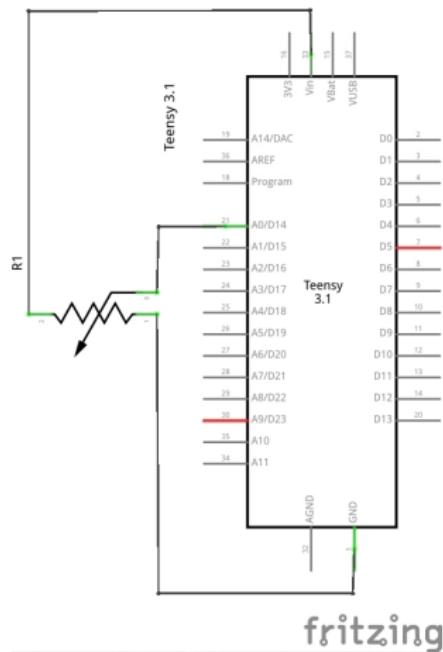
- Use various combinations of  $22k\Omega$ ,  $47k\Omega$ ,  $100k\Omega$ ,  $220k\Omega$  resistors.
- Calculate the Series resistance and the voltage between the two resistors in your Lab Notebook.
- Measure with your multimeter and compare.

# 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



- 1 Utilize `analogRead()` to measure analog input across potentiometer (voltage divider) using Pin 14.
- 2 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

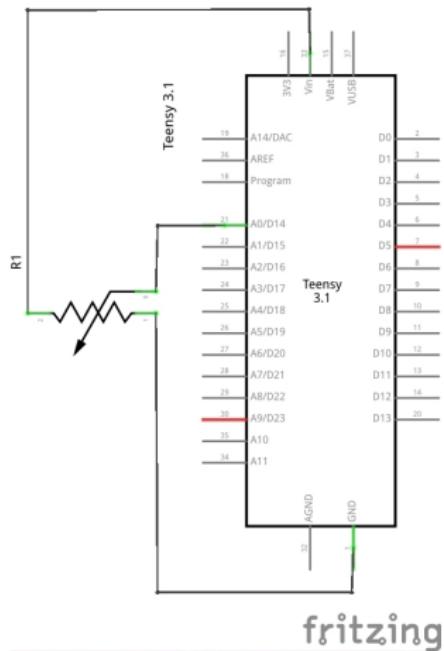
Return statement,  
datatype matches  
declaration.

Curly braces required.

# Basic Structure of Arduino Sketch Revisited

```
1 // include header files, declare variables
2 // declare global variables
3 void setup() {           // runs once
4   //initialize variables, begin processes
5 }
6
7 void loop() {           // loops indefinitely
8   doubleNum = twotimes(Num)
9 }
10
11 int twotimes(int Num) {
12   int answer;    // answer is a local variable
13   answer = 2 * Num;
14   return answer;
15 }
```

# Assignment L03\_06\_AnalogInput



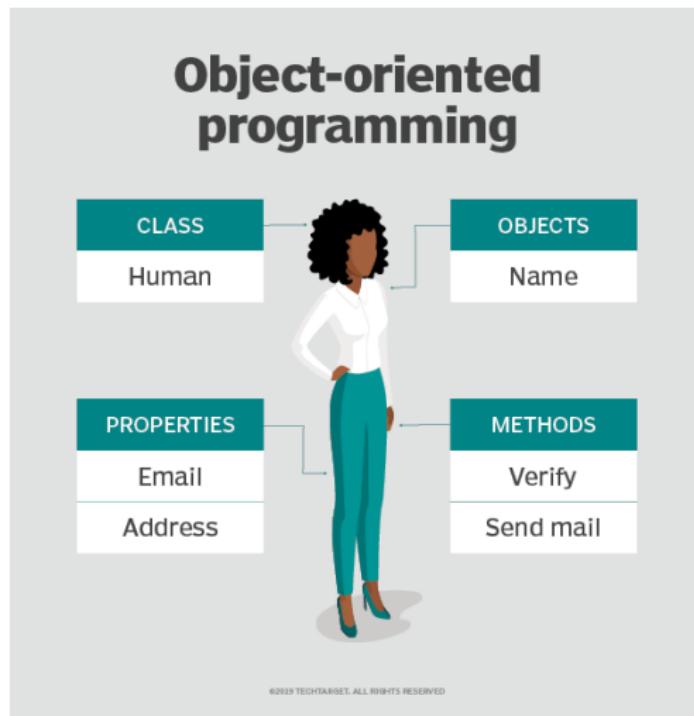
- ➊ Utilize `analogRead()` to measure analog input across potentiometer (voltage divider) using Pin 14.
- ➋ Determine the range of the `analogRead` across the entire range of the potentiometer.
- ➌ Write a function, `in2volts()`, that converts the analog input value to voltage.

# IoT Humor

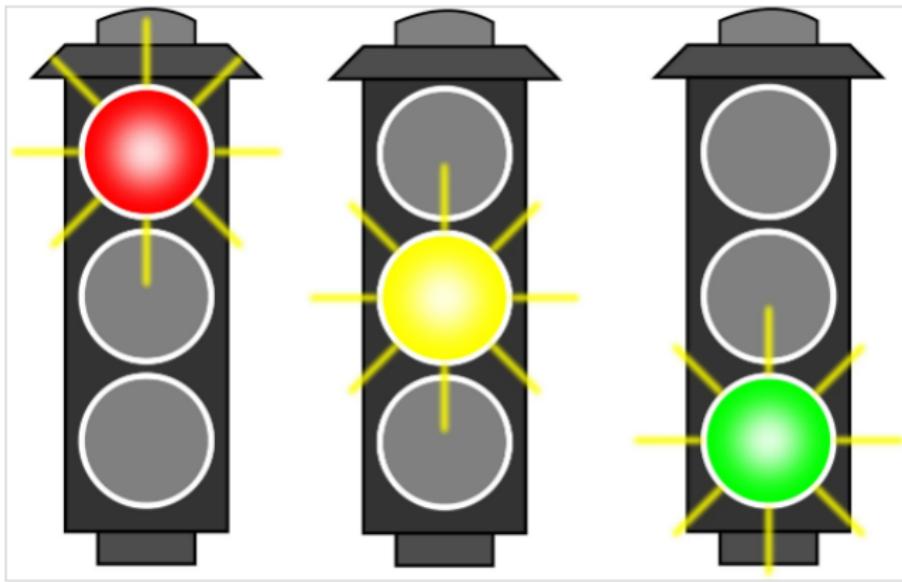


*"I remember when you could only lose a chess game to a supercomputer."*

# Objects

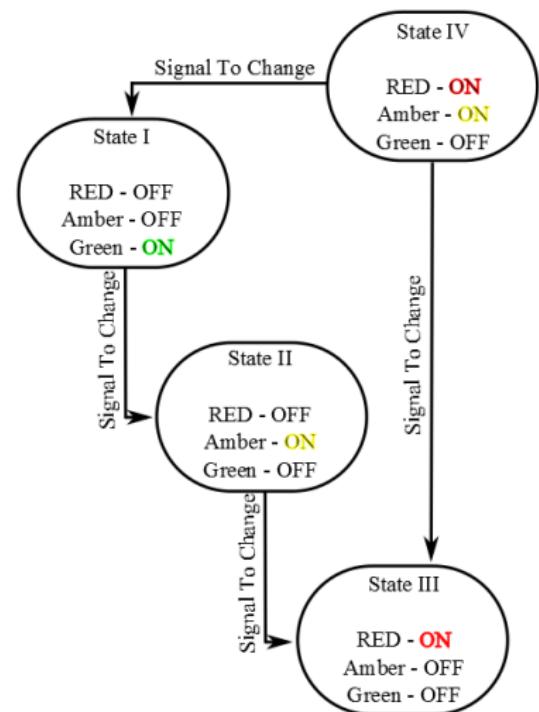
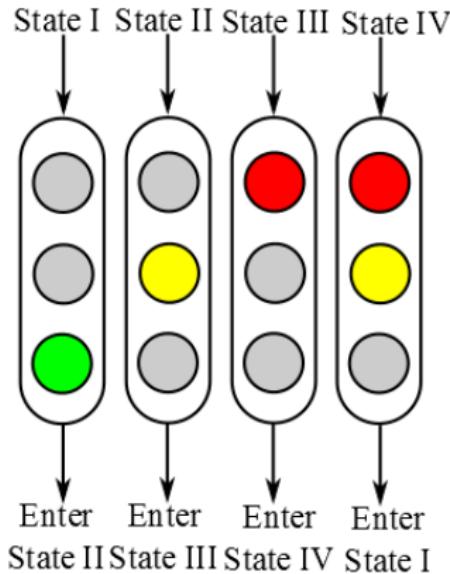


# Traffic Light



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
- 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 <library.h> // library files
3 int class_size; // declare global variables
4 Adafruit_BME280 bme; // name object in class
5
6 void setup() {
7     oled.init(); // initialize objects
8     bme.begin(0x76); // begin processes
9     arraySize = sizeof(array)/4; // set variables
10 }
11
12 void loop() {
13     // functionality of your code
14     // this loops indefinitely
15 }
```

# OneButton Declarations

```
1 #include "OneButton.h"
2 OneButton button1(pin, activeLOW, pullUP);
3
4 void setup() {
5     button1.attachClick(click1);
6     button1.attachDoubleClick(doubleclick1);
7     button1.attachLongPressStart(longPressStart1);
8     button1.attachLongPressStop(longPressStop1);
9     button1.attachDuringLongPress(longPress1);
10    button1.setClickTicks(250);
11    button1.setPressTicks(2000);
12 }
```

- 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 doubleClick() {
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 flash.

## ② L04\_02\_oneButtonLED

- Toggle LED on/off with buttonState.

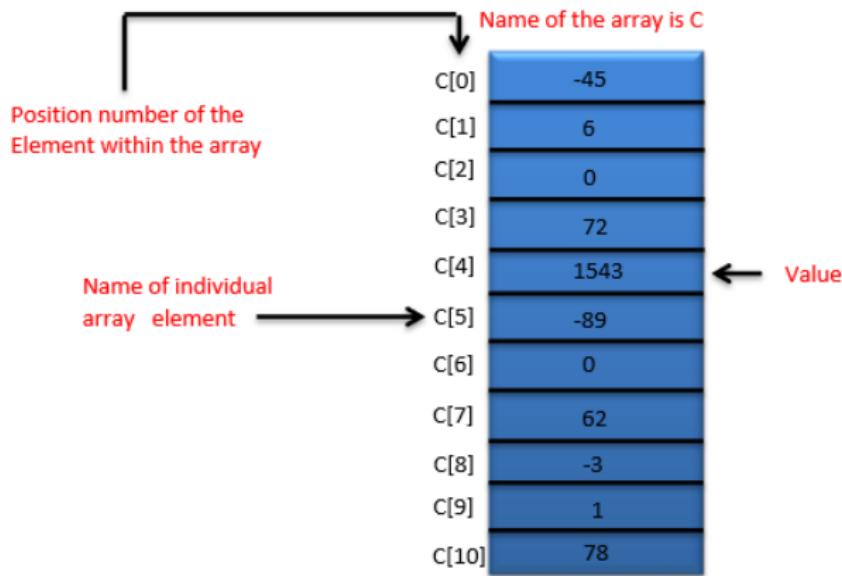
## ③ L04\_03\_oneButtonLEDblink

- When ON, toggle from solid to blinking with flash.

# 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



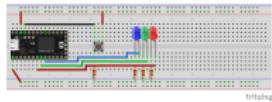
- Syntax: datatype var[ ] = {element 1, element 2, element 3};
- Example: int ledArray[ ] = {greenPin, yellowPin, redPin};

# 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, cycle to next LED
- longPressStart() - light up the three LEDs in sequence
- longPressStop() - light up the three LEDs in reverse order

# Assignment: Serial Read

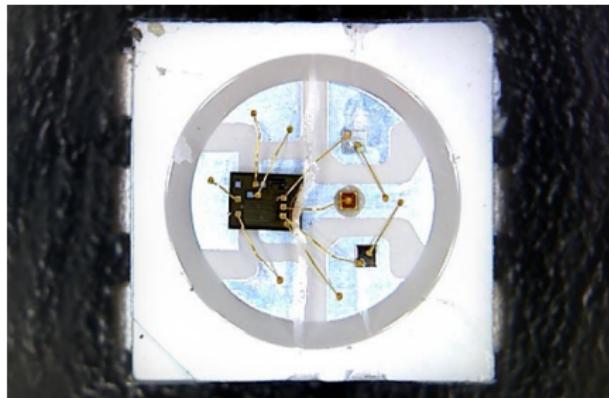


## Serial Read Example

### 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

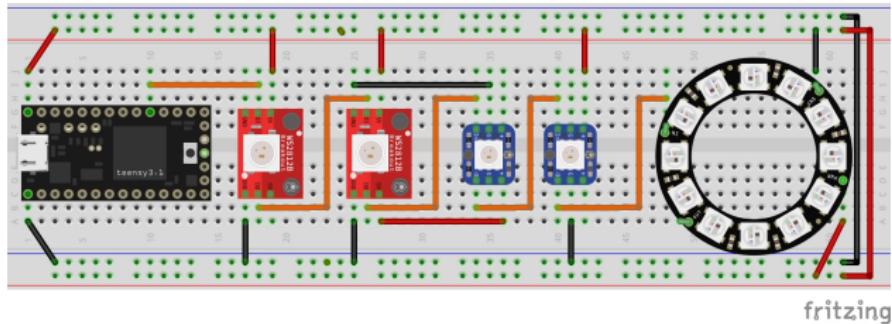
# NeoPixels



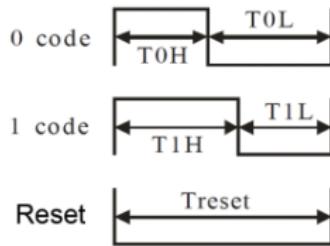
NeoPixels are:

- 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.

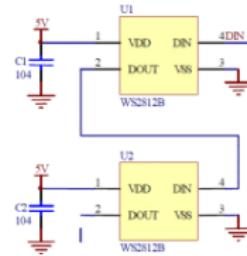
# NeoPixel Programming



WS2812 Protocol



LED-Chain



# NeoPixel Declaration

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

# 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

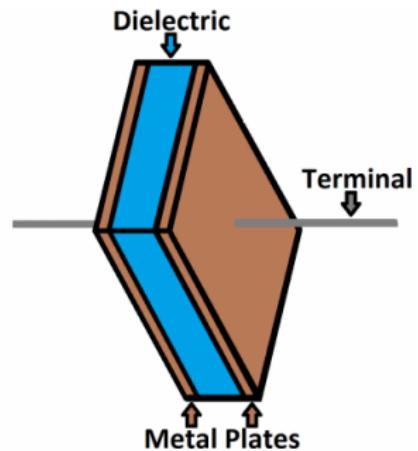
# IoT Humor



# Capacitors

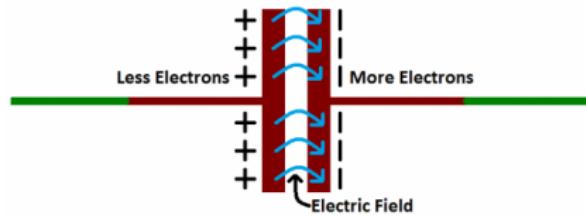
A capacitor is created out of two metal plates and an insulating material called a dielectric. The metal plates are placed very close to each other, in parallel, but the dielectric sits between them to make sure they don't touch.

- 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

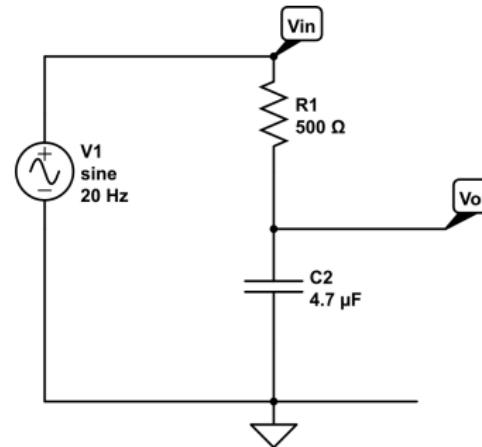
When current flows into a capacitor, the charges get "stuck" on the plates because it can not get past the insulating dielectric. Electrons build up on one of the plates, and it becomes overall negatively charged. The large amount of negative charges pushes away like charges on the other plate, making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



The stationary charges on these plates create an electric field, which influence 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.

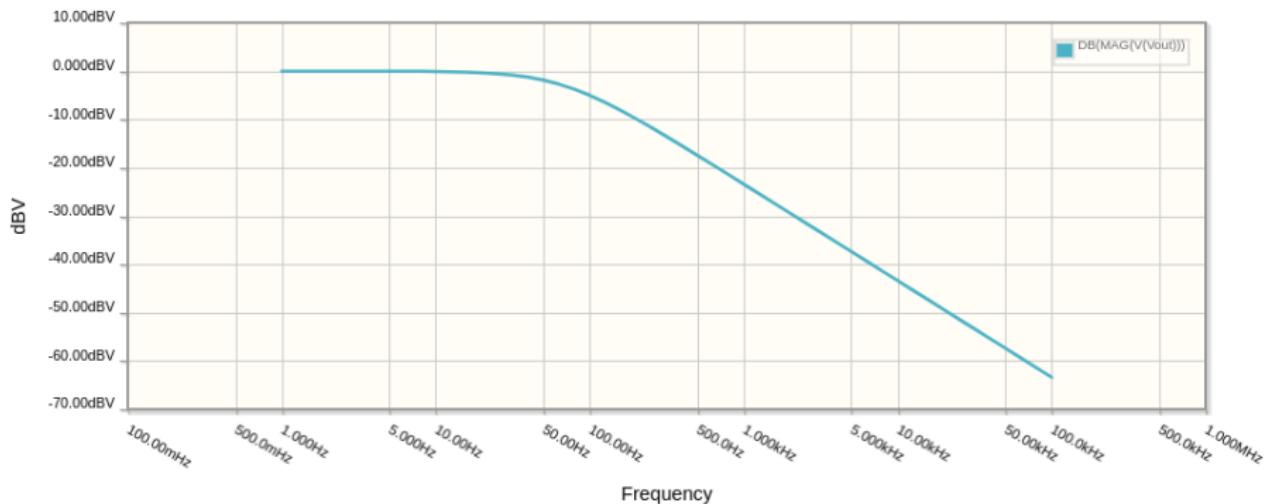
# Low Pass Filter - cutoff frequency $f_c$

- 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

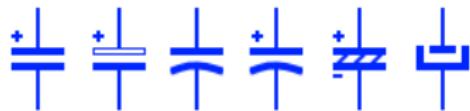


$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi(500)(4.7 \times 10^{-6})} = 67.5678 \text{ Hz}$$

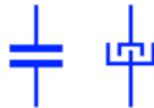
# Capacitors - does it matter how they are placed

- 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.

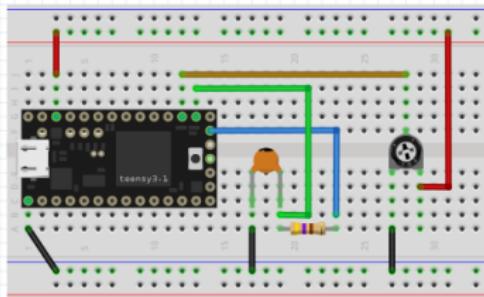
Polarized Electrolytic Capacitor



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.

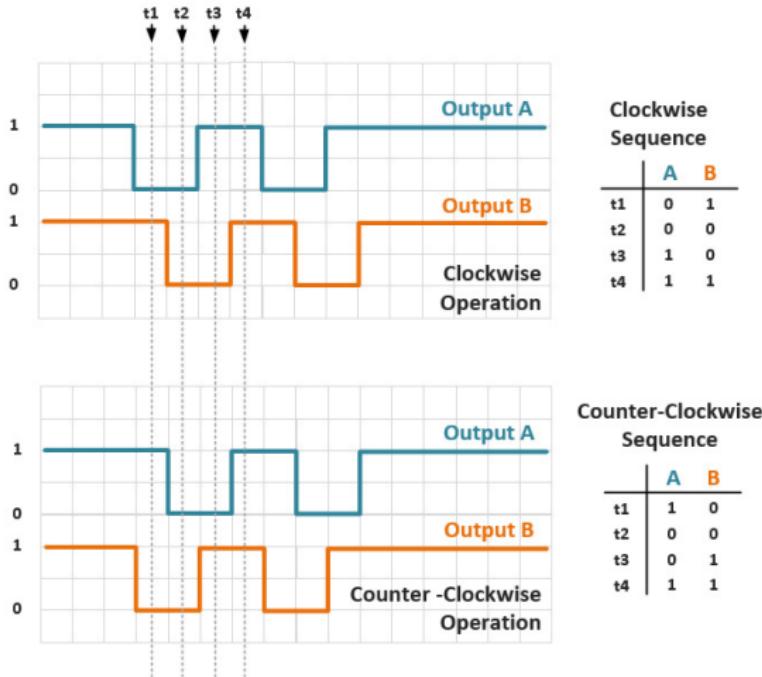
Using Serial Plotter:

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

# Encoders



# Encoders



# 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 }
```

# Assignment: Encoders

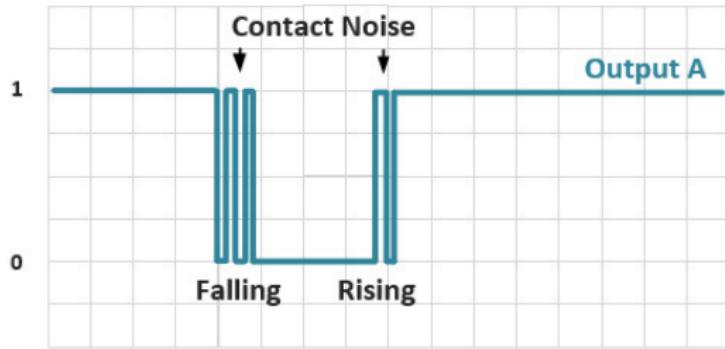


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

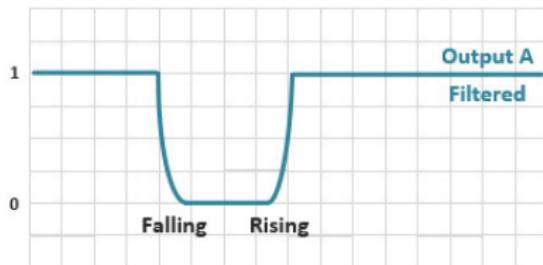
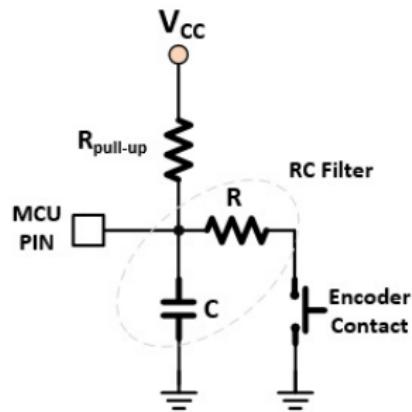
- ➊ L06\_01\_encoder
  - Write encoder position to the screen
- ➋ L06\_02\_encoderScaled
  - The encoder has 96 positions
  - Manually map the encoder to 12 pos ( $0-7 = 0, 8-15 = 2$ , etc.)
  - Next, use the map() function:
- ➌ L06\_03\_encoder\_NeoPixel
  - Use the encoder to light up the pixel ring
- ➍ L06\_03\_encoder\_switch
  - Connect your microcontroller to the encoder switch and LEDs

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

# Encoder Jitter



# Encoder - Low Pass Filter



# Assignment: Encoders

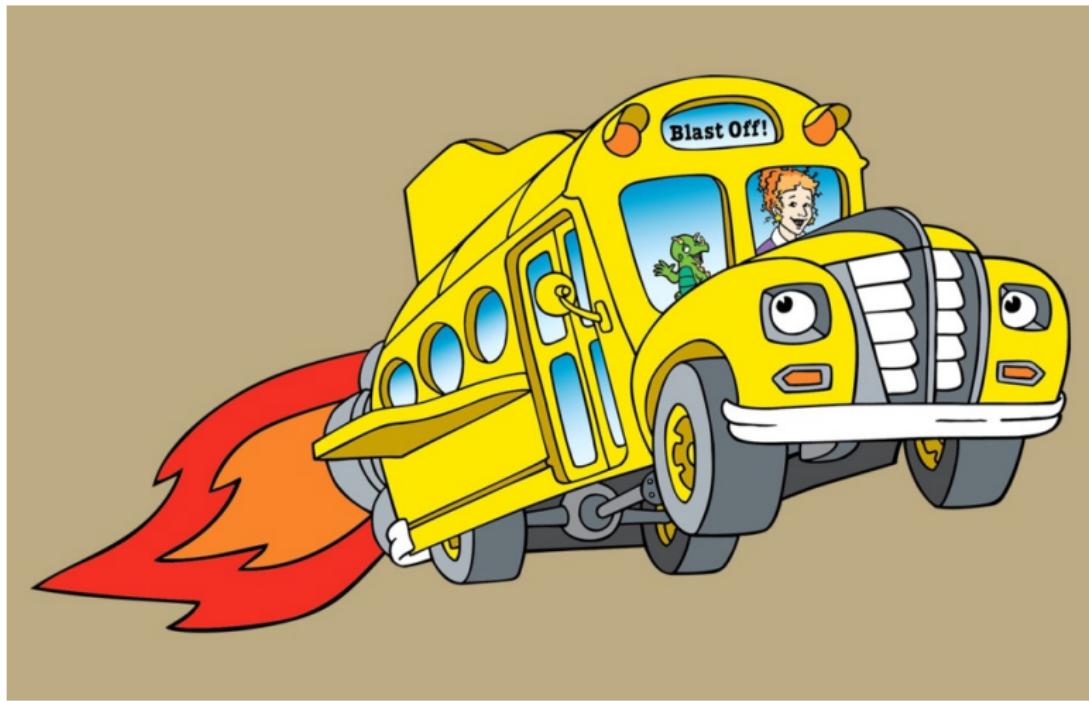


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

## 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.

# Buses and Interfaces

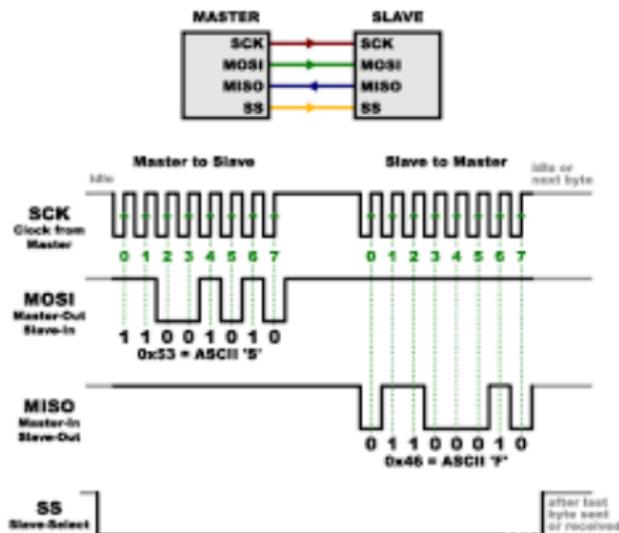


# UART



Universal Asynchronous Receiver/Transmitter

# Serial Peripheral Interface



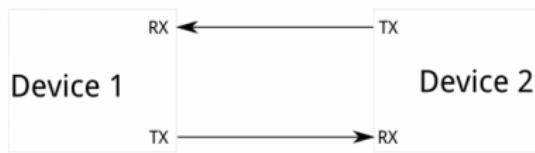
- Master Out, Slave In (MOSI) connects to Data In
- Master In, Slave Out (MISO) connects to Data Out

# Serial Peripheral Interface

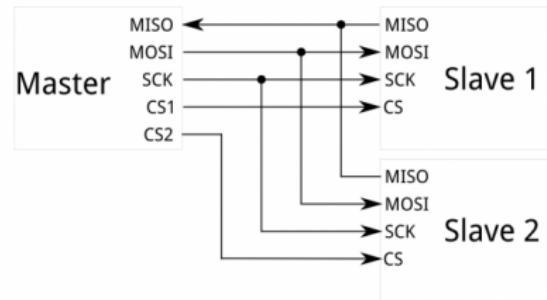


# Serial +/−

## UART



## SPI



# Assignment: Write to $\mu$ SD Card



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

## ① L07\_01\_dataLogger

- String datatype
- SPI and SD libraries
- File object
- SD.begin()
- dataFile.print()
- Simulate 3 data streams and write to SD card

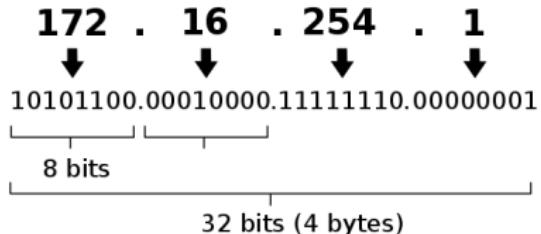
# The Internet



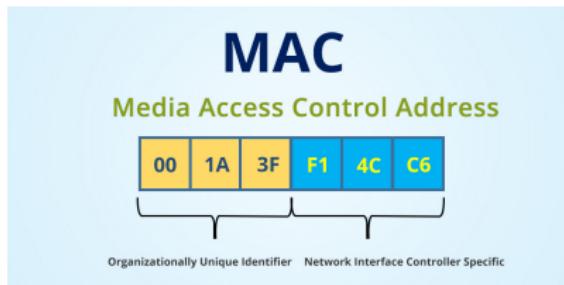
# IP Addresses

- 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

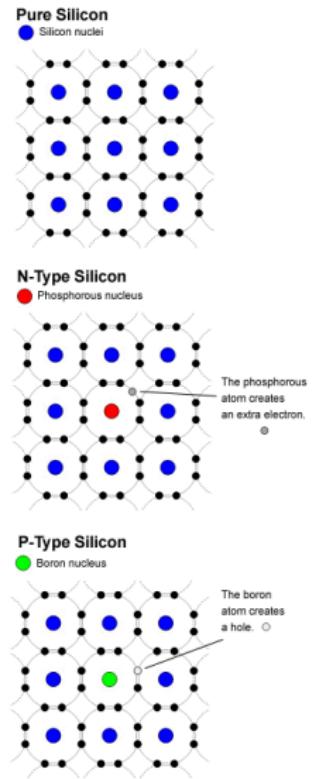


- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

- ➊ Create a mac.h header file
- ➋ L08\_00\_EthernetTest
  - Modify to use your mac.h file
- ➌ L08\_01\_Wemo
  - Using the wemo.h library and WemoH\_Example as a template, create code that turns on and off multiple Wemo Smart Outlets in the classroom.
- ➍ L08\_02\_Wemo\_Object
  - Modify the wemo.h library from being a function to being Class and Methods.
  - Modify your wemo code the create and use a wemo object.

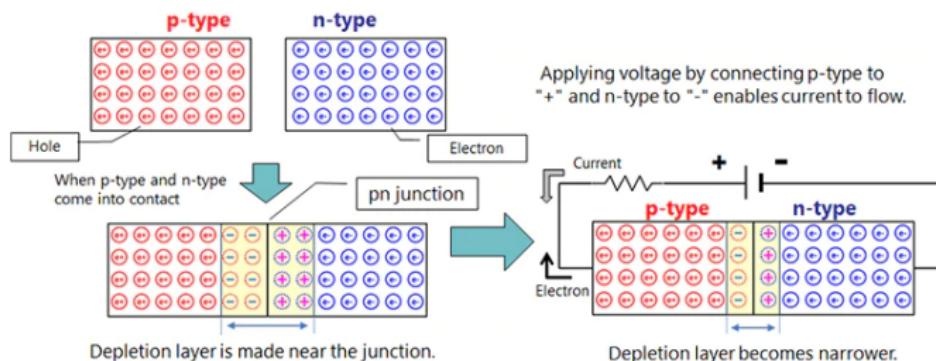
# Semiconductor

- 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.



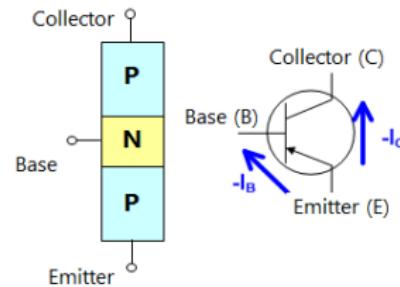
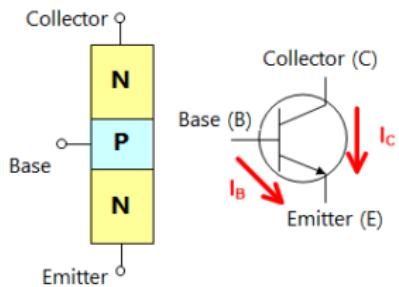
# pn junction diode

- 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.



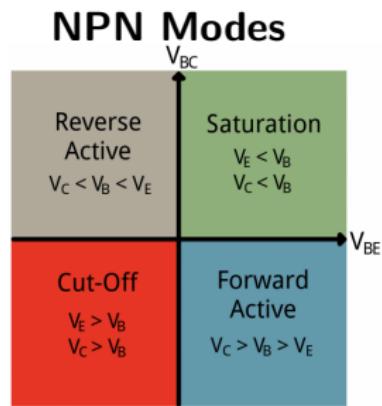
# Bipolar Junction Transistor

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.



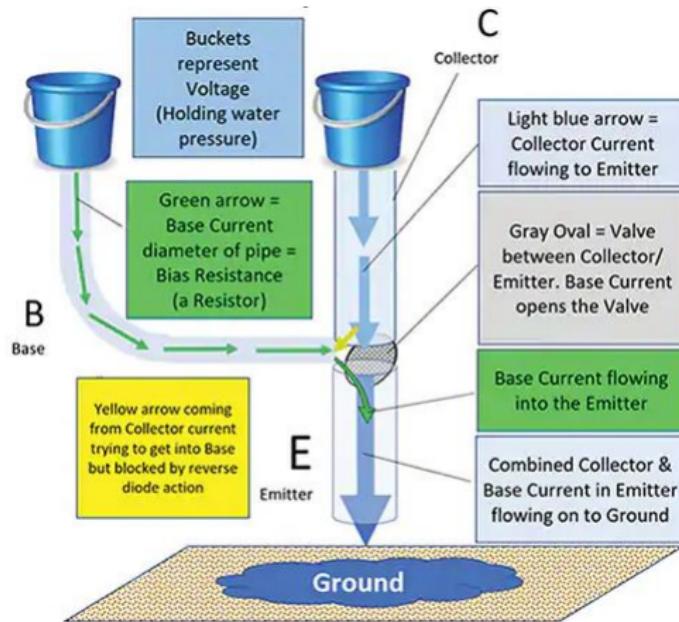
# Bipolar Junction Transistor - Modes of Operation

- **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).



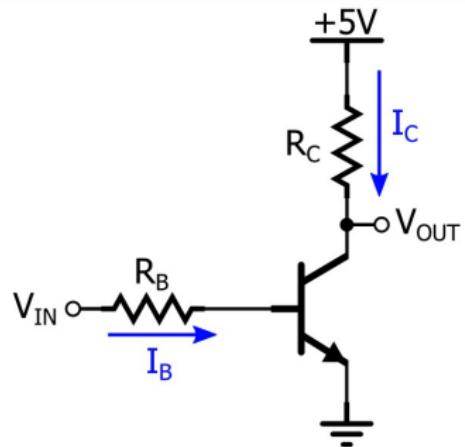
Voltage relations	NPN Mode	PNP Mode
$V_E < V_B < V_C$	Active	Reverse
$V_E < V_B > V_C$	Saturation	Cutoff
$V_E > V_B < V_C$	Cutoff	Saturation
$V_E > V_B > V_C$	Reverse	Active

# 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.

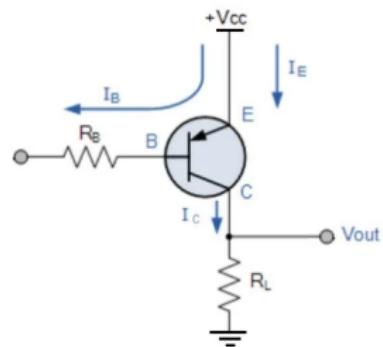


**NOTE:**  $V_{OUT}$  is an amplified but inverted signal of  $V_{IN}$ . This simple circuit will step-up an 0 - 3.3V output from the microcontroller to 0 - 5.0V (inverted). The low impedance of the output will also provide sufficient current to drive a higher current device (e.g., a relay).

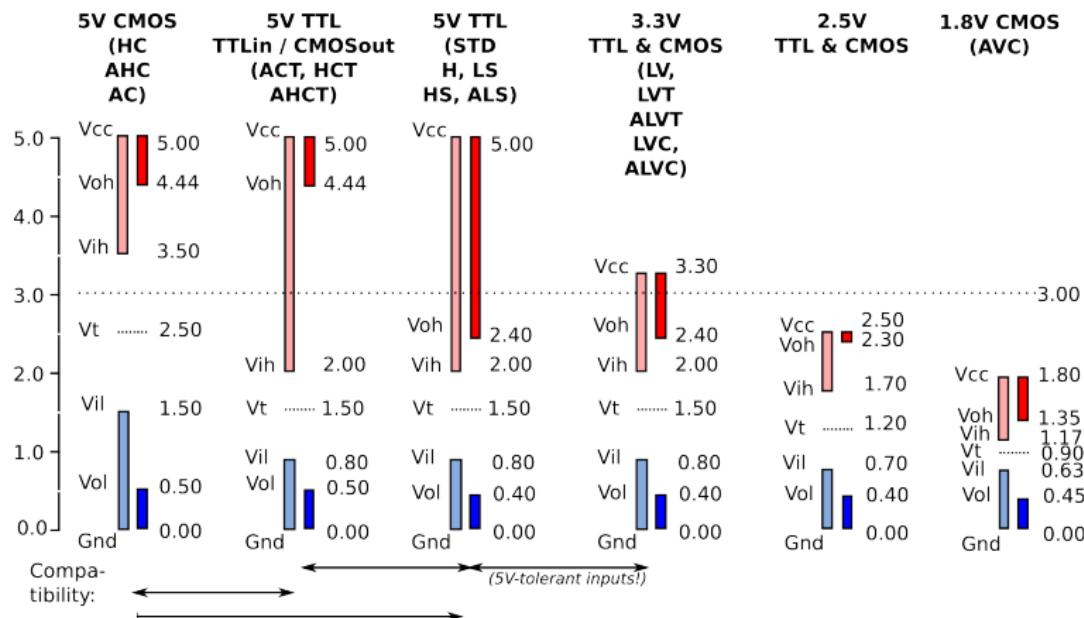
# PNP Transistor

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

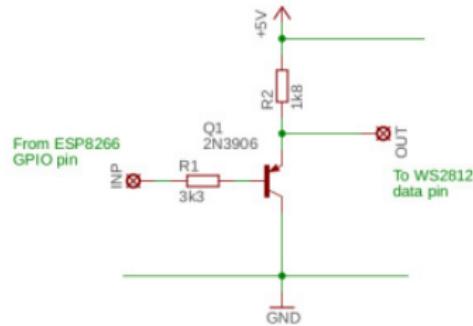


Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).

Or, what is wrong with the NeoPixels.

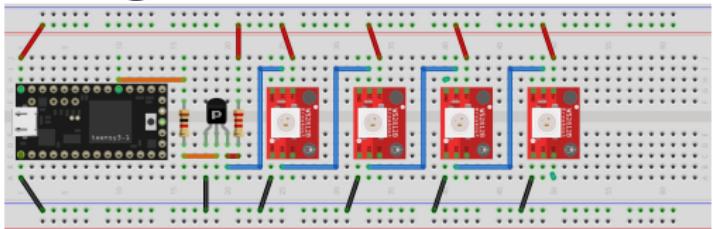
# Emitter Follower

- NeoPixels are designed around 5V CMOS transistors
  - $V_{IH} > 3.5V$
  - 3.3V Microcontroller
  - $V_{OH} = 3.3V$
- An Emitter Followers (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.



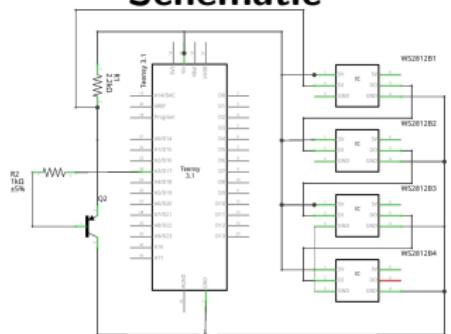
# Emitter Follower Layout

## Fritzing



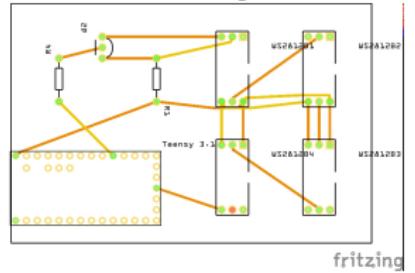
fritzing

## Schematic



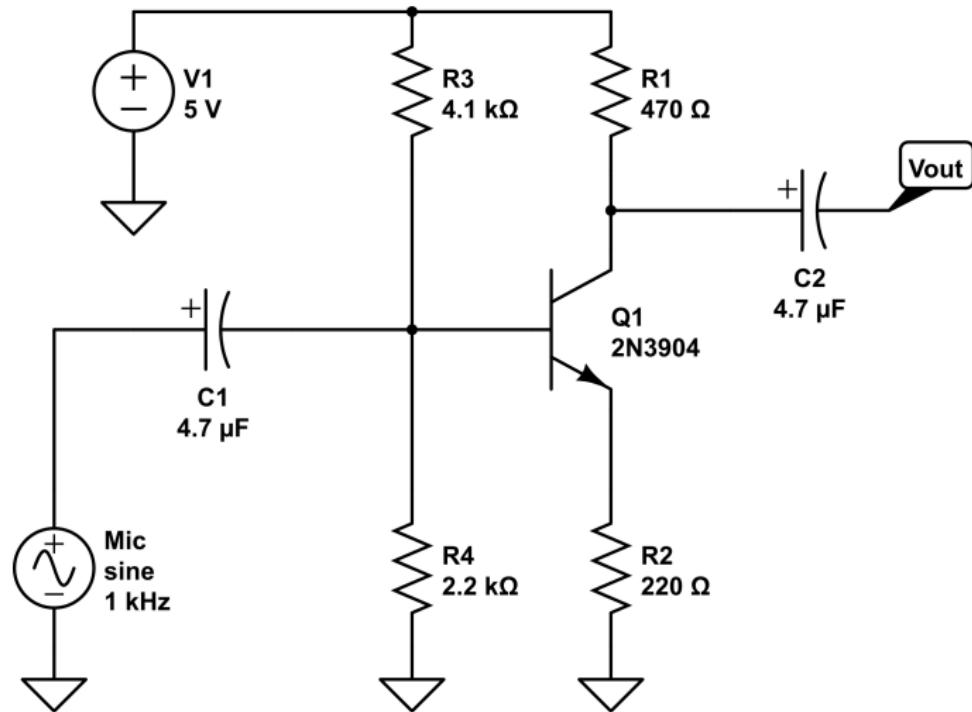
fritzing

## PCB Layout



fritzing

# NPN Pre-Amplifier Circuit

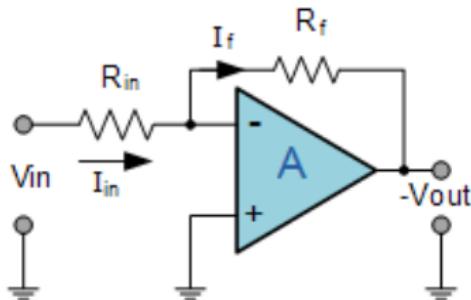


# Two Stage Amplifier



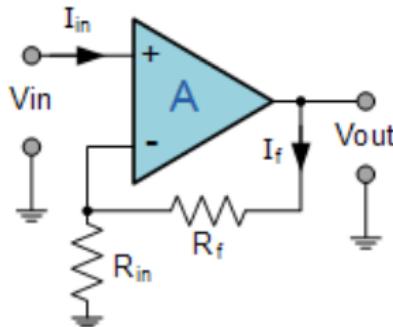
# Op Amp Lesson - Under Construction

## Inverting Op-amp



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

## Non-inverting Op-amp



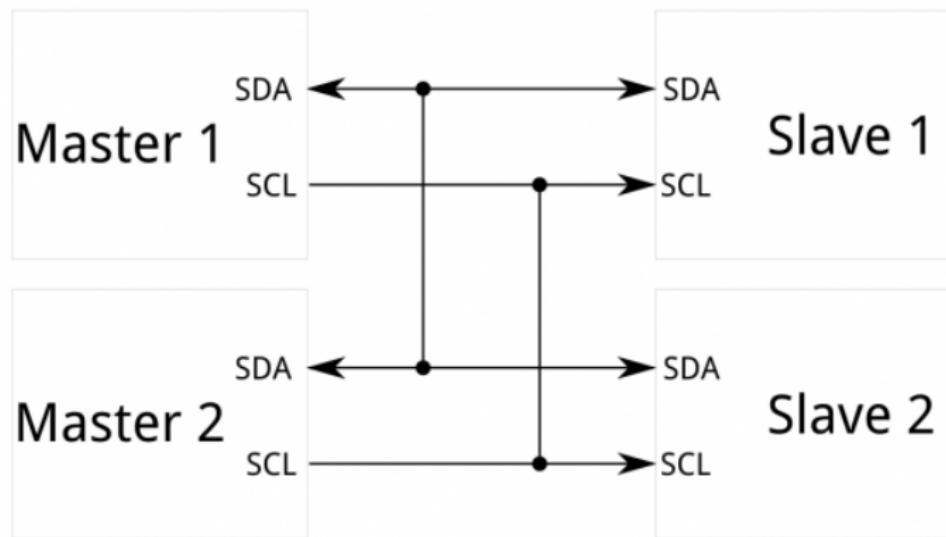
$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

# Assignment: Amplifiers



- ① L09\_01\_NeoPixel
- ② L09\_02\_NPNAMP
  - Make NPN Amp
- ③ L09\_03\_OpAmp
  - Op Amp

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

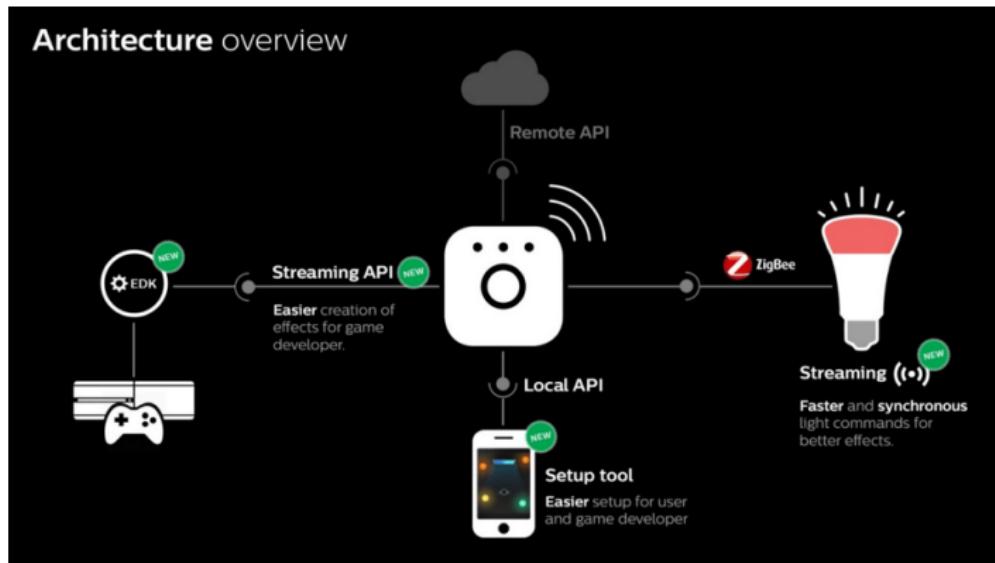


# Assignment: I<sup>2</sup>



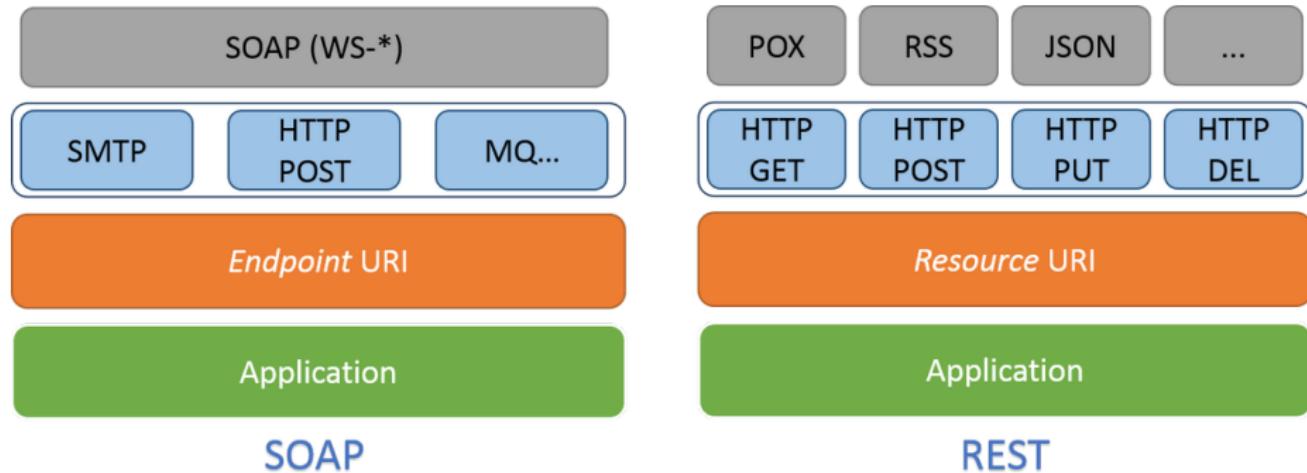
- ① L10\_01\_OLEDWrite
  - TASKS
- ② L10\_02\_BME280
  - TASKS
- ③ L10\_03\_BME280\_OLED
  - TASKS
- ④ L10\_04\_BME280\_SDMicro
  - TASKS

# 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



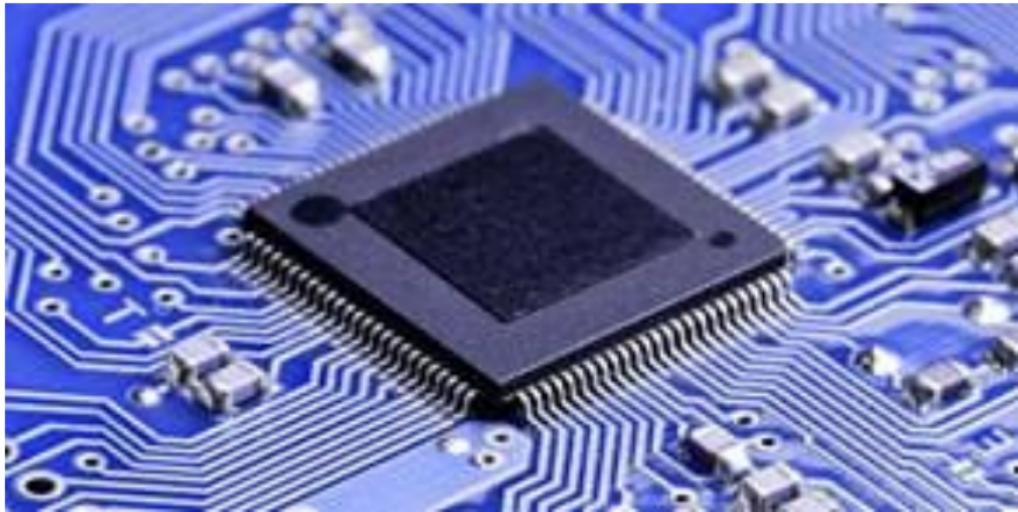
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

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

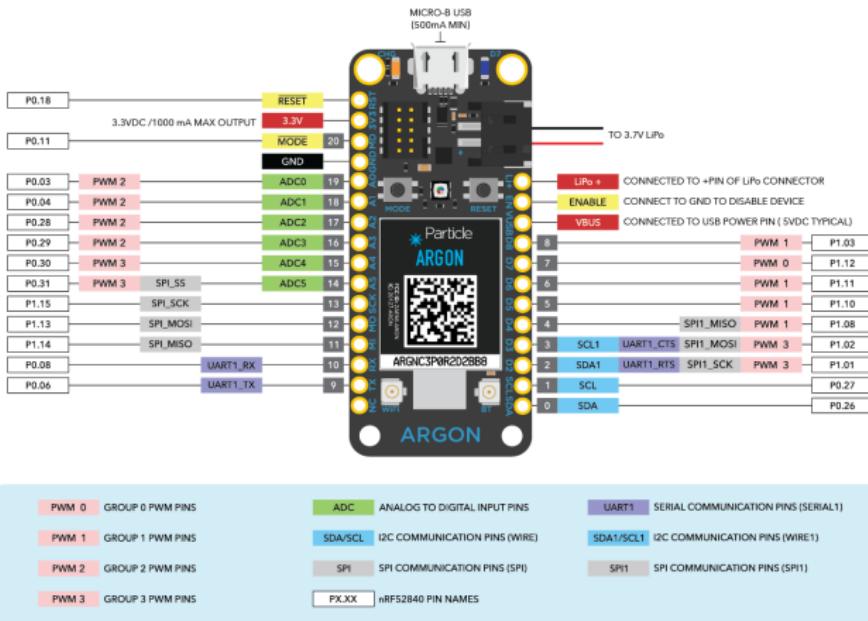
- button that turns on and off the Hue light at your pod
- uses the encoder to change the brightness of the Hue bulb
- has a method of cycling the Hue light through the colors of the rainbow.



# Our Second Microcontroller



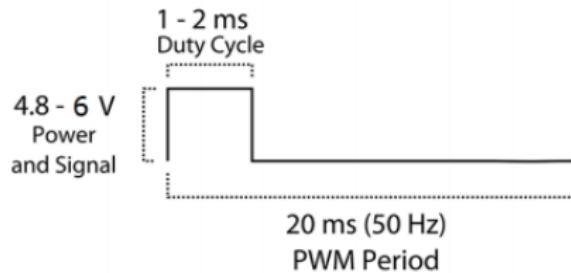
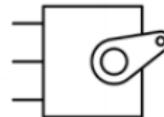
# Particle Argon Pin Layout



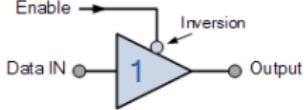
v1.0

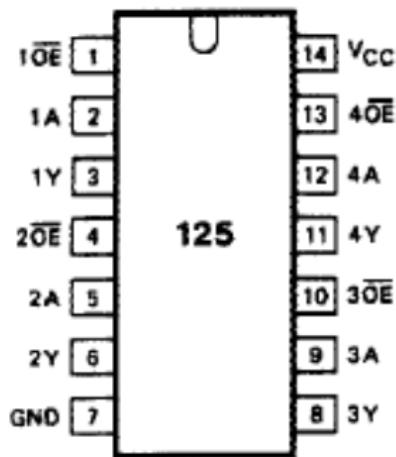
# Servo Motors

PWM=Orange (⊤⊤)  
Vcc = Red (+)  
Ground=Brown (-)



# TriState Buffer

Symbol	Truth Table		
	Enable	IN	OUT
	0	0	0
	0	1	1
	1	0	Hi-Z
	1	1	Hi-Z
Read as Output = Input if Enable is NOT equal to "1"			



# Powershell Commands Simplified

```
1 mkdir <directory name> //make a directory
2 rmdir <directory name> //remove directory
3 ls //list contents of current directory
4 pwd //print path of current directory
5 cd <directory name> // change directory
6 cd .. //change directory to the directory above
7 new-item <file name> //create a file
8 cp <file name> <new file name> //copy file
9 mv <file name> <directory> //move file to new dir
10 rm <file name> //delete file
11 cat <file name> //display contents of file
```