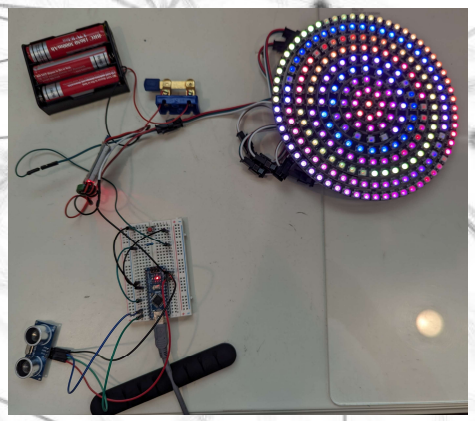
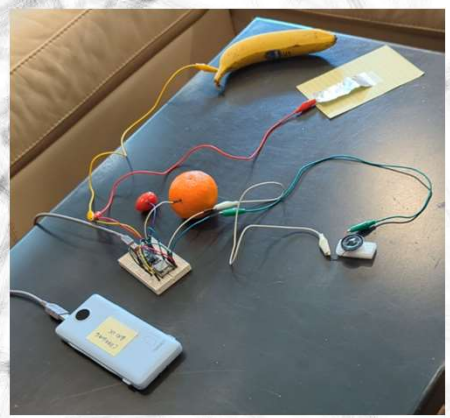
Take Our Daughters And Sons to Work Day

https://github.com/Mark-MDO47/TODAS_DaughtersAndSons

June 2024





The Arduino Experience!

TODAS – Electricity, Arduinos, Fun

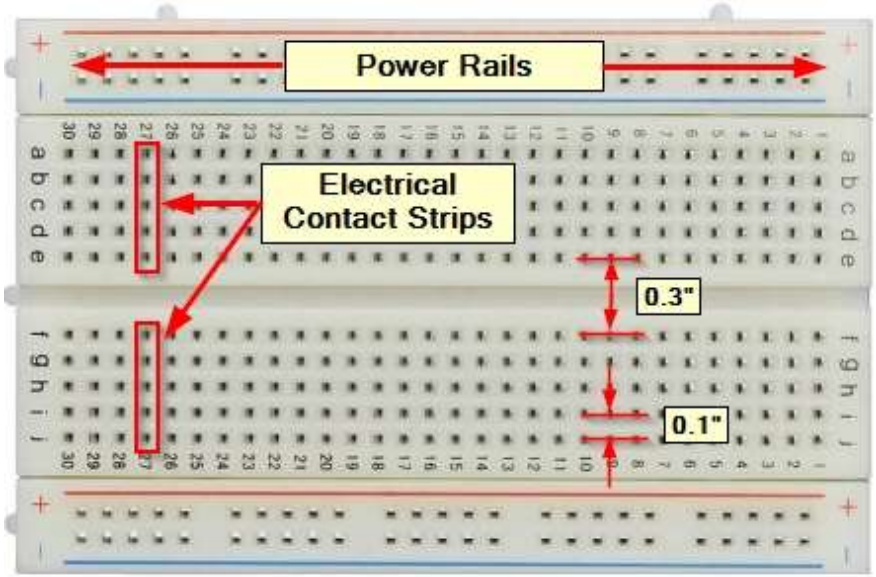
- Introduction
- Water Flow - a way to think about Electricity
- Arduino (computer on a chip), buttons, and LEDs
- Sonar Control and big LED ring
- Bananas and Sounds

- Resources

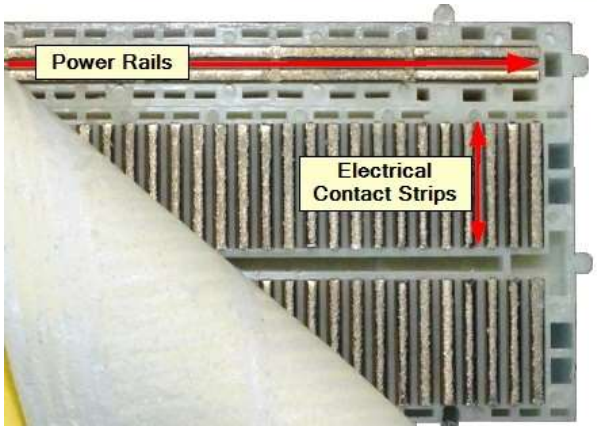
Introduction

- We have one hour – it is just an introduction
  - The idea is to do a quick overview
  - Questions OK but this is not a deep course- we'll keep moving
  - Lots of resources on the web for further investigation
- 4 projects, 3 using a small computer called Arduino
  - Do some wiring
  - Customize some software
  - Have some fun

01 – Breadboards – how do they work?

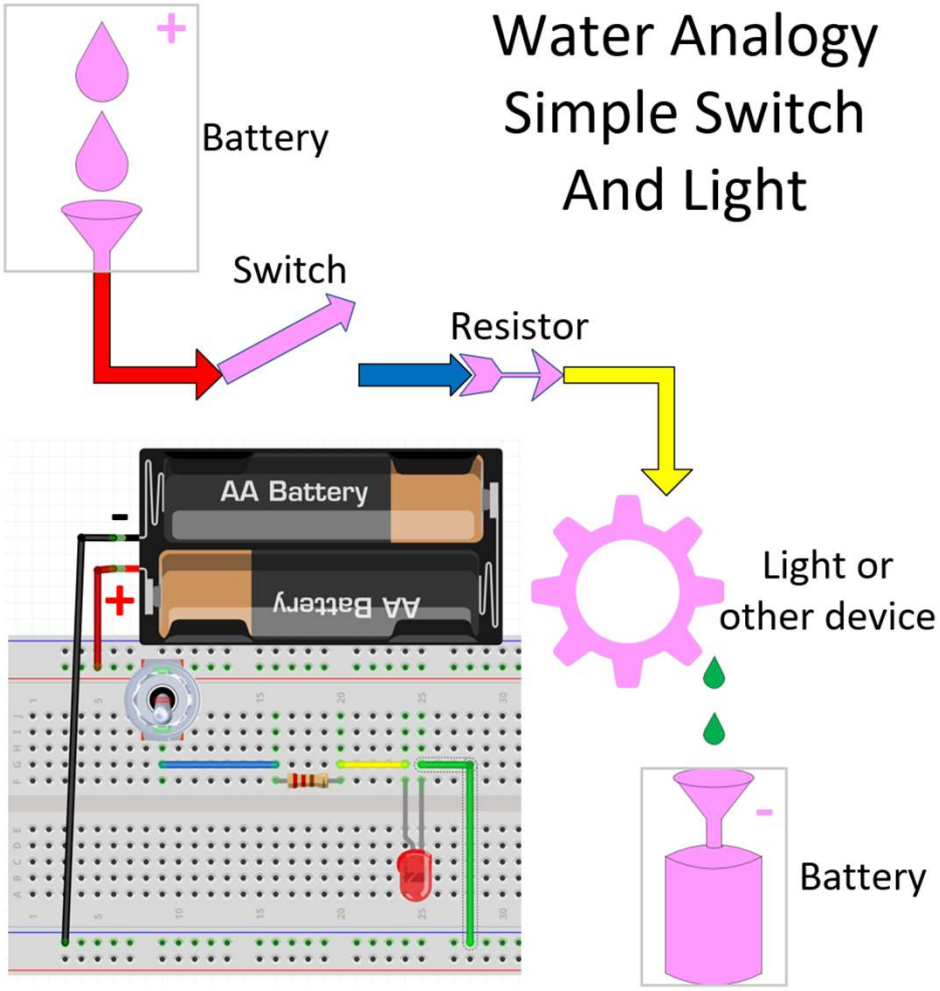


- Sides have power rails – length
- Center has contact strips – width



- Images from protosupplies.com

01 – Water Analogy – Simple Circuit



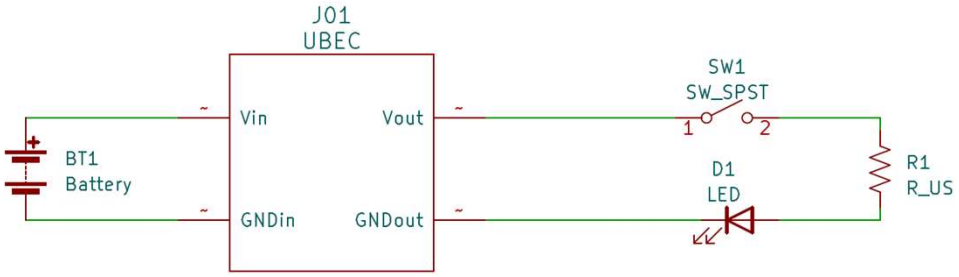
Electrons in wires actually flow from negative to positive, but many electrical symbols are drawn as if the current flow is from positive to negative so let's get used to it!

Positive to negative flow was good enough for Benjamin Franklin and for me!

"Pictorial" circuit is "Fritzing Diagram"

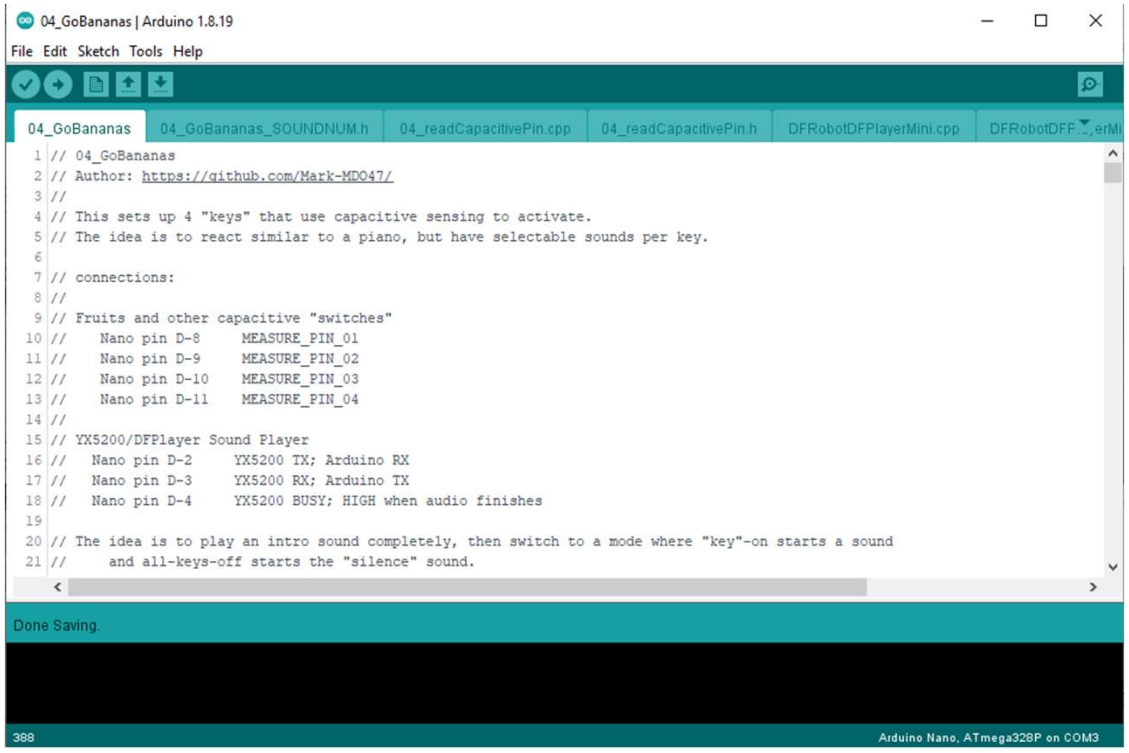
- https://fritzing.org/ - 8 Euros

01 Simple Circuit Schematic



- The Universal Battery Eliminator Circuit (photo of type I used) produces 5 volts if the input is from 5.5 Volts to 26 Volts.
- The resistor prevents sending too much current through the LED; that would burn it up.
- Schematic diagram is from KiCad
  - https://www.kicad.org/ - KiCad is free!

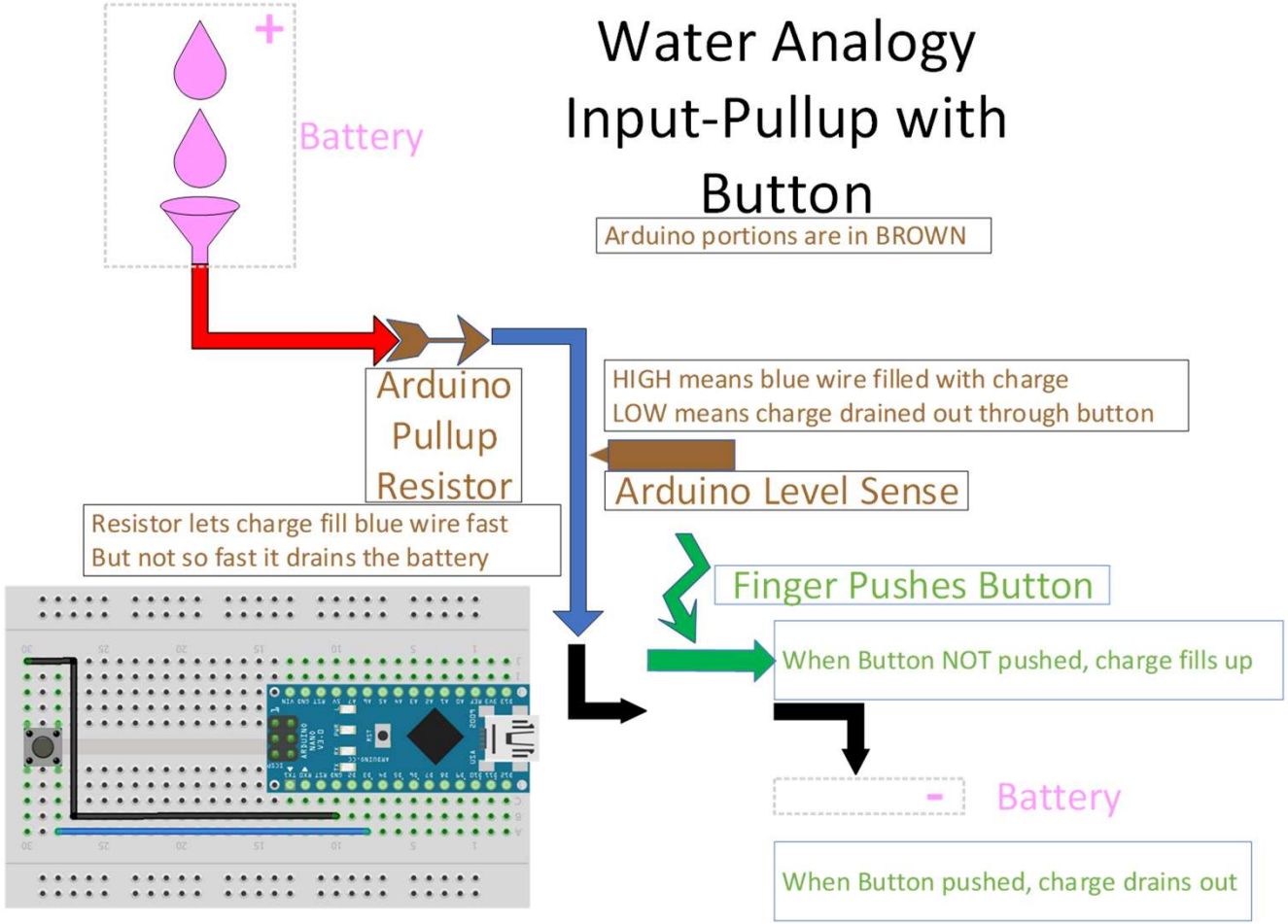
02 – Arduino IDE



- IDE means Integrated Development Environment
- This is how we edit, compile and load our software

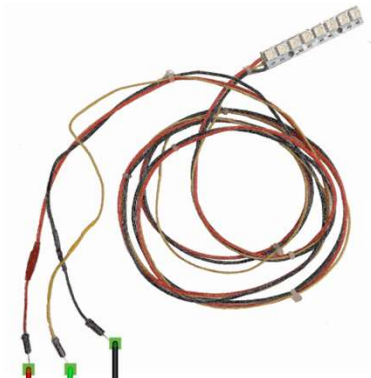
- https://www.arduino.cc/en/software - Arduino IDE is free!

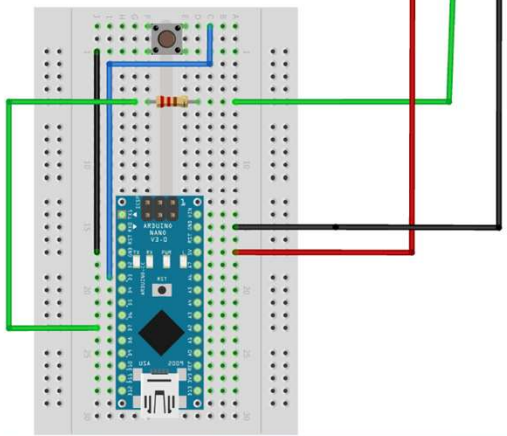
02 – Water Analogy Input - Pullup & Button



Electrons in wires actually flow from negative to positive, but often easier to understand by pretending flow from positive to negative

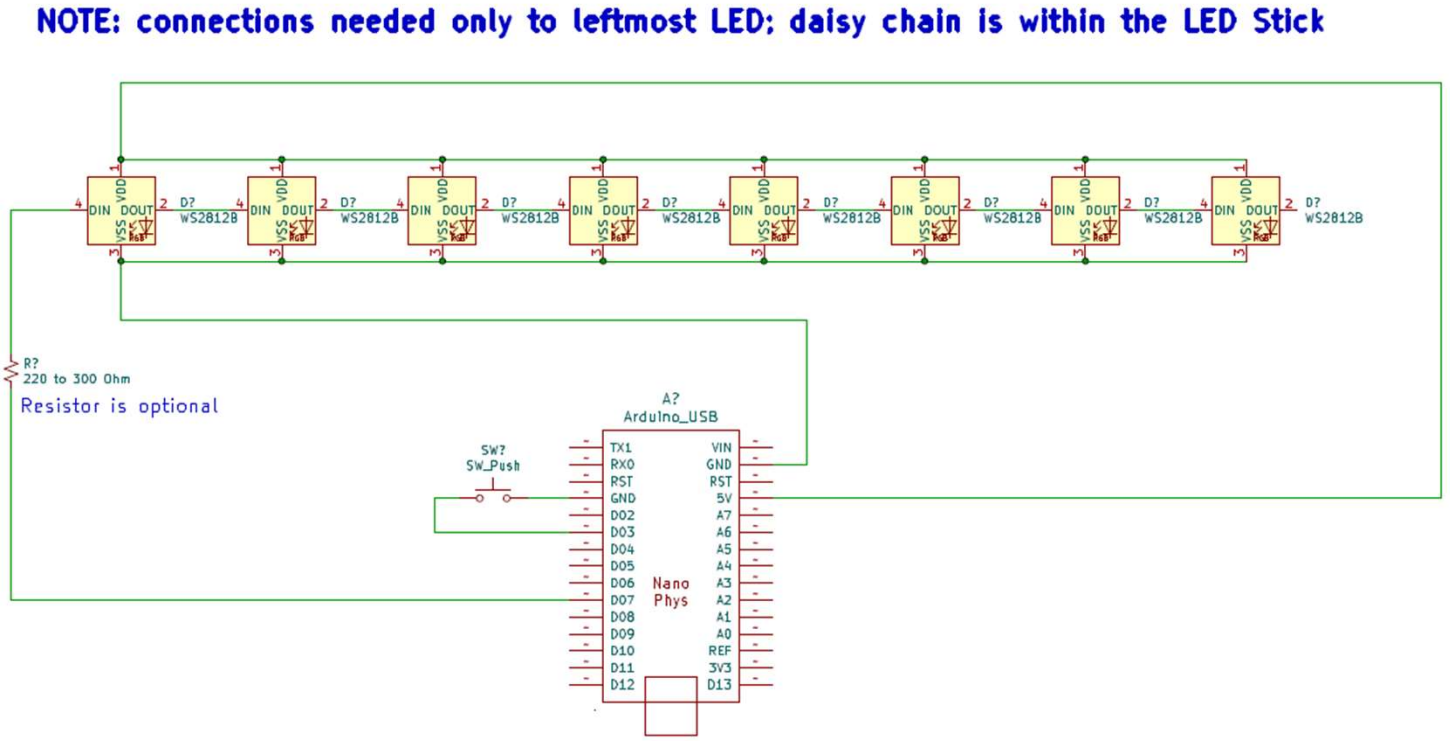
02 – Arduino Moving Color Lights





- "Fritzing diagram" of color lights in motion!
  - LED is Light Emitting Diode
- Configure software for color & speed choices
- Press button to go fast, release to go slow
- How does it work? SOFTWARE!
  - Arduino pins can be input or output – you tell it which
  - Read the pin for the button
    - If not pressed – use slow speed
    - If pressed – use fast speed
- TLDR – WS2812B individually addressable color LEDs using FastLED library

02 - Arduino Moving Color Lights Schematic



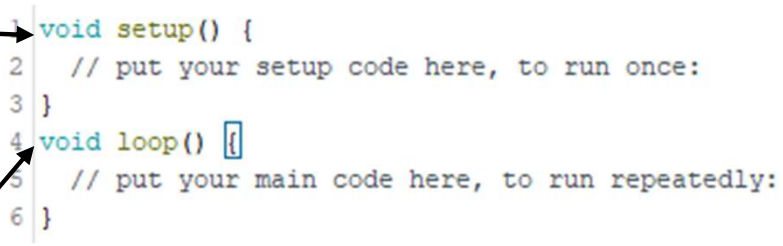
- The LED stick has eight small chips that control 3 LEDs each: Red, Green, and Blue.
- 3 LEDs are close together so they look like one color

02 – Arduino Inputs

- When button is pushed, Arduino input is LOW
- When button is not pushed, Arduino input is HIGH
- We want to do LED motion fast when the button is pushed
  - Assign name to the pin we use
  - Make the pin an input
  - If the state is HIGH (button not pushed) go fast else go slow

02 - Arduino Code Structure – Empty Sketch

- Empty “sketch” has
  - setup() – called once at start
    - Initialize hardware
    - Initialize software
  - loop() – called repeatedly
    - Process events
    - Generate outputs



02 – Arduino Button Input Software

```
33 #define BUTTON_PIN 3 // press to press for FAST_INTERVAL timing; release for SLOW_INTERVAL
```

This defines a name for pin D3 for the button

```
142   pinMode(BUTTON_PIN, INPUT_PULLUP); // digital INPUT_PULLUP means voltage HIGH unless grounded
```

Inside “setup()“ - pinMode sets that pin to be an input with “pullup”

It is HIGH unless connected to ground by pushing button

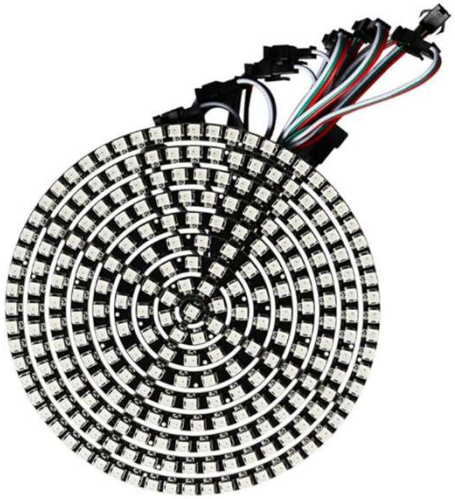
This is near the start of “loop()”

- If button (HIGH) NOT pushed we set “interval” to SLOW
- Else button (LOW) IS pushed; we set “interval” to FAST

We use “interval” to decide when to step pattern forward

```
165   if (HIGH == digitalRead(BUTTON_PIN)) {
166   interval = SLOW_INTERVAL;
167   } else {
168   interval = FAST_INTERVAL;
169   }
170
171   if (currentMillis - previousMillis >= interval) {
```

03 – Go Big with Ultrasonic Sonar Control

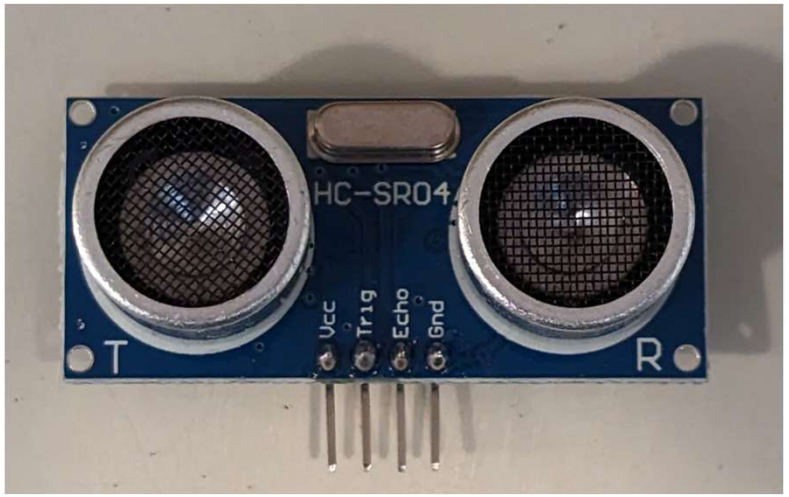


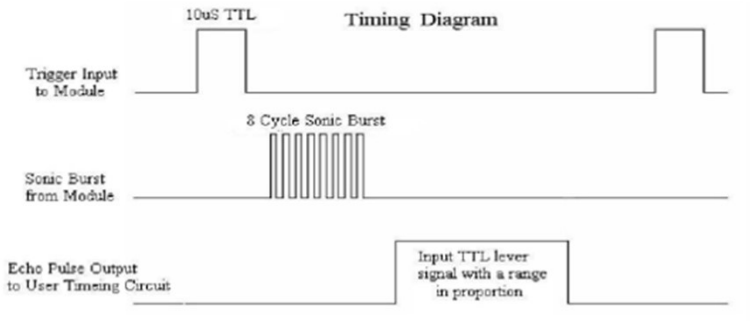
- 8 LEDs is fun – but 241 LEDs is MORE fun!
  - TLDR – WS2812B individually addressable color LEDs using FastLED Library
- Displays moving patterns we choose
- Disk is mostly wired; less soldering needed
- Needs more power than Arduino can give

- Video: https://youtube.com/shorts/0KehSIJmKcs

- Pictures from amazon.com
  - TLDR - https://www.amazon.com/WESIRI-WS2812B-Individually-Addressable-Controller/dp/B083VWVP3J

03 – Ultrasonic Sonar Control





- Ultrasonic Sonar sensor detects distance
- Bounces ultrasonic sound off objects

- Set “Trig” HIGH then LOW to start sonic burst
- Measure time until “Echo” is HIGH
- Divide time by speed of sound to get distance

- TLDR diagram from the sparkfun.com HC-SR04 spec.

03 – Ultrasonic Sonar Detector Software

Include library, define names for pins

"my_ultra" is how I use the HC-SR04

Make a global "gUltraDistance" for debugging

"handle_ultra" returns a pattern number

"my_ultra.read(CM)" gives distance in CM stored in "gUltraDistance"

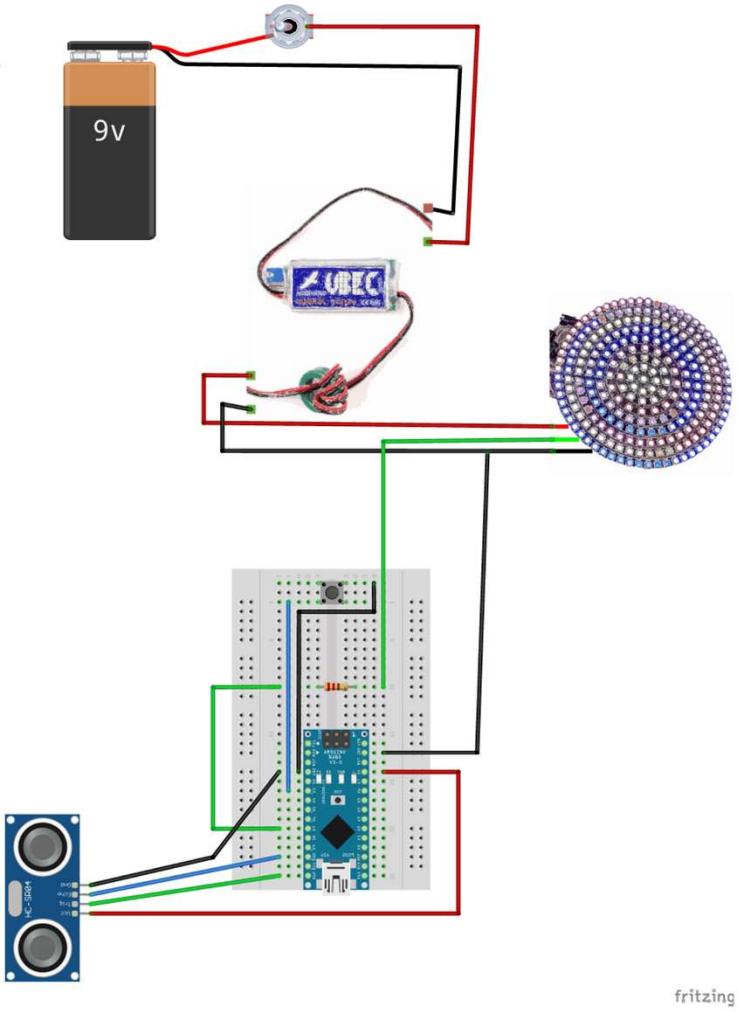
Use math to turn distance into "pattern"

Return "pattern" (number) to caller

```
#include <FastLED.h>
#include <Ultrasonic.h>

// Ultrasonic HC_SR04 definitions
#define ULTRA_TRIG_PIN 12 // HC-SR04 Trigger digital pin
#define ULTRA_ECHO_PIN 10 // HC-SR04 Trigger echo pin
#define ULTRA_CM_PER_REGION 9 // HC-SR04 every this many CM is a different pattern
#define ULTRA_IGNORE_INITIAL_CM 3 // HC-SR04 ignore the first 3 CM since valid range st

// instantiate my HC-SR04 data object
Ultrasonic my_ultra = Ultrasonic(ULTRA_TRIG_PIN, ULTRA_ECHO_PIN); // default timeout is
```

```
int gUltraDistance = 0; // latest measured distance in centimeters

////////////////////////////////////////////////////////////////////////
// handle_ultra() - process HC-SR04 data.
//    returns: pattern number 0 <= num <= PATTERN_MAX_NUM
//

int handle_ultra() {
  int pattern; // integer pattern number from 0 thru 5 inclusive
  // get the range reading from the Ultrasonic sensor in centimeters
  int ultra_dist;

  gUltraDistance= (my_ultra.read(CM));
  ultra_dist = gUltraDistance - ULTRA_IGNORE_INITIAL_CM;
  if (ultra_dist < 0) ultra_dist = 0;
  pattern = ultra_dist / ULTRA_CM_PER_REGION;
  if (pattern > 5) pattern = 5;

  return(pattern);
} // end handle_ultra()
```

03 – Fritzing Diagram



Replace 8-LED stick with 241-LED Disk

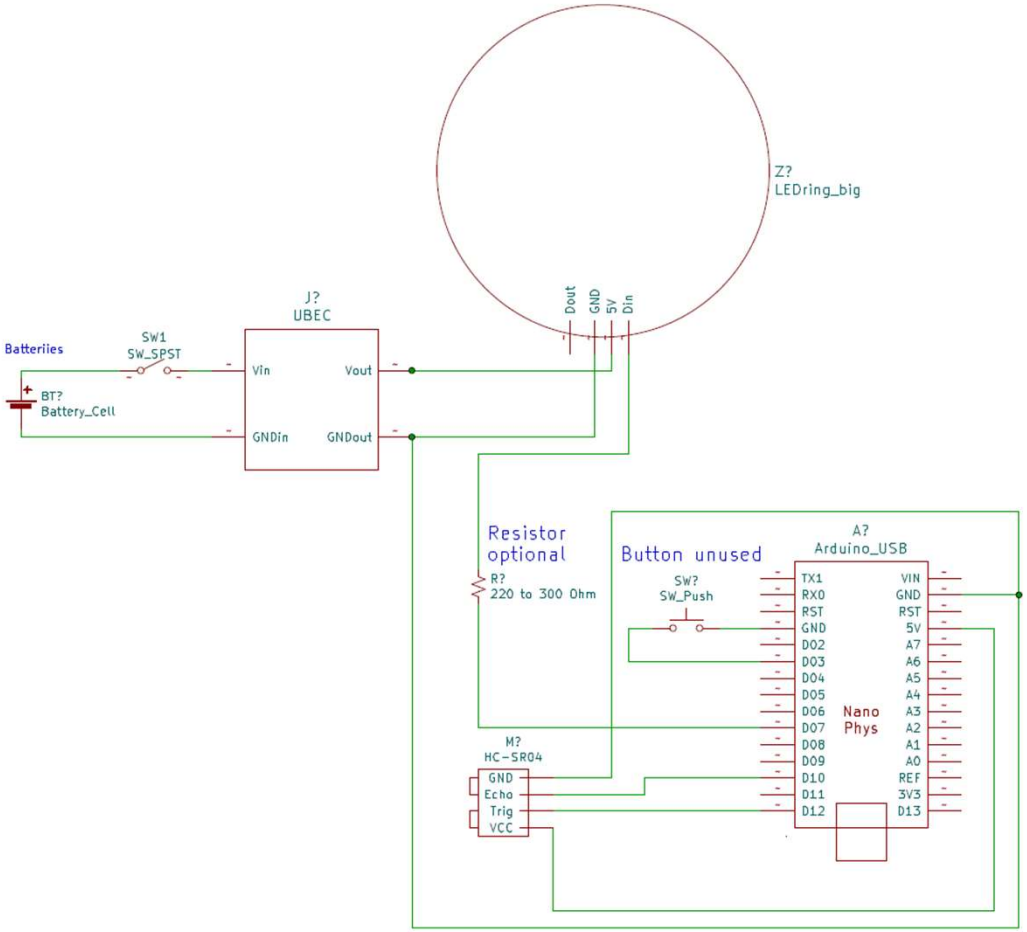
- Separate power for 241-LED Disk
- We add the battery and UBEC because the 241 LEDs take a lot of power!
- Two wires for connection

Add new HC-SR04 Ultrasonic Sensor

- Four wires for connection

Button unused – leave it there for next session

03 – Schematic Diagram



The button connected to pin D03 is not used for this project.

We leave it on the breadboard for next session

We connect the UBEC ground to Arduino, LEDs, and HC-SR04.

That allows both LED and HC-SR04 to recognize the voltages from the Arduino.

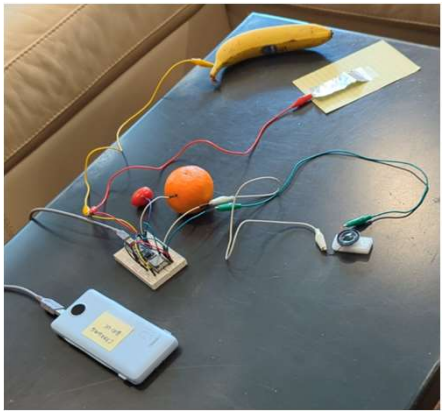
Two wires connect the LED, UBEC, and Batteries.
Four wires connect the HC-SR04 Ultrasonic Range Detector.

First switch on UBEC/LED power, then connect Arduino.

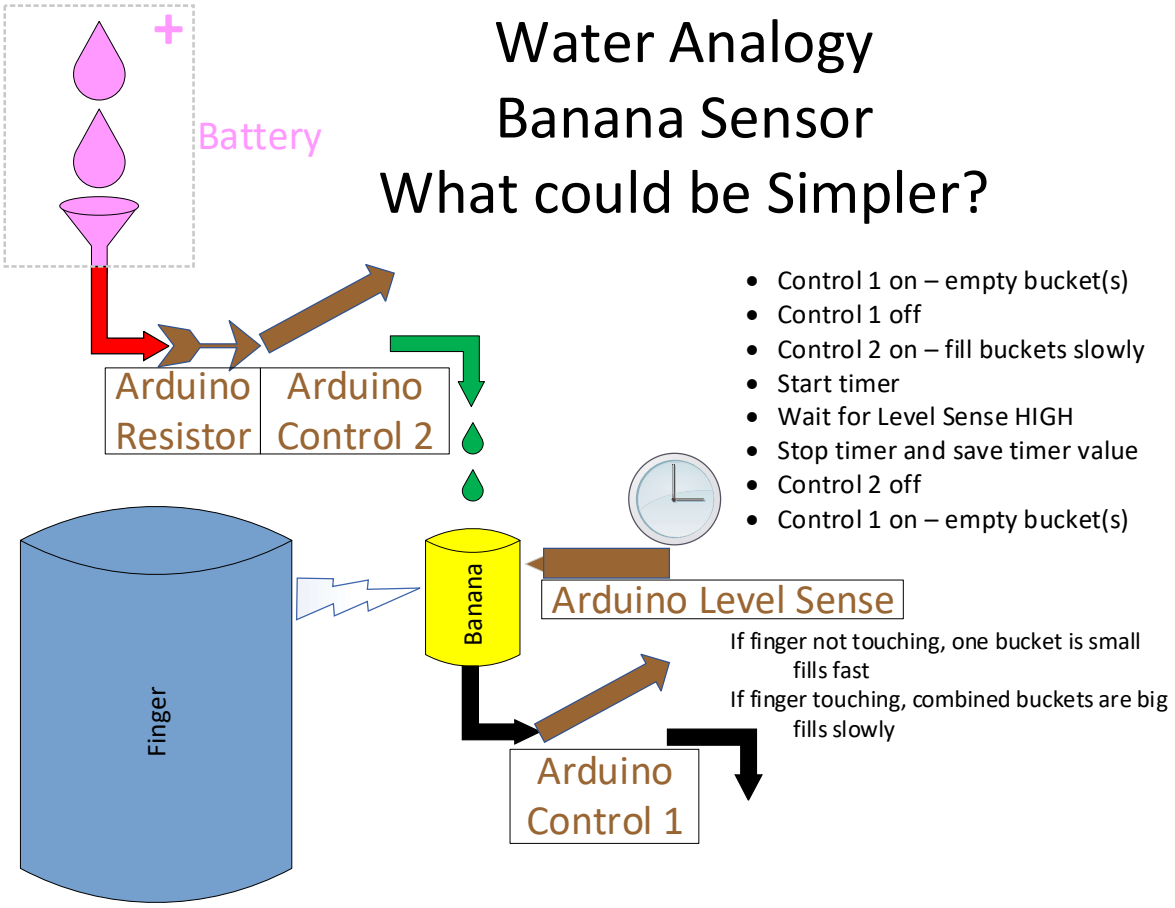
04 – Go Bananas!

- A Banana Piano! Several "key" fruit types…
  - Uses "capacitive sensing" – depends on how fast Arduino can charge it up
    - Touching banana makes it take longer to charge
- Digitized sounds are played when the "key" is pressed
- Only one sound at a time is played
- Sounds are "sampled"
  - Digital storage format
- Video: https://youtu.be/EC1qHbE89JI

- TLDR – banana image by krakenimages.com on Freepik



04 – Water Analogy - Banana Input



Water Analogy
Banana Sensor
What could be Simpler?

- Control 1 on – empty bucket(s)
- Control 1 off
- Control 2 on – fill buckets slowly
- Start timer
- Wait for Level Sense HIGH
- Stop timer and save timer value
- Control 2 off
- Control 1 on – empty bucket(s)

If finger not touching, one bucket is small fills fast
If finger touching, combined buckets are big fills slowly

Control 1 – pin OUTPUT set to LOW
Control 2 – pin INPUT with pull-up resistor

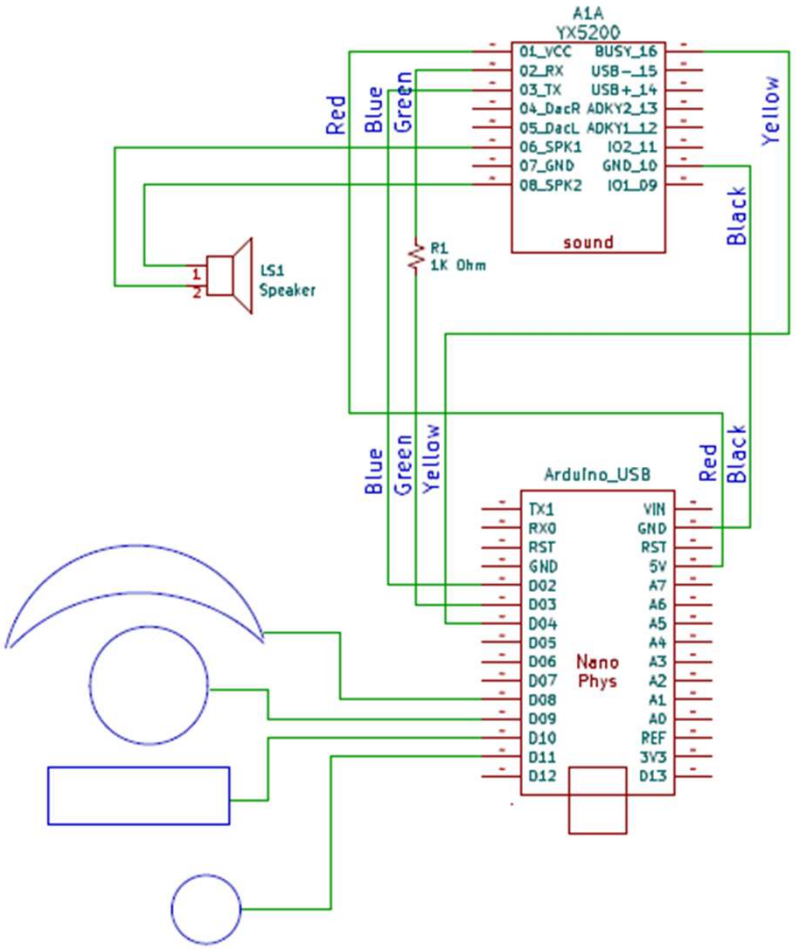
Note that this is an analog of how it works – not exactly correct

The diagram shows the flow of the code

The code uses tricks to be fast – makes code harder to understand

- But that is all hidden in 04_readCapacitivePin.*

04 Bananas Schematic



Bottom half has Arduino and fruits

- Capacitive sensing of fruits

Top half has sound module and speaker

- UART serial interface
- Universal Asynchronous Receiver / Transmitter

04 Bananas Software – customization

```
// "pin index" state:
//   -1 means nothing selected
//   0-3 (3 = NUM_MEASURE_PINS-1) represent pins MEASURE_PIN_01 to MEASURE_PIN_03,
//   any other value is invalid
int8_t gCurrentPinIndex = -1; // Index number of which PinIndex is current - nothing selected
int8_t gPrevPinIndex = -1; // previous PinIndex - nothing selected

// "pin index" to sound mapping
//    pin index goes from -1 to 3 (3 = NUM_MEASURE_PINS-1)
//    we add one to that number to go from 0 to 4
//       0 = Silence sound
//       1 through 4 = MEASURE_PIN_01 through MEASURE_PIN_04
uint16_t gPinIndex2SoundNum[1+NUM_MEASURE_PINS] = { SOUNDNUM_silence, SOU
```

Place sound numbers into array gPinIndex2SoundNum[]

Calls to pin2soundnum() will convert PinIndex to sound number

Most of routine is error checking (this is normal)

Only one line is needed to actually do translation

Could just use gPinIndex2SoundNum[pinIndex+1] if wanted to skip error checking

```
////////////////////////////////////////////////////////////////////
// pin2soundnum(pinIndex) - convert pin index to sound number
//
// pinIndex to sound mapping
//    pinIndex goes from -1 to 3 (3 = NUM_MEASURE_PINS-1)
//    we add one to that number to go from 0 to 4
//       0 = Silence sound
//       1 through 4 = MEASURE_PIN_01 through MEASURE_PIN_04

uint16_t pin2soundnum(int8_t pinIndex) {
  uint16_t soundNum = SOUNDNUM_INVALID;
  if ((pinIndex >= -1) and (pinIndex < NUM_MEASURE_PINS)) {
    soundNum = gPinIndex2SoundNum[pinIndex+1];
  } else {
    Serial.print("ERROR pin2soundnum() - pinIndex="); Serial.pri
    soundNum = SOUNDNUM_INVALID; // not really needed
  }
  return(soundNum);
} // end pin2soundnum()
```

04 Bananas Software – loop() - repeats

- Executes block every 50 milliseconds
- Gets active PinIndex
- If this is change in PinIndex: start sound
  - Could be “silence” sound
- If not a change, still holding: repeat sound
- No key (fruit) touched: repeat silence
- Extra credit: can you simplify the last if/else/endif ?
- Extra credit: can you write simple code to replace EVERY_N_MILLISECONDS(50) ? Do we even need this?

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// loop()
void loop() {
  EVERY_N_MILLISECONDS( 50 ) {
    gCurrentPinIndex = handle_capacitive(); }
    if (gPrevPinIndex != gCurrentPinIndex) {
      // "key" (PinIndex) is different than before so start a new sound
      //    -1 will start the silent sound, otherwise the chosen key sound will start
      gPrevPinIndex = gCurrentPinIndex;
      DFstartSound(pin2soundnum(gCurrentPinIndex), SOUND_DEFAULT_VOL);
    } else if (DFcheckSoundDone()) {
      if (gCurrentPinIndex >= 0) {
        // PinIndex is not -1 so we are still holding a key down - restart sound
        gPrevPinIndex = gCurrentPinIndex;
        DFstartSound(pin2soundnum(gCurrentPinIndex), SOUND_DEFAULT_VOL);
      } else {
        // PinIndex is -1 so no key is held down - start silence sound
        gPrevPinIndex = -1;
        DFstartSound(pin2soundnum(gCurrentPinIndex), SOUND_DEFAULT_VOL);
    } // end if
  } // end EVERY_N_MILLISECONDS
} // end loop()
```

04 Bananas Software – setup() - one time

Initialize serial debug interface

Do capacitive sensing setup

Initialize sound card interface

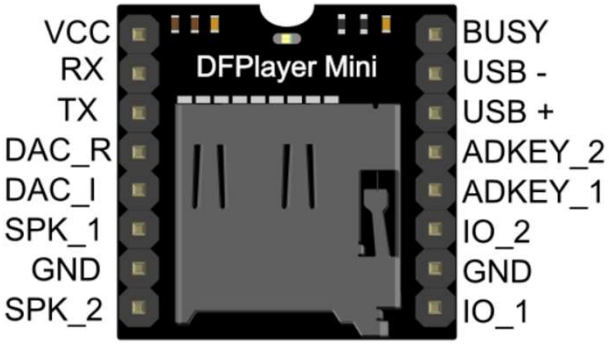
Initialization complete

Play intro sound then start

```
////////////////////////////////////////////////////////////////////////////////
// setup()
void setup() {
  Serial.begin(115200);         // this serial communication is for general debug; set
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  Serial.println(""); // print a blank line in case there is some junk from power-on

  CapacitiveSetup();

  pinMode(DPIN_AUDIO_BUSY,  INPUT_PULLUP); // HIGH when audio stops
  mySoftwareSerial.begin(9600); // this is control to DFPlayer audio player
  // initialize the YX5200 DFPlayer audio player
  DFsetup();

  Serial.println("TODAS init complete...");

  // play the INTRO sound to completion, then allow normal loop() processing
  DFstartSound(SOUNDNUM_introduction, SOUND_DEFAULT_VOL);
  while (!DFcheckSoundDone()) {
    delay(10); // wait for the INTRO sound to finish
  } // end while
  Serial.println("Intro Sound Complete");
  gCurrentPinIndex = gPrevPinIndex = -1;
  DFstartSound(SOUNDNUM_silence, SOUND_DEFAULT_VOL);

} // end setup()
```

04 – Digitized Sound Output

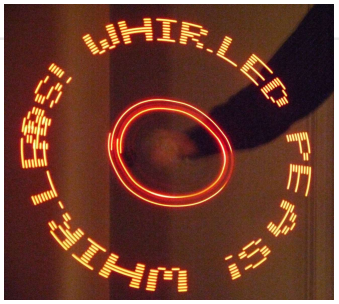
- DFPlayer (YX5200) accepts SD card with digitized audio

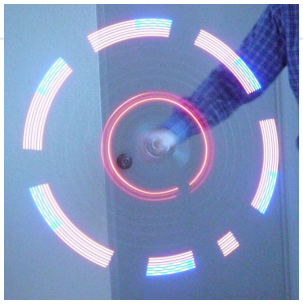


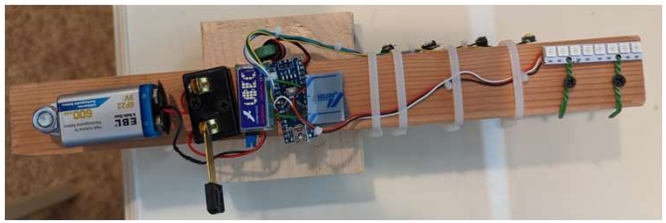
- Plays SD card audio files by number
- UART interface to Arduino
- Direct mono speaker output
- Line-out stereo output, usable for BlueTooth

- Many tricks to using YX5200
  https://github.com/Mark-MDO47/AudioPlayer-YX5200
- Espeak for robotic voice (free!)
  https://github.com/Mark-MDO47/RubberBandGun/tree/master/sounds
- Audacity for sound processing (free!)
  https://www.audacityteam.org/

Resources – Arduino

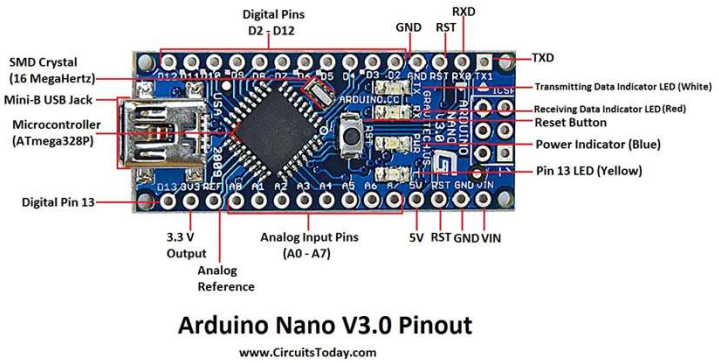






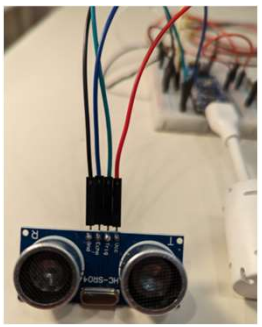
If you already know a bit of programming…

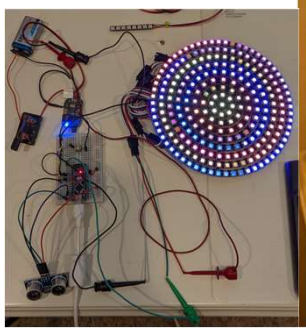
https://github.com/Mark-MDO47/ArduinoClass

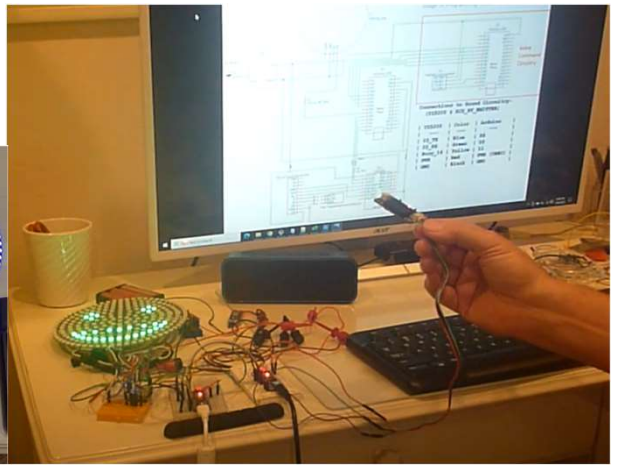


Arduino Nano V3.0 Pinout

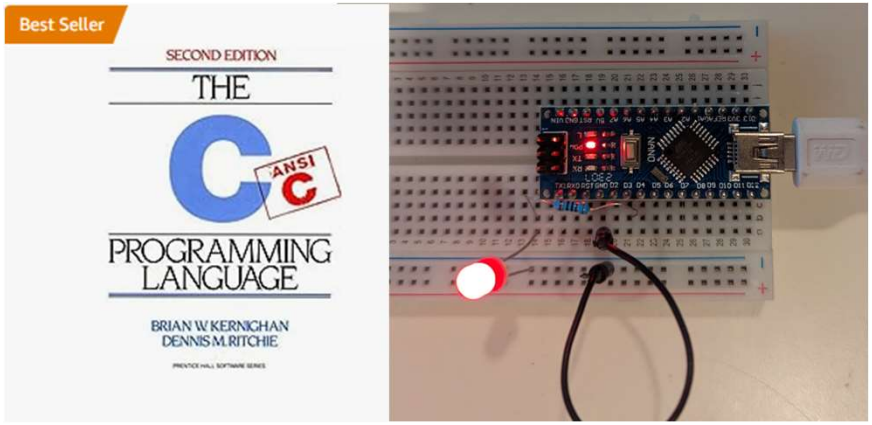
www.CircuitsToday.com







Resources – C for Arduino



If you can use a programming refresher…

https://github.com/Mark-MDO47/CforArduinoClass