RAMPED – Summer 2016

Lesson Plan: Tanya Harris
The Game of Pig: Using the Arduino as dice!

P = Pretest (think essential questions)
O = Objectives (measurable - see Bloom's taxonomy)
C = Catch (hook, anticipatory set, etc... use different senses, not a question)
A = Activity (procedure of what the students should do)
R = Review (how will students go over what they've learned?)
A = Assessment (formative and/or summative)
P = Posttest (same as pretest for comparison purposes)
S = Standards (Wyoming, NGSS, etc...) showcasing crosscutting concepts[1]

Pretest Questions	What is an Arduino? Answer: An electronic platform that can be used to build interactive projects. What is a resistor? Answer: An electronic component that limits or regulates the flow of electricity. What is probability? Answer: The likeliness something will happen. How many possible outcomes can you get when you roll two dice?
Objectives	Obj. 1: Students will be introduced to the Arduino platform. Obj. 2: Students will have an introduction to basic electronics. Obj. 3: Students will be introduced to computer programming Obj. 4: Students will understand the probability of two dice being rolled and their outcome.
Catch	1. Teach students to play the game of pig with two dice. (Rules to the game is attached) What if I told you, you can play this dice game without me giving you any dice? How would you do that? ● Have students brainstorm ideas outloud.
Activity	Students will build an Arduino Breadboard capaple of running the Dice program on the Arduino. They will then play the game of pig using the Arduino. (Activity and Schematics attached.) They will then calculate the Probability of the combination of possible outcomes for each total of two dice. (Worksheet Attached)
Review	1. What is a Breadboard? 2. What is a Resistor? 3. What is an LED 4. What is probabilty? 5. How many possible combinations are there when you role two dice?

[1] http://ngss.nsta.org/CrosscuttingConceptsFull.aspx

Assessments	The ability to play the game will assess wheter or not the Arduino is functional. The probability chart that the student will do will assess there ability to understand outcomes of two dice.
Posttest Questions (same as pretest questions)	What is an Arduino? Answer: An electronic platform that can be used to build interactive projects. What is a resistor? Answer: An electronic component that limits or regulates the flow of electricity. What is probability? Answer: The likeliness something will happen. How many possible outcomes can you get when you roll two dice?
Standards	CCSS.MATH.CONTENT.7.SP.C.7 Develop a probability model and use it to find probabilities of events. CCSS.MATH.CONTENT.7.SP.C.7.B Develop a probability model (which may not be uniform) by observing frequencies in data generated from a chance process.
Crosscutting Concepts from NGSS	MS-ETS1-4 Engineering Design Develop a model to generate data for iterative testing and modification of a proposed object, tool, or process such that an optimal design can be achieved. Performance Expectation. (Grade: Middle School (6-8))

PIG

A Probability Experiment

DIRECTIONS

- The teacher needs two dice.
- Toss the die and announce the results.
- Students write down that number.
- Toss the dice and announce the results.
- Students write down that number and add it to the previous number.
- Toss the die and announce the results.
- Students write down that number and add it to the previous total.
- Continue playing and accumulating points.
- Players may continue to accumulate points until a pair-of-ones are tossed. When a pair-of-ones is tossed, every student still playing loses all of his/her points for that round.
- A player may decide to stop at any point before the dice are thrown again. He/she puts down his/her pencil and stands quietly at the desk. Once standing, the student may not collect any more points. He/she gets to keep all of the points earned before standing.
- Play continues until a pair-of-ones are thrown, or until all students are standing.
- A game is three rounds. Highest point total wins the game.

MODIFICATIONS: Let students write down points until they wish to stop. At the end of the round, let students total all points, using a calculator if desired.

Pig:

RAMPED – Summer 2016

Probability of Two Dice

Total to Roll	**Ways to Get the Total**	**Probability of that Roll**
2	**1**	**1 /36**
3		**/ 36**
4		**/ 36**
5		**/ 36**
6		**/ 36**
7	**6**	**6 /36 = 1/6**
8		**/ 36**
9		**/ 36**
10		**/ 36**
11		**/ 36**
12		**/ 36**

RAMPED – Summer 2016

ANSWER KEY: PROBABILITY OF TWO DICE

When he's done, the chart should look like this:

Total to Roll	Ways to Get the Total	Probability of that Roll
2	1	1 / 36
3	2	2 / 36 = 1/18
4	3	3 / 36 = 1/12
5	4	4 / 36 = 1/9
6	5	5 / 36
7	6	6 / 36 = 1/6
8	5	5 / 36
9	4	4 / 36 = 1/9
10	3	3 / 36 = 1/12
11	2	2 / 36 = 1/18
12	1	1 / 36

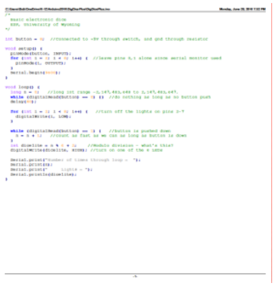
Project: digital dice

Open **DigitalDice.pdf**

(from Workshop Arduino Directory)

Let’s figure out what it does

Bob: Click Here or
open DigitalDice.ino



```
/*
  Basic electronic dice
  RBP, University of Wyoming
*/

int button = 8;  //connected to +5V through switch, and gnd through resistor

void setup() {
  pinMode(button, INPUT);
  for (int i = 2; i < 8; i++) {  //leave pins 0,1 alone since serial monitor used
    pinMode(i, OUTPUT);
  }
  Serial.begin(9600);
}

void loop() {
  long n = 0;    //long int range -2,147,483,648 to 2,147,483,647.
  while (digitalRead(button) == 0) {}  //do nothing as long as no button push
  delay(40);

  for (int i = 2; i < 8; i++) {   //turn off the lights on pins 2-7
    digitalWrite(i, LOW);
  }

  while (digitalRead(button) == 1) {   //button is pushed down
    n = n + 1;   //count as fast as we can as long as button is down
  }
  int dicelite = n % 6 + 2;     //Modulo division - what's this?
  digitalWrite(dicelite, HIGH); //turn on one of the 6 LEDs

  Serial.print("Number of times through loop =  ");
  Serial.print(n);
  Serial.print("     Light# = ");
  Serial.println(dicelite);
}
```

Here’s our plan

- When we push the button
- Arduino counts as fast as it can

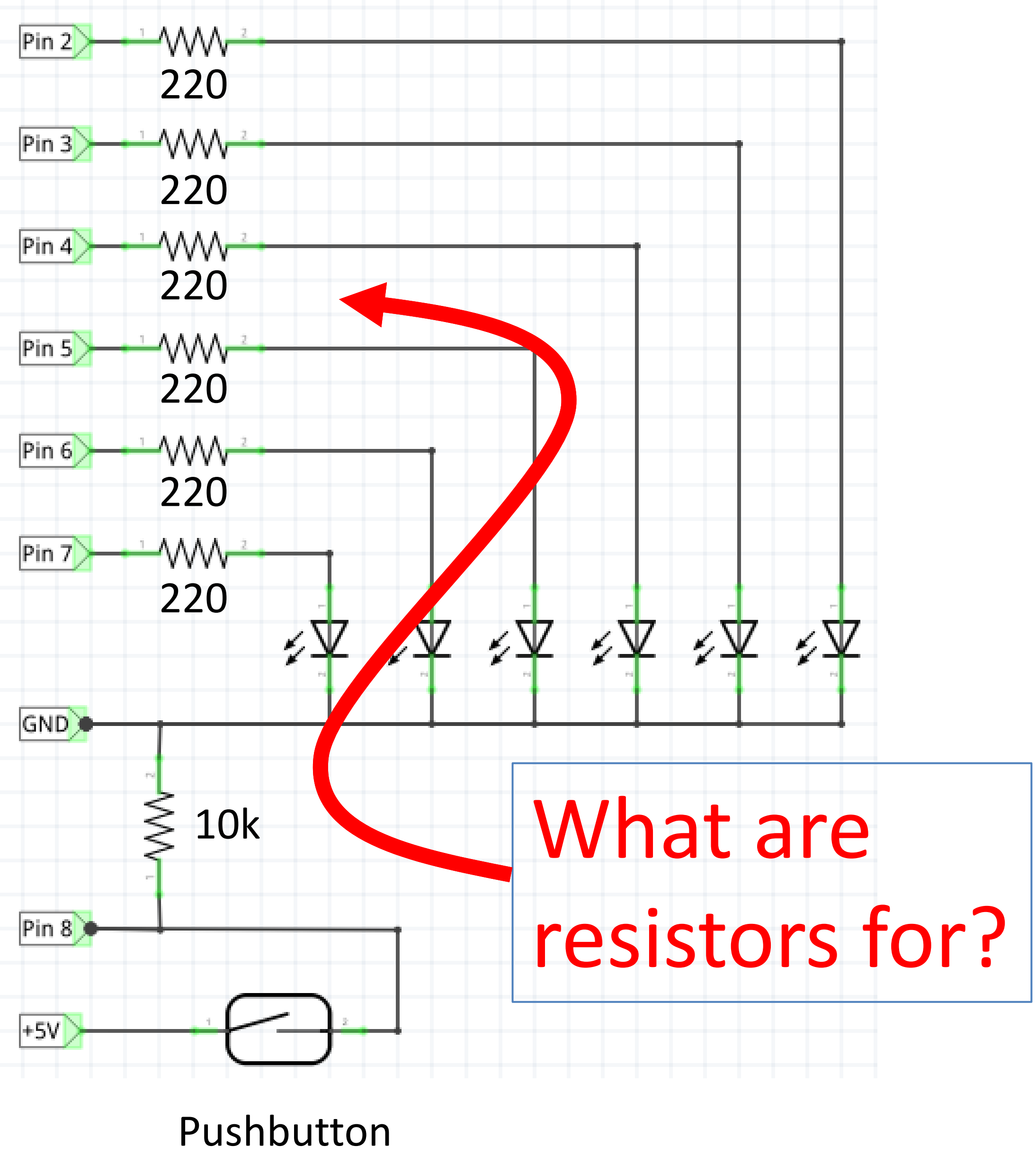
  n = number of times through loop()

  Resulting n is a “random number”
- Divide n by 6 and keep remainder
  - Modulo division: m = n%6 = 0, 1, 2, 3, 4, or 5
- Light LED on pin (m+2) represents dice throw

Type In the Code

- Load the program “DigitalDice.ino”
  - This has some (not all) of the C code entered
  - You type in the remaining code!!

Initial Digital Dice Schematic

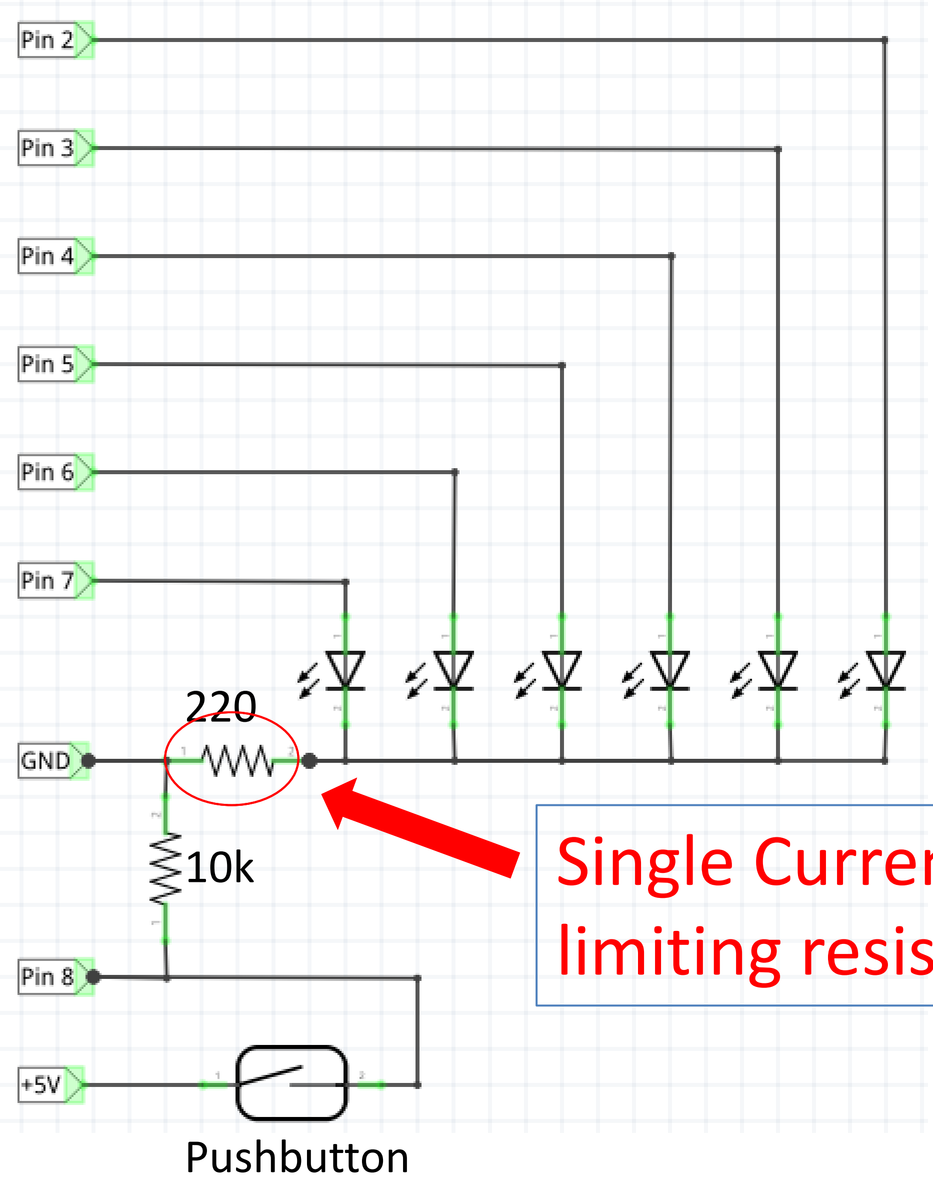


What are resistors for?

How it works:

- *Our software turns the pin* ***on*** *(+5V) or* ***off*** *(0V)*
- *Only one LED lights at a time*

Easier Digital Dice Schematic

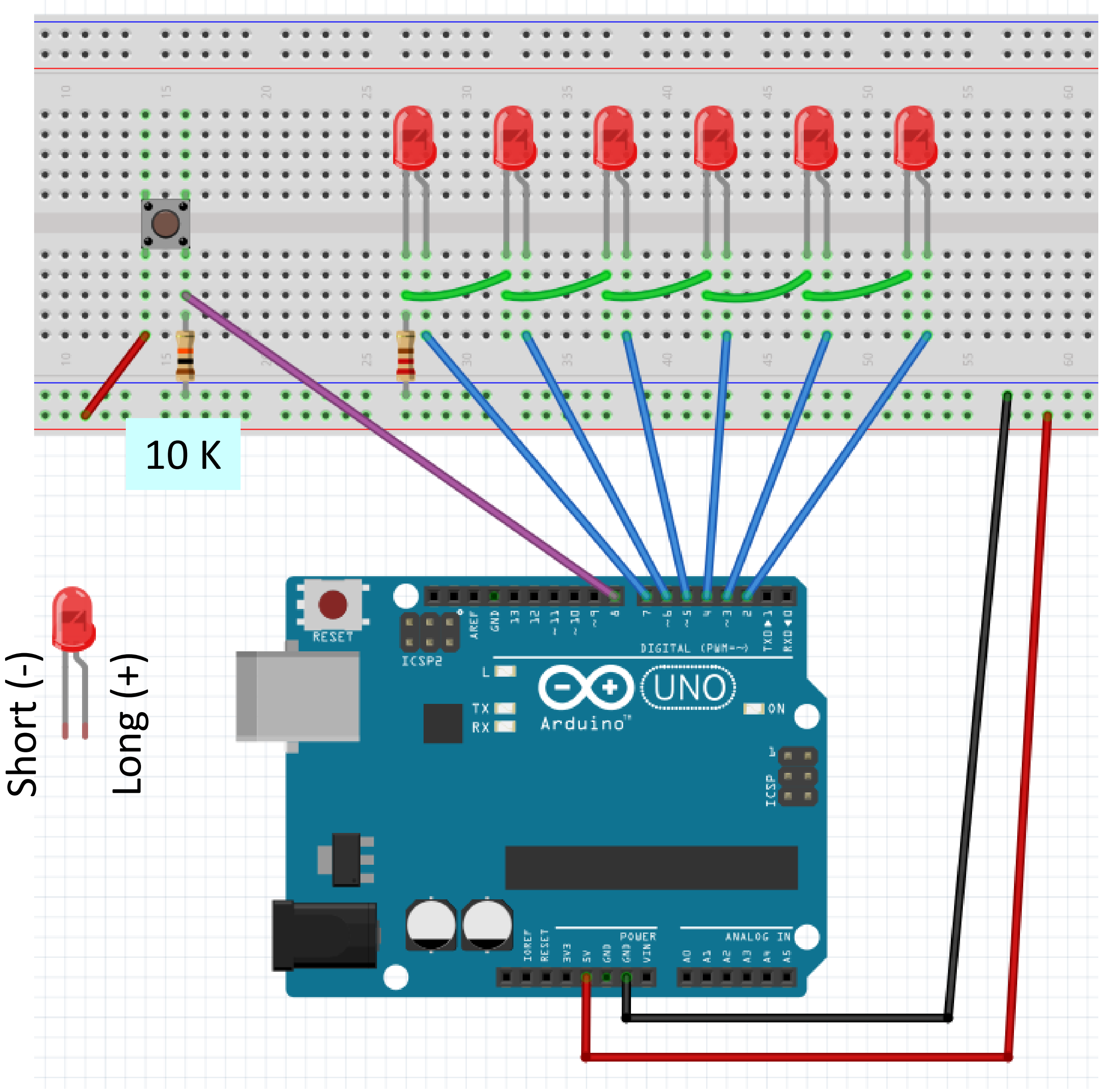


Single Current limiting resistor

***Notice*:**

- *At any time, only one LED is on*
- *Current only flows in one wire*
- *Only one current limiting resistor needed*

Build This!

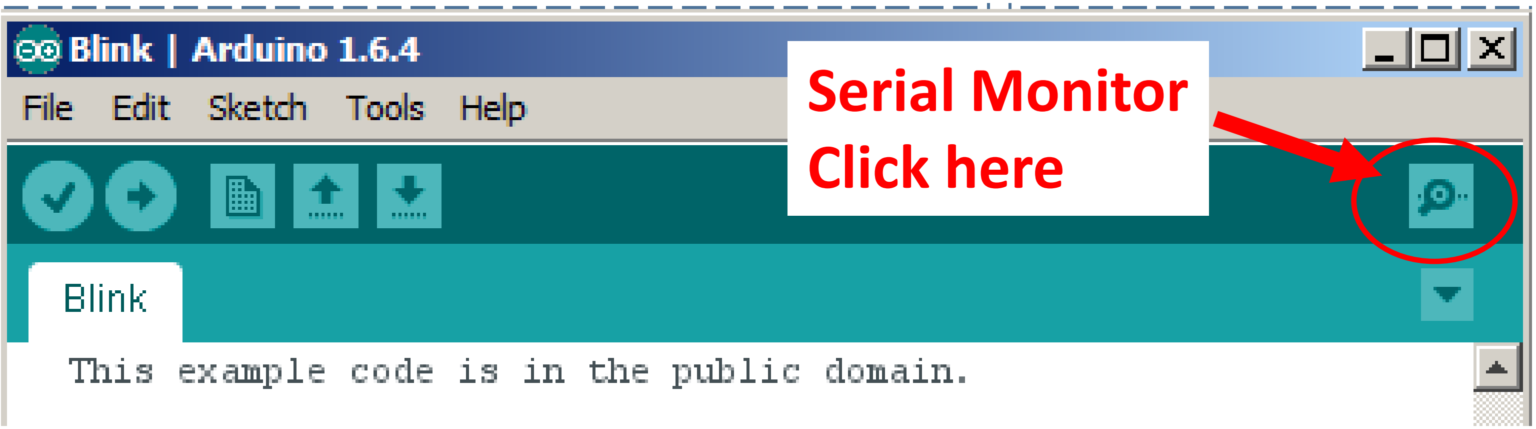


LED Legs

- **Short** goes to resistor (GND)
- **Long** goes to pins (which are +5V)

Let’s Run It

- Upload sketch and **start the Serial Monitor**
- Push the button, roll the die, and light the lights
  - Bonus: Serial Monitor reports how long you held button down (# times through loop())
- Occasional weird results
  - “Switch Bounce” to be discussed later



This example code is in the public domain.

Serial Monitor Click here

How does it work?

- Please look at the Arduino code

Using a “for” statement

- Repeat a task until condition met

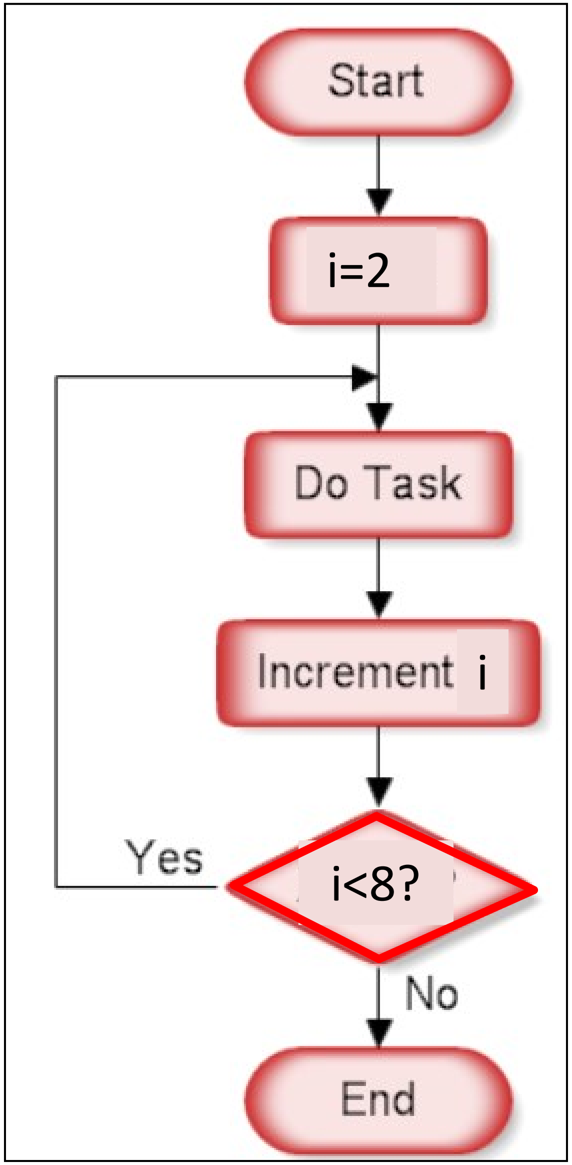
```
      Initialize      test   increment
for (int i=2;  i<8;   i++){
   pinMode(i, OUTPUT);        // “task”
}
```

Where ”i++” increments “i”

(Instead we could write: i=i+1)

Equivalent to:

```
pinMode(2,OUTPUT);
pinMode(3,OUTPUT);
pinMode(4,OUTPUT);
pinMode(5,OUTPUT);
pinMode(6,OUTPUT);
```



“for” loop flowchart

Variable Types

- There are many *types* of variables in C
- “integer” type

  ```
  int n=0;  //declares n to be integer
  ```

  - Range of possible values: -32,768 to 32,767
- Another type is “long int” or just “long”

  ```
  long n=0;
  ```

  - long int range: -2,147,483,648 to 2,147,483,647
- We’ll use long int since n can be really big!

How to wait for button push

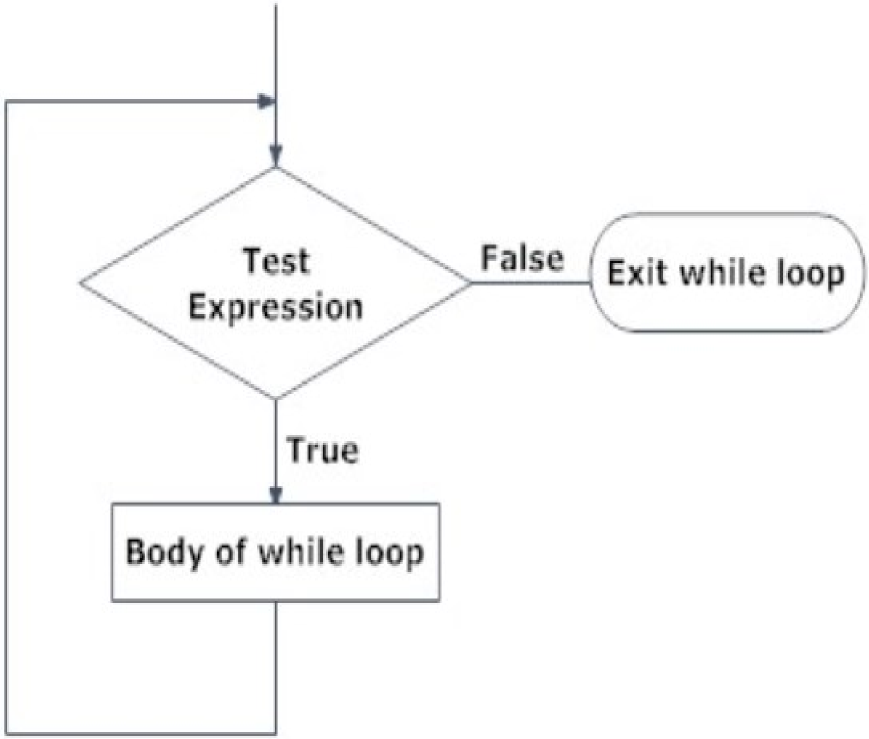
```
void loop(){

  while (digitalRead(button) == 0) {}
      //stuck here as long as button not pushed


 //button has been pushed; turn off the LEDS on pins 2-7
 for (int i = 2; i < 8; i++) {
  digitalWrite(i, LOW);
 }
```

Do nothing

Same as:

```
digitalWrite(2,LOW);
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
digitalWrite(6,LOW);
digitalWrite(7,LOW);
```

“while” loop



```
//Example: how many times can
//Arduino loop while button pushed
i=0;
while(digitalRead(8)==HIGH){
   i=i+1; //button pressed
}
```

```
//Example – repeat blink forever
while(1){
  //same as while(HIGH)
  //also same as while(true)
   digitalWrite(6,HIGH);
   delay(1000);
    digitalWrite(6,LOW);
    delay(500);
}
```

Dice continued

```
 while (digitalRead(button) == 1) {
        //stay here as long as button remains pushed
        n = n + 1;   //replace n with new value n+1
}
 Serial.print("Number of times through loop =  ");
 Serial.print(n);
 int dicelite = n%6+2;   //add 2 to get pin#
//Modulo division = remainder after divide
 Serial.print("     Light# = ");
 Serial.println(dicelite);
 digitalWrite(dicelite, HIGH); //turn on one of the 6 LEDs
}
```

0%6=0
1%6=1
2%6=2
3%6=3
4%6=4

5%6=5
6%6=0
7%6=1
…
27%6=3

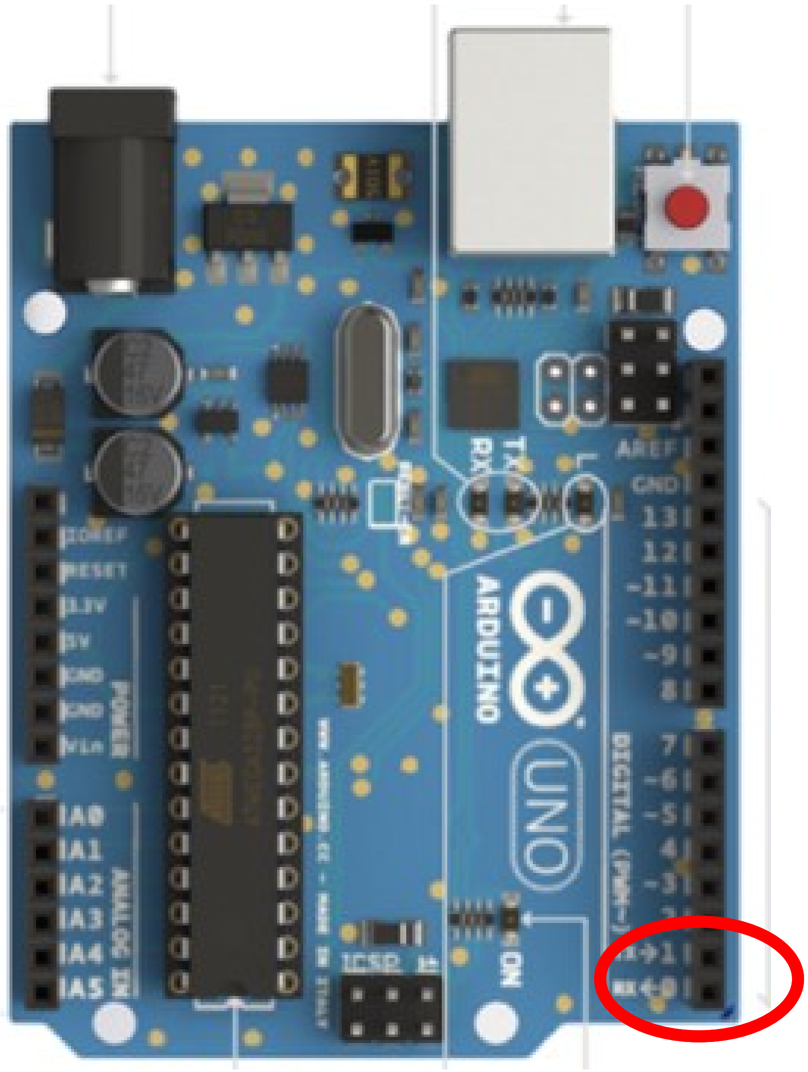
Meaning of “n = n+1”

- In Algebra it makes no sense
- In Computers it means:
  - Compute the value **n+1**
  - Store this value back into variable **n** (replacing whatever value was stored there)
- Result is to increment the variable “**n**”

Serial Monitor

- We can send information from Arduino to PC/MAC!
- “Serial communication” sends data (0’s and 1’s) on pins 0 and 1 from Arduino to the PC
- **Serial.begin(9600)** initializes serial communication at a rate of 9600 bits per second
- **Serial.print**: sends text or values to PC
- **Serial.println**: same as print, but adds “line feed” (like ‘Return’ on your keyboard)

Some Advice

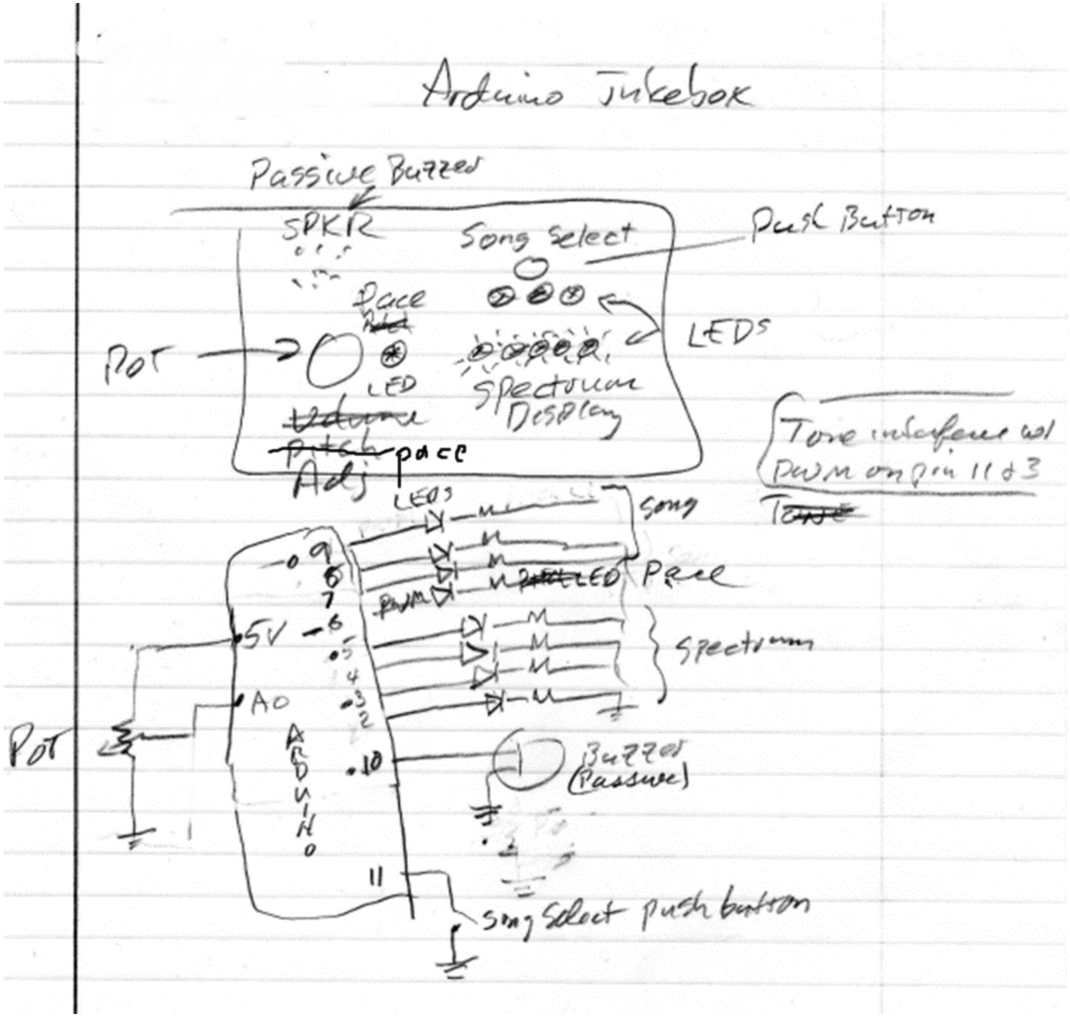


- Pins 0 and 1 have dual use (USB communication)
- Avoid using these
- Things might not work as expected!

Big Design Project

Arduino Jukebox

Initial Design



Our Goals:

- Play songs
- Use song select button
- Variable pace
- Spectrum display

Why?

- Goal is NOT to show how to build a dumb project
- Goal is to show how to build ANY project
- Be thinking about your OWN projects:
  - How to reuse computer code
  - How to repurpose hardware ideas

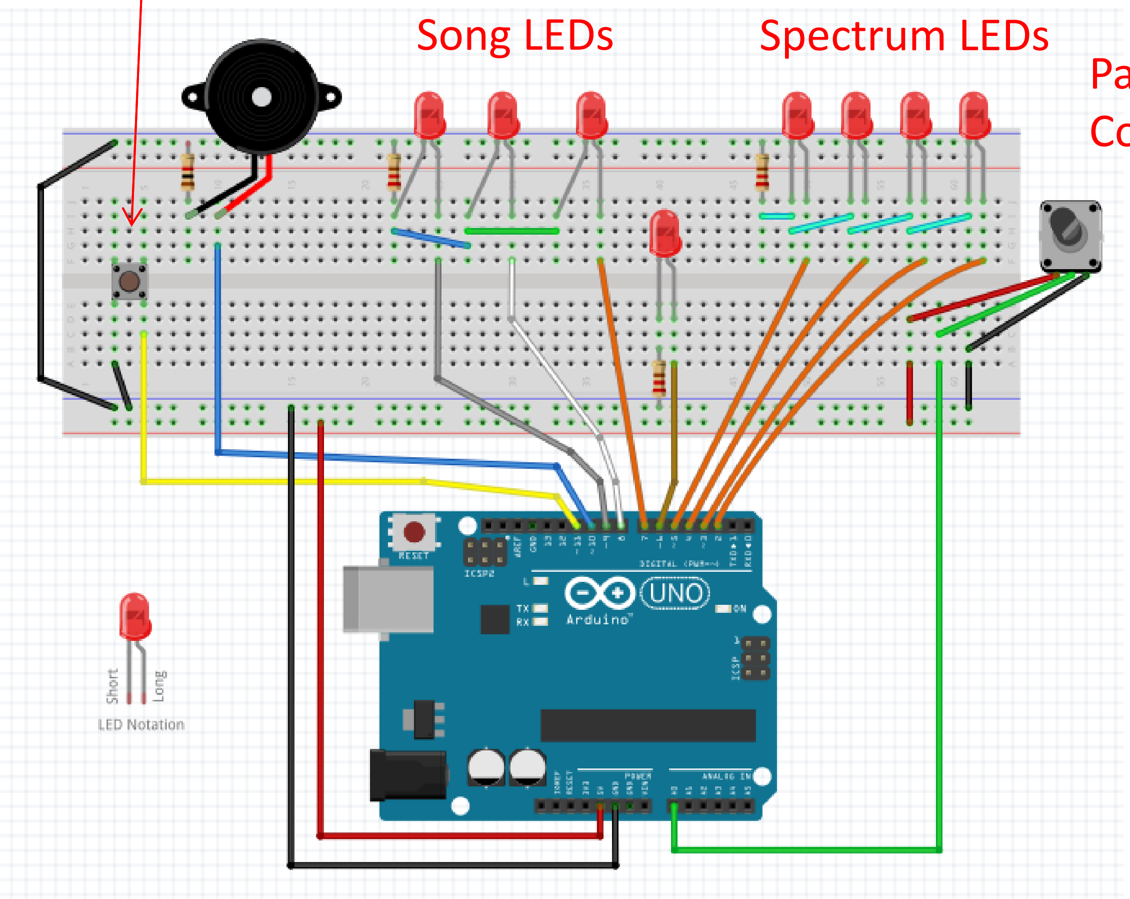
Things We Need to Learn

- General Design and Build process
- Using push buttons as Arduino inputs
- Analog input and output
  - Using potentiometer to control Arduino
  - Using analog output to control LED intensity
- Science of sound
  - What is frequency spectrum and ‘pitch’?
  - How to make Arduino generate sound

Design and Build Process

- Brainstorm design objectives and high-level sketch
- Sketch a schematic showing connections (best guesses)
- "NEVER" build entire system and THEN debug it
  - Debugging is way harder, often futile!!!!
  - Interactions between subsystems get complicated.
    - Especially if they all contain errors!!!!!!
- Start with only one subsystem
  - Thoroughly test and debug hardware and software
  - When it works, add next subsystem and debug it

Final System



Song Select button

Speaker

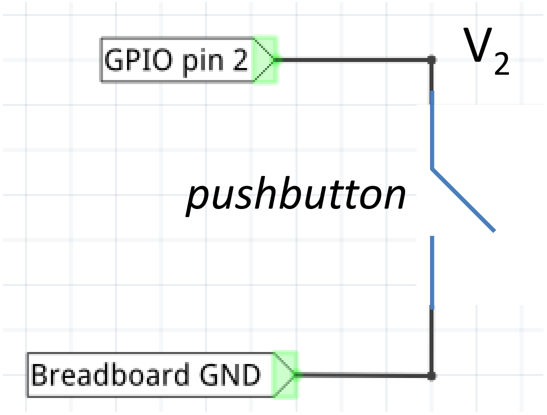
Song LEDs

Spectrum LEDs

Pace Control

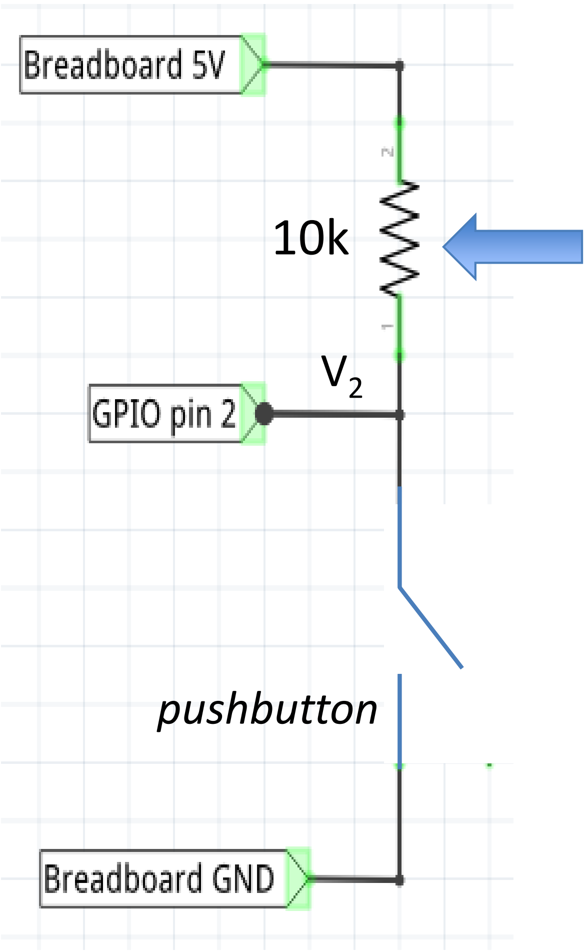
- We’ll build and test it section by section
- First: pushbutton song select

A Push Button as Arduino Input



- **Push the Button**:
  Arduino measures Pin 2:
  V_2 = 0V = LOW
  That's Good!
- **Don't push the Button**:
  Now Pin 2 not connected.
  V_2 = ???
  - Called "floating input"
  - **That's BAD!**

Fix using a "**Pull-up**" resistor



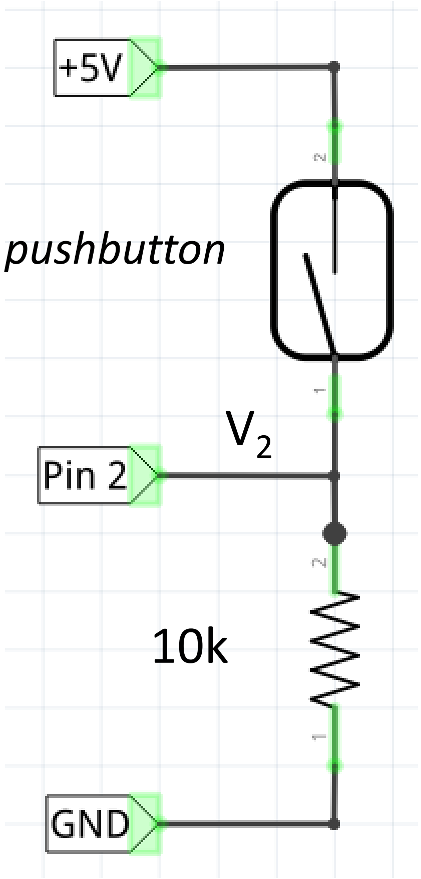
(Almost no current flows into an input pin. Since I=0 then voltage drop across resistor is V=IR=0. So $V_2 \approx +5V$ = HIGH)

Add a "**pull-up**" resistor

- *Push button: V_2 = 0 = LOW*
- *Button **not** pushed:*
  V_2 is "pulled up" to +5V = HIGH

- *(If use jumper instead of Resistor, push button shorts +5V to GND and damages Arduino)*

Or use a **Pull-<u>down</u>** resistor



Result

- *Push causes Pin 2 to be HIGH*
- *UnPush causes Pin 2 to be LOW*
- *(We used this method in Digital Dice project)*

Which is better?

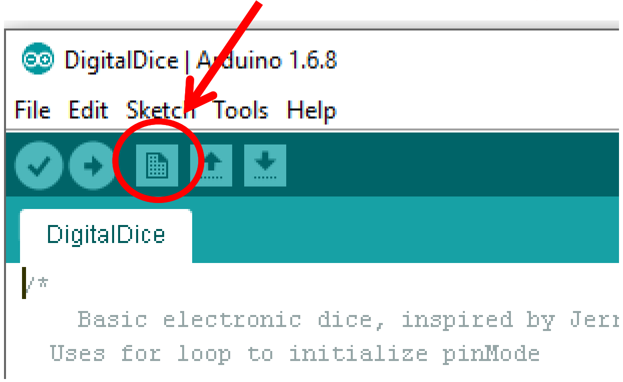
- *Your call. It's arbitrary!*
- *We'll use both*

Pushbutton and One LED

- Start a new Arduino project
- Type in the following (don't worry about the comments)

```
int songBut = 11;     // the number of the pushbutton pin
int songLED =  9         // the number of the LED pin

void setup() {
   pinMode(songLED, OUTPUT);
   pinMode(songBut, INPUT);
}
```

Click here to open new project



```
/*
   Basic electronic dice, inspired by Jerr
 Uses for loop to initialize pinMode
```

Keep Typing

```
void loop() {
  int button = digitalRead(songBut);   //read button
  if (button == LOW) {                //button pushed
    digitalWrite(songLED, HIGH);       //turn on LED
  }
  else {                               //button not pushed
    digitalWrite(songLED, LOW);          //turn off LED
  }
}
```

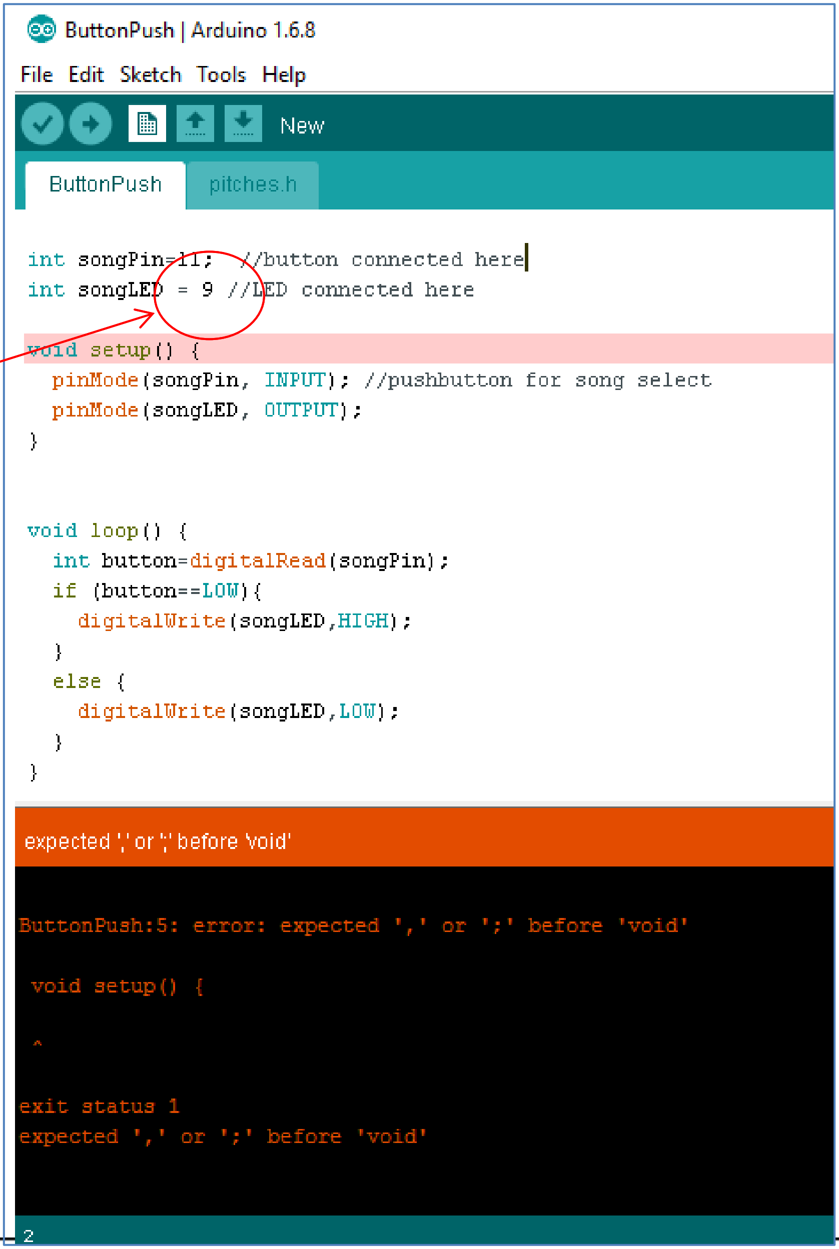
- What does “if – else” statement do?
- Upload and push button. Does it work?

“Compilation Errors”

Missing “;”

Click and drag to enlarge window

Error Messages

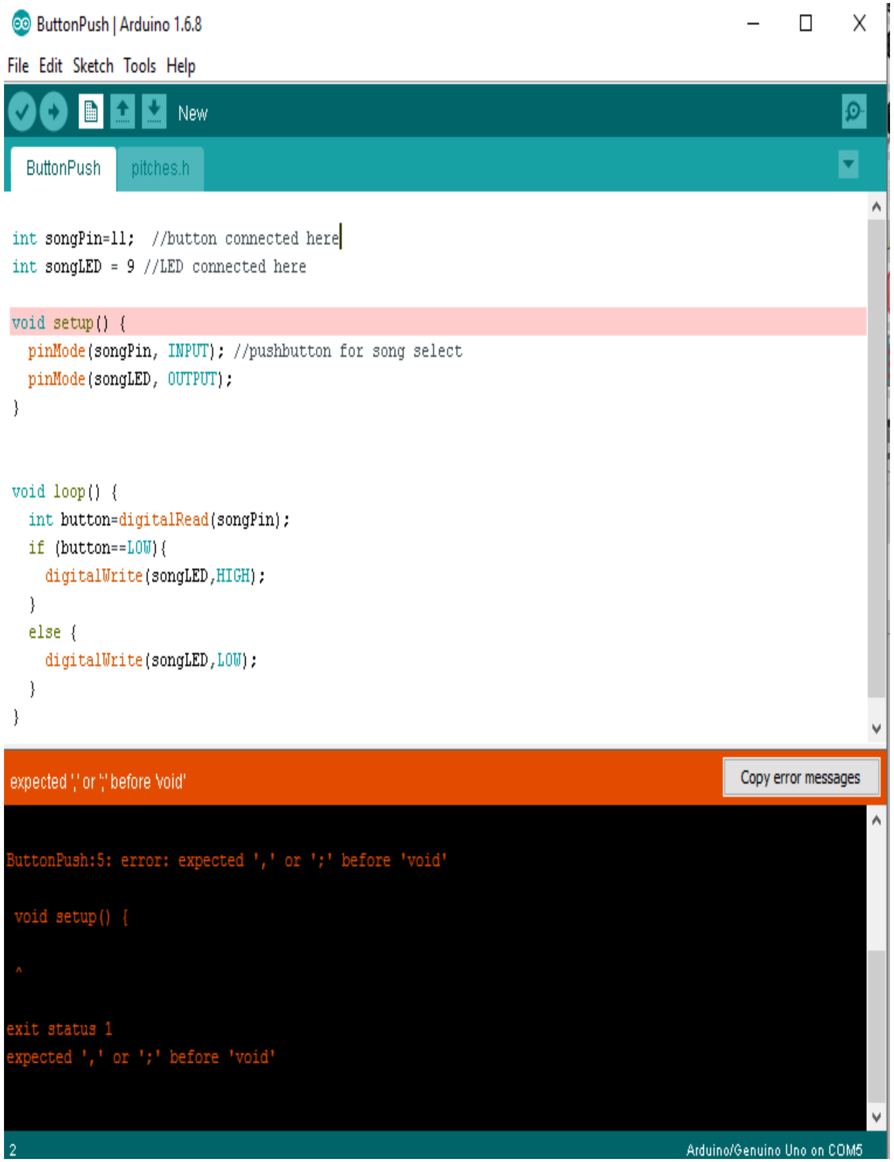


```
int songPin=11;   //button connected here
int songLED = 9 //LED connected here

void setup() {
  pinMode(songPin, INPUT); //pushbutton for song select
  pinMode(songLED, OUTPUT);
}


void loop() {
  int button=digitalRead(songPin);
  if (button==LOW){
    digitalWrite(songLED,HIGH);
  }
  else {
    digitalWrite(songLED,LOW);
  }
}
```

expected ',' or ';' before 'void'

```
ButtonPush:5: error: expected ',' or ';' before 'void'

 void setup() {

 ^

exit status 1
expected ',' or ';' before 'void'
```

“Compilation Errors”


ButtonPush | Arduino 1.6.8
File Edit Sketch Tools Help
New
ButtonPush
pitches.h
Copy error messages
Arduino/Genuino Uno on COM5

```
int songPin=11;  //button connected here
int songLED = 9 //LED connected here

void setup() {
  pinMode(songPin, INPUT); //pushbutton for song select
  pinMode(songLED, OUTPUT);
}


void loop() {
  int button=digitalRead(songPin);
  if (button==LOW){
    digitalWrite(songLED,HIGH);
  }
  else {
    digitalWrite(songLED,LOW);
  }
}
```

expected ',' or ';' before 'void'

```
ButtonPush:5: error: expected ',' or ';' before 'void'

 void setup() {

 ^

exit status 1
expected ',' or ';' before 'void'
```

2

To ask for help via e-mail

- Click here to copy error messages

- Paste into your email message:
  CTRL-V (or right-click Paste)

Common “Compile” Errors

- Capitalization errors: remember C is case sensitive
- Brackets aren’t balanced: () {} []
- Missing ‘;’ after command

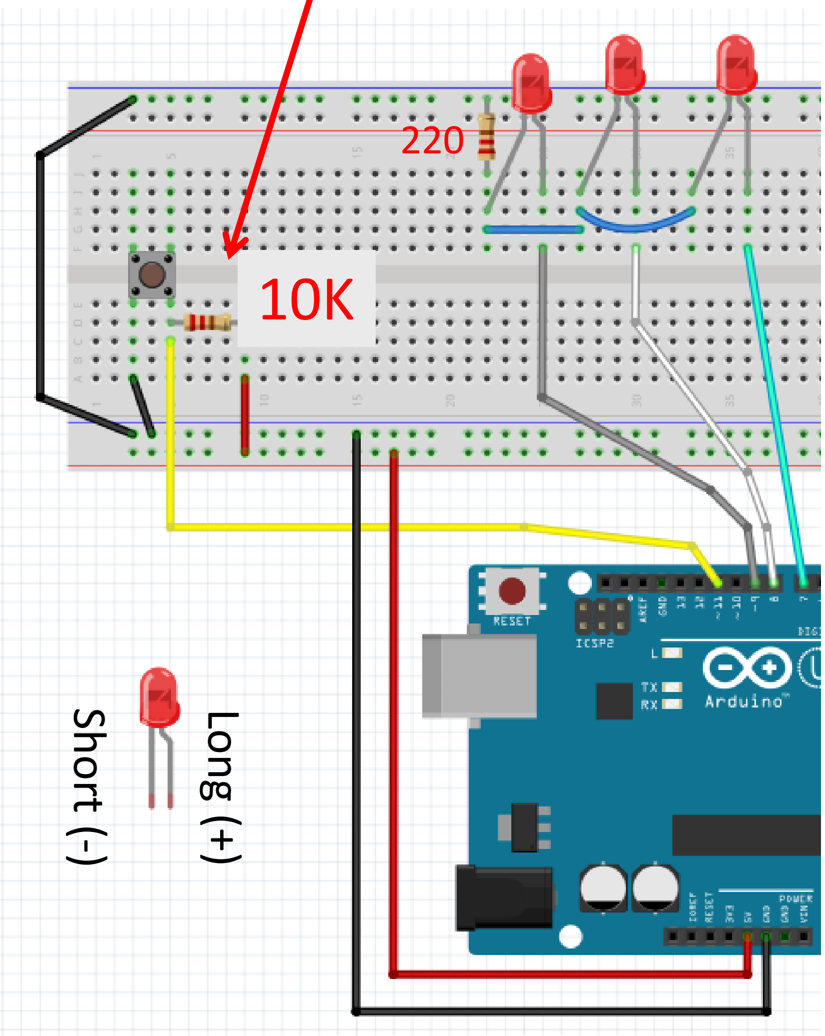
e.g., **pinmode(9,Output)) //generates errors!**

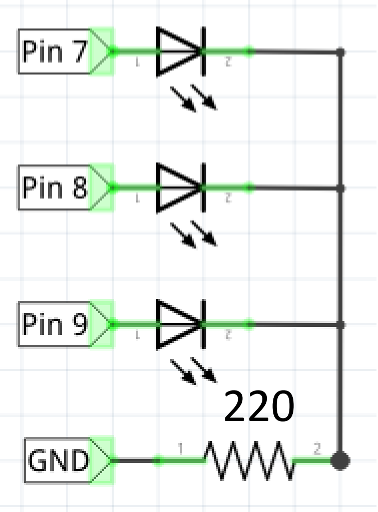
1. Capitalization: pinmode→pinMode, Output→OUTPUT
2. Brackets: pinmode(9,Output)) → pinmode(9,Output)
3. Missing ‘;’
4. Corrected: **pinMode(9,OUTPUT); //Correct!**

Build Song-Select Circuit

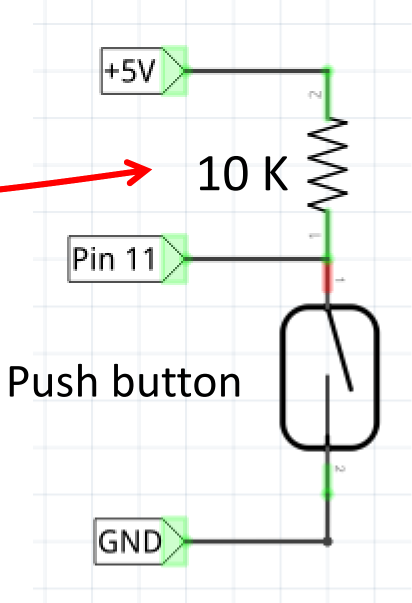
Pull-up Resistor

Song LEDs





Pull-up Resistor



Experiments

1. Fix compile errors and get circuit working

   Pushing button should light LED on pin 9

2. Next, remove the 10K pull-up resistor
   - Does the push-button still work?
   - Probably not (floating input!) Try the following:

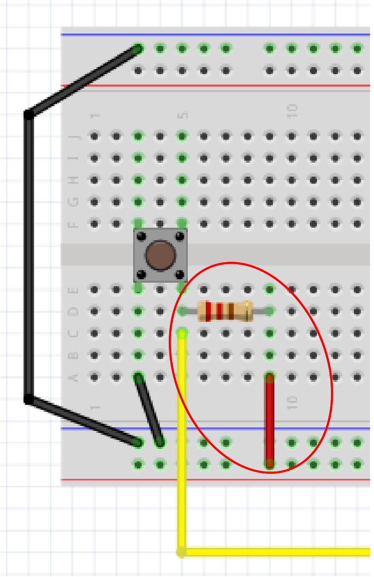
3. Change: pinMode(songBut, INPUT);

   to: pinMode(songBut, INPUT_PULLUP);
   - Upload the modified code
   - Does the push-button work now?

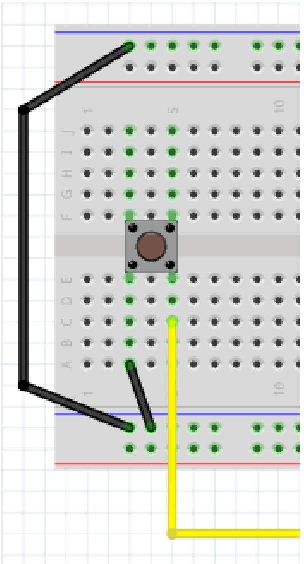
Result

- Arduinos have optional **built-in** pull-up resistors
  - Activate using:
    pinMode(buttonPin, INPUT_PULLUP)

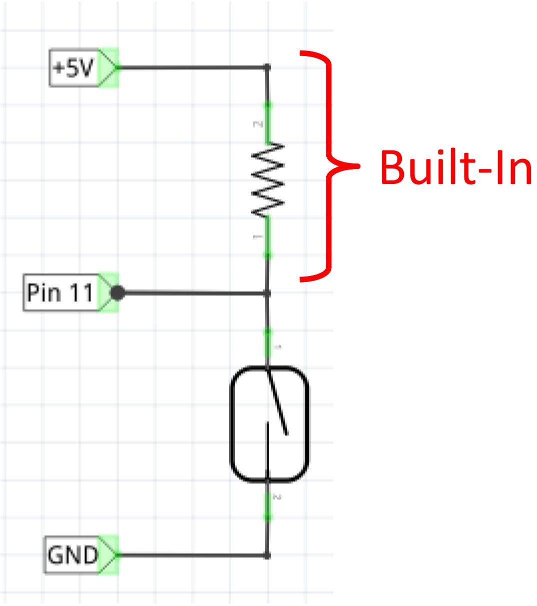
With Pull-up



Without Pull-up



Cheaper, Easier



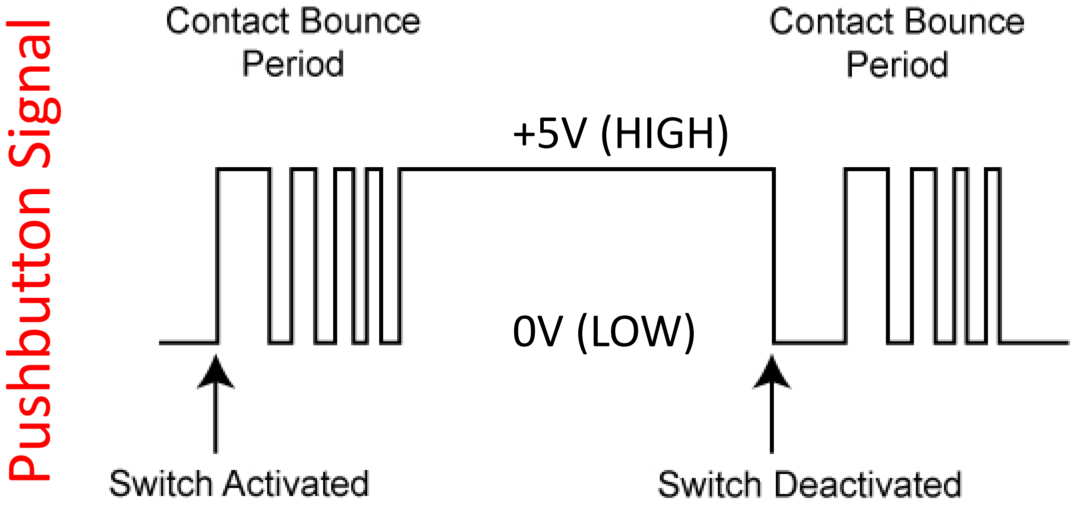
Next: Pushbutton+3 LEDs

- Please open “**ArduinoJukeboxButton.ino**” *(from Workshop Arduino directory)*

- Objectives
  - Each button push should select a different song
    - “song” variable changes 0-1-2-0-1-2-0-1-2-0 etc.
    - Song selection occurs at instant button is released
    - Alternatively we could select song when button is pressed
  - Light an LED depending on value of “song” variable
  - Avoid “Switch Bounce”

Switch Bounce

- Switch contacts bounce when turned on or off
- Result is a messy signal sent to Arduino



Problem:
One button push registers as many pushes

Our Solution:
When we detect switch change, wait 40 ms to allow bounce to settle. Then continue our program

Simple Debouncing

(just look, don’t type)

```
int lastButton=HIGH;    // HIGH is value if button not pushed
loop(){
  button = digitalRead(songBut);    // Read the button voltage
  //Check for change in button status; if so wait for bounce to settle:
  if (button != lastButton) delay(40);
  //Check: Previous state=pushed, current state=not pushed? Button was released.
  if (button == HIGH && lastButton == LOW) { do stuff}
  lastButton=button; //save button state in preparation for next loop
}
```

IF statements

- General form of “if statement”:
  if (Boolean expression) {bunch of statements}
- “Boolean” → value is true or false
- Examples of Boolean expressions:

X==Y	meaning: X equals Y
X!=Y	meaning: X not equal to Y
X>=Y	meaning: X greater or equal to Y
X==Y && W==3	meaning: “&&” is AND
X>Y || X==5	meaning: “||” is OR

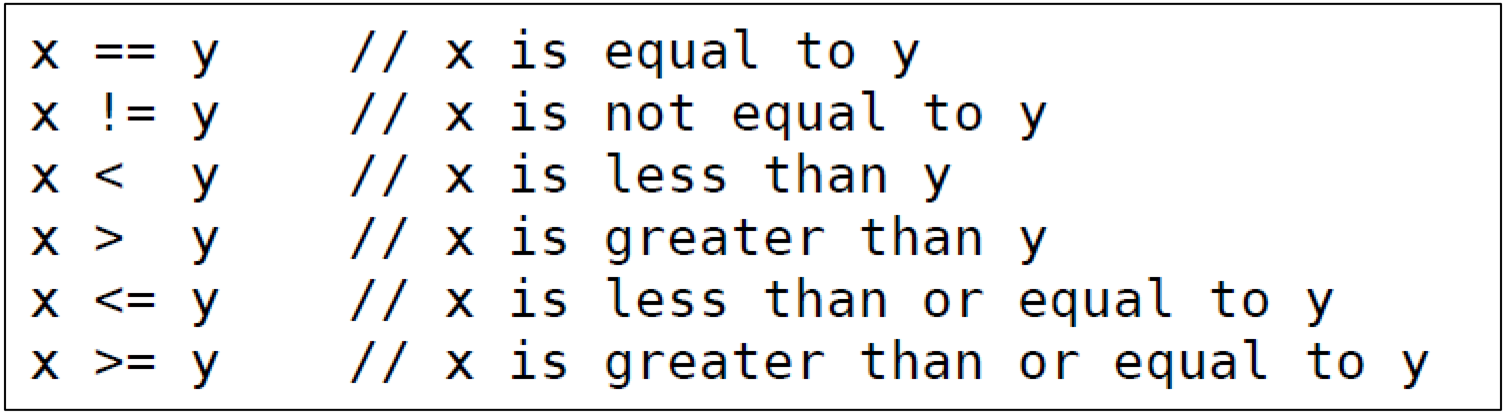
Lots of Choices

```
if (someVariable ?? value)
{
  doSomething;
}
```

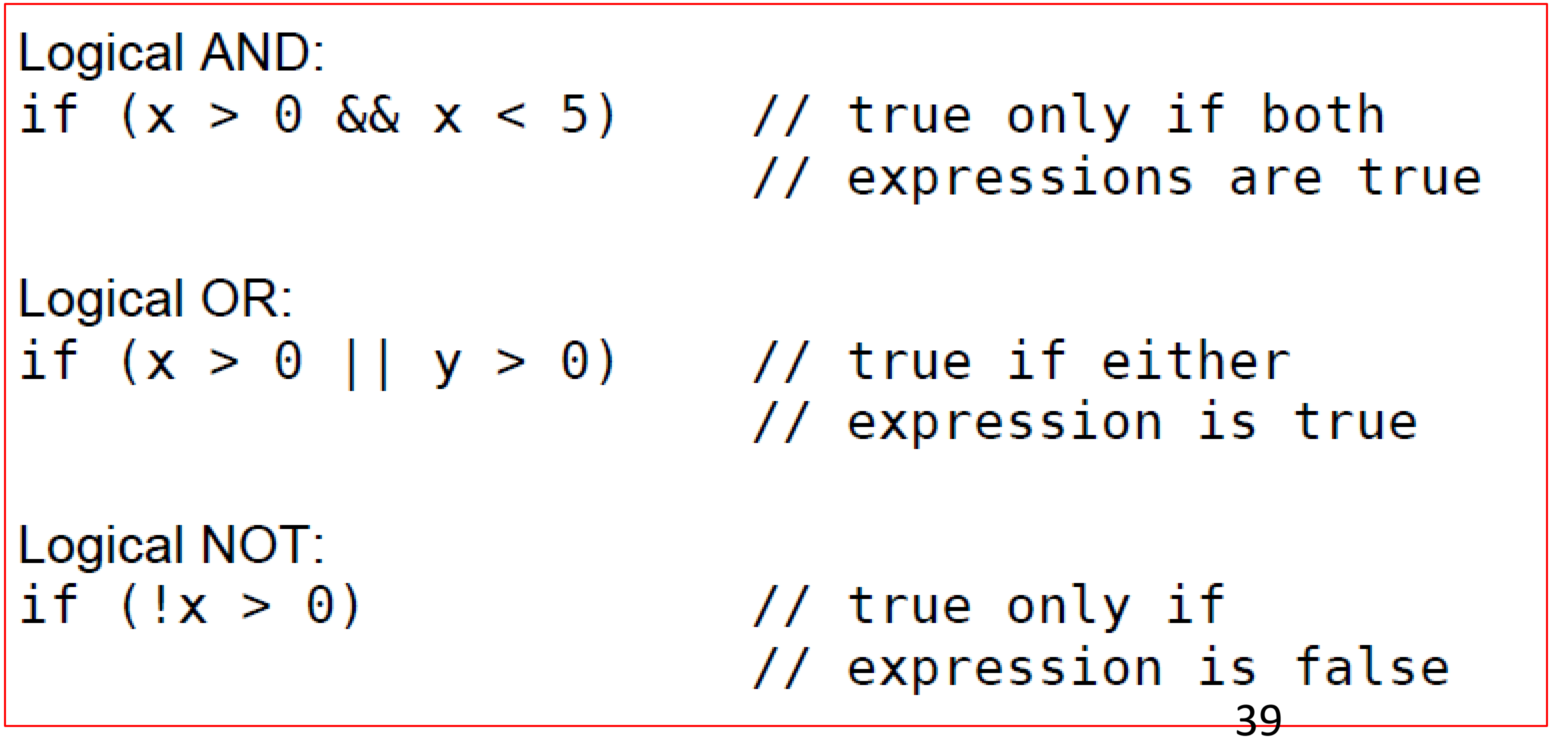
```
if (inputPin == HIGH)
{
  doThingA;
}
else
{
  doThingB;
}
```

```
if (inputPin < 500)
{
  doThingA;
}
else if (inputPin >= 1000)
{
  doThingB;
}
else
{
  doThingC;
}
```

More Boolean Expressions



```
x == y     // x is equal to y
x != y     // x is not equal to y
x <  y     // x is less than y
x >  y     // x is greater than y
x <= y     // x is less than or equal to y
x >= y     // x is greater than or equal to y
```



Logical AND:

```
if (x > 0 && x < 5)      // true only if both
                         // expressions are true
```

Logical OR:

```
if (x > 0 || y > 0)      // true if either
                         // expression is true
```

Logical NOT:

```
if (!x > 0)              // true only if
                         // expression is false
```

Boolean Expressions:
What is Truth?

In computer languages, false is 0 and true is anything except 0

- Examples of variables that are "false"
  - ❑ X=**false**; y=LOW; z=0; w=(5==8);
- Examples of variables that are "true"
  - ❑ X=**true**; y=HIGH; z=1; z=39; w=3+2*7;
  - ❑ X=anything_but_false

Arrays

- An Array is just a list of values:

```
//pin numbers for song selection LEDs
int songLED[] = {7, 8, 9}; //declare and initialize
```

- Array values are indexed. For example:

songLED[0] has value 7

songLED[1] has value 8

int song=2;

songLED[song] has value 9

- Valid index values: 0, 1, 2
- Invalid index values: -1, 3, 27, etc.

The Code: setup()

```
int song=0;                //index of song to play. Possible values are 0,1,2
int NSongs = 3;            //number of songs and LEDs
int songBut = 11;          //pin connected to song button
int songLED[] = {7, 8, 9};     //pins for song selection LEDs
int lastButton = HIGH;         //starting button value assuming not pushed
int button;

void setup() {
  pinMode(songBut, INPUT_PULLUP);
  for (int k = 0; k < NSongs; k++) {
    pinMode(songLED[k], OUTPUT);
  }
digitalWrite(songLED[song],HIGH);        //turn on LED for starting song
}
```

What does this do?

The code: loop()

```
void loop() {   //This code will repeat over and over forever
  button = digitalRead(songBut);
  if (button != lastButton) delay(40);     //wait for bounce to settle
  if (button == HIGH && lastButton == LOW) { //button released
    digitalWrite(songLED[song], LOW);   //turn off old song LED
    song = song + 1;                    //increment song variable
    if (song >= NSongs) song = 0;       //keep song between 0, 1, 2
    digitalWrite(songLED[song], HIGH);  //turn on new song LED
  }                                 //done processing button release
  lastButton = button;              //get ready for next loop
}
```

Load and Go

- Make sure you've opened: "ArduinoJukeboxButton.ino"
- Upload it and try pushing the button.
- Remarks
  - This is a LOT of information!!!
  - At first you might not be able to write code like this
  - Become comfortable with modifying existing code (e.g., Arduino Examples) for your own applications

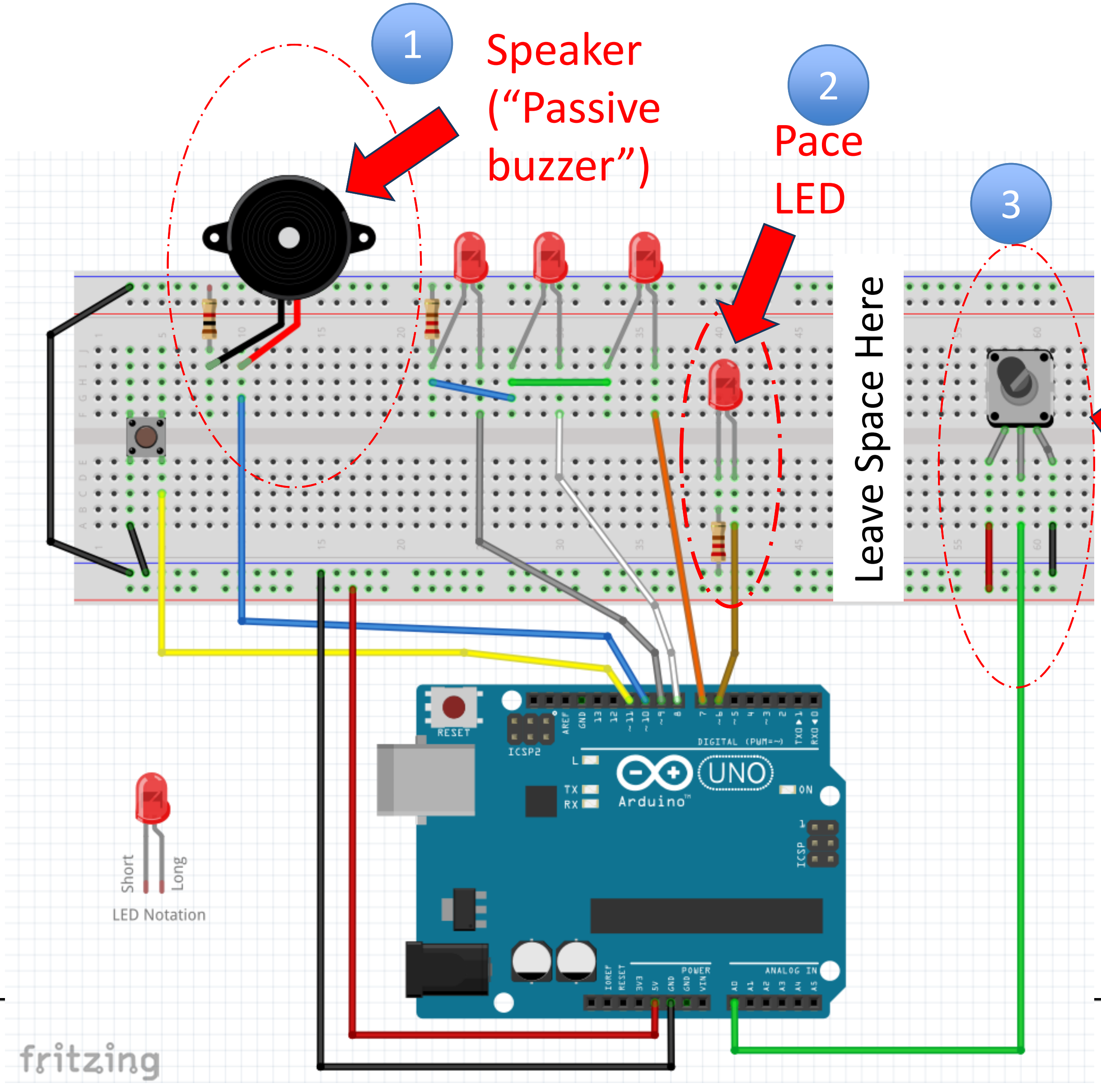
Analog Inputs

Analog-to-Digital Converters

Potentiometers ("Pots")

Serial Monitor

Next: Add the Circled Components



1 Speaker ("Passive buzzer")

2 Pace LED

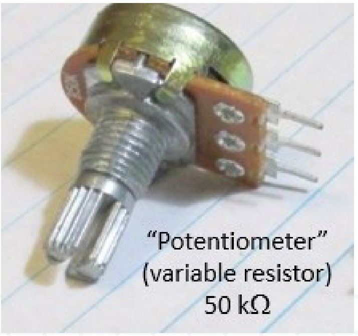
3 Pace Control (potentiometer)

Add these components to your circuit

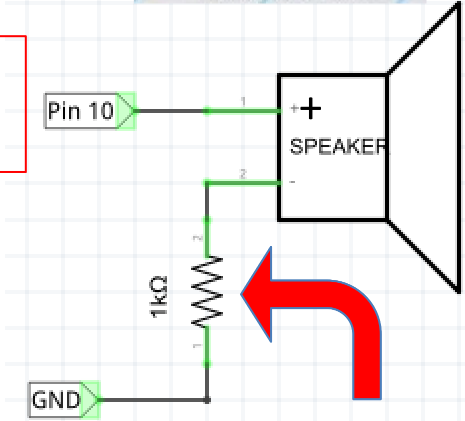
Do not remove previous parts!

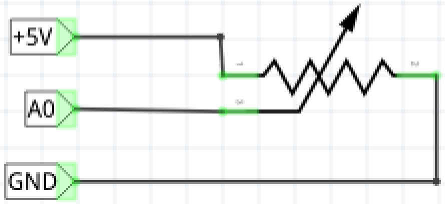
Analog Schematics

Passive Buzzer uses piezo effect to make a speaker



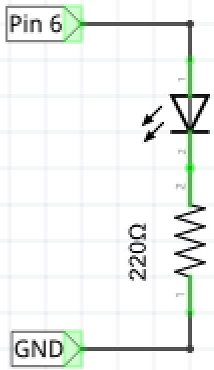
Note the '+' side!





Pace Control

Pace LED

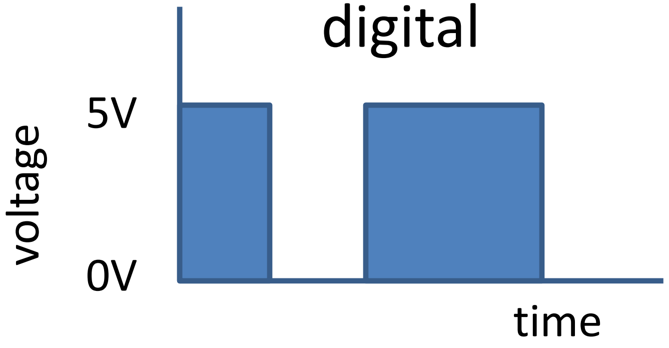


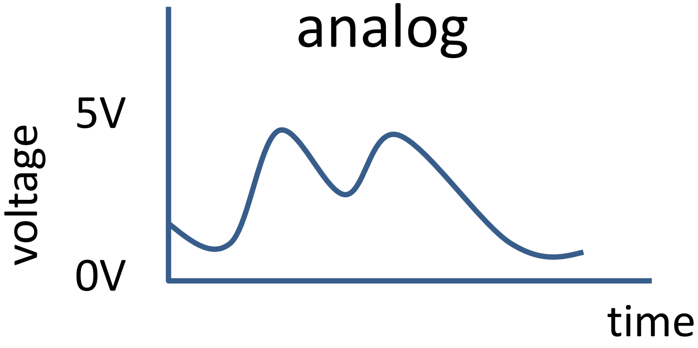
Current limiting resistor:

- Use 1000 Ohm resistor (quiet)
- Or 220 Ohm (louder)
- Or no resistor (loudest)

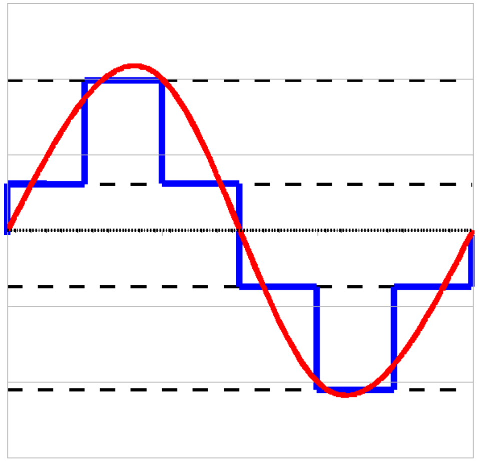
Analog Signals

- **Digital** signals are 0V (“LOW”) or +5V (“HIGH”)
- **Analog** signals vary continuously between 0 and 5V



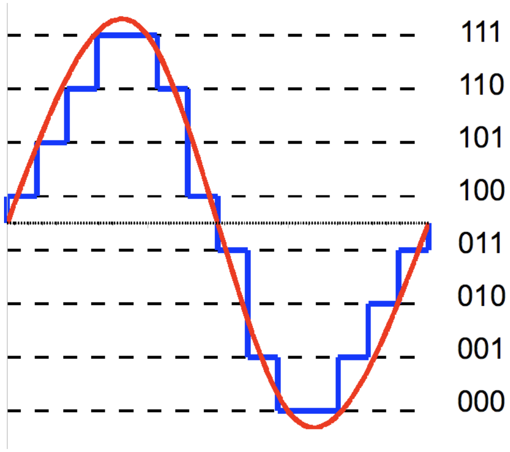


Analog-to-Digital Converter “ADC”



2-bit ADC

11	Eleven
10	Ten
01	One
00	Zero



3-bit ADC

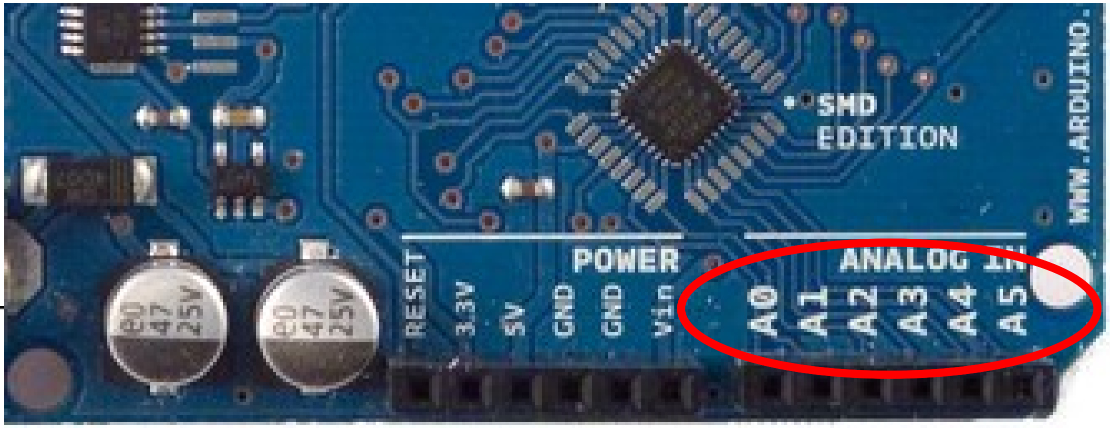
- Analog voltage measured at Arduino input pin
- Converted to Digital value used in Program

Analog Inputs

- Arduino Uno has six ADC input pins
  - A0, A1, A2, A3, A4, and A5
  - These can also be used as digital input/outputs
- Input voltages between 0 and 5 volts
- ADC value is 10 bit integer
  - There are 2^{10} = 1024 possible integer values
  - Values range from 0 to 1023

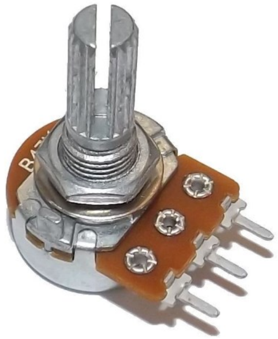
Binary Numbers

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
And so on



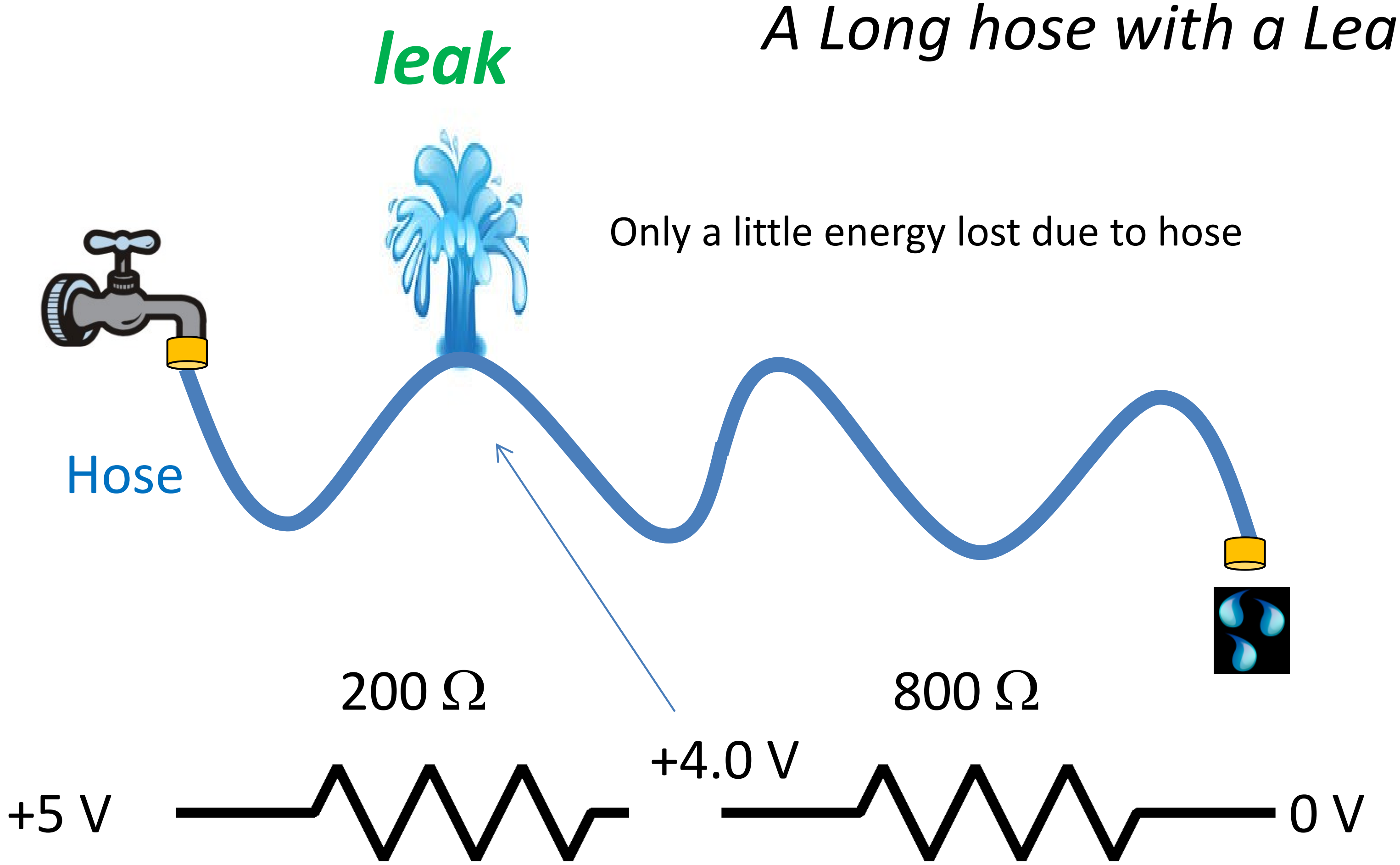
Experiments with Analog Input

- We’ll generate an analog voltage that is easily controlled using a “Potentiometer” or Pot
- Arduino ADC will measure the voltage
- Software will take action depending on value



Potentiometer

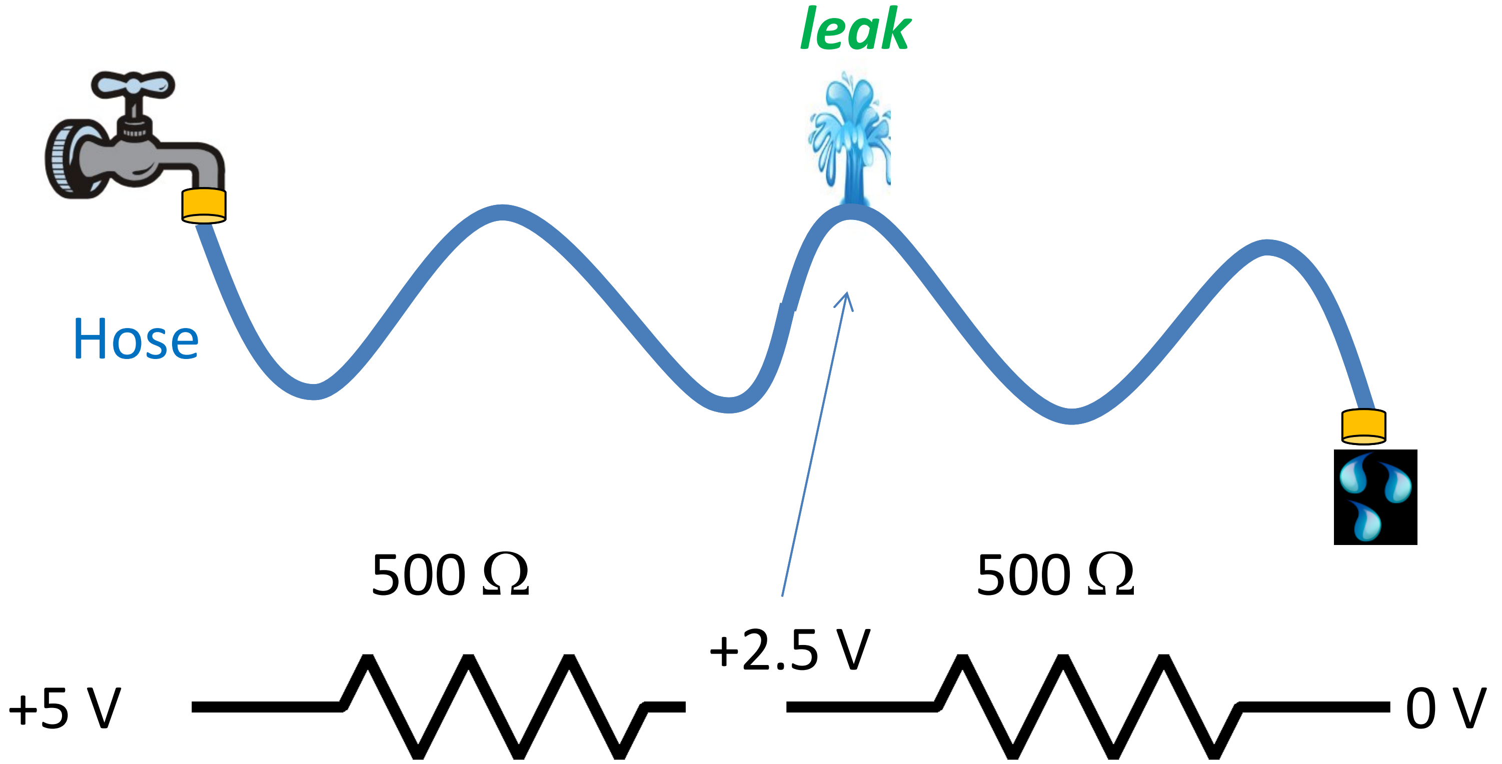
A Long hose with a Leak



Only a little energy lost due to hose

Total Resistance 200+800=1000 Ω

Same hose different Leak



leak

Hose

500 Ω

500 Ω

+5 V

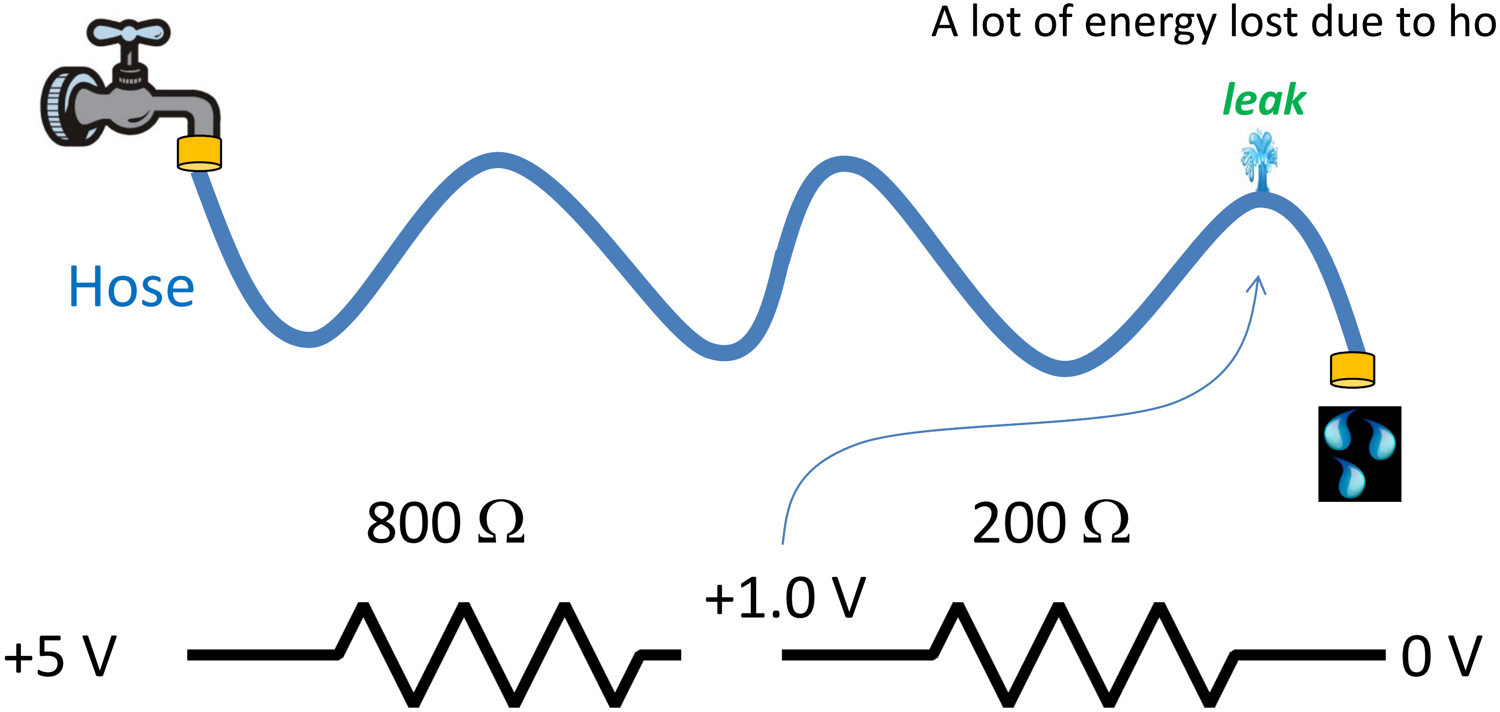
+2.5 V

0 V

Total Resistance is same: 500+500=1000 Ω

Same hose different Leak

A lot of energy lost due to hose

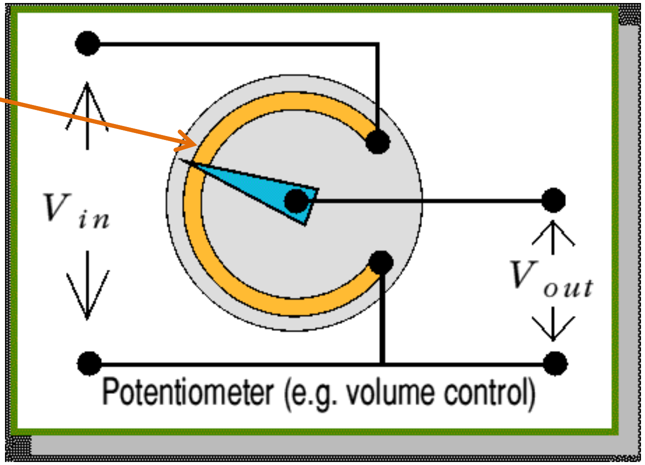


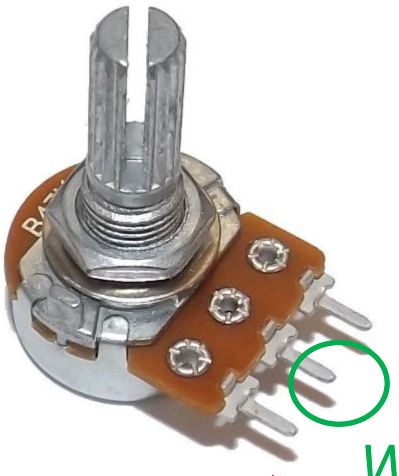
Total Resistance is same: 800+200=1000 Ω

Potentiometer or “pot

Resistive material

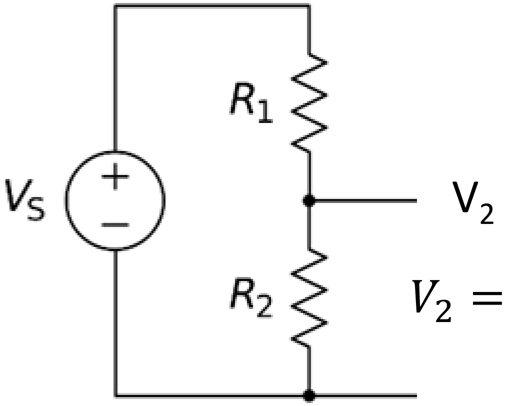
Wiper position determines R_1 and R_2





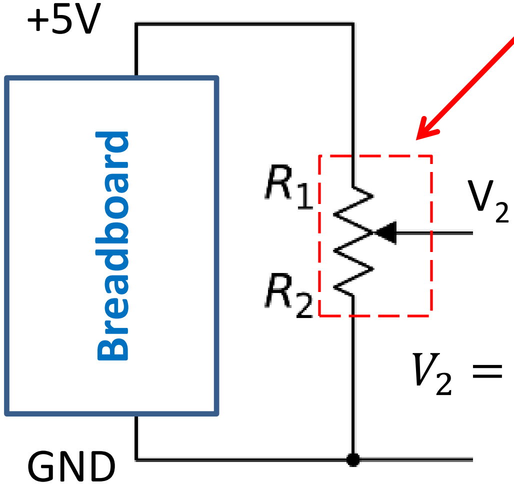
Wiper

Pot



$$V_2 = V_s \frac{R_2}{R_1 + R_2}$$

Equivalent



Wiper

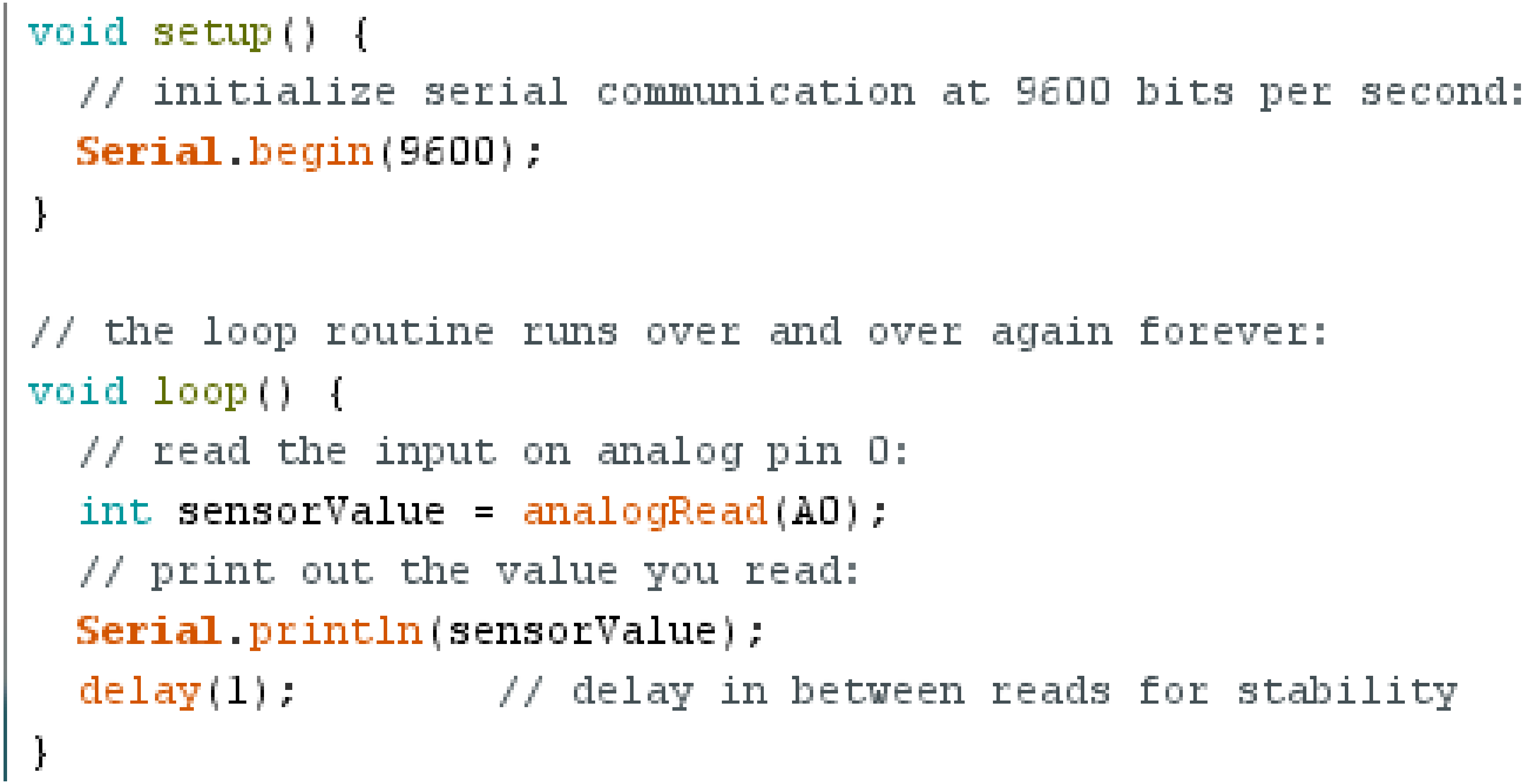
$$V_2 = 5 \frac{R_2}{R_1 + R_2}$$

Voltage Divider

Potentiometer

Experiment

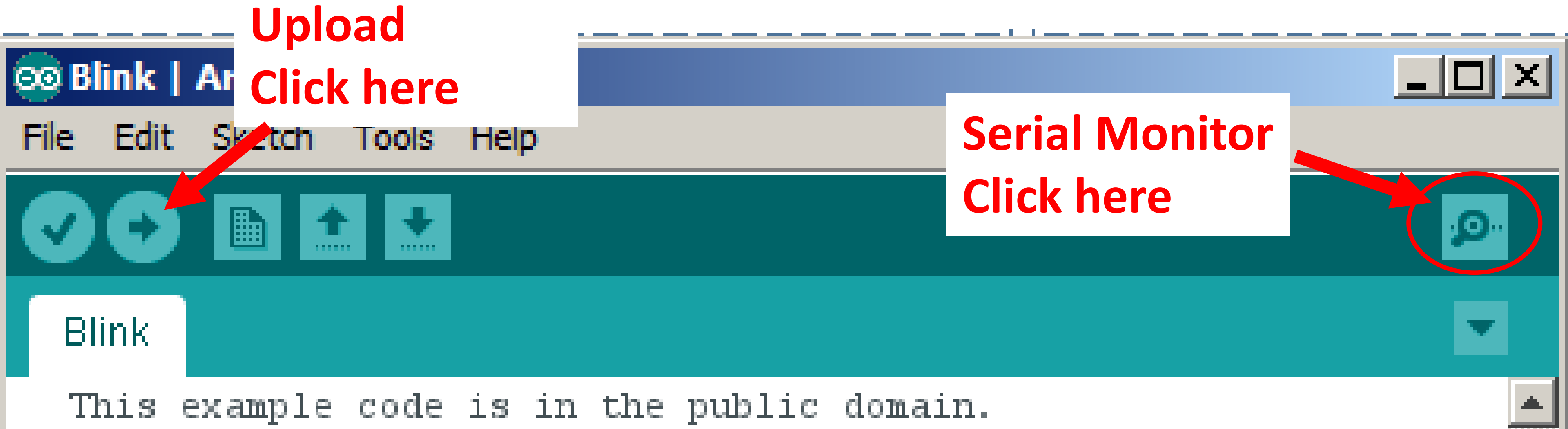
- Get the built-in Arduino sketch:
  File -> Examples -> Basics-> AnalogReadSerial



```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

Upload the Sketch

- Upload Sketch
- Turn on Serial Monitor
- Turn the potentiometer
- What are the minimum and maximum values you see?



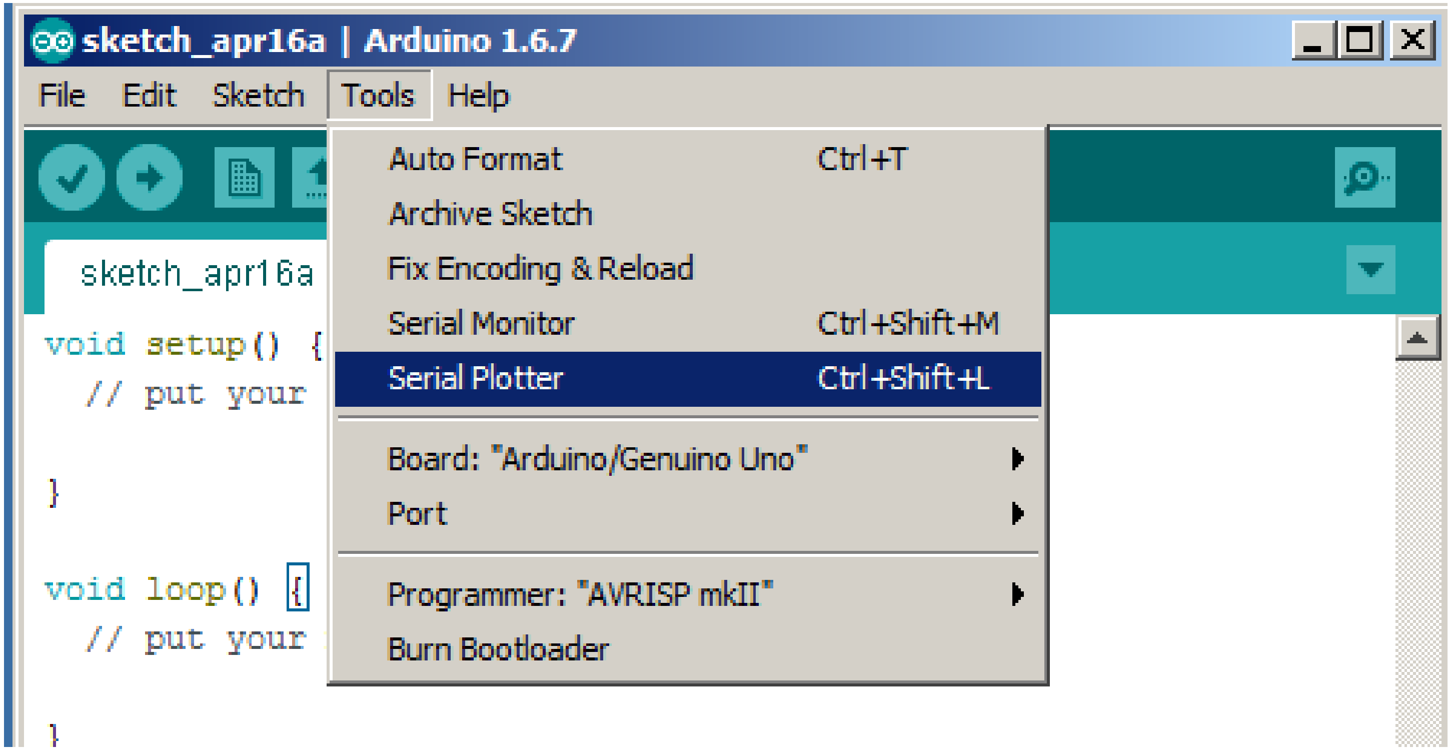
Upload
Click here

Serial Monitor
Click here

This example code is in the public domain.

Serial Plotter

- Arduino provides simple/useful data plots



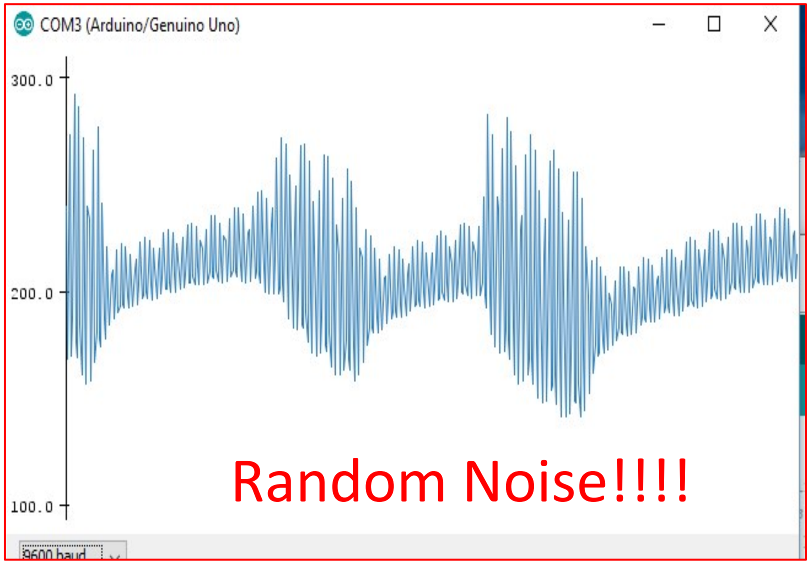
sketch_apr16a | Arduino 1.6.7
File
Edit
Sketch
Tools
Help
sketch_apr16a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
Board: "Arduino/Genuino Uno"
Port
Programmer: "AVRISP mkII"
Burn Bootloader

```
void setup() {
  // put your

}

void loop() {
  // put your

}
```

Experiment: Floating Input

- Recall – unconnected pin called a “floating input”
  - Let’s see why this is a bad thing

1. Disconnect wire to A0
2. Activate Serial Plotter
3. Move your finger on to and off of A0



Random Noise!!!!

__Side note:__

- *Could we use this to detect your finger and light an LED?*
- *Could this be useful?*

Controlling brightness using Analog Output

- Pins 3, 5, 6, 9, 10, 11 have "**analog**" output capability

Example Commands:

```
int Red=6;                    //red LED on pin 6
int brightness=64;            //not very bright
pinMode(Red, OUTPUT);   //set up pin 6 for output
analogWrite(Red, brightness);
```

Legal range: $\mathbf{0 \le \boldsymbol{brightness} \le 255}$

Details: analogWrite(6, D)

- Pin 6 output alternates between 0V and 5V
- “Duty Cycle” = % of time ON (at 5V) = D/256 x100 %
  - E.g., analogWrite(6, 64)
    Duty cycle = 64/256*100 = 25% On Time
- Called “Pulse-Width Modulation” (PWM)
- “Persistence of Vision”: *Appears* to be constantly ON

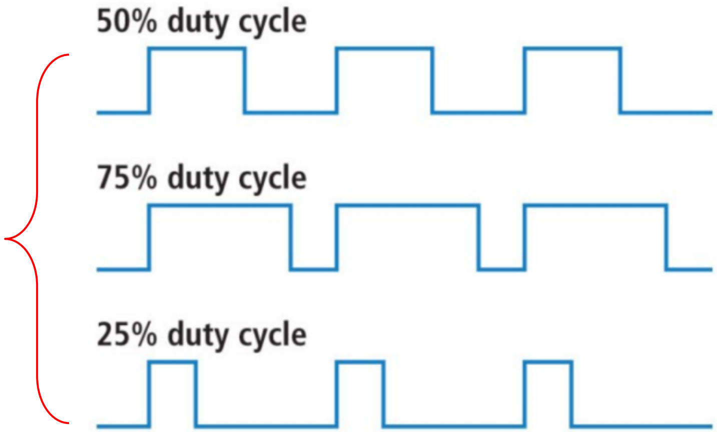
It’s not really analog!

D = 128

D = 192

D = 64

Pin 6 Output



Medium

Bright

Dim

Time ->

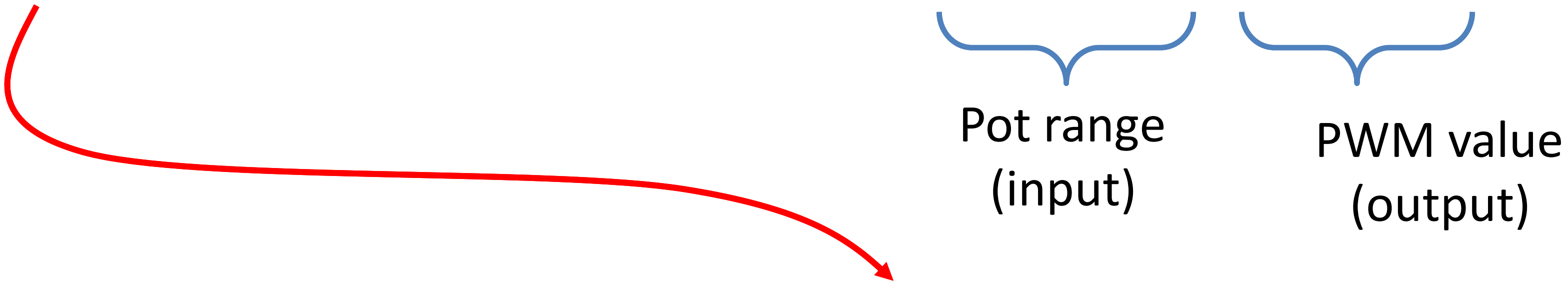
Experiment with Analog Output

- Open: **File->Examples->Analog-> AnalogInOutSerial**
- Modify the program to use Pin 6 rather than Pin 9 (where our LED is connected).
- Upload and turn on Serial Monitor
- Turn the dial on the potentiometer

- We will use this to control the “pace” of music

When you turn the knob:

- Pot values range from 0 to 1023.
- PWM value should range 0 to 255
- outputValue=**map**(sensorValue, 0, 1023, 0, 255);



Pot range (input)

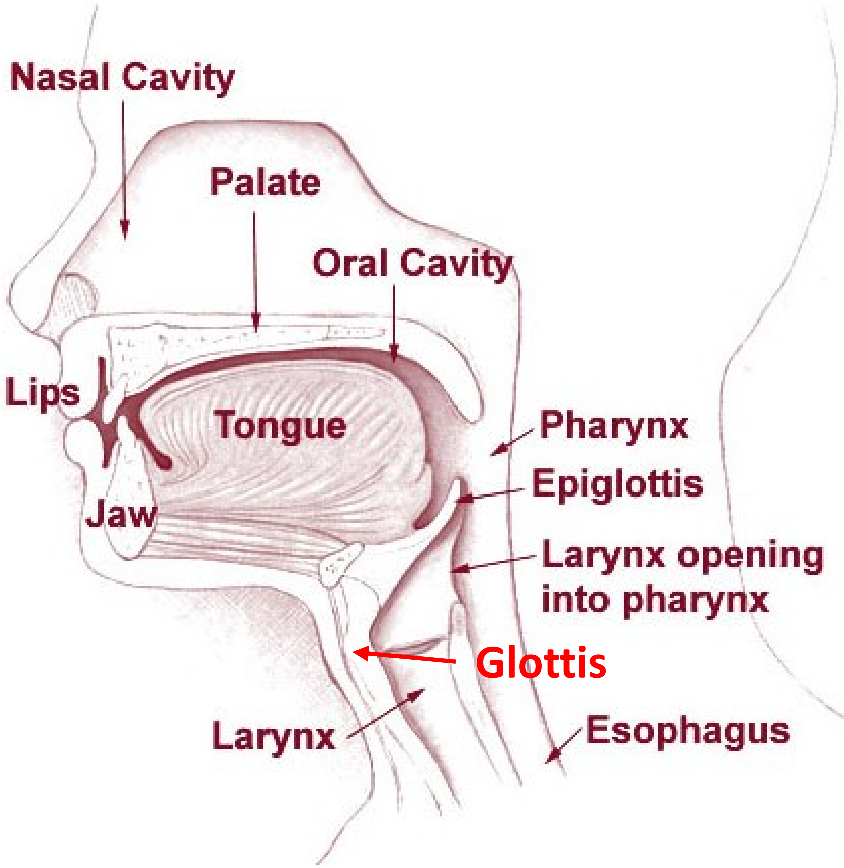
PWM value (output)

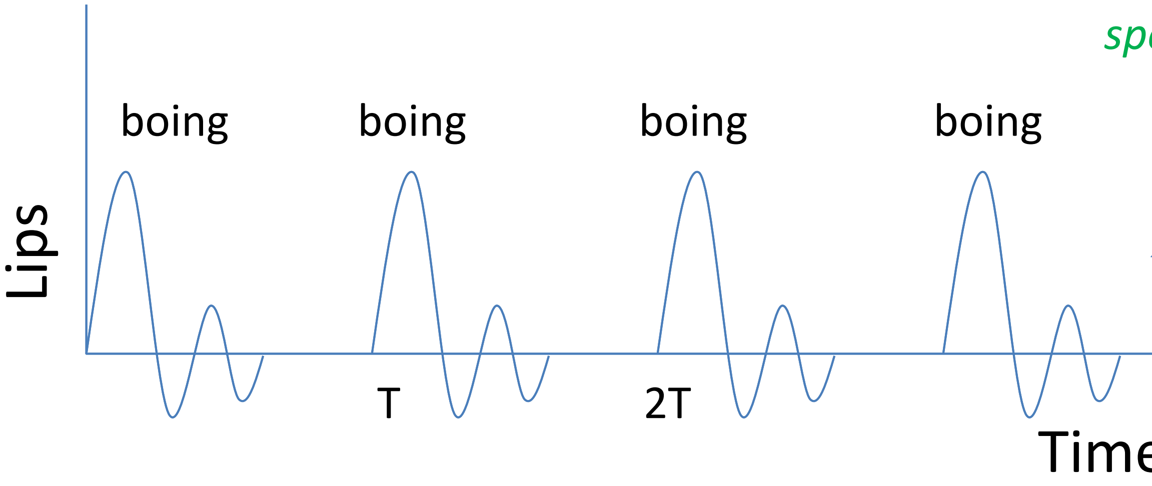
- analogWrite(analogOutPin, outputValue);
  - Changes LED brightness from dark (0) to bright (255)

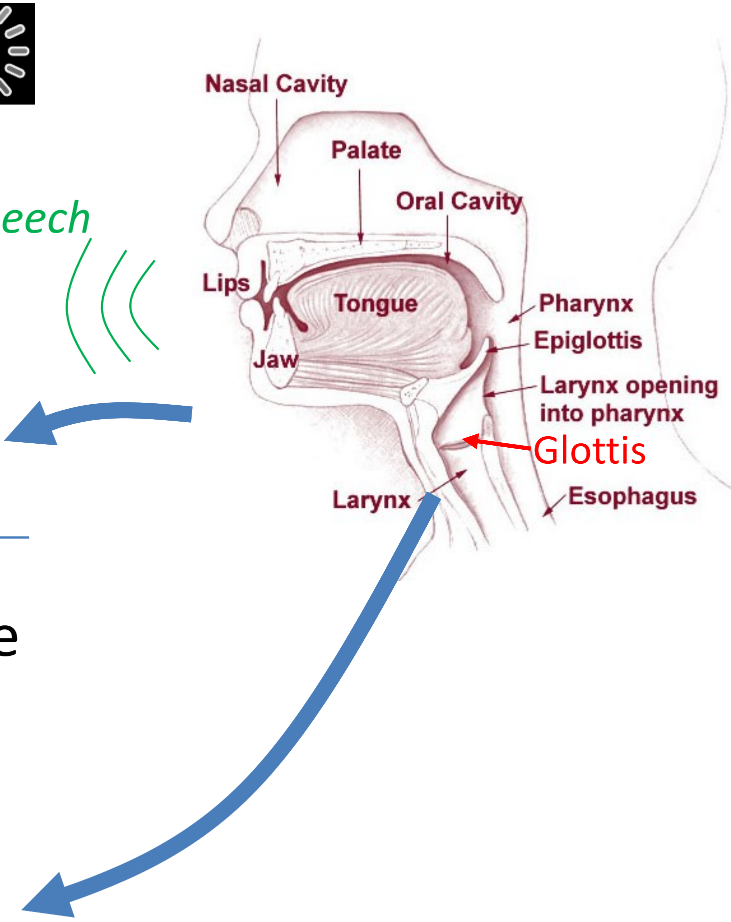
Sounds

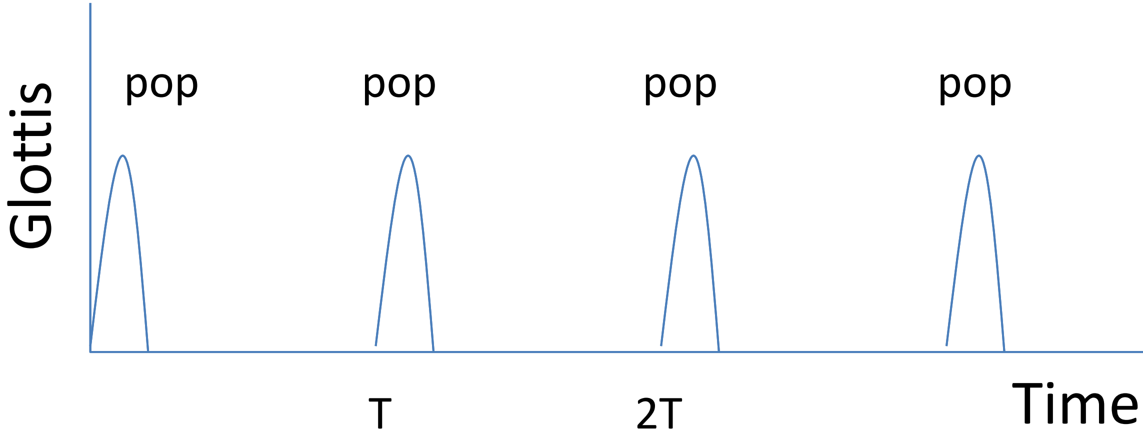
Using Arduino

Human Speech









“Pitch period” = T sec

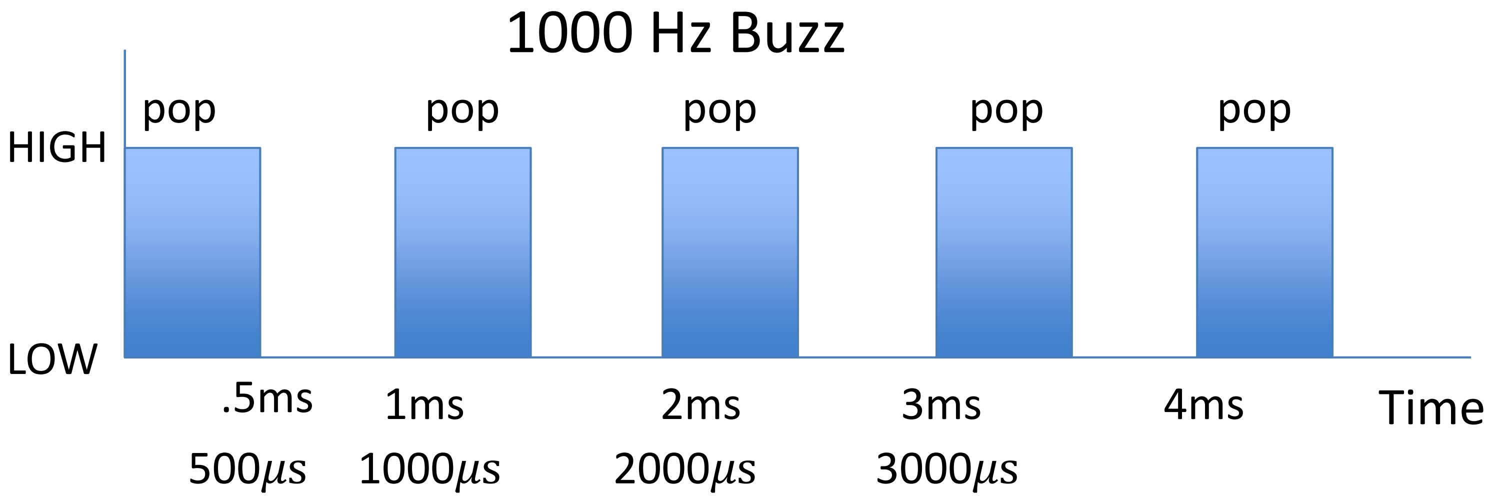
“Pitch frequency” = 1/T Hz

Pitch and Timbre

- Same story for trombones and guitars
- Pitch is the rate of pops (pulses/sec or Hertz).
  - Muscles in the larynx change the pitch
  - Say “Ahhhh” and change pitch
- Timbre is the quality or character of sound
  - Modified by tongue, mouth, nose, teeth, lips
  - Hold pitch constant and say: “Laramie”, “Wyoming”, “Meow”

Sound Generation using Arduino

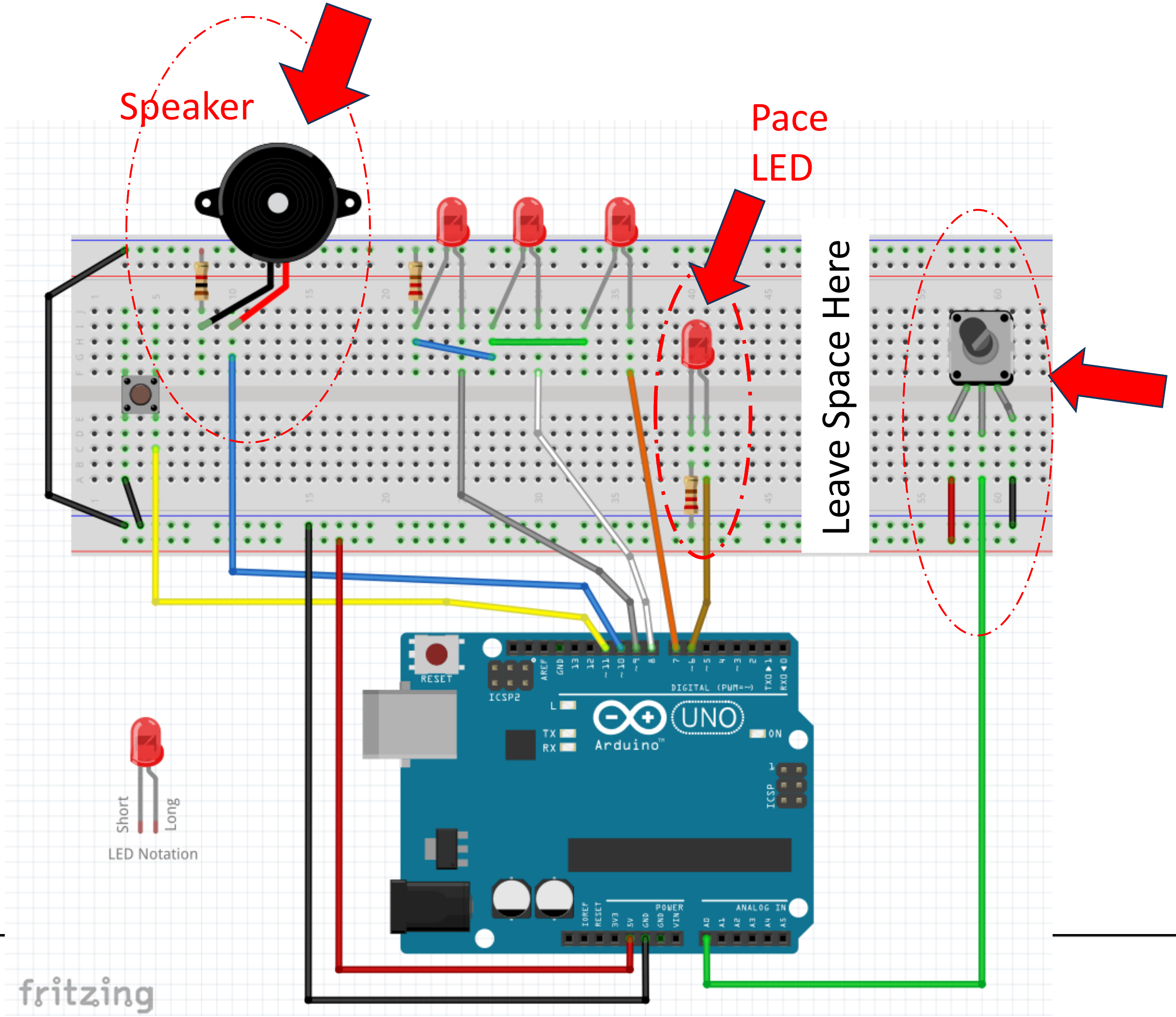
- Generate sound similar to glottis sound
  - We can vary the pitch
  - But changing timbre is “beyond our scope” ☹



How to control pitch frequency

- New Function: delayMicroseconds(d)
- Suppose we want F=1000 pulses per sec (Hz)
  - Time per pulse T = 1/1000 =.001 seconds
  - And 0.001 sec = 1 ms = 1000 µs
  - Time between pulses = T/2 = .5 ms = 500 µs
  - delay(.5) won’t work! Fraction delays not allowed.
  - delayMicroseconds(500) does work

Your Breadboard should look like this:



Buzzer Experiment

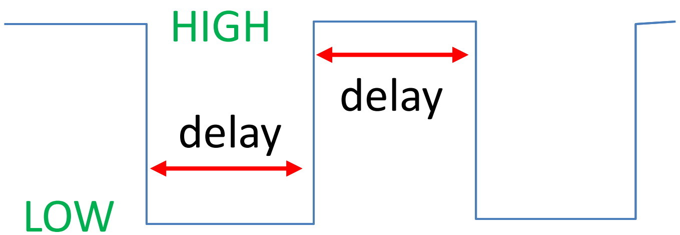
- The next program experiments with Arduino sound outputs
- Pot will control buzzer pitch frequency

Load this code: Pitcher.ino

from Workshop Arduino directory

```
int BUZpin = 10;   //passive buzzer
int POTpin=A0;   //potentiometer
int readValue;   //input potentiometer value
int delayMic;    //delay in microseconds

void setup() {
  pinMode(BUZpin, OUTPUT);
}
void loop() {
  readValue=analogRead(POTpin);   //value between 0 and 1023
  delayMic=map(readValue,0,1023,500,5000); //500<delayMic<=5000

  digitalWrite(BUZpin, HIGH);
  delayMicroseconds(delayMic);
  digitalWrite(BUZpin, LOW);
  delayMicroseconds(delayMic);
 }
```

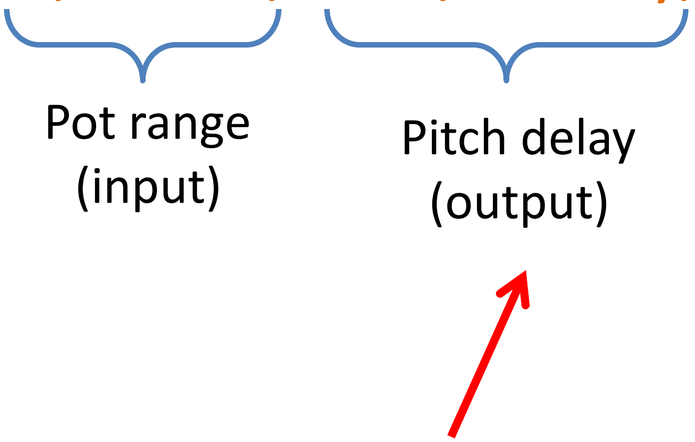
Just like the Blink Program!



Just like Blink!

Map Function

- Pot values range from 0 to 1023.
- We want delays to range 500 to 5000 μs
- delay=map(readValue, 0, 1023, 500, 5000);



Pot range (input)

Pitch delay (output)

Using calculator we get:

Delay (microsec)	Frequency=1/2D
500	1000 Hz
1000	500 Hz
2000	250 Hz
5000	100 Hz

Please play with these values and see what happens!

Play!

- Upload and play around
- Next: we figure out how to program in a tune

More Sound

Learn about **Include Files**

Learn about **#define**

Learn about **arrays**

First look at “**for**” statement

Learn about **Reset button**

Playing a Song

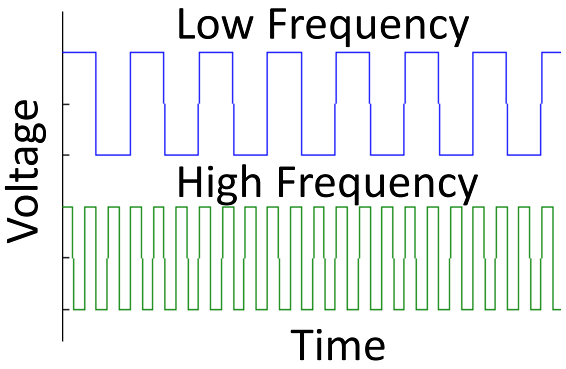
- Same Circuit using built-in function “tone”
  - Easier than generating our own square wave
- File -> Examples –> Digital -> toneMelody
  - Program assumes buzzer is on pin 8
  - Our buzzer is on pin 10.
  - Please change the program accordingly!
- Let’s briefly discuss the program

Tone Function

```
tone(pin, frequency, duration);
```

or use three commands:

```
tone(pin, frequency);
delay(duration);
noTone(pin); //turns off tone
```

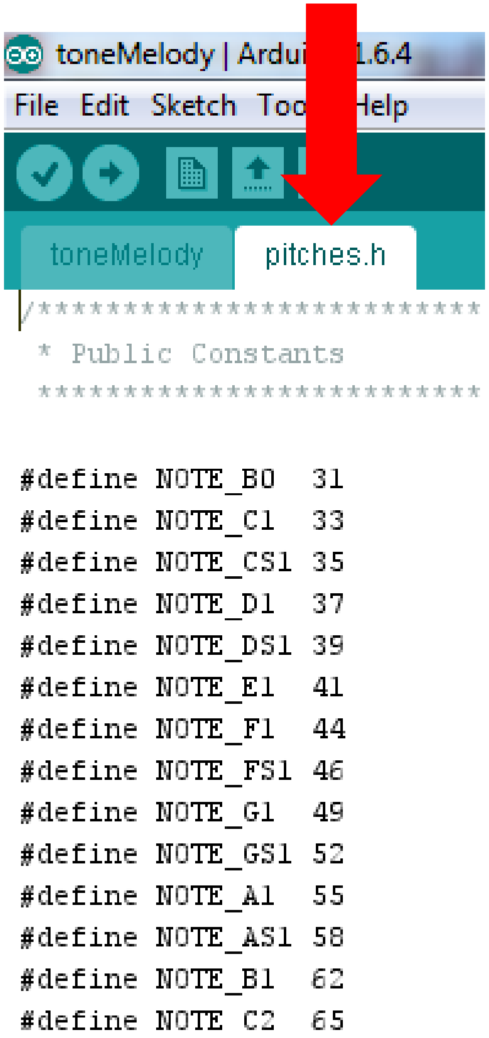


- Result – Output on pin is +5V, 0V, +5V, etc. at rate ‘frequency’ for ‘duration’ milliseconds
  - When output to a speaker (or piezo buzzer) it sounds like a tone

Include File

- include “pitches.h”
  - Combines C-code in file pitches.h into your sketch
  - Neat way to organize by keeping code separate from definitions
  - “**#define**” gives names to values
    - Example: **NOTE_C1** is now equivalent to 33

Click here

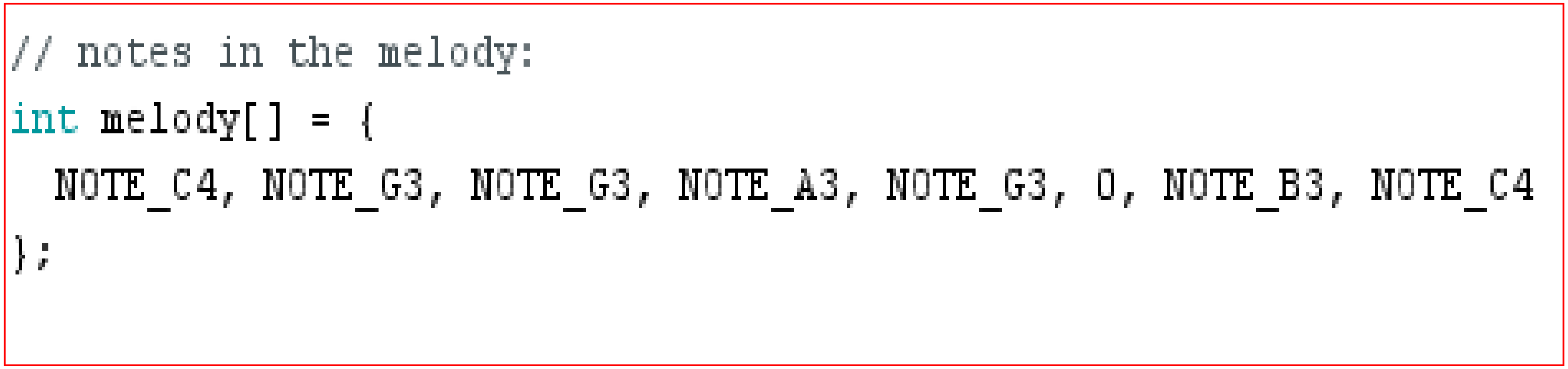


```
/*************************
 * Public Constants
 *************************


#define NOTE_B0   31
#define NOTE_C1   33
#define NOTE_CS1  35
#define NOTE_D1   37
#define NOTE_DS1  39
#define NOTE_E1   41
#define NOTE_F1   44
#define NOTE_FS1  46
#define NOTE_G1   49
#define NOTE_GS1  52
#define NOTE_A1   55
#define NOTE_AS1  58
#define NOTE_B1   62
#define NOTE C2   65
```

Etc. Etc.

Arrays (again)

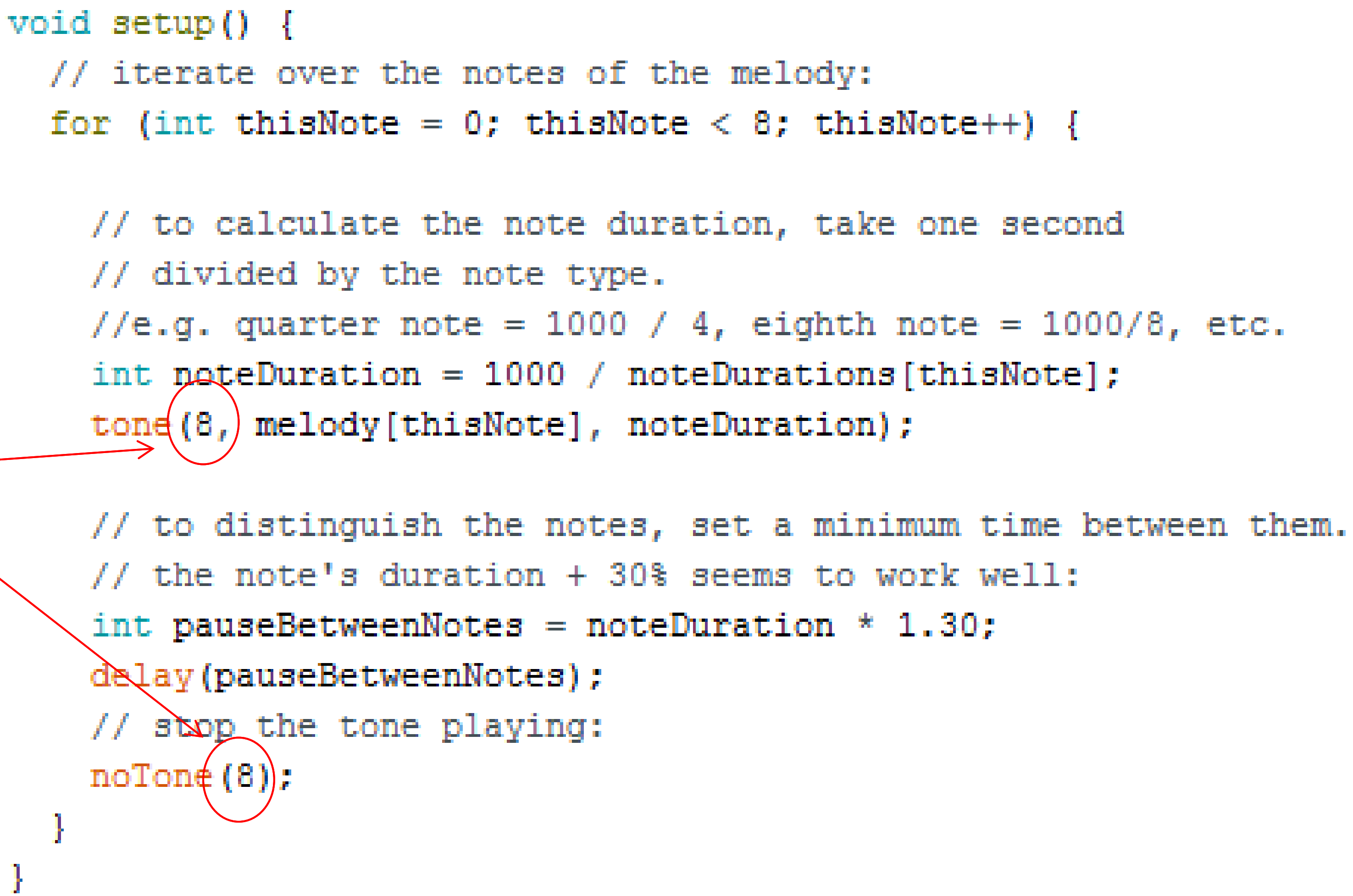
- Arrays are used to store *lists* of values
  - **Melody** stores 8 notes (pitch frequencies) of a song:



```
// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};
```

- The first note in the song is: **melody[0]=NOTE_C4**
- The second note is: **melody[1] =NOTE_G3**
- The k-th note is: **melody[k-1]**

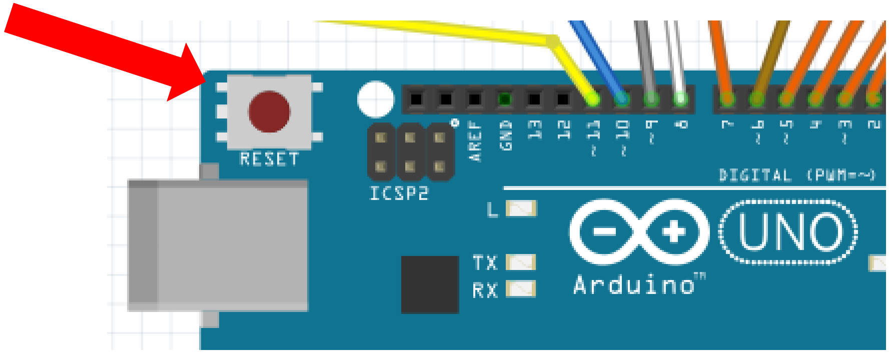
The Code



```
void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(8, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(8);
  }
}
```

change to 10

Reset button

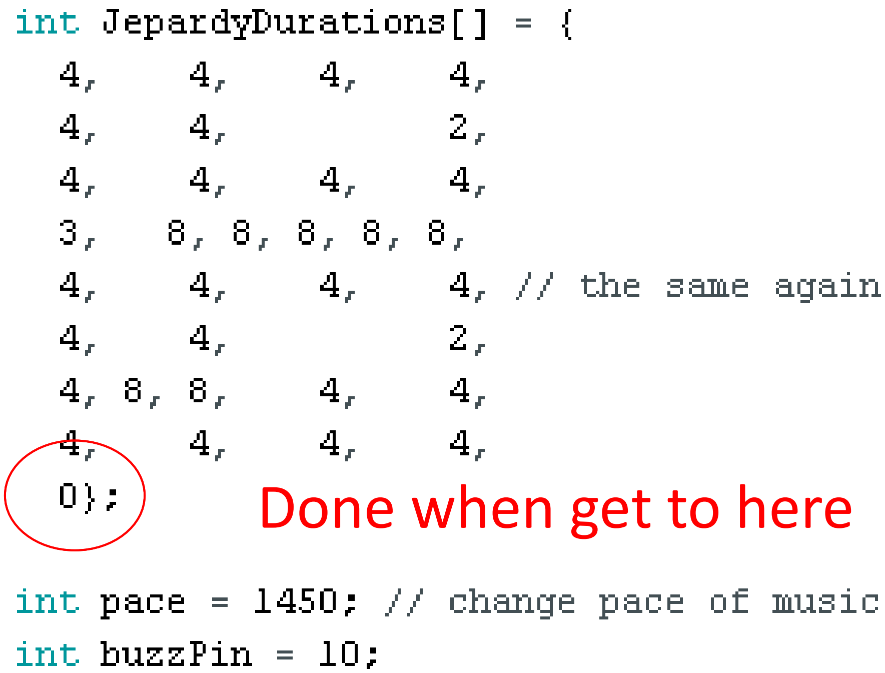


- The good stuff in this program occurs in **setup()** *(which runs only once).*
  - **loop()** function is empty!
- This means the melody will play only once.
- To replay it, push the **Reset** button
  - This causes Arduino to re-run the program.
  - You do not have to Upload the sketch to re-run it!

Our Project: Start with a single song

- Open playSong1.ino (from Workshop Arduino directory)
  - Note: **pitches.h** must be in playSong1 directory
- Melody code was moved from setup() into loop(). This way it repeats forever.
- How does program know when song over?
  - "for" loop terminates song when k-th note has zero duration, i.e., duration[k] == 0

Duration array ends with ‘0’



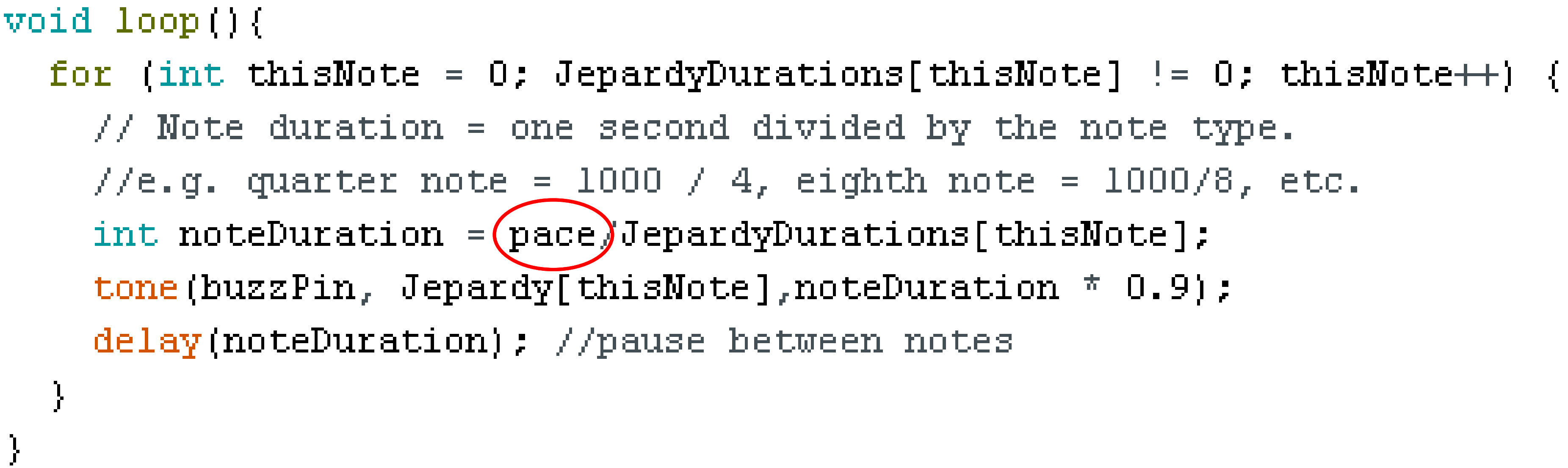
```
int JepardyDurations[] = {
  4,    4,    4,    4,
  4,    4,          2,
  4,    4,    4,    4,
  3,   8, 8, 8, 8, 8,
  4,    4,    4,    4, // the same again
  4,    4,          2,
  4, 8, 8,    4,    4,
  4,    4,    4,    4,
  0};

int pace = 1450; // change pace of music
int buzzPin = 10;
```

Done when get to here

“Not Equal”

```
for (int thisNote = 0; JepardyDurations[thisNote] != 0; thisNote++) {
  statements to play notes…
}
```

Code for playSong1



```
void loop(){
  for (int thisNote = 0; JepardyDurations[thisNote] != 0; thisNote++) {
    // Note duration = one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = pace/JepardyDurations[thisNote];
    tone(buzzPin, Jepardy[thisNote],noteDuration * 0.9);
    delay(noteDuration); //pause between notes
  }
}
```

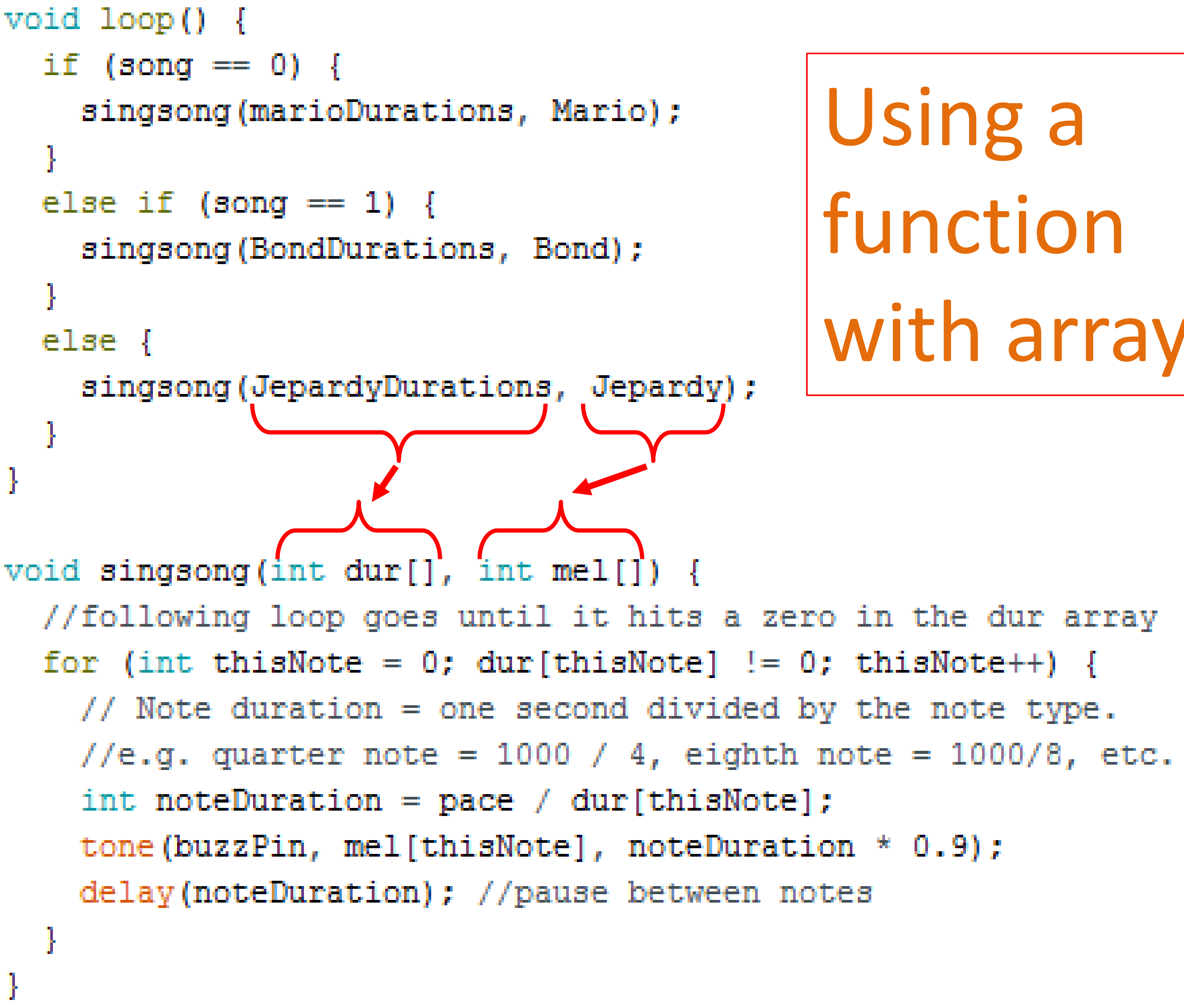
Note: pace = duration of “whole note”

- Usually pace=1000 ms (one second)
  - – E.g., quarter note is 1000/4 milliseconds long
- When pace > 1000, song is slower

Our Project: we want three songs

- Now load playSong3.ino from workshop Arduino directory
- We'll use a function to play a song
- Investigate code
- Upload and see if it works
  - Try different values for 'song'

Using a function with arrays



```
void loop() {
  if (song == 0) {
    singsong(marioDurations, Mario);
  }
  else if (song == 1) {
    singsong(BondDurations, Bond);
  }
  else {
    singsong(JepardyDurations, Jepardy);
  }
}

void singsong(int dur[], int mel[]) {
  //following loop goes until it hits a zero in the dur array
  for (int thisNote = 0; dur[thisNote] != 0; thisNote++) {
    // Note duration = one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = pace / dur[thisNote];
    tone(buzzPin, mel[thisNote], noteDuration * 0.9);
    delay(noteDuration); //pause between notes
  }
}
```

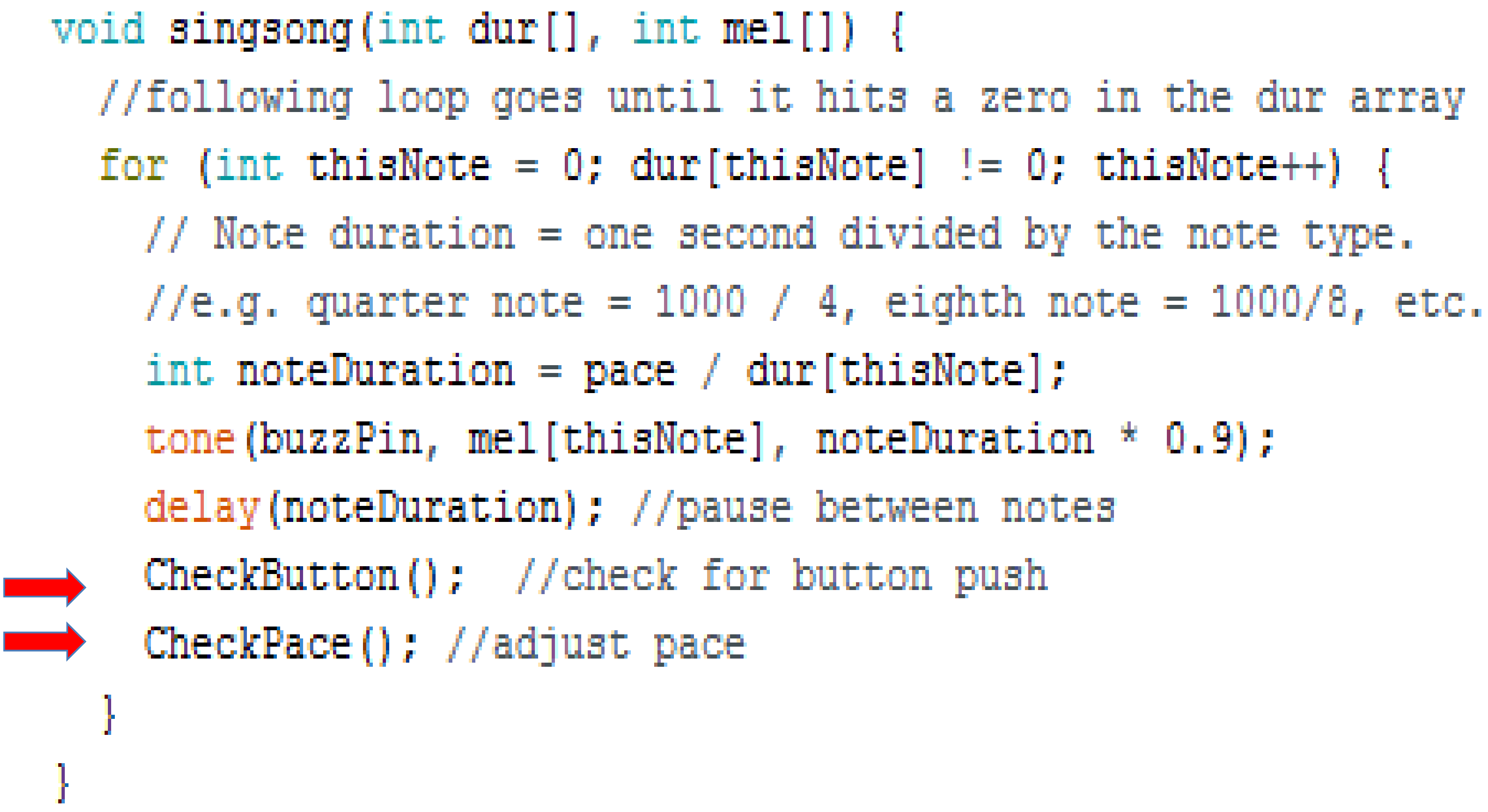
Finally: include pace and song selection

- Now load playSongButton.ino

  in Workshop Arduino directory
- **Reuses** code to check pushbutton

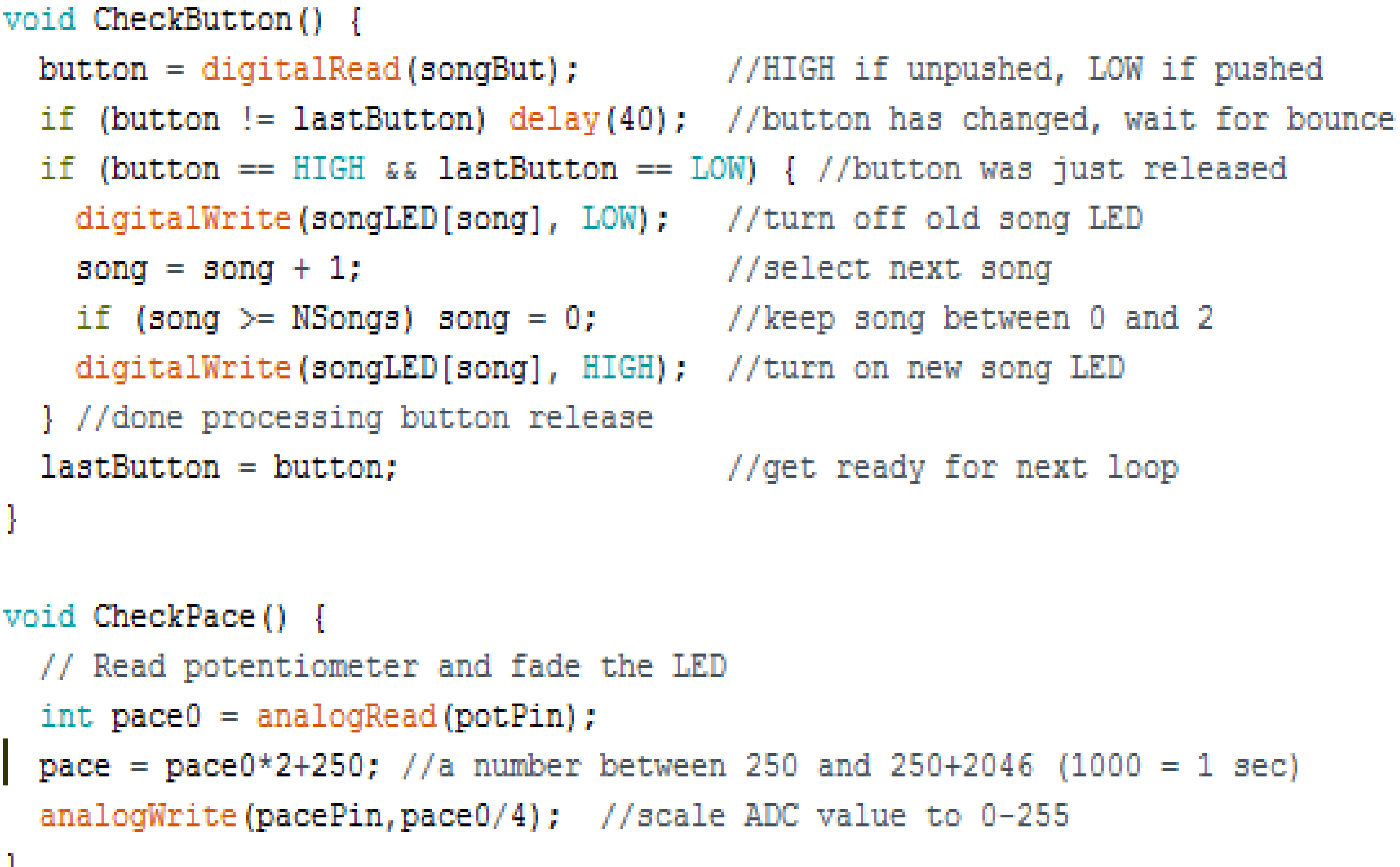
- Then load playSongPace.ino
- **Reuses** code to adjust pace
- Upload and see if it works

Modified singsong function



```
void singsong(int dur[], int mel[]) {
  //following loop goes until it hits a zero in the dur array
  for (int thisNote = 0; dur[thisNote] != 0; thisNote++) {
    // Note duration = one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = pace / dur[thisNote];
    tone(buzzPin, mel[thisNote], noteDuration * 0.9);
    delay(noteDuration); //pause between notes
    CheckButton();  //check for button push
    CheckPace(); //adjust pace
  }
}
```

CheckButton() and CheckPace()



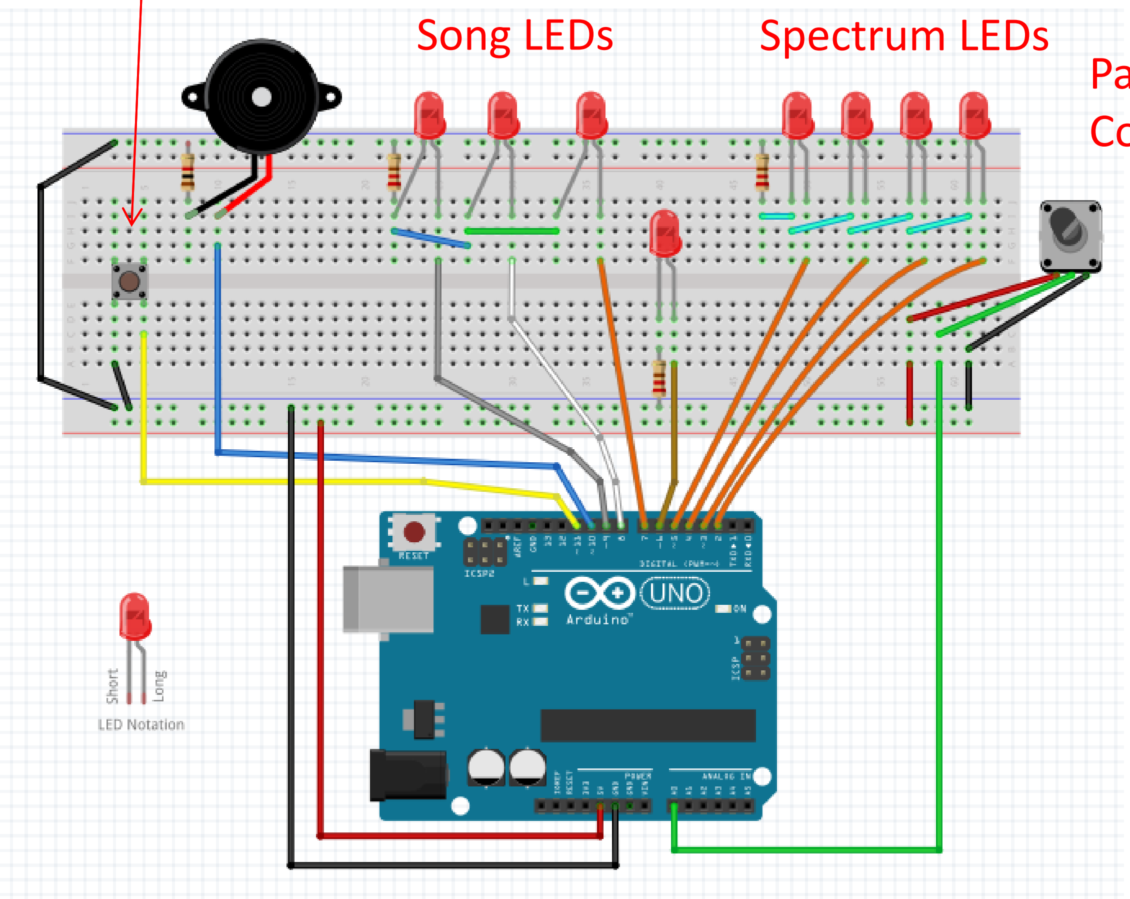
```
void CheckButton() {
  button = digitalRead(songBut);        //HIGH if unpushed, LOW if pushed
  if (button != lastButton) delay(40);  //button has changed, wait for bounce
  if (button == HIGH && lastButton == LOW) { //button was just released
    digitalWrite(songLED[song], LOW);   //turn off old song LED
    song = song + 1;                    //select next song
    if (song >= NSongs) song = 0;       //keep song between 0 and 2
    digitalWrite(songLED[song], HIGH);  //turn on new song LED
  } //done processing button release
  lastButton = button;                  //get ready for next loop
}

void CheckPace() {
  // Read potentiometer and fade the LED
  int pace0 = analogRead(potPin);
  pace = pace0*2+250; //a number between 250 and 250+2046 (1000 = 1 sec)
  analogWrite(pacePin,pace0/4);  //scale ADC value to 0-255
}
```

Final Project: Add A Spectrum Display

(if you have time)

- Add 4 LEDs and resistors
- Modify program to do following:
  - Light LED1 if frequency tone ≤ 100 Hz
  - Light LED2 if frequency 100 < tone ≤ 150 Hz
  - Light LED3 if frequency 150 < tone ≤ 200 Hz
  - Light LED4 if frequency 200 < tone Hz
  - *Hint: write a function similar to CheckPace() or CheckButton() to light the proper LED*
  - *Hint: The final circuit was shownearlier*
  - *Hint: my C-code is **playSongSpectrum.ino***

Final System



Song Select button

Speaker

Song LEDs

Spectrum LEDs

Pace Control

- ~~We'll build and test it section by section~~
- We have built and tested this **section by section!**