Arduino Programming

Chonnam National University
School of Electronics and Computer Engineering

Kyungbaek Kim

Arduino Board

- “Strong Friend” Created in Ivrea, Italy in 2005 by Massimo Banzi & David Cuartielles
- Open Source Hardware
- ATMEL Processor
- Coding is accessible and Transferrable
  → C++, Processing, java

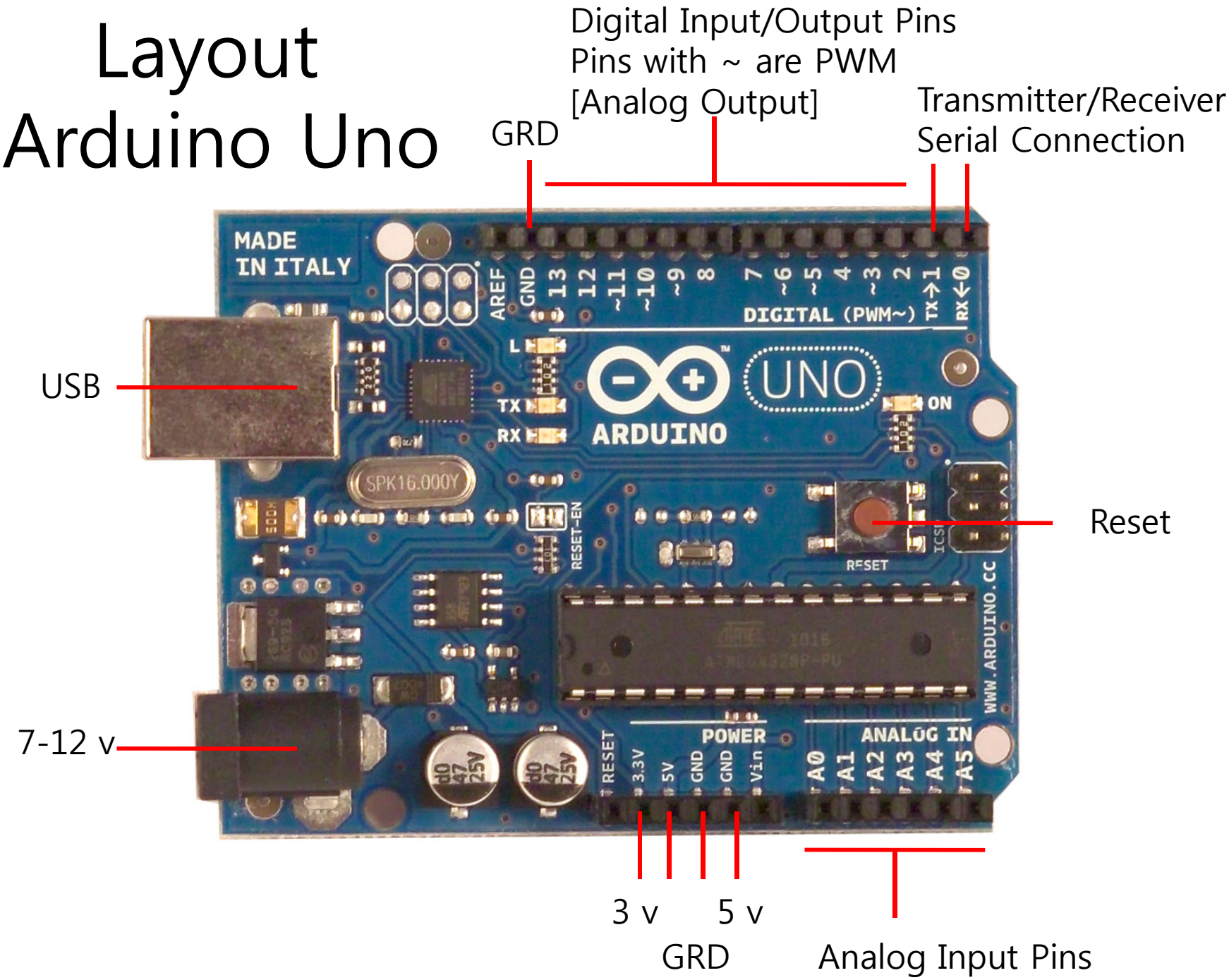
Arduino

- The go-to gear for artists, hobbyists, students, and anyone with a gadgetry dream.
- Arduino rose out another formigable challenge:
  - How to teach students to create electronics, fast
  - http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

Getting started

- SW Installation
  - Arduino, v.1.0+
  - Drivers
- Materials
  - Arduino I/O
  - Digital I/O
  - Serial cables

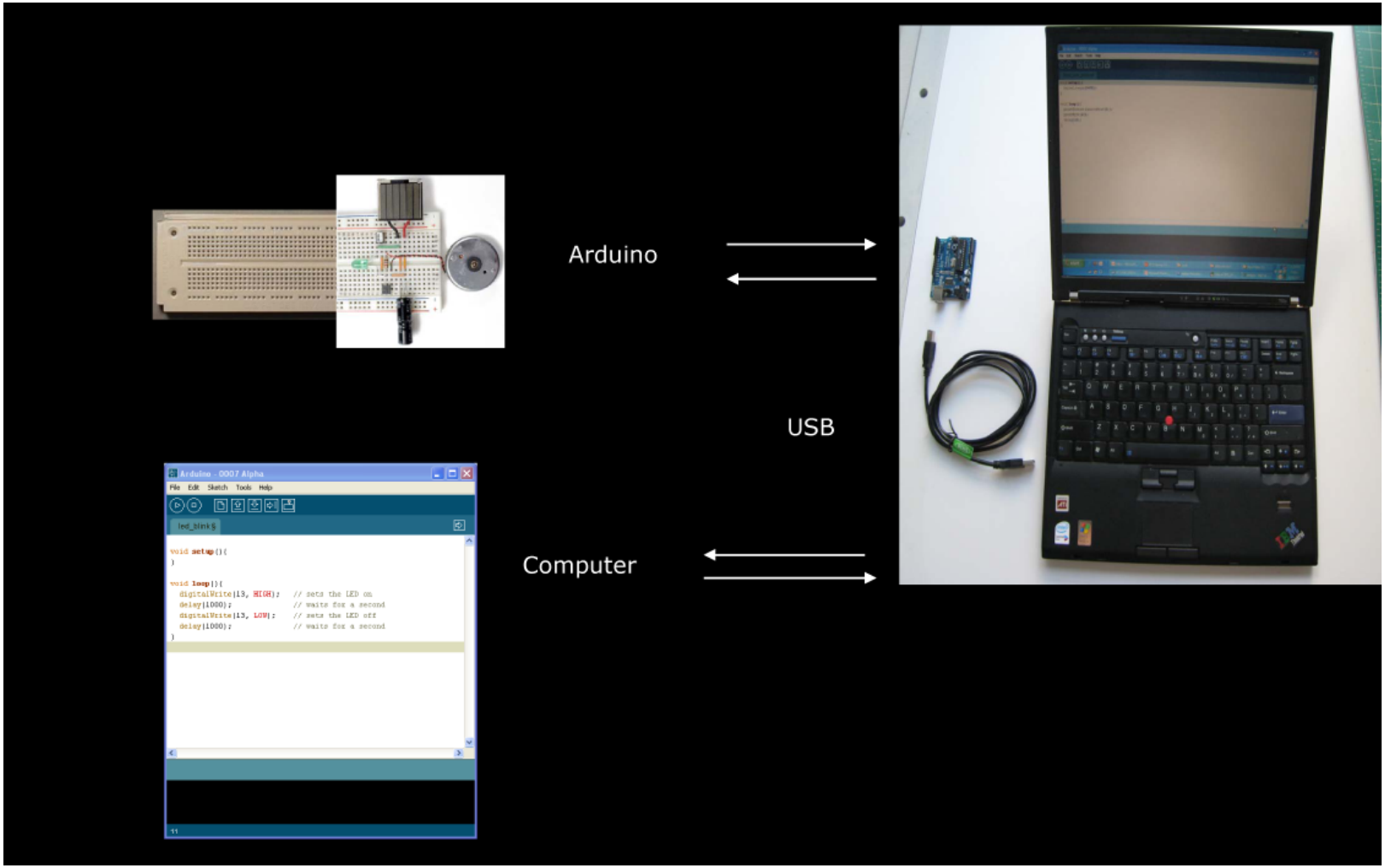
Layout
-Arduino Uno



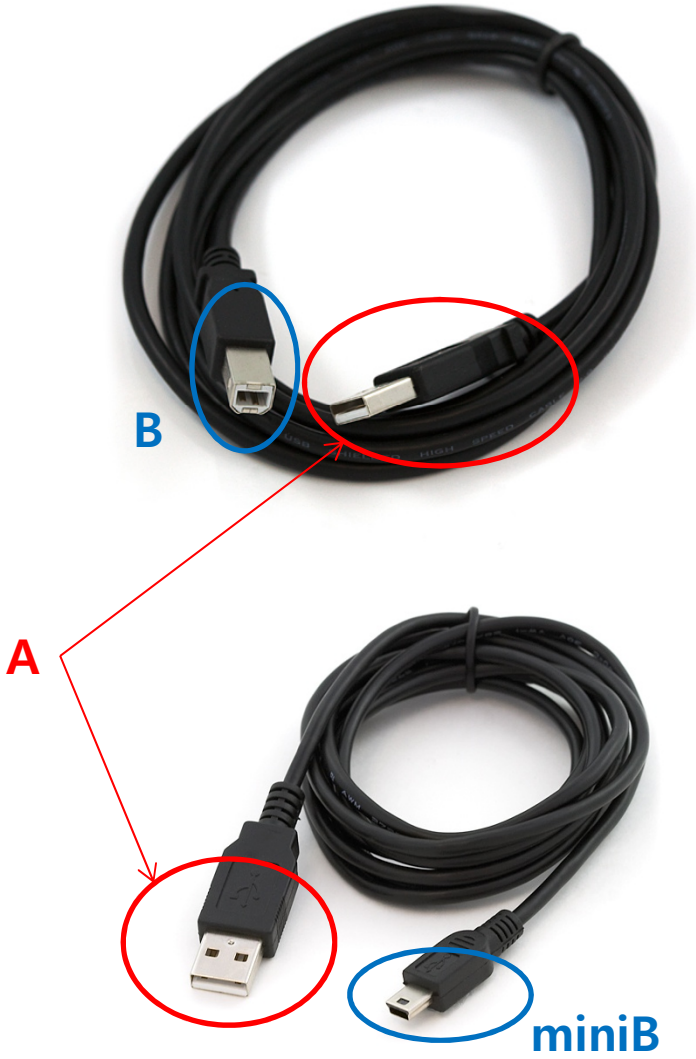
Other specifications

- Microcontroller ATmega328
- Operating Voltage 5V
- Input Voltage (recommended)7-12V
- Input Voltage (limits)6-20V
- Digital I/O Pins 14
- (of which 6 provide PWM output)
- Analog Input Pins 6
- DC Current per I/O Pin 40 mA
- DC Current for 3.3V Pin 50 mA

Setting up Host/Guest system



Connection



USB Cable A to B

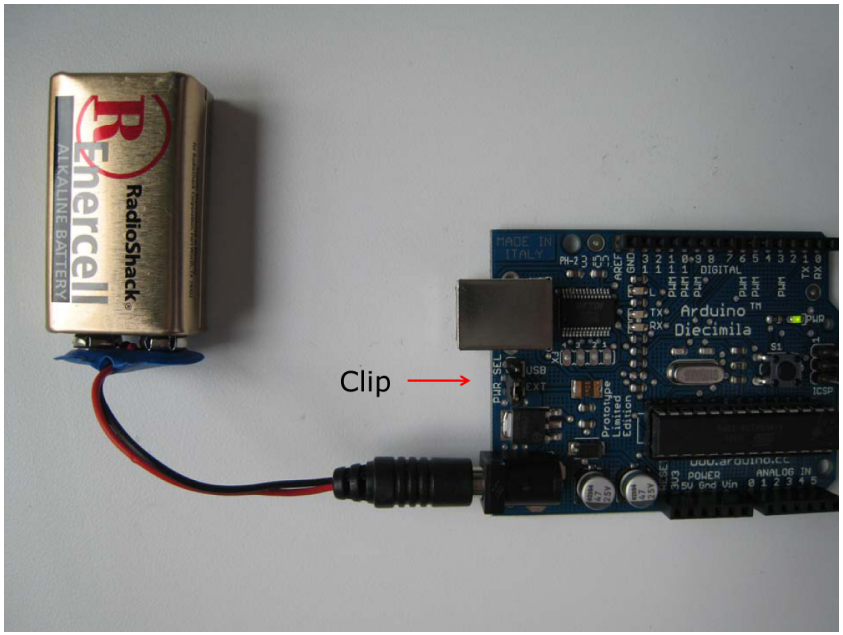
This USB Cable type is the one that allows for connecting normal Arduino Boards to the computer. They come in black and white and in various lengths.

USB Cable A to miniB

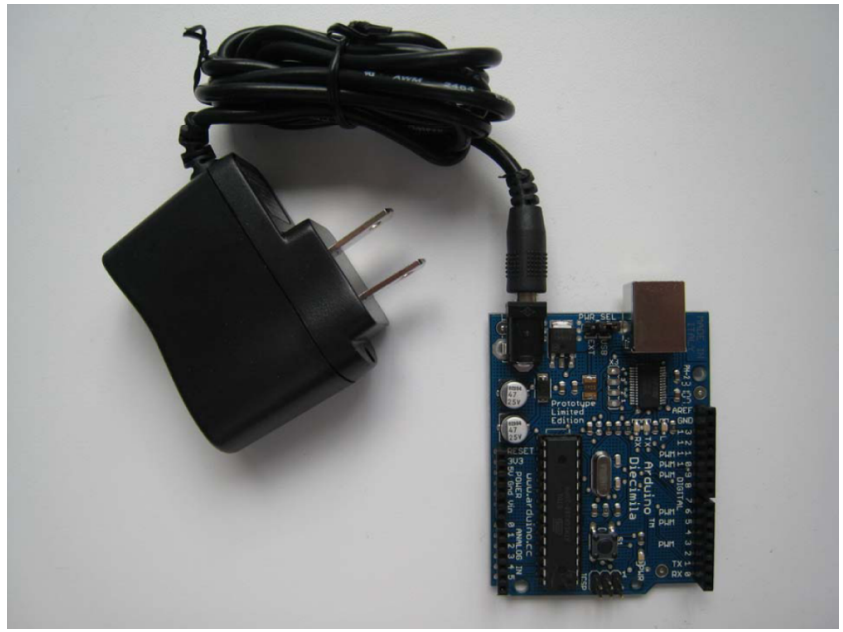
For Arduino Mini Pro and Lilypad you need USB miniB for connecting to computer.

Power options

Powered by Battery



Powered by Wall-plug



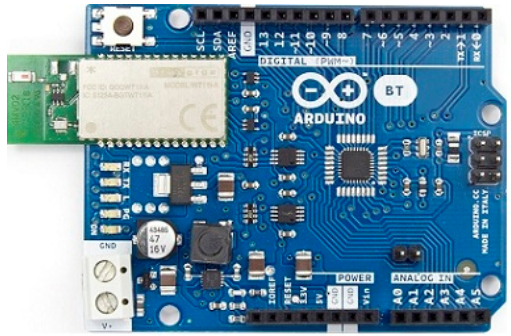
Other Choices of Arduino

- **Arduino Nano**
  - A surface mount breadboard embedded version with integrated USB. It is a smallest, complete, and breadboard friendly. It has everything that Diecimila has (electrically) with more analog input pins and onboard +5V AREF jumper.
- **Arduino BT**
  - An Arduino board with built-in bluetooth module, allowing for wireless communication.
- **Arduino MEGA ADK**
  - An Arduino board with USB host interface to connect with Android based phones.
- **LilyPad Arduino**
  - A microcontroller board designed for wearables and e-textiles. It can be sewn to fabric and similarly mounted power supplies, sensors and actuators with conductive thread.

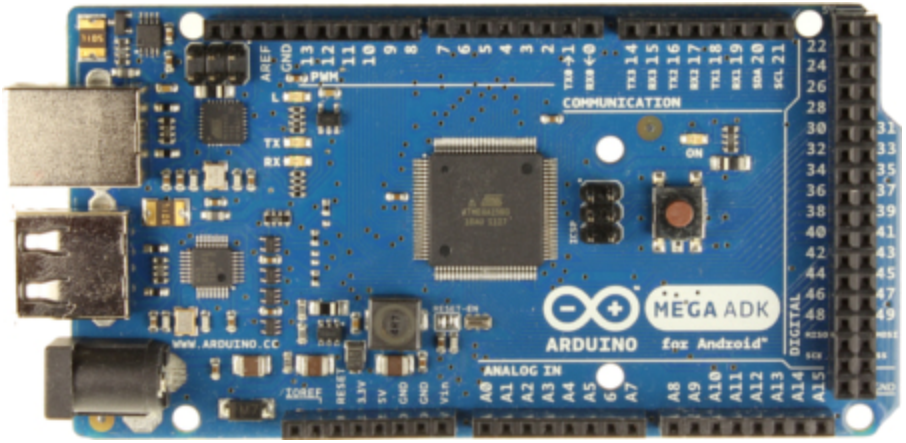
Other Choices of Arduino

Arduino Nano

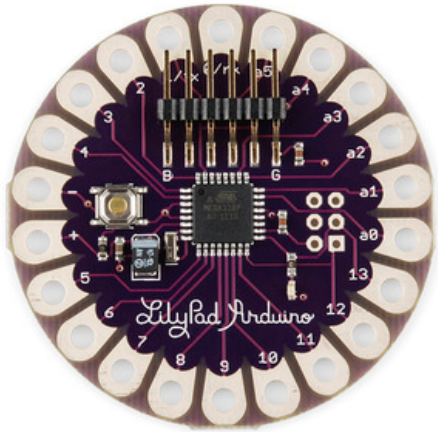
Arduino BT



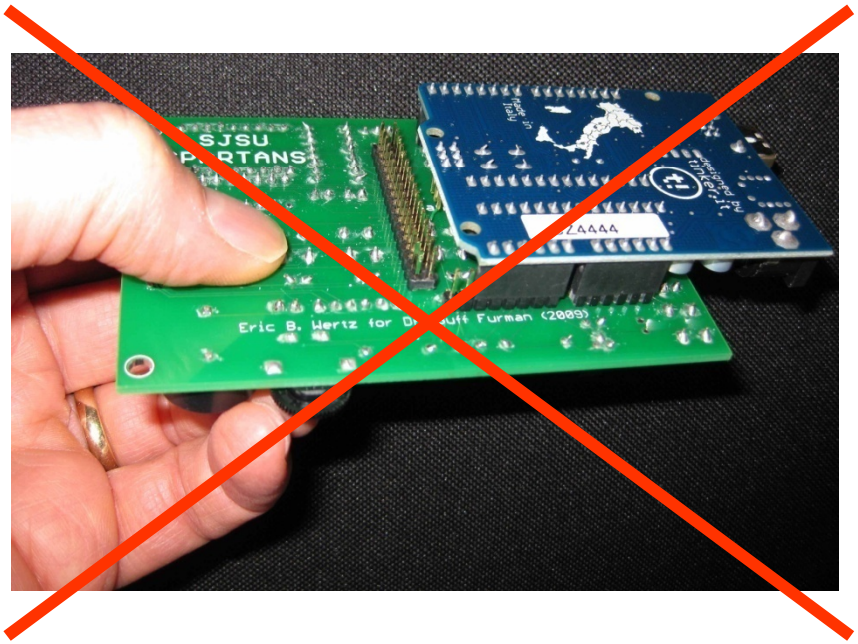
Arduino MEGA ADK

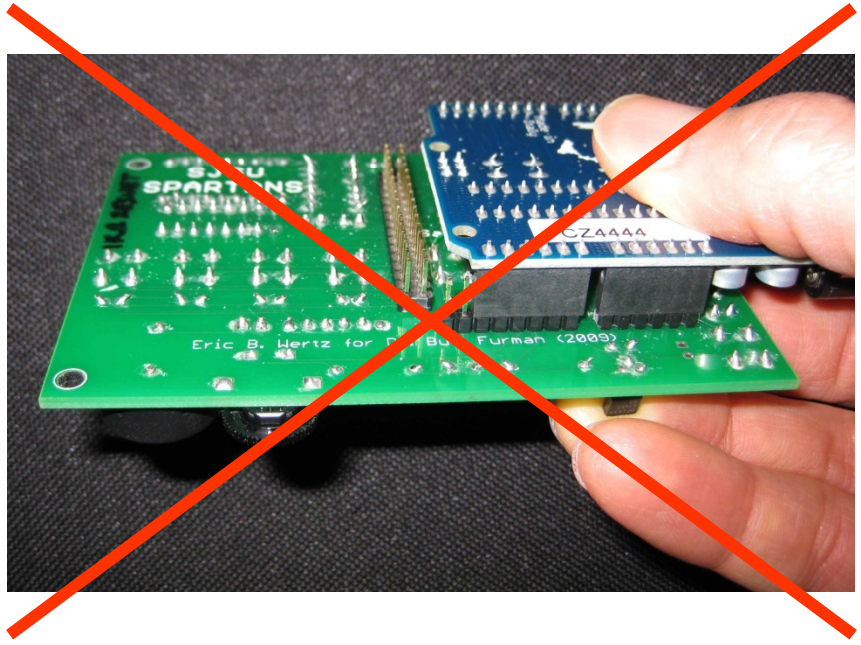


Arduino Lilypad



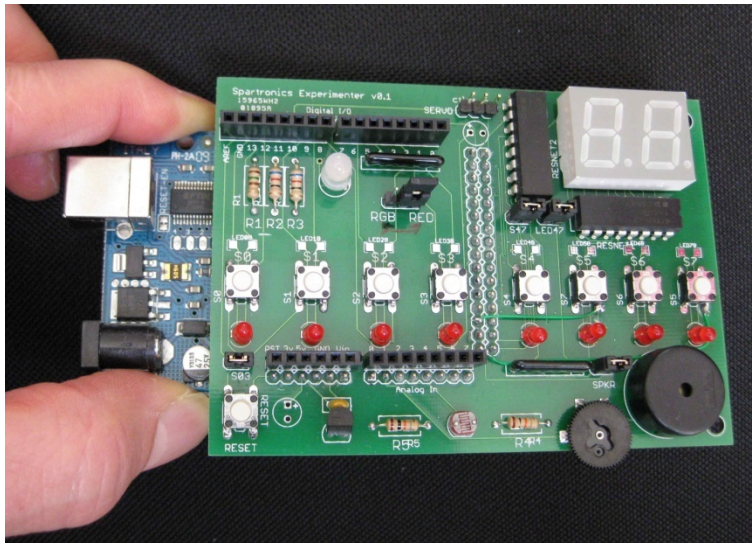
Handling the Arduino - How *NOT* to Do It!

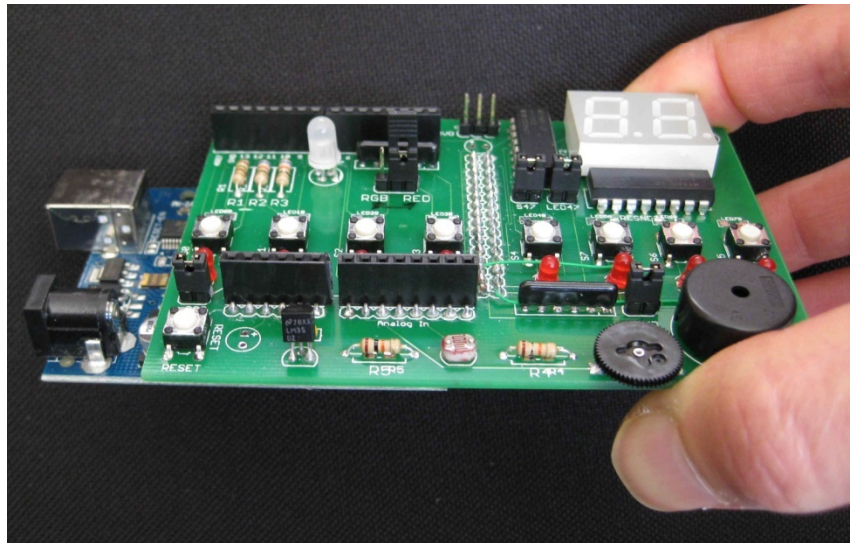




Handling the Arduino - The Proper Way

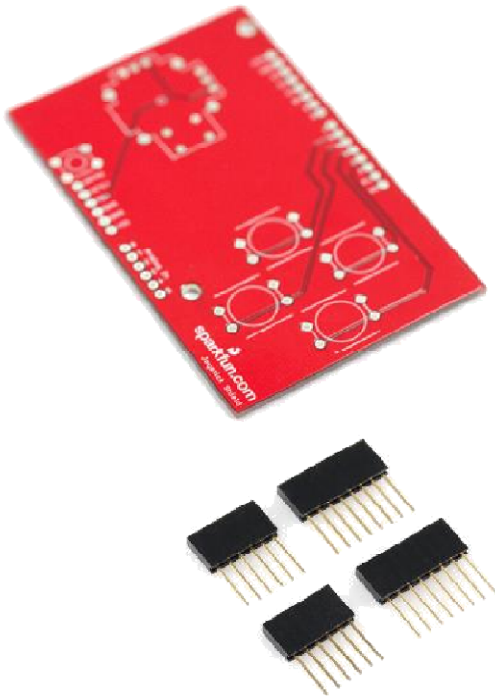
Proper Handling - by the ***edges***!!!

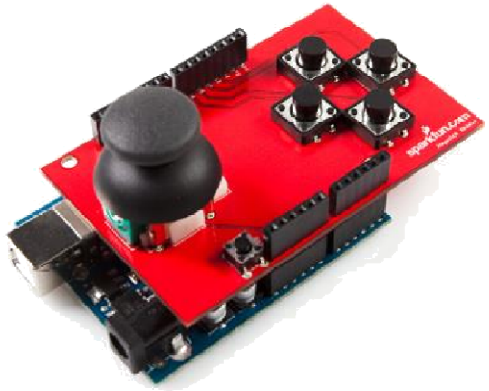




Arduino shields

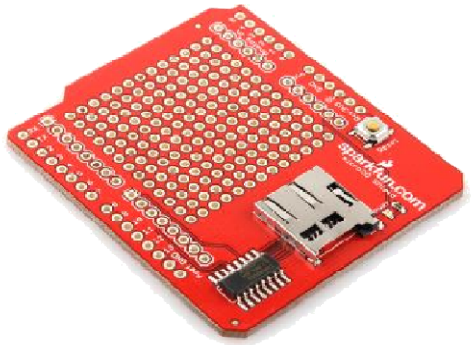
PCB → Built Shield → Inserted Shield





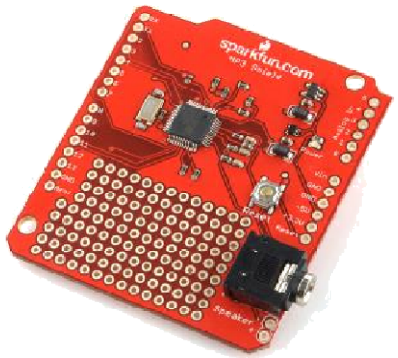
Arduino Shields

Micro SD



The microSD Shield equips your Arduino with mass-storage capability

MP3 Trigger



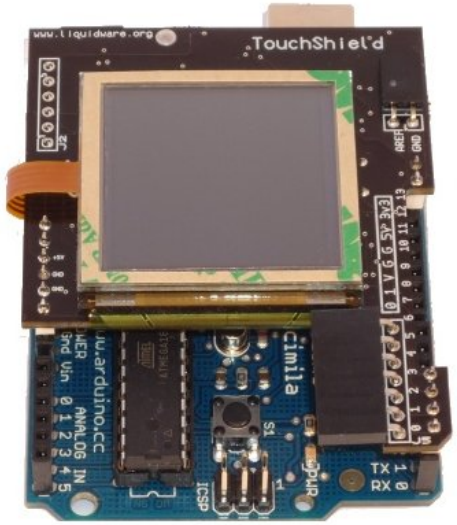
Playing MP3 files

LCD

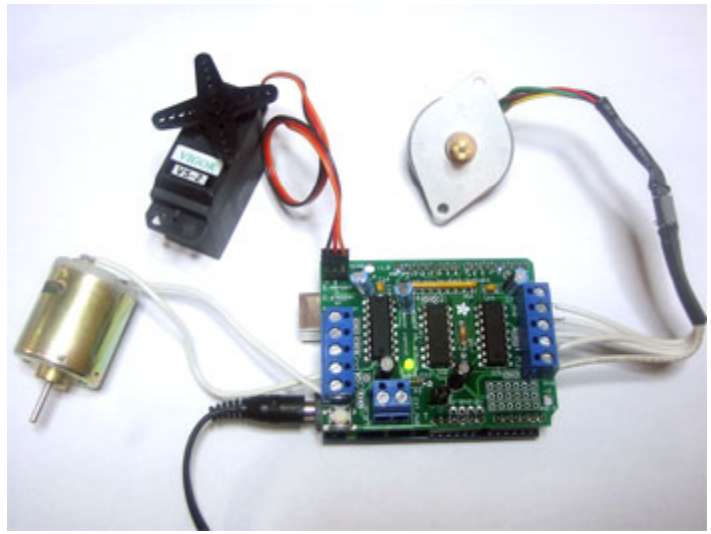
Typical LCD Display.

Arduino Shields

OLED touch screen shield



Servo/Stepper/DC Motor shield



A shield that can control 2 hobby servos and up to 2 unipolar/bipolar stepper motors or 4 bi-directional DC motors.

Arduino Shields

GPS & Datalogging shield

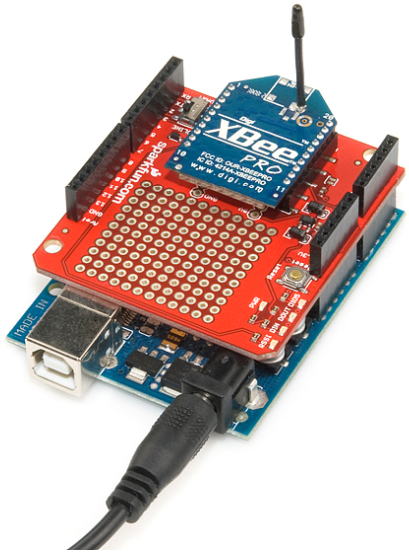
Connects up a GPS module and can log location, time/date as well as sensor data to an SD memory flash card.

XPort/Ethernet shield

Allows use of an XPort module for connecting to the Internet as a client or server.

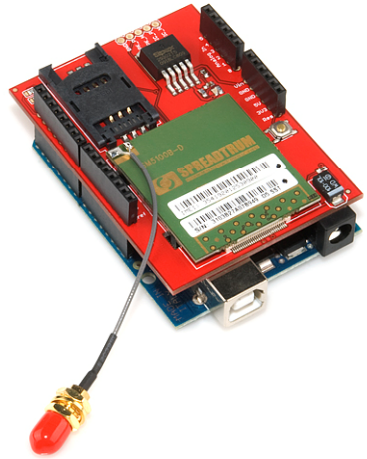
Arduino Shields

XBEE Module and XBEE Shield



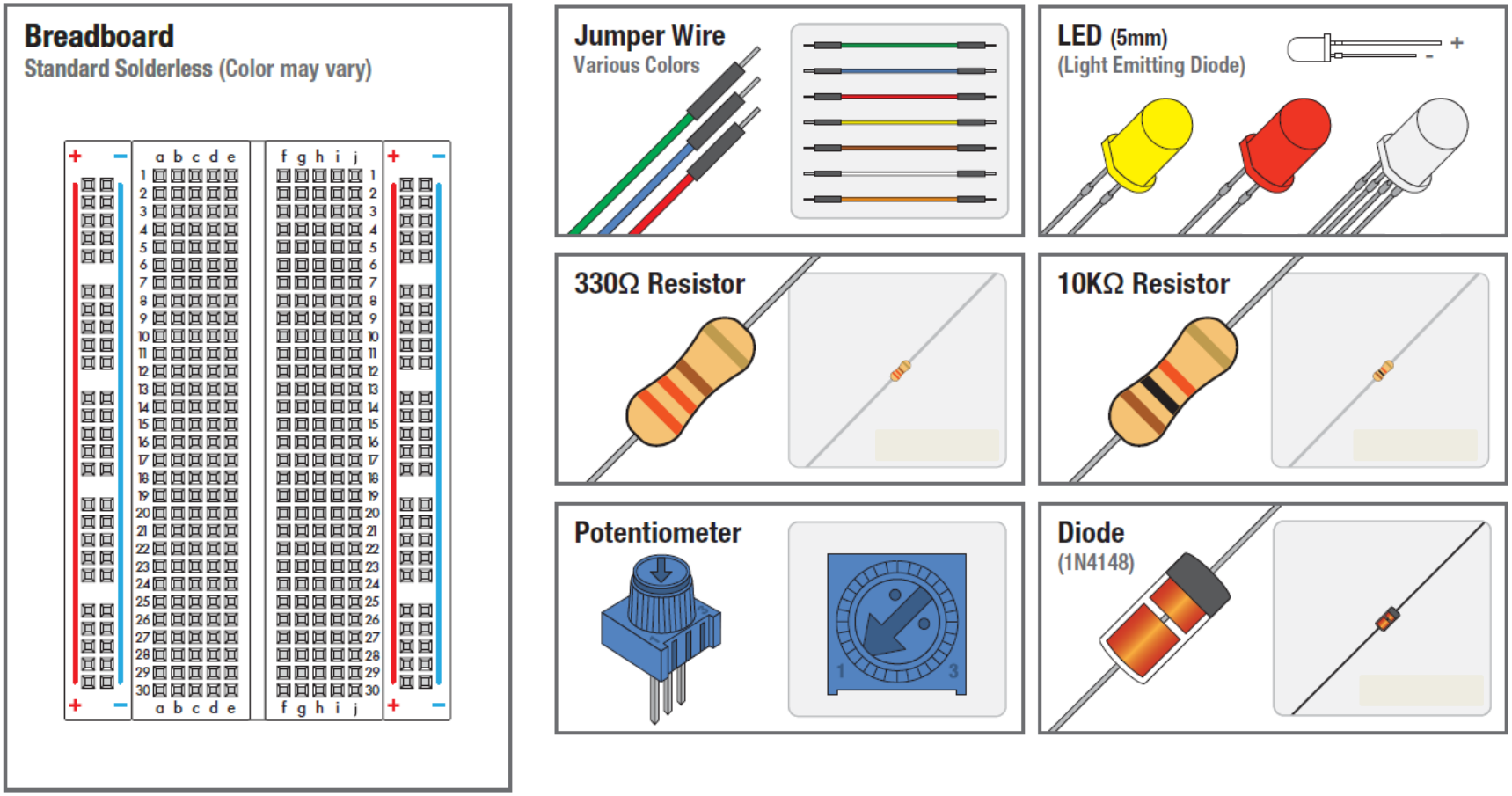
These modules take the 802.15.4 stack (the basis for Zigbee) and wrap it into a simple to use serial command set.

Cellular Shield with SM5100B

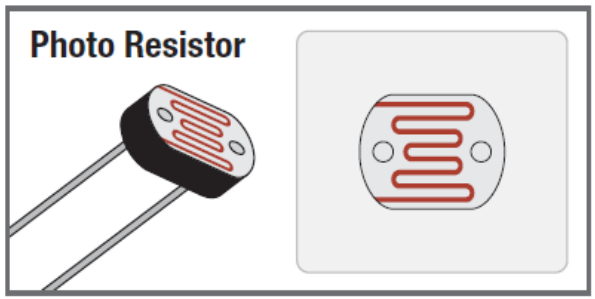


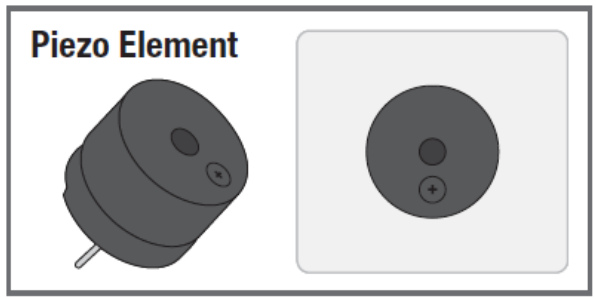
This allows you to easily add SMS, GSM/GPRS, and TCP/IP functionalities to your Arduino-based project.

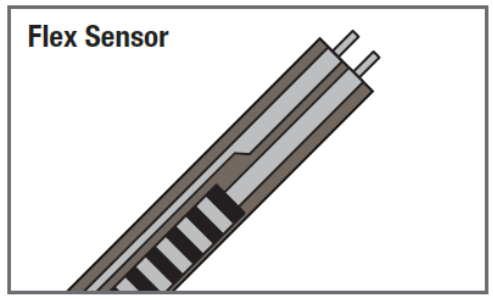
Electronics components

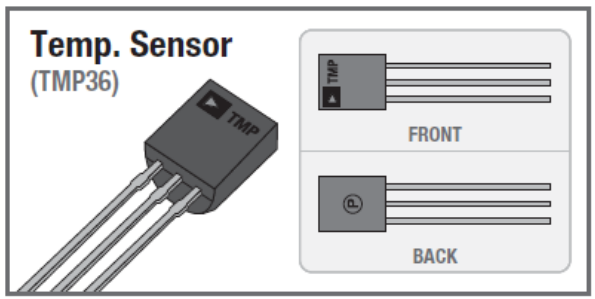


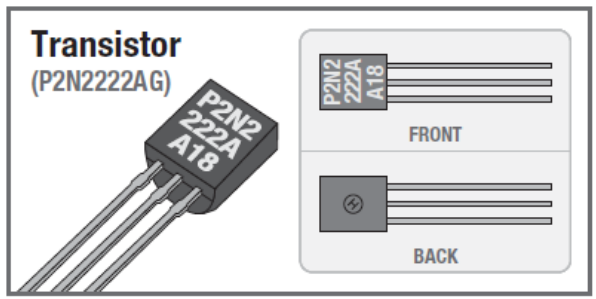
Electronics components

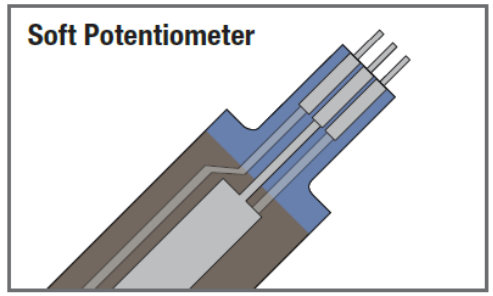


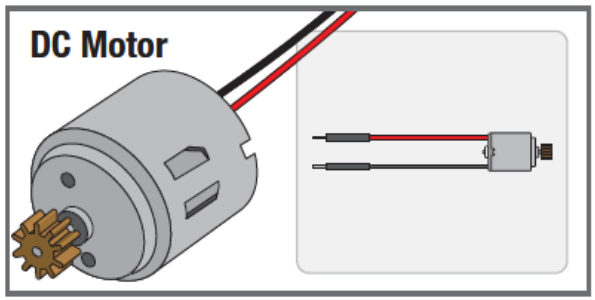


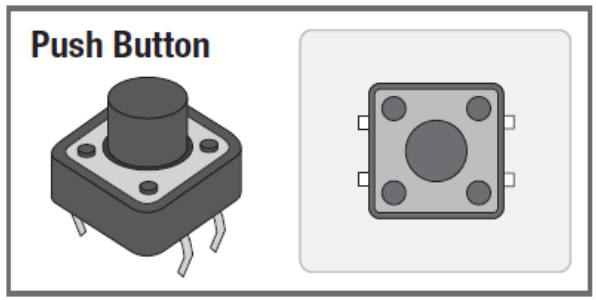


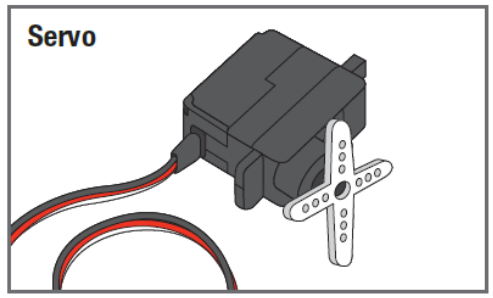




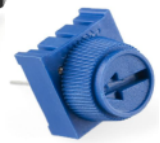








Electronics components details

Name	Image	Type	Function	Notes
Push Button		Digital Input	Switch - Closes or opens circuit	Polarized, needs resistor
Trim potentiometer		Analog Input	Variable resistor	Also called a Trimpot.
Photoresistor		Analog Input	Light Dependent Resistor (LDR)	Resistance varies with light.
Relay		Digital Output	Switch driven by a small signal	Used to control larger voltages
Temp Sensor		Analog Input	Temp Dependent Resistor	
Flex Sensor		Analog Input	Variable resistor	
Soft Trimpot		Analog Input	Variable resistor	Careful of shorts
RGB LED		Dig & Analog Output	16,777,216 different colors	

Ohm's Law

- Ohm's Law describes the direct relationship between the Voltage(V), Current(I), and Resistance(R) of a circuit
- The three different forms of Ohm's Law are follows:

$$V = I R \qquad I = V / R \qquad R = V / I$$

Electrical Properties

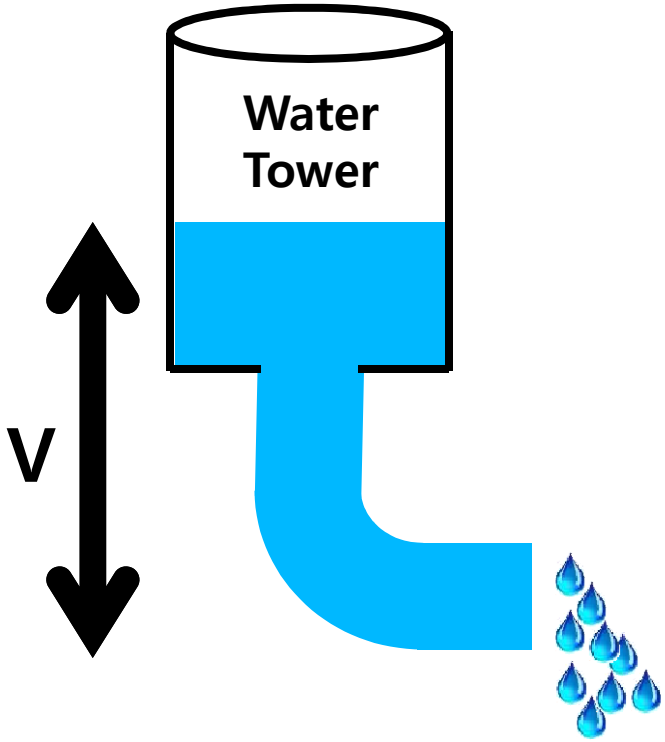
Voltage V	Current I	Resistance R
• Defined as the amount of potential energy in a circuit. • Units: Volts (V)	• The rate of charge flow in a circuit. • Units: Amperes (A)	• Opposition to charge flow. • Units: Ohms (Ω)

Current Flow Analogy

High Current

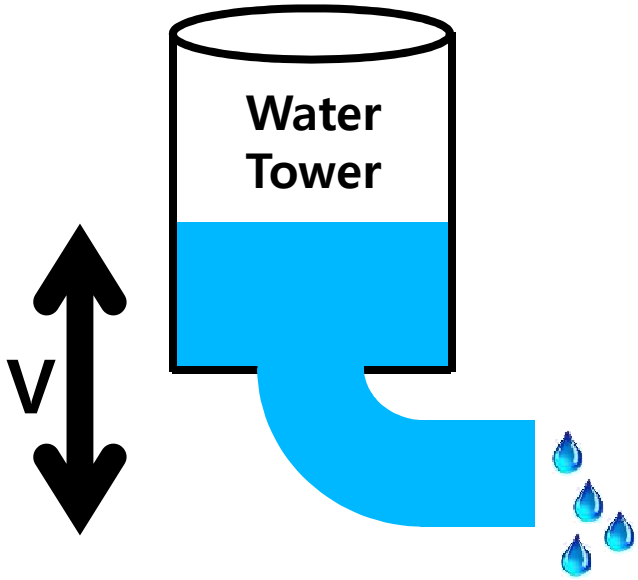
Low Current

Voltage Analogy



More Energy == Higher Voltage

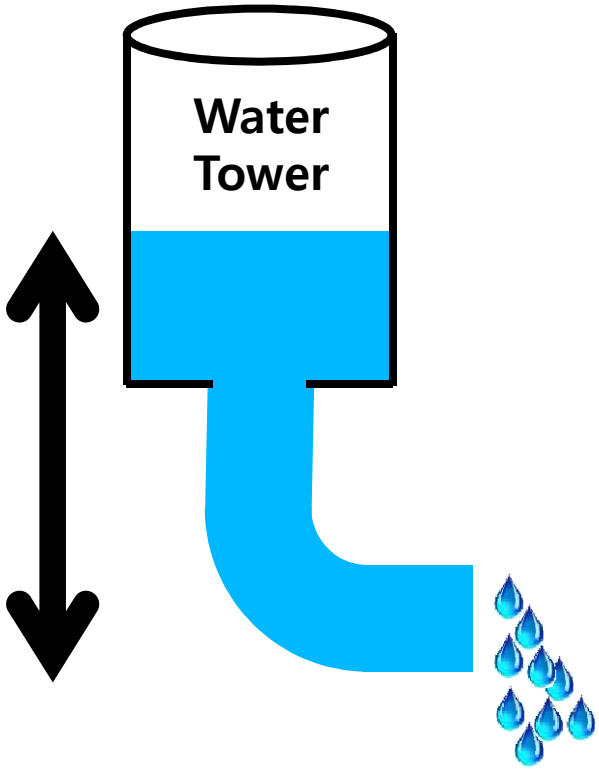
$V = I_R$



Less Energy == Lower Voltage

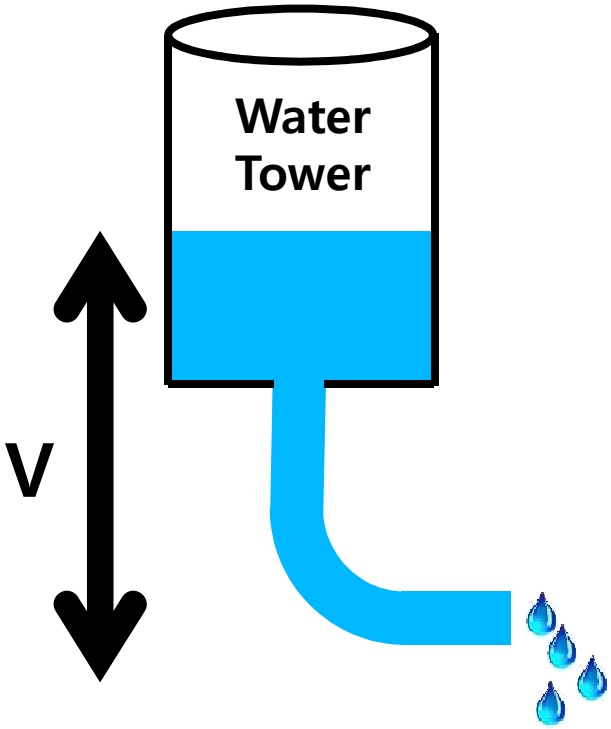
$V = I_R$

Resistance Analogy



Big Pipe == Lower Resistance

$V = IR$

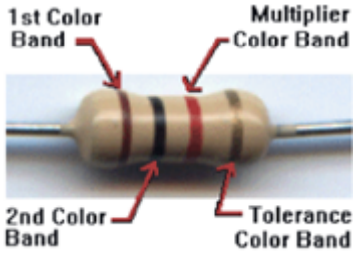


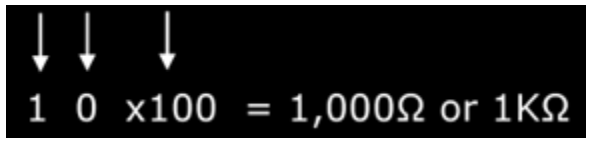
Small Pipe == Higher Resistance

$V = IR$

Resister Color Coding

- To identify the resistance we can use a code system
  - We position them so that the gold strip is on the right side and then we measure





Color	First Strip	Second Strip	Third Strip
Black	0	0	x1
Brown	1	1	x10
Red	2	2	x100
Orange	3	3	x1,000
Yellow	4	4	x10,000
Green	5	5	x100,000
Blue	6	6	x1,000,000
Purple	7	7	
Gray	8	8	
White	9	9	

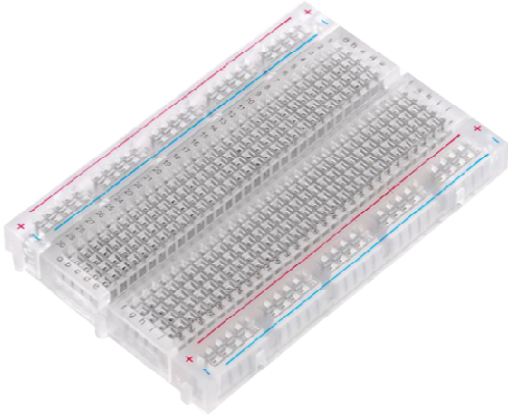
Circuit

- The word “circuit” is derived from the circle. An Electrical Circuit must have a continuous LOOP from Power (V_{cc}) to Ground (GND).

- Continuity is important to make portions of circuits are connect. Continuity is the simplest and possibly the most important setting on a multi-meter. Sometimes we call this “ringing out” a circuit.

Prototyping Circuits

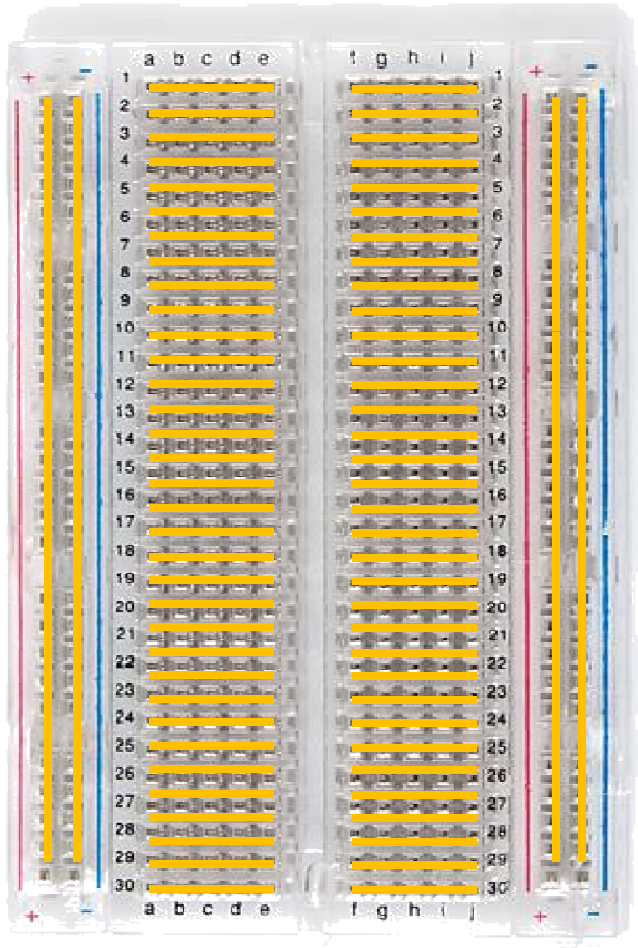
- Solderless Breadboard
  - One of the most useful tools in an engineer or Maker's toolkit
  - The three most important things
    - A breadboard is easier than soldering
    - A lot of those little holes are connected
    - Sometimes breadboards break



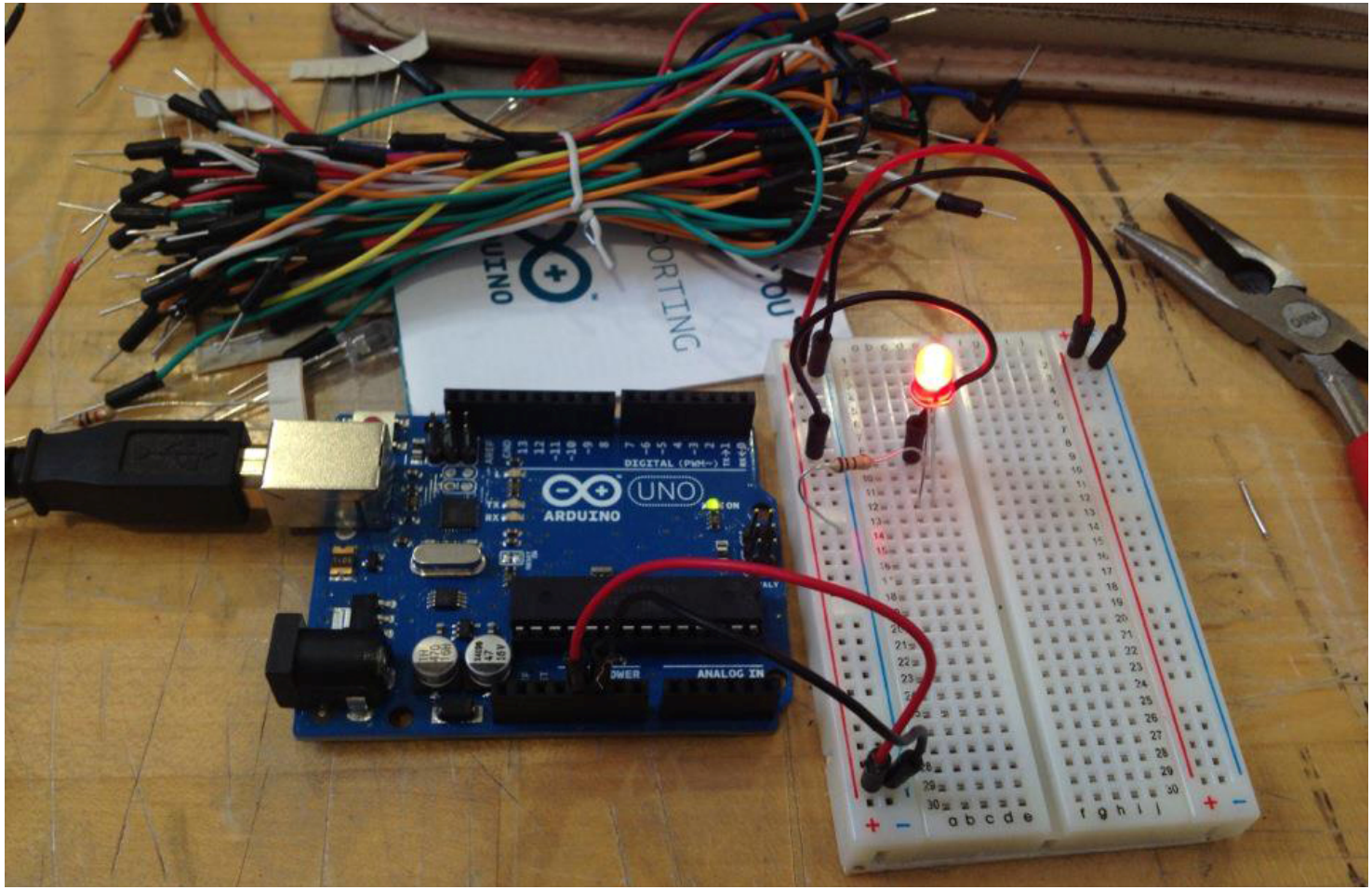
Solderless Breadboard

- Each row (horiz.) of 5 holes are connected.

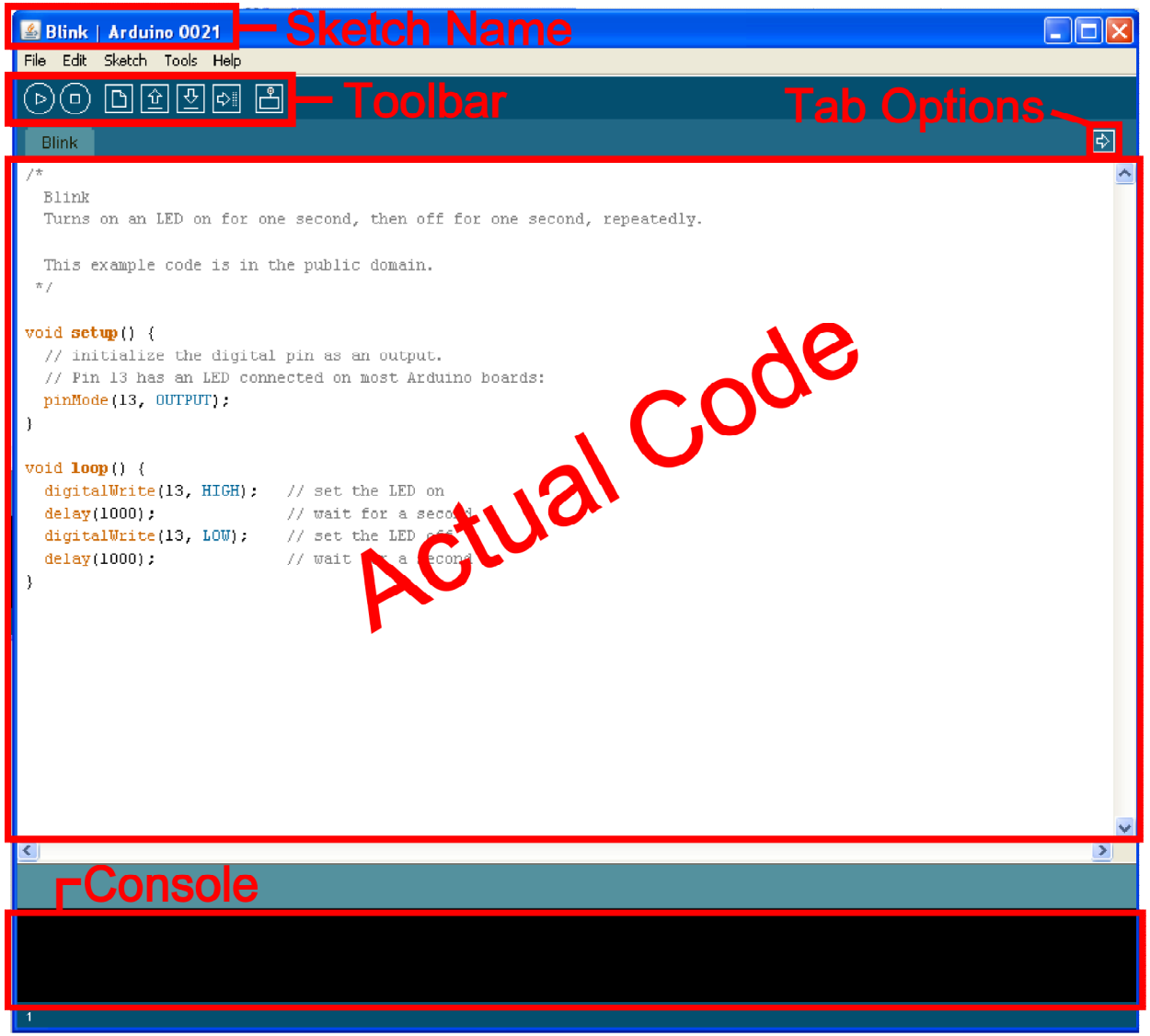
- Vertical columns – called power bus are connected vertically



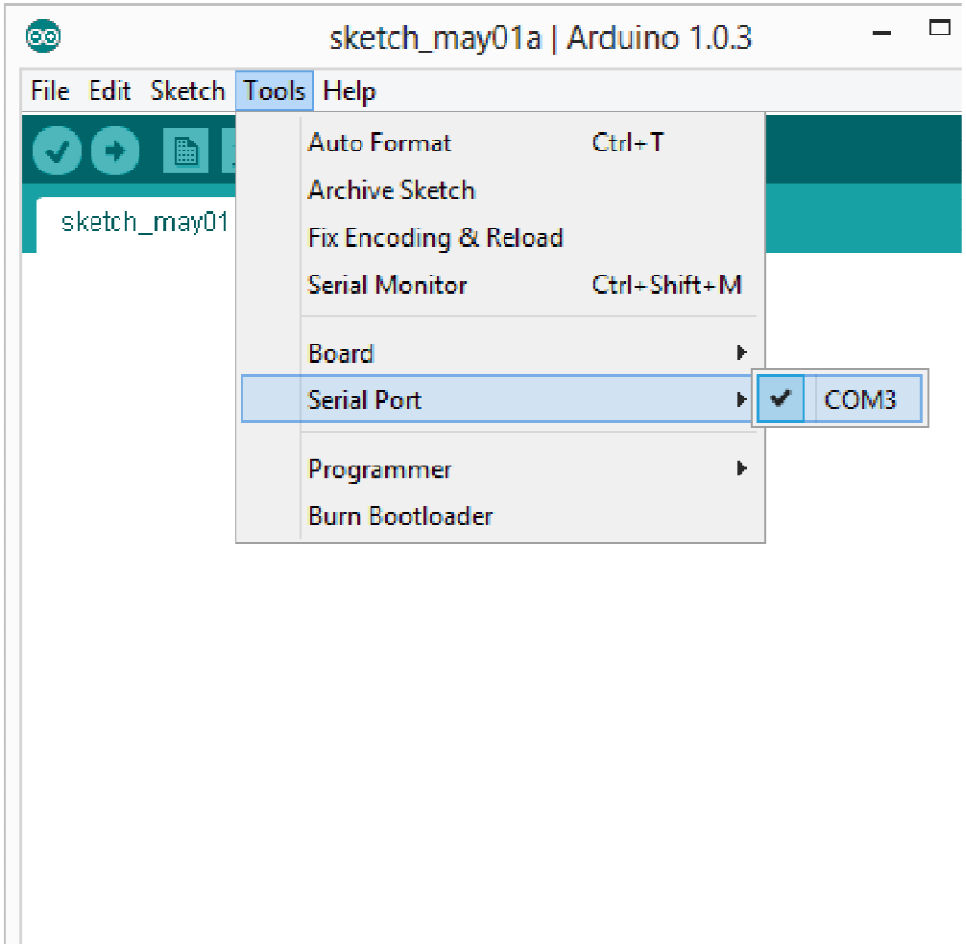
Building a circuit and start Programming



Arduino
Integrated Development Environment (IDE)



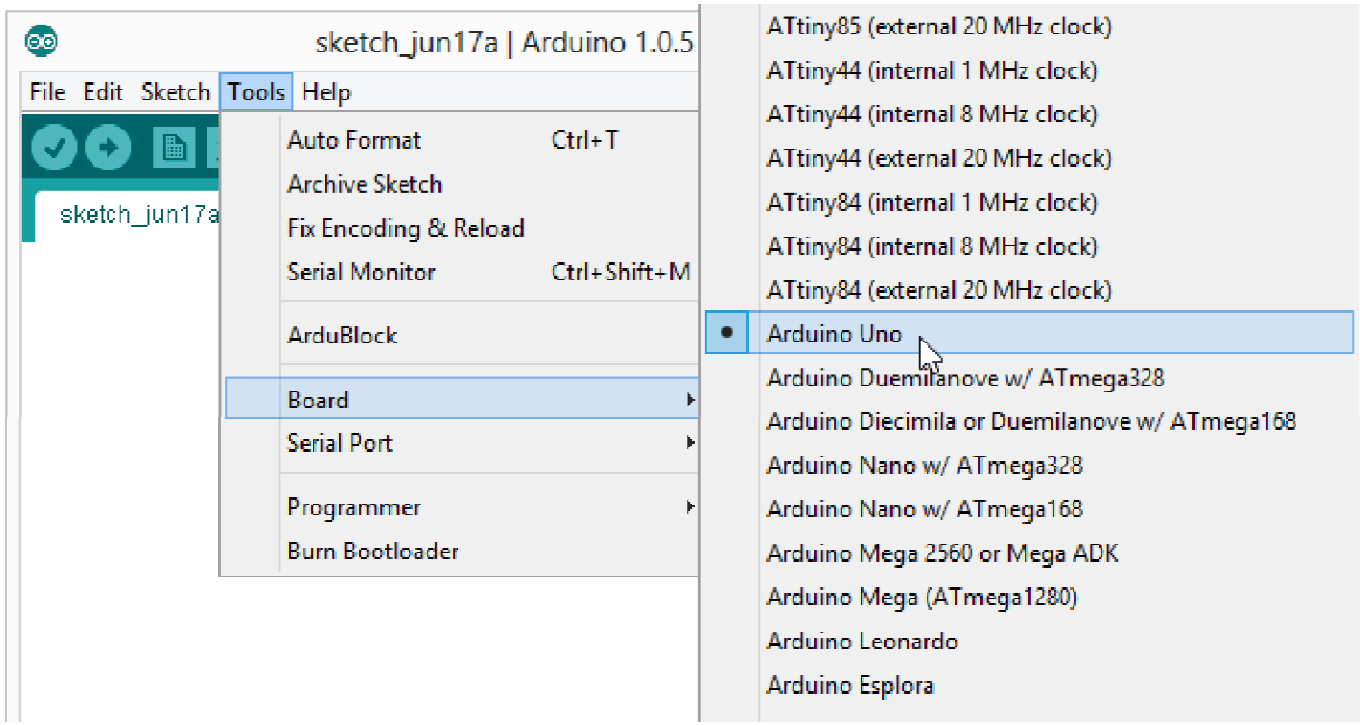
Tools → Serial Port



- Your computer communicates to the Arduino microcontroller via a serial port → through a USB-Serial adapter.

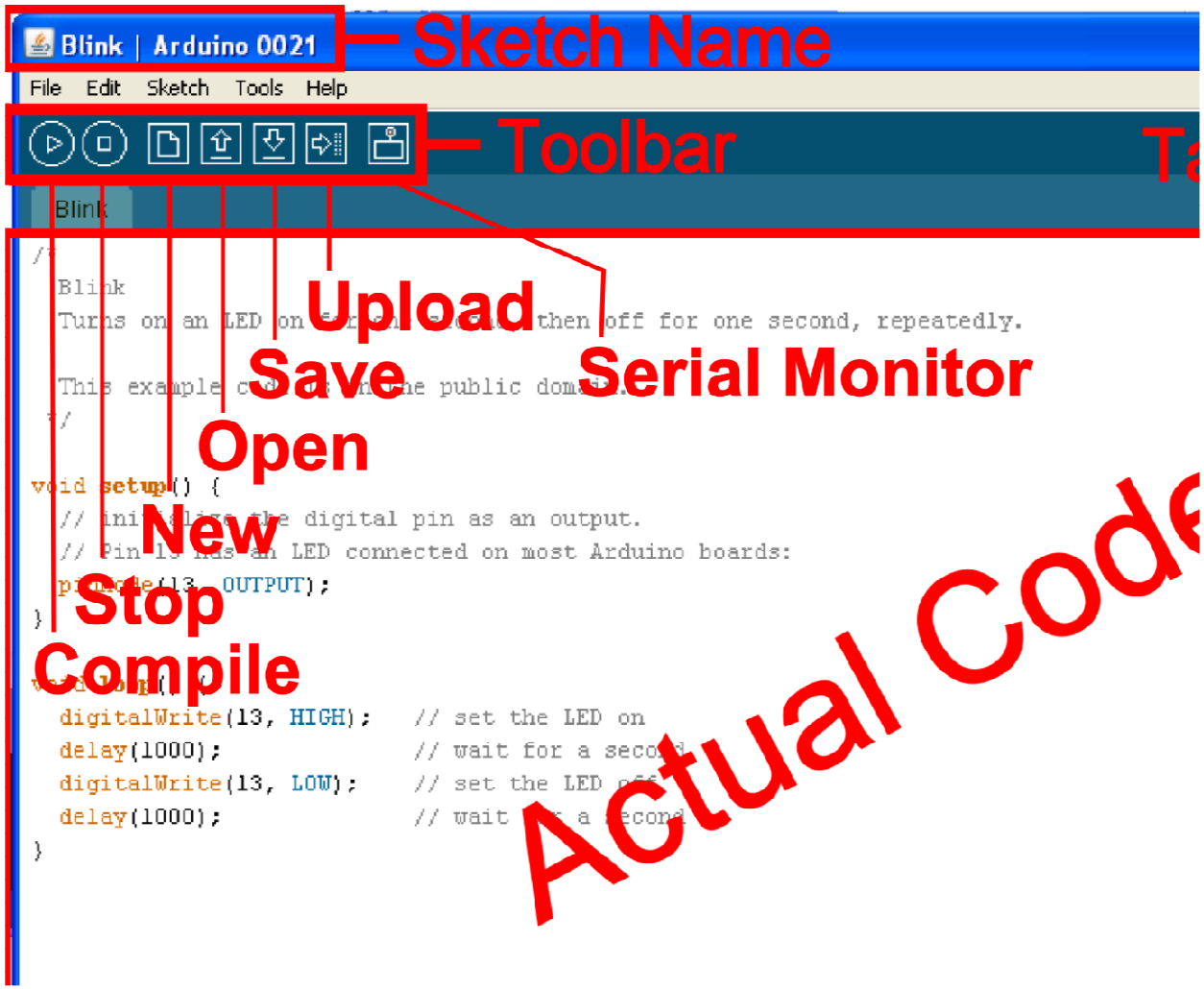
- Check to make sure that the drivers are properly installed.

Tools→Board

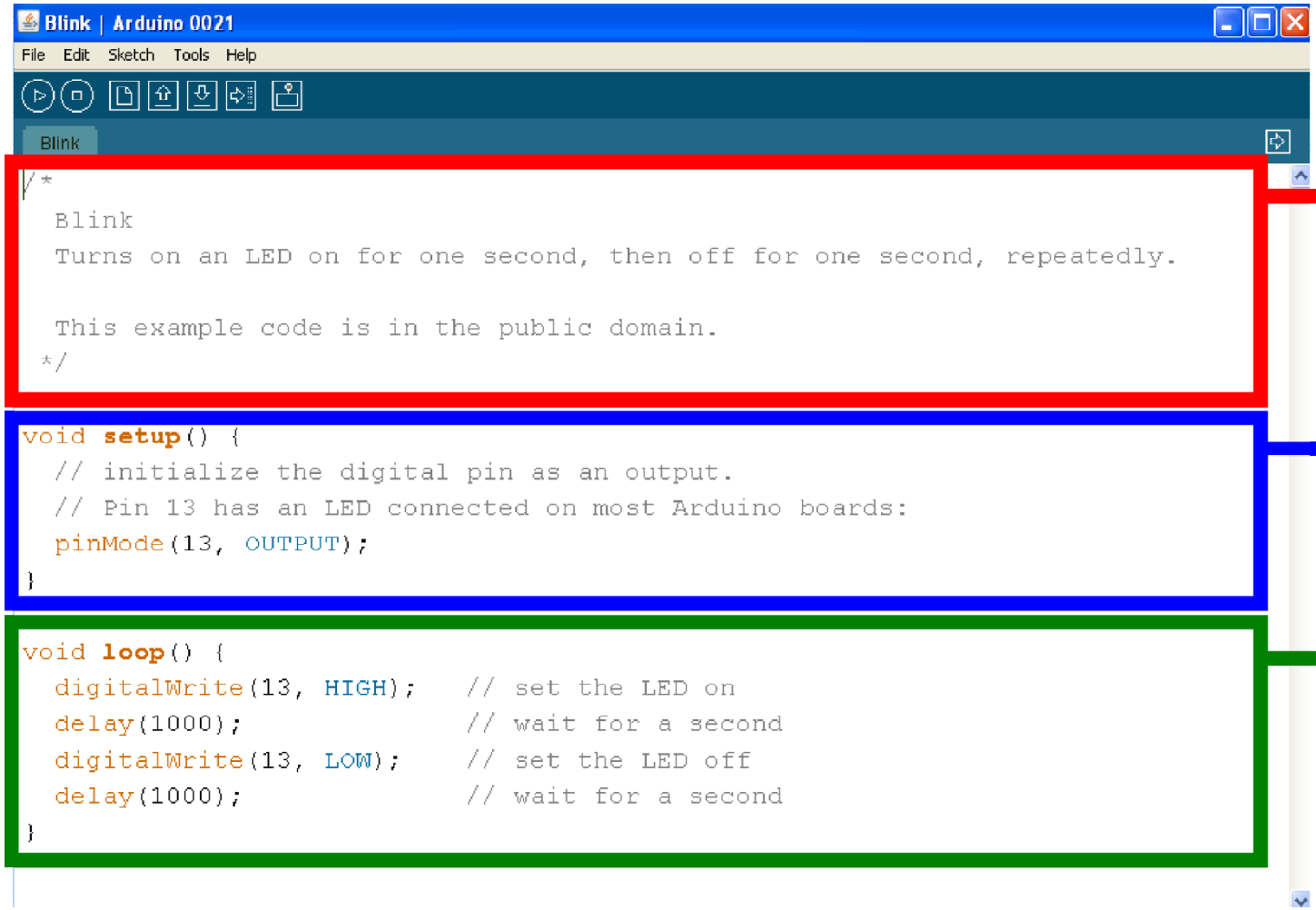


•Next, double-check that the proper board is selected under the Tools→Board menu.

Toolbar


Blink | Arduino 0021
Sketch Name
File Edit Sketch Tools Help
Toolbar
Blink
Upload
Save
Serial Monitor
Open
New
Stop
Compile
Actual Code
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
This example code is in the public domain.
*/
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
}
void loop() {
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
}

Parts of the Sketch



Comments / Explaining the game

Setup / Stretching or tying shoes

Loop / Playing the game

Comments

- Comments can be anywhere
- Comments created with // or /* and */
- Comments do not affect code
- You may not need comments, but think about the community!

Operators

- The equals sign
  - = is used to assign a value
  - == is used to compare values
- And & Or
  - && is “and”
  - || is “or”

Variables

- Basic variable types
  - Boolean
    - boolean variableName;
  - Integer
    - int variableName;
  - Character
    - char variableName;
  - String
    - stringName [];

Assigning Variables

Boolean: ***variableName = true;***

or ***variableName = false;***

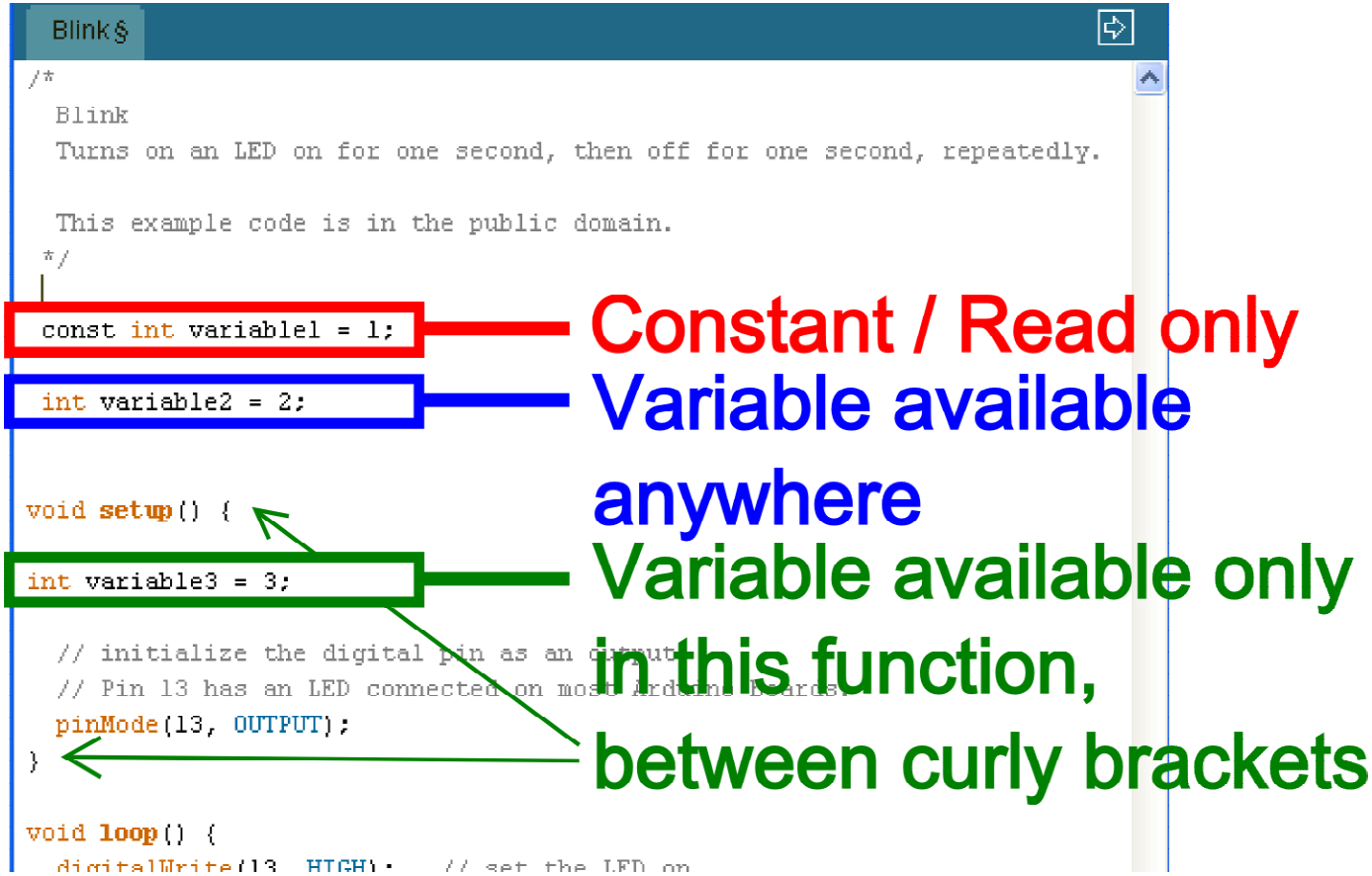
Integer: ***variableName = 32767;***

or ***variableName = -32768;***

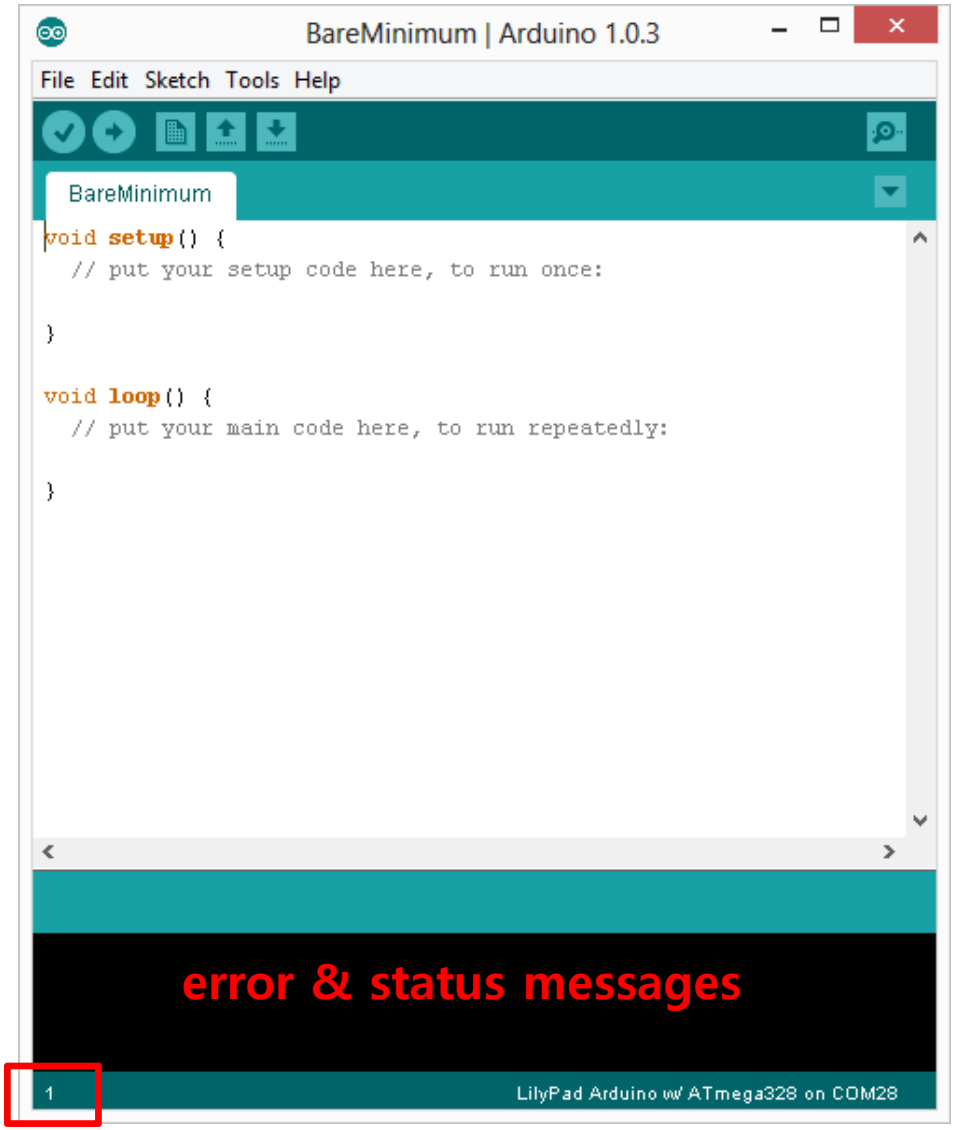
Character: ***variableName = 'A';***

or ***stringName = "SparkFun";***

Variable Scope



Basic sketch



Two required functions / methods / routines:

```
void setup()
{
      // runs once
}

void loop()
{
      // repeats
}
```

Setup

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

The setup function comes before the loop function and is necessary for all Arduino sketches

void setup()

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

The setup header will never change, everything else that occurs in setup happens inside the curly brackets

pinMode (input/output) setup

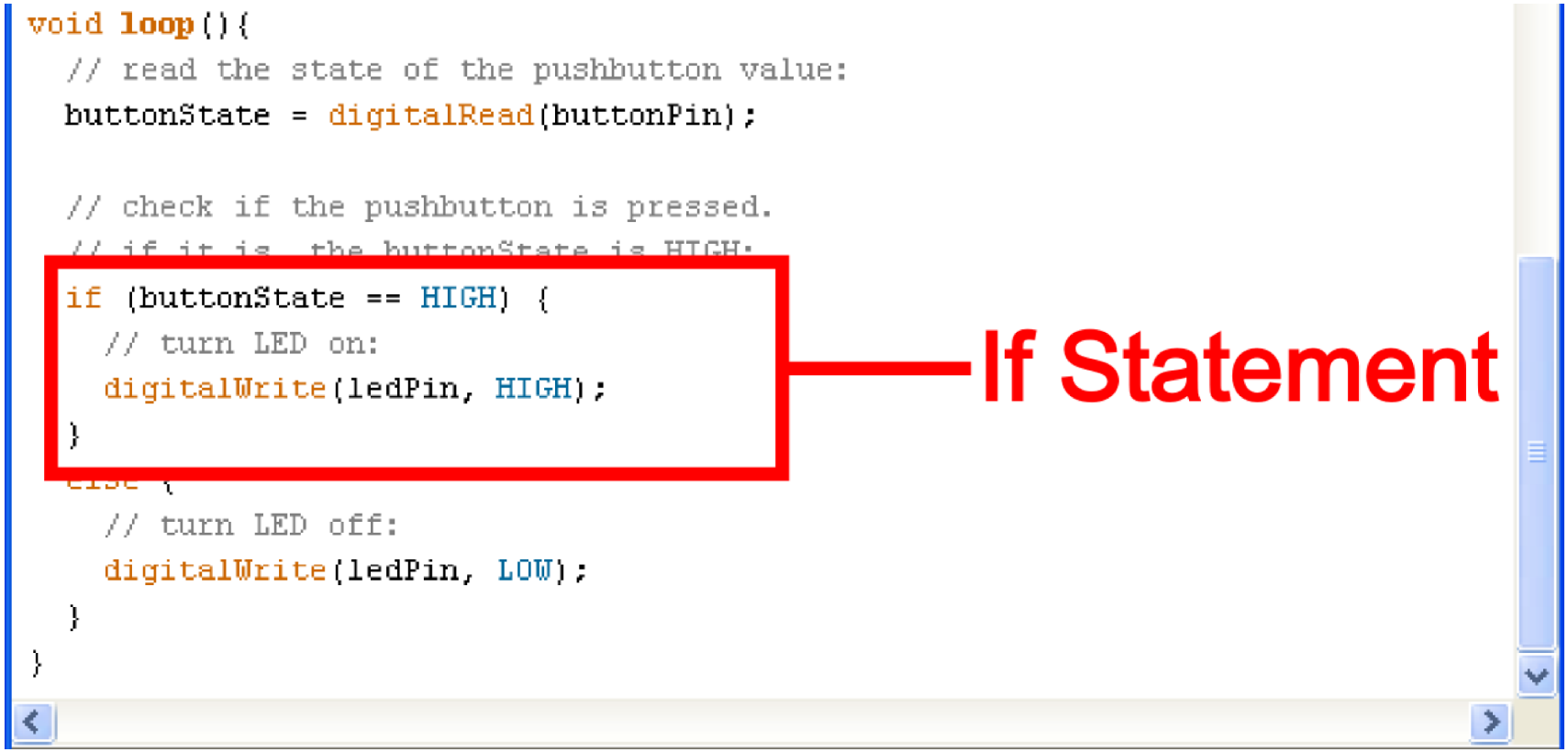
```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

Outputs and Inputs are declare in setup, this is done by using the pinMode function

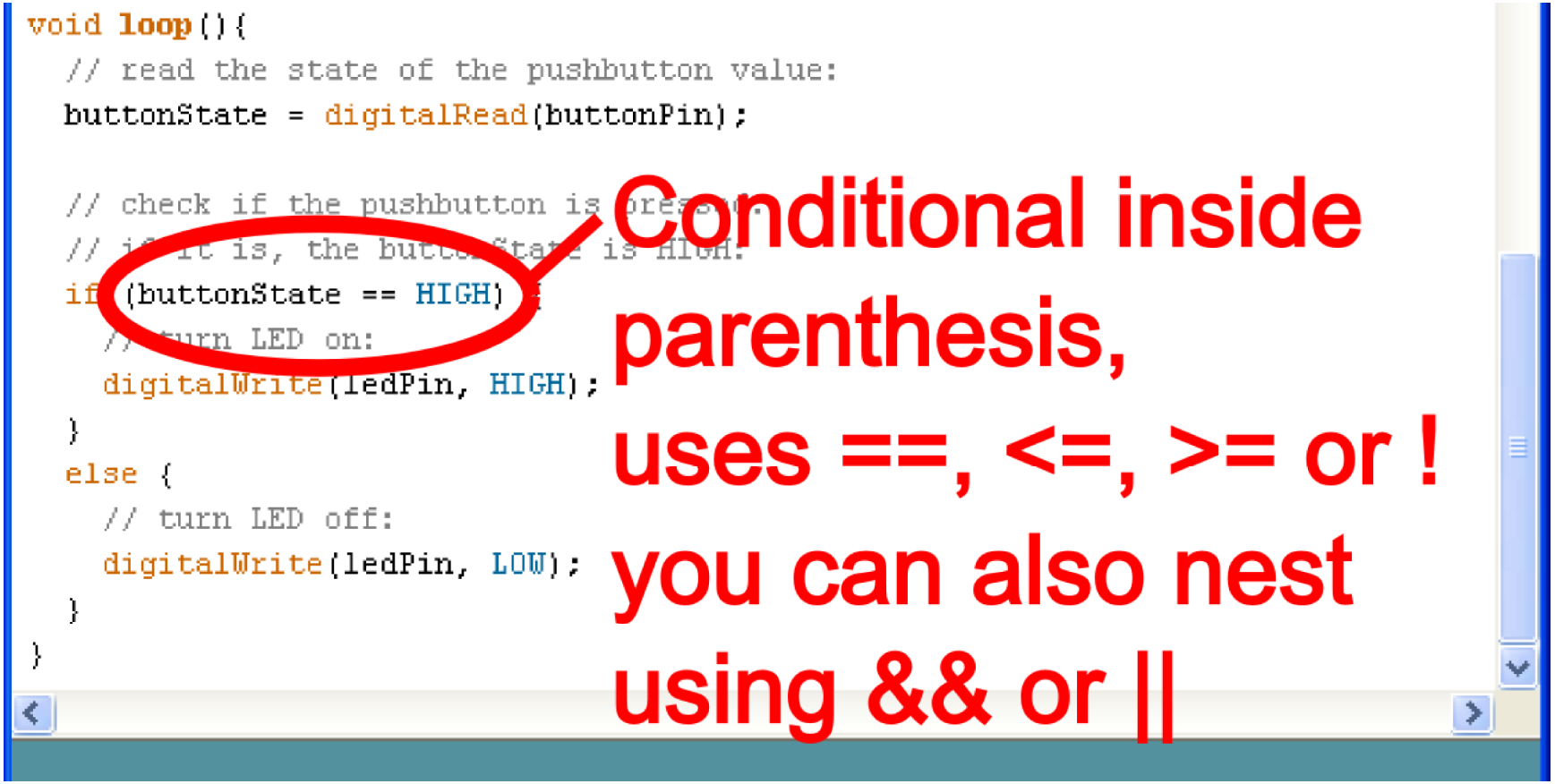
This particular example declares digital pin # 13 as an output, remember to use CAPS

If Statements

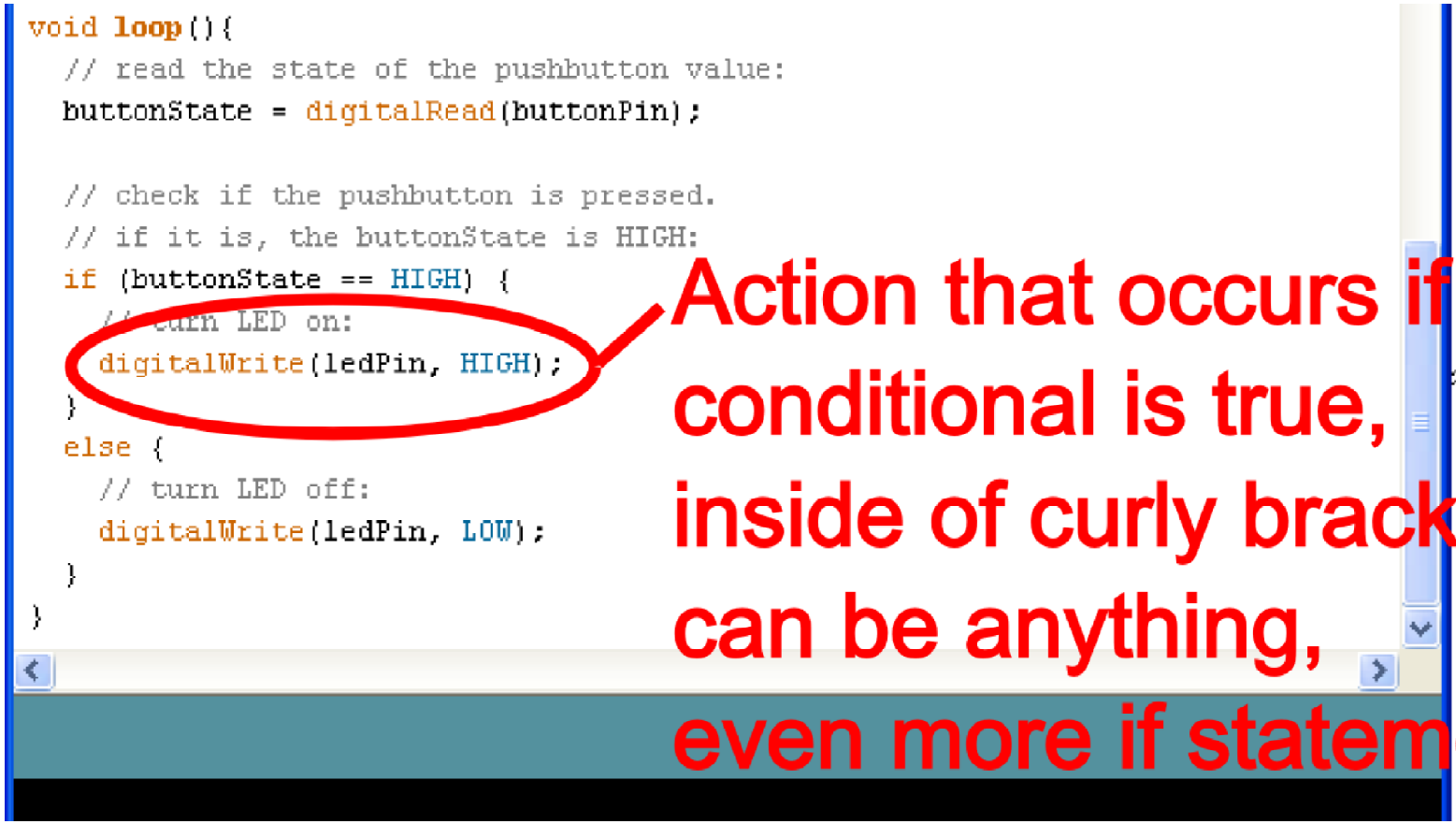
- if (this is true) { do this; }



Conditions


void loop(){
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);
// check if the pushbutton is pressed.
// if it is, the buttonState is HIGH:
if (buttonState == HIGH) {
// turn LED on:
digitalWrite(ledPin, HIGH);
}
else {
// turn LED off:
digitalWrite(ledPin, LOW);
}
}
Conditional inside parenthesis, uses ==, <=, >= or ! you can also nest using && or ||

Action

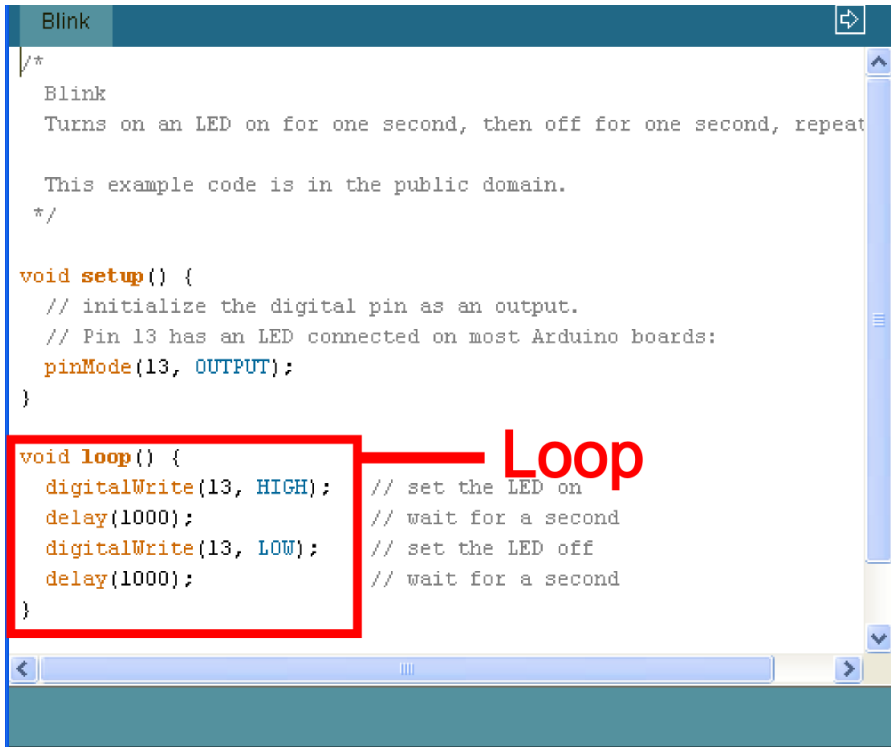

void loop(){
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);
// check if the pushbutton is pressed.
// if it is, the buttonState is HIGH:
if (buttonState == HIGH) {
// turn LED on:
digitalWrite(ledPin, HIGH);
}
else {
// turn LED off:
digitalWrite(ledPin, LOW);
}
}
Action that occurs if conditional is true, inside of curly brackets, can be anything, even more if statements

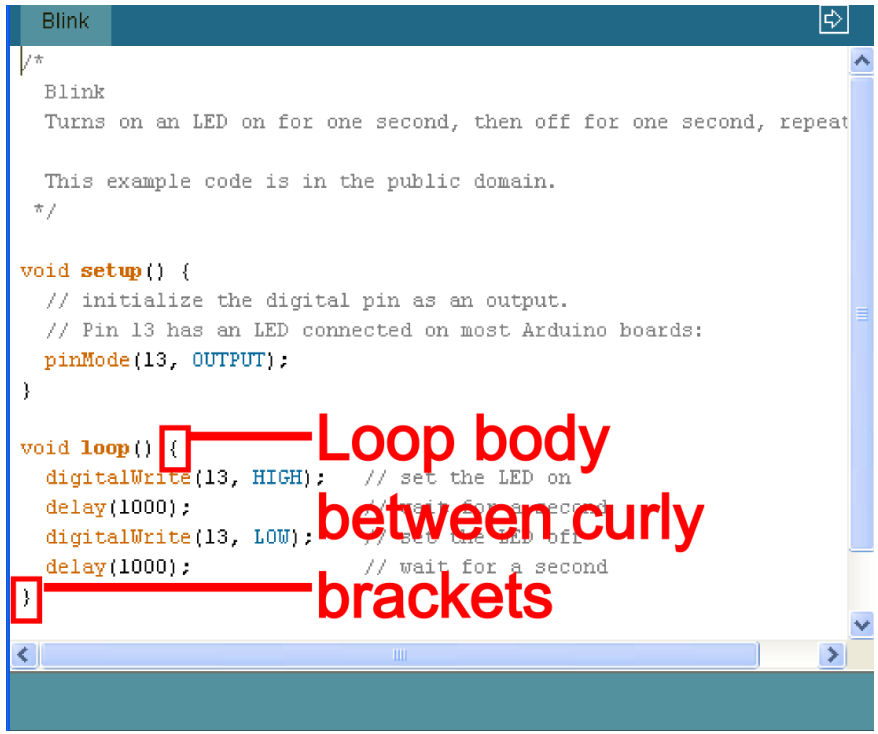
else

```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Else, optional

Basic Repetition : Loop





The "void" in the header is what the function will return(or spit out) when it happens, in this case it returns nothing so it is void

The "loop" in the header is what the function is called, sometimes you make the name up, sometimes (like loop) the function already has a name

Basic Repetition : for

```
for (int count = 0; count<10; count++)
{
//for action code goes here
//this could be anything
}
```

```
void setup()
{
  //Set each pin connected to an LED to output mode (pulling high
  for(int i = 0; i < 8; i++){                    //this is a for loop and will r
      pinMode(ledPins[i],OUTPUT); //we use this to set each LED p
  }                                              //the code this replaces is

  /* (commented code will not run)
   * these are the lines replaced by the for loop above they do e
   * same thing the one above just uses less typing
  pinMode(ledPins[0],OUTPUT);
  pinMode(ledPins[1],OUTPUT);
  pinMode(ledPins[2],OUTPUT);
  pinMode(ledPins[3],OUTPUT);
```

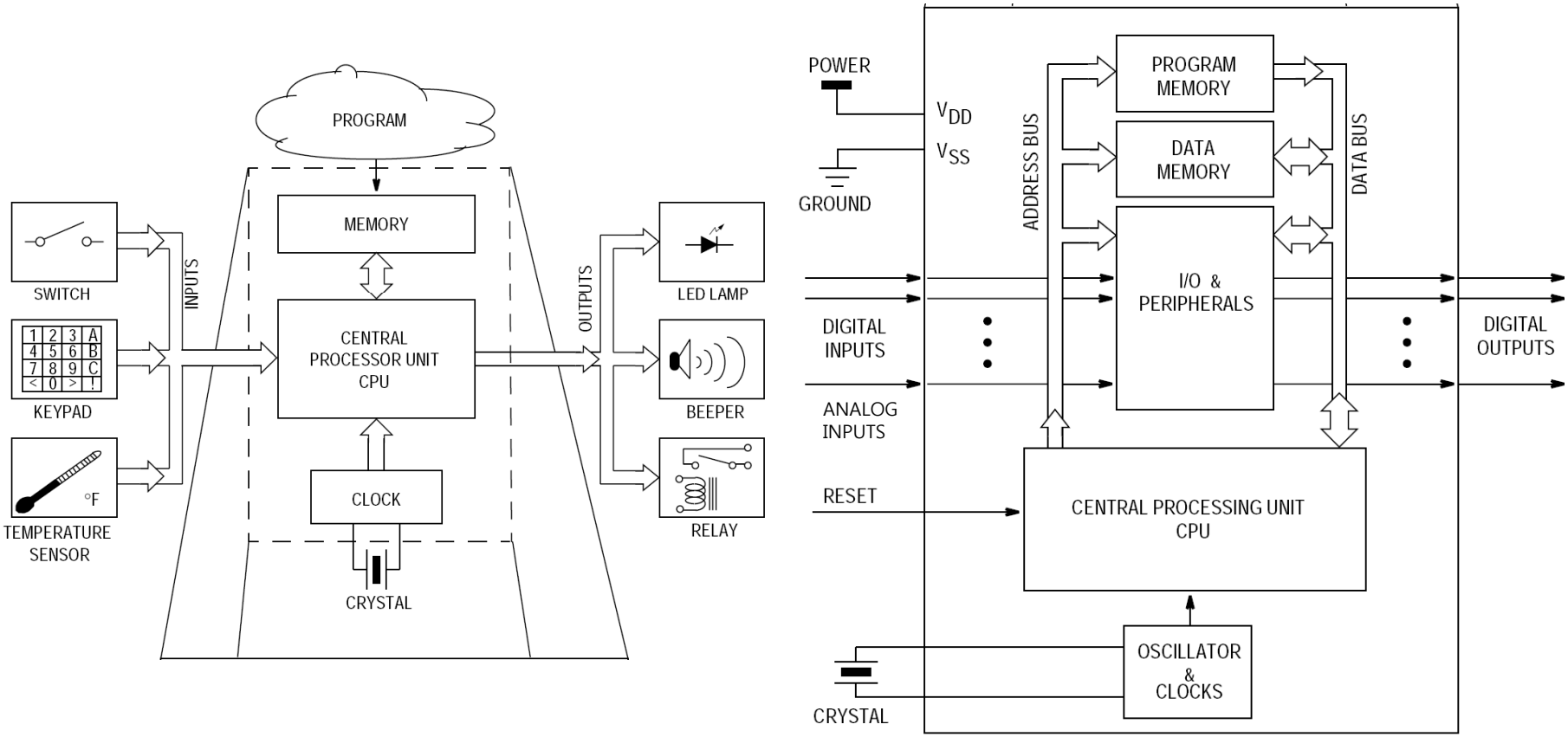
For loop

Basic Repetition : while

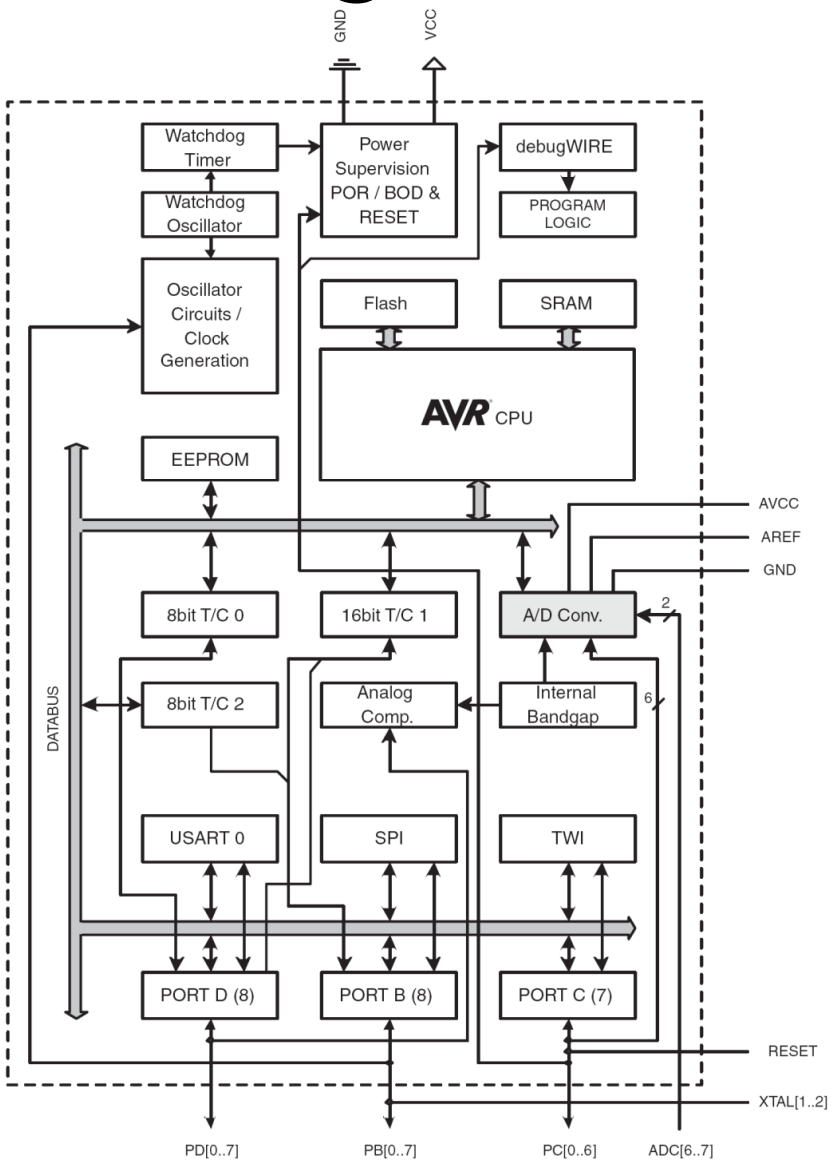
```
while ( count<10 )
{
//while action code goes here
//should include a way to change count
//variable so the computer is not stuck
//inside the while loop forever
}
```

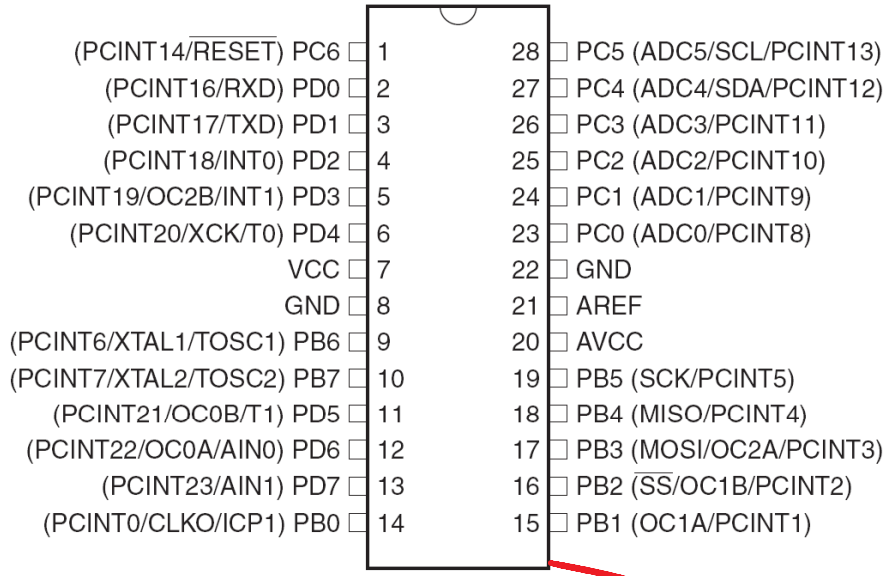
```
while ( digitalRead(buttonPin)==1 )
{
//instead of changing a variable
//you just read a pin so the computer
//exits when you press a button
//or a sensor is tripped
}
```

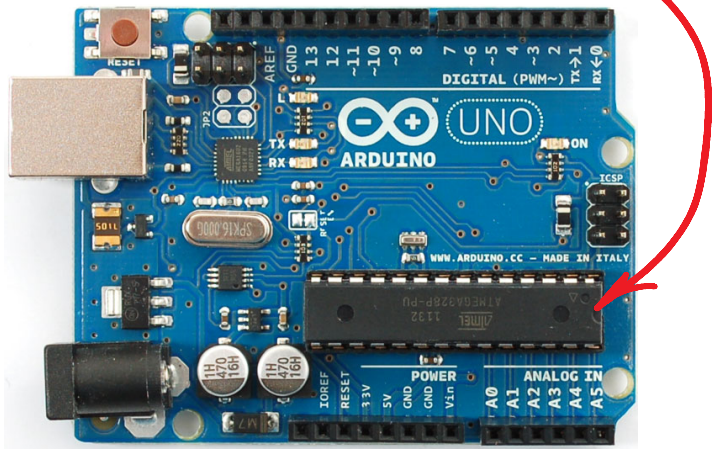
What is Microcontroller for an integrated embedded system?



Atmega 328 Internal Architecture





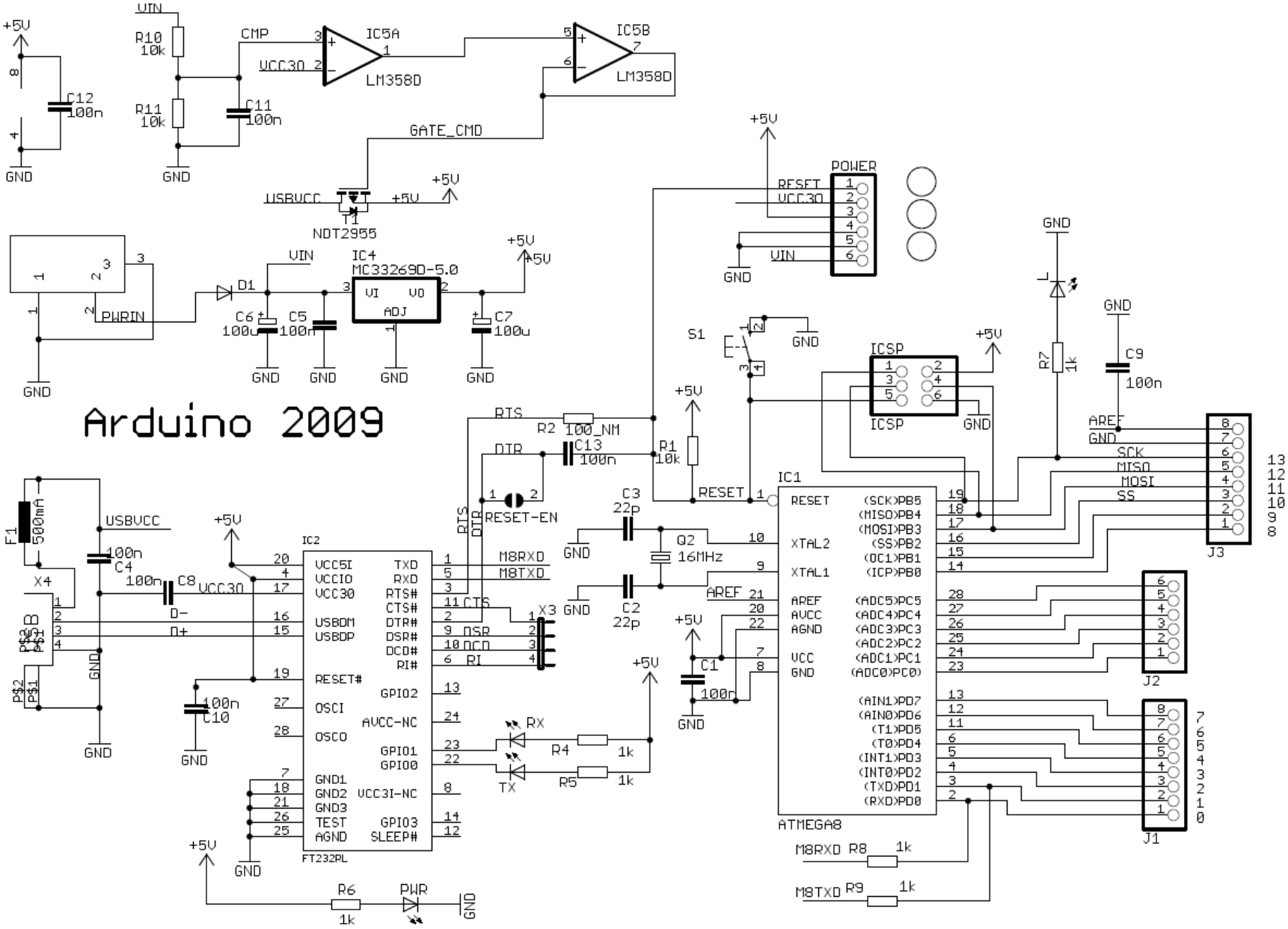


ATmega328 Features

Features

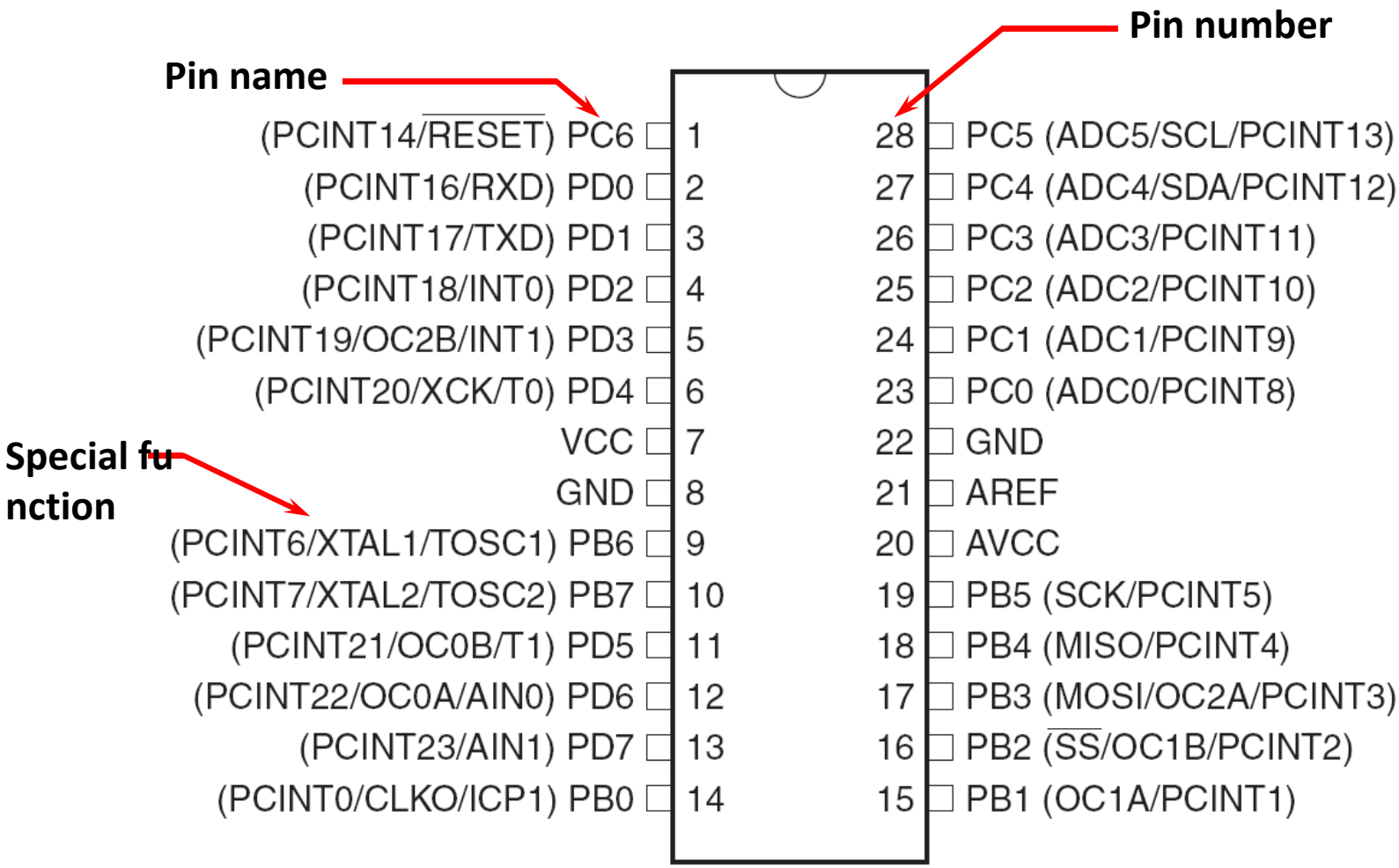
- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1K Bytes EEPROM
  - 512/1K/1K/2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf



http://arduino.cc/en/uploads/Main/arduino-duemilanove-schematic.pdf

ATmega328 Microcontroller



Absolute Maximums

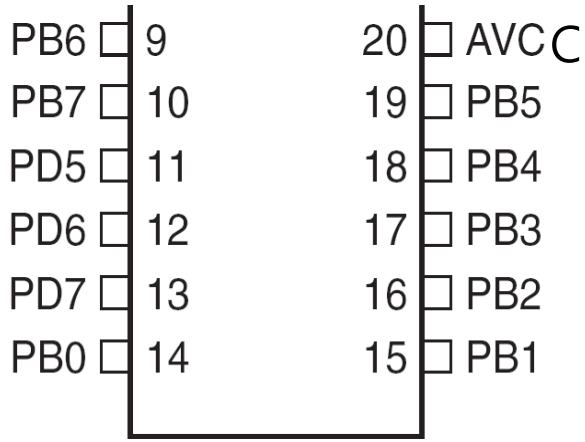
28.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Microcontroller Ports and Pins

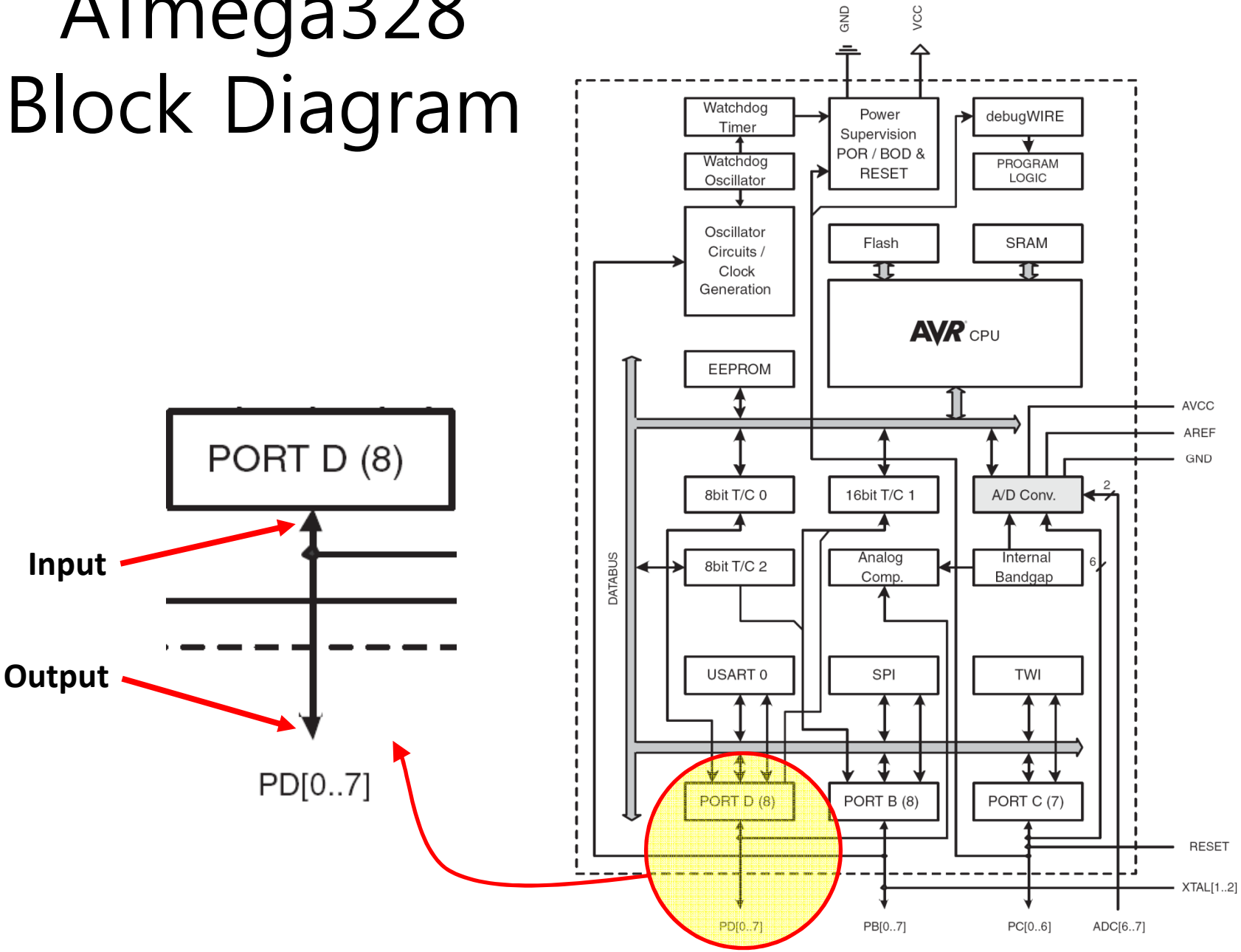
- The communication channels through which information flows into or out of the microcontroller
  - Ex. PORTB
    - Pins PB0 – PB7
      - May not be contiguous
      - Often bi-*directional*



PortPin Data Directionality

- Input
  - When you want to take information from the external world (sensors) ***into*** the MCU
- Output
  - When you want to change the state of something ***out*****side** the MCU (turn a motor on or off, etc.)
- Pins default to input direction on power-up or reset
- Your program can set or change the directionality of a pin at any time

ATmega328 Block Diagram



Input vs. Output

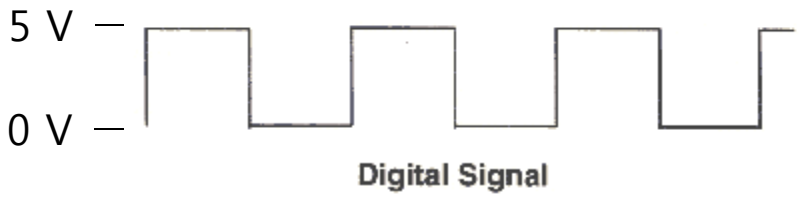
- Referenced from the perspective of the microcontroller (electrical board)
- Input
  - A signal/information going into the board
- Output
  - Any signal exiting the board
- Almost all systems that use physical computing will have some forms of output

Input vs. Output

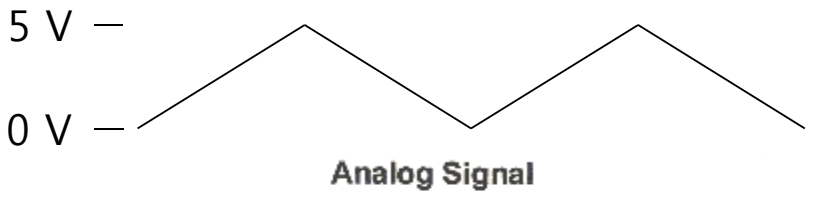
- Input Examples
  - Button Switches
  - Light Sensors
  - Flex Sensors
  - Temperature Sensors
- Output Examples
  - LEDs
  - Servo/DC motor
  - A piezo buzzer
  - relay

Analog vs. Digital

- Microcontrollers are digital/discrete devices
  - On or Off
- Analog signals are anything that can be a full range of values



Digital Signal



Analog Signal

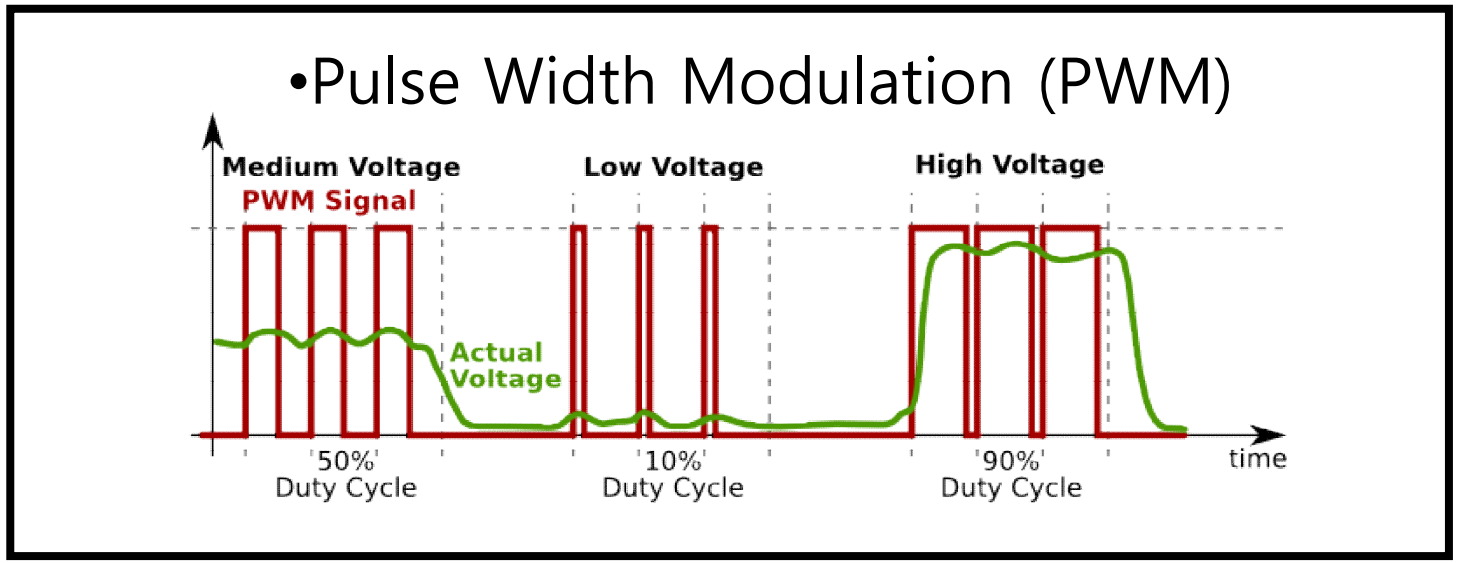
Analog vs. Digital

- A few pins on the Arduino allow for us to modify the output to mimic an analog signal.

- This is done by a technique called:
- Pulse Width Modulation (PWM)

Analog vs. Digital

- To create an analog signal, the microcontroller uses a technique called PWM. By varying the duty cycle, we can mimic an "average" analog voltage.



Setting the Pin Data Direction

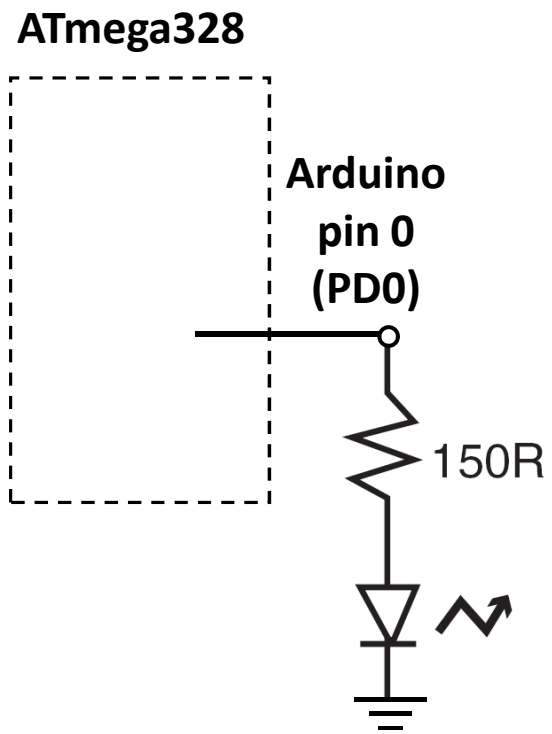
- Arduino
  - pinMode(*pin_no.*, *dir*)
    - Ex. Make Arduino pin 3 (PD3) an *output*
      - **`pinMode(3, OUTPUT);`**
      - **`pinMode(PIN_D3, OUTPUT); // with me106.h`**
  - Note: one pin at a time
    - Suppose you wanted Arduino pins 3, 5, and 7 (PD3, PD5, and PD7) to be outputs?
    - Is there a way to make them all outputs at the same time? → YES

Pin Voltages

- Microcontrollers are fundamentally ***digital*** devices. For digital IO pins:
  - Information is 'coded' in two discrete states:
    - HIGH or LOW (logic: 1 or 0)
    - Voltages
      - TTL
        - » 5 V (for HIGH)
        - » 0 V (for LOW)
      - 3.3 V CMOS
        - » 3.3 V (for HIGH)
        - » 0 V (for LOW)

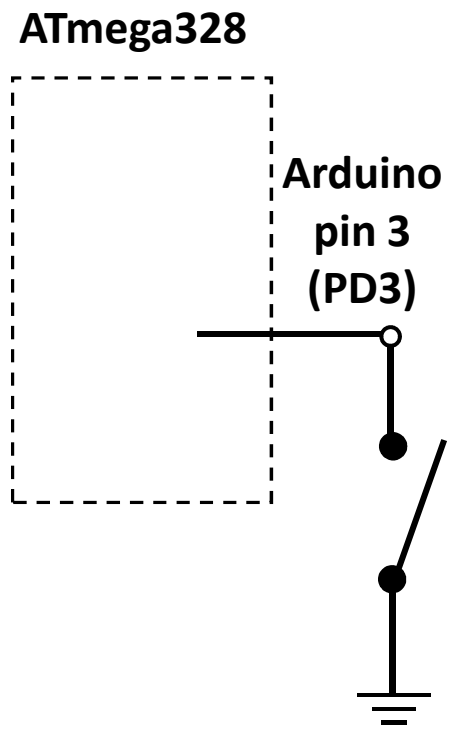
Pin Used as an Output

- Turn on an LED, which is connected to pin Arduino pin 0 (PD0)
  - pin 0 (PD0), OUTPUT
    - **`pinMode( 0 , OUTPUT );`**
  - Turn on the LED
    - **`digitalWrite(0,HIGH);`**
  - Turn off the LED
    - **`digitalWrite(0,LOW);`**



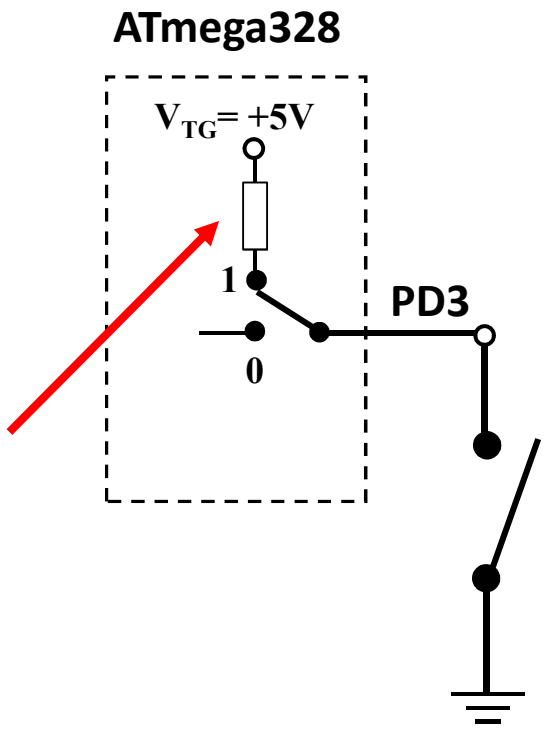
Pins as Inputs and Pull-up Resistors - 1

- Using a switch as a sensor
  - Ex. Seat belt sensor
  - Detect the switch ***state***
    - **`pinMode( 3 , INPUT );`**
    - What will the voltage be on PD3 when the switch is closed?
      - GND
    - What will the voltage be on PD3 when the switch is open?
      - Indeterminate!



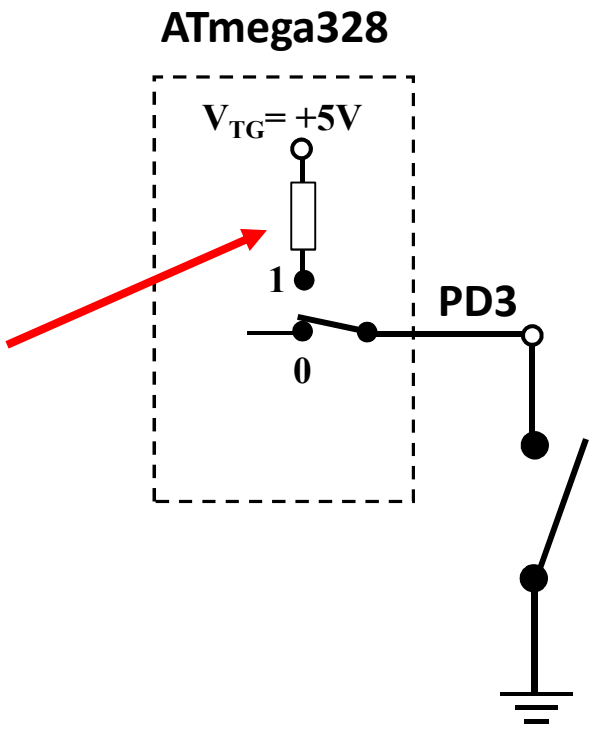
Pins as Inputs and Pull-up Resistors - 2

- Switch as a sensor, cont.
  - Make the voltage on the pin *determinate* by turning on the pull-up resistor for PD3
    - Assuming PD3 is an input:
      - **`digitalWrite(3,HIGH);`** turns on the "pull-up" resistor
      - **`pinMode(3,INPUT_PULLUP);`**
    - What will the voltage on PD3 be when the switch is open?
      - V_{TG}
    - What will the voltage on PD3 be when the switch is closed?
      - Drop to GND



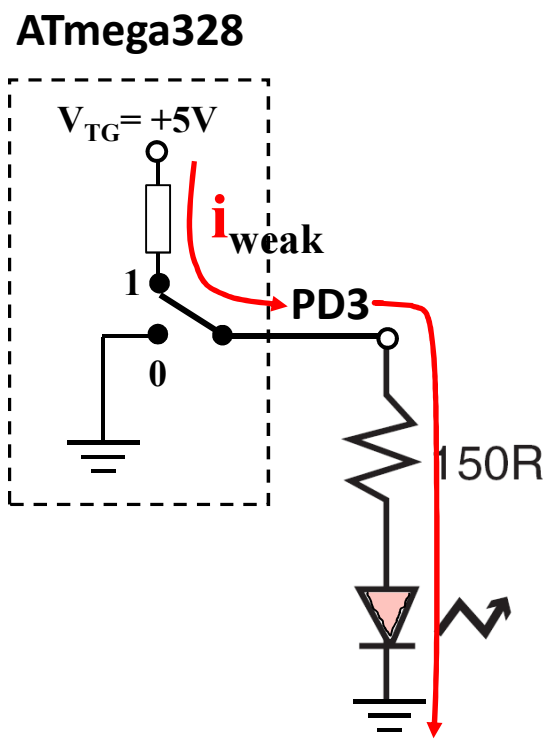
Pins as Inputs and Pull-up Resistors - 3

- Switch as a sensor, cont.
  - To turn *off* the pull-up resistor
    - Assuming PD3 is an input:
      - **`digitalWrite(3,LOW);`** turns the "pull-up" resistor off



Pins as Inputs and Pull-up Resistors - 4

- Possibility of 'weak drive'
  - Pin set as an *input* with a pull-up resistor turned on can source a small current



Setup Internal Pullup Registers

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
  Serial.begin(9600);
  digitalWrite(12, HIGH);
}
```

You can also create internal pullup resistors in setup, to do so digitalWrite the pin HIGH

This takes the place of the pullup resistors.

Handling Data Direction of multiple pins

- All the work of MCU happens through *register s* (special memory locations)
  - Registers on the Atmega328 are 8-bits wide
- The data direction register (DDRx) handles the data directions for pins in PORTx

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	**DDB7**	**DDB6**	**DDB5**	**DDB4**	**DDB3**	**DDB2**	**DDB1**	**DDB0**	**DDRB**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Data Direction Register

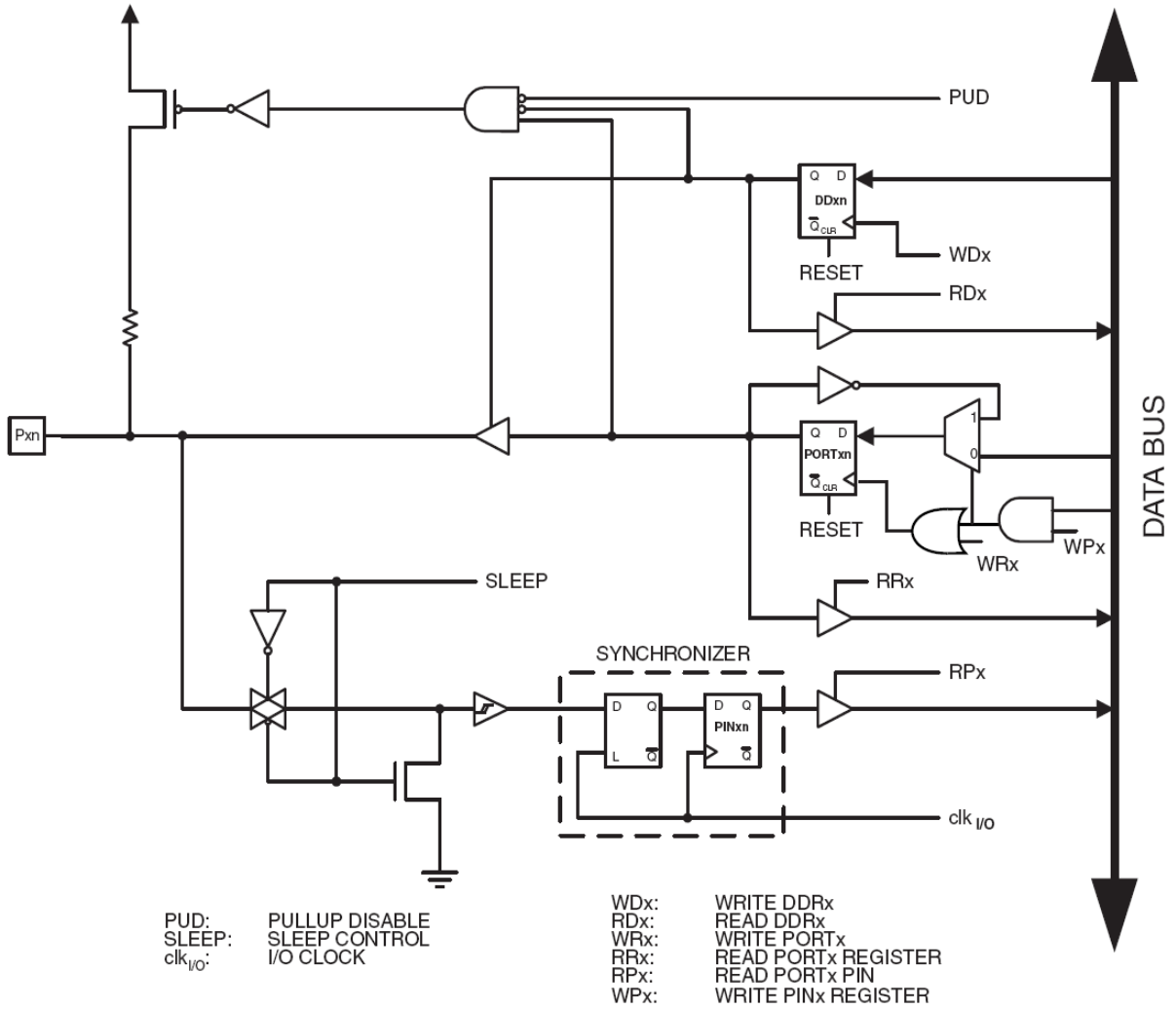
- If the bit is *zero* -> pin will be an *input*
  - Making a bit to be zero == '**clearing** the bit'
- If the bit is *one* -> pin will be an *output*
  - Making a bit to be one == '**setting** the bit'
- To change the data direction for a set of pins belonging to PORTx at the same time:
  1. Determine which bits need to be set and cleared in DDRx
  2. Store the binary number or its equivalent (in an alternate base, such as hex) into DDRx

ATmega328 Registers of Interest

- For digital IO, the important registers are:
  - DDRx
    - Data Direction bit in DDRx register (read/write)
  - PORTx
    - PORTx data register (read/write)
  - PINx
    - PINx register (read only)

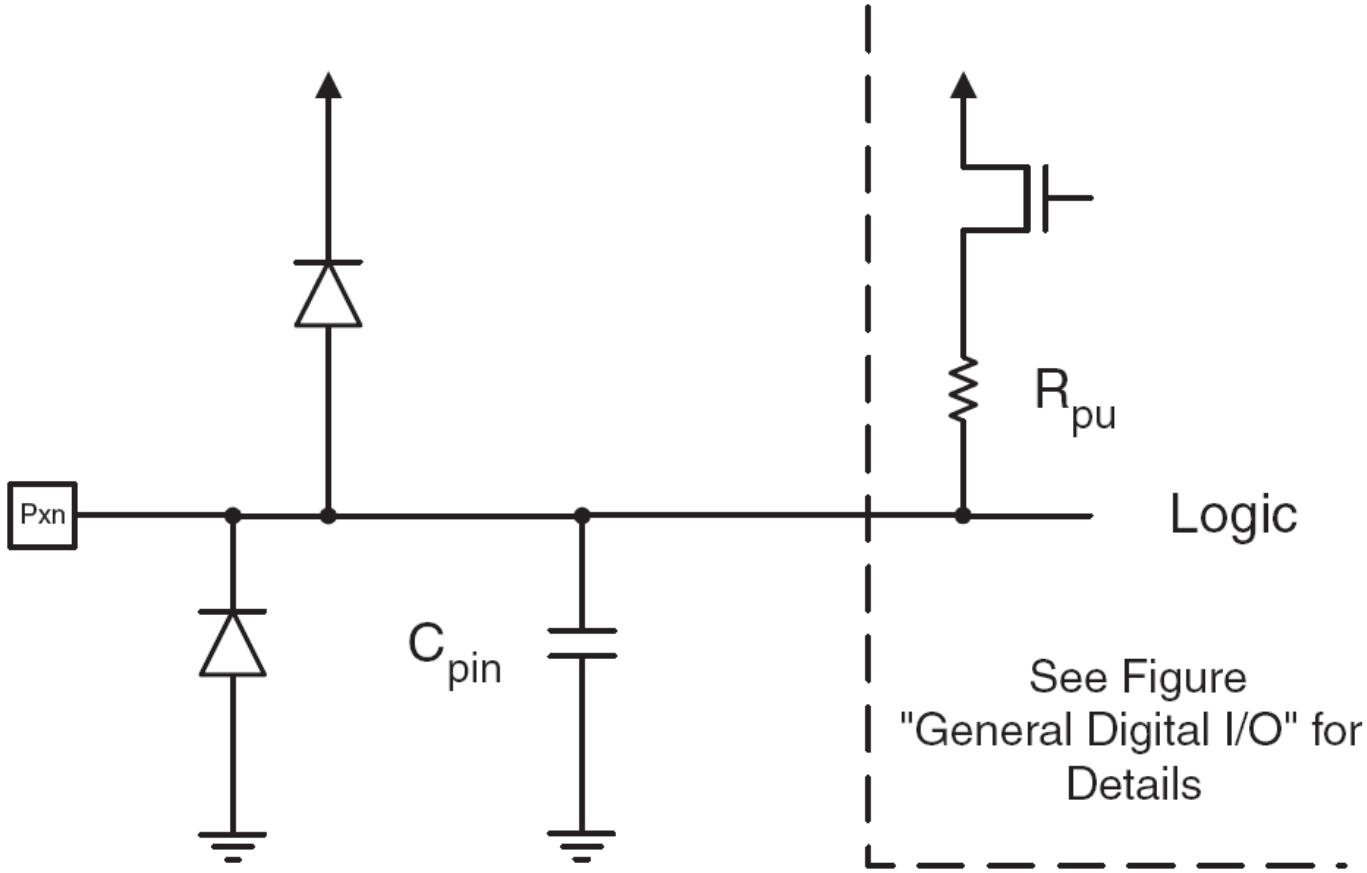
General Digital I/O

Figure 13-2. General Digital I/O[1]



General Digital I/O

Figure 13-1. I/O Pin Equivalent Schematic



Register Details

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

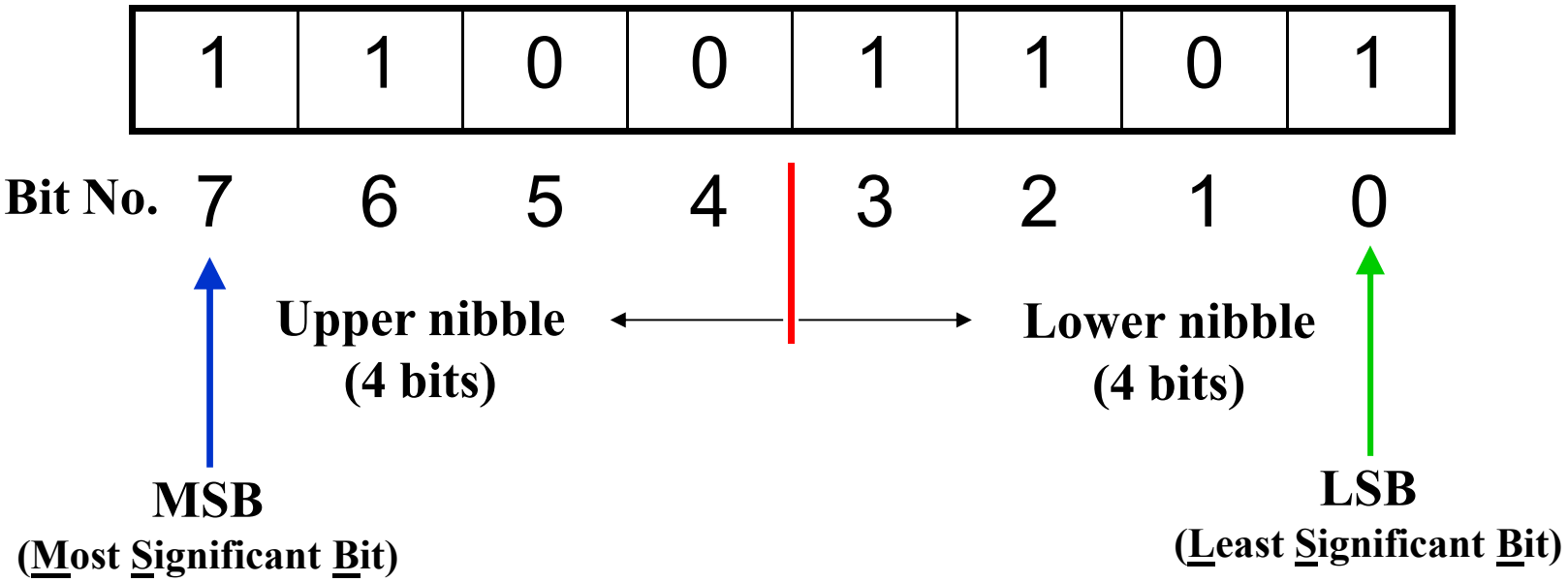
PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Binary and Hexadecimal

- Microcontrollers are fundamentally digital (as opposed to 'analog') and use ***binary*** logic
  - Two states: high and low, 1 or 0, on or off
    - Often 5V or 0V
  - One binary digit is called a ***bit***
    - It can take on two possible states: 1 or 0
  - Eight binary digits are called a ***byte***
  - Four binary digits are called a ***nibble***

Byte and bits



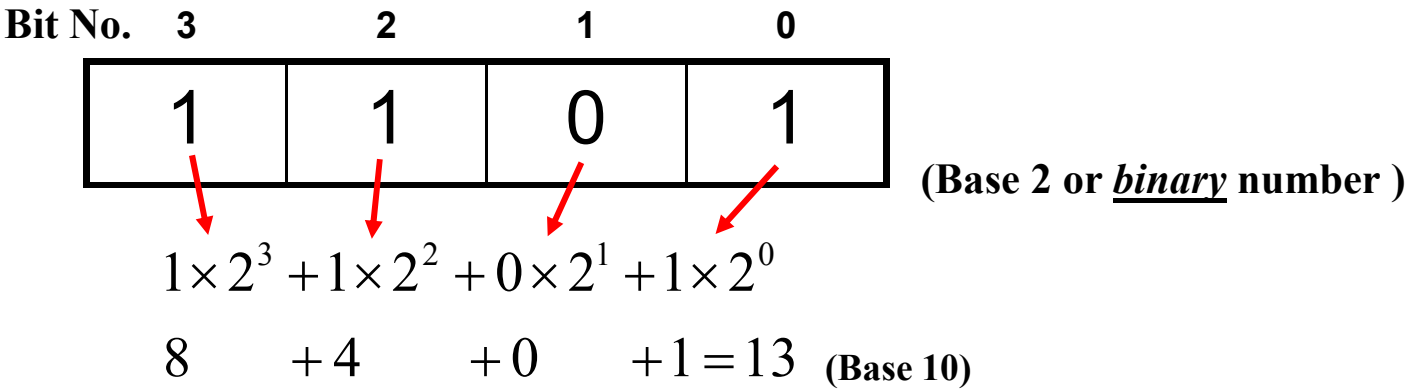
Place value

Place Value

1 1 3 8 (Base 10 or ***decimal*** number)

$1\times10^3+1\times10^2+3\times10^1+8\times10^0$

$1000+100+30+8=1138$ (Base 10)



- **What range of decimal values can 4 bits represent?** 0 to 15
- **How many values in total can 4 bits represent?** 16

Hexadecimal

Binary					HEX
0	0	0	0		**0**
0	0	0	1		**1**
0	0	1	0		**2**
0	0	1	1		**3**
0	1	0	0		**4**
0	1	0	1		**5**
0	1	1	0		**6**
0	1	1	1		**7**
1	0	0	0		**8**
1	0	0	1		**9**
1	0	1	0		**A**
1	0	1	1		**B**
1	1	0	0		**C**
1	1	0	1		**D**
1	1	1	0		**E**
1	1	1	1		**F**

Why is hex important?

One hex digit can be used as *shorthand* to represent four binary digits

Two hex digits can be used as *shorthand* to represent eight binary digits or one byte

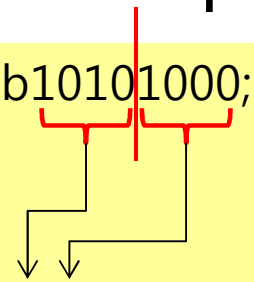
Example 1

- Make Arduino pins 3, 5, and 7 (PD3, PD5, and PD7) to be outputs
- Arduino approach

```
pinMode(3, OUTPUT);
pinMode(5, OUTPUT);
pinMode(7, OUTPUT);
```

- Alternate approach

```
DDRD = 0b10101000;

or

DDRD = 0xA8;

or

DDRD | = 1<<PD7 | 1<<PD5 | 1<<PD3;
```



Example 2

- Make pins Arduino pins 0 and 1 (PD0 and PD1) inputs, and turn on pull-up resistors
- Arduino approach

```
pinMode(0, INPUT);
pinMode(1, INPUT);
digitalWrite(0, HIGH);
digitalWrite(1, HIGH);
```

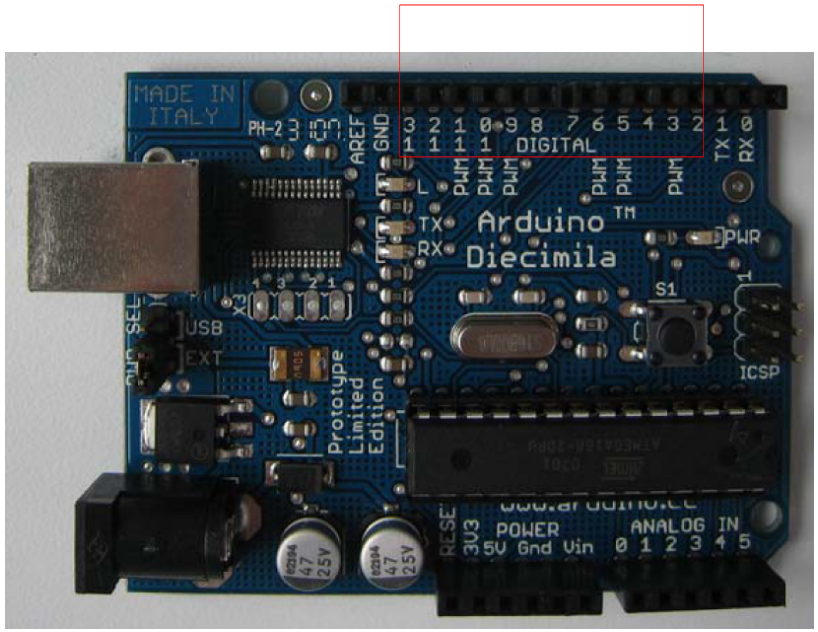
- Alternate approach

```
DDRD = 0; // all PORTD pins input
s
PORTD = 0b00000011;
or
PORTD = 0x03;

or better yet:
```

```
DDRD & = ~(1<<PD1 | 1<<PD0);
PORTD | = (1<<PD1 | 1<<PD0);
```

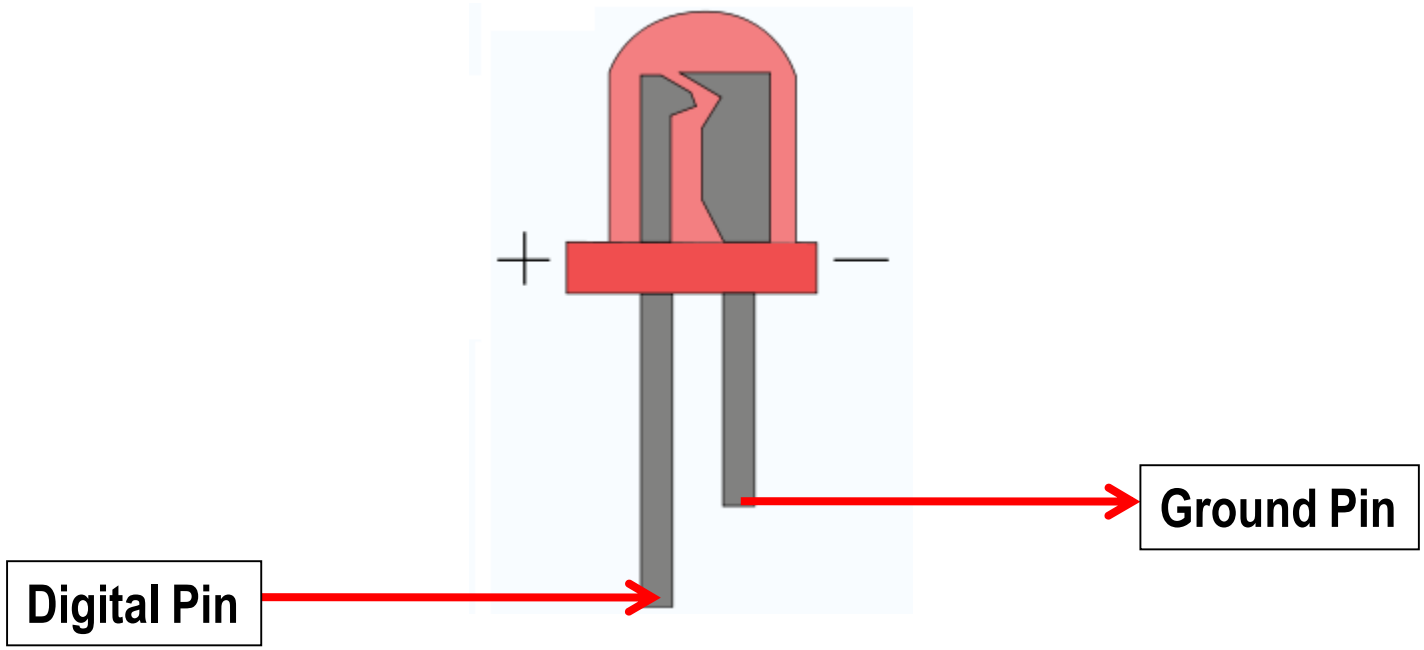
Digital Output

- Digital Out put is defined as sending **on/off** or **0/1** signals from one of the digital pins on the Aurduino board (**pin 2-13**) to the electronic actuator that recognize on/off or 0/1 signal.
- The so-called digital pins are highlighted here.



Digital Output-LED

- LED (Light Emitting Diode) is a light feature that can be used as an actuator of the space.
- Being a Diode, an LED is a directional piece meaning that it is activated only if it is placed in the circuit in the right direction



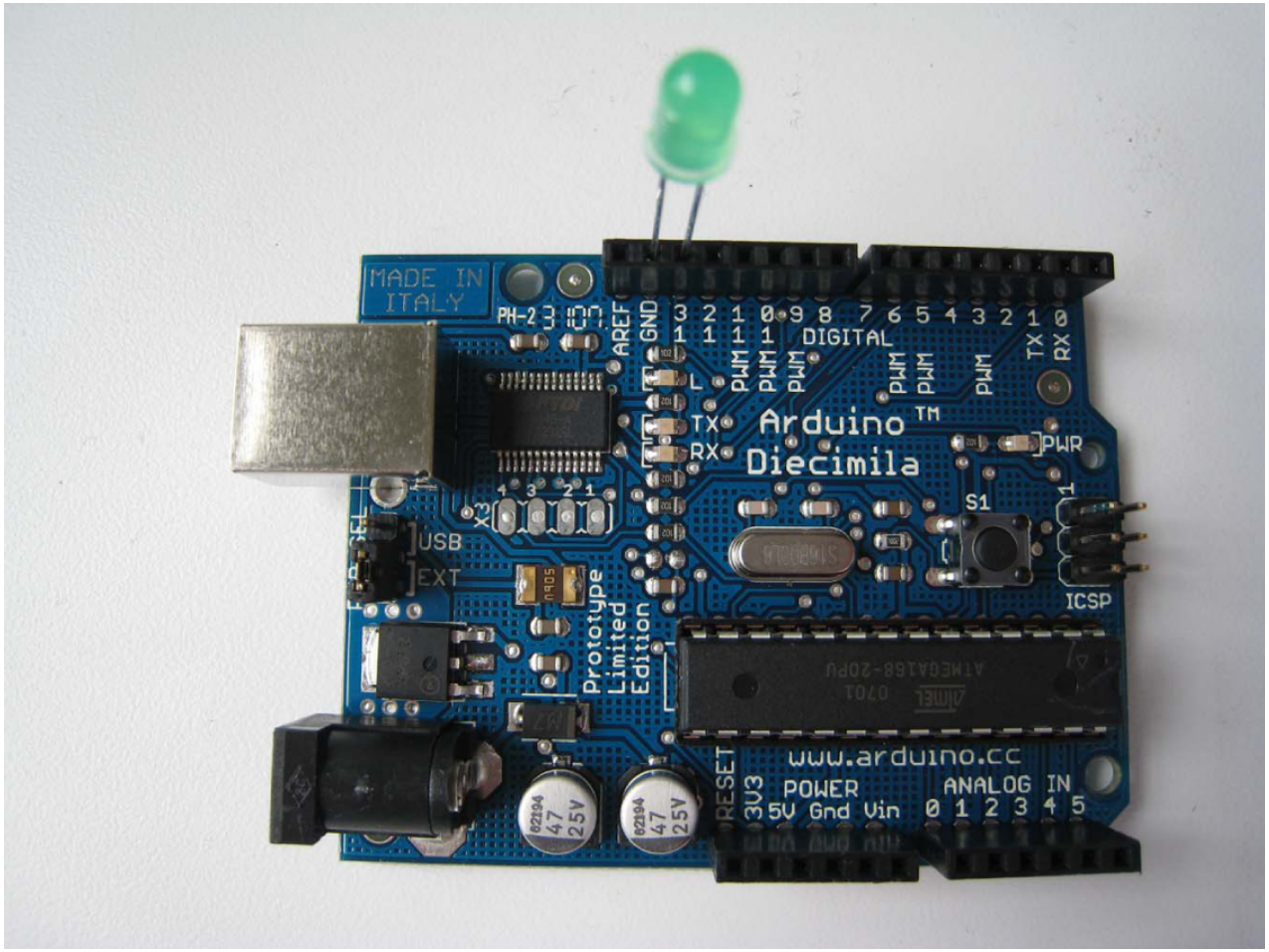
Ground Pin

- For electricity to flow in a circuit, we need difference in level of electricity energy.
- In Arduino board this difference is provided by making a circuit between one of the output pins and ground pin.

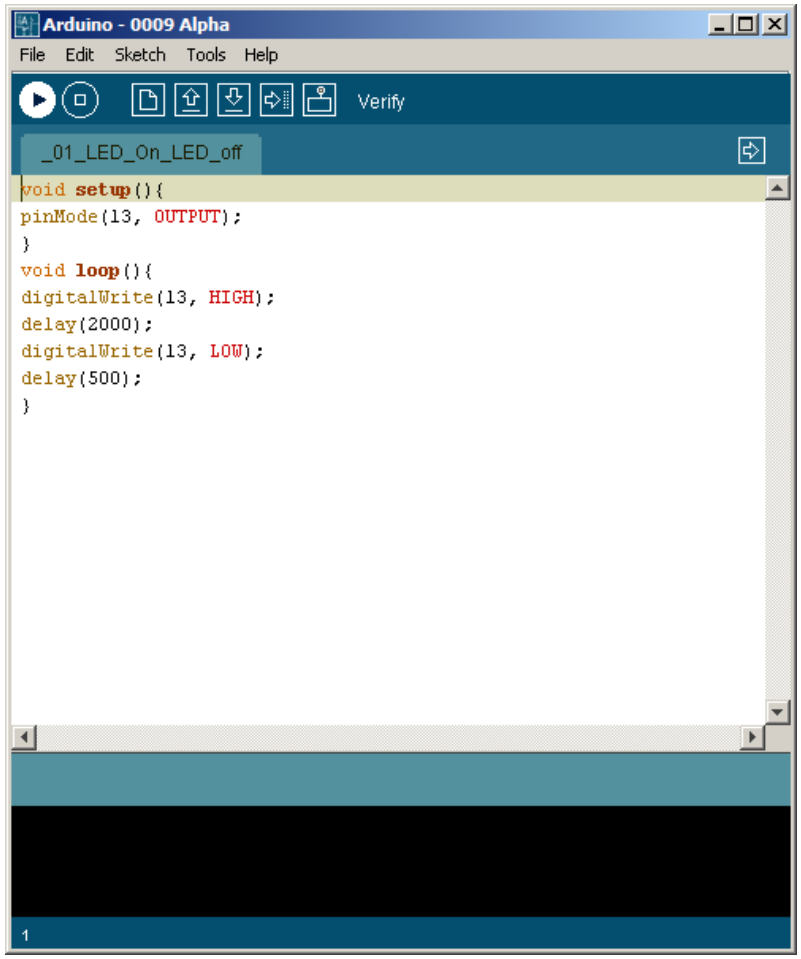
Ground Pin

- When we send a signal through output pin any signal that is not 0 or LOW will provide the desired difference between the two ends of the circuit and will result in electricity flow between the digital output pin and ground pin
- The level of electricity energy at Ground pin is zero, as a result any non zero signal on the digital pin gives us a difference and an electricity flow.
- You can also create this situation using two output pins, one sending the low signal and one sending a high signal. The low signal pin in this case will function as the ground.

Simplest LED circuit with Arduino



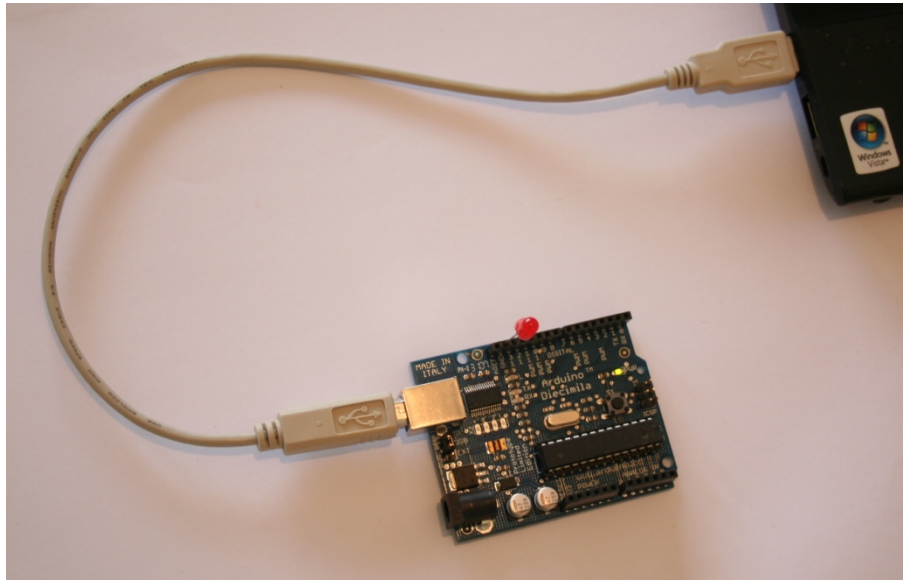
Digital Output LED : Software

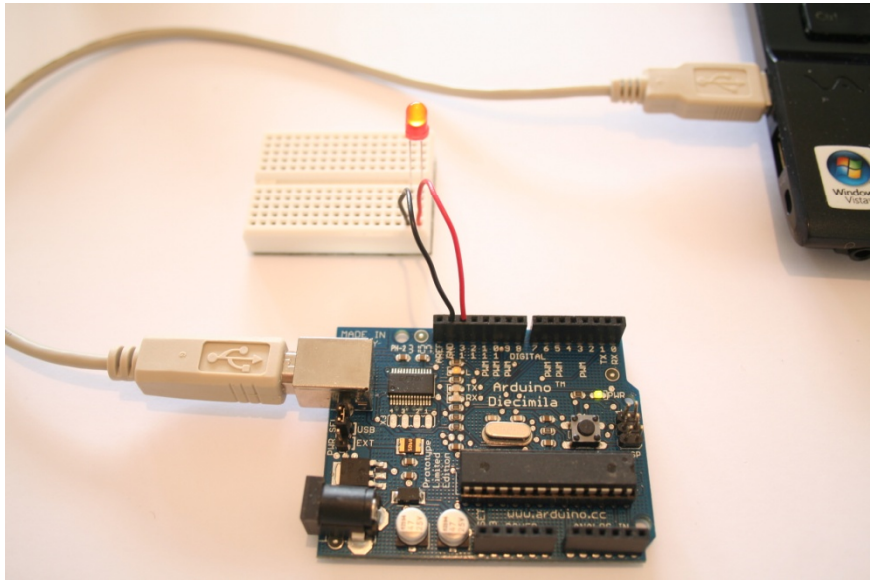



```
void setup(){
pinMode(13, OUTPUT);
}
void loop(){
digitalWrite(13, HIGH);
// delay for 1000 ms
delay(1000);
digitalWrite(13, LOW);
delay(1000);
}
```

1. Write the code
2. Compile the code
3. Check Arduino Port Connection
4. Upload the Code
5. The Arduino and Connected Circuits start to show behavior based on the uploaded code

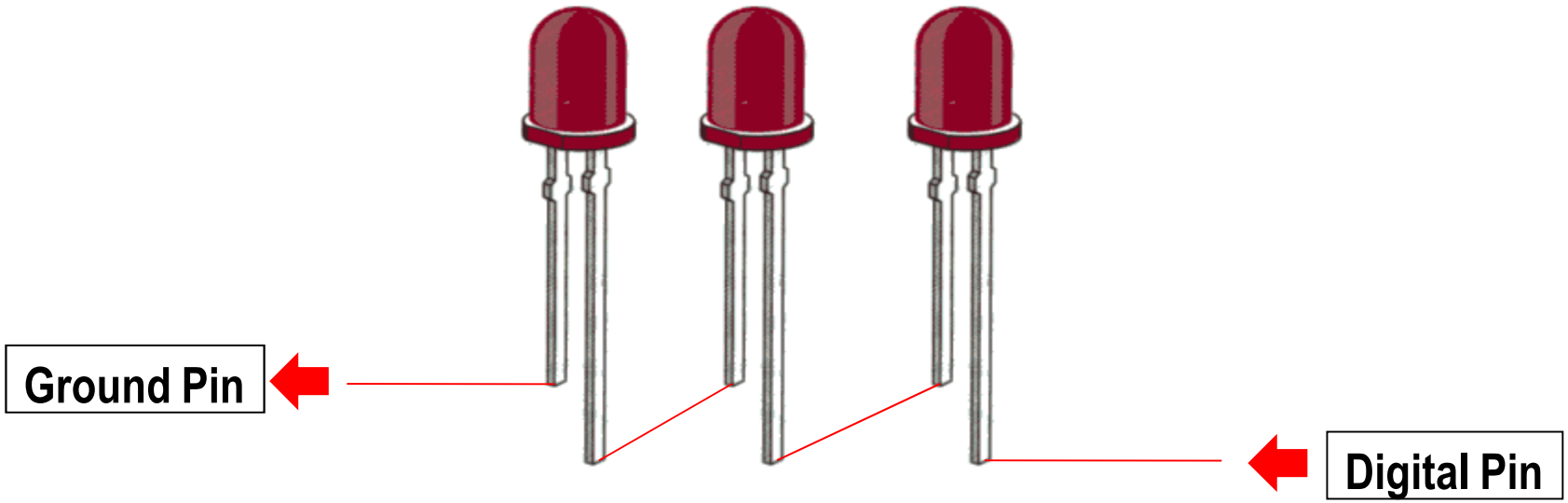
Circuit with Breadboard





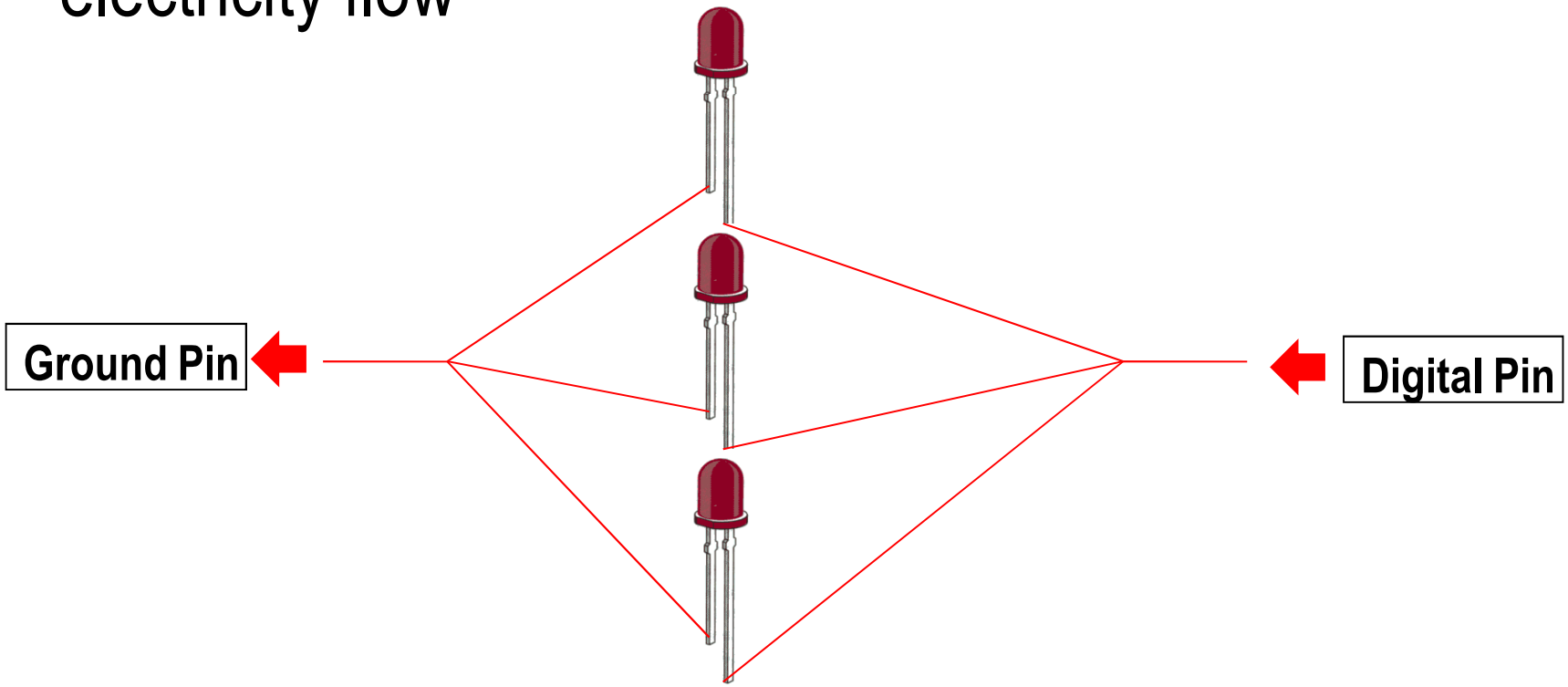
Serial Connection

- In Serial connection, adding more electricity consuming elements results in weaker electricity flow.
- In case of Arduino Board adding more than three High intensity LEDs will result in so weak an electricity flow that the LEDs will not turn on



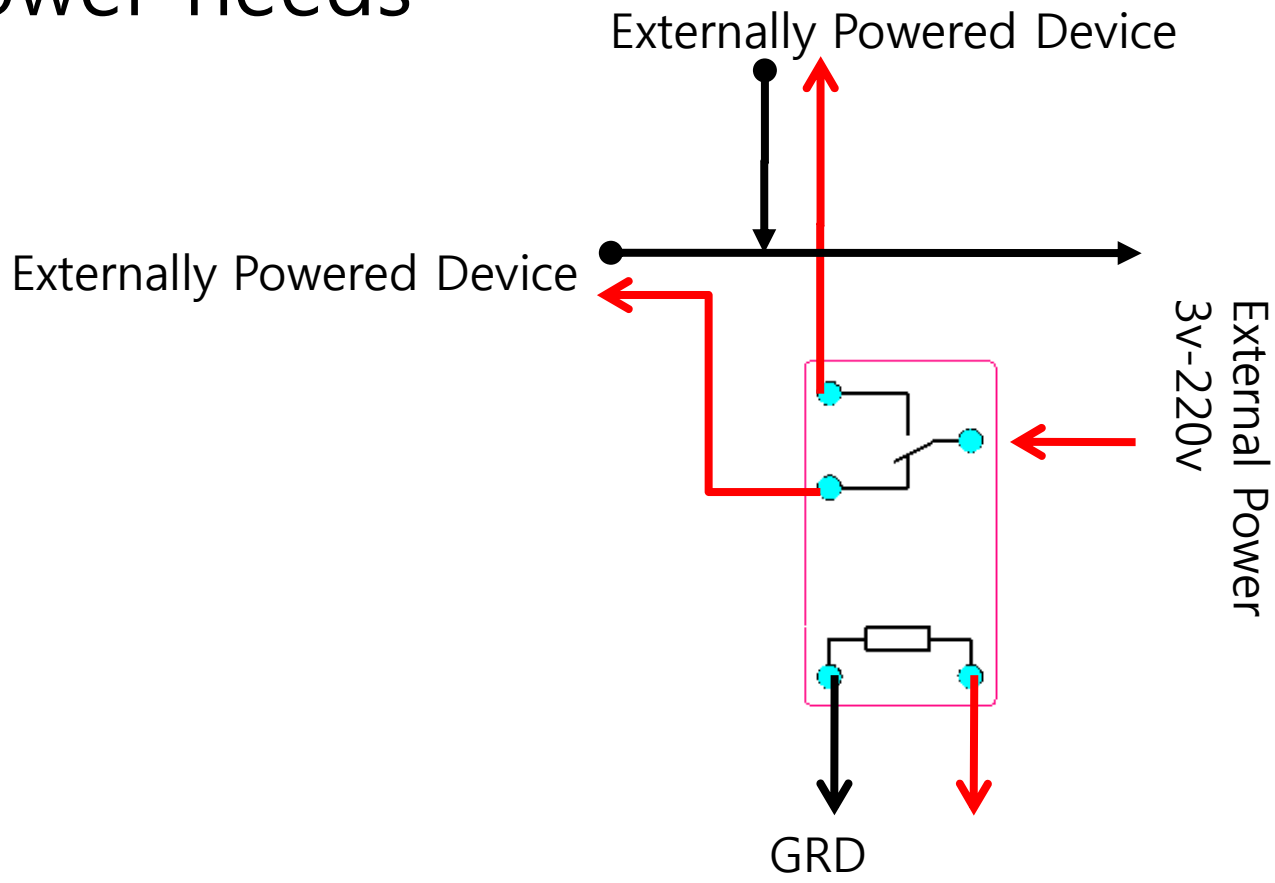
Parallel Connection

- In Parallel connection, adding more electricity consuming elements do not result in decrease of electricity flow



Digital Out with a relay

- Controlling any Electrical Devices with any power needs



Digital Input

- Connect digital input to your Arduino using Pins # 0 – 13
  (Although pins # 0 & 1 are also used for programming)
- Digital Input needs a `pinMode` command:

```
pinMode (pinNumber, INPUT);
```

  *Make sure to use ALL CAPS for **INPUT***

- To get a digital reading:

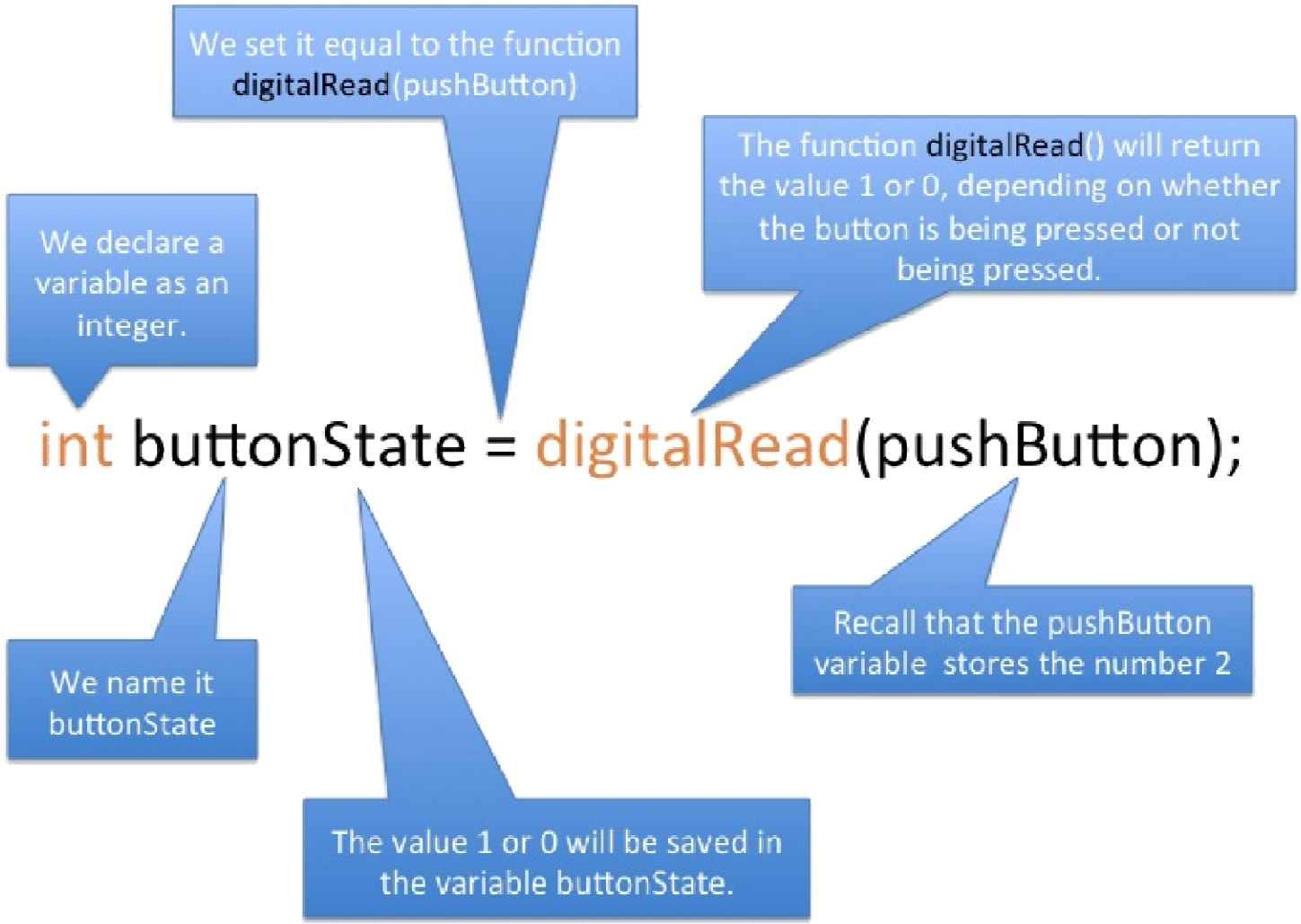
```
int buttonState = digitalRead(pinNumber);
```

- Digital Input values are only **HIGH** (On) or **LOW** (Off)

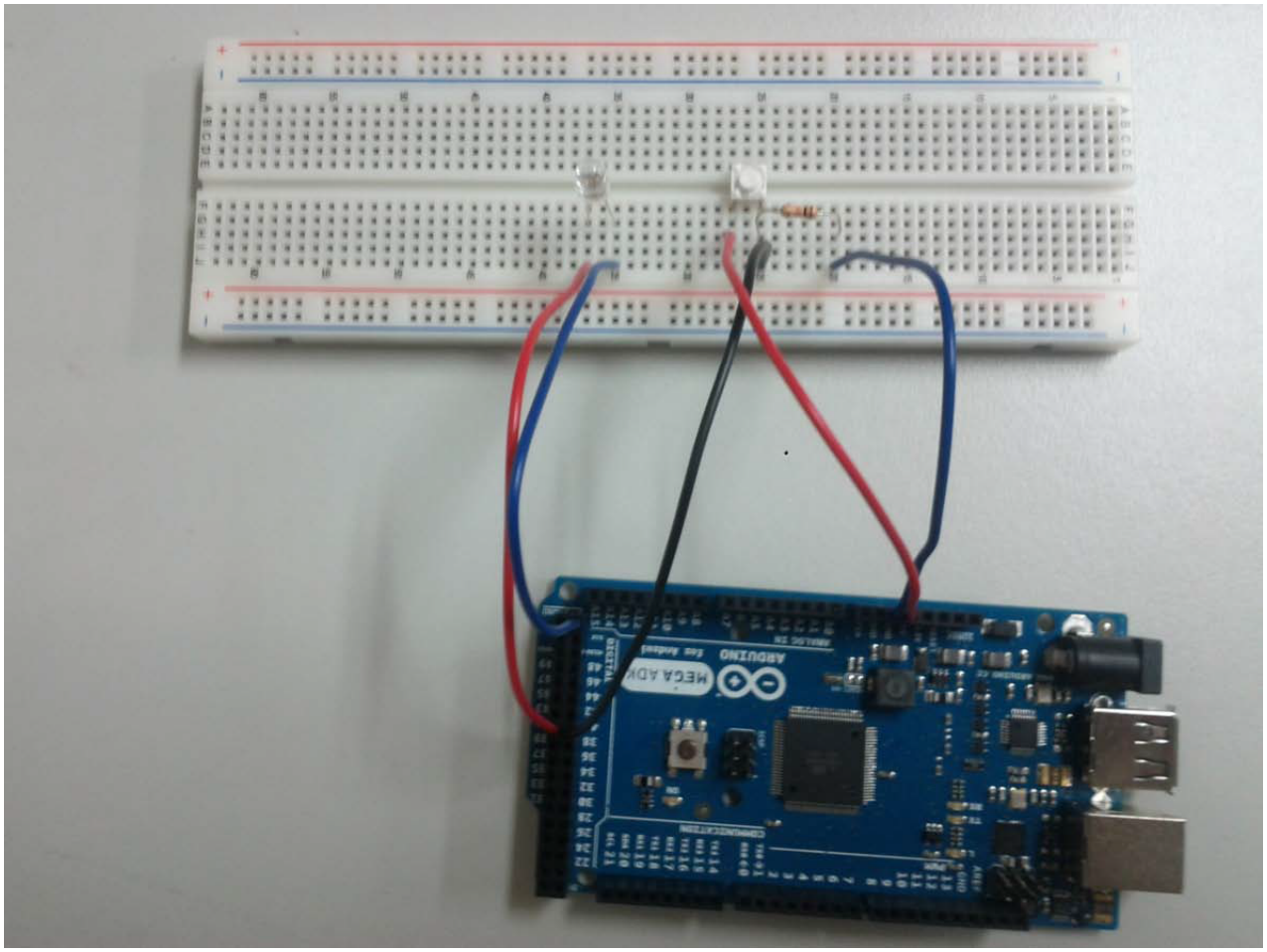
Digital Sensors

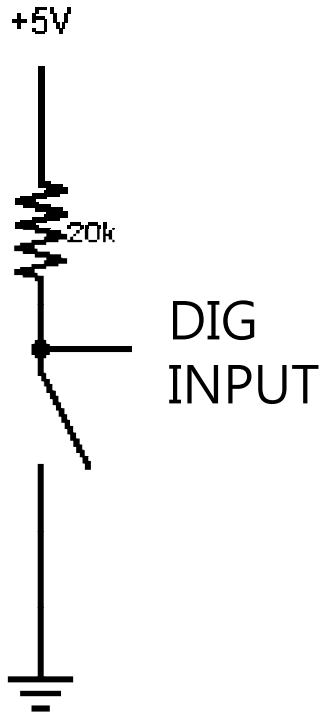
- Digital sensors are more straight forward than Analog
- No matter what the sensor there are only two settings: On and Off
- Signal is always either HIGH (On) or LOW (Off)
- Voltage signal for HIGH will be a little less than 5V on your Uno
- Voltage signal for LOW will be 0V on most systems

DigitalRead



Basic Digital Input Circuit





Digital Input Sketch

```
#define LED 41 // the pin for the LED
#define BUTTON 40 // the input pin where the pushbutton is connected
int val = 0; // val will be used to store the state of the input pin
void setup() {
  pinMode(LED, OUTPUT); // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}
void loop(){
  val = digitalRead(BUTTON); // read input value and store it
  // check whether the input is HIGH (button pressed)
  if (val == HIGH) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Another Digital Input Circuit

```
#define LED 41 // the pin for the LED
#define BUTTON 40 // the input pin where the pushbutton is connected
int val = 0; // val will be used to store the state of the input pin
int state = 0; // 0 = LED off while 1 = LED on
void setup() {
  pinMode(LED, OUTPUT); // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}
void loop() {
  val = digitalRead(BUTTON); // read input value and store it
  // check if the input is HIGH (button pressed) and change the state
  if (val == HIGH) {
    state = 1 - state;
  }
  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Any Problem??

Problems of the code

- Signal bouncing
  - Because of incomplete physical operation of the button, signal is not constant but bouncing
- Loop is too fast
  - Multiple digitalRead operations
  - Can not make sure “when” the digitalRead function get the input

Improved Code

```
int val = 0; // val will be used to store the state of the input pin
int old_val = 0; // this variable stores the previous value of "val"
int state = 0; // 0 = LED off and 1 = LED on

void loop(){
  val = digitalRead(BUTTON); // read input value and store it yum, fresh
  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)){
    state = 1 - state;
  }
  old_val = val; // val is now old, let's store it
  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
  delay(200)
}
```

Approach of the improved code

- Only triggered with “rising” signal
  - Old is low, new is high
- Make slower the loop
  - Delay 200 ms
  - Eliminating the effect of bounced signal

Implementation of Double Click

```
void loop(){
  val = digitalRead(BUTTON); // read input value and store it yum, fresh
  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)){
    button_counter++;
    startTime = millis();    // get the system clock
  }
  old_val = val; // val is now old, let's store it

  if( startTime + interval < millis() ){
    if (button_counter == 1) {
      digitalWrite(LED, HIGH); // turn LED ON
    } if( button_counter == 2 ){
      digitalWrite(LED, LOW); // turn LED OFF
    }
    button_counter = 0;
  }
}
```

Any Problem??

Problems of the code

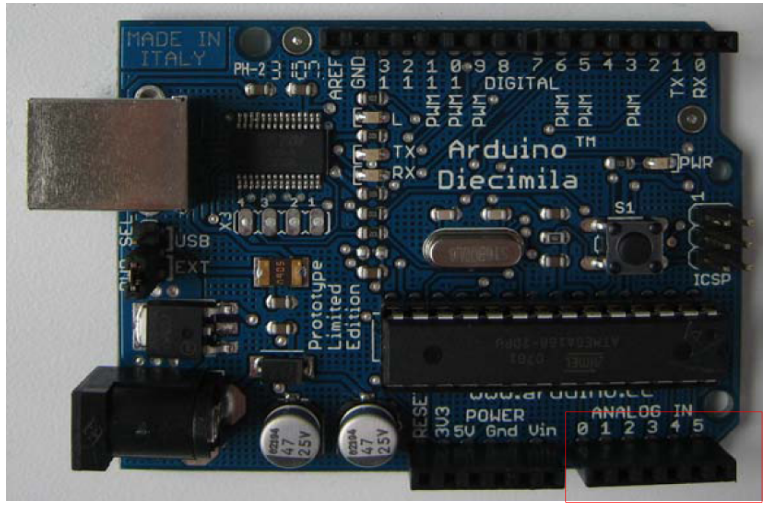
- if(startTime + interval < millis())
  - This statement happens for every loop
  - Must be the overhead
- Solution?
  - Use one more flag for indicating that the button is pushed at least one time

Improved code

```
void loop(){
  val = digitalRead(BUTTON); // read input value and store it yum, fresh
  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)){
    button_counter++;
    bt_flag = 1;
    startTime = millis();    // get the system clock
  }
  old_val = val; // val is now old, let's store it

  if( bt_flag == 1 && startTime + interval < millis() ){
    if (button_counter == 1) {
      digitalWrite(LED, HIGH); // turn LED ON
    } if( button_counter == 2 ){
      digitalWrite(LED, LOW); // turn LED OFF
    }
    button_counter = 0;
    bt_flag = 0;
  }
}
```

Analog Input

- Analog In 0,1,2,3,4,5 are used for Analog Input
- Arduino uses a 10-bit A/D Converter
  - this means that you get input values from 0 to 1023
    - 0 V → 0
    - 5 V → 1023



Testing code for Analog Input

- Use the same circuit for Analog Output
  - We are going to use light sensor analog input

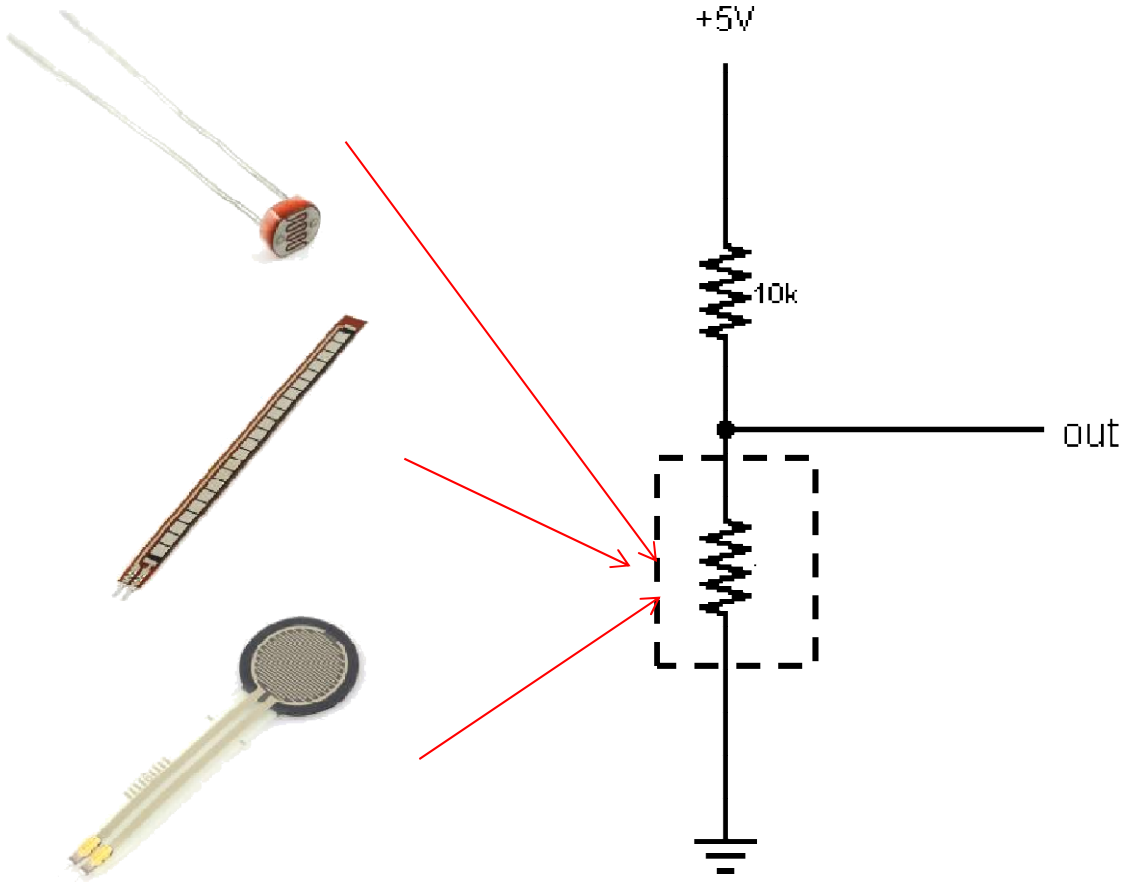
```
#define LEDa 2
#define LED 40 // the pin for the LED
int val = 0; // variable used to store the value coming from the sensor
void setup() {
  pinMode(LED, OUTPUT); // LED is as an OUTPUT
  // Note: Analogue pins are automatically set as inputs
}
void loop() {
  val = analogRead(0); // read the value from the sensor
  digitalWrite(LED, HIGH); // turn the LED on
  delay(val); // stop the program for some time
  digitalWrite(LED, LOW); // turn the LED off
  delay(val); // stop the program for some time
  analogWrite(LEDa, val/4);
}
```

Blinking frequency changes whenever the light sensor is covered by objects

Analog Sensors

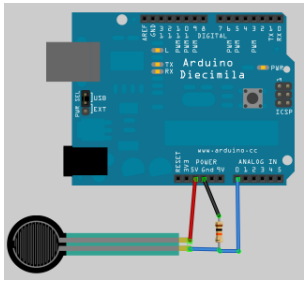
Sensors	Variables
Mic	soundVolume
Photoresistor	lightLevel
Potentiometer	dialPosition
Temp Sensor	temperature
Flex Sensor	bend
Accelerometer	tilt/acceleration

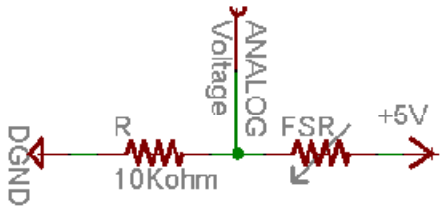
Analog Sensors :
2 Pin Analog Sensors = var. resistor



Analog Sensors: Force Sensitive Resistor (FSR)

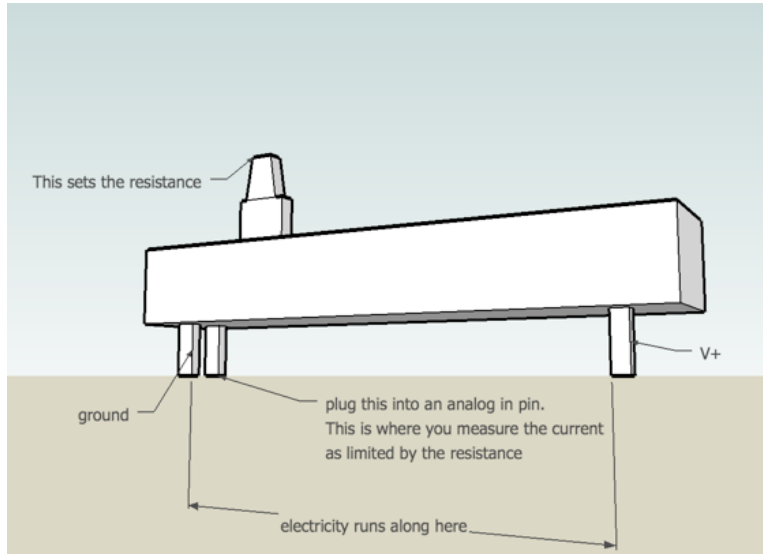
- FSRs are sensors that allow you to detect physical pressure, squeezing and weight.
- FSR is basically a resistor that changes its resistive value (in ohms Ω) depending on how much its pressed.
- While FSRs can detect weight, they're a bad choice for detecting exactly how many pounds of weight are on them.
  - However, for most touch-sensitive applications like "has this been squeezed or pushed and about how much" they're a good deal for the money!





Analog Sensors: Slide Potentiometer

- Slide Potentiometer is basically a variable resistance with a tactile interface.
  - It can be used to detect linear disposition.
- Ypu need a pull down resistor (10K) while connecting the Potentiometer to ground.
- Connecting the sensor to 5V or 3V changes the range of the sensibility of the sensor.



Analog Sensors: PIR Motion Sensor

- This is a simple to use motion sensor. Power it up and wait 1-2 seconds for the sensor to get a snapshot of the still room.
  - If anything moves after that period, the 'alarm' pin will go low.
- The alarm pin is an open collector meaning you will need a pull up resistor on the alarm pin.
- Take note that by default the sensor read is 1023 and once motion is detected it alternates between 1023 and 18.

Analog Sensors: Distance Sensor

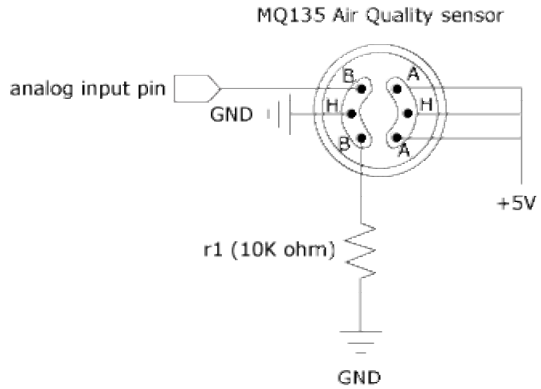
- GP2D12 Analog Distance Sensor
  - Black to Ground, Red to 5V and Yellow to Analog Pin.
- The one that you have is sensitive between 10-80cm.

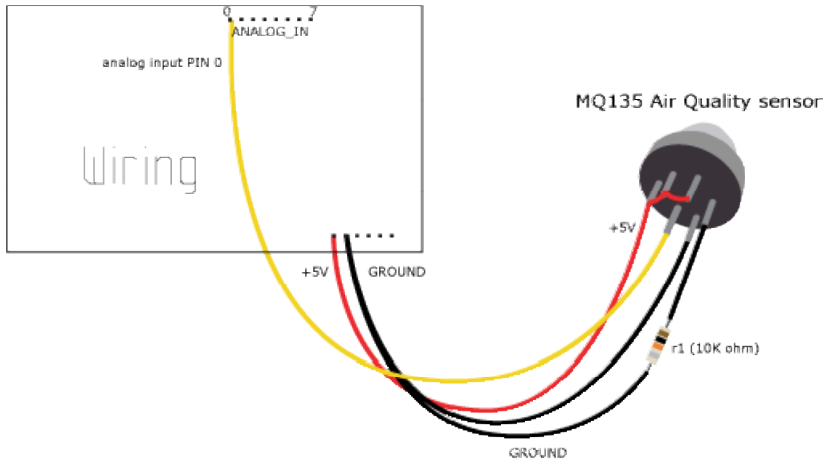
Analog Sensors: Distance Sensor

- IR Distance Sensor
  - Detecting objects between 2 and 10 cm (0.8" and 4") away.
  - With its quick response time, small size, and low current draw, this sensor is a good choice for non-contact object detection, and our compact carrier PCB makes it easy to integrate into your project.
  - The Power is 5V.

Analog Sensors: Air Quality Sensor

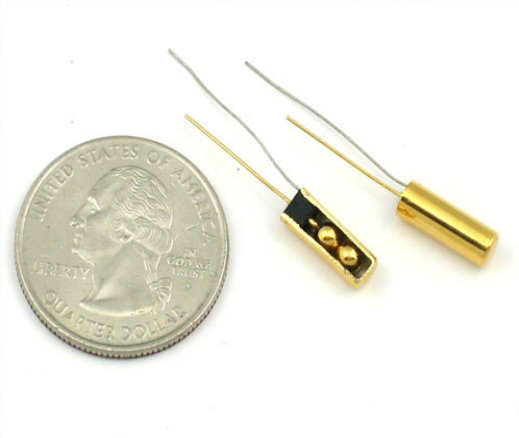
- MQ7 Air Quality Sensor
  - A simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air.
  - Detecting CO concentrations anywhere from 20 to 2000ppm.
- The sensor's output is an analog resistance.
  - The drive circuit is very simple; all you need to do is power the heater coil with 5V, add a load resistance, and connect the output to an ADC.
  - High sensitivity and fast response time.

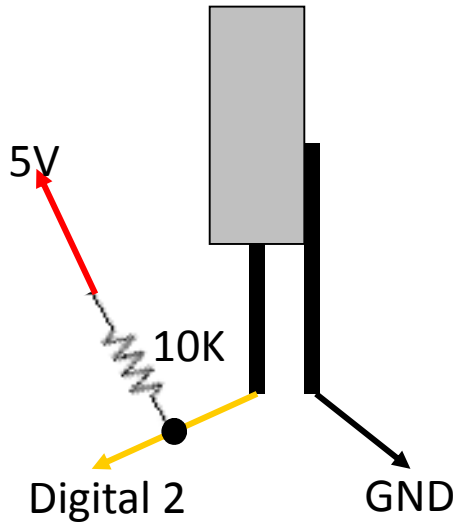




Digital Sensor: Tilt Ball Switch Angle

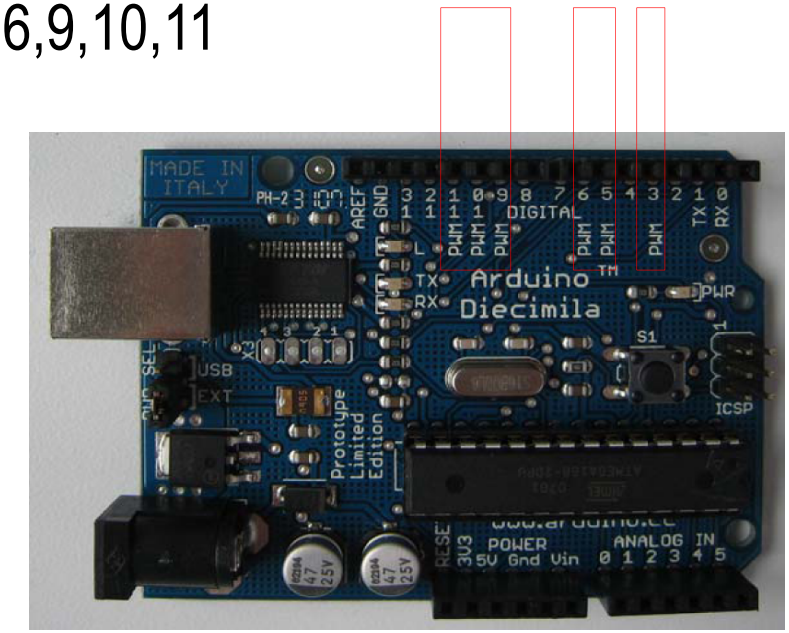
- The "poor man's" accelerometer!
- Tilt sensors are switches that can detect basic motion/orientation.
  - The metal tube has a little metal ball that rolls around in it, when its tilted upright, the ball rolls onto the contacts sticking out of end and shorts them together



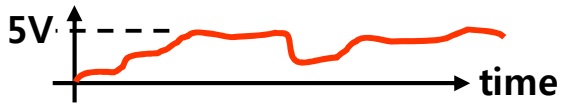


Analog Output

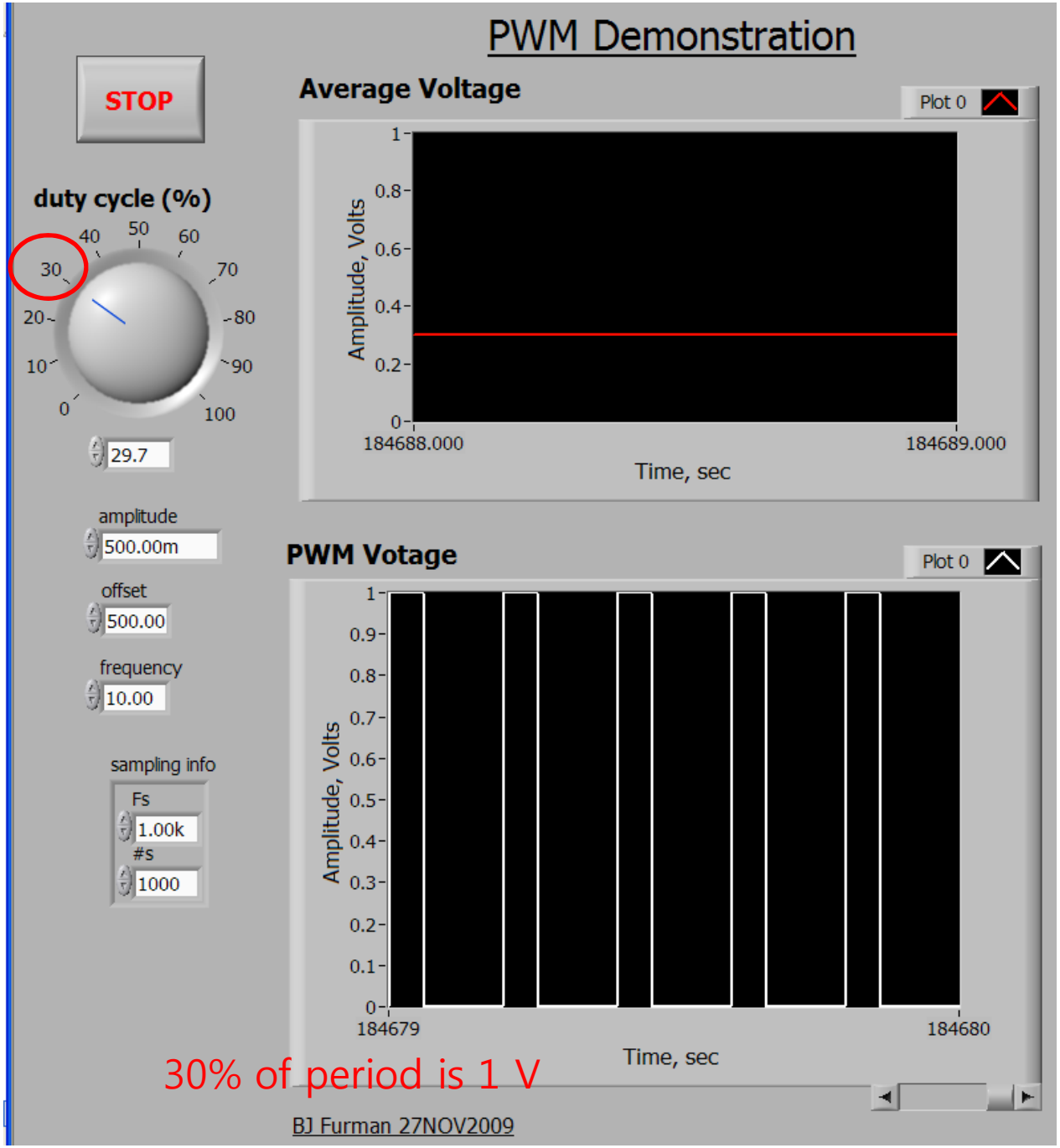
- Analog Out put is defined as sending signals from one of the digital pins on the Aurduino board that range between two extremes: **0-255**
- Out of 13 Digital pins on Arduino board the following pins can be used to signal out Analog output: 3,5,6,9,10,11
- These are the pins with PWM label next to them on the board



Analog Out (PWM) Concept



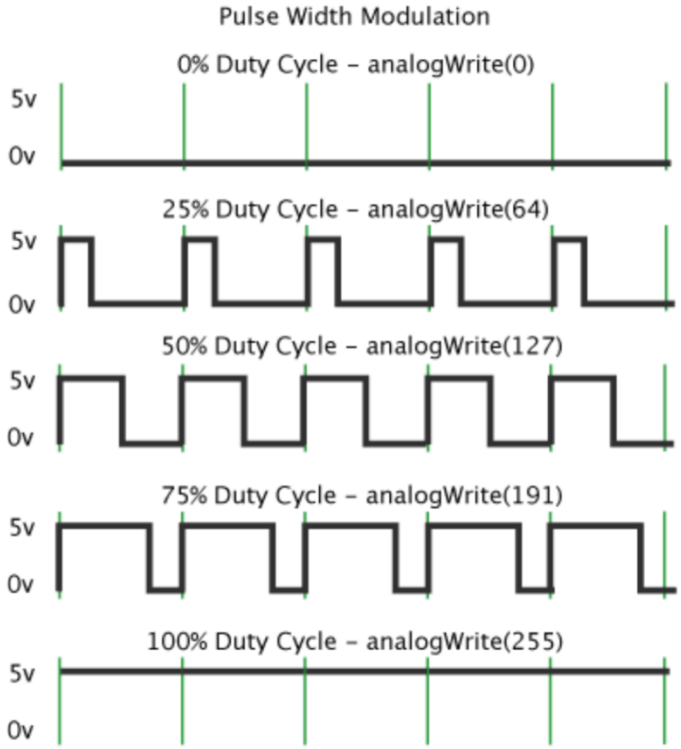
- No facility exists on most microcontrollers to *directly* output an analog voltage (i.e., a voltage that varies continuously over the range of 0 to 5V)
  - Use Pulse Width Modulation (PWM) to approximate
    - Digital outputs are capable of 0V or 5V
    - Over a fraction (t_{on}) of a time period t_{cycle}, keep pin at 5V, the rest of the time, at 0V
      - Average voltage is proportional to t_{on}/t_{cycle}, called the 'Duty Cycle'


PWM Demonstration
STOP
duty cycle (%)
0
10
20
30
40
50
60
70
80
90
100
29.7
amplitude
500.00m
offset
500.00
frequency
10.00
sampling info
Fs
1.00k
#s
1000
Average Voltage
Plot 0
Amplitude, Volts
1
0.8
0.6
0.4
0.2
0
184688.000
184689.000
Time, sec
PWM Votage
Plot 0
Amplitude, Volts
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
184679
184680
Time, sec
30% of period is 1 V
BJ Furman 27NOV2009

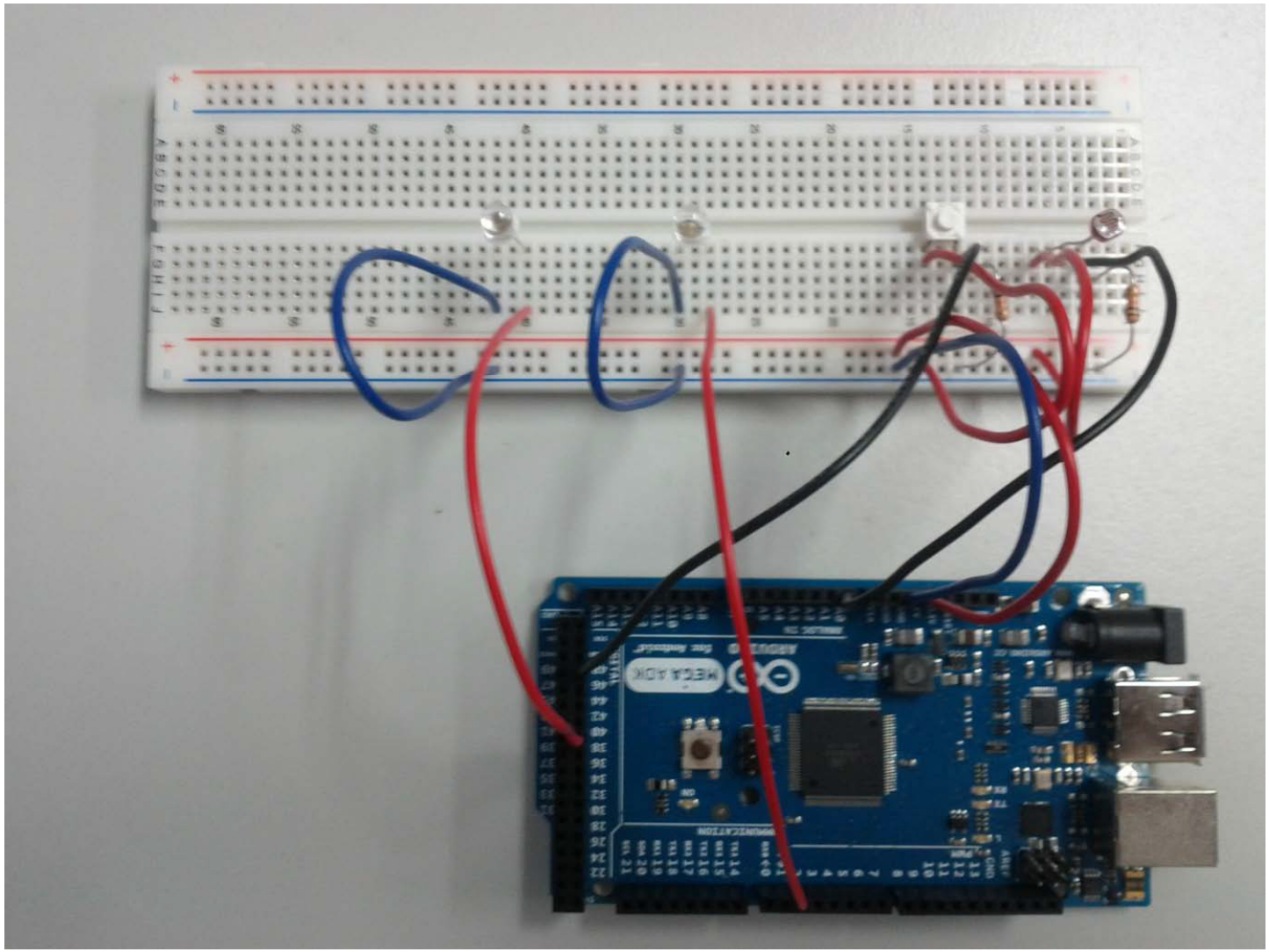
0.3 v

analogWrite()

- **`analogWrite`**(*pin*, ***value***);
  - 0 ≤ ***value*** ≤ 255
    - 0% duty cycle --> 0 V
      - **`analogWrite`**(pin, 0);
    - 100% duty cycle --> 5 V
      - **`analogWrite`**(pin, 255);



Analog Out Circuit



Fade in and Fade out LED

```
#define LED 2 // the pin for the LED
int i = 0; // We'll use this to count up and down
void setup() {
  pinMode(LED, OUTPUT); // tell Arduino LED is an output
}
void loop(){
  for (i = 0; i < 255; i++) { // loop from 0 to 254 (fade in)
    analogWrite(LED, i); // set the LED brightness
    delay(10); //Wait 10ms because analogWrite is instantaneous and
    //we would not see any change
  }

  for (i = 255; i > 0; i--) { // loop from 255 to 1 (fade out)
    analogWrite(LED, i); // set the LED brightness
    delay(10); // Wait 10ms
  }
}
```

Emulating PWM with Digital Out

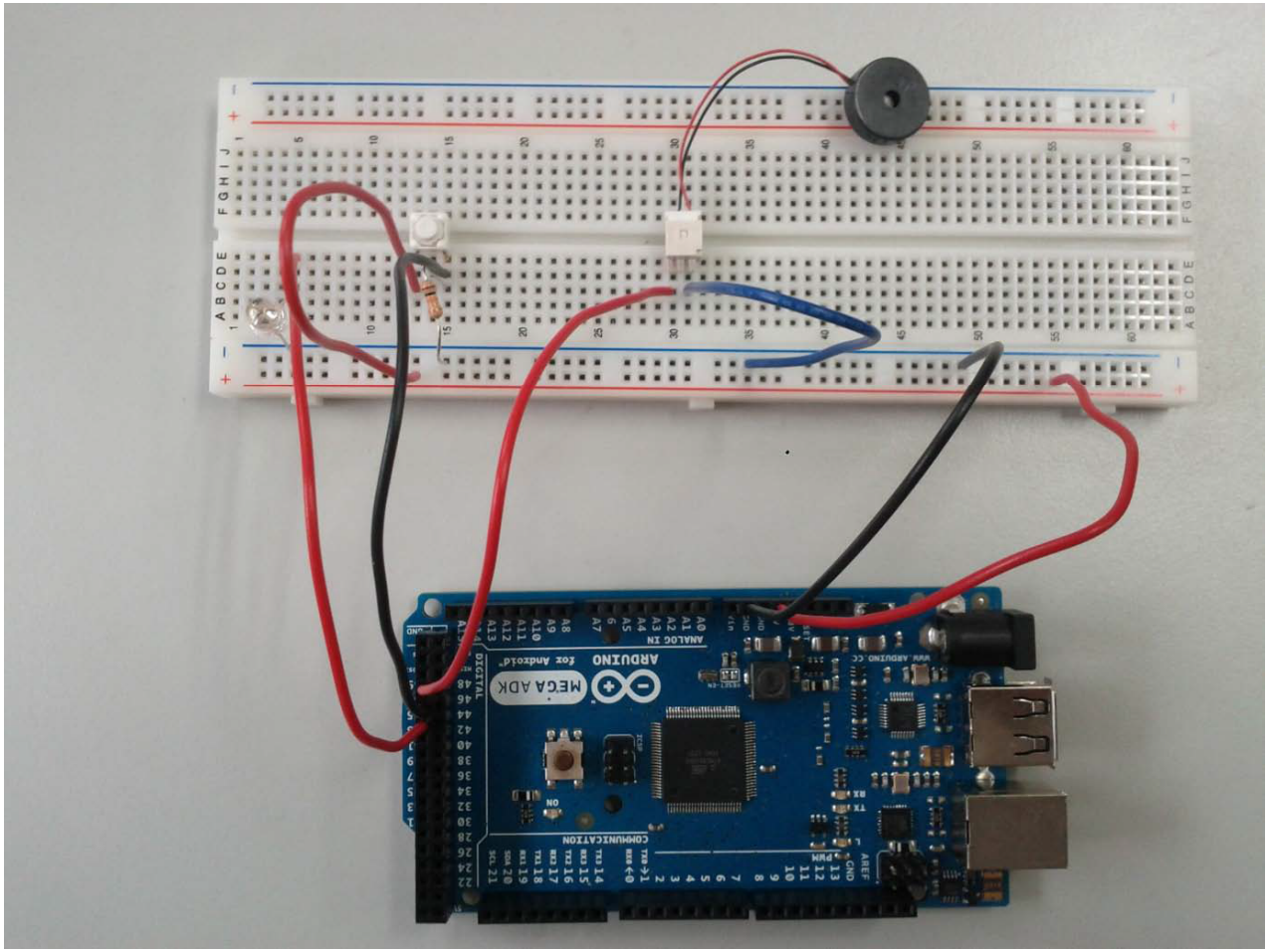
```
void ledOn(int duty){
  float tmp = (float)period * (float)duty / (float)100;

  if( tmp == 0 ){
    digitalWrite(LED, LOW);
  } else {
    digitalWrite(LED, HIGH);
    delayMicroseconds((int)tmp);
    digitalWrite(LED, LOW);
    delayMicroseconds(period - (int)tmp);
  }
}
```

Where we can use this?
➔ Piezo Sound, Servo Motor Control

Piezo

- A Piezo is an electronic piece that converts electricity energy to sound.
  - It is a digital output device.
  - You can make white noise or even exact musical notes (frequencies for musical notes) based on the duration that you iterate between HIGH and LOW signals.
- A Piezo is a directional piece, meaning that it has a positive and negative pole.
  - The positive pole should be connected to the digital output pin that you allocate to control the piezo and the negative pole should be connected to Ground pin

Piezo circuit



Playing piezo with Digital Output

```
#define speakerOut 49
#define c4 3830 // 261 Hz
#define d4 3400 // 294 Hz
#define e4 3038 // 329 Hz
#define f4 2864 // 349 Hz
#define g4 2550 // 392 Hz
#define a4 2272 // 440 Hz
#define b4 2028 // 493 Hz
#define c5 1912 // 523 Hz
int tone_ = c4;
void setup(){
  pinMode(speakerOut, OUTPUT);
}
void loop(){
  digitalWrite(speakerOut,HIGH);
  delayMicroseconds(tone_ / 2);
  digitalWrite(speakerOut,LOW);
  delayMicroseconds(tone_ / 2);
}
```

Playing a melody

```
int melody[] = {c4, d4, e4, f4, g4, a4, b4, c5};
int tone_ = c4;
long duration = 200000;
long pause = 10;

void loop(){
  long elapsed_time = 0;
  for( int i = 0; i < 8; i++ ){
    tone_ = melody[i];
    elapsed_time = 0;
    while( elapsed_time < duration ){
      if( elapsed_time + tone_ <= duration ){
        digitalWrite(speakerOut,HIGH);
        delayMicroseconds(tone_ / 2);
        digitalWrite(speakerOut,LOW);
        delayMicroseconds(tone_ / 2);
      }
      elapsed_time += tone_;
    }
    delay(pause);
  }
}
```

Playing a melody with beats (1/2)

```
#define buttonIn 47
#define ledOut 45
#define speakerOut 49

#define c3 7692 // 130 Hz
#define d3 6802 // 147 Hz
#define e3 6060 // 165 Hz
#define f3 5714 // 175 Hz
#define g3 5102 // 196 Hz
#define a3 4545 // 220 Hz
#define b3 4048 // 247 Hz
#define c4 3830 // 261 Hz
#define d4 3400 // 294 Hz
#define e4 3038 // 329 Hz
#define f4 2864 // 349 Hz
#define g4 2550 // 392 Hz
#define a4 2272 // 440 Hz
#define b4 2028 // 493 Hz
#define c5 1912 // 523 Hz
#define r1 0

int melody[] = {c4, c4, e4, e4, a3, a3, c4, e4, d4, d4, f4, f4, g3, g3, b3, b3, r1};
int beats[] = {22, 10, 22, 10, 22, 10, 24, 8, 22, 10, 22, 10, 22, 10, 22, 10, 16};
int MAX_MELODY = sizeof(melody)/2;
```

Playing a melody with beats (2/2)

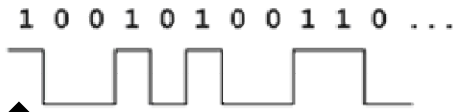
```
long tempo = 15000;
int pause = 10;
int tone_ = c4;
int beat = 0;
long duration = 0;
void play_tone(){
  long elapsed_time = 0;
  if( tone_ > 0 ){
    while( elapsed_time < duration ){
      if( elapsed_time + tone_ <= duration ){
        digitalWrite(speakerOut,HIGH);
        digitalWrite(ledOut,HIGH);
        delayMicroseconds(tone_ / 2);
        digitalWrite(speakerOut,LOW);
        digitalWrite(ledOut,LOW);
        delayMicroseconds(tone_ / 2);
      }
      elapsed_time += tone_;
    }
  }
  else{
    delayMicroseconds(duration);
  }
  delay(pause);
}
```

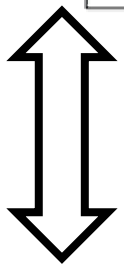
```
void setup(){
  pinMode(buttonIn, INPUT);
  pinMode(ledOut, OUTPUT);
  pinMode(speakerOut, OUTPUT);
}

int val;
void loop(){
  val = digitalRead(buttonIn);
  if (val == HIGH) {
    for(int i = 0; i < MAX_MELODY ; i++ ){
      tone_ = melody[i];
      beat = beats[i];
      duration = beat * tempo;
      play_tone();
    }
  }
  else{
    digitalWrite(ledOut,LOW);
    delay(100);
  }
}
```

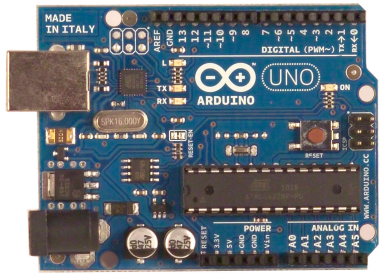
Using Serial Communication

- **Method used to transfer data between two devices.**

Data passes between the computer and Arduino through the USB cable. Data is transmitted as zeros ('0') and ones ('1') sequentially.







Arduino dedicates Digital I/O pin # 0 to receiving and Digital I/O pin #1 to transmit.

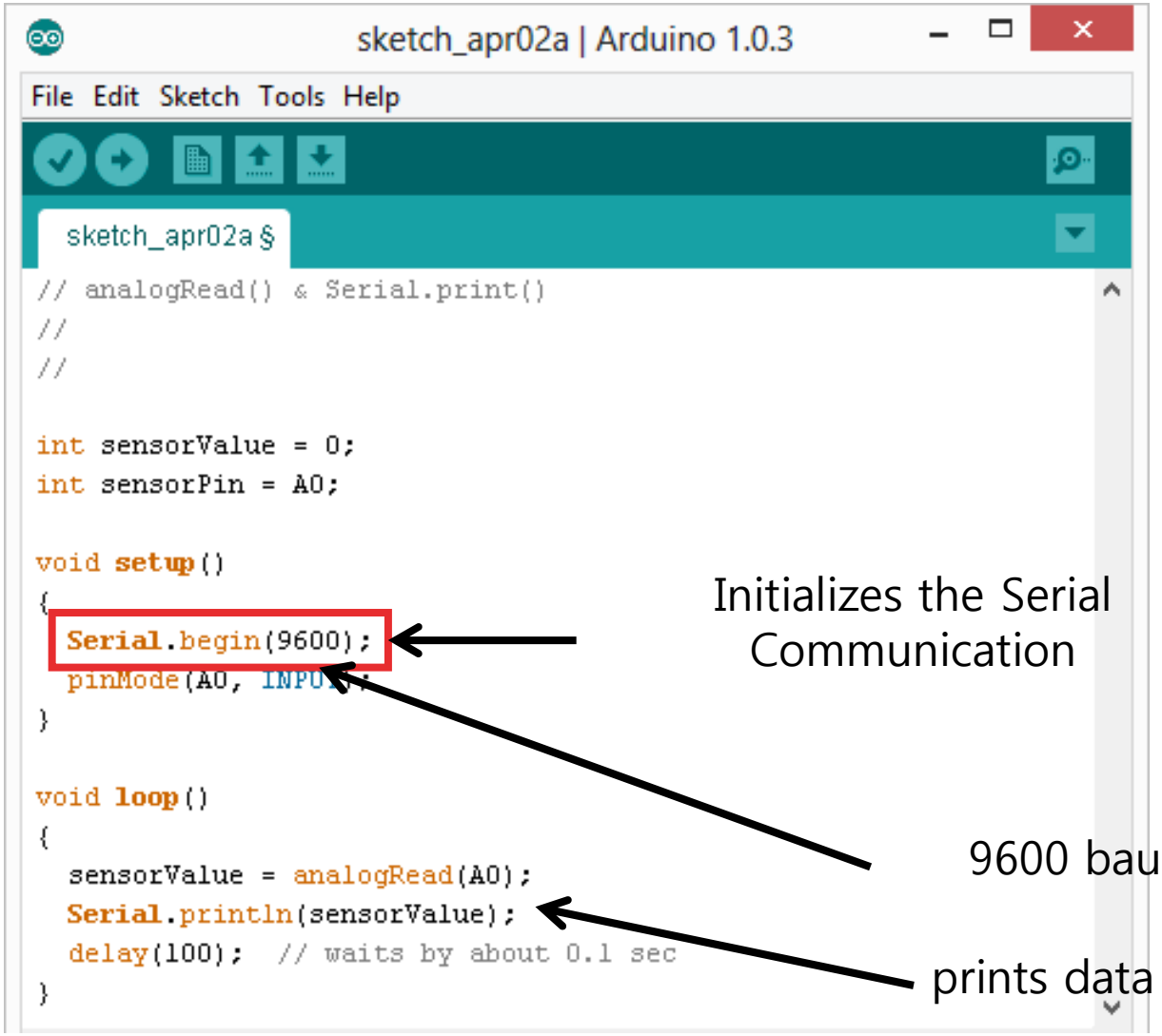
Serial setup

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}
```

Serial communication also begins in setup

This particular example declares Serial communication at a baud rate of 9600. More on Serial later...

Serial Monitor & analogRead()



Setup Interrupts

- attachInterrupt(interrupt, function, mode)

You can designate an interrupt function to Arduino UNO

pins # 2 and 3

This is a way around the linear processing of Arduino

Setup Interrupts

- attachInterrupt(interrupt, function, mode)

Interrupt: the number of the interrupt, 0 or 1, corresponding to Arduino UNO pins # 2 and 3 respectively

Function: the function to call when the interrupt occurs

Mode: defines when the interrupt should be triggered

Setup Interrupts

- attachInterrupt(interrupt, function, **mode**)

LOW whenever pin state is low

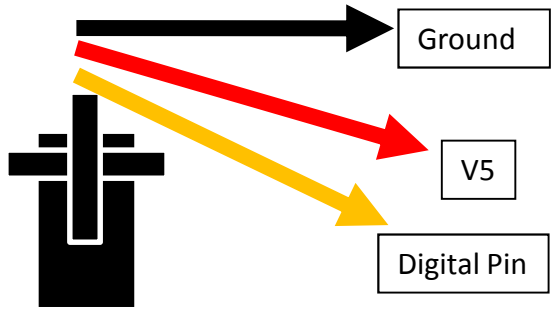
CHANGE whenever pin changes value

RISING whenever pin goes from low to high

FALLING whenever pin goes from low to high

Don’t forget to CAPITALIZE

Servo Motor



- Servo Motors are electronic devices that convert digital signal to rotational movement.
  - There are two sorts of servo motors: Standard servos that their rotation is limited to maximum of 180 degrees in each direction and Continuous Rotation Servos that can provide rotation unlimitedly in both directions
- A servo motor is a motor that pulses at a certain rate moving its gear at a certain angle.
  - It has three connections: **the black is ground, the red is connected to 5V, and the white (yellow wire here) is set to the digital pin.**

Using “Servo.h” to rotate servo to exact angle

```
#include <Servo.h>
Servo myservo;  // create servo object to control a servo
int pos = 0;    // variable to store the servo position
void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}
void loop()
{
  myservo.attach(9);
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for(pos = 180; pos>=1; pos-=1)     // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  myservo.detach();                  //Detach the servo if you are not controling it for a while
  delay(2000);
}
```

Using emulated PWM for rotating server to exact angle

```
void setup(){
pinMode(5,OUTPUT); //set the pin to output
Serial.begin(9600); //open the serial to print
Serial.print("Ready"); //write a message
Serial.println();
}
void loop(){
int val = Serial.read(); //read the serial to see
if(val>='0' && val <= '9'){ //which key was pressed
val = val - '0'; //convert the character to an integer
val = val * (180/9); //9 divisions of 180 degrees
Serial.print("moving servo to ");
Serial.print(val);
Serial.println();
for(int a=0; a<100; a++){
int pulseWidth = (val*11)+500; // See the formula above
digitalWrite(5,HIGH);
delayMicroseconds(pulseWidth);
digitalWrite(5,LOW);
delay(10);
}
}
}
```

Continuous rotation

- As opposed to standard Servo that its rotation is limited to 180 degrees both ways, a continuous rotation servo can keep rotating unlimitedly-again both ways- based on the frequency that is pulsed out to it.
- There is a specific frequency at which the Servo motor should be static and beyond and before which the servo will change in its rotation direction.
- There is a pin on the servo motor that enables us to adjust the servo for its static frequency.

Adjustment

- Upload the following code to the board and while the servo is connected, try to adjust the pin until the servo motor is static.
- Once the servo is adjusted to this code any pulse grater than 1500 will result in rotation in one direction while any pulse less than 1500 will result in rotation in the other direction

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500); // 1.5ms This is the frequency at which the servo motor should be static
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```

Testing Code

```
void setup()
{
pinMode(5,OUTPUT);
}
void loop()
{

//Rotating in One direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1800);
digitalWrite(5,LOW);
delay(20); // 20ms
}

//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}
```

```
//Rotating in the other direction
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1200);
digitalWrite(5,LOW);
delay(20); // 20ms
}

//Stop
for (int i = 0; i <= 200; i++)
{
digitalWrite(5,HIGH);
delayMicroseconds(1500);
digitalWrite(5,LOW);
delay(20); // 20ms
}
}
```