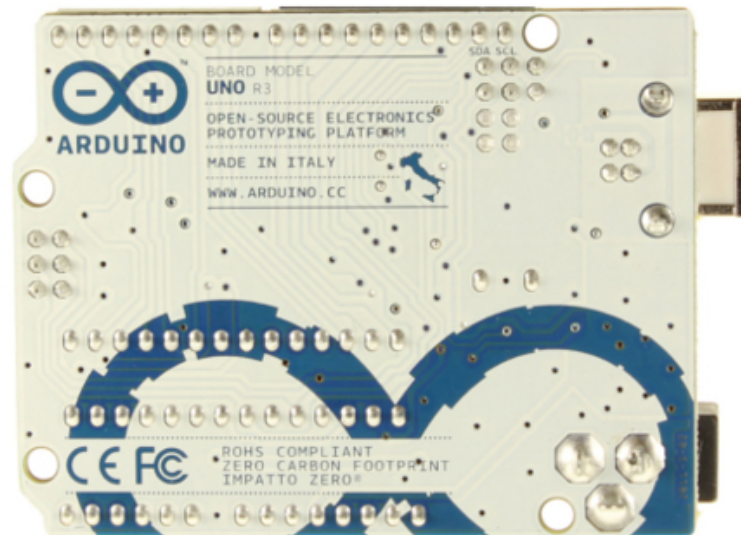
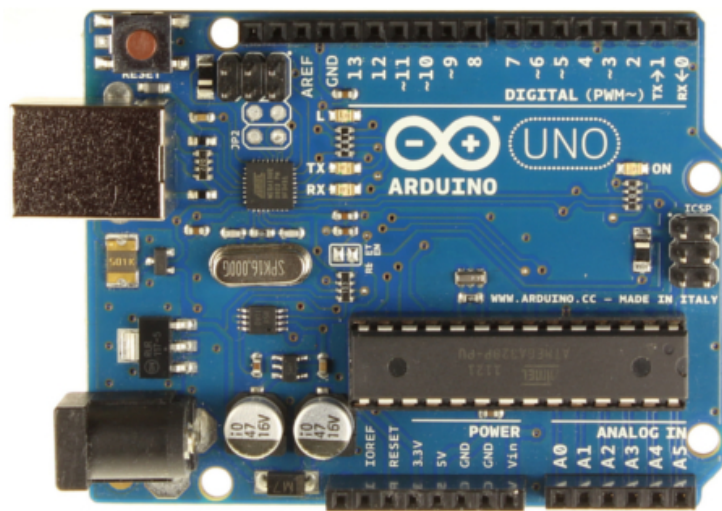


# Arduino and basic C review

# Course requirements

---

- Arduino Uno R3 - \$29.95
- USB Cable A to B - \$3
- Access to a computer - Either Mac or PC is fine
- Download free Arduino IDE from [www.arduino.cc](http://www.arduino.cc)



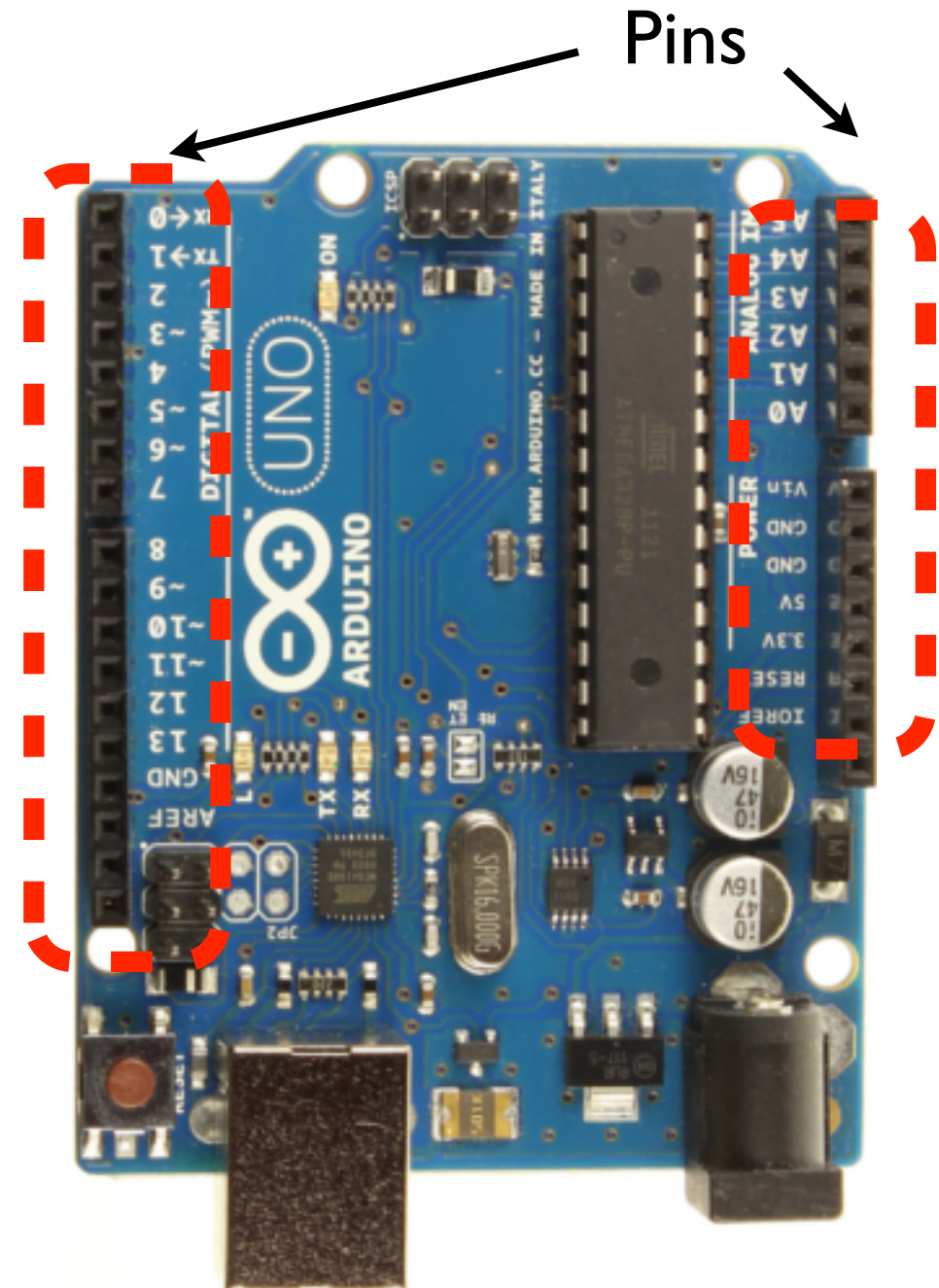
# Introduction to Arduino development

---

- This topic is a very basic introduction to the Arduino development and the prototype board.
- This is so you can refresh your basic C programming skills.
- In the process you will also become comfortable with the Arduino prototype board.
- In future classes we will be delving deeper into the architecture of the ATmega so we can implement and study real time systems.

# Arduino

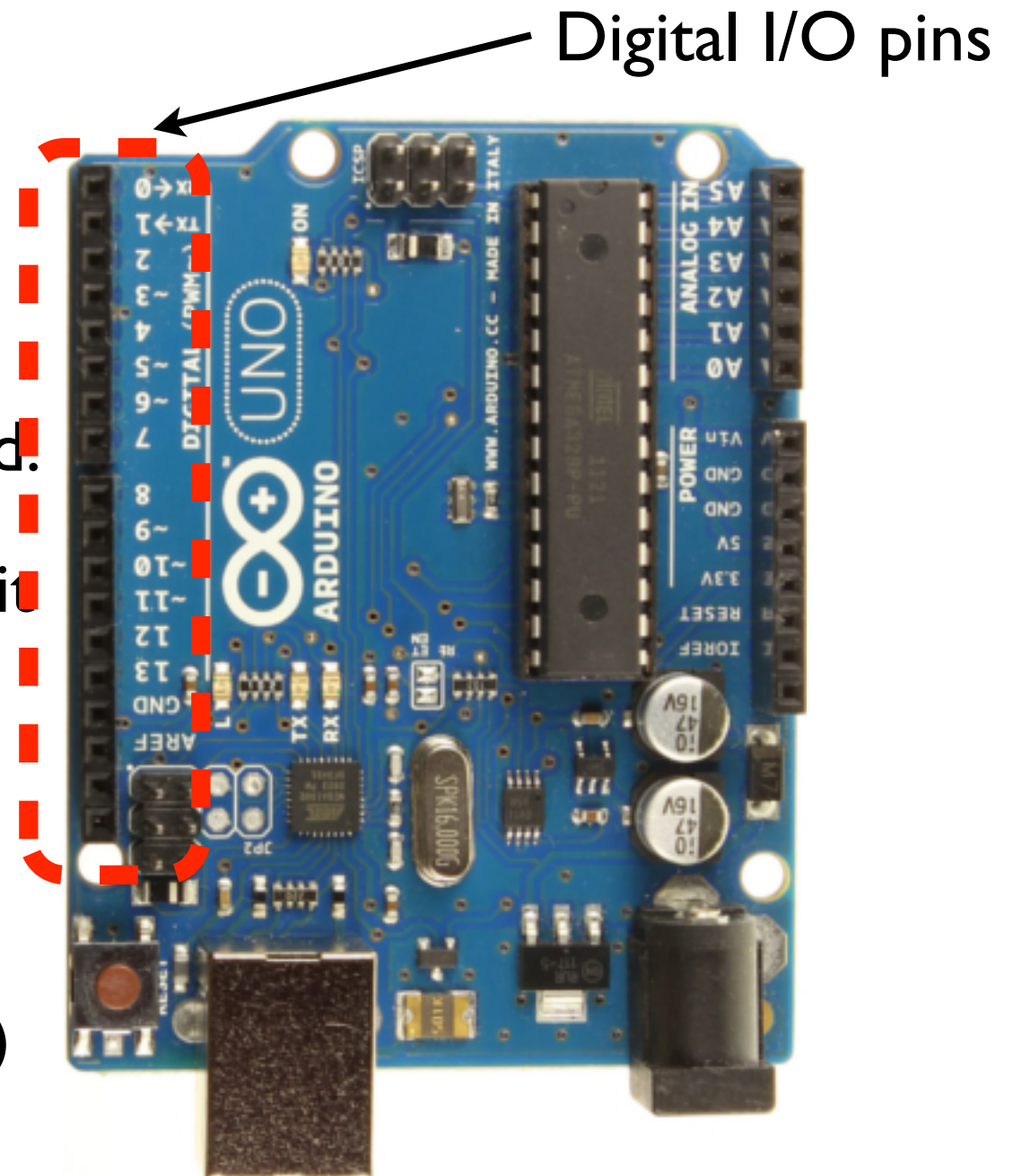
- This is the prototype board we will be using in our class.
- Broadly speaking, any embedded program can:
  1. Turn ON/OFF pins.
  2. Read data from pins.





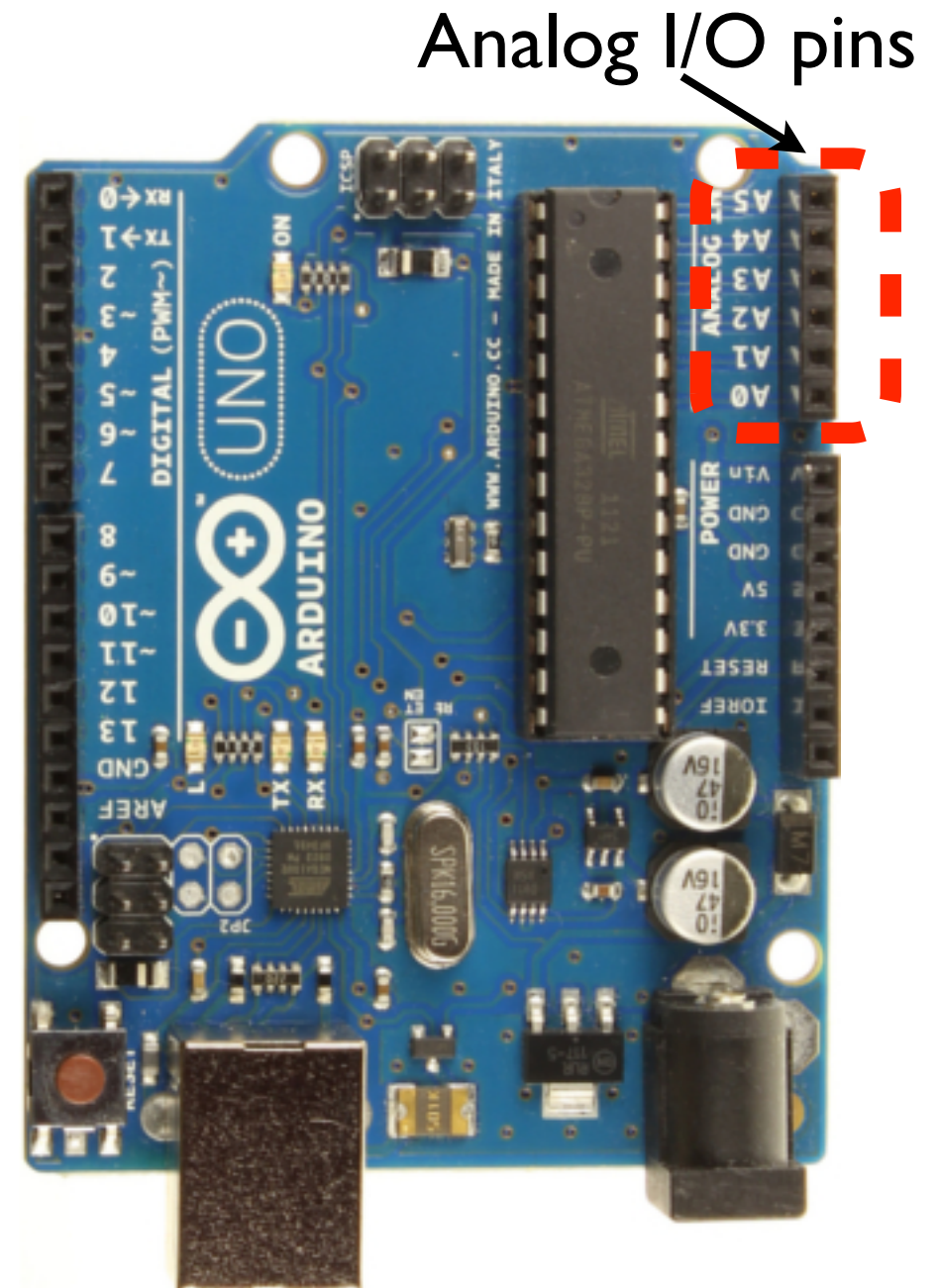
# Digital I/O

- Input/Output (I/O) is done through pins.
- Input = read physical world data.
- Output = write data to the physical world.
- You insert a wire into a pin and connect it to something else.
- We can program a pin to be input **OR** output.
- Digital pins have two values: high (5 Volts) or low (0 Volts).



# Analog I/O

- Analog I/O has a range of numbers.
- Output : 0 ... 255 (256 voltage steps from 0 to 5V).
- Input: 0 ... 1023 (1024 voltage steps from 0 to 5V).
- Analog input pin: we can use it for example to determine the distance of an object via infra-red sensor.
- Analog output pin: we can use it for example to set speed of a motor or the brightness of a LED.

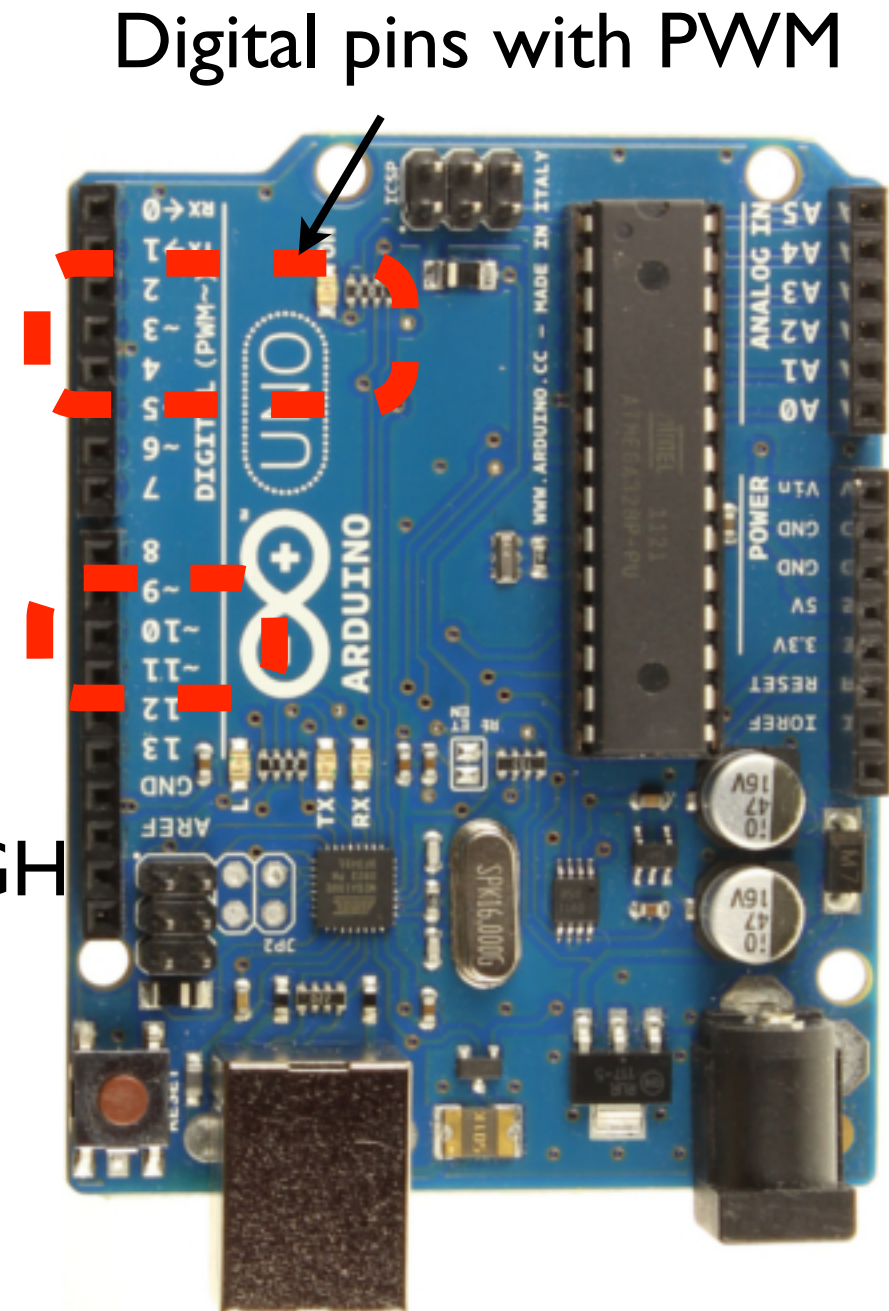




# Pulse Width Modulation

---

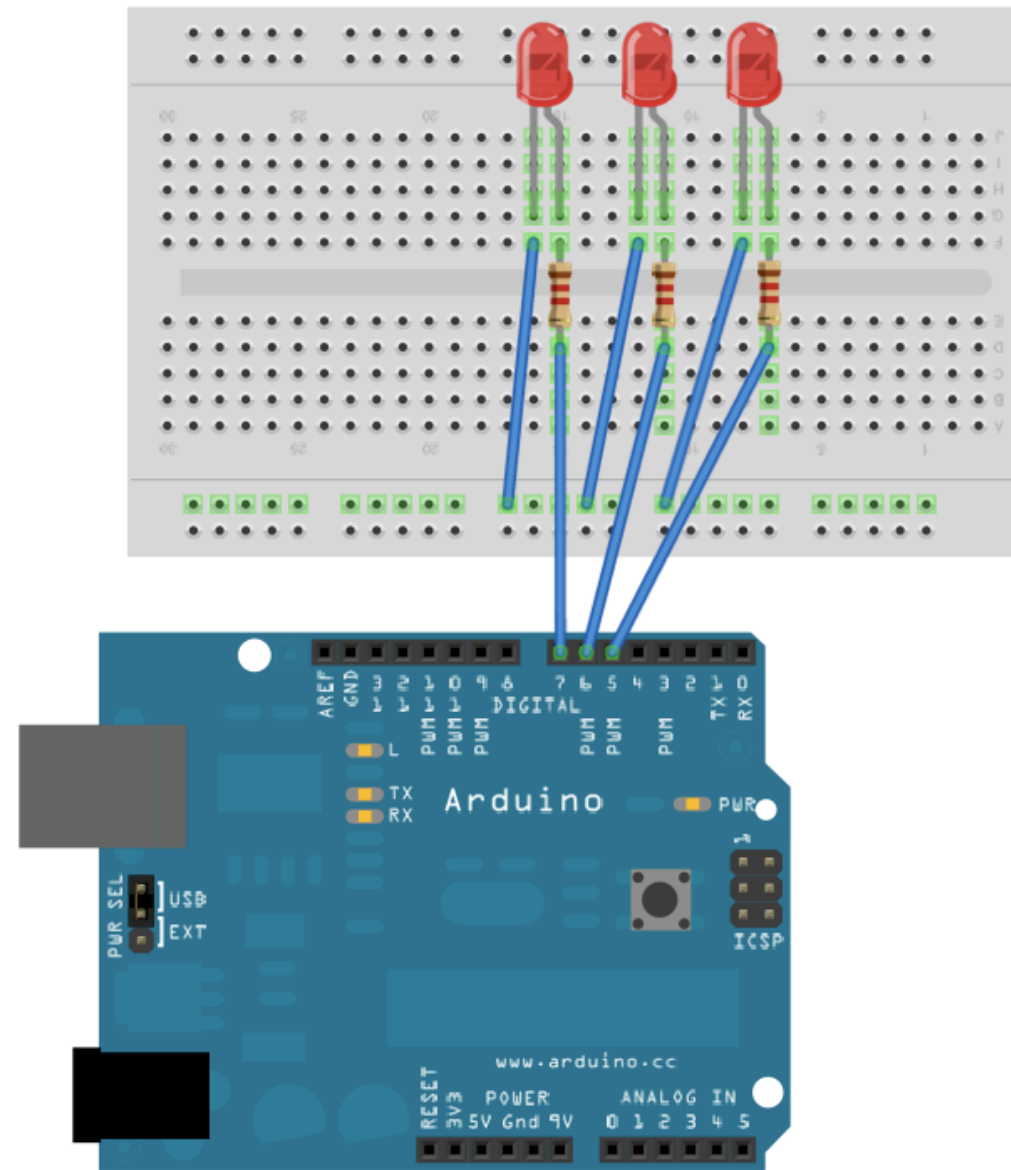
- Some digital ports can be programmed to output analog signals.
- We enable those ports for output with pulse width modulation (PWM).
- PWM is obtained by varying between HIGH and LOW at the appropriate interval of time.



# Dimming an LED

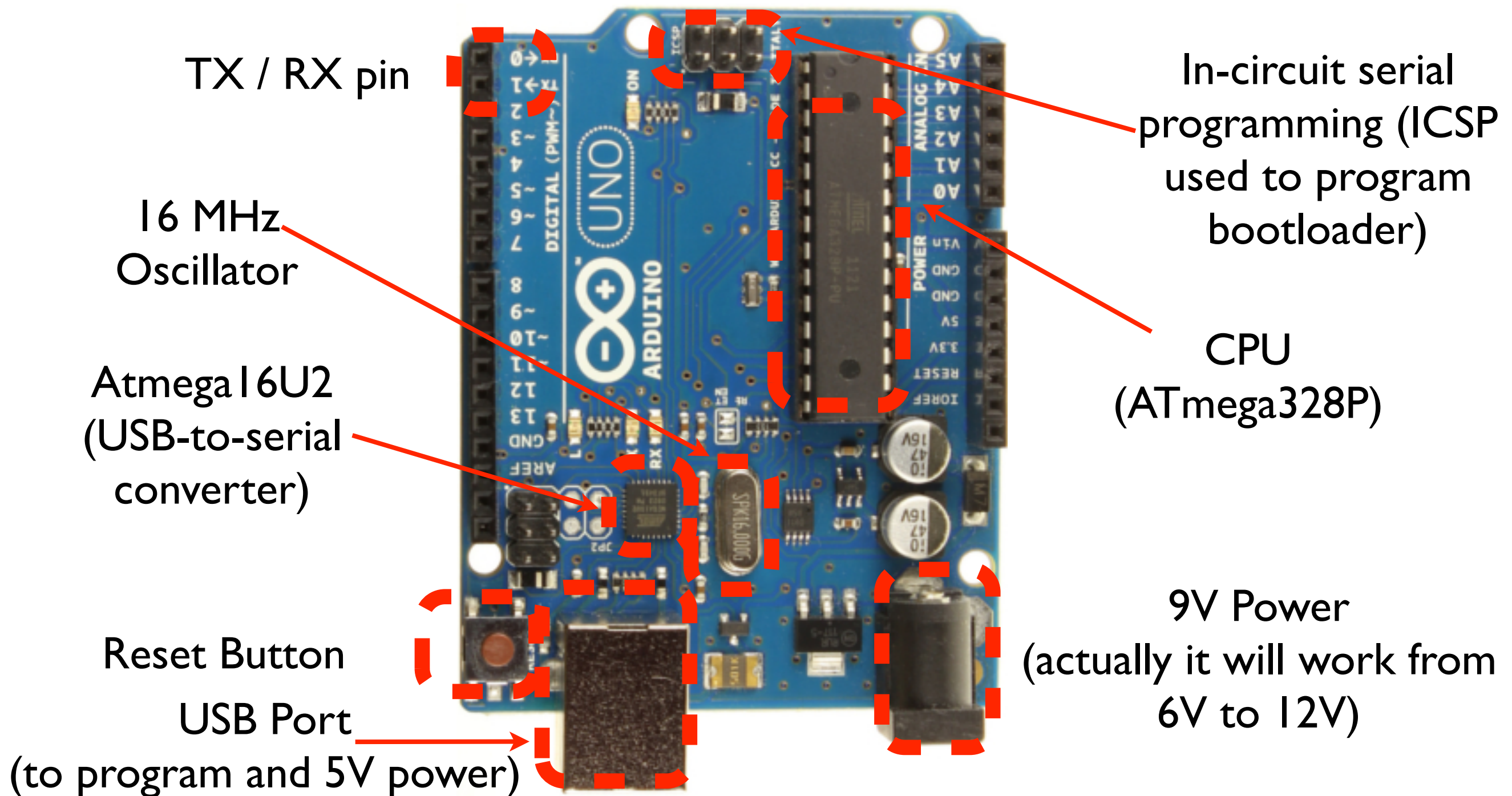
---

- Let me specify pin 7 to be an output pin (5V).
- LED will be ON.
- If turn pin 7 ON and OFF very fast, our eyes will see the light dimming dimming.





# Other board components

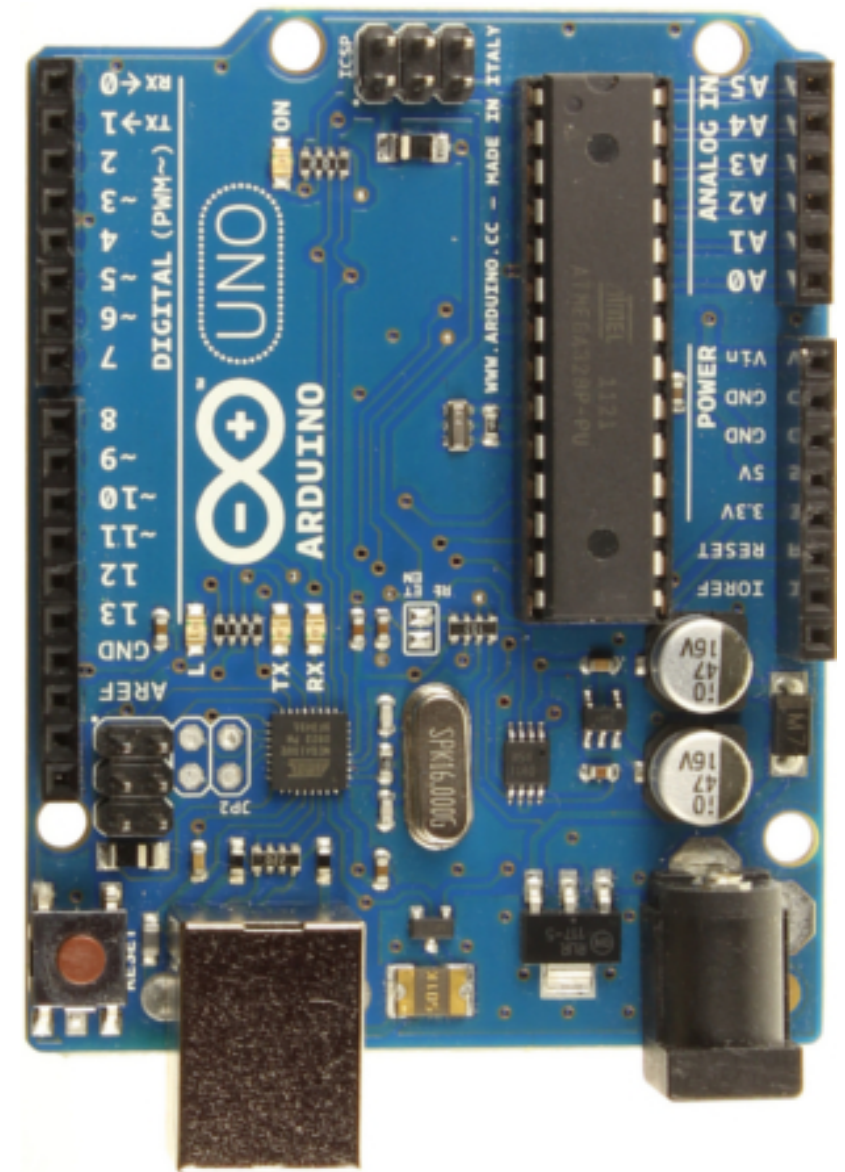


# Arduino Uno specifications

---

## Summary

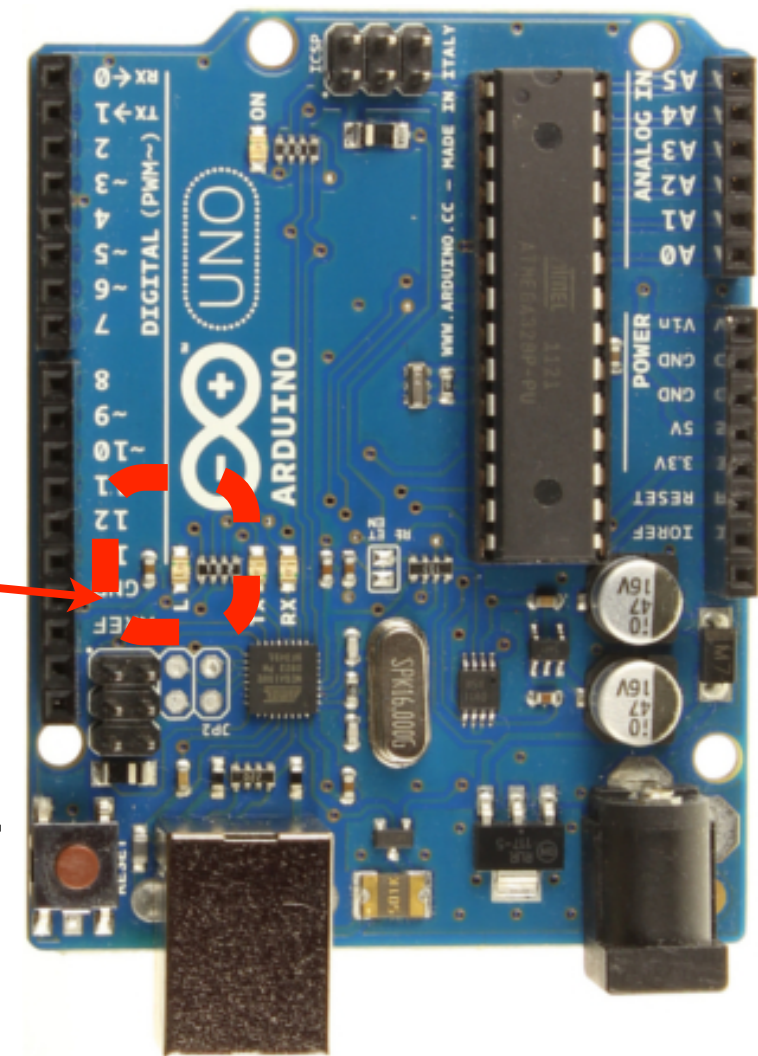
Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz





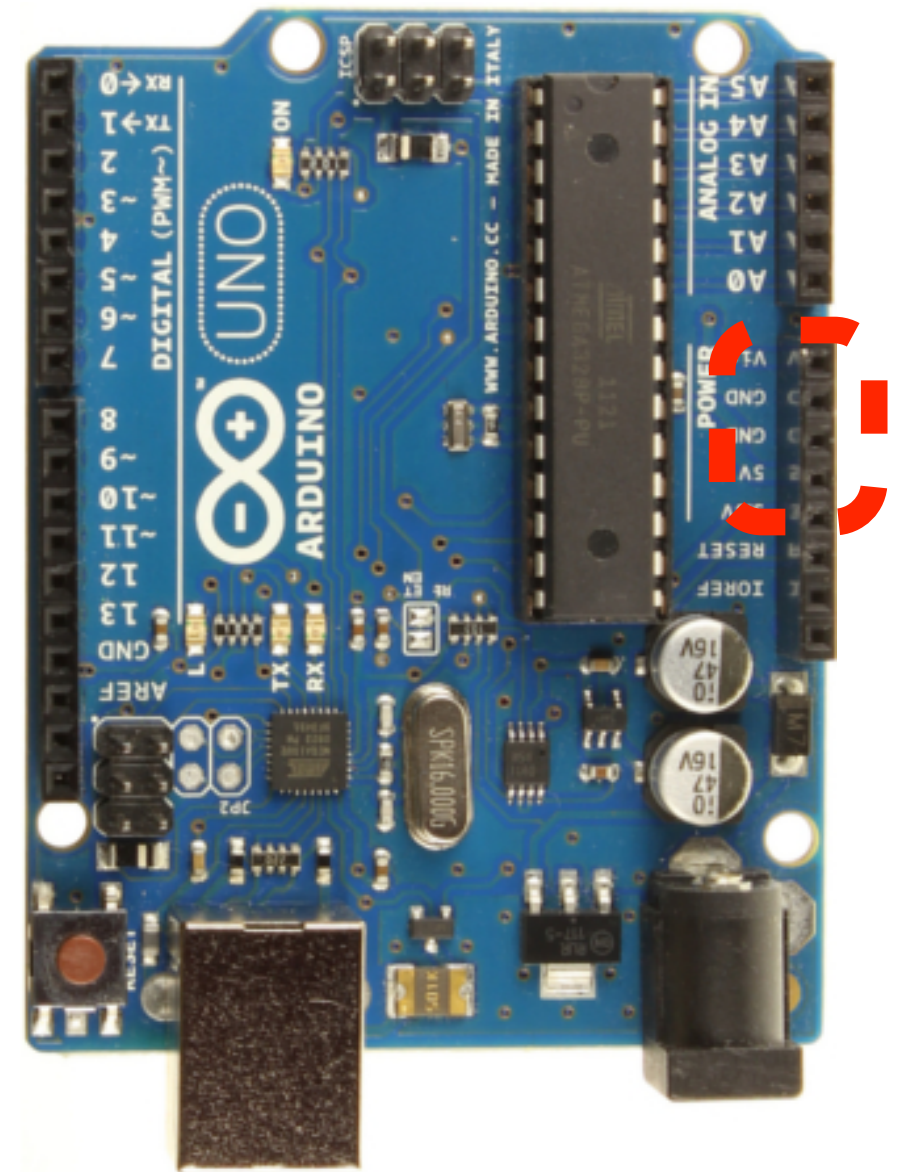
# Bootloader

- The bootloader is a small piece of software that is burned onto the chips that come with your Arduino boards. It allows the user to upload programs to the board without external hardware.
- When you reset the Arduino board, it runs the bootloader (if present) and digital pin 13 is pulsed.
- The bootloader waits a few seconds for data to arrive from the the USB port. Seconds later, the bootloader launches the newly-uploaded program. If no data arrives from the USB port, the bootloader launches whatever program was last uploaded onto the chip.



# Power pins

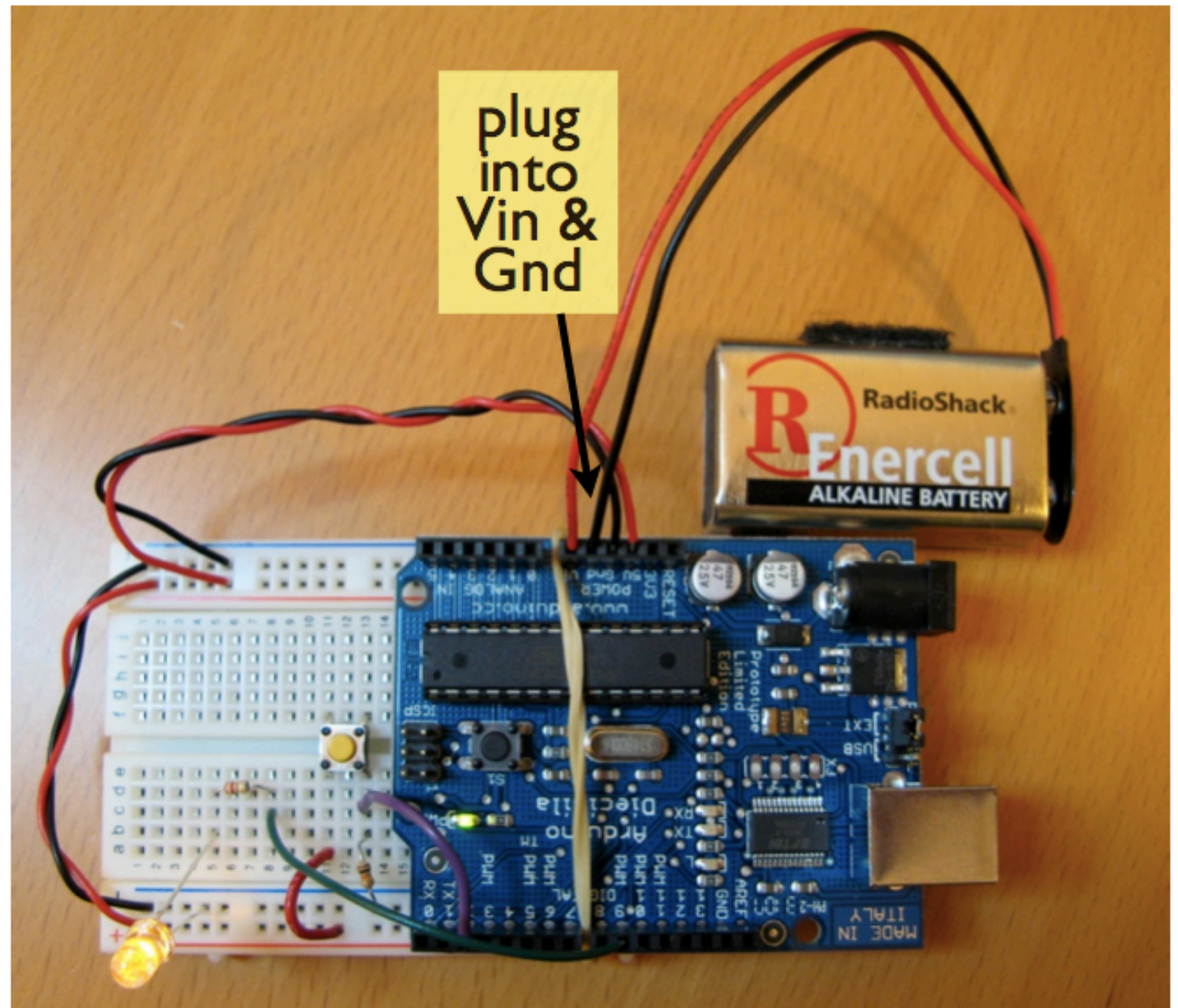
- **VIN** - You may provide a regulated input voltage to the Arduino board as opposed to 5 volts from the USB connection or other regulated power source.
- **5V** - When power is provided to the board, this pin has 5V (reference to **GND** pin)
- **3V3** - A 3.3 volt supply generated by the on-board regulator (reference to **GND** pin). Maximum current draw is 50 mA.





# Battery powered

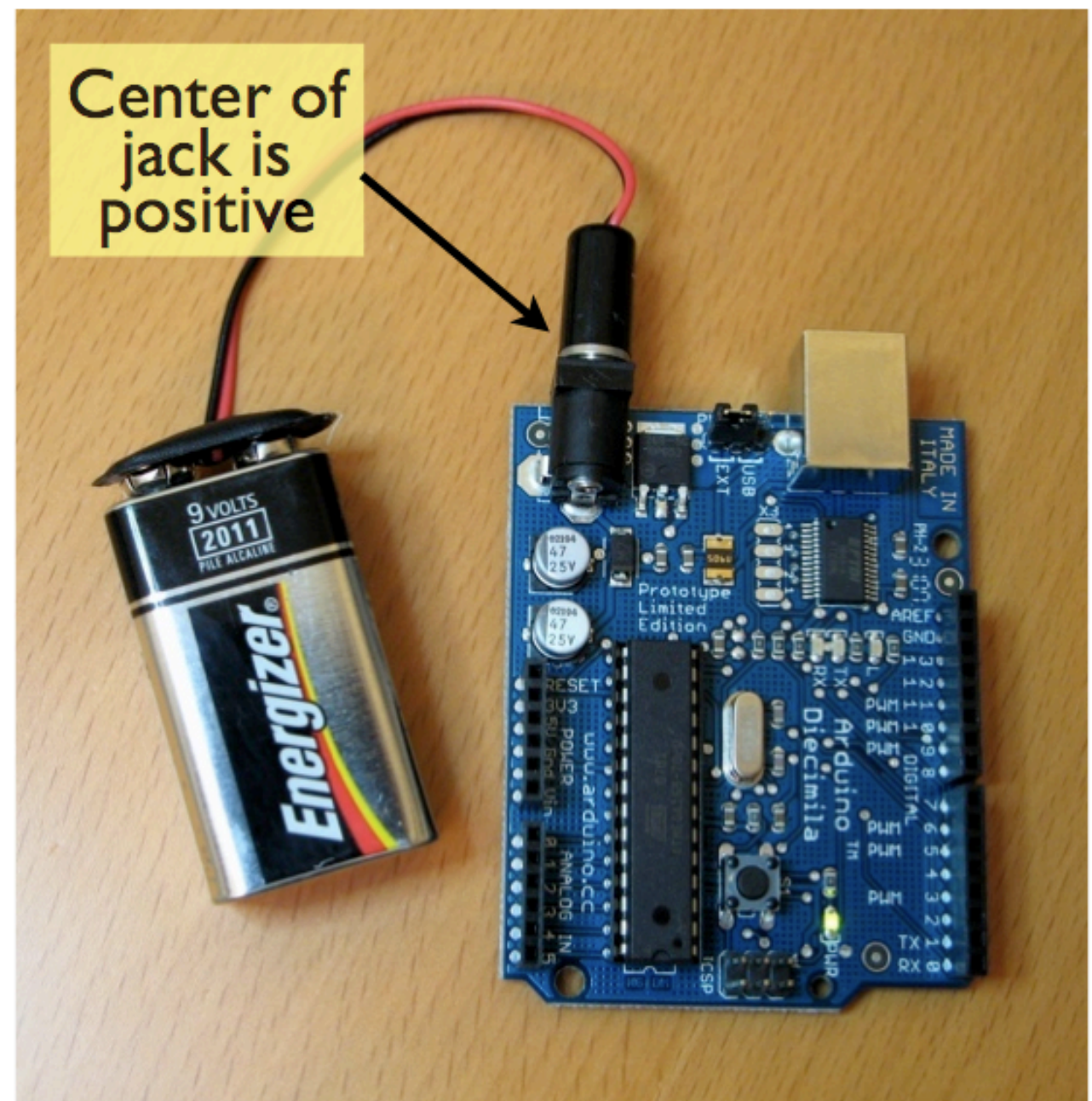
- First, program sketch into Arduino
- Unplug USB cable
- Plug in power (7-12VDC)
- Power LED lights up. It works!
- Reverse steps to reprogram



# Battery powered

---

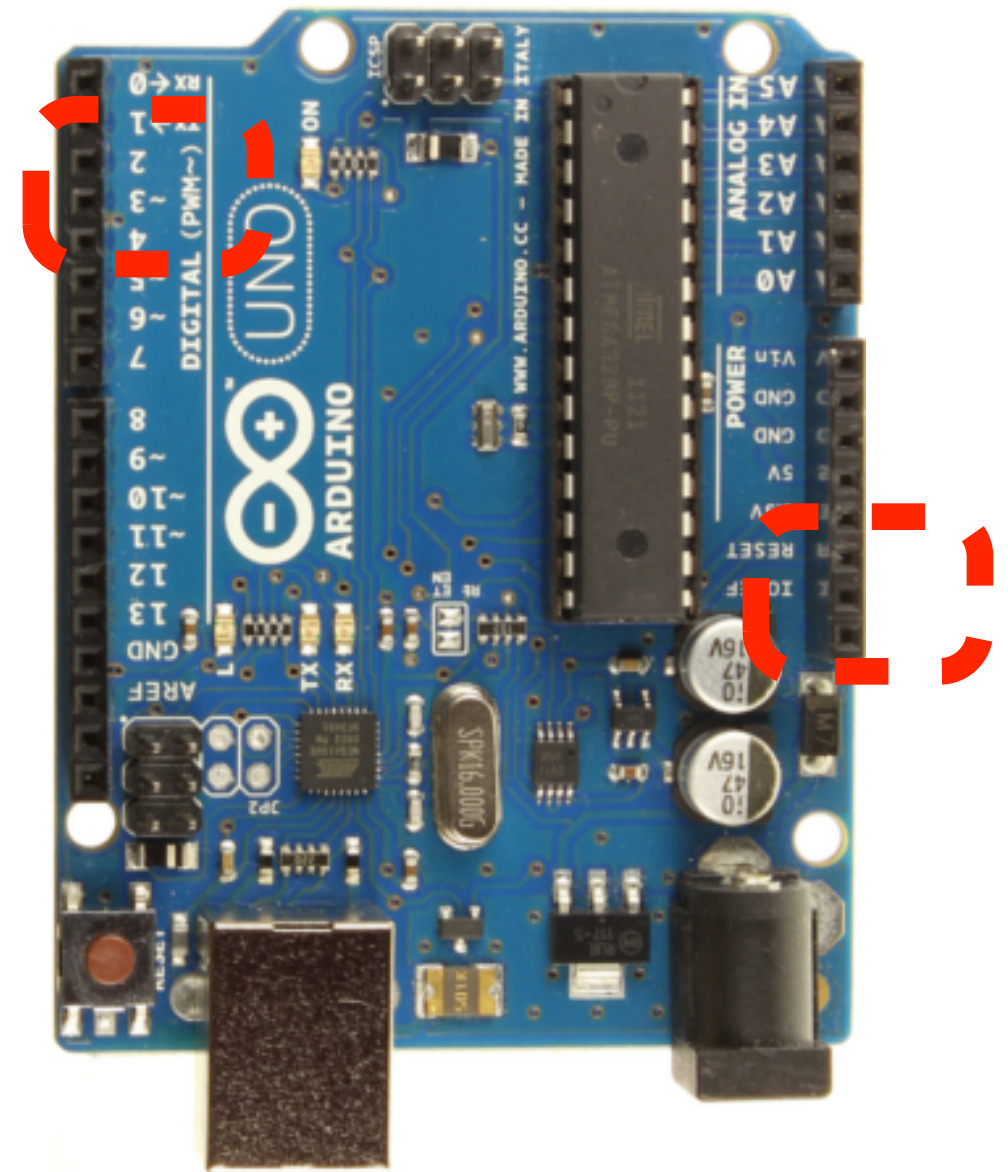
- Plugging into the sockets is kind of fiddly
- Better to plug into the power jack
- Works great, but requires a little soldering





# Other pins

- Digital Pin 2 and 3 can be used as **external interrupts** (trigger an interrupt on a low value, a rising or falling edge, or a change in value).
- **AREF** - Reference voltage for the analog inputs.
- **Reset** - Bring this line LOW to reset the micro-controller.

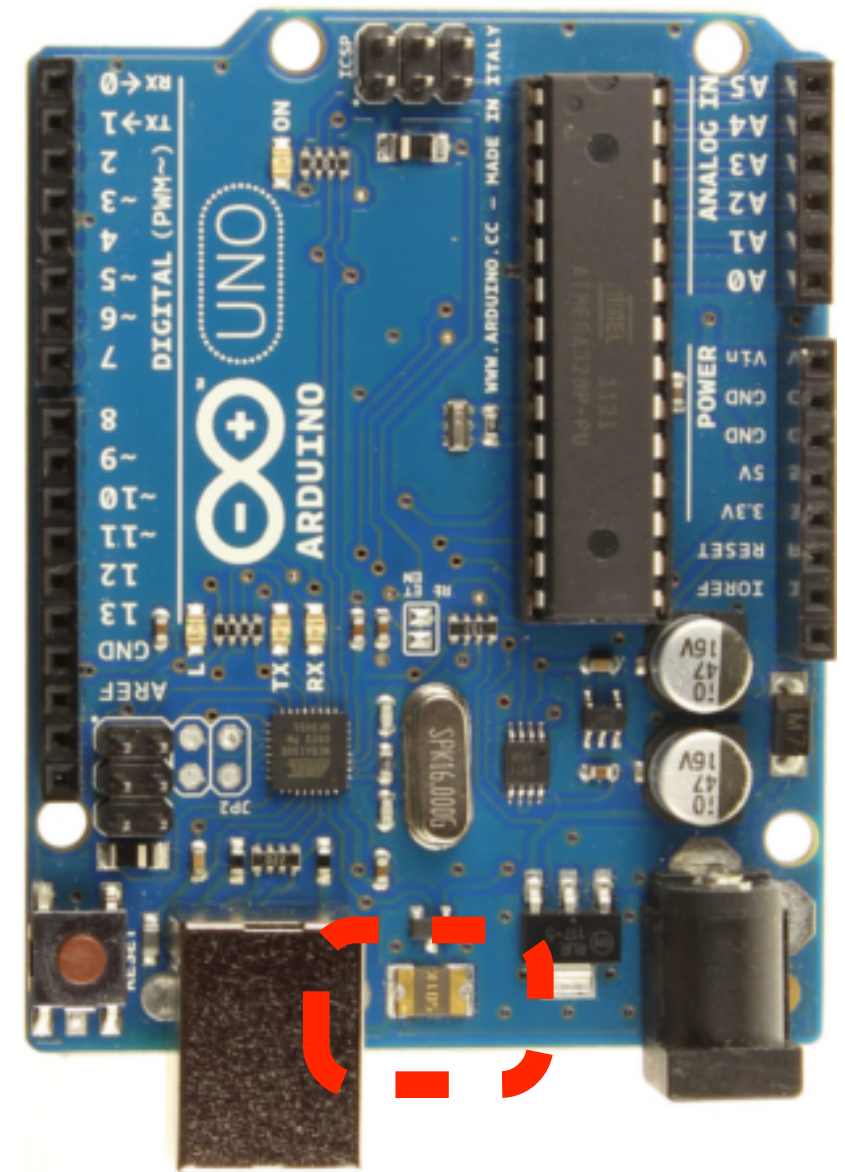


# Polyfuse

- What is that funny looking chip?



- Its a resettable polyfuse that protects your computer's USB ports from shorts and over-current. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection.

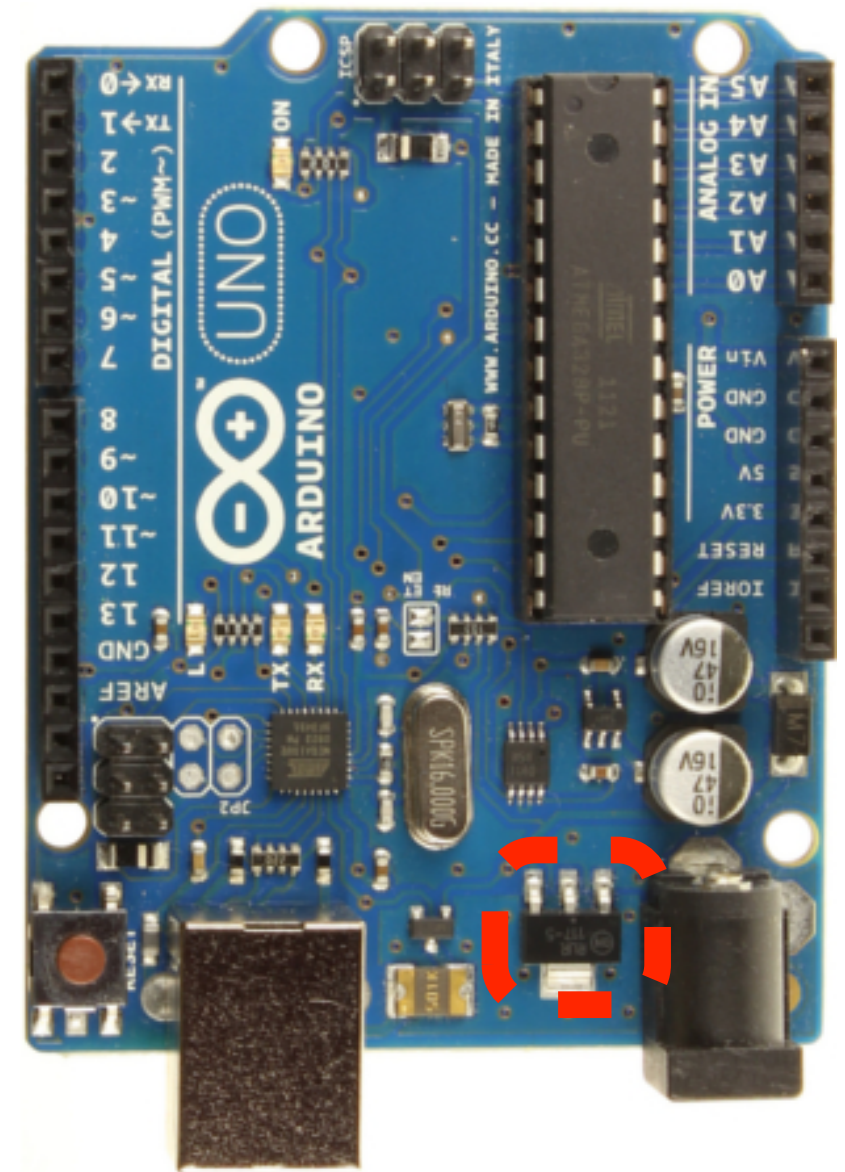




# AMS1117 Voltage regulator

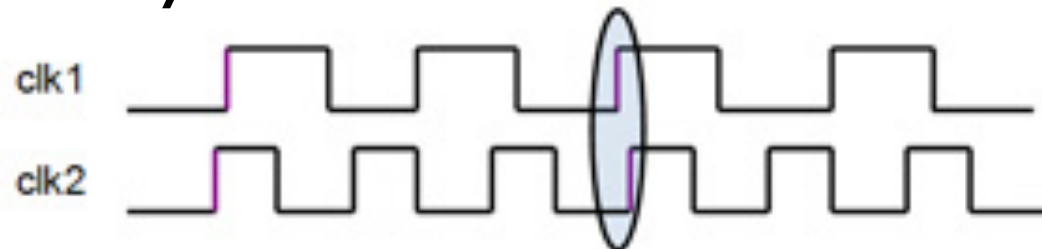
---

- Takes in a 9V voltage as an input
- The output is a regulated 5V

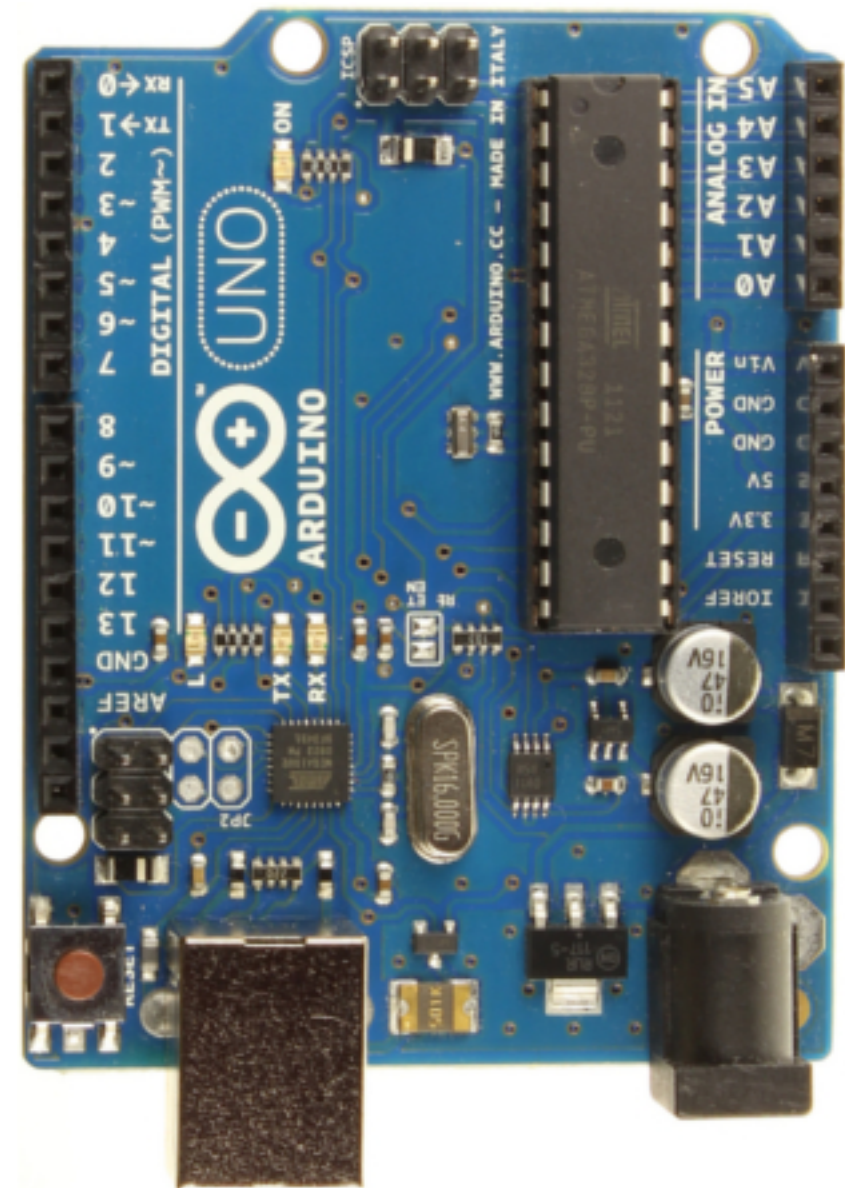


# How fast is 16 MHz ?

- Every micro-controller has a clock



- $f = 1 / T$  (frequency =  $1 /$  clock period)
- $16 \times 10^6 \text{ Hz} = 1 / T \Leftrightarrow T = 62.5 \text{ ns}$
- But can you turn ON and OFF a light every 62.5 ns?



# Why is Arduino so successful?

---

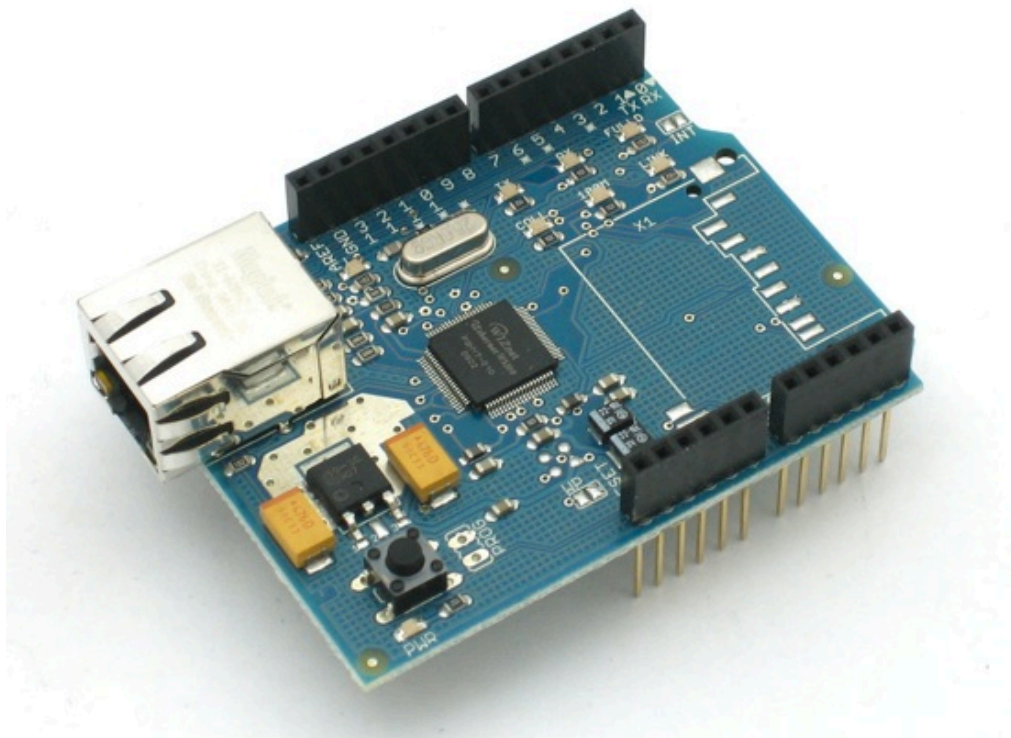
- Improvement over other micro-controllers for hobby usage: Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard ...
- Inexpensive
- Cross-platform
- Simple, clear programming environment: It uses a simplified set of libraries (called wiring) that are built over C
- Open source and extensible software/hardware
- Arduino Shields



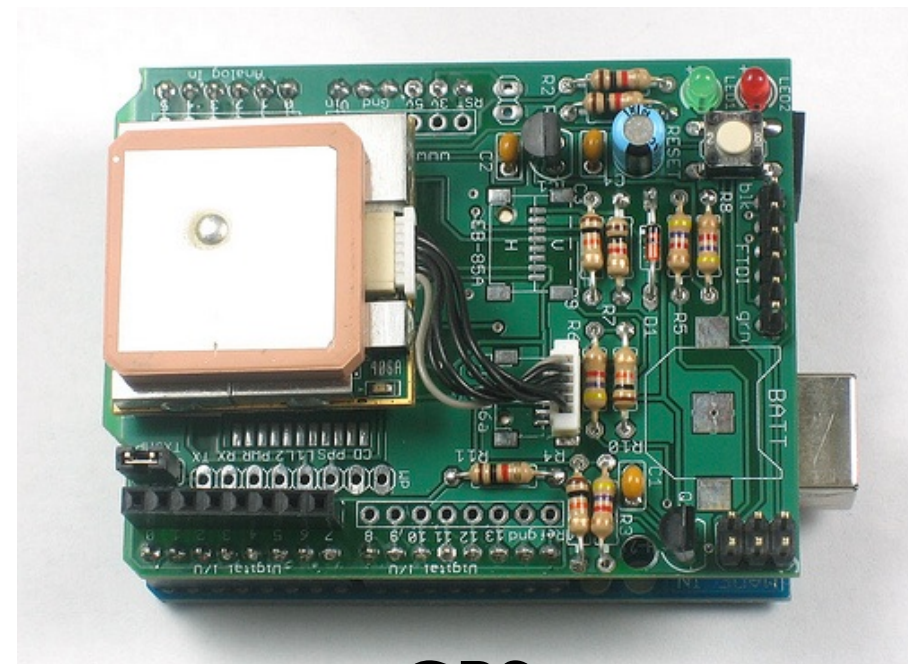
# Arduino shields

---

- Shields are boards that can be plugged on top of the Arduino
- They extend its capabilities.
- Internet connectivity, wireless communication, motor controls



Ethernet connectivity



GPS



# Program example using Arduino Wiring libraries

---

```
int ledPin = 13;    // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts

void setup() {
    // initialize the digital pin as an output:
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);    // set the LED on
    delay(1000);                   // wait for a second
    digitalWrite(ledPin, LOW);     // set the LED off
    delay(1000);                   // wait for a second
}
```

Wiring libraries make programming much easier, but we lose a lot of internal micro-processor control.

# WARNING


---

- Wiring libraries are only compatible with the micro-controller that is inside the Arduino board (Atmel ATmega).
- We will only use the Wiring libraries in order to refresh you C-programming skills and introduce the development environment.
- In some future topics, we will interact directly with the micro-controller with pure ANSI-C so that our code language is compatible with other micro-controllers.
- We will also look at the ATmega328P architecture in detail.

# Arduino IDE

---

- Simple to use
- Open Source (Free)
- Programming style similar to C
- You can download it from [www.arduino.cc](http://www.arduino.cc)



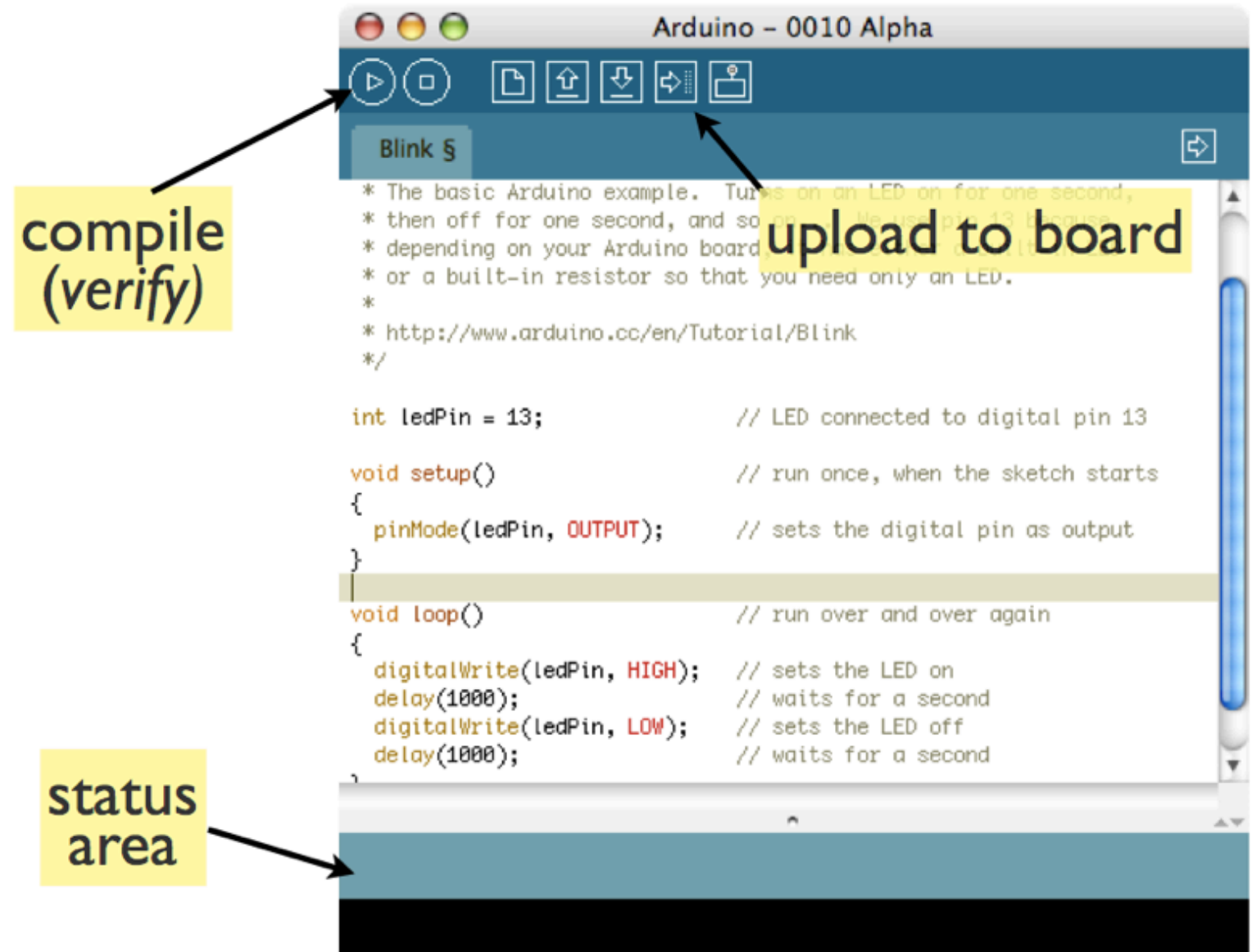
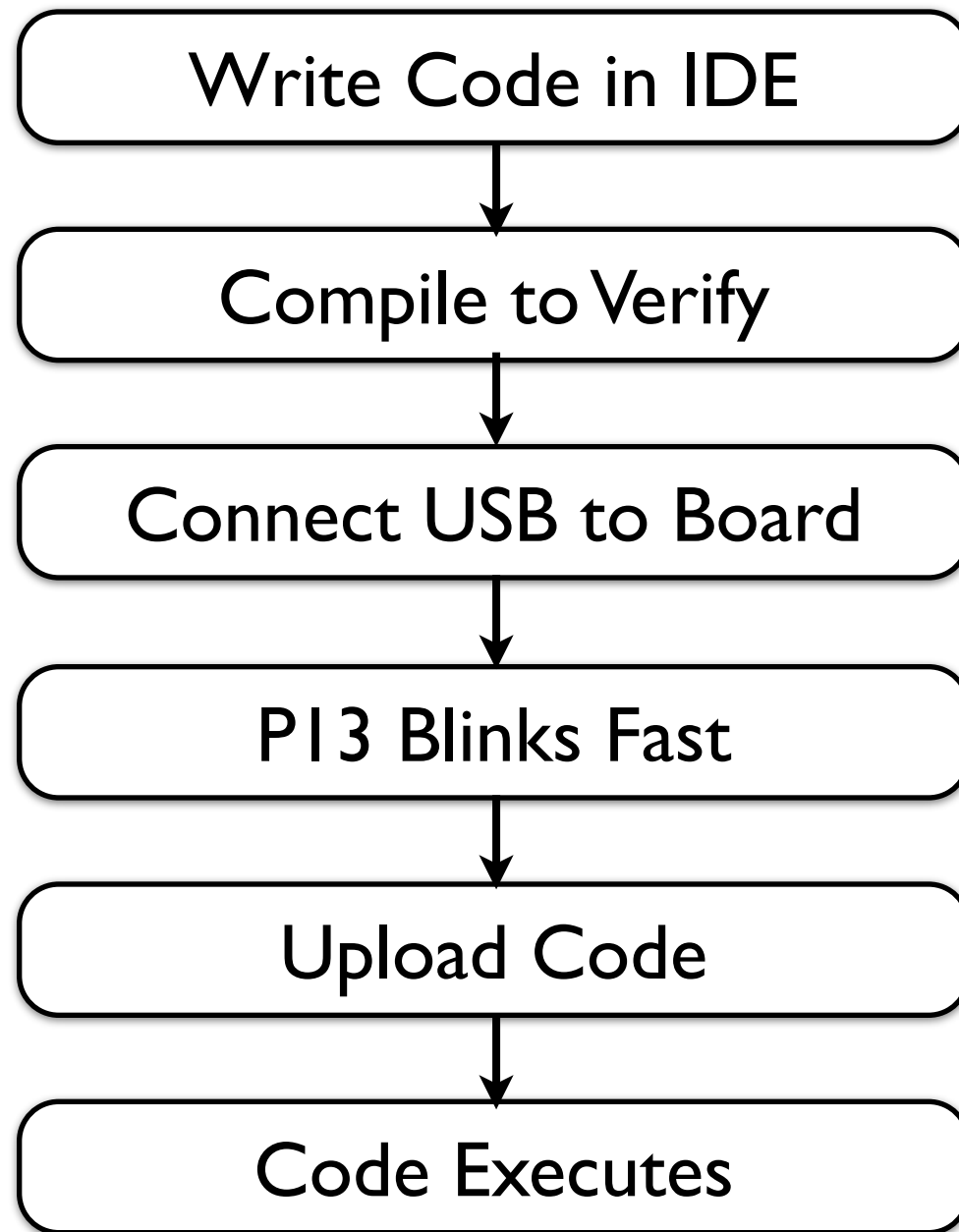
The screenshot shows the Arduino IDE window titled "Arduino - 0011 Alpha". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu bar is a toolbar with icons for running, stopping, saving, opening, and other functions. The main text area displays the "Blink" sketch, which is a basic Arduino example that turns an LED on for one second, then off for one second, and so on. The code is as follows:

```
/*  
 * Blink  
 *  
 * The basic Arduino example. Turns on an LED on for one second,  
 * then off for one second, and so on... We use pin 13 because,  
 * depending on your Arduino board, it has either a built-in LED  
 * or a built-in resistor so that you need only an LED.  
 *  
 * http://www.arduino.cc/en/Tutorial/Blink  
 */  
  
int ledPin = 13;           // LED connected to digital pin 13  
  
void setup()               // run once, when the sketch starts  
{  
  pinMode(ledPin, OUTPUT); // sets the digital pin as output  
}  
  
void loop()               // run over and over again  
{  
  digitalWrite(ledPin, HIGH); // sets the LED on  
  delay(1000);               // waits for a second  
  digitalWrite(ledPin, LOW);  // sets the LED off  
  delay(1000);               // waits for a second  
}
```

At the bottom of the window, there is a status bar that says "Done compiling." and "Binary sketch size: 1098 bytes (of a 14336 byte maximum)". The line number 22 is also visible.



# Arduino programming flow

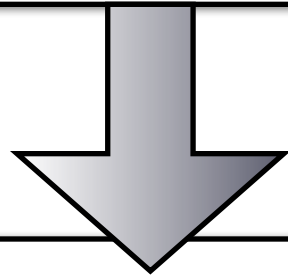


# Anatomy of a “sketch”

---

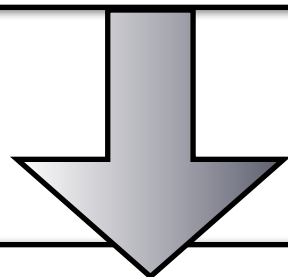
Global Variables

Declare global variables on top



Setup()

Code that is executed only once at beginning (eg: set pins)



Loop()

Code that runs repeatedly, after setup()

# Programming in the Arduino environment using Wiring.

---

Language is standard C.

Wiring library has many useful functions:

- `pinMode()` – set a pin as input or output
- `digitalWrite()` – set a digital pin high/low
- `digitalRead()` – read a digital pin's state
- `analogRead()` – read an analog pin
- `analogWrite()` – write an “analog” value
- `delay()` – wait an amount of time

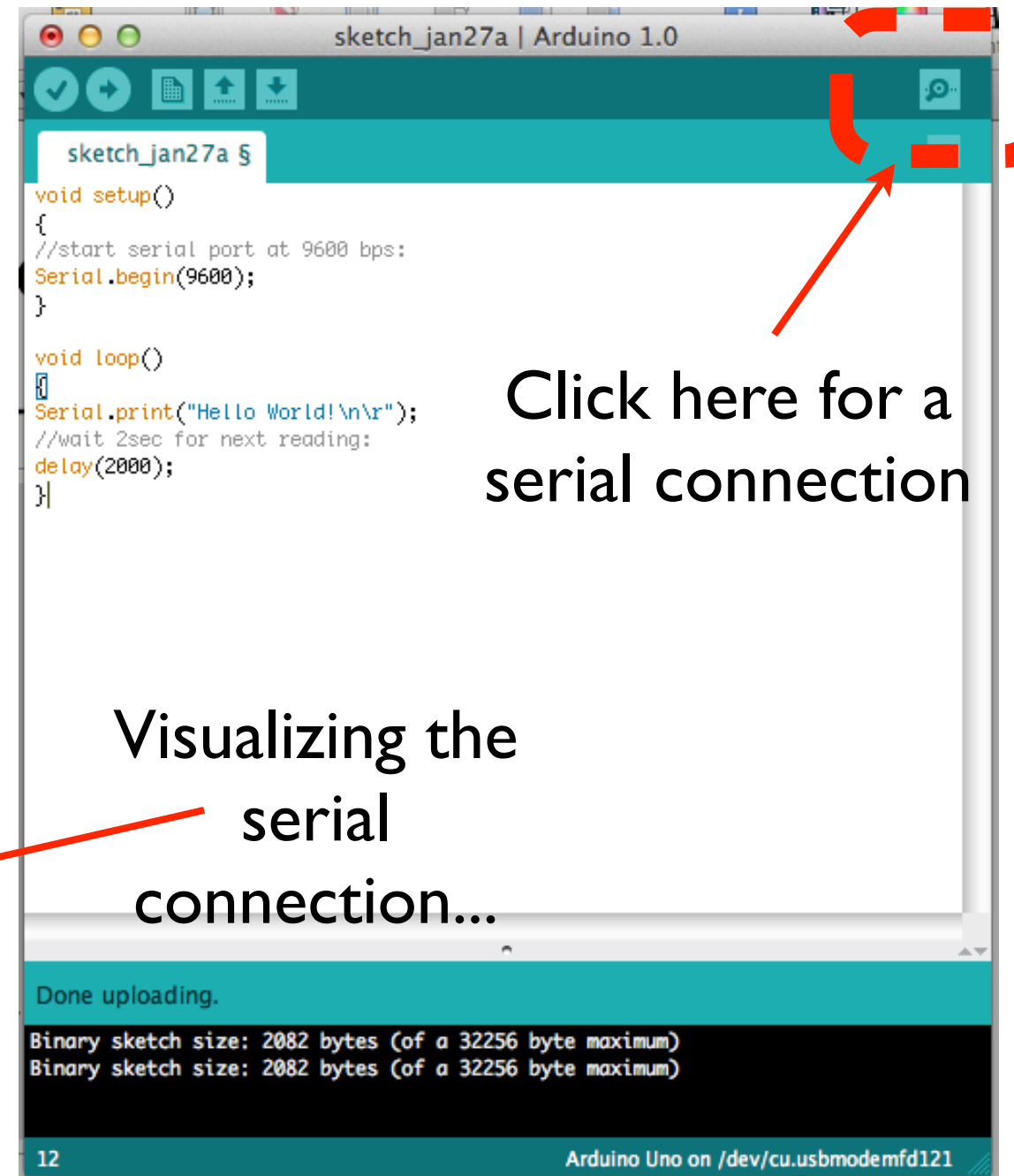
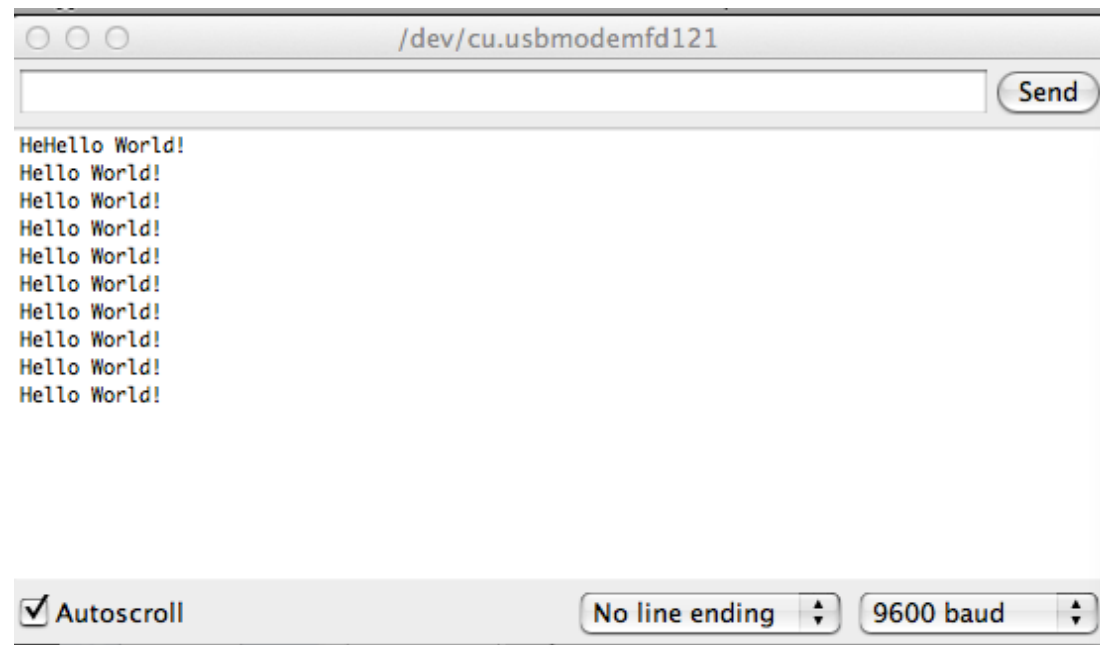


# How to write a “Hello world” program?

# “Hello World” program

```
void setup()
{
  Serial.begin(9600); //start serial comm.
}

void loop()
{
  Serial.print("Hello World!\n\r");
  delay(2000); //wait 2sec
}
```



# How to use digital I/O pins

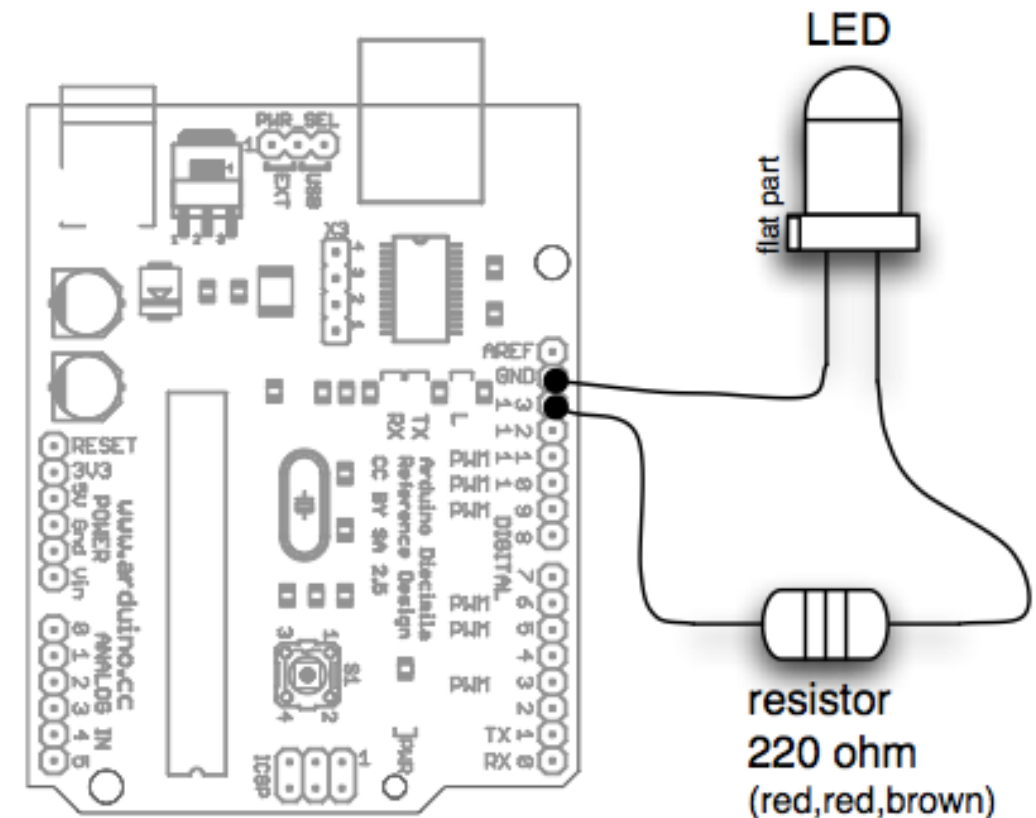


# “Hello World” with a LED

```
int ledPin = 13; // LED connected to pin 13
```

```
void setup()
{
    //start serial port at 9600 bps:
    Serial.begin(9600);
}

void loop()
{
    Serial.print("Hello World!\n\r");
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```



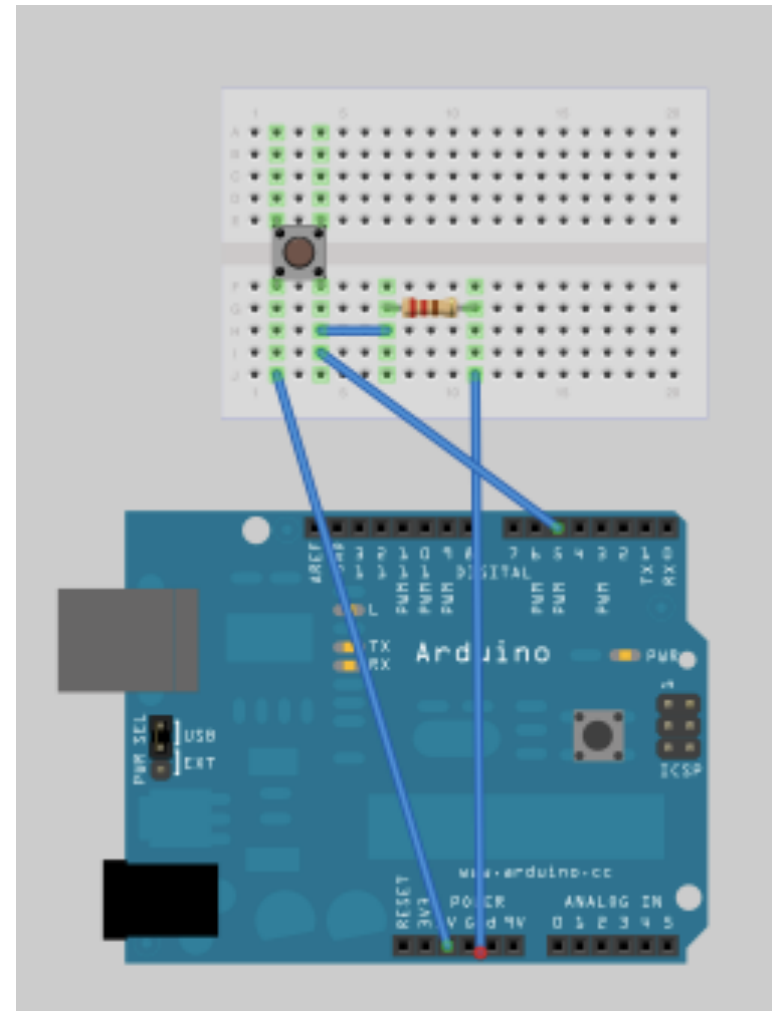
**wiring diagram**

# Reading input button

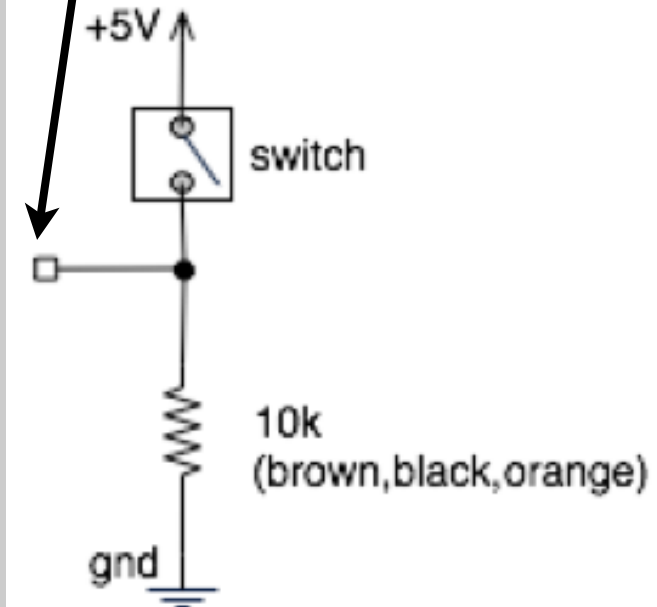
```
int pinUT=5;
int data=0;

void setup()
{
    //pin5 in set to receive data
    pinMode(pinUT, INPUT);
    //initiate serial communication
    Serial.begin(9600);
}

void loop() {
    data=digitalRead(pinUT);
    Serial.println(data);
    //wait for a second
    delay(1000);
}
```



connect to  
digital pin 5



Serial monitor will report “1”  
when button is pressed.  
It will report “0” otherwise.

# How to establish a serial connection



# Serially controlled LED

---

- It is possible to control the Arduino by a serial connection.
- Every time you press a key, it will assign a decimal value (ASCII).
- That value is sent to the Arduino through a serial connection.
- ... **But aren't we using a USB cable?**

# ASCII Table

---

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

# Character 'a' will turn LED ON

---

```
int ledPin = 13; // LED connected to digital pin 13
int inByte = 0;
void setup()
{
    pinMode(ledPin, OUTPUT); // sets the digital pin as output
    Serial.begin(9600); // initiate serial communication
}

void loop()
{
    while (Serial.available() > 0) { inByte = Serial.read(); }
    Serial.println(inByte); //print contents of inByte

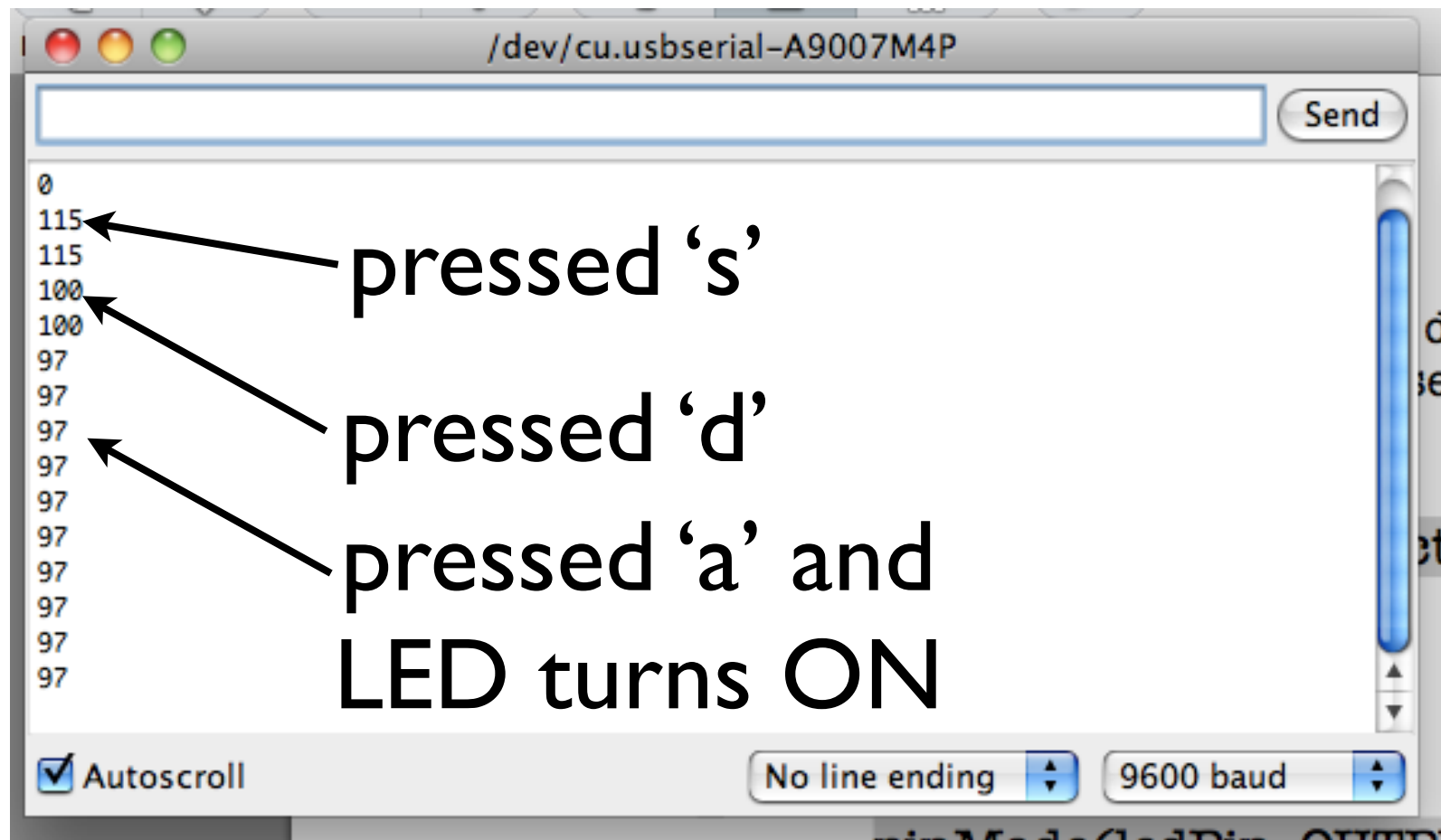
    if (inByte == 97) { digitalWrite(ledPin, HIGH); } //sets the LED ON

    else { digitalWrite(ledPin, LOW); } //sets the LED ON

    delay(1000); //wait 1 second
}
```



# Serial I/O



# How to use analog I/O pins

# PWM Signals

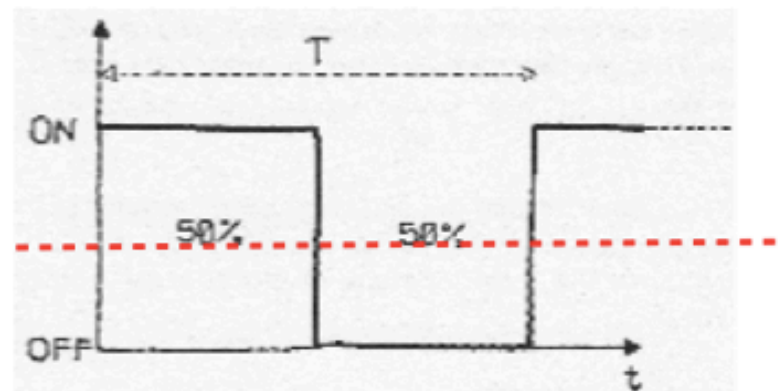
---

- Pulse Width Modulated (PWM) Signals.
- $\mu$ Cs cannot generate analog output, but we can fake it by creating digital signals with different “duty cycles” - signals with different pulse widths.
- To the analog world the different duty cycles create different effective voltages.
- **Note #1:** Analog output can only happens on digital pins through PWM.
- **Note #2:** Analog input can only happen on analog pins!



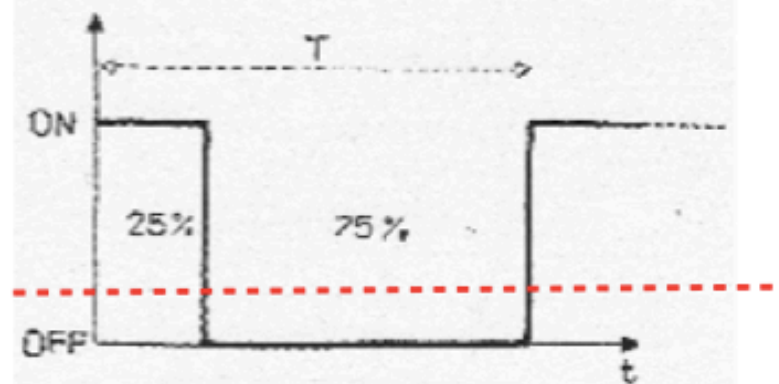
# PWM Signals

50% Duty Cycle



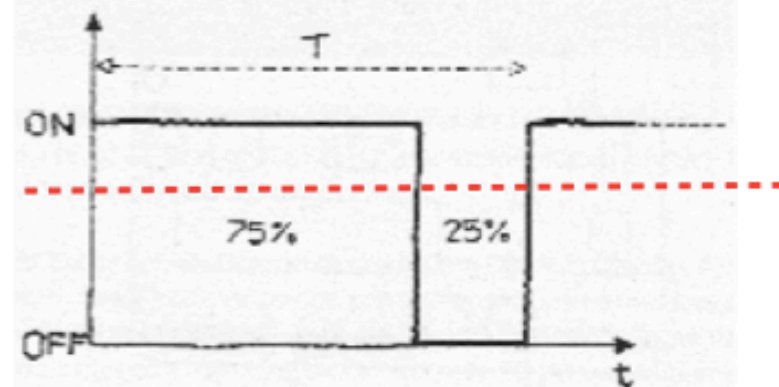
Effective  
Voltage

25% Duty Cycle



Effective  
Voltage

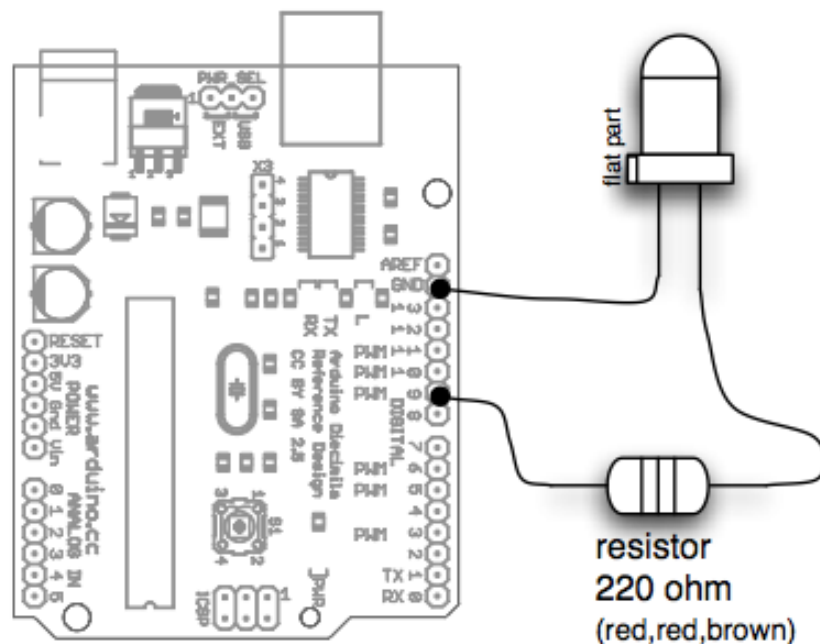
75% Duty Cycle



Effective  
Voltage

# Analog output: varying LED brightness

- Analog I/O has a range of values.
- Output: 0 ... 255 (256 voltage steps from 0 to 5V).
- Input: 0 ... 1023 (1024 voltage steps from 0 to 5V).



wiring diagram

## Code Example:

```
int ledPin=9;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  int value=0;

  //loops value from 0 to 255
  for (value=0; value<255;value++)
  {
    //write the index of the loop to the pin
    analogWrite(ledPin,value);
    //wait 10ms
    delay(10);
  }
}
```

Note:  
0 = 0V  
255 = 5V

# Sample PWM / “Analog” write code

---

```
int outputPin = 9; //output pin (led)

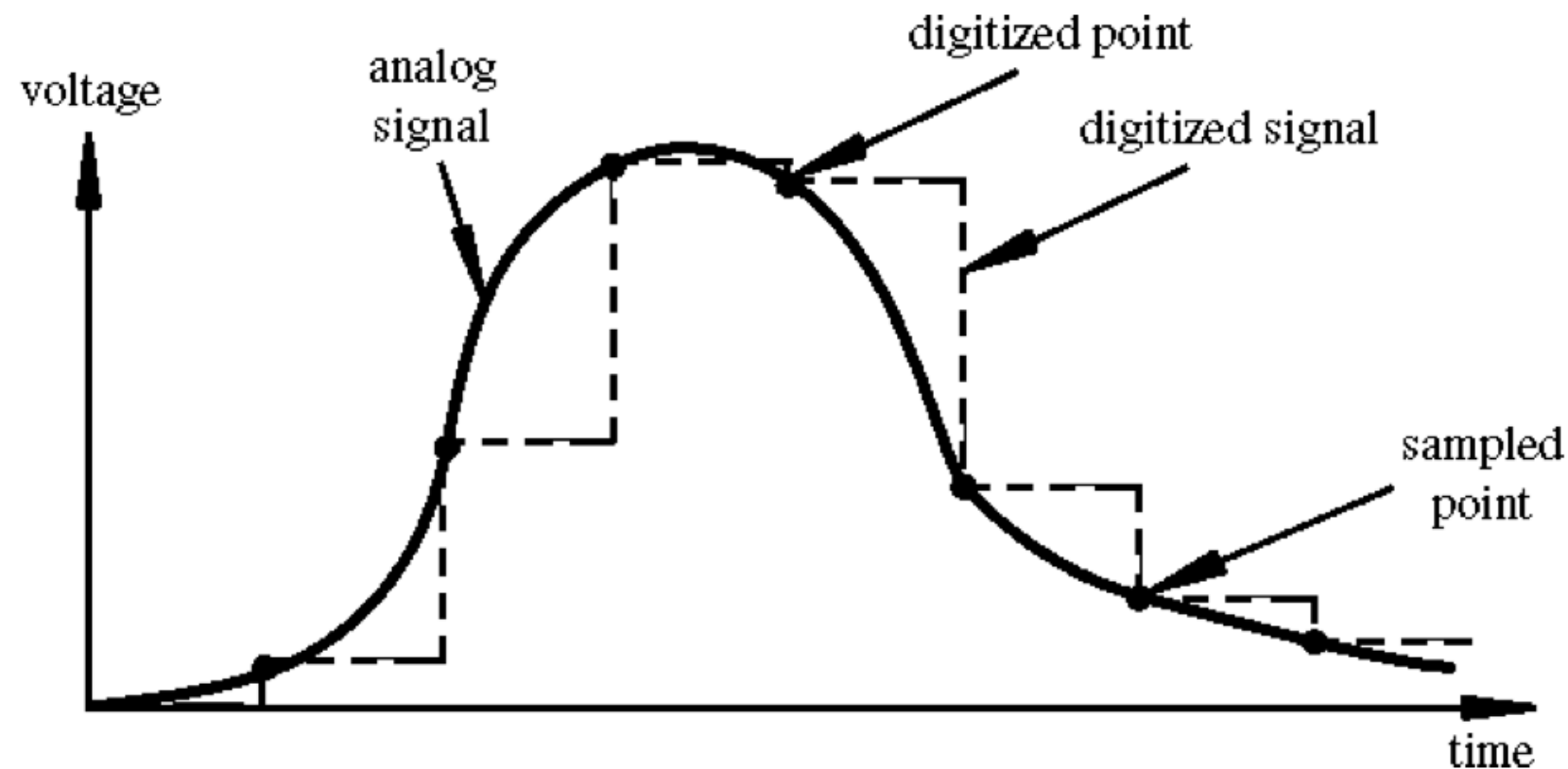
void setup()
{
    pinMode(ledPinA, OUTPUT);
}

// the loop() method runs over and over again,
void loop()
{
    //analogValue of 255 will output 5V on pin 9
    //analogValue of 0    will output 0V on pin 9
    int analogValue=255;
    analogWrite(outputPin,analogValue) ;
}
```



# Analog input

---

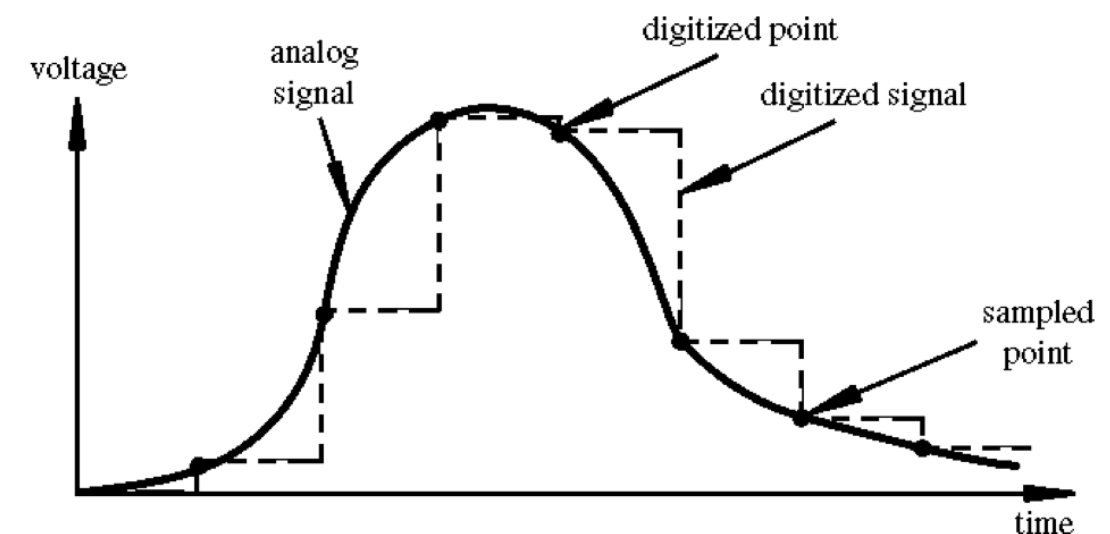


Arduino splits analog signals  
into digital points

# Analog input

---

- Many states, not just two (HIGH/LOW)
- Number of states (or values, or “bins”) is resolution
- Common computer resolutions:
  - 8-bit =  $2^8 = 256$  values
  - 16-bit =  $2^{16} = 65,536$  values
  - 32-bit =  $2^{32} = 4,294,967,296$  values



# Analog input

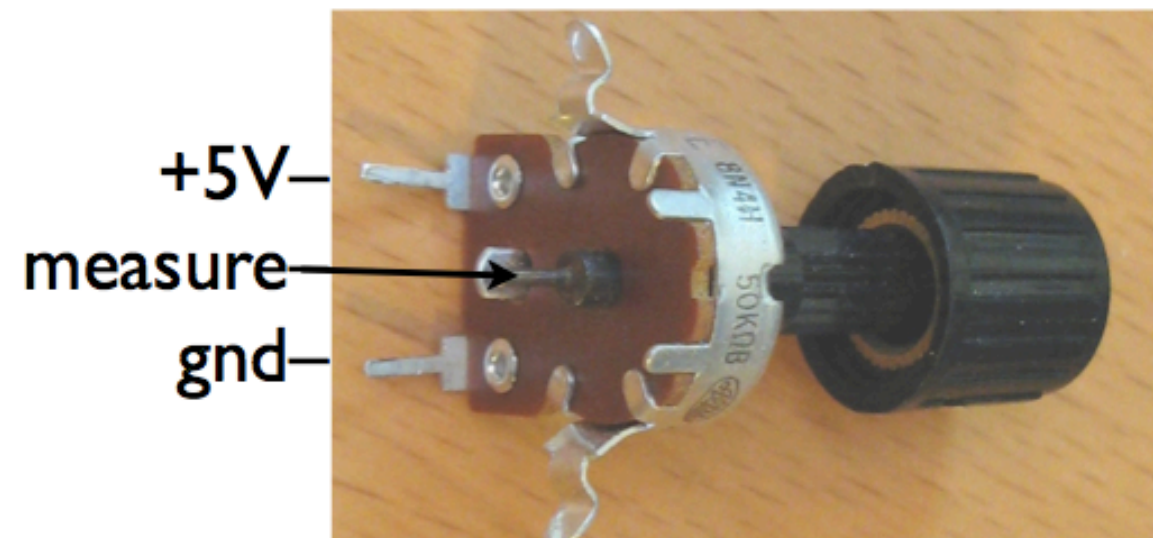
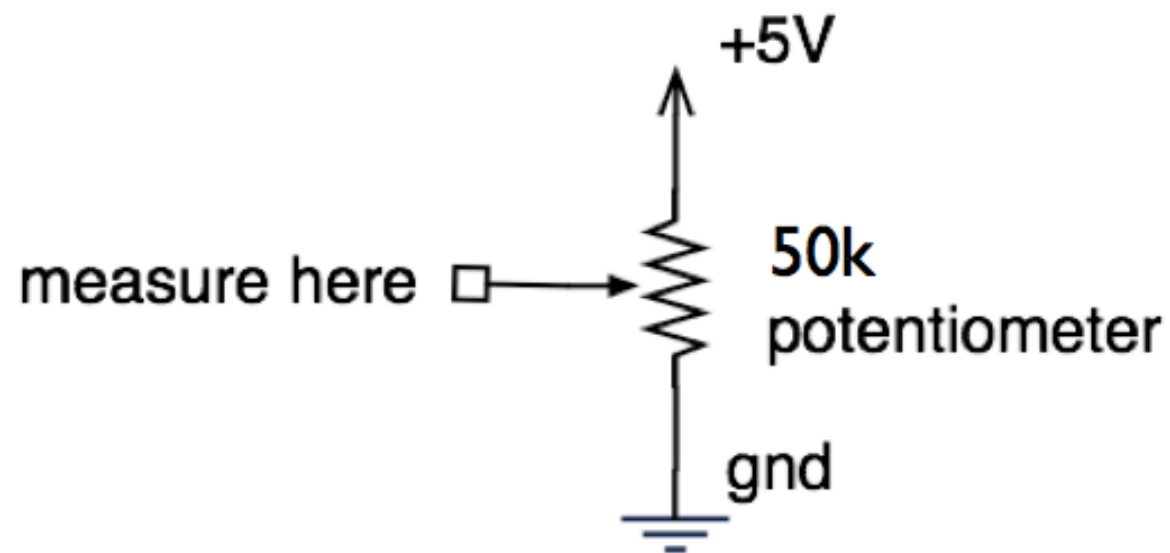
---

- Arduino (ATmega328) has six ADC inputs (ADC = Analog to Digital Converter)
- Reads voltage between 0 to 5 volts
- Resolution is 10-bit (1024 values)
- In other words,  $5/1024 = 4.8$  mV smallest voltage change you can measure
- Examples of varying voltage inputs: Temperature sensors, light sensors, ...

# Potentiometers

---

With a potentiometer we can create an analog input



The pot you have



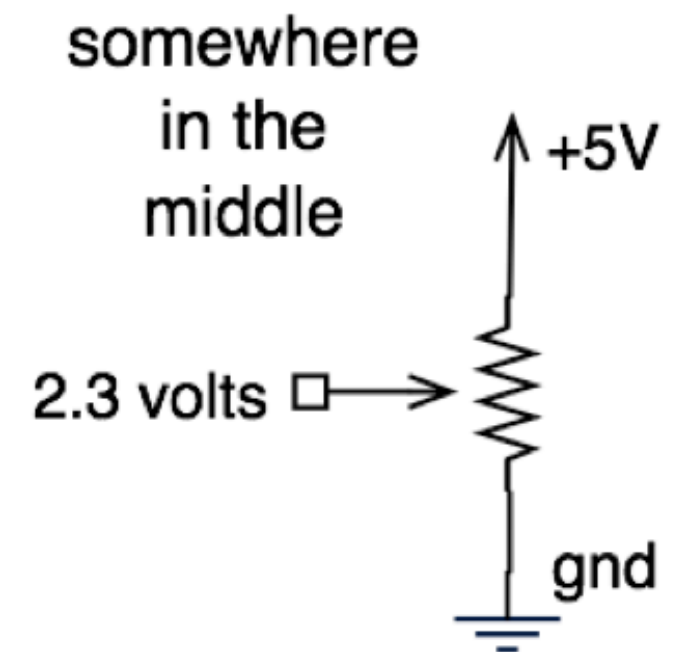
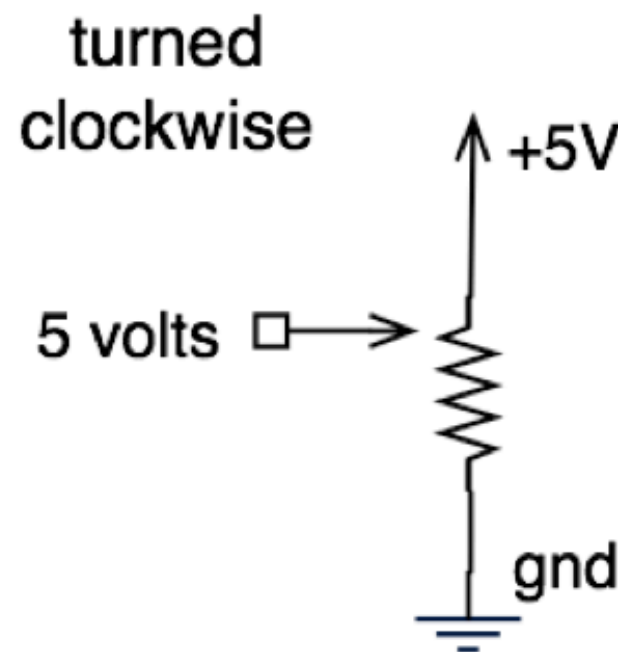
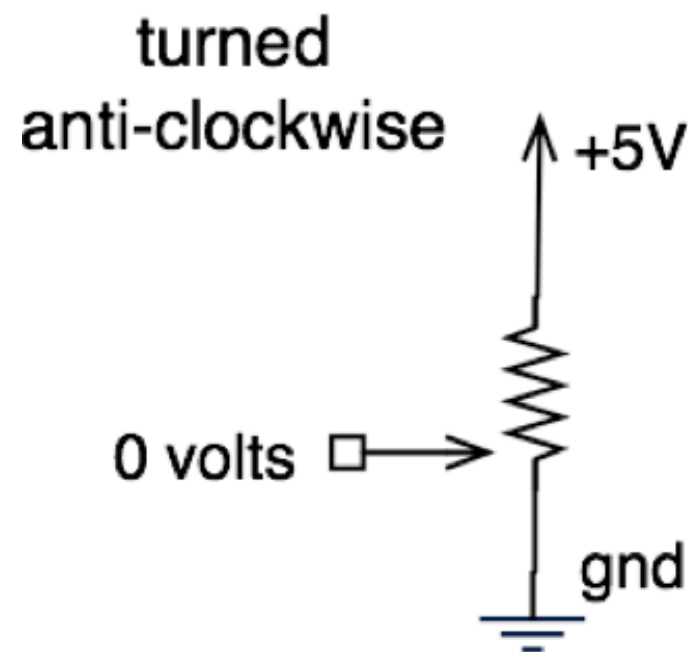
pots also look like this



# Potentiometers

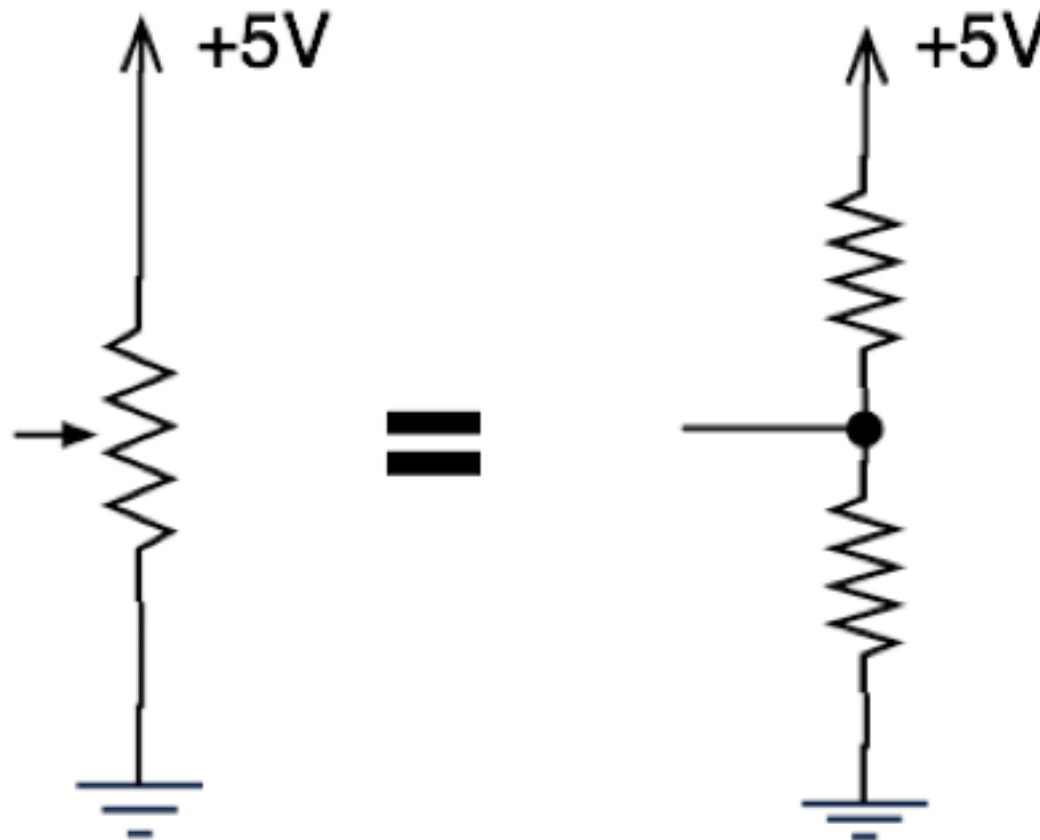
---

Moving the knob is like moving where the arrow taps the voltage on the resistor



# Voltage divider

- Potentiometers are example of a voltage divider
- Voltage divider splits a voltage in two
- Same as two resistors, but you can vary them

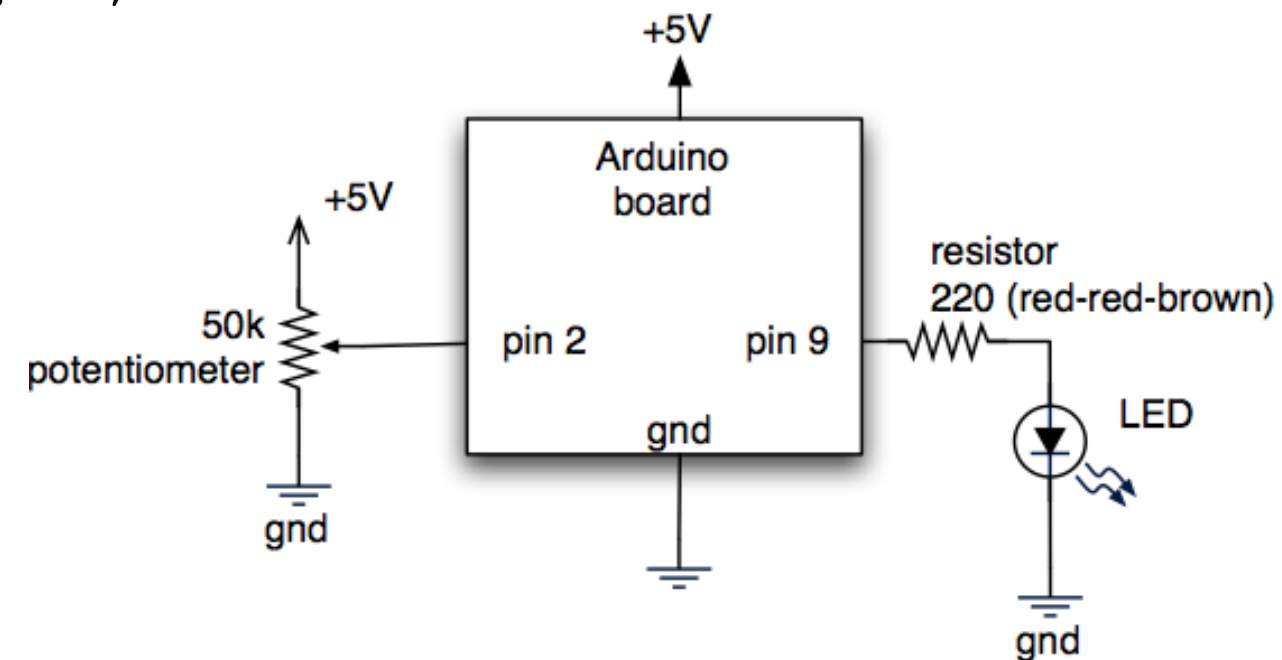


# Sample analog read code

```
int ledPinA = 9; //output pin (led)
int sensorPinA = 2; // input pin for the potentiometer
int sensorValueA = 0; //analog valueA

void setup() {
  Serial.begin(9600);
  pinMode(ledPinA, OUTPUT);
}

// the loop() method runs over and over again,
void loop()
{
  sensorValueA = analogRead(sensorPinA);
  digitalWrite(ledPinA, HIGH);
  Serial.println(sensorValueA);
  delay(sensorValueA);
  digitalWrite(ledPinA, LOW);
  delay(sensorValueA);
}
```



# Sensing the dark: photocells

---

- aka. photoresistor, light-dependent resistor
- A variable resistor
- Brighter light == lower resistance
- Photocells you have range approx. 0-10k-1M



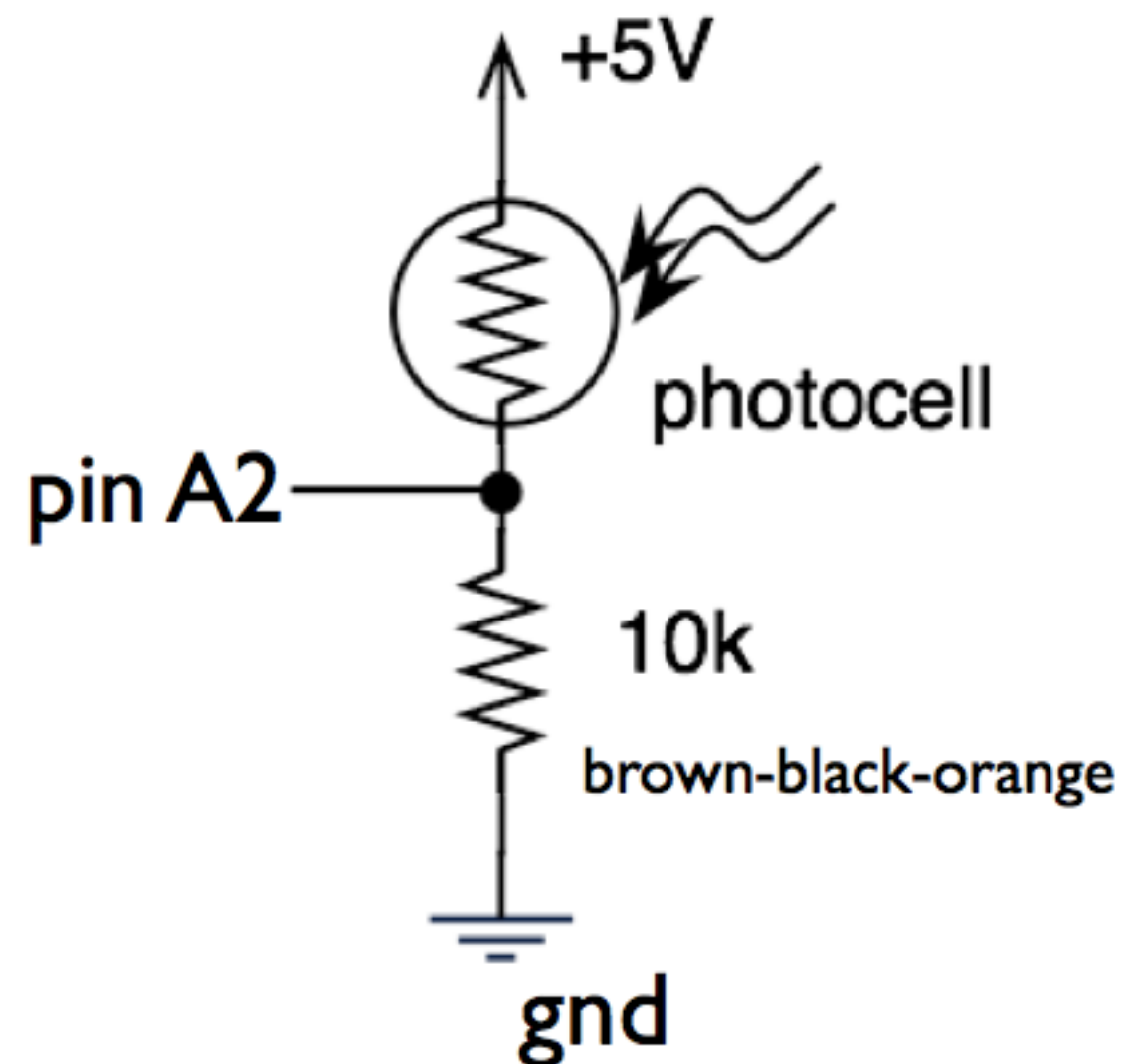
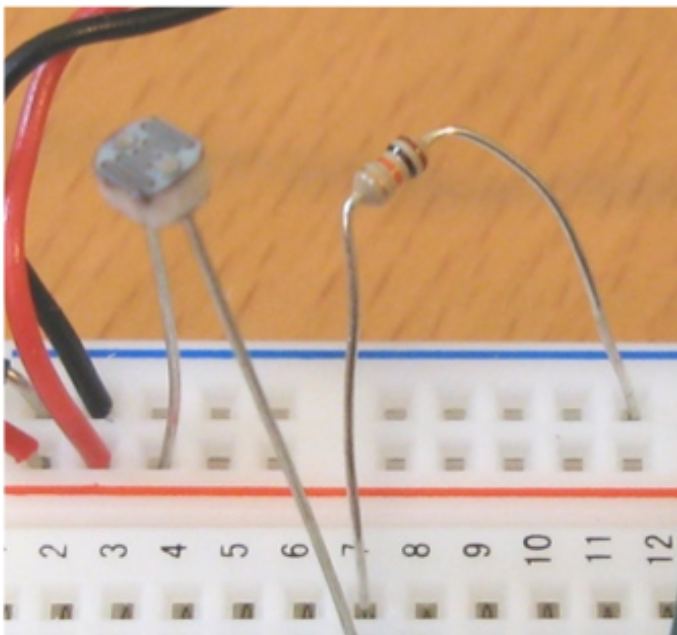
schematic symbol





# Photocell circuit

- Implementation is very similar to a potentiometer
- Pin A2 is the input to an analog port.

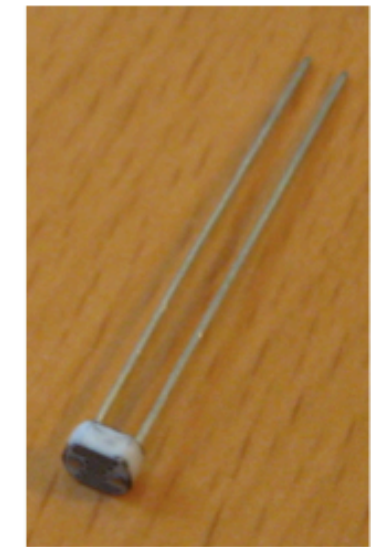
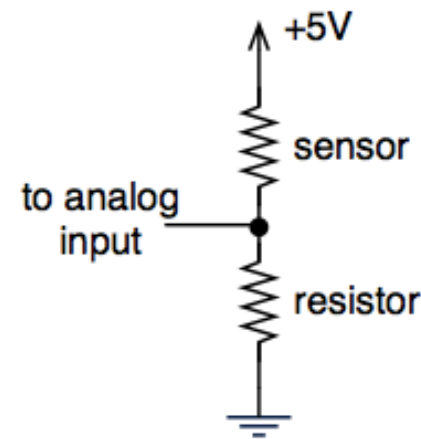


# Resistive sensors

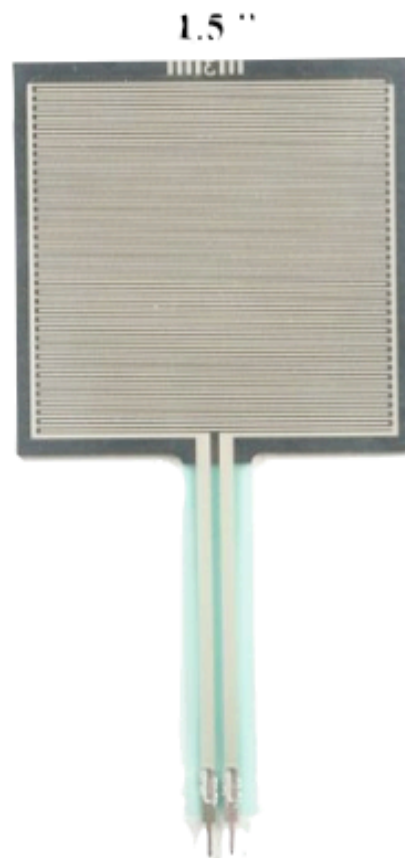


thermistor  
(temperature)

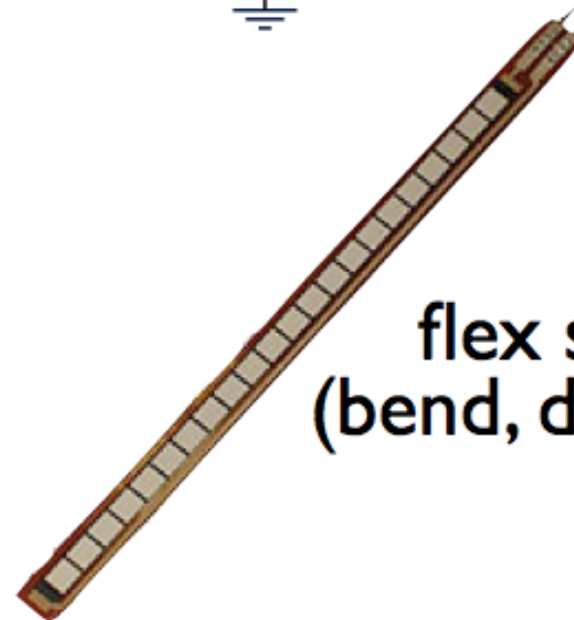
circuit is the same  
for all these



photocell  
(light)



force sensors  
(pressure)



flex sensor  
(bend, deflection)

also air pressure  
and others

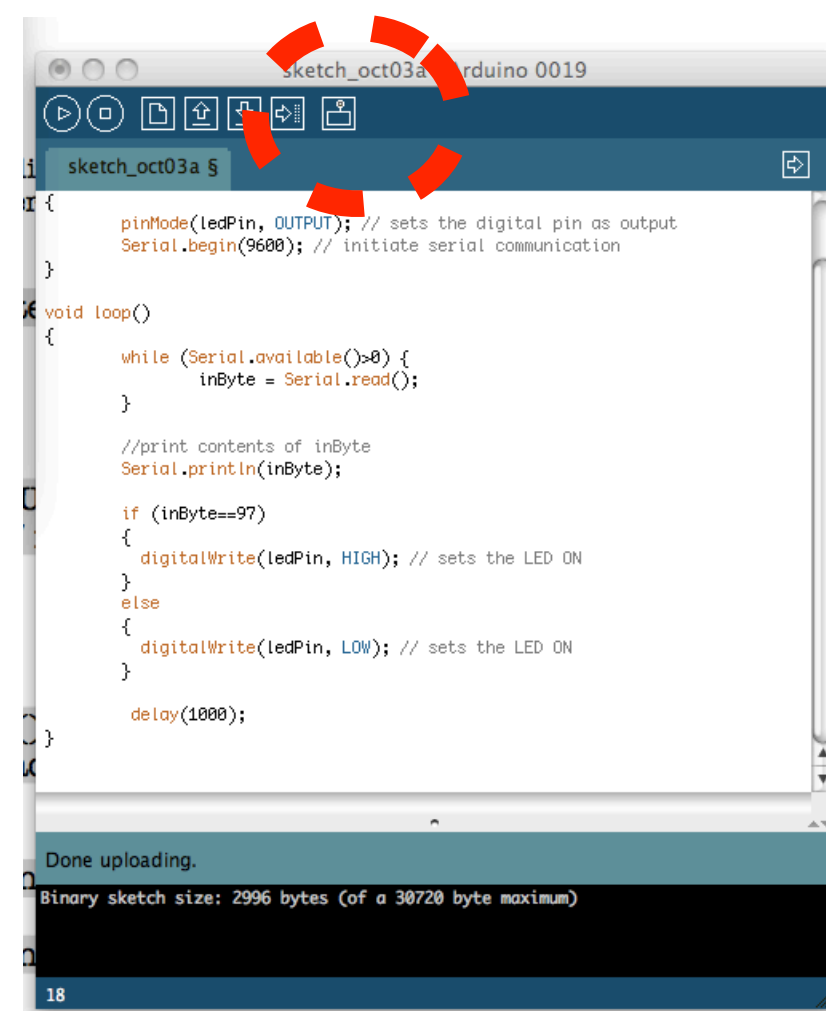
# Resistive sensor code

---

```
int sensorPinA = 2; // input pin for the
potentiometer
int sensorValueA = 0; //analog valueA

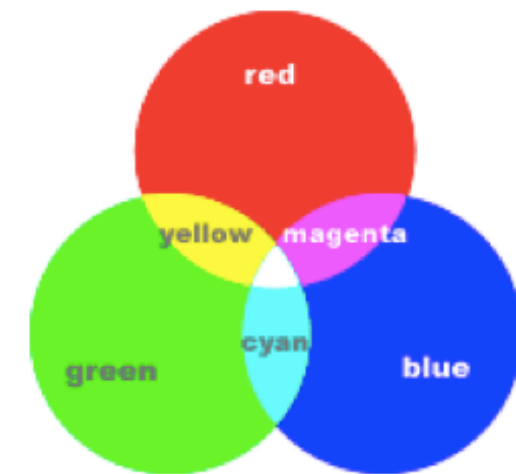
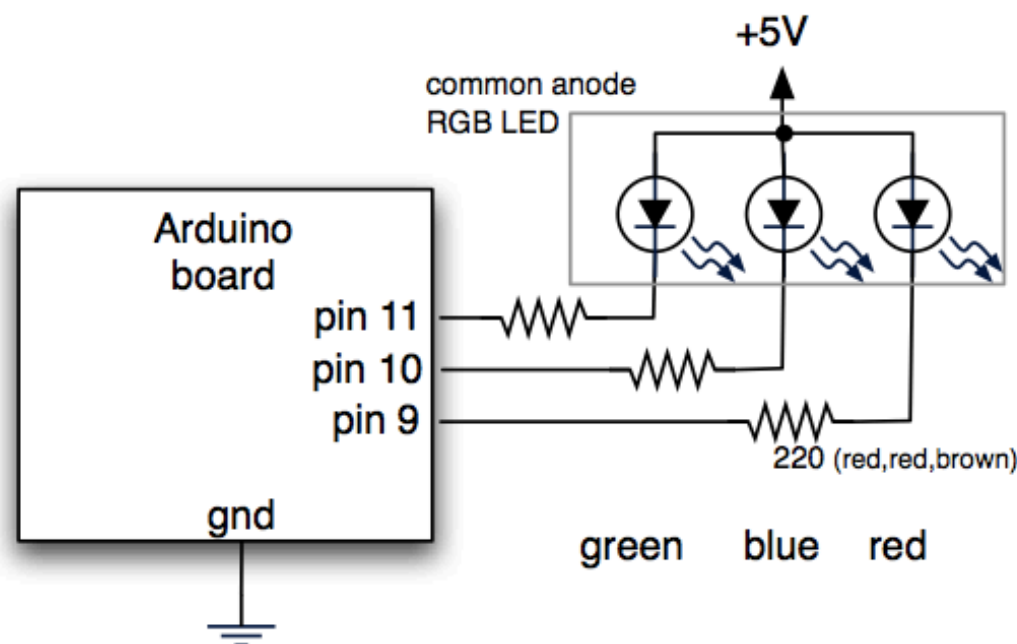
void setup()    {
    Serial.begin(9600);
}

// the loop() method runs over and over again,
void loop()
{
    sensorValueA = analogRead(sensorPinA);
    Serial.println(sensorValueA);
    delay(100);
}
```



# RGB LEDs

With just 3 LEDs you can make any\* color



With RGB you can  
make any color  
(except black)

Mixing light is the additive color model

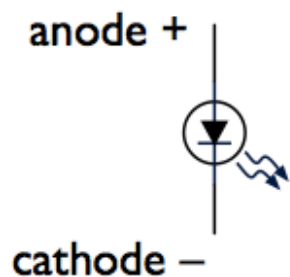
(paint is subtractive color, and can give you brown)



# RGB LEDs are perfect for analog output pins

---

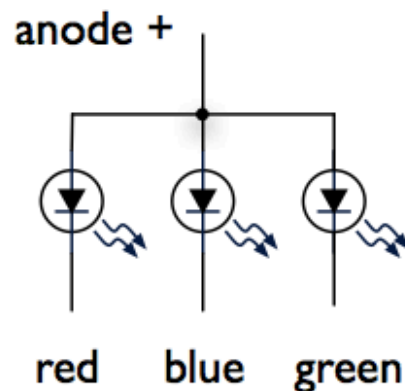
## Normal LED



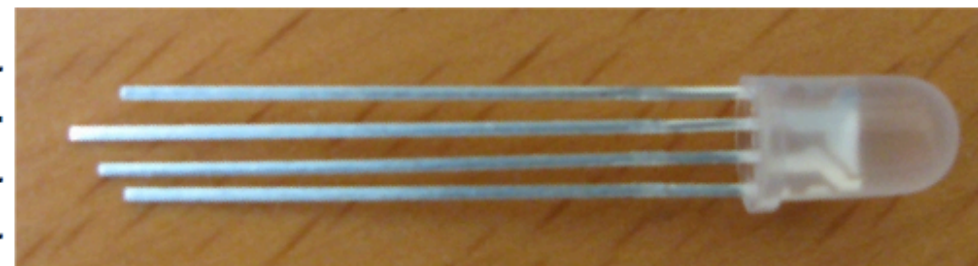
anode +  
cathode -



## RGB LED



red cathode -  
anode +  
green cathode -  
blue cathode -



actually 3 LEDs in one package

# Here is an example of RGB LED usage

---

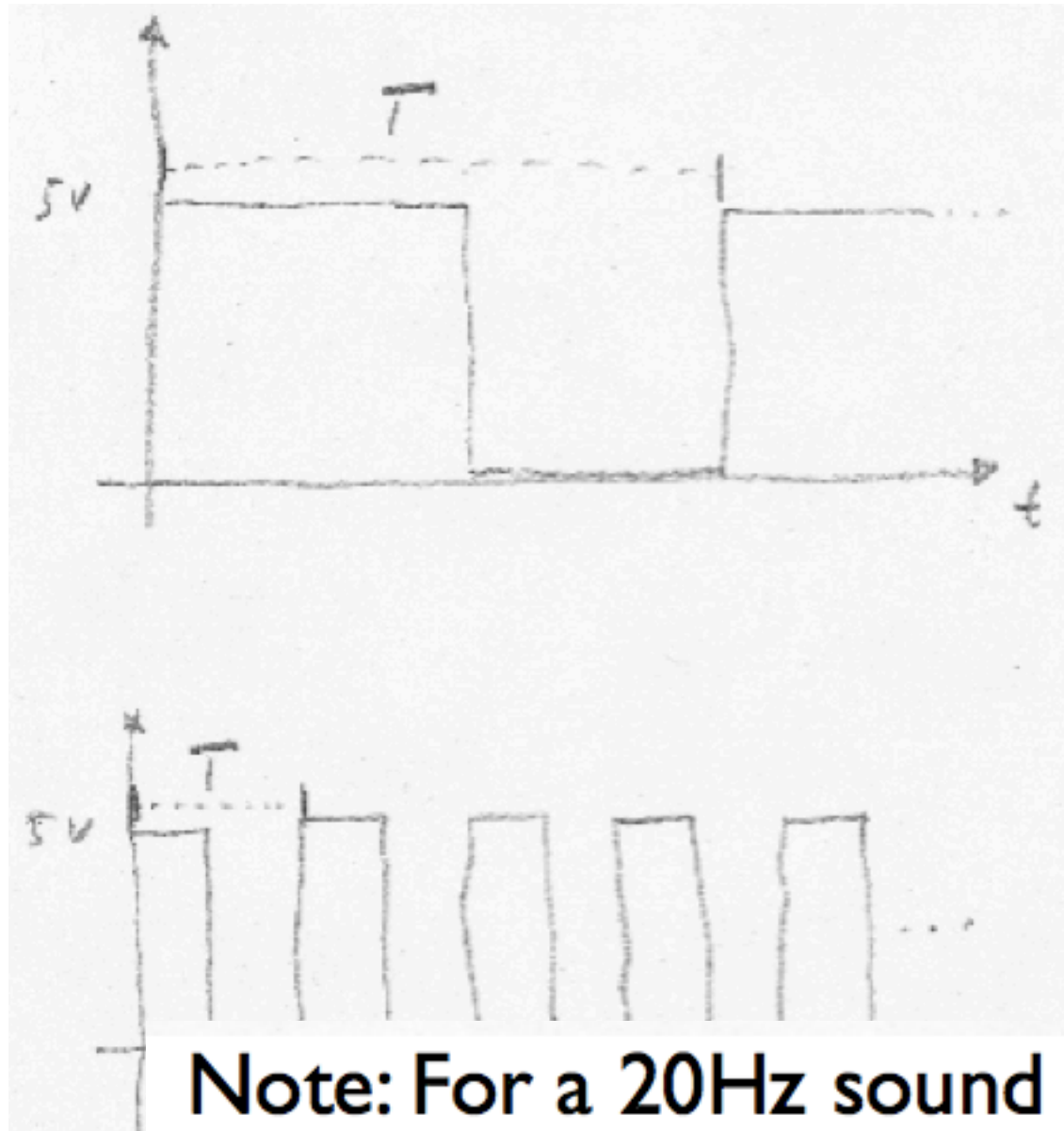
16x16x16 LED Cube



<http://www.youtube.com/watch?v=3K8i0JQzx2w&>

# Other sample introductory code

# Simulating sound waves



Low pitched tone – long  
period  $T$

High pitched tone – short  
period  $T$

**Note:** For a 20Hz sound wave we have  $T = 50\text{ms}$ ,  
for a 200Hz sound wave we have  $T = 5\text{ms}$ .



# Make a theremin

---

- “ooo-weee-ooooo”
- The original spooky sound machine
- Works by measuring your body’s electric field
- We’ll use a resistive sensor in lieu of RF
- <http://www.youtube.com/watch?v=pSzTPGINa5U>

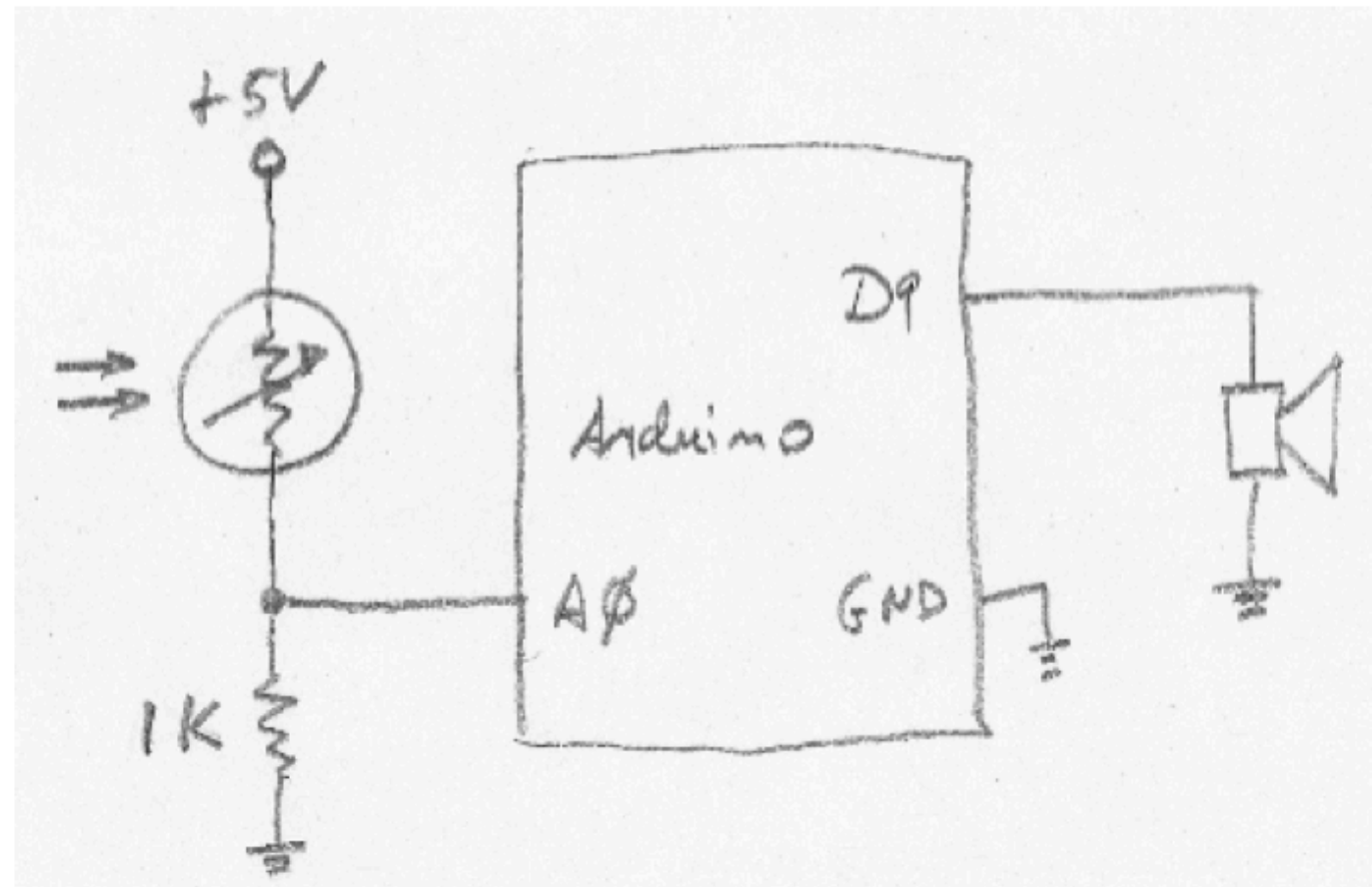


*Leon Theremin*

# Theremin

---

- Read an analog signal generated through a resistive sensor
- We interpret the digitized value from the A/D conversion as the period of the sound wave we want to generate
- Generate one period of the sound wave, output it to the speaker and then sample the input again



# Theremin code

---

```
int SpeakerPin = 9; //output pin (led)
int sensorPin = 2;  // analog input pin
int sensorValue = 0; //analog value

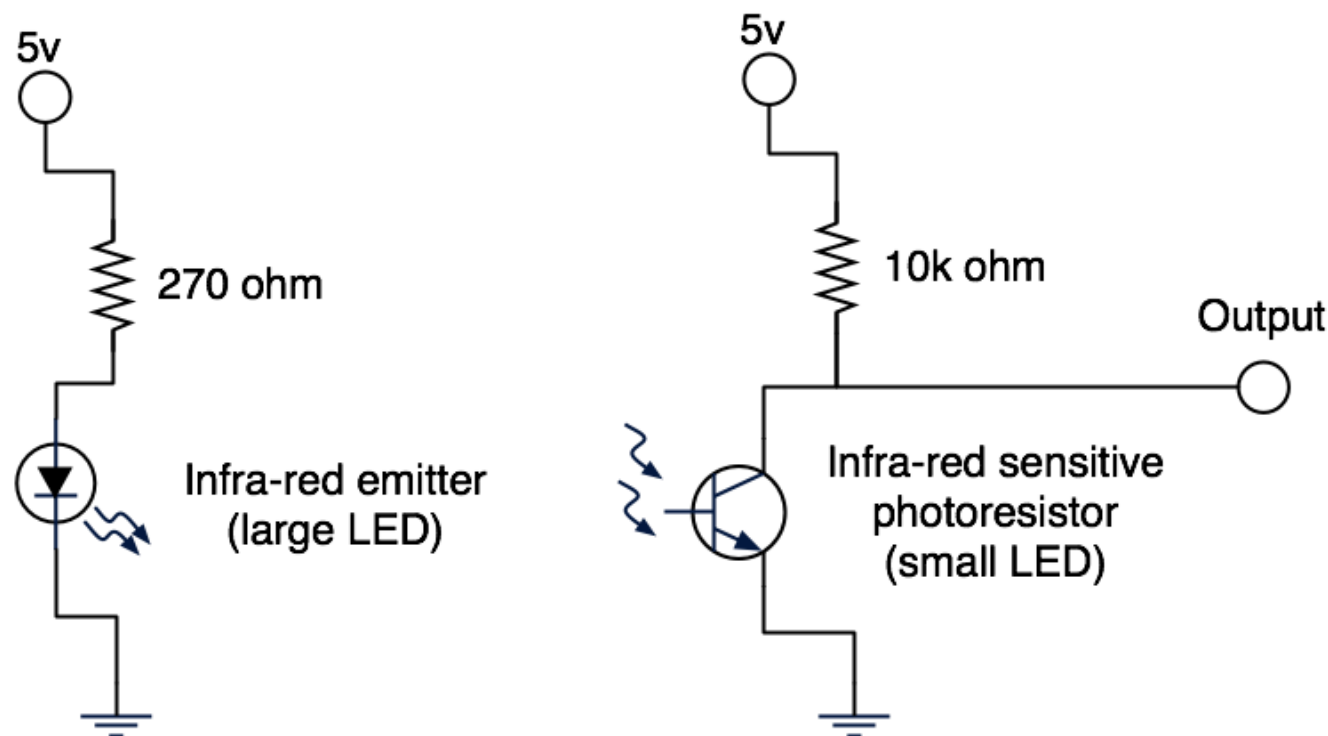
void setup() {
    Serial.begin(9600);
    pinMode(SpeakerPin, OUTPUT);
}

void loop()
{
    updateRate=50;
    sensorValue = analogRead(sensorPin);
    sensorValue = sensorValue*2;
    Serial.println(sensorValue);

    for (int i=0; i<updateRate ; i++)
    {
        digitalWrite(SpeakerPin,HIGH);
        delayMicroseconds(sensorValue);
        digitalWrite(SpeakerPin,LOW);
        delayMicroseconds(sensorValue);
    }
}
```

- Variable `updateRate` is number of cycles sound will play.
- Play with `sensorValue` & `updateRate` variables to alter sensitivity, pitch and timbre.

# Infrared (IR) emitter and detector



An infra-red detector is a photodetector that reacts to infrared (IR) radiation.

- You can't see infra-red light but you can detect it.
- IR detectors / emitters are used in remote controls
- The Infra-red detector behaves like a transistor.
  - No IR detection: transistor will act like an open circuit with output = HIGH.
  - IR detection: some current will flow to ground, so output < HIGH.

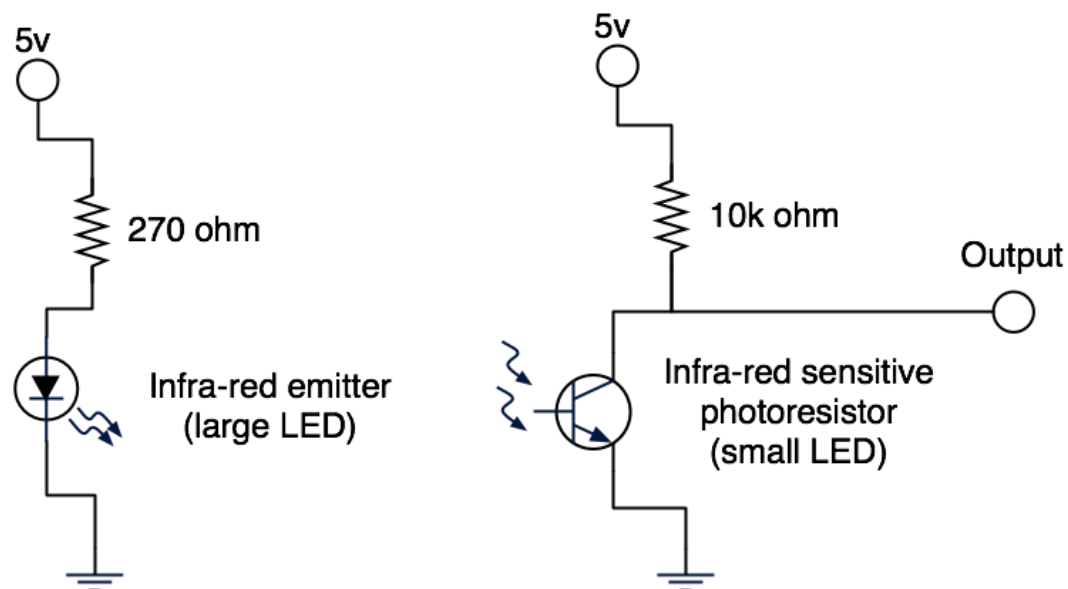


# IR emitter and detector code

```
int IR_input_pin = 5; //analog pin
int IR_data=0;

void setup() {
    Serial.begin(9600);
}

void loop()
{
    IR_data=analogRead(IR_input_pin);
    Serial.println(IR_data);
    delay(1000);
}
```



1. We can implement the emitter and detector circuits in two separate prototype boards.

2. IR emitters and detectors are very sensitive to current.

3. Connect the detector output to your Arduino analog pin 5.

4. Load the code, and you will detect a voltage on pin 5.

# Basic C review

## Loops, variables, arrays

# Hello World

## Brackets

define code blocks

```
void setup() {  
  Serial.begin(9600);  
  Serial.print("Hello World\n");  
  Serial.println("Hello World");  
}
```

```
void loop()  
{  
}  
}
```

## Brackets

define code blocks

**Note: Everything that is not a code block must end with a ;**

1. Arduino programs **MUST** have the setup and loop functions.
2. Functions always begin with **{** and end with **}**.
3. When the code is uploaded to the board, whatever is in the setup function will be executed first, and only once.
4. After executing the setup function, the program flow will go to the loop function, which will repeat until there is power.
5. Serial.print and Serial.println both print stuff to the screen. Serial.println will automatically insert a line feed.

# Commenting your code

---

```
/*  
This is a very long comment  
that takes multiple lines...  
*/  
  
void setup() {  
    Serial.begin(9600);  
    Serial.print("Hello World\n");  
    //this is a small comment  
    Serial.println("Hello World");  
}  
  
void loop()  
{  
}
```

1. It is considered good programming style to comment code.
2. Comments can be written anywhere in the code: any characters between `/*` and `*/` are ignored by the compiler and can be used to make the code easier to understand.
3. Comments can also be written after a `//`
4. Also, make sure to insert tabs in your code to make it align inside functions.



# C doesn't care much about spaces

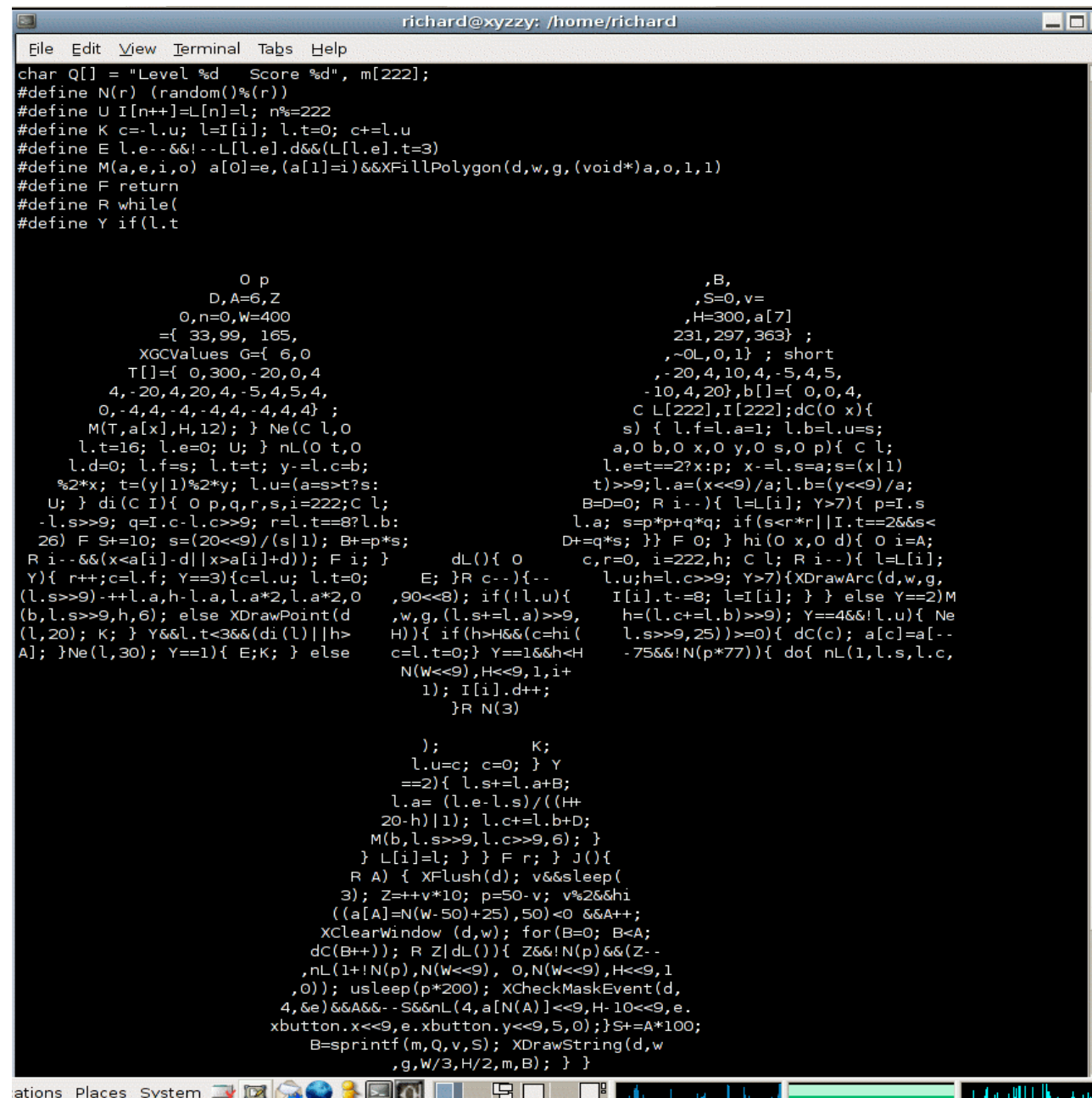
---

```
void setup() { Serial.begin(9600);  
Serial.print("Hello World\n");  
Serial.println("Hello World");} void  
loop() {}
```

```
void setup() {  
    Serial.begin(9600);  
    Serial.print("Hello World\n");  
    Serial.println("Hello World");  
}  
void loop()  
{  
}
```

1. Both of these programs are exactly the same as the original as far as your compiler is concerned.
2. Note that words have to be kept together and so do things in quotes.
3. How we **SHOULD** lay out our C program to make it look nice?

# Bad Coding Style



```
richard@xyzy: /home/richard
File Edit View Terminal Tabs Help
char Q[] = "Level %d   Score %d", m[222];
#define N(r) (random()%(r))
#define U I[n++] = L[n] = l; n%=222
#define K c = l.u; l = I[i]; l.t = 0; c += l.u
#define E l.e = &&! --L[l.e].d &&(L[l.e].t = 3)
#define M(a,e,i,o) a[0] = e, (a[1] = i) && XFillPolygon(d,w,g, (void*)a, 0, 1, 1)
#define F return
#define R while(
#define Y if(l.t

    O p
    D,A=6,Z
    O,n=0,W=400
    =( 33,99, 165,
    XGCValues G={ 6,0
    T[]={ 0,300,-20,0,4
    4,-20,4,20,4,-5,4,5,4,
    0,-4,4,-4,-4,4,-4,4,4} ;
    M(T,a[x],H,12); } Ne(C l,0
    l.t=16; l.e=0; U; } nL(0 t,0
    l.d=0; l.f=s; l.t=t; y = l.c=b;
    %2*x; t=(y|1)%2*y; l.u=(a>s?t?s:
    U; } di(C I){ O p,q,r,s,i=222;C l;
    -l.s>>9; q=I.c-l.c>>9; r=l.t==8?l.b:
    26) F S+=10; s=(20<<9)/(s|1); B+=p*s;
    R i--&&(x<a[i]-d||x>a[i]+d)); F i; }
    Y){ r++;c=l.f; Y==3){c=l.u; l.t=0;
    (l.s>>9)-++l.a,h-l.a,l.a*2,l.a*2,0
    (b,l.s>>9,h,6); else XDrawPoint(d
    (l,20); K; } Y&&l.t<3&&(di(l)||h>
    A]; }Ne(l,30); Y==1){ E;K; } else

    dL(){ O
    E; }R c--){--
    ,90<<8); if(!l.u){
    ,w,g,(l.s+=l.a)>>9,
    H){ if(h>H&&(c=hi(
    c=l.t=0; } Y=1&&h<H
    N(W<<9),H<<9,l,i+
    1); I[i].d++;
    }R N(3)

    ); K;
    l.u=c; c=0; } Y
    ==2){ l.s+=l.a+B;
    l.a=(l.e-l.s)/((H+
    20-h)|1); l.c+=l.b+D;
    M(b,l.s>>9,l.c>>9,6); }
    } L[i]=l; } } F r; } J(){
    R A { XFlush(d); v&&sleep(
    3); Z+=v*10; p=50-v; v%2&&hi
    ((a[A]=N(W-50)+25),50)<0 &&A++;
    XClearWindow (d,w); for(B=0; B<A;
    dC(B++)); R Z|dL(){ Z&&!N(p)&&(Z-
    ,nL(1+!N(p),N(W<<9), 0,N(W<<9),H<<9,1
    ,0)); usleep(p*200); XCheckMaskEvent(d,
    4,&e)&&A&&- S&&nL(4,a[N(A)]<<9,H-10<<9,e.
    xbutton.x<<9,e.xbutton.y<<9,5,0); }S+=A*100;
    B=sprintf(m,Q,v,S); XDrawString(d,w
    ,g,W/3,H/2,m,B); } }
```

# Basic C operands

---

## Control Structures

- + if
- + if...else
- + for
- + switch case
- + while
- + do... while
- + break
- + continue
- + return
- + goto

## Comparison Operators

- + == (equal to)
- + != (not equal to)
- + < (less than)
- + > (greater than)
- + <= (less than or equal to)
- + >= (greater than or equal to)

## Further Syntax

- + ; (semicolon)
- + {} (curly braces)
- + // (single line comment)
- + /\* \*/ (multi-line comment)
- + #define
- + #include

## Boolean Operators

- + && (and)
- + || (or)
- + ! (not)

## Arithmetic Operators

- + = (assignment operator)
- + + (addition)
- + - (subtraction)
- + \* (multiplication)
- + / (division)
- + % (modulo)

## Compound Operators

- + ++ (increment)
- + -- (decrement)
- + += (compound addition)
- + -= (compound subtraction)
- + \*= (compound multiplication)
- + /= (compound division)
  
- + &= (compound bitwise and)
- + |= (compound bitwise or)

# Variables

---

```
/*  
creates a variable whose name is pin,  
whose value is 13, and whose type is  
int.  
*/  
  
int pin = 13;  
  
void setup() {  
    Serial.begin(9600);  
}  
  
void loop()  
{  
}
```

1. A variable is a place to store a piece of data. It has a name, a value, and a type.
2. We must declare the type of every variable we use in C.
3. Every variable has a type (e.g. int) and a name.
4. Declarations of types should always be together at the top of main or a function.
5. Main types of variables are boolean, char, byte, float, string, ...



# Naming variables

---

```
/*  
creates a variable whose name is pin,  
whose value is 13, and whose type is  
int.  
*/  
  
int pin = 13;  
  
//naming a variable abc is cryptic  
int abc = 12;  
  
void setup() {  
    Serial.begin(9600);  
}  
  
void loop()  
{  
}
```

1. Variables in C can be given any name made from numbers, letters and underlines which is not a keyword and does not begin with a number.
2. A good name for your variables is important.
3. Ideally, a comment with each variable name helps people know what they do.
4. Good programmers use well chosen variable names and comments on variables.

# ASCII Table

---

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

# Variable types: int & char

---

```
/*
creates a variable whose name is pin,
whose value is 13, and whose type is
int.
*/

int pin = 13;

/*
these next two variables are equivalent
*/
char myChar_1 = 'A';
char myChar_2 = 65;

void setup() {
    Serial.begin(9600);
    Serial.println(pin);
}

void loop()
{
}
```

1. Integer (int) variables take 16 bits, and are able to store a number (-32,768 to 32,767).
2. Character (char) variables take 8 bits of memory and stores a character value. They can have decimal values from -128 to 127.
3. Refer to the ASCII table to assign the proper variable value.

# Floating point variable: float

---

```
int x;
int y;
float z;

void setup() {
  Serial.begin(9600);
  /*
   * y will contains 0
   * ints can't hold fractions
   */

  x = 1;
  y = x / 2;
  z = (float)x / 2.0;
  // z now contains .5
  Serial.println(x);
  Serial.println(y);
  Serial.println(z);
}

void loop()
{
}
```

1. float is a datatype for floating-point numbers, a number that has a decimal point.
2. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers.
3. Floating-point numbers can be as large as  $3.4028235E+38$  and as low as  $-3.4028235E+38$ . They are stored as 32 bits (4 bytes) of information.
4. Floats have only 6-7 decimal digits of precision. That means the total number of digits, not the number to the right of the decimal point.

# Beware of the floating point

---

```
float myfloat;  
float sensorCalbrate = 1.117;  
  
void setup()  
{  
    Serial.begin(9600);  
}  
  
void loop()  
{  
}
```

1. Floating point numbers are not exact, and may yield strange results when compared.
2. For example  $6.0 / 3.0$  may not equal 2.0. You should instead check that the absolute value of the difference between the numbers is less than some small number.
3. Floating point math is also much slower than integer math in performing calculations
4. Programmers often go to some lengths to convert floating point calculations to integer math to increase speed.



# Variable type: double

---

```
double sensorCalbrate = 22231.117;

void setup()
{
    Serial.begin(9600);
}

void loop()
{
}
```

1. Double precision floating point number. Occupies 4 bytes.
2. The double implementation on the Arduino is currently exactly the same as the float, with no gain in precision.
3. We can store larger numbers on it.

# More variable types

---

```
signed int myInt  = -12;  
unsigned int myInt =  12;
```

```
long longVar  = 1234;  
short shortVar = 10;
```

```
const float PI = 3.14
```

```
void setup()  
{  
    Serial.begin(9600);  
}
```

```
void loop()  
{  
}
```

1.unsigned means that an int or char value can only be positive. signed means that it can be positive or negative.

2.long means that an int variable has more capacity short means they have less.

3.const means a variable which doesn't vary – useful for physical constants or things like PI.

# Operations on Variables

---

```
int intA      = -12;
int intB      = 12;
float floatC   = 15.11;
float floatD   = -22.11;

int intRes     = 0;
float floatRes = 0;
const float PI = 3.14

void setup()
{
    Serial.begin(9600);
    floatRes = floatC + floatD;
    intRes = intA * intB;
    intRes++; //intRes = intRes +1
    floatRes *= 3;
    floatRes /= PI;
    PI = floatRes + floatC; //ERROR!
}

void loop() {}
```

- With two variables we can +, -, \* and /
- ++ (add one) e.g. increment++;
- -- (subtract one) e.g. countdown--;
- += (add to a variable) e.g. a+= 5;
- -= (subtract from variable) e.g. num\_living-= num\_dead;
- \*= (multiply a variable) e.g. no\_bunnies\*=2;
- /= (divide a variable) e.g. fraction/= divisor;
- (x % y) gives the remainder when x is divided by y

➡ remainder= x%y; (ints only)

# Casting between variables

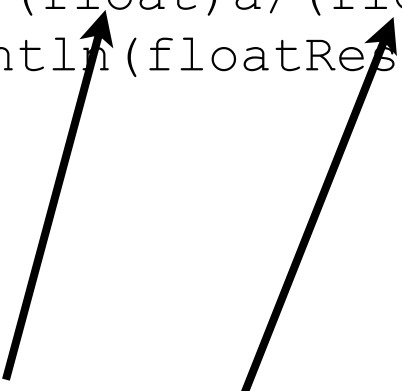
---

```
int a      = 3;
int b      = 5;

float floatRes = 0;

void setup()
{
    Serial.begin(9600);
    floatRes = a/b;
    Serial.println(floatRes);
    floatRes = (float)a/(float)b;
    Serial.println(floatRes);
}

void loop() {}
```



Cast ints a and b to be floats!

- Recall the trouble we had dividing ints
- A cast is a way of telling one variable type to temporarily look like another.
- By using (type) in front of a variable we tell the variable to act like another type of variable.
- We can cast between any type. Usually, however, the only reason to cast is to stop ints being rounded by division.

# Summary of variables

---

- boolean : 1 bit (true or false)
- char : 8 bit (-128 to 127)
- byte or unsigned char: 8 bit (0 to 255)
- int : 16 bit (-32,768 to 32,767)
- word or unsigned int : 16 bit (0 to 65535)
- long: 32 bit (-2,147,483,648 to 2,147,483,648)
- unsigned long : 32 bit (0 to  $2^{32} - 1$ )
- float : 32 bits (3.4028235E+38 to -3.4028235E+38)
- double : same as float in the Arduino environment



# If statements

---

```
void setup()
{
    Serial.begin(9600);

    int a      = -12;
    int b      = 12;

    if (a >= b)
    { Serial.println (">="); }

    if (a == b)
    { Serial.println ("=="); }

    if (a < b)
    { Serial.println ("<"); }
}

void loop() {}
```

With *if statements* we are comparing two variables!

```
If (variable comparison operation variable) {
    some code here
}
else {
    some code here
}
```

Where *comparison operation* can be:

- > (greater than)
- < (less than)
- >= (greater than or equal)
- <= (less than or equal)
- == (equals)
- != (not equals)

# Reading serial data in different formats

---

```
int readKey = 0;
void setup()
{
  //initiate serial communication
  Serial.begin(9600);
}

void loop()
{
  while (Serial.available() > 0)
  {
    readKey = Serial.read();
  }

  //report what key was printed
  Serial.println(readKey, BYTE); //prints "N"
  Serial.println(readKey, BIN); //prints "1001110"
  Serial.println(readKey, DEC); //prints "78"

  //wait half a second
  delay(500);
}
```

- We can print out the contents of the readKey variable in different formats.
- We can print the character (BYTE), its binary value (BIN) or its decimal value (DEC).

# Functions

---

```
float valA; float valB;

void setup() { Serial.begin(9600); }

float return_maximum(float vA, float vB)
{
    if (vA > vB) { return (vA); }
    else { return (vB); }
}

void loop()
{
    //random number from 10 to 19
    valA = random(10, 20);
    //random number from 0 to 19
    valB = random(0, 20);

    //calling the function
    float res = return_maximum(valA, valB);
    Serial.println(res);

    delay(1000);
}
```

- The function is one of the most basic things to understand in C programming.
- A function is a sub-unit of a program which performs a specific task.
- We have already (without knowing it) seen many functions from the arduino C library – `Serial.println()`, `setup()` and `loop()`.
- We need to learn to write our own functions.
- Functions take arguments (variables) and may return an argument. Think of a function as extending the C language to a new task.

# Functions calling functions

---

```
float valA=12; float valB=3;
```

```
float return_minimum(float vA, float vB)
{
    if (vA > vB) { return (vB); }
    else { return (vA); }
}
```

```
float return_number(float vA, float vB)
{
    float value = return_minimum(vA,vB);
    if (vA > vB) { return (vA + value); }
    else { return (vB - value); }
}
```

```
void setup() {
    Serial.begin(9600);
    float res=return_number(valA,valB);
}
```

```
void loop()
{
```

```
    (. . .)
} CPE 355 - Real Time Embedded Kernels - Spring '12
Nuno Alves (nalves@wne.edu), College of Engineering
```

- Once you have written a function, it can be accessed from other functions.
- We can therefore build more complex functions from simpler functions

# Void functions

---

```
float valA=12.5; float valB=16.1;
```

```
void setup() { Serial.begin(9600); }
```

```
float return_maximum(float vA, float vB)
{
    if (vA > vB) { return (vA); }
    else { return (vB); }
}
```

```
void print_hello_message()
{
    Serial.println("hello world");
}
```

```
void loop()
{
    //calling the function
    float res = return_maximum(valA, valB);
    Serial.println(res);
    print_hello_message();
    delay(1000);
}
```

- A function doesn't have to take or return arguments. We prototype such a function using *void*.

- A function can take any number of arguments mixed in any way.

- A function can return at most one argument.

- When we return from a function, the values of the argument **HAVE NOT CHANGED**.

- We can declare variables within a function just like we can within `loop()` - these variables will be deleted when we return from the function



# Where do functions go?

---

```
float valA=12.5; float valB=16.1;
```

```
void setup() { Serial.begin(9600); }
```

```
float return_maximum(float vA, float vB)
{
    if (vA > vB) { return (vA); }
    else { return (vB); }
}
```

```
void print_hello_message()
{
    Serial.println("hello world");
}
```

```
void loop()
{
    //calling the function
    float res = return_maximum(valA, valB);
    Serial.println(res);
    print_hello_message();
    delay(1000);
}
```

- `loop()` is a function just like any other.
- Functions must be entirely separate from each other.
- Functions must be declared on before they are called. Its not a bad idea to declare them all the way on top.
- You can also declare functions on external files.

# Variable scope

---

```
float valA=12.5; float valB=16.1;

void setup() { Serial.begin(9600); }

void print_variable_scope()
{
    int intVariable=10;
    Serial.println(intVariable);

    Serial.println(valA); //valA is global

    //ERROR! Variable res is out of Scope
    Serial.println(res);
}

void loop()
{
    print_variable_scope();
    float res = 12.5;
    Serial.println(res);
    //ERROR! intVariable is out of Scope
    Serial.println(intVariable);
}
```

- The scope of a variable is where it can be used in a program
- Normally variables are local in scope - this means they can only be used in the function where they are declared (main is a function)
- We can also declare global variables.
- If we declare a variable outside a function it can be used in any function beneath where it is declared
- Global variables are A BAD THING... Try to minimize their use.

# Loops

---

```
void setup() { Serial.begin(9600); }
```

```
void loop()
```

```
{
```

```
    int i;
```

```
    //for loop
```

```
    for (i=0; i < 10 ; i++)
```

```
    {
```

```
        Serial.println(i);
```

```
    }
```

```
    //while loop
```

```
    while (i<10)
```

```
    {
```

```
        Serial.println(i);
```

```
        i++;
```

```
    }
```

```
    delay(1000);
```

```
}
```

- Two of the most used workhorses in C are the for and while loops.

- Which is clearer in this case? A for or while loop?

## For Loop

```
for (initialiser ; condition ; increment) {  
    code;  
}
```

## While Loop

```
initialiser;  
while (condition) {  
    code;  
    increment;  
}
```

# Why are global variables bad?

---

```
int i;

void print_stars()
{
    for (i=0 ; i<5 ; i++)
    {
        Serial.print("*");
    }
    Serial.print("\n");
}

void setup()
{
    Serial.begin(9600);
    for (i=0; i < 5 ; i++)
    { print_stars(); }
}

void loop()
{
}
```

- In this code we want to print a matrix of 5x5, filled with stars.
- This code will only print one row of five stars!

# Avoiding a global variable

---

```
void print_stars()
{
    int i;
    for (i=0 ; i<5 ; i++)
    {
        Serial.print("*");
    }
    Serial.print("\n");
}

void setup()
{
    int i;
    Serial.begin(9600);
    for (i=0; i < 5 ; i++)
    { print_stars(); }
}

void loop()
{
}
```

- In this code we want to print a matrix of 5x5, filled with stars.
- This code will only print what we want...



# #define and const

---

```
const int pin = 13;
#define PI 3.14
#define GRAV_CONST 9.807
#define HELLO_WORLD "Hello World!\n"

void setup() {
    Serial.begin(9600);
}

void loop()
{
    float res = 10 * PI;
    Serial.println(res);
    Serial.print(HELLO_WORLD);
    //ERROR! Can't modify const variable;
    pin = pin+2;
    Serial.print(pin);
}
```

- The #define preprocessor command replaces one thing with another before compilation.
- Now, anywhere in the program we use PI or GRAV\_CONST it will be replaced with the replacement string BEFORE the real compiler starts (that's why we call it pre-processing)
- Const variable types will prevent its modification!

# Creating an array

---

- An array is a collection of variables that are accessed with an index number.
- All of the methods below are valid ways to create (declare) an array.
  - `int myInts[6];`
  - `int myPins[] = {2, 4, 8, 3, 6};`
  - `int mySensVals[6] = {2, 4, -8, 3, 2};`
  - `char message[6] = "hello";`
- You can both initialize and size your array, as in `mySensVals`. Note that when declaring an array of type `char`, one more element than your initialization is required, to hold the required null character (`"\0"`).

# Accessing an array

---

- Arrays are zero indexed, that is, the first element of the array is at index 0,
- It also means that in an array with ten elements, index nine is the last element. Hence, if `int myArray[10]={9,3,2,4,3,2,7,8,9,11};`
  - `myArray[9]` contains 11
  - `myArray[10]` is invalid and contains random stuff
- To assign a value to an array:
  - `mySensVals[0] = 10;`
- To retrieve a value from an array:
  - `x = mySensVals[4];`

# Array + loop example

---

```
#define ARRAYPOS 10
void setup()
{
  Serial.begin(9600);
  int myArray[ARRAYPOS];

  for (int i=0; i < ARRAYPOS ; i++)
  {
    myArray[i]=random(0,100);
    Serial.print(myArray[i]);Serial.print(" ");
  }
  Serial.print("\n");

  float sum=0;
  for (int i=0; i < ARRAYPOS ; i++)
  { sum=sum + myArray[i]; }
  sum = sum / ARRAYPOS;
  Serial.print("Average = ");
  Serial.println(sum);
}
void loop() {}
```

- In this example, I create a `#define` with the value 10
- Then I create an array with 10 memory elements of type *int*
- I proceed to fill it with 10 random values.
- Finally, I calculate the variable of those 10 values.

# 2D Array

---

- 2D arrays are very similar to 1D arrays.
- I can initialize the array with something like:

```
float array1 [10][5];
```

- That will create a 2D array with 10 rows and 5 columns of a float datatype.
- I can also initialize the contents of the 2D array as follows:

```
float array2 [3][2] = { {10,20},{3,4},{40,0}};
```



# Other useful Arduino (wiring) functions

# Arduino map() function

---

- **map(value, fromLow, fromHigh, toLow, toHigh)**

- Extremely useful function: Re-maps a number from one range to another.
- That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.
- Example:
  - Map an analog value (0 to 1023) to 8 bits (0 to 255)

```
void setup() {}

void loop()
{
    int val = analogRead(0);
    val = map(val, 0, 1023, 0, 255);
    analogWrite(9, val);
}
```

# random() function

---

- The random function generates pseudo-random numbers.
- Syntax:
  - random(max)
  - random(min, max)

## Code Example:

```
void loop() {  
    int randomNumber;  
    // print a number from 0 to 299  
    randomNumber = random(300);  
    Serial.println(randomNumber);  
  
    // print a number from 10 to 19  
    randomNumber = random(10, 20);  
    Serial.println(randomNumber);  
  
    delay(50);  
}
```

# Controlling Arduino From Other Programs

---

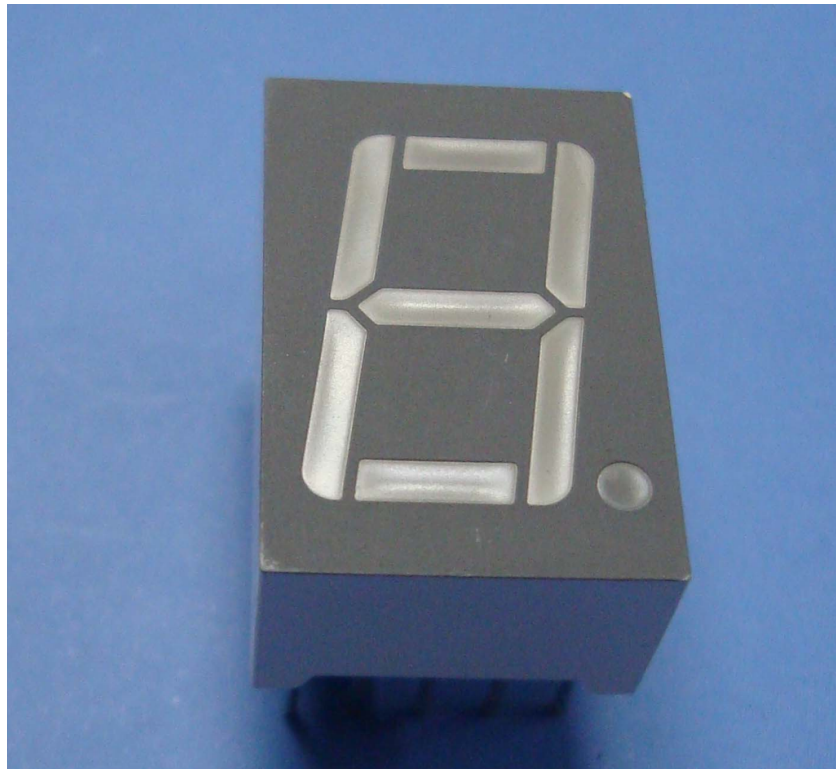
- Any program on the computer, not just the Arduino software, can control the Arduino board
- On Unixes like Mac OS X & Linux, even the command-line can do it:

```
demo% export PORT=/dev/tty.usbserial-A3000Xv0
demo% stty -f $PORT 9600 raw -parenb -parodd cs8 -hupcl -cstopb clocal
demo% printf "1" > $PORT      # rotate servo left
demo% printf "5" > $PORT      # go to middle
demo% printf "9" > $PORT      # rotate servo right
```

# Interfacing with Seven Segment Displays



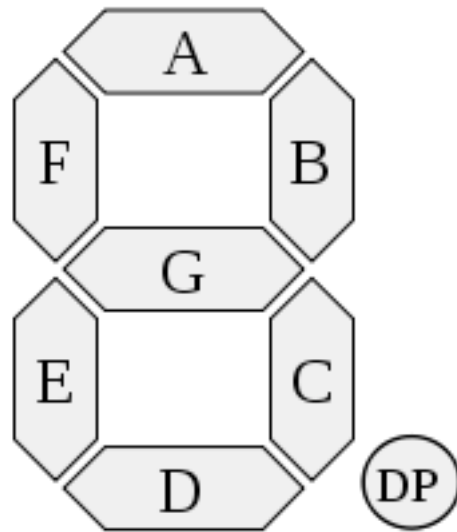
# 7-Segment LED Displays



- One of the old, but simpler methods of displaying numeric information is to use one or more 7-Segment numeric displays connected to your board.
- A seven segment display is composed of LED seven elements.
- Individually on or off, they can be combined to produce simplified representations of the arabic numerals.
- In most applications, the seven segments are of nearly uniform shape and size.
- The numbers are usually arranged in an slanted arrangement, which aids readability.

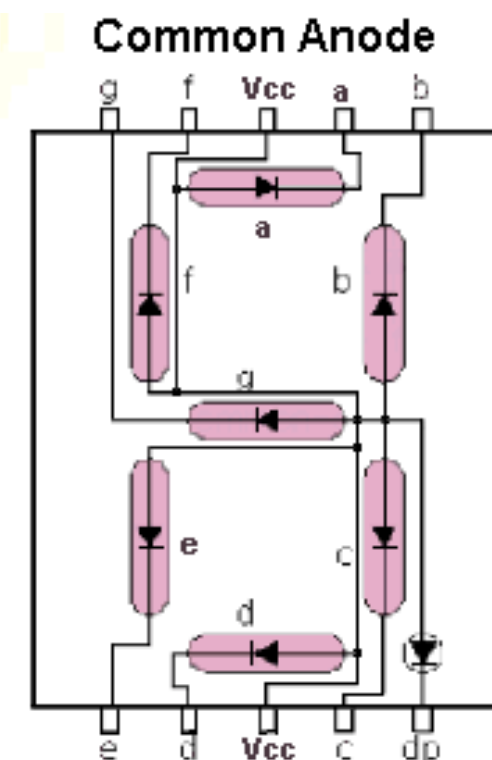
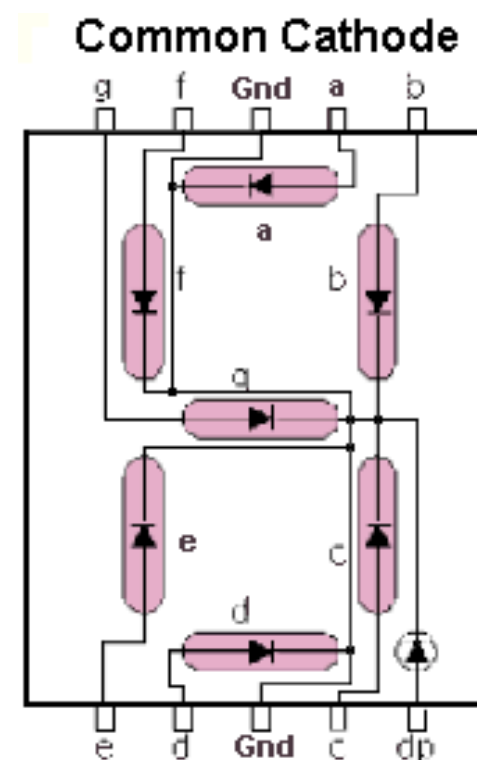


# 7-Segment LED Displays



- The segments of a 7-segment display are referred to by the letters A to G.
- Some displays have the optional decimal point, used for the display of non-integer numbers.
- Circuit connections vary from different manufacturers.

Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1



```
int pinId[7]={4,7,9,10,5,6,8};
int readKey=-1;
```

```
void setup()
{
    for (int i=0; i<7; i++)
    { pinMode(pinId[i],OUTPUT); }
    Serial.begin(9600);

    for (int i=0; i<7; i++)
    digitalWrite(pinId[i],HIGH);
}
```

```
void loop()
{
    while (Serial.available()>0)
    readKey = Serial.read();
    if (readKey=='0')
    {
        char data={B11000000};
        for (int i=0; i<7; i++)
        digitalWrite(pinId[i],bitRead(data,i));
    }

    if (readKey=='1')
    {
        char data={B11111001};
        for (int i=0; i<7; i++)
        digitalWrite(pinId[i],bitRead(data,i));
    }
}
```

# Seven-segment display interface code

*bitRead()* - Reads a bit of a number

Syntax - bitRead(x, n)

Parameters:

x: the number from which to read

n: which bit to read, starting at 0 for the rightmost bit

*bitWrite()* - Writes a bit of a number

Syntax - bitWrite(x, n,b)

Parameters:

x: the number from which to read

n: which bit to read, starting at 0 for the rightmost bit

b: boolean value