

# Electronics and Programming for Responsive Environments by Christian Nold, 2011

## Version 4

[www.electronics.softhook.com](http://www.electronics.softhook.com)

### Timetable:

10:00 Responsive Environments & Basics electronics concepts

10:20 Demonstrate Electronics Kits & What is Arduino?

10:40 TEA BREAK

11:00 Setup your Arduino & Intro to Programming & Controlling LEDs

12:30 LUNCH

13:30 Basic Inputs digital and analogue Switch & LDR & Complex Outputs using transistors

15:30 Tea Break

15:45 Servo Motors

16:15 What can Processing do?

### What are Responsive Environments?

Its both fad with many names such as Ambient Intelligence, Pervasive Computing, Internet of Things but also an enduring fantasy of the environment being 'readable' or perhaps even alive. The ancient Greeks saw the weather as being the work of the gods themselves. By observing the environment they could understand the will of the Gods. Today we try to 'read' the city using RFID tags and sensors to understand the behaviour of people or the city itself.

Interestingly the ancient Egyptians simulated these responsive phenomena in their religious buildings. They had water bowls that were continuously filling and draining. They had Fire jets that roared up automatically when you entered a room. These elements where there to give you a sense of the numinous i.e. The presence of a divinity. Even today when people use these 'magical' elements in buildings they are designed to remind us of the higher intelligence (or not) that was used to design these spaces.

In a sense these Responsive Environments are a simulation of a lost nature. We love the idea that when we push something that 'thing' might be alive enough and intelligent enough to choose to push back. The particular 'aliveness' of a space is a powerful definition of the meaning of a space.

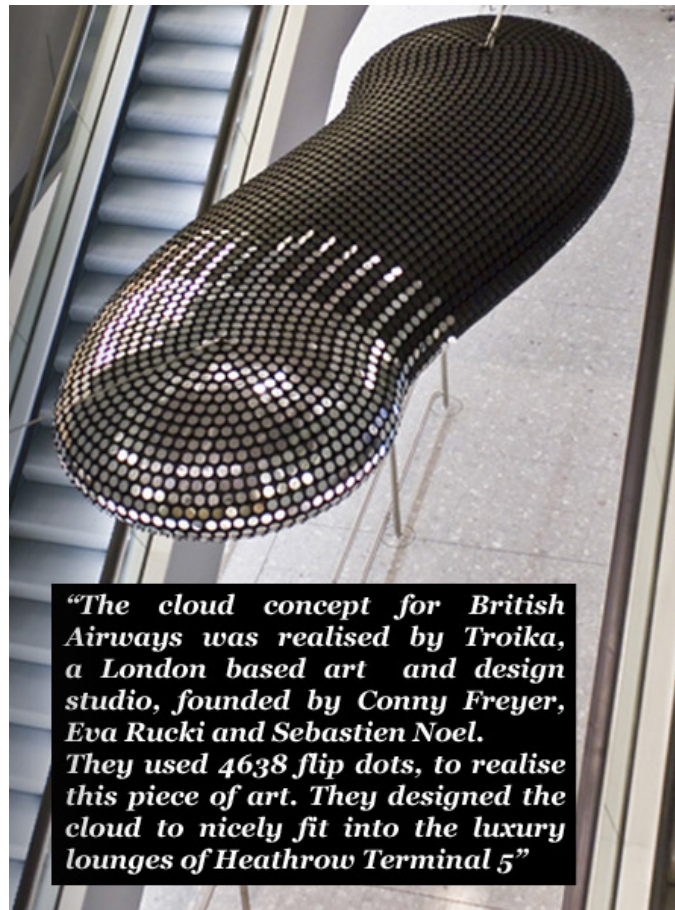
The type and quality of the interaction that we design makes a statement about our relationship with other people and the world. If we design simplistic and crude interactions we let people know that we think they are stupid. If we design mysterious interactions that people don't understand then we create almost religious spaces of awe. If we design playful or serious or sophisticated interactions they are all EXPRESSIONS of our thoughts and ideals as designers as well as communicate an atmosphere to how people should behave with each other and react to the space.

### How can we make good interactions?

I would argue that its about working on two levels simultaneously and bouncing backwards and forwards.

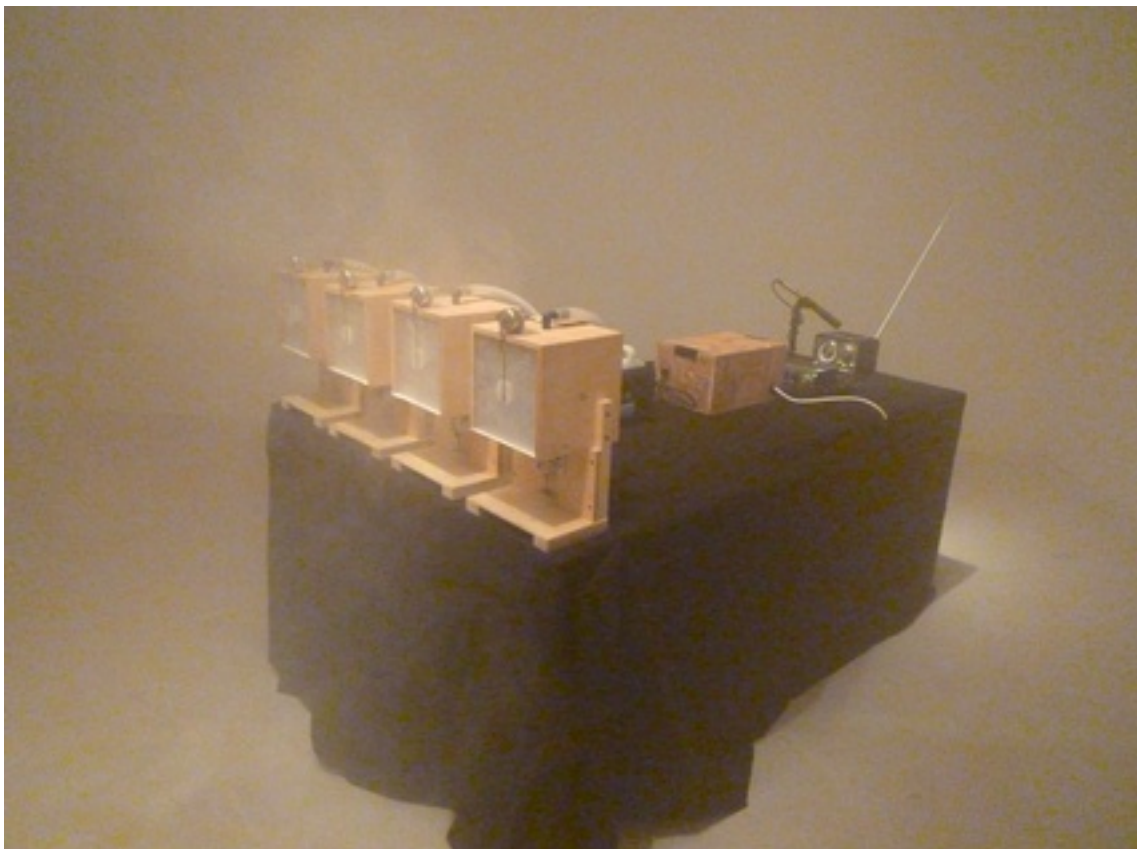
1 Conceptual idea level

2 Playful experimental level that explores the material of electronics/programming

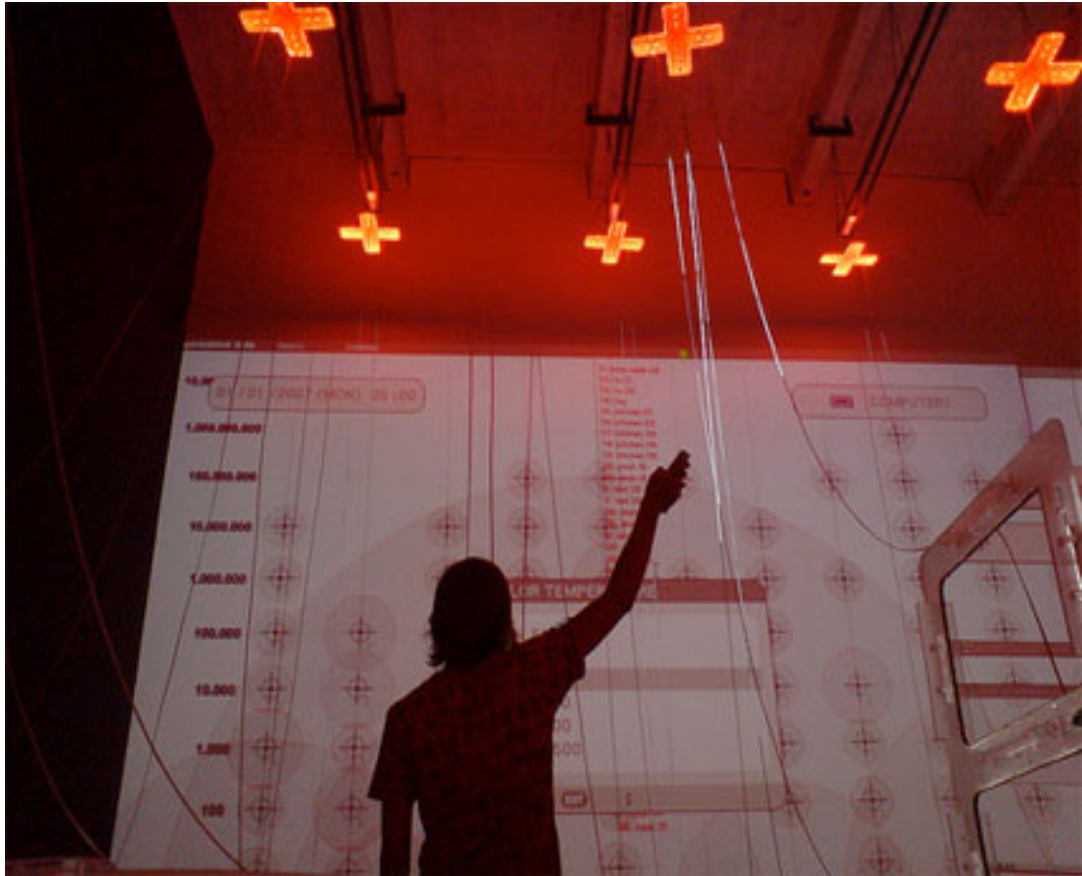


*"The cloud concept for British Airways was realised by Troika, a London based art and design studio, founded by Conny Freyer, Eva Rucki and Sebastien Noel. They used 4638 flip dots, to realise this piece of art. They designed the cloud to nicely fit into the luxury lounges of Heathrow Terminal 5"*









## Analogue Electronics

These days very few people have the skills to analogue circuits from scratch. The main advantage of these home built analogue electronics is low cost and being able to get them mass manufactured. Unfortunately there are many disadvantages. If you build a circuit from scratch they are often unpredictable, can be complex and only ever have one function.

There are some useful analogue electronics that I recommend that are available in solder-it-yourself or ready built kits. Their advantage is that they very easy to use and always work. The disadvantage is that they are very limited in flexibility and fairly expensive. So if you find one that does exactly what you need than that is fantastic!

Some examples:

### Light / Dark Activated Switch (Schmitt Trigger) from Quasar

This Light Kit uses a Light dependent resistor to turn a relay on and off. This relay can be used to turn other things on and off such as a motor, solenoid etc. There is another switch that allows you to select if it reacts to light or to dark. A little variable resistor allows you to change the sensitivity of the circuit. The relay has only two states it can be in ON or OFF so it can't be used to vary the speed of a motor for example depending on the light level.

### 1047 - Sound Activated Latching Relay Switch from Quasar

This particular sound circuit reacts to noise and switches a relay on. It uses uses latching, meaning that it stays in an on state until there is another sound.

### 3030 - PIR Motion Detector (Adjustable PIR & Delay) from Quasar

This sophisticated kit can detect people's presence and turn something on for a preset amount of time. The sensitivity and other elements can all be varied.

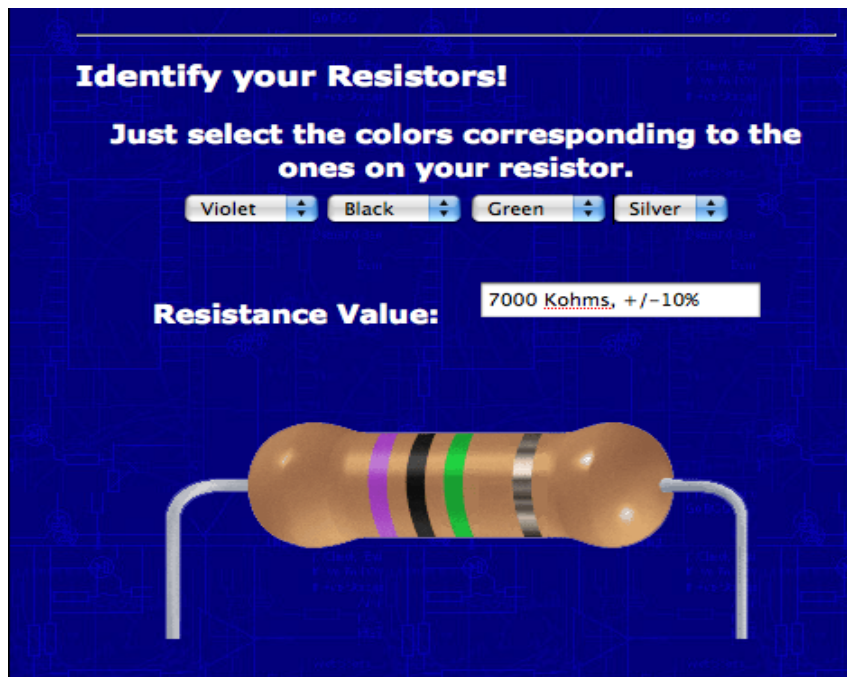
Spend a bit of time looking at the Quasar [website to get an idea of what you can do easily](#)

## Identifying Resistors

We will be using 3 different Resistors for this workshop:

- 150 Ohms - for the LEDs
- 1 K Ohm - for protecting the transistor
- 10 K Ohm - for the Pull-Down resistor

To identify them you can use the colour rings on the side using this website for example:  
<http://avishowtech.com/mbhp/res.html>



BUT the best and most accurate way though is using the Multimeter on the Ohm setting

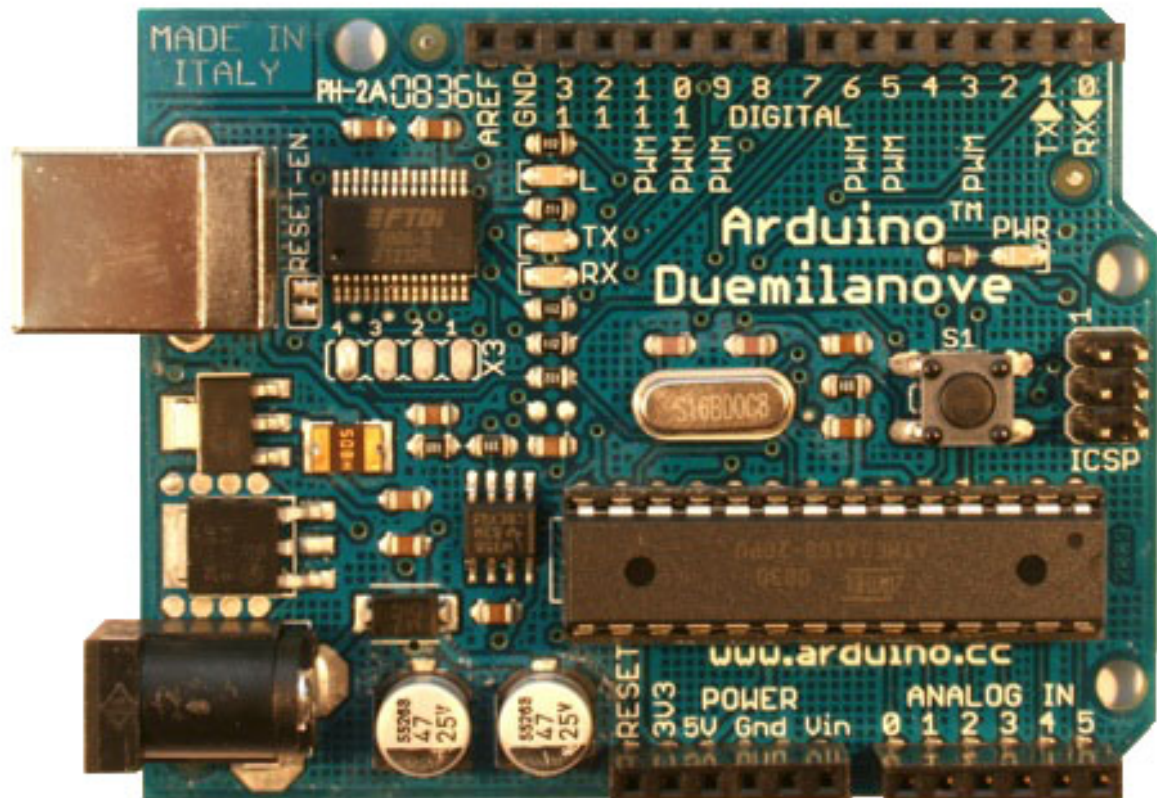
## Digital Electronics

### What is Arduino?

Its a Micro controller. Basically a little computer that you program on your PC and then can be used stand along or connected to a laptop. It is basically a simple brain that can read signals from a variety of sensors, perform some calculations and then control a number of outputs. Arduino has 14 digital in and out pins as well as 6 analogue-in pins. Each digital pin can output about 40ma which is enough to drive a small LED but not much more.

- The Arduino Project was developed out of an educational environment and is therefore great for newcomers to get things working quickly.
- It is a Multi Platform environment – it can run on windows, mac, linux
- It is Based on the Processing programming IDE and makes it easy to move between them
- It is programmed via a USB cable not a serial port
- It is Open Source hardware + software – if you wish you can download the circuit diagram, buy all the components, and make your own, without paying anything to the makers of Arduino.
- The hardware is relatively cheap. The USB board cost about £25 and replacing a burnt out chip on the board is easy and costs no more than £4. So you can afford to make mistakes

Most importantly, there is an active community of users using it, so there plenty of people who can help.



## Basic Introduction to Programming Principles

Some of you may never have programmed before. If you have you may skip this bit....

Programming the Arduino Board depend on two basic types of programming structure: loops and conditional statements. A loop is necessary to allow the processor to continually read the states of its inputs as well as to update the states of its outputs. Conditional statements will be used to specify the logic that you are trying to embody in your program. A simple programme can always be mocked up as sentence.

"When it is very dark I want an LED to turn on and a motors to start turning slowly"

This can then be turned into pseudo code:

If light level is less than a certain amount then

- Turn Light on
  - Turn motor on slow
  - Wait a bit
- Loop again

Then you can translate this into the actual programming structure like this

```
void setup()
{
  .... Here we setup which pins we want as inputs and outputs as well as serial settings.
}
```

and:

```
void loop()
```

```
{
.....Here the goes the actual program that checks the light level and turns the
}
```

The second function, void loop() will then be called indefinitely until you turn the Arduino board off. So this is where we want all our conditional logic to be stored. It is the real program and where we can control the “flow” of the program.

## Variables

A variable is a container that you can use to store data. You need to “declare” what sort of data you will store there. There are a few basic data types:

int: an integer is a whole number i.e. 1,2,3,5 etc.

float: A float is a variable with decimal places i.e. 4.767 or 2.543

## Commands

We only need 4 basic commands for this workshop:

digitalWrite which can turn an LED on and off (HIGH or LOW)

digitalRead which reads the state of a pin (1 or 0)

analogWrite which can dim an LED (0-255)

analogRead which reads the state of a pin (0-1023)

## Conditional Statements

If ... Then do something

```
if(expression)
{
    statement;
}
```

Where the expression could be one of the following:

a == b	a equals b
a != b	a is not equal to b
a > b	a is greater than b
a < b	a is less than b
a <= b	a is less than or equals to b
a >= b	a is greater than or equals to b

And the statements can be anything you like (including more conditional statements).

## For loops

```
for (i=1; i <= 8; i++)
{
    statement;
}
```

## Delays

Delay function is useful for embedded programming to slow down the rate at which the processor updates. This is useful in debugging and general flow control.



## Blink LED (Hello World)

### use this to test if you have a hardware problem

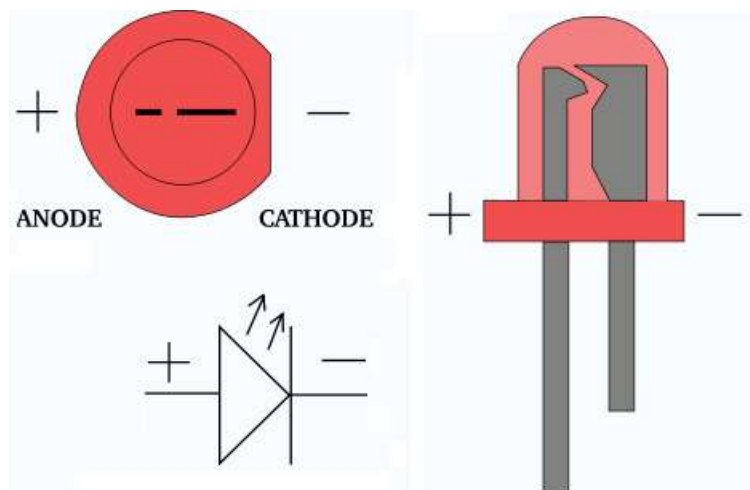
Please compile this code and send it to the Arduino. If it works then you should see the tiny LED next to pin 13 flash.

```
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  
}
```

Here we are using the command digitalWrite(whichPin, value) which requires a pin number as well as either HIGH or LOW. By using digitalWrite HIGH and then LOW we can make the LED flash. Please try changing the value of the delay. The number there is in milliseconds.

## Basic Outputs

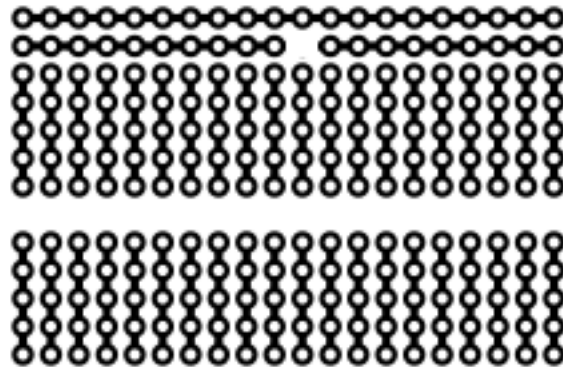
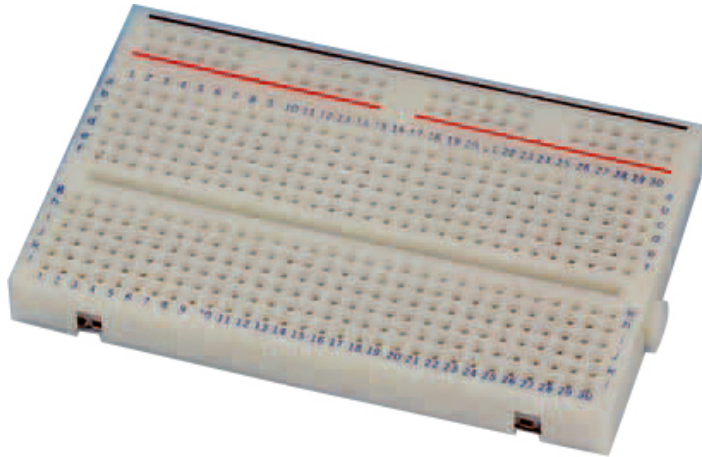
### LEDs

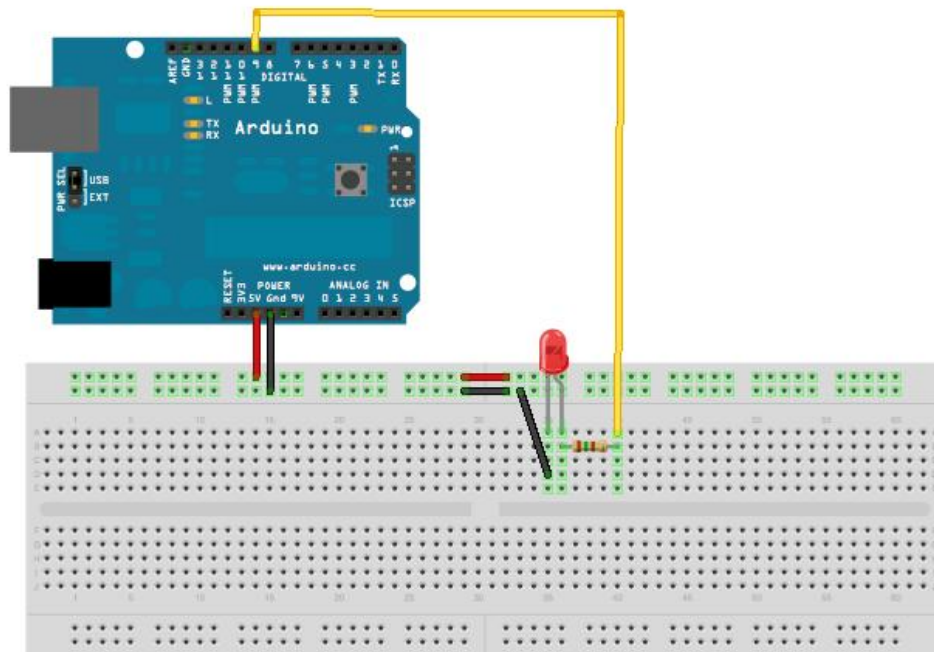


LEDs have polarity! Please note the length of the legs

## Breadboard

Please make sure you understand how breadboards connect. Take a look at the Ground and Power rails at the top.





The LED we are using for the workshop requires an **150 Ohm resistor**.

Here we will use the Blink example again but we are using an external LED attached to Pin 9 so in the code change the ledPin variable to 9.

```

/*
Blink an external LED on pin 9
The LEDs we are using for the workshop require a 150 Ohm resistor.
*/

int ledPin = 9; // LED connected to digital pin 9
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
}

```

Next we can try dimming the LED. We just have to change the code while leaving the electronics the same. The code we will use here uses Pulse Width Modulation (PWM) to simulate an analogue output with the analogWrite command.

```
int brightness = 0;  // how bright the LED is
int fadeAmount = 5;  // how many points to fade the LED by

void setup() {
  // declare pin 9 to be an output:
  pinMode(9, OUTPUT);
}

void loop() {
  // set the brightness of pin 9:
  analogWrite(9, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Please note the use of a limiting **150 Ohm resistor** here. The idea is to limit the current to an acceptable level for the LED. On the next page we have a screen shot from Maplin showing the detail of the LEDs. What is important here is the Forward Voltage = 1.8V and the Forward Current = 30ma

$(\text{Supply Voltage} - \text{Voltage Drop}) / \text{Ampere} = \text{Resistance}$

Since the Arduino is supplying 5 volts the formula is  $5 - 1.8 / 0.03 = 106.67 \text{ Ohms}$

The closest resistor we have handy is 150 Ohm which is good enough but not maximum brightness.



Super-Bright LEDs : 3mm Standard LEDs : Maplin

http://www.maplin.co.uk/Module.aspx?ModuleNo=2064

LED calculator

Product Features

Super-bright red LEDs in four sizes. The LEDs fit the appropriate panel mounting clips. The cathode is denoted by the flat on the body and by the short lead. The LEDs have a diffused lens giving a wide viewing angle. Specifications:

Technical Specification

Back to Top

Forward voltage at IF = 1.8V  
20mA  
Forward current (max): 30mA  
Reverse voltage (max): 5V  
Power dissipation (max): 100mW  
Peak wavelength: 660nm  
Case size (dia.): 3mm 5mm 8mm 10mm  
Light output typical at IF = 20mA 80mcd 150mcd 200mcd 200mcd  
Viewing angle 100 120 140 140

Download Centre - Technical, Firmware and Information  
UK18.pdf  
UK19.pdf

Back to Top

Frequently Asked Questions

Submit A Question

Check Your Local Store's Availability....

3mm Red 58 Diff LED £0.39 UK18U Postcode or Town: Go

LED calculator for single LEDs

http://led.linear1.org/1led.wiz

LED center

LED basics LED lighting LED science Practical LEDs LED products

LED calculator

This is the new version of the single LED series resistance calculator, good for when you have a single LED and need to know "what resistor should I use with my LED?" This calculator determines that for you.

The LED series/parallel array wizard is available for those of you who need to do calculations involving more than one LED. The wizard will help you pick the resistors make the connections for any number of LEDs.

LED calculator: current limiting resistor value

5 Source voltage ☒  
1.8 diode forward voltage ☒  
30 diode forward current (mA) ☒

Find R

The wizard recommends a 1/4W or greater 120 ohm resistor. The color code for 120 ohms is brown red brown.

5V 120 ohms, 1/4W 1.8V @ 30 mA

led.linear1.org

Link to this solution: <http://led.linear1.org/1led.wiz?VS=5;VF=1.8;ID=30>

\* This calculator rounds the resistance up to the next standard resistor value. You should actually be able to buy a 5% resistor with the value returned by the calculator.

search the LED center:

the linear1 network  
where all my interests come together

discussion forum  
ask your questions in the forums

linear1 case mode  
many illustrated case mod walkthroughs

LED series parallel array wizard  
a calculator to help design large arrays of LEDs

candela to lumen conversion wizard  
compare the light output of LEDs with different beam angles

privacy policy  
review this site's privacy policy

Latest articles:

LED Mag-Lite now available  
posted September 20, 2006

Hello Make: readers  
posted January 3, 2006

LED Candles from Hampton Bay  
posted December 3, 2005

12VDC LEDs from Best Hong Kong  
posted November 29, 2005

LED Industry in 'dire need' of uniform standards  
posted November 11, 2005

The LED dance floor craze  
posted November 9, 2005

One-cell LED flasher circuit  
posted November 9, 2005

## Wiring up LEDs

Wiring up LEDs in real world situations is actually more difficult than it seems. Often you want to light up a whole series of LEDs at the same time. There are two ways of wiring them, in series and in parallel. In general it is better to wire LEDs in parallel so that each one is its own circuit and has its own LED. In the real world it is often good though to go for a combination of series and parallel. Have a look at the Array wizard below which shows you how to wire up 10 LEDs for example. Its a super useful website

Remember that if you want to control them from the Arduino though you will need a Transistor (more of them later) in the circuit.

Firefox File Edit View History Delicious Bookmarks Tools Window Help

LED series parallel array wizard

http://led.linear1.org/led.wiz

Now: 9 °C Sat: 9 °C Sun: 7 °C

LED center

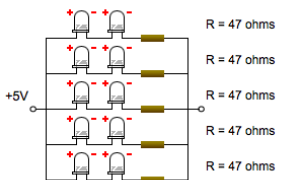
[LED basics](#) [LED lighting](#) [LED science](#) [Practical LEDs](#) [LED products](#)

### LED series/parallel array wizard

The LED series/parallel array wizard is a calculator that will help you design large arrays of LEDs. The [LED calculator](#) was great for single LEDs—but when you have several, the wizard will help you arrange them in a series or combined series/parallel configuration. The wizard determines the current limiting resistor value for each portion of the array and calculates power consumed. All you need to know are the specs of your LEDs and how many you'd like to use.

Source voltage  ☒ diode forward voltage  ☒  
diode forward current (mA)  ☒  
number of LEDs in your array   
View output as: ☐ ASCII ☐ schematic ☒ wiring diagram ☒  
☐ help with resistor color codes

Solution 0: 2 x 5 array uses 10 LEDs exactly



$R = 47 \text{ ohms}$   
 $R = 47 \text{ ohms}$   
 $R = 47 \text{ ohms}$   
 $R = 47 \text{ ohms}$   
 $R = 47 \text{ ohms}$

The wizard says: In solution 0:

- each 47 ohm resistor dissipates 42.3 mW
- the wizard thinks 1/4W resistors are fine for your application ☒
- together, all resistors dissipate 211.5 mW
- together, the diodes dissipate 540 mW
- total power dissipated by the array is 751.5 mW
- the array draws current of 150 mA from the source.

search the LED center:

[the linear1 network](#)  
where all my interests come together

[discussion forum](#)  
ask your questions in the forums

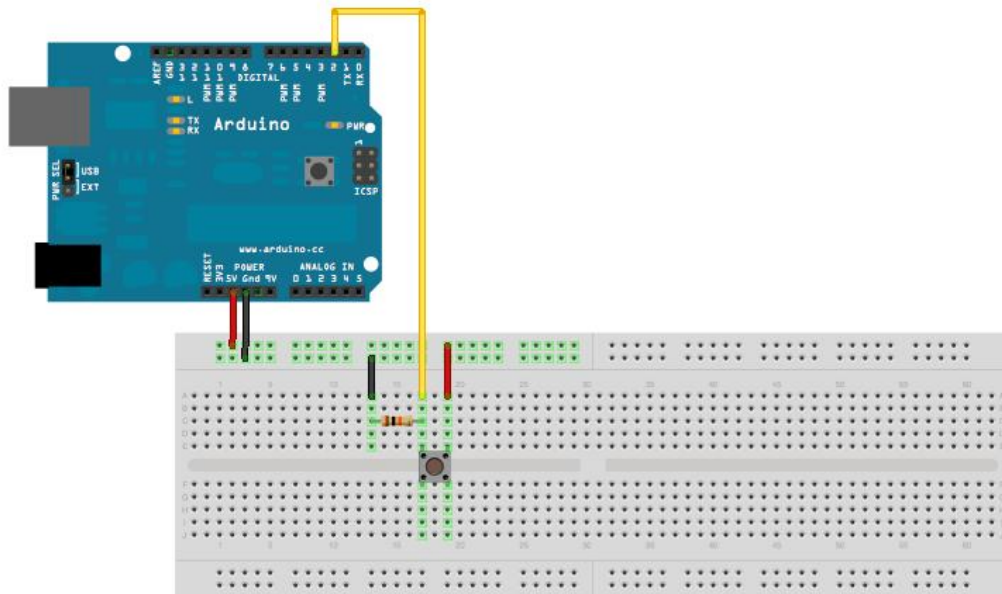
[linear1 case mode](#)  
many illustrated case mod walkthroughs

[LED series parallel array wizard](#)  
a calculator to help design large arrays of LEDs

[candela to lumen conversion wizard](#)  
compare the light output of LEDs with different beam angles

[privacy policy](#)  
review this site's privacy policy

## Basic INPUT: Push Buttons - anything that can make/break a circuit



Here we are reading a basic push button. Please note we are using a **10 KOhm** resistor to act as a Pull-Down Resistor connected to ground. When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

Imagine following the yellow sensor wire. If the button is not pressed then we go through the resistor to Ground (black wire) so the variable `buttonState = 0` . If you press the button, suddenly it is easier to go to Power (red wire) by avoiding the resistor so `buttonState = 1`.

```
int buttonPin = 2;    // the number of the pushbutton pin
int ledPin = 13;      // the number of the LED pin
int buttonState = 0;  // variable for reading the pushbutton status
```

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}
```

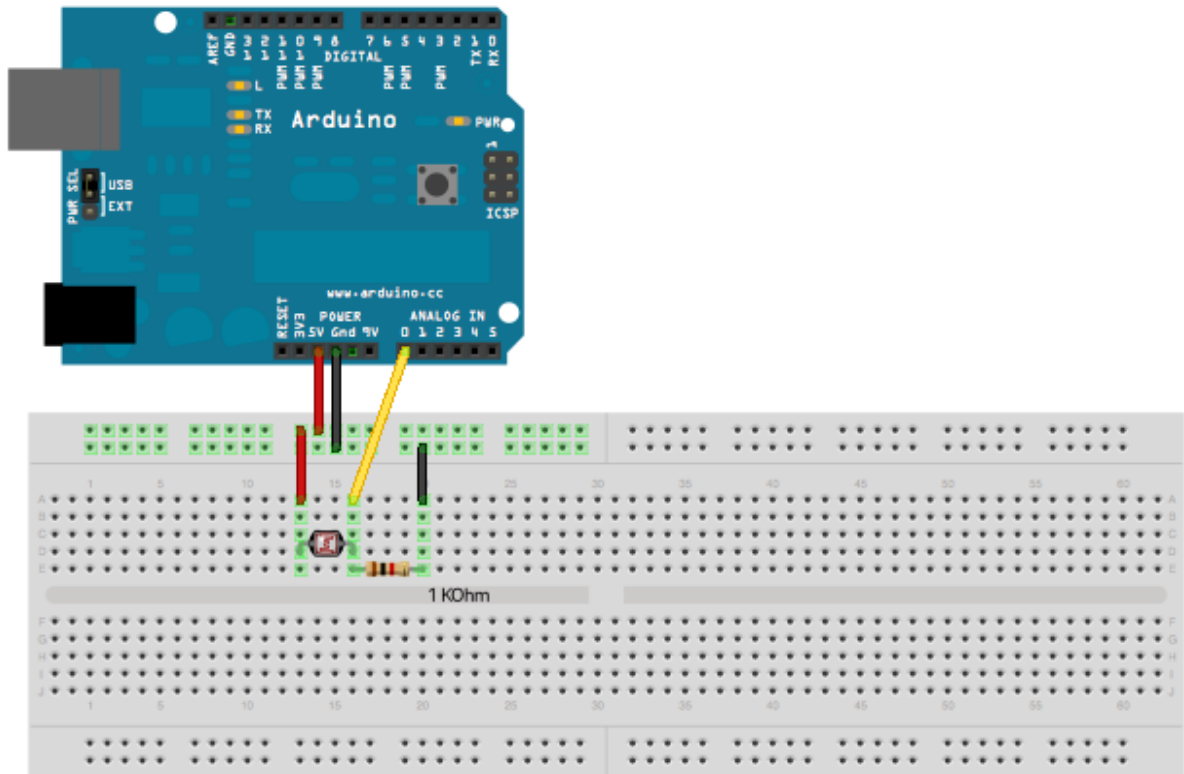
```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

Instead of a boring push-switch you can use the same code for a tripwire, tilt switch, a touch contact or anything where two bits of metal suddenly connect or disconnect.

## Light Dependent Resistor (LDR)

Here we are using a very useful principle called the voltage divider where the proportion between two resistors gives us a varying voltage which we can measure. Because one resistor is fixed while the other varies depending on light level we can measure the voltage using `analogRead` (note the American spelling). In this example we are reading out the value via the serial port at the speed of 9600.

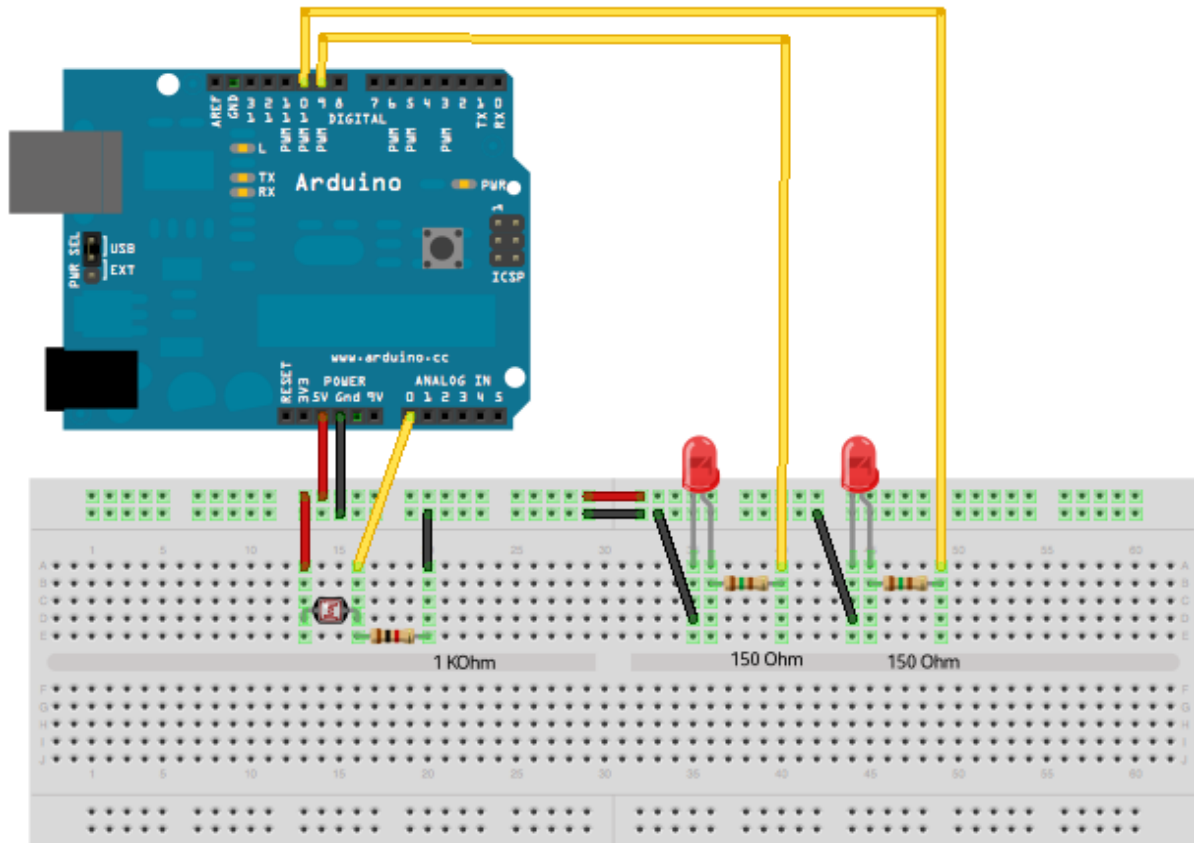


```
/*  
  AnalogReadSerial  
  Reads an analog input on pin 0, prints the result to the serial monitor  
  */  
  
void setup() {  
  Serial.begin(9600);  
}  
  
void loop() {  
  int sensorValue = analogRead(A0);  
  Serial.println(sensorValue, DEC);  
}
```



## Connect Inputs to Outputs

Try reading the LDR and making it control the brightness of one or even two LEDs. You will need both `analogRead` and `analogWrite`. The code below controls 1 LED try controlling both. For an extra challenge try making one LED glow brightly while other is dim and vice versa when you wave your hand over the LDR .



```
int ledPin = 9;    // the number of the first LED pin
int ledPin2 = 10;  // the number of the second LED pin

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

void loop() {

  // read the analog input into a variable:
  int analogValue = analogRead(0);

  // print the result:
  Serial.println(analogValue);

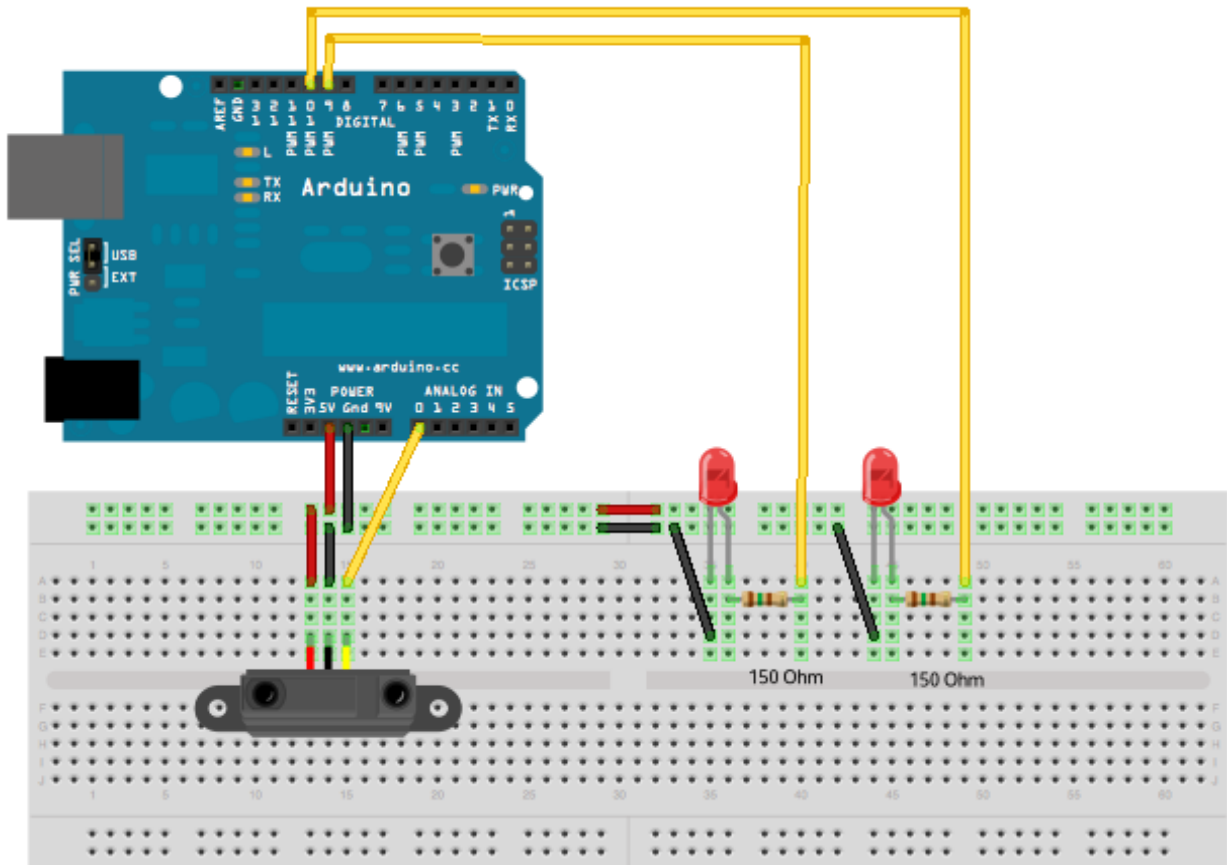
  //turn on LED with analogValue – to get it into the 0-255 range we are dividing the value by 4:
  analogWrite(ledPin, analogValue/4);
  analogWrite(ledPin2, analogValue/4);

  delay(10);
}
```

## Measuring Distance - Infrared

Here we are going to use a basic infrared sensor which has a range of about 10-80cm. The basic idea is this: a pulse of IR light is emitted by the emitter. This light travels out in the field of view and either hits an object or just keeps on going. In the case of no object, the light is never reflected and the reading shows no object. If the light reflects off an object, it returns to the detector. The device outputs the voltage corresponding to the detection distance so we can just read the voltage using `analogRead` as before

You might find that the `map` command very useful which allows you to scale numbers from one range of values to another range. See if you can convert the numbers you read into centimetres.



**The Arduino code for this set-up is exactly the same as the previous example.**

Advanced project using the sensor, involves mounting it on a servo motor to scan a space and visualise it in Processing

<http://www.uchobby.com/index.php/2009/03/08/visualizing-sensor-with-arduino-and-processing/>

If you want to read things at greater distances have a look at Ultrasound sensors that can work up to 7 meters distance. They are a little bit more difficult to use though since they work like radar. You need to time the delay between the pulse and the return signal in order to work out the distance. There is an example available though.

## Types of Motors

There are 3 different types.

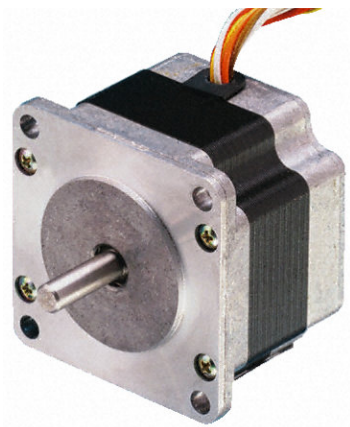
[DC Motors](#) which are easy to control from Arduino and cheap and can be geared to make them move slower. Gearing them gives them more torque as well. You need an H-bridge in order to change direction.



[Servo Motors](#) are used in model aeroplanes. They are easy to control very accurately in terms of positioning but pretty weak and can't be geared. They normally only move 180 degrees but you can find ones that are continuous one or even modify them.



[Stepper Motors](#) which are very accurate and powerful but expensive and hard to control ie. You need to by an [expensive controller board](#) and or do some fancy programming to control them. Notes [here](#)

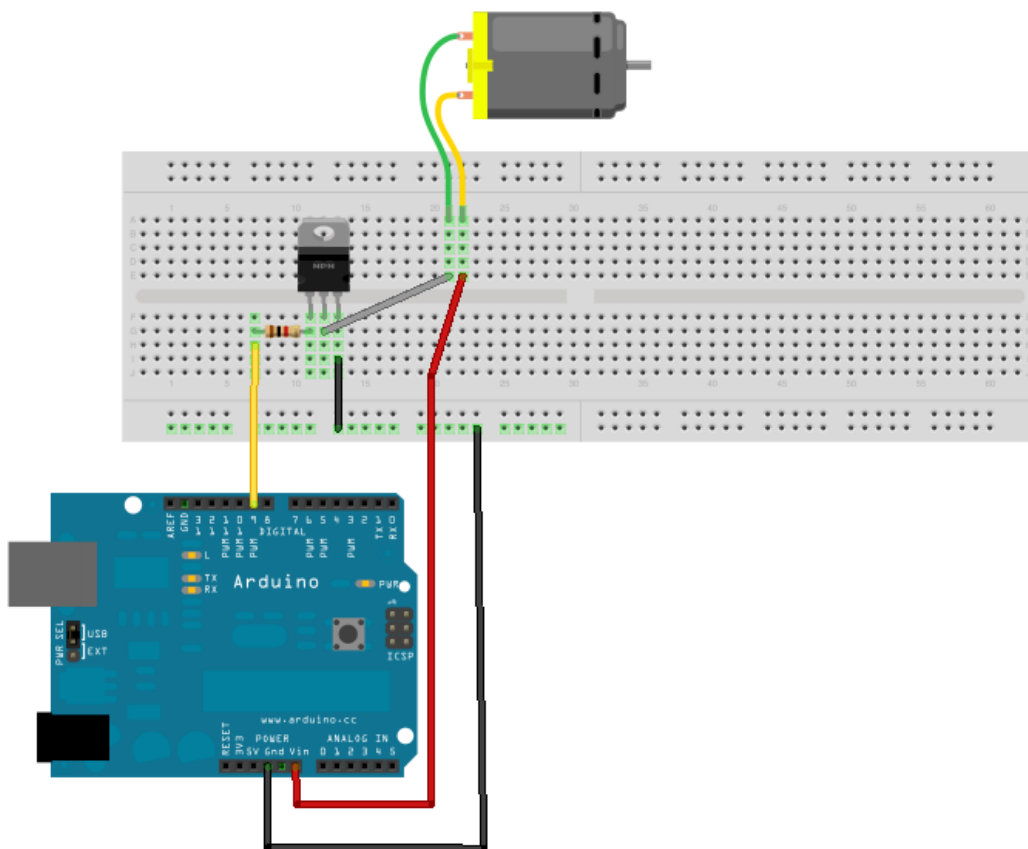


## Controlling Simple Motors and bigger loads with Transistors

A lot of what Bartlett students want to do involves motors so let's look at them in more detail. Each one of the Arduino digital pins can only output 40ma which is fairly little, certainly not enough to drive a motor. So to control bigger loads such as a motor, powerful LEDs, solenoids we use a transistor. Transistors are super useful in that they are essentially digital switches that allow us to have 2 separate circuits. The control circuit based on the Arduino runs at 5v and low current, while the one the other side can run at much higher voltages and currents. If you use a relay as we saw in the electronics kits which are similar to transistors you can even switch on mains power.

In the diagram below. The Arduino is powered using an external power supply to give it 9volts. It is this 9volts that will be powering the motor. We are using a TIP 120 transistor (NPN).

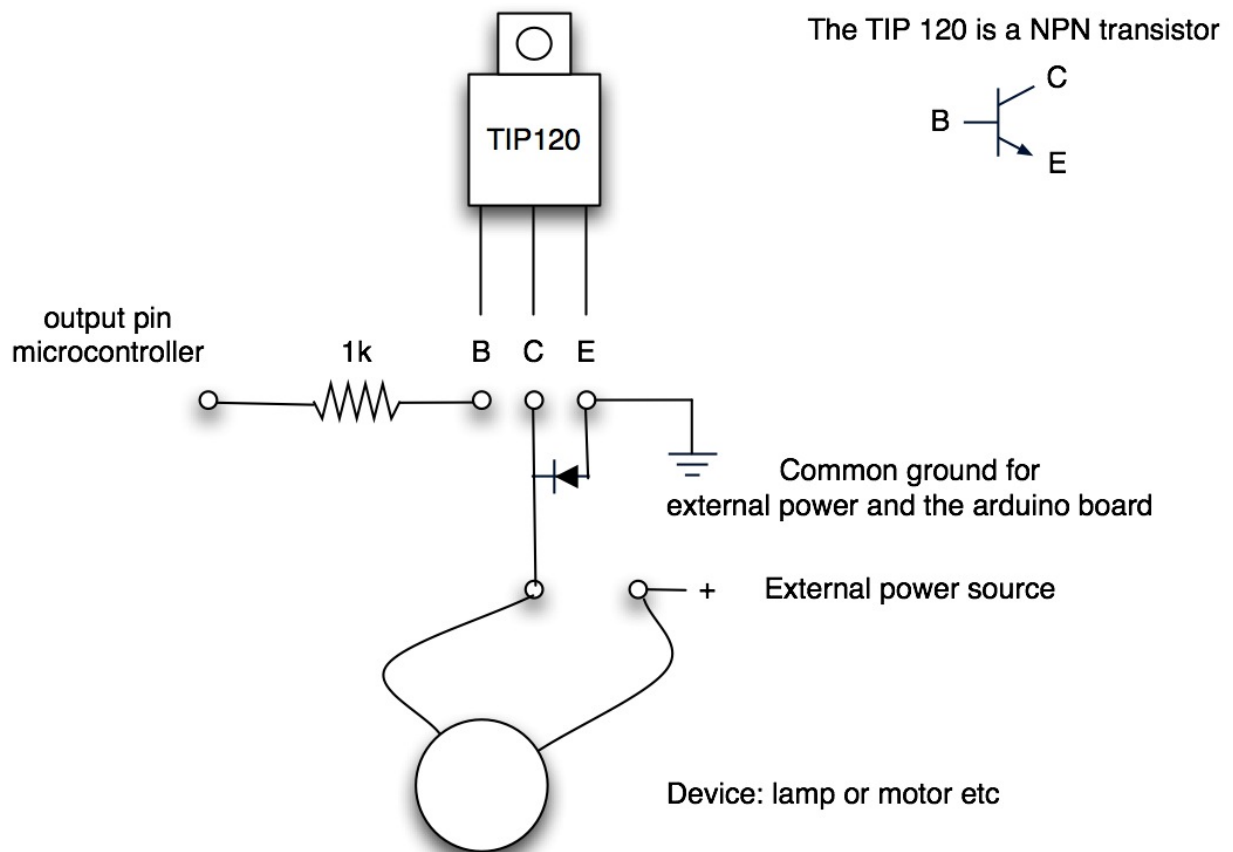
The resistor in the diagram is there to protect the transistor (3 legged component). There is a diode inside the transistor to protect against inductive loads where the sudden interruption of current flow often leads to a sharp rise in voltage across the device. This sharp rise in voltage can lead to failure of the controlling device. So the diode acts as a snubber and stops any of this back current.



The code to control the motor is basically an analogWrite which allows you to control the speed of the motor.

I also have some 1 watt LEDs that we can try and power in the same way. They require a very precise value of resistor. In this case 8.2 Ohms – If I can find some we can try them out. Just add the resistor and the LED instead of the motor in the diagram above.



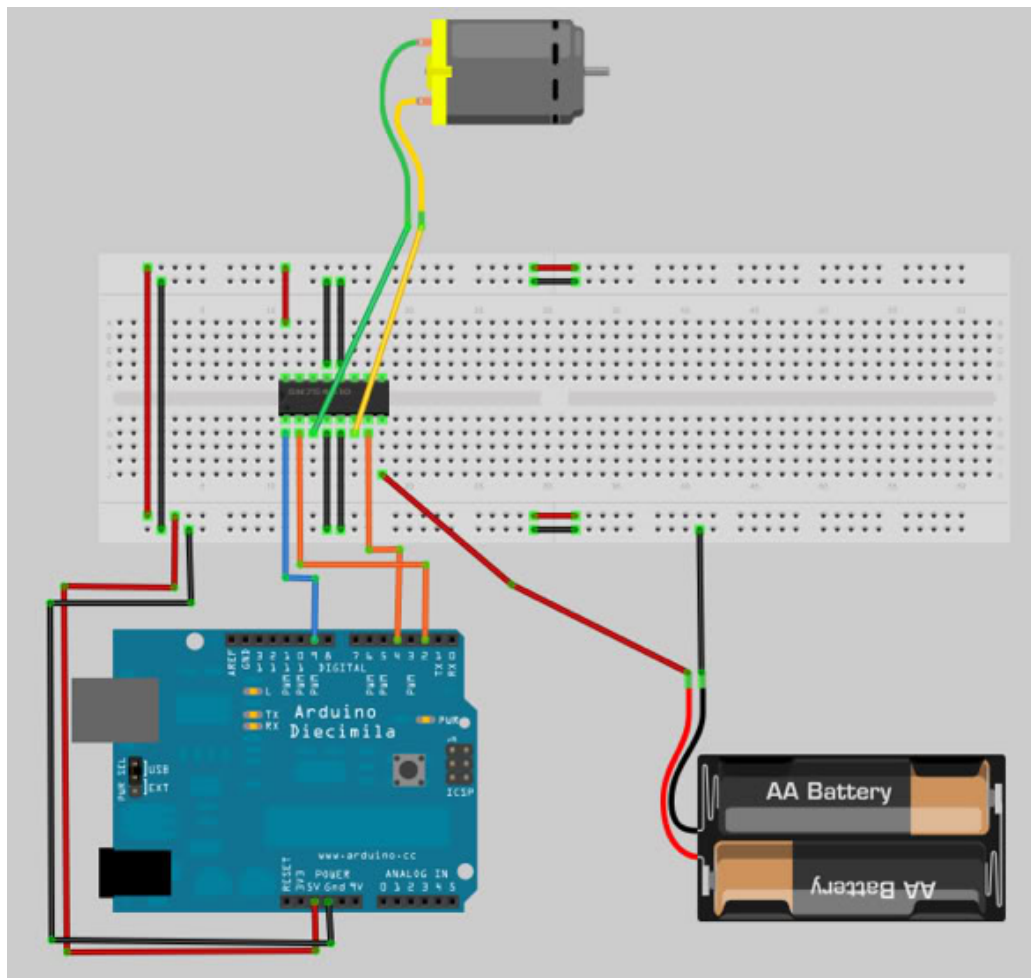


**B**ase to microcontroller  
**E**mitter to ground  
**C**ollector to device

Diode from E to C when connected to inductive load like a DC motor

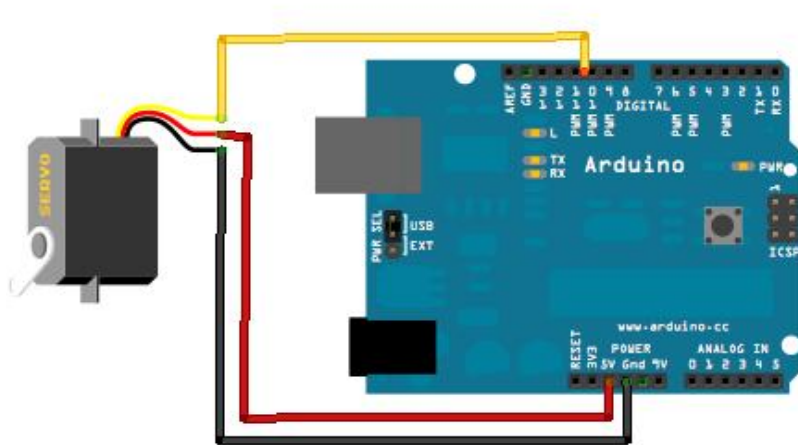
## H-bridge for changing direction of little motors

To change the direction of a DC motor is rather more difficult since you need an H bridge which is basically a chip. This is not something we will do now but I have included a diagram of how to wire it up to give you a sense of what is involved.



## Controlling Position and direction with Servo Motors

Servos are fantastic for precise and simple control from Arduino. You can use up to 8 simultaneously but read the notes on the Arduino website ...



This set-up is physically simple but the programming includes a few new concepts. At the beginning we create a virtual 'servo object', which we then need to attach to a particular pin. In this case pin 11. After this we can do `myservo.write` in the same way as the `analogWrite` earlier, except the values should be from 0-180. That value is in degrees.

```
#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()
{
  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
  }
}
```

If you find the servo behaving erratically you can try two things:

## **1 Change the initial servo.attach statement**

`servo.attach(pin, min, max)`

pin: the number of the pin that the servo is attached to

min (optional): the pulse width, in microseconds, corresponding to the minimum (0-degree) angle on the servo (defaults to 544)

max (optional): the pulse width, in microseconds, corresponding to the maximum (180-degree) angle on the servo (defaults to 2400)

## **2 Power the motor from a separate power supply**

You can use the external power for the Arduino. Many Servo work best at 6volts.

Here is a guide on how to modify Servos to run continuously  
[http://www.societyofrobots.com/actuators\\_modifyservo.shtml](http://www.societyofrobots.com/actuators_modifyservo.shtml)



## Advanced Programming - Timers (Blink without delay)

To do some more complicated things with Arduino we need to learn to abandon the Delay command and start using a timer method to do something at regular intervals. This programming method is key for being able to do multiple things with your Arduino since there is no multitasking like with your laptop.

The advantage of this method is that the Arduino is not stuck waiting for the delay to time out so that it can do multiple things almost at the same time such as blinking 2 LEDs at different time intervals. Why not try setting that up?

```
/* Blink without Delay
```

```
Turns on and off a light emitting diode(LED) connected to a digital
pin, without using the delay() function. This means that other code
can run at the same time without being interrupted by the LED code.
*/
```

```
int ledPin = 13;    // the number of the LED pin
int ledState = LOW;    // ledState used to set the LED
long previousMillis = 0;    // will store last time LED was updated
long interval = 1000;    // interval at which to blink (milliseconds)
```

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

```
void loop()
{
  // here is where you'd put code that needs to be running all the time.
```

```
  // check to see if it's time to blink the LED; that is, if the
  // difference between the current time and last time you blinked
  // the LED is bigger than the interval at which you want to
  // blink the LED.
  unsigned long currentMillis = millis();
```

```
  if(currentMillis - previousMillis > interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
```

```
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;
```

```
    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
```

```
  }
}
```

## Computer to Arduino | Arduino to Computer

### What can Processing do that Arduino can't ?

Processing is a piece of software that has access to all the power of your computer system. By connecting Arduino to Processing it can trigger and analyse **video and sound** and connect to the world via the **internet**.

Unfortunately the means for Arduino to talk to Processing is via serial which is relatively complicated. Luckily there is an interesting new Processing library called [Arduino](#) which allows you to control the Arduino board entirely from within Processing.

Why not try that library from Processing or try out the two serial communication examples that come included with Processing.

Try **simpleWrite** and **simpleRead**.

Start with the simpleWrite example which turns an LED on and off using your mouse. Try to use these two examples with the hardware setup that we used earlier with the DC motor for example and you will begin to see the power of this setup.