Electronics for Artists

Iain Sharp
lushprojects.com

Part 2: Arduino

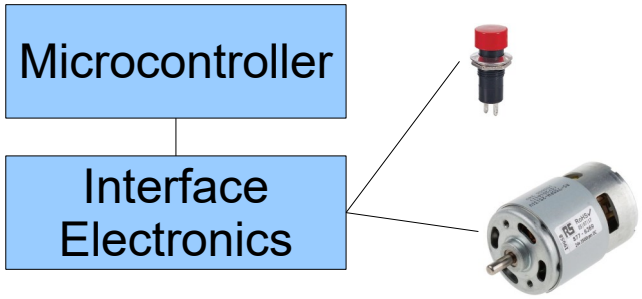
Microcontrollers and Ardunios

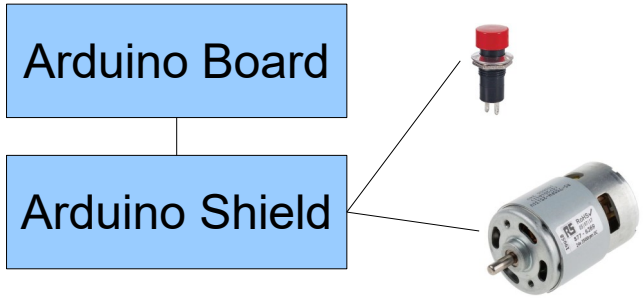
Microcontrollers are low-powered, single chip, computers designed to control simple devices like domestic appliances. A typical microcontroller is roughly equivalent to an 1980s computer like a ZX Spectrum in terms of computing power.

Microcontrollers run software to control a device. The microcontroller is interfaces to the physical word using special interfacing electronics.

The Arduino is a system of software and hardware that makes microcontroller capabilities easily available to the hobbyist.

Sensors, switches, motors etc.





Arduino Uno Anatomy

Pin 13 LED
Shows the state (high or low) of pin 13. Lights up with the pin is high.
Labelled “L”

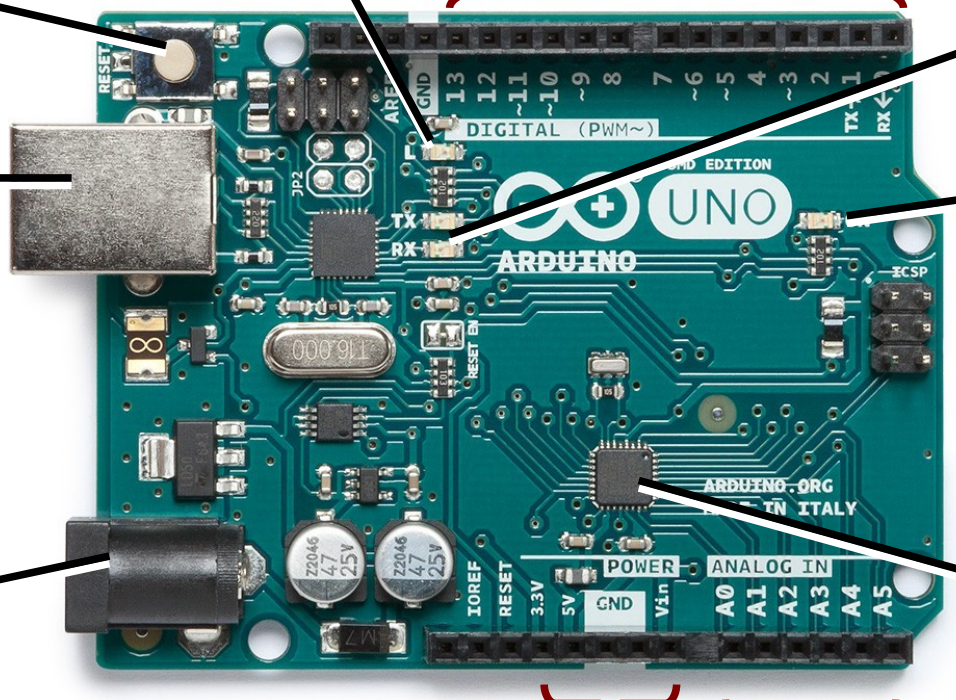
14 Digital pins labelled 0 to 13
Digital = “High” or “Low”; Input or Output
High is about 5 Volts; Low is about 0 Volts
Try and avoid using pins 0 and 1 as they are also used with programming the Arduino

Reset button
Press to restart the program. Reset happens automatically on power-on or when programming is completed.

programming LEDs
Flash to show programming operation

USB Connector
Connects to host computer

Power LED
Shows when Arduino is on



ATMEL Microcontroller
The “brains” of the Arduino

Power Connector
Provides power to circuit for operation without a host computer. Recommended range is 7 to 12 volts. A voltage regulator on the board will generate a 5V supply.

Power Connectors
Provides **limited** power to external circuits.

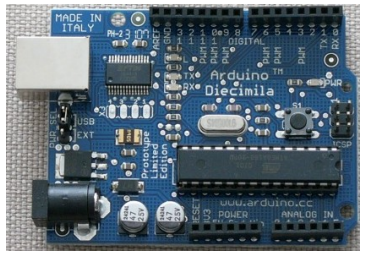
6 Analog Inputs labelled 0 to 5
Can be used to measure electrical voltages between 0 Volts and 5 Volts. Used to measure properties of the real world when used with the right sensors

An Arduino for (almost) every situation

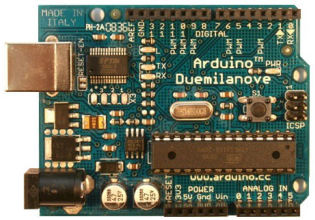
The main line

All have same board connections and support one of two microcontrollers – the ATMEGA168 or ATMEGA328. The 328 has twice as much memory.

Arduino Decimila



Arduino Duemilanove



Simplified power supply (automatic selection of USB or external power)

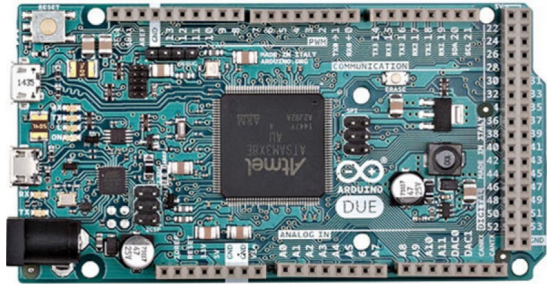
Arduino Uno

Modernized USB interface

Some black sheep

Arduino Due

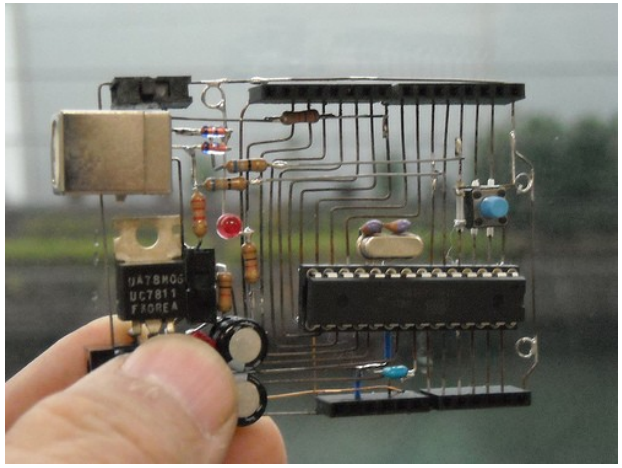
More powerful processor and more connectors.



Arduino Nano

Compact version

Arduino Skeleton



Made without a circuit board! Not commercially available.

… and many many more.

All share the same concepts, programming techniques and development tools

Arduino Shields

e.g. https://coolcomponents.co.uk/collections/arduino-shields
https://www.adafruit.com/category/17_21

Arduino vs Raspberry Pi

Very broadly:
- The Arduino is a low-cost device that is dedicated to controlling electronic systems
- The Raspberry Pi is a low-cost (and low power) desktop computer that can also control electronic systems via the GPIO

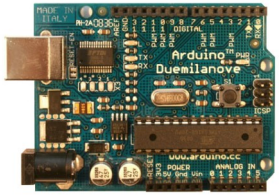
	Arduino	**Raspberry Pi**
Programing environment	Via host computer	Built-in
Programing language	C	Python / Anything
Onboard video and audio	No	Yes
Real-time control	Good	Mostly bad
Physical Form	Many official and unofficial options	Several options, but less diverse than the Arduino

Arduino programming

1. Write program on your computer

2. Send (“Upload”) the program to Arduino

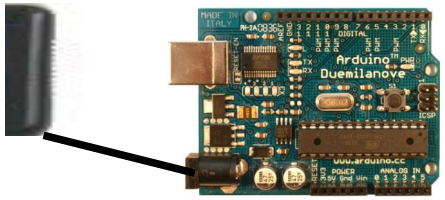
USB cable



3. Arduino runs program

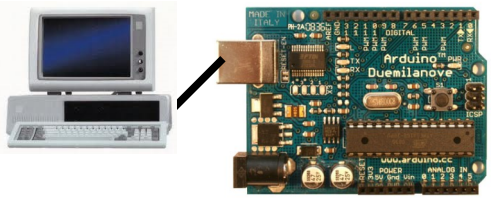
3a. Independently of host computer

Connect a suitable power source and the Arduino can run on its own



3b. With a host computer

A computer can power the Arduino and exchange information over USB



Getting Ready to Program the Arduino

In order to program the Arduino you can use their web-based environment or a destop application. We will use the desktop application in this workshop.

The Arduino web site does a good job of documenting this for various systems and is kept up to date, so just go here and follow the instructions for your type of computer:
https://www.arduino.cc/en/Guide/HomePage

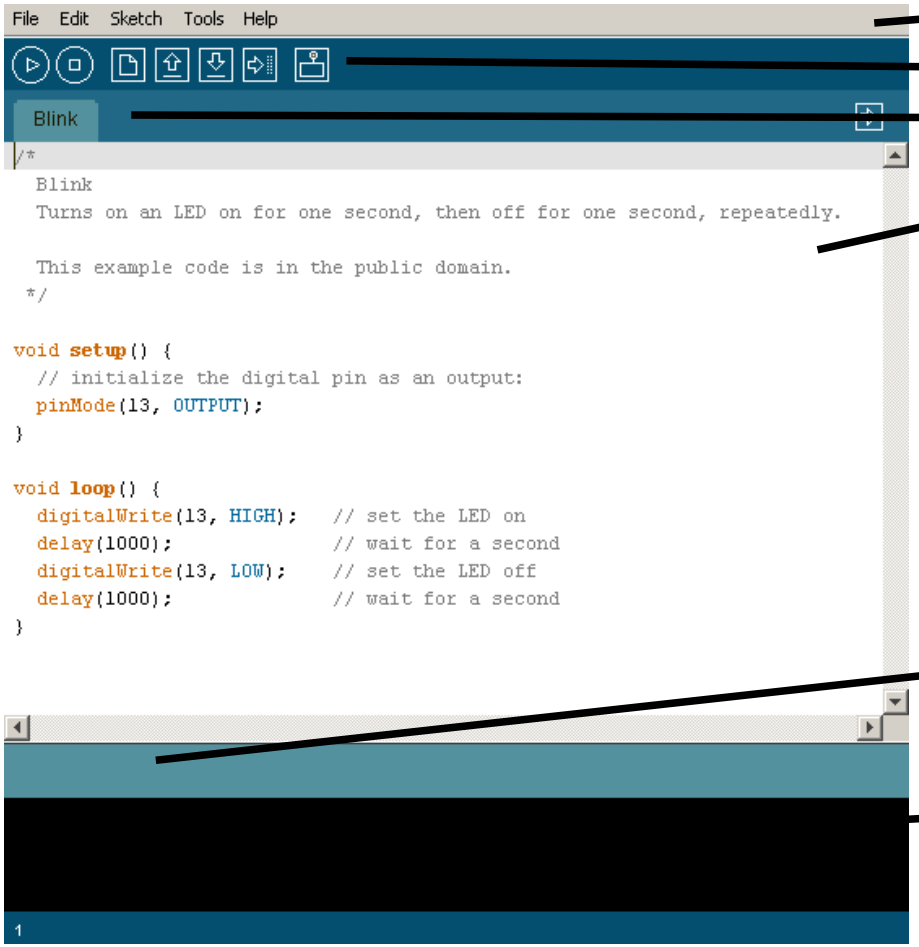
About Arduino programming

The Arduino program tells the Arduino what to do. The program is written in a programming language called Processing. Processing is based on another language called “C”.

A program for the Arduino is called a “Sketch”. Sketches are written in the Arduino programming environment which also provides the ability to upload the sketch in to the Arduino and communicate with the Arduino when it is running the program.

The Sketch consists of a number of commands in sequence. Normally the commands are executed one after another in the order they appear in the Sketch. However there are also special commands that can change the order of execution.

Arduino programming Environment



Configure the Desktop Application

Configure the “Board” and the “Port”

The desktop program needs to know what board it is controlling. Choose:

Tools → Board → Arduino/Genuino Uno

It also needs to know what port to connect to the board. This can be a little experimental, but on most machines there will be an obvious choice. Choose:

Tools → Port → COM4 (or whatever seems appropriate!)

An Arduino Blink Program

1) Connect your Arduino to the host computer using a USB lead.
2) Run the Arduino software on your computer.
3) From the File menu choose Examples → 01.Basics → Blink. The simple Blink sketch should load.
4) Click the Verify button to check the Sketch is OK. In the text area at the bottom of the screen a message like “Binary sketch size: 1008 bytes (of a 14336 byte maximum)” should appear once the verification is complete.

 Verify Button

5) Click the Upload button to upload the sketch on to the Arduino. During the upload the LEDs labelled “Rx and Tx” will flash to show the transmission of data. Once the sketch is uploaded you will get a message “Done Uploading” and the sketch will start to run. The pin 13 LED should flash.

 Upload Button

6) Try changing the values in brackets in the “delay” function call in the “loop” function. Repeat steps 4 and 5 and see if you notice a difference.

```
/*
  Blink
  Turns on an LED on for one second,
  then off for one second, repeatedly.
  This example code is in the public domain.
 */


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


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

Comment: Everything between a **/*** and a ***/** is a comment to help the reader understand the sketch. It is ignored by the Arduino

Setup function: Every sketch must have a setup function. This is run when the Arduino is reset to prepare for the rest of the program

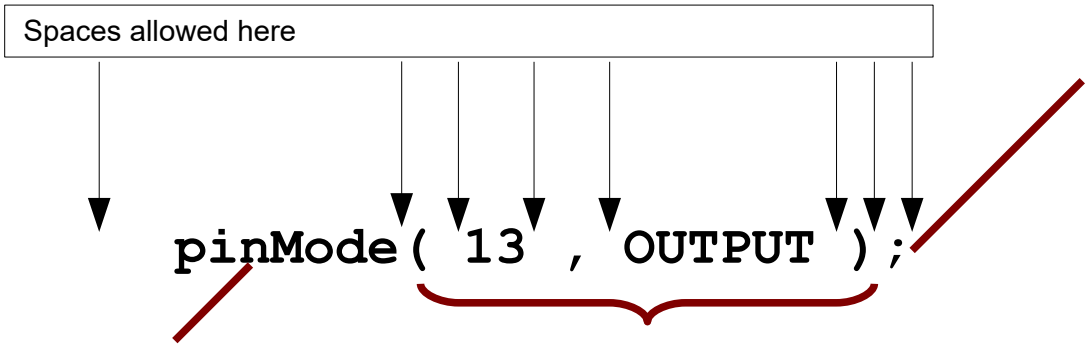
Comment: Everything on a line after a “//” is also a comment

Function call: This is a piece of program that triggers a function to perform a required action. In this case the function tells the Arduino to use digital pin number “LED_BUILTIN” as an output.

Loop function: Every sketch must have a loop function. After the setup the Arduino repeatedly performs the actions in the loop function

Using Functions

A function is a self-contained piece of computer program that is collected in to a wrapper to make it easy for a programmer to use. Processing contains many already defined functions you can use in your programs. When you want to use a function you write a function call in to the program. Here is an example function call from the Blink Program.



Delimiter: In Processing each statement in the language must end with a semi-colon. This is called a delimiter. Missing semi-colons (or semi-colons in the wrong place) are a common source of problems.

Name of the function: shows which function is being called. In Processing, the names are case sensitive (small and big letters must be correct). The name can't contain spaces.

Parameters of the function: Data values that are needed by the function are called “parameters”. They appear in brackets after the name. Different values are separated by commas. The number and types of data values is specified in the function definition.

In this case the first parameter is the pin number being set. The second parameter shows whether this pin is an INPUT or an OUTPUT.

If there are no parameters you still must include an empty pair of brackets.

When the program reaches the function call it goes and performs the tasks defined by the function. Once the function as finished the program resumes just after the function call. It is possible for one function to call another function in a nested structure.

Built-In Functions

We have already seen three built-in functions: PinMode(), digitalWrite() and delay(). There are many others. You can explore them through the example sketches
http://arduino.cc/en/Tutorial/HomePage
or at
http://arduino.cc/en/Reference/HomePage

Defining Functions

As well as using pre-prepared functions, you can also define your own. You **must** define a “setup()” and “loop()” function.

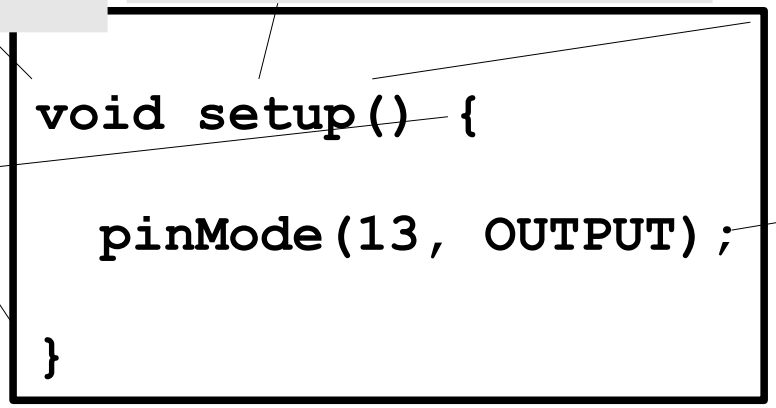
A weakness of processing is that the function definitions are a little complicated when you first meet them. This is due to the decision to build Processing on top of “C”. If you don't understand the detail, just treat this as a piece of magic that you can copy from examples as you need to.

Here is an example function definition from the “Blink” program.

Return types: Functions can return a value once they are completed. Simple functions don't do this, and to tell the computer you don't want to return a value you must put the word “void” in front of the function definition.

Name of the function: choose a name without spaces which is meaningful to you and doesn't overlap with any other function names. The names “setup” and “loop” have special meanings

Function parameters: We have seen how you can use parameters when you call functions. When a function is defined the parameter definitions go here. If there are no parameters you must include a pair of empty brackets to show that the parameter list is empty.



Curly brackets around the function body: A pair of curly brackets goes round the “body” of the function. The body contains the instructions for what to do when the function is called.

Statements: The function body is made up of one or more statements that do the work of the function. Another function call is an example of a statement. Each statement is separated by a semi-colon.

Blink Program with a Variable

1) Go back to the “Blink” sketch and edit it on the screen so it looks like the example on the right (changes are shown in bold).

2) Verify the sketch using the verify button.

3) Upload the sketch to the Arduino and check that it works.

```
int waitTime;

void setup() {

  // initialize the digital pin as an output

     waitTime = 500;

     pinMode(LED_BUILTIN, OUTPUT);

}



void loop() {

  digitalWrite(LED_BUILTIN, HIGH);  // set the LED on

  delay(waitTime);                  // wait for delay specified

  digitalWrite(LED_BUILTIN, LOW);   // set the LED off

  delay(waitTime);                  // wait for delay specified
}
```

Variable declaration: This line in the sketch creates a new variable called “waitTime”. A variable is a labelled box which is used to contain data used in the program. The data can be read or changed during the program.

In this case waitTime will contain integer values. We use the word “int” to tell the computer we want waitTime to be an integer.

A variable declared outside any function can be seen by all functions in a sketch.

Variable assignment: This statement puts a value (500) in to the variable waitTime.

Variable use: We can now use the variable name to mean the contents instead of actually writing “500”.

LED Chain Circuit

1) Construct the circuit with the Arduino and breadboard as shown on the right. We will use variations of this circuit for the next few experiments.
2) In the sketch used in the previous experiment change the first parameter in the calls to pinMode and digitalWrite from “LED_BUILTIN” to “2”. Upload this sketch and you should see the left-hand LED flash.

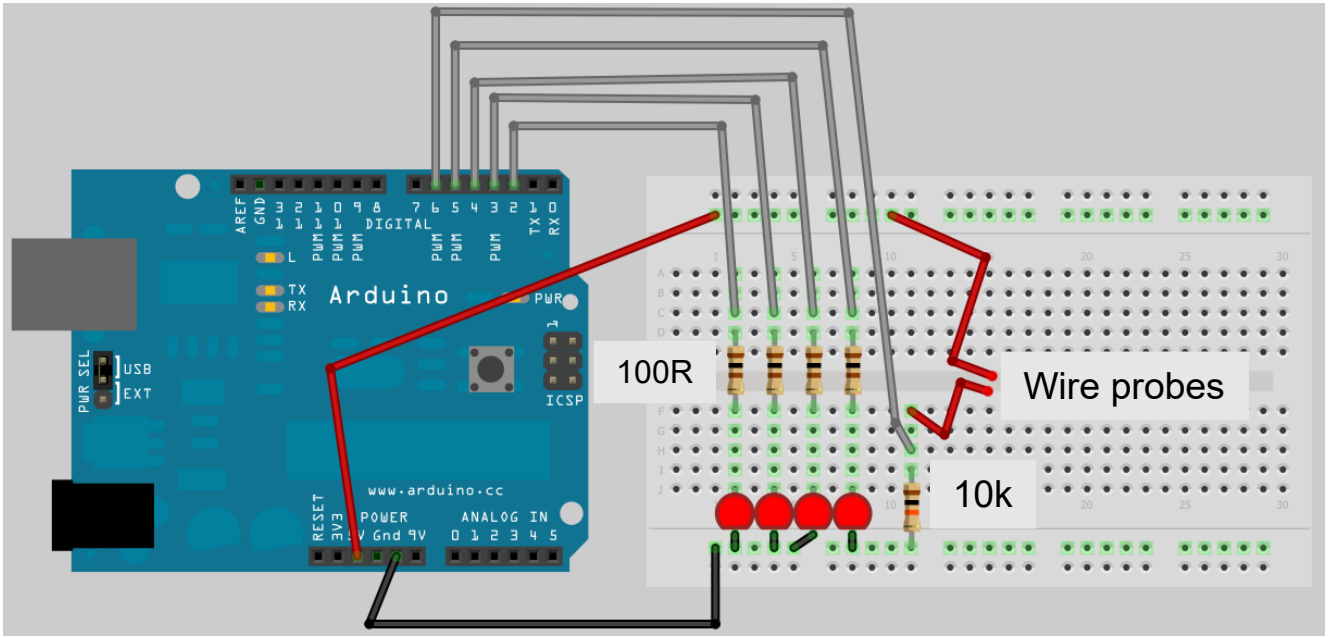
3) Using the menu option File → Open... open the sketch “chain.pde”. Upload this sketch to the Arduino. You should see the lights move in a chain pattern.
4) Try modifying the Chain sketch to get other patterns – eg reversing the direction of the chain, or having the LEDs light in pairs.

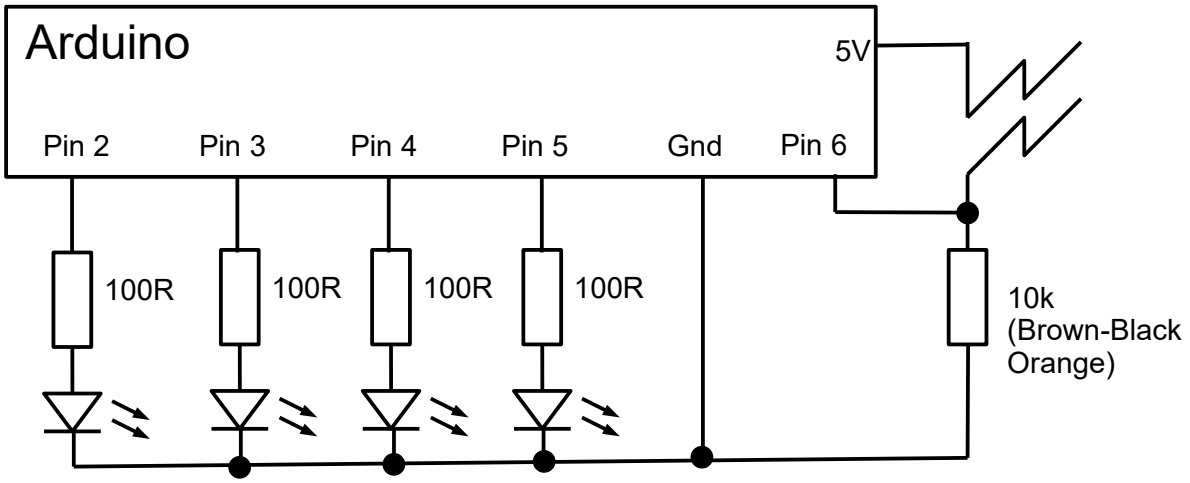
5) Using the menu option File → Open... open the sketch “digitalInput.pde”. Upload this sketch to the Arduino.
6) With the digital input sketch running try touching the ends of the wire probes together. What happens? In a more permanent circuit the probes could be replaced by a switch or a button.

The digitalInput sketch uses a new capability – the possibility to treat an Arduino pin as an input. It also uses an “if” statement (see next slide) to change the behaviour of the Arduino based on the input.

Let's look at what is happening electrically. When the probes are disconnected the 10k resistor “pulls” the voltage at the input low. When the probes connect the input is connected directly to the 5V supply from the Arduino and the input goes high. This is a very common configuration. The 10k resistor is called a “pull down resistor”.

NOTE: We are taking power for the circuit from the Arduino

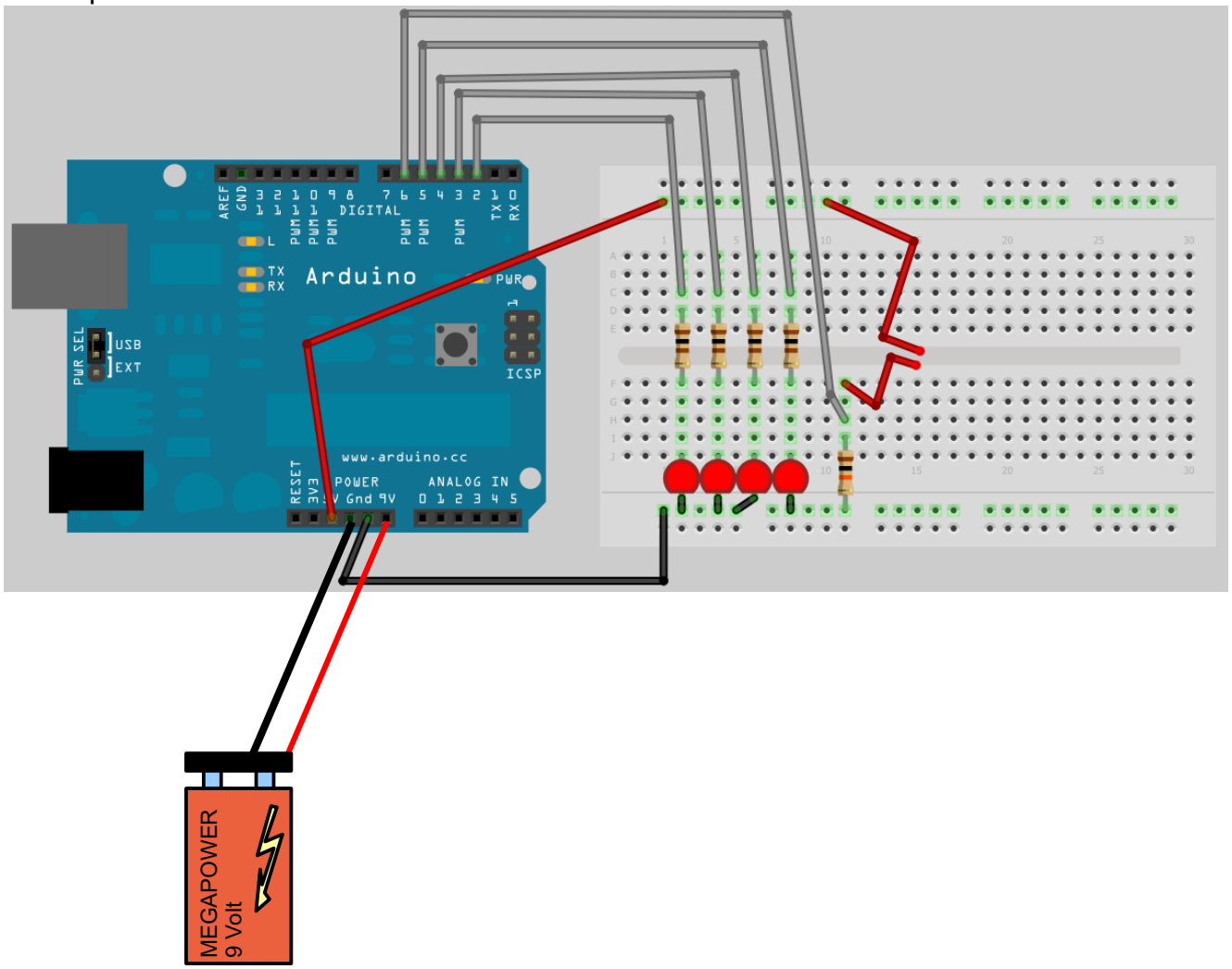




Removing the PC

Once the Arduino is programmed it no longer needs a PC. You can power it from a bettery or other suitable power source.

To test this, disconnect the USB and connect the positive battery lead to the “9V” or “Vin” pin and the negative battery lead to the other GND pin.



New Program Elements in digitalInput sketch

```
pinMode(inputPin, INPUT);
```

Set Pin Mode: Done in the setup function. This says that we want an input on the inputPin.

```
inputValue=digitalRead(inputPin);
```

Reading the input: This is done every time the sketch goes round the main loop. The sketch looks at the value of the input pin and stores either HIGH or LOW in the variable called “inputValue”. Think of the variable as a labelled box to store values. Like other variables the type of “inputValue” was declared at the start of the program

IF statement: This is used if you want to make the sketch do different things based on the environment or the results of a past activity.

Condition: After the word “if” you have brackets that contain the condition that is being tested. The two equals signs (“==”) means “has the value of” or “is the same as”. Here we see if the variable inputValue contains the value “LOW”.
Other possible conditions include “!=” for “not equal to”, “>” for “greater than” and “<” for “less than”.

```
if (inputValue==LOW) {
     ... things to do...
  }
   else {
     ... things to do....
  }
```

True actions: After the condition you have a list of statements surrounded by curly brackets. These are what the sketch does if the condition being tested turns out to be true.

False actions: After the true actions you may have the world “else” and then another list of actions in surrounded by curly brackets. These are what the sketch does if the condition being tested is false.

Making Mistakes (and fixing them)

Mistakes – we all make them. In computer programming there are two different types of mistake you can make in the software. A “syntax error” means you have written something in the program that the computer can't understand. If your sketch contains syntax errors it can't be uploaded to the Arduino.

A “semantic error” means your sketch is understandable but you discover it doesn't do what you meant it to do. This normally means you have made a logical mistake in designing your sketch.

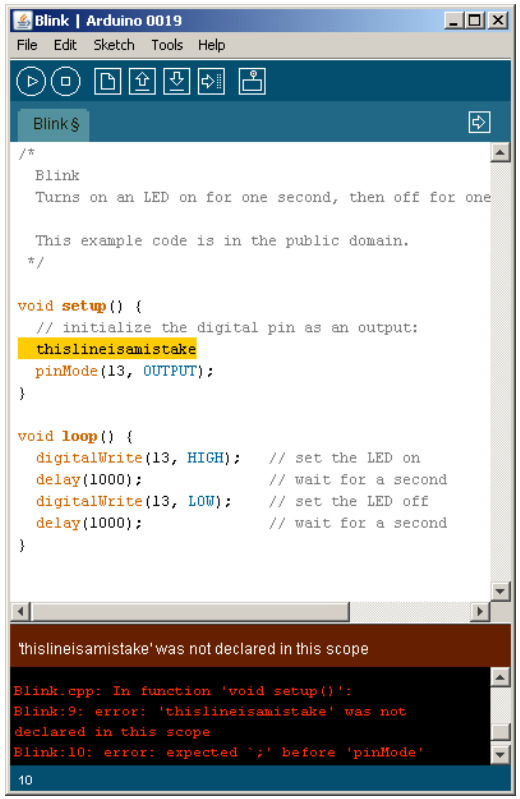
Syntax Errors

Syntax errors will be detected when you either verify a sketch or try and upload it to the Arduino. **When a syntax error is detected the status bar will turn from blue to red** and red error messages will appear in the program notification area. The area where the computer detected the mistake will be highlighted in the editor.

Unfortunately Processing doesn't handle syntax errors very well. Its messages can be very hard to understand (even for experts) and it doesn't always guess right where the error occurred

Here are some things to check to try and fix any errors:
- Mistyping of names (spelling or case of letters)
- Missing semicolons after statements
- Unpaired round brackets or curly brackets
- Spaces in the middle of names

To minimise problems with syntax errors I suggest you start by evolving the examples and make small steps towards what you want, frequently verifying so that mistakes get caught early.



Semantic Errors

Semantic errors are normally called “bugs”. They could be anything from a program that does nothing to a program that works 99% of the time but occasionally fails unexpectedly.

The process of finding and fixing semantic errors is often a piece of detective work. Normally you should test your sketches thoroughly to make sure that they behave the way you want them to under all circumstances.

If a sketch doesn't do what you expect then try and work out why it follows the behaviour it's showing. Pretend you are the Arduino and “dry run” the instructions in your head. Think about where in your sketch the problem may be and what circumstances trigger the problem. Try and narrow down the range of possible points where the problem originates.

Making A Noise

Let's move beyond blinking LEDs. In this experiment we will add a speaker to the Arduino and start to make some noise. This also provides a useful demonstration of how to connect larger loads to the Arduino.

1) Add the extra elements shown to the existing circuit.
2) Open the sketch “digitalInputwithtone” and upload it to the Arduino.
3) When you touch the wire probes together you should now get a tone from the speaker.
4) Examine the sketch and see the new elements that create the tone. Try changing the tone behaviour.

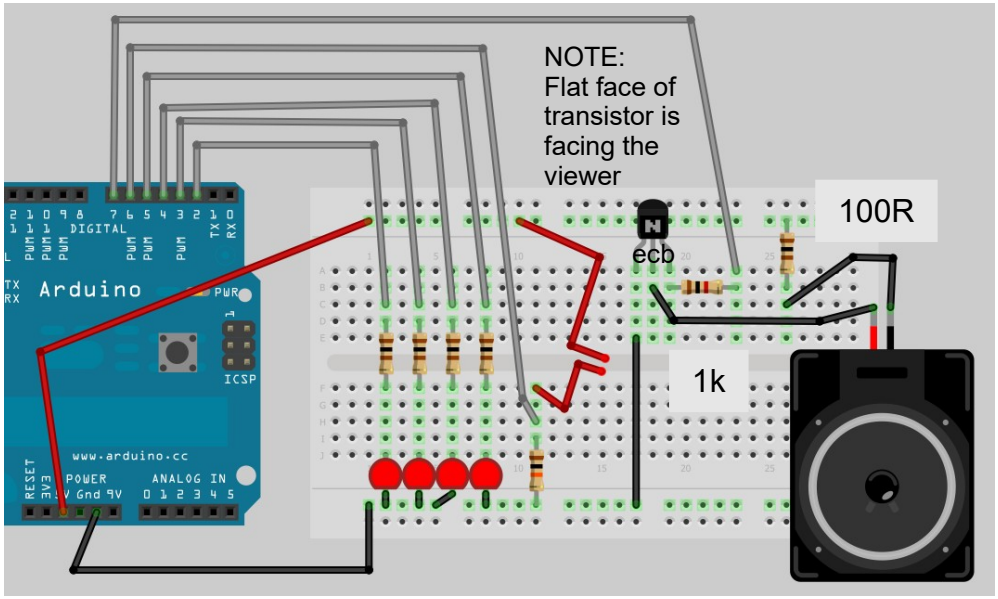
How the electronics works

The Arduino pins have a very limited capacity to drive electrical current. One LED is OK, but much more is dodgy. We can use a transistor to boost the output to drive more demanding loads – in this case a speaker.

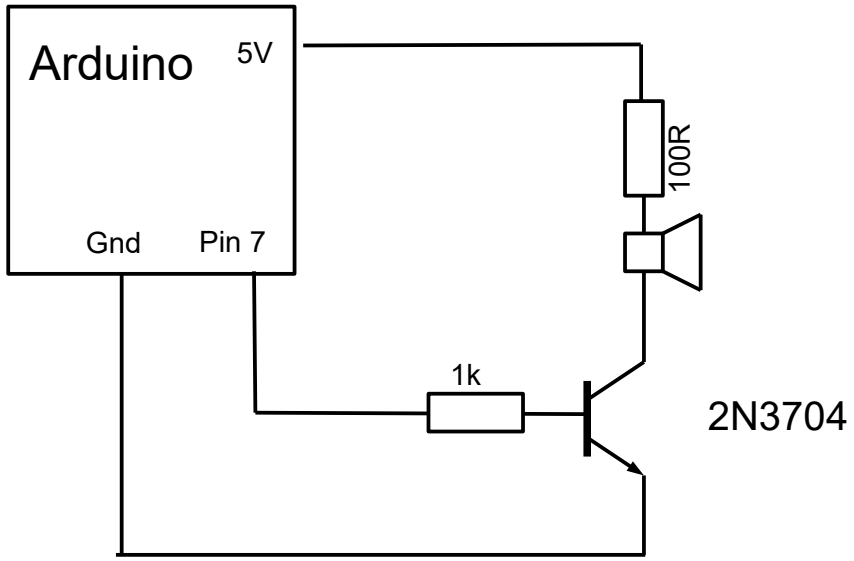
In the first transistor experiment we showed how a small current on the base of the transistor could drive a much bigger current through the collector. We use exactly this idea here. The transistor is controlled from the Arduino by connecting the base to an output pin via a resistor (1k in this example). The resistor limits the current taken from the Arduino.

Even with a transistor, connecting 5V directly to the speaker is too much. The 100R resistor in the collector limits current through the speaker.

NB: Many examples on the web don't put a resistor in the base. Without the base resistor the circuit is poorly designed any may damage the Arduino.



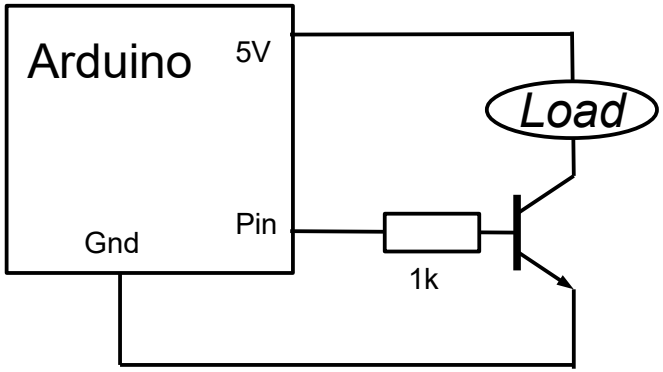
Note: Circuit Diagram only shows the speaker connection



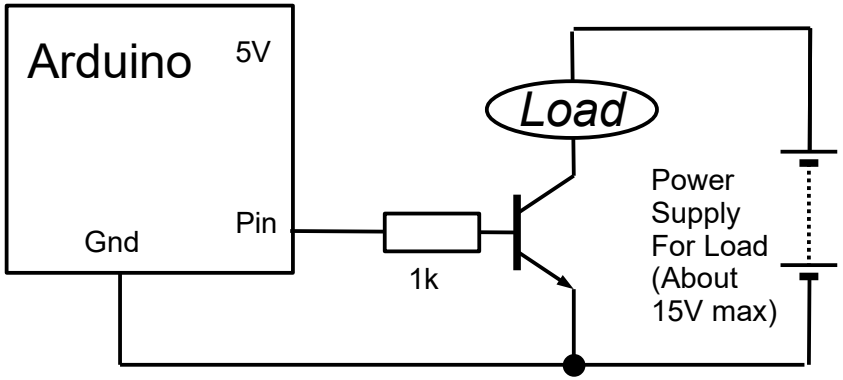
Driving Loads

We saw in the last experiment how a transistor can be used to let the Arduino drive an electrical load like a speaker which is too big for the Arduino to drive on its own (for reference the maximum current load per pin is 40mA). The transistor configuration is a general technique that can be used to drive most low-voltage electrical loads.

For smaller loads (say up to 250mA) this general circuit can be used.



For larger loads, or loads that need a different power supply voltage to the Arduino's 5V the following general circuit can be used. You may need to check the suitability of the transistor. The 2N3704 we are using in the experiments has a maximum capacity of 500mA. To test the current taken by a load connect it to a power supply with a current meter in the path.

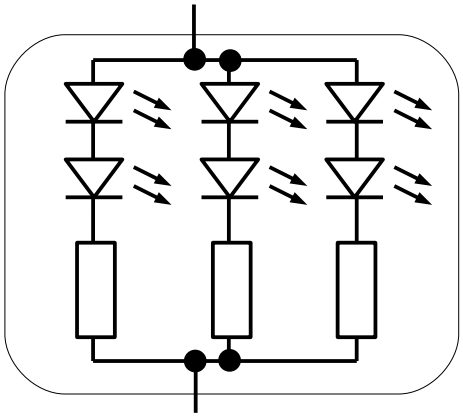


Some useful loads

You can use the general circuit patterns shown on the left to drive these loads.

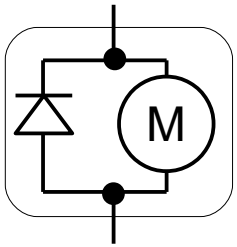
LED Chains

With a transistor you can drive lots of LEDs from a single pin. Various calculators on the web can help you find the right configuration for the number and type of LEDs you want to use. Note though that the online calculartors don't always follow good design rules so learning the manual approach is still useful.



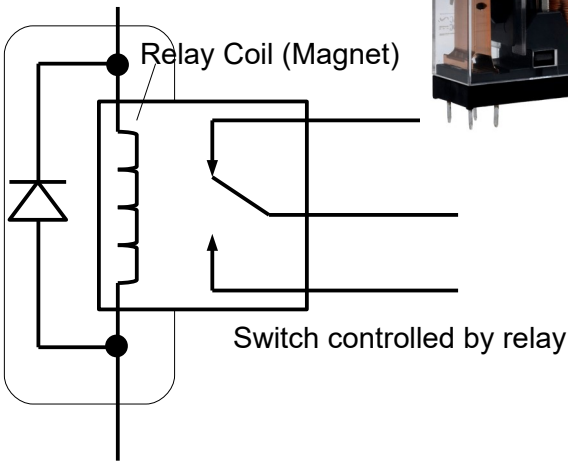
Small Motors

Small motors (like those in toys) can be controlled. Watch the current and voltage requirements – even small motors can have a high current when stalled. A diode should be connected as shown to protect the circuit from a reverse voltage that any electromagnet generates when it is switched off.



Relay

A relay is an electrical switch which is moved by an electromagnet. By connecting the magnet as a load to the Arduino the switch can be used to control another circuit completely independently. This has many uses – eg if you are controlling something that is too high power to be easily done with just a transistor, or if you don't know the full electrical characteristics of the thing being controlled (eg you are faking a button press on another piece of equipment). Relay coils also need a protection diode as shown



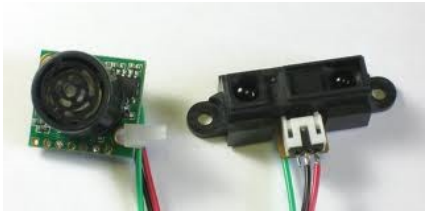
Other things you can connect (with the right electronics)

Servo Motors
Servo motors were originally designed for use in radio control models. They have a lever which is moved through about 280 degrees of rotation by the motor. The Arduino has special commands to control stepper motors.

Stepper Motors
Stepper Motors are special motors that move in individual steps. They can be used for precise control over position or speed of rotation.

Infrared Choppers
An infrared light can be shone through a gap to detect levers or other mechanical parts that interrupt the light beam

.

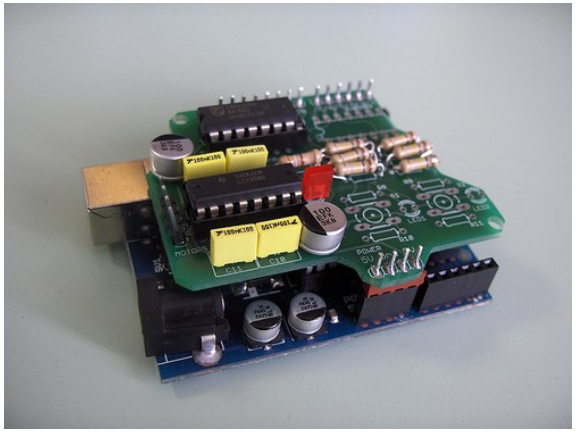


Distance Sensors

Arudino Shields

To make it easy to connect the standard Arduinos to external systems you can use pre-made “Arduino Shields”. These connect on top of the Arduino board.

Example shields:
- Motor control
- Ethernet
- Xbee
- Servo motor and Stepper Motor shields



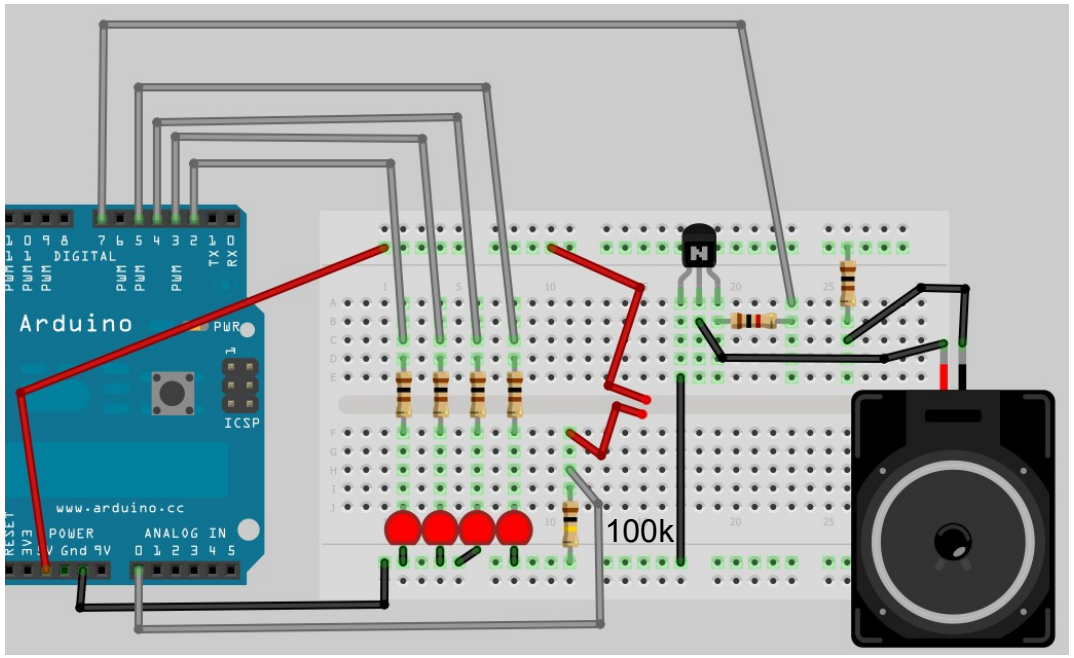
Arduino Analog Inputs

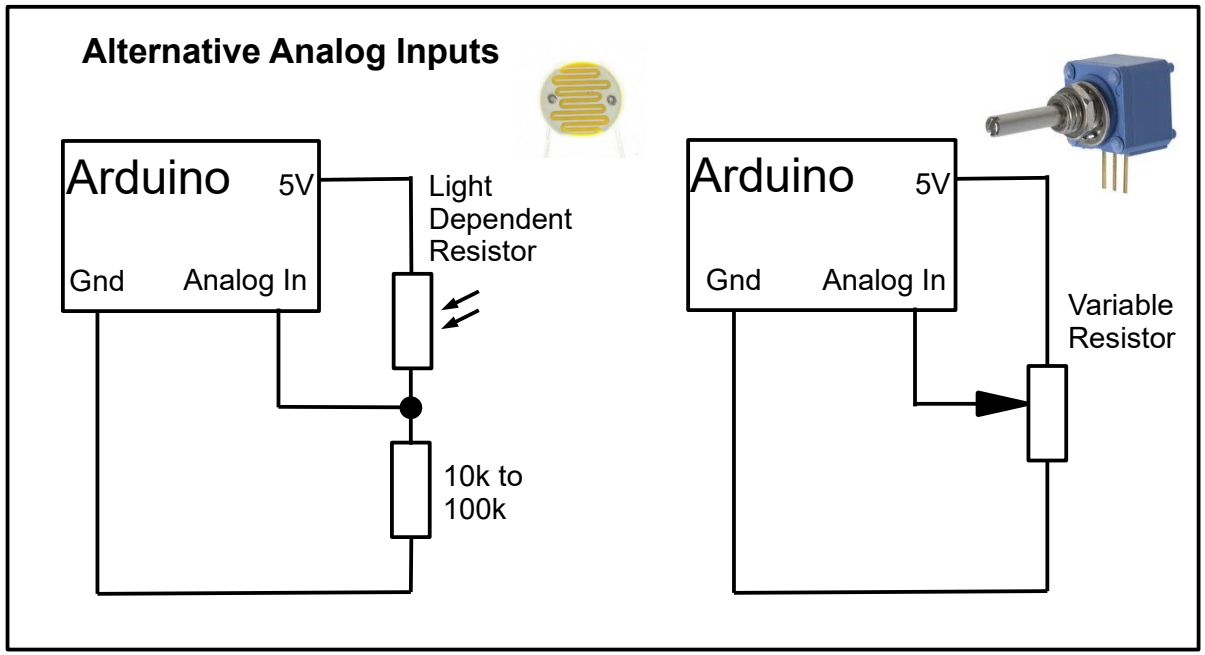
We've seen the Arduino's digital inputs in action. Now we will look at an analog input. The analog inputs can measure and report the voltage at the pin in the range 0V to 5V.

1) Modify the circuit as shown (replace the 10k resistor with a 100k resistor and move the input wire from digital Pin 5 to Analog Pin 0).
2) Open the sketch “simpleorgan” and upload the the Audruino
3) Try touching the wire probes to a pencil line track or to your skin. You should get an organ effect.

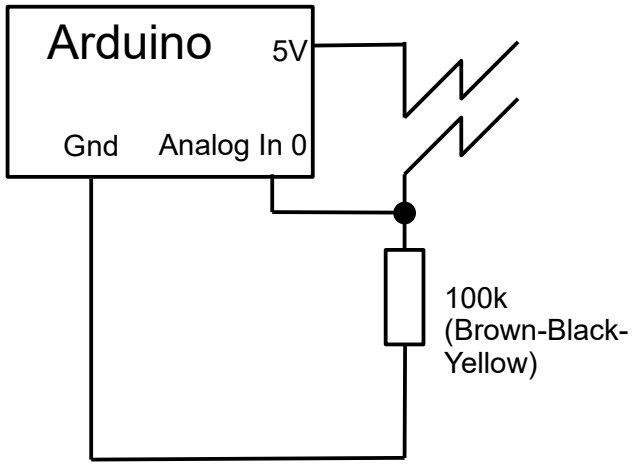
The circuit used here is called a “voltage divider”. The voltage at at the Analog input pin depends on the ratio of the 100k resistor to the resistance between the two probes. As the resistance between the probes decreases the voltage at the analog input goes up.

The analog input can be connected to many types of sensor. There are some common examples shown below.





Note: Circuit Diagram only shows the analog input



Servo Motor Example

Servo motors were originally designed for use in radio controlled models. They can change their angle based on a control signal. The Arduino can control the motor.

Construct a circuit with the servo motor as shown.

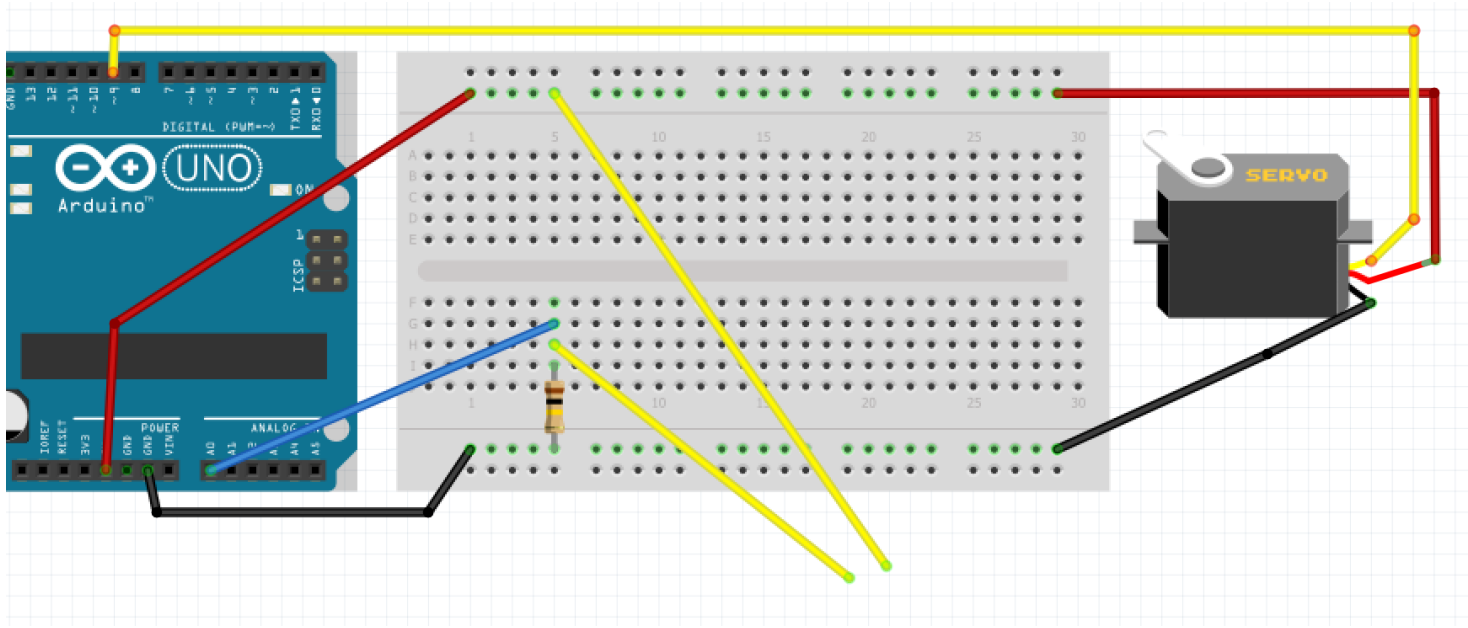
Try the examples:

File→ Examples → Servo → Sweep

And

File → Examples → Servo → Knob

In the second example the analog input will control the servo position.



Pulse Width Modulation (PWM)

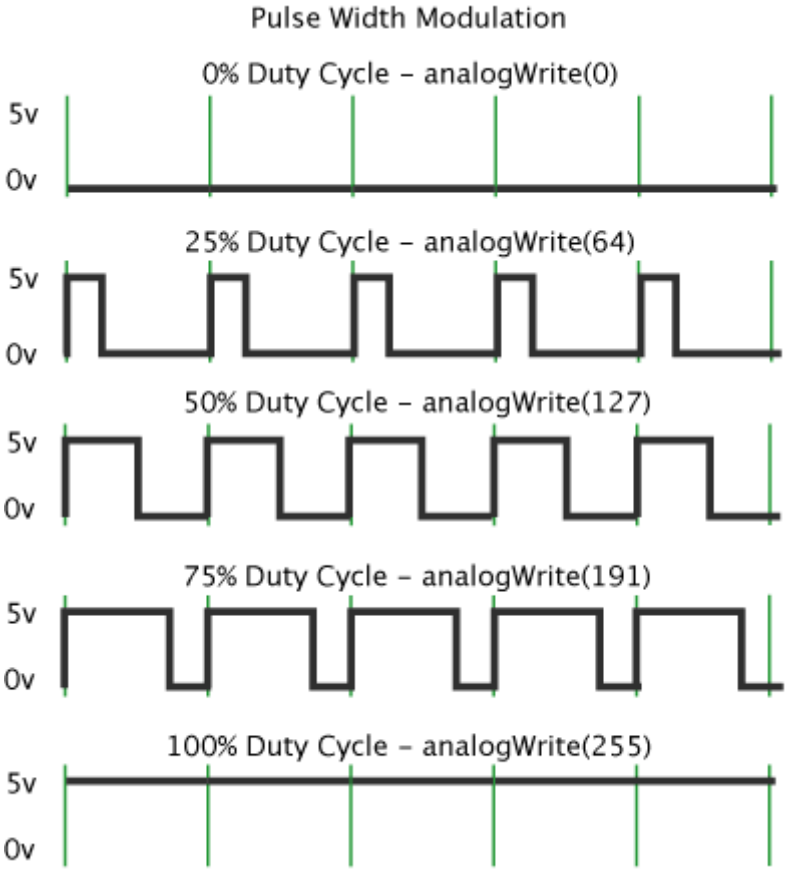
The Arduino only has digital (0V or 5V) outputs. However it can output an approximation of an intermediate analog voltage using a technique called Pulse Width Modulation (PWM). In PWM a digital output moves between the on and off states very quickly. The ratio of the the “on” time to the “off” time (called the “duty cycle”) can be varied to create an average voltage which is intermediate between on and off voltages.

PWM signals aren't suitable for everything, but they do work well for some types of system – in particular:
LEDs where the rate of blinking is too fast for the human eye to see, and
Small motors which can have the speed varied by PWM.

PWM in the Arduino

PWM is only available on the special outputs labelled with “PWM” on the board. There are further restrictions if you want to combine PWM with the “tone” function. To send a PWM signal set the pin as an output and use the analogWrite() function with a value between 0 for all off and 255 for all on (see diagram). The designers of Arduino made a big mistake naming this function – there is no analog output. “PWMWrite()” would be a much better name!

To test the effect of PWM on an LED keep the same circuit as before and load the sketch “organandlight” and upload to the Arduino This sends a PWM signal to pin 6 which depends on the pitch of the organ – the LED should be bright for low notes and dimmer for higher notes.



LED Off

Motor Off

LED Dim

Motor Slow

LED Bright

Motor Fast

Arduino Organ II with light effects

A fun thing to finish (and show a more complicated level of Arduino programming).

Using the same electronics as before load the sketch “organandlight2” and upload to the Arduino. You should now have an organ with lights that flash in sequence based on the frequency.