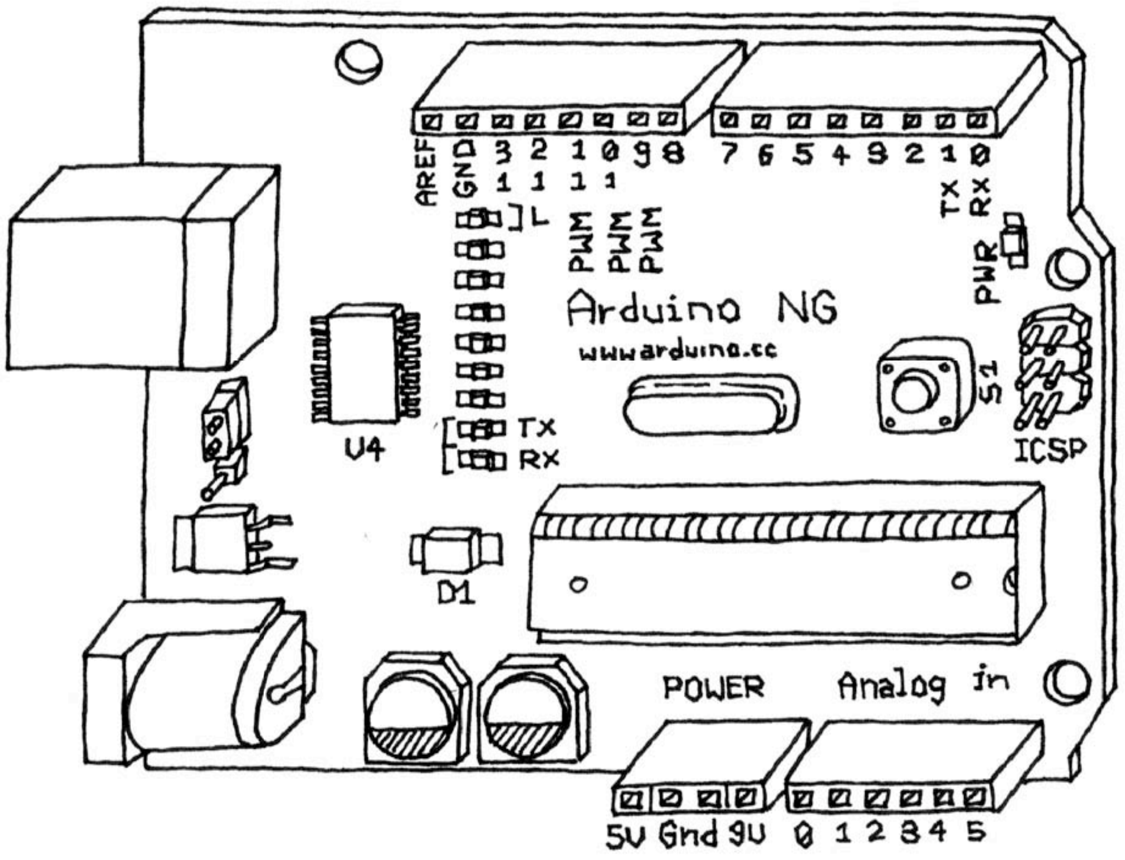
Getting Started with

Arduino 3rd release



by Massimo Banzi

Aknowledgments

This booklet contains materials or code written by Erica Calogero, David Cuartielles, Jeff Gray, Tom Igoe, David Mellis and Christian Nold.

The Arduino team is composed of:
Massimo Banzi, David Cuartielles, Tom Igoe, David Mellis and Gianluca Martino.

The Arduino team would like to thank the following people and institutions for the support in making this booklet:

Ailadi Cortelletti for the typesetting of the second release.
Elisa Canducci for the illustrations.
Alexander Zivanovic for corrections and feedback.

Interaction Design Lab, Milano
Domus Academy, Milano
Interaction Design Institute, Milano
Malmö University, Faculty of Art, Culture and Communication (K3)

Edited and typeset by Alexandra Deschamps-Sonsino

Index

Introduction

Arduino is an open-source physical computing platform based on a simple i/o board and a development environment that implements the Processing language. Arduino can be used to develop stand-alone interactive objects or can be connected to software on your computer (e.g. Flash, Processing, Max/MSP). The boards can be assembled by hand or purchased pre-assembled; the open-source IDE can be downloaded for free.

Arduino is different from other platforms that can be found on the market for a number of reasons:

- 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, Macintosh and Linux.
- It is based on the Processing programming IDE
- It is programmed via a USB cable not a serial port. This is useful because many modern computers don't have serial ports anymore.
- It is Open Source hardware and 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 cheap. The USB board cost about €20 and replacing a burnt out chip on the board is easy and costs no more than € 5. So you can afford to make mistakes.
- There is an active community of users online so help is never very far away.

In short, this booklet is designed to help beginners to understand what benefits they can get from learning how to use the Arduino platform and adopting its philosophy.

Note: This booklet was written for the "original" Arduino users, designers and artists; therefore it tries to explain things in a way that might drive some engineers crazy. Actually one of them defined the introductory chapters as "fluff". That's precisely the point. Let's face it: most engineers aren't able to explain what they do to another engineers let alone a regular human being. Let's now delve deep into the fluff.

Foreword

A few years ago I was given a very interesting challenge: teach designers the bare minimum in electronics so that they could build interactive prototypes for their projects.

I started following a subconscious method of teaching electronics that resembled the way I was taught at school. I eventually realised that it simply didn't work as well as I would have liked and reminded me of sitting in class, bored to death as someone threw theory at me without any practice. I already knew electronics in a very empirical way: through a lot of hands-on experience and very little theory and started to think about how that had happened.

As a child, I was fascinated by discovering how things worked and used to take them apart. This passion grew as I targeted any unused object in the house and would take it to bits. Eventually, people would bring all sorts of devices for me to dissect. My biggest project at the time was a dishwasher and an early computer that came from an insurance office which had a huge printer, electronics cards, magnetic card readers and many other parts. This proved quite a challenge to take apart.

After quite a lot of this dissecting I knew what electronic components were and roughly what they did. On top of that, my house was full of old electronics magazines that my father must have bought at the beginning of the '70s and I spent countless hours reading the articles and looking at the circuit diagrams without understanding very much.

This process of reading these articles in the light of knowledge acquired while taking apart circuits created a slow virtuous circle.

A breakthrough happened one Christmas. My dad gave me a kit that would allow teenagers to learn about electronics. Every component was housed in a plastic cube with the electronic symbol written on top of it. The cubes would magnetically snap together with other cubes establishing a connection. Little did I know that the toy was designed by Dieter Rams (landmark designer from Germany) in the 60's.

With this new tool I could quickly put together circuits and try out what happened. The prototyping cycle was getting shorter and shorter.

From then on I built radios, amplifiers, circuits that would produce horrible and occasionally nice sounds, rain sensors & tiny robots.

I spent a long time finding an English word that could sum up a way of working without a specific plan, starting with one idea and ending up with a completely unexpected result and "Tinkering" came along. I subsequently discovered that this way of operating has been used in many other fields to describe people who set themselves on a path of exploration. A generation of French directors who gave birth to the "Nouvelle Vague" were called the "tinkerers" for example. My favorite definition of tinkering comes from an exhibition held at the Exploratorium in San Francisco:

> *"Tinkering is what happens when you try something you don't quite know how to do, guided by whim, imagination and curiosity. It's about letting the spark of an idea ignite, no matter how far the idea is. When you tinker, there are no instructions, but there is no failure, no right or wrong ways of doing things.*
>
> *[It's] about figuring out how things work – and about reworking them. It's about pursuing a fascination and letting it unfold into a process of discovery, about seeing the potential in a pile of junk and creatively re-imagining it as a treasure. Contraptions, machines, wildly mismatched objects working in harmony- this are(check quote) the stuff of tinkering. Tinkering is, at its most basic, a process that marries play and inquiry"*

From my early experiments I recognised how much experience you need in order to be able to create a circuit that would do what I wanted starting from the basic components.

In the summer of 1982 ,I went to London for the first time with my parents and spent many hours visiting the Science Museum. They had just opened a new wing dedicated to computers and, by following a series of guided experiments, I learnt the basics of binary math and programming.

It's there I realised that in a lot of applications engineers were not building circuits from basic components anymore but that they would implement a lot of the intelligence in their products using microprocessors. Software was replacing many hours of electronic design and would allow for an even shorter tinkering cycle.

When I came back home, I started to save up some money in order to buy a computer to learn how to program.

My first and most important project after that was to use my brand new ZX81 computer to control a welding machine. Although not a very exciting project, there was a need for it and it was a great challenge because I had just learnt how to program. At that point, it became clear that writing lines of code would take less time than replacing complex circuits.

20 years later, I'd like to think this experience allows me to teach people who don't even remember taking any math class and infuse them with the same enthousiam and ability to tinker than I had in my youth and have kept since.

Massimo.

/ what is interaction design?

There are many definitions of Interaction Design but the one I prefer is:

"Interaction Design is about the design of any interactive experience".

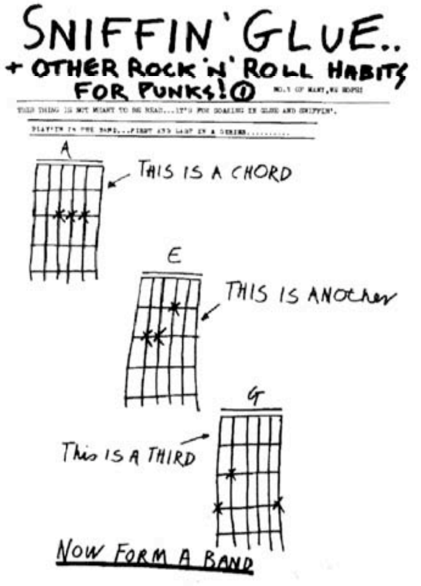
It also applies to the creation of meaningful experiences between us (humans) and artefacts. It is also a good way to explore the creation of beautiful and maybe even controversial relationships between us and technology. Interaction design attempts to emcourage design through an iterative process based on prototypes of ever increasing fidelity. This approach, also part of some types of more traditional design fields, can be extended to include prototyping with technology and in particular with electronics. This particular type of Interaction Design is called Physical Computing.

This booklet is in no way a substitute for a book on Physical Computing however and we recommend you buy Tom Igoe's excellent "Physical Computing" book.

/ what is physical computing?

Physical Computing involves prototyping with electronics, turning sensors, actuators and microcontrollers into materials for designers and artists.
It involves the design of interactive objects that people can interact with using sensors and actuators controlled by a behaviour implemented as software running inside a microcontroller (a small computer on a single chip).

In the past, using electronics meant having to deal with engineers all the time and this kept the designer from playing directly with the medium. Most of the tools were meant for engineers and required extensive knowledge.
In recent years, microcontrollers have become cheaper and easier to use allowing the creations of better tools. With Arduino, we tried to bring these tools one step closer to the novice , allowing anyone to start building projects after only a 3 or 4 day-long workshop. A designer or artist can get to know the basics of electronics and sensors very quickly and start building prototypes with very little investment.



// From the punk-zine "sniffin' glue" England, circa 1977

The Arduino way

The Arduino philosophy is based on making design rather then talking about it. It is a constant search for faster and more accurate ways to build better prototypes. We have explored many prototyping techniques and developed ways of thinking with our hands.
The classic engineer relies on a strict process for getting from point A to B while the Arduino way is based on getting lost on the way and finding C instead.
This is the process of tinkering that we are so fond of: playing with the medium in an open-ended way and finding the unexpected. In this search, we also selected a number of software packages that enable the process of constantly manipulating the software and hardware.

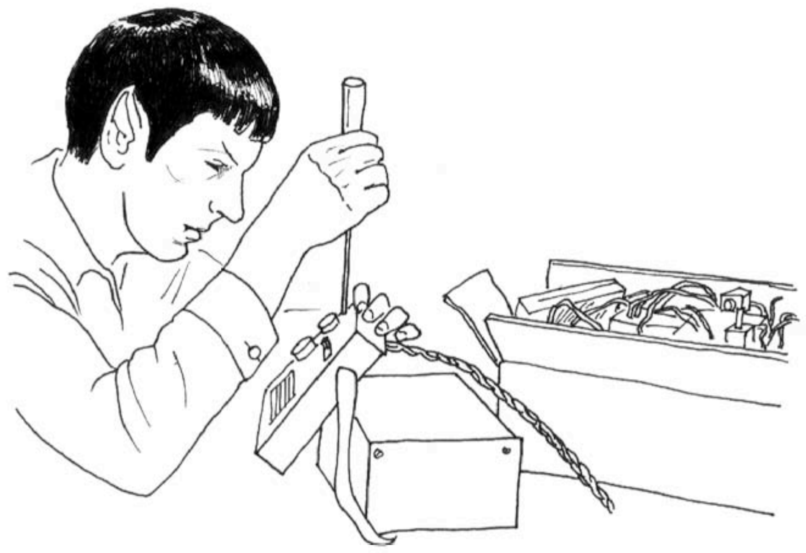
Another concept we developed is the idea of "opportunistic prototyping". Why spend time and energy building from scratch, a process that requires time and profound technical knowledge, while we can take already-made devices and hack them in order to exploit the hard work done by large companies and good engineers?

This became evident in Ivrea where the demise of Olivetti means that the junkyards in the area are full of computer parts, electronic components and devices of any sort which we could buy for a few euros and hack into our prototypes.

The last element is the community: engage people and push them to share by being the first to share. We stand on the shoulders of the giants of Open Source.

The next few paragraphs present some references that have inspired the "Arduino Way".

/ tinkering



We believe it is essential to play with the medium, exploring different possibilities directly with hardware and software, sometimes without a definite goal.
We call this process *tinkering*.

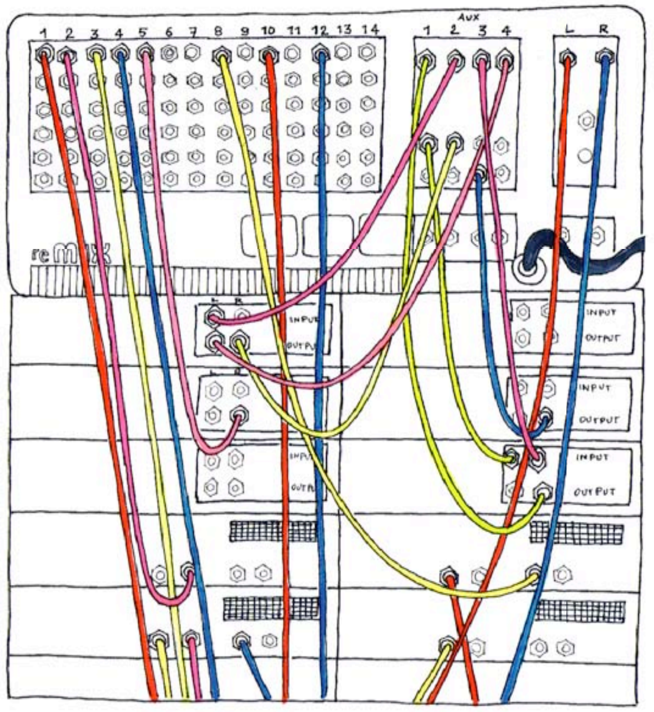
> *"Tinkering is what happens when you try something you don't quite know how to do, guided by whim, imagination and curiosity."*

When you tinker, there are no instructions - but there are also no failures, no right or wrong way of doing things. It's about figuring out how things work and reworking them.

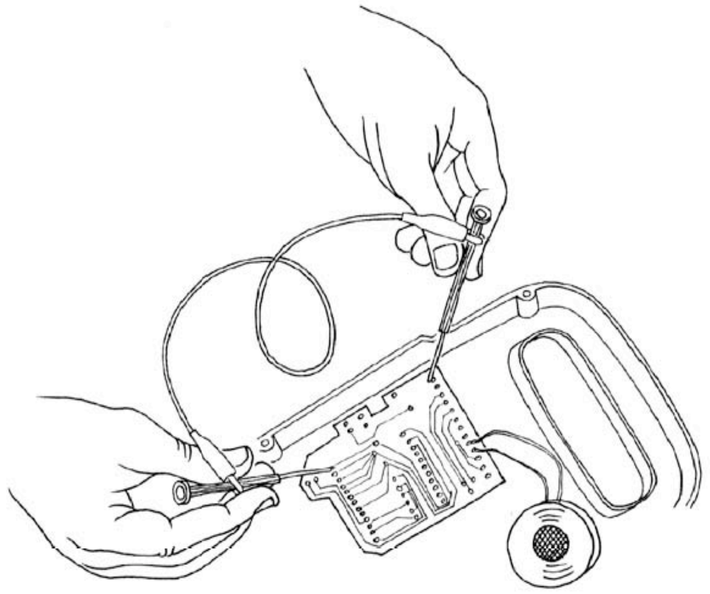
Re-using existing technology is one of the best ways of tinkering. Getting cheap toys or old discarded equipment and hacking them to make them do something new is one of the best ways to get to great results.

/ *patching*

Robert Moog built his analogue synthesizers in a modular fashion and the musician could try endless combinations by "*patching*" together different modules with cables. This made the synthesizer look like an old telephone switch which, combined with the numerous knobs, was the perfect platform for tinkering with sound and music. This technique has been translated into the world of software (Max/MSP and Pure Data for example).



/ *circuit bending*



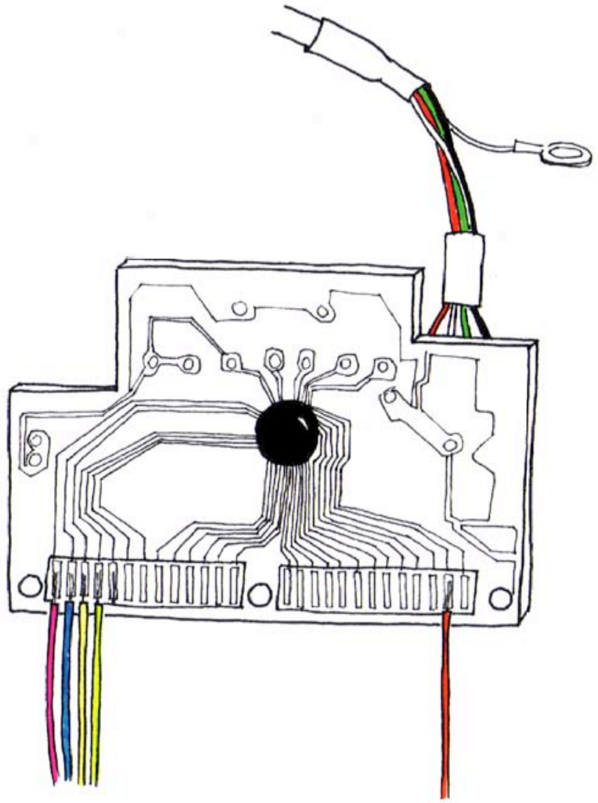
Circuit bending is one of the most interesting forms of tinkering.

It's the creative short-circuiting of low voltage, battery-powered electronic audio devices such as guitar effects, children's toys or small synthesizers to create new musical instruments and sound generators. The heart of this process is "the art of chance".
It began in 1966 when Reed Ghazala by chance, shorted-out a toy amplifier against a metal object in his desk drawer, resulting in a stream of unusual sounds.

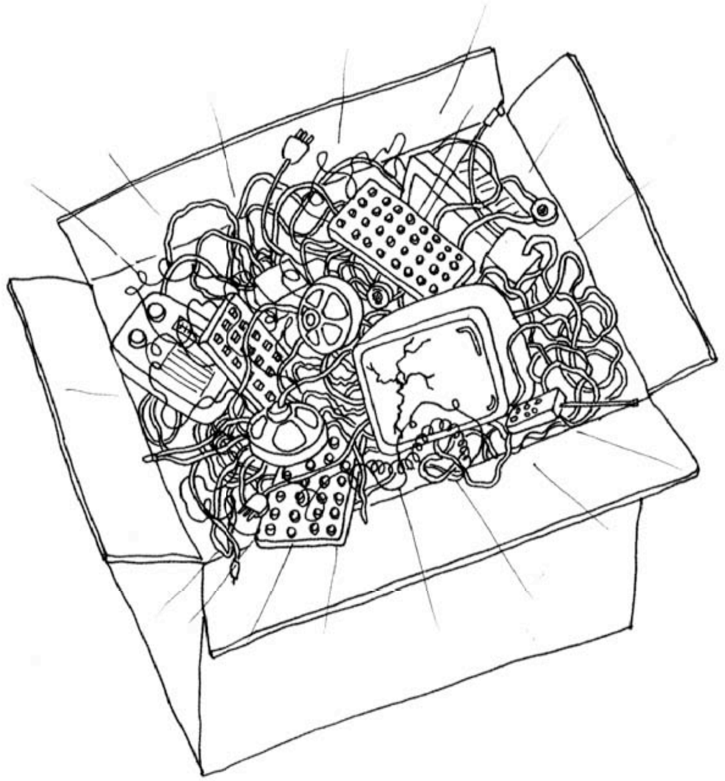
/ keyboard hacks

Taking apart a keyboard reveals a very simple and cheap device.

The heart of it is this small board. By replacing the keyboard matrix with sensors we can implement new ways to interact with software. This is a key hardware component to learn about when starting in Physical Computing.
There is a good example here:
http://www.extremetech.com/article2/0,1697,1610427,00.asp



/ we love junk

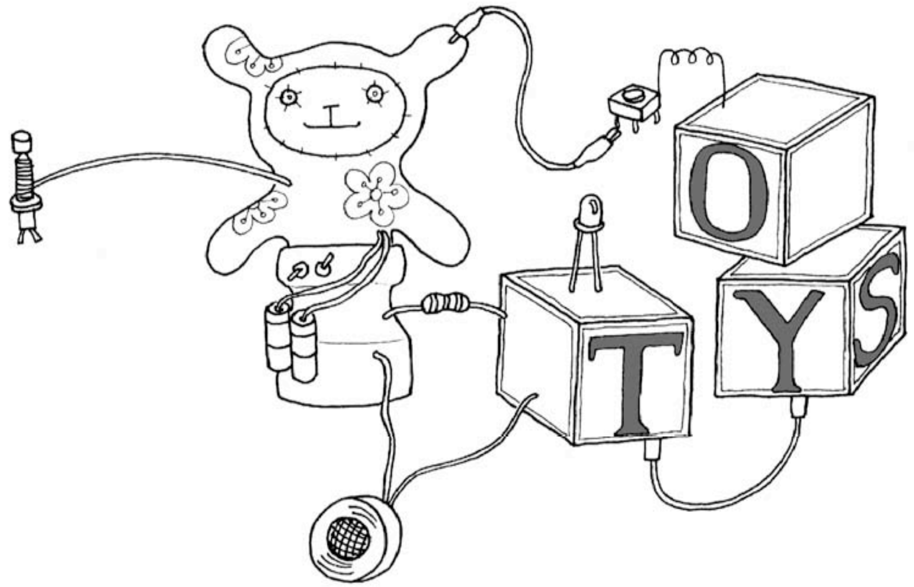


One of the best ways to quickly get to results is to find a great source of technology junk. You can then use it to quickly and cheaply prototype an experience.

Accumulate junk and go through it before starting to build something from scratch.

/ toy hacking

Toys are a fantastic source of cheap technology to hack and reuse. “Lowtech sensors and actuators” from Husman Haque and Adam Somlai-Fisher is a great example of this approach.

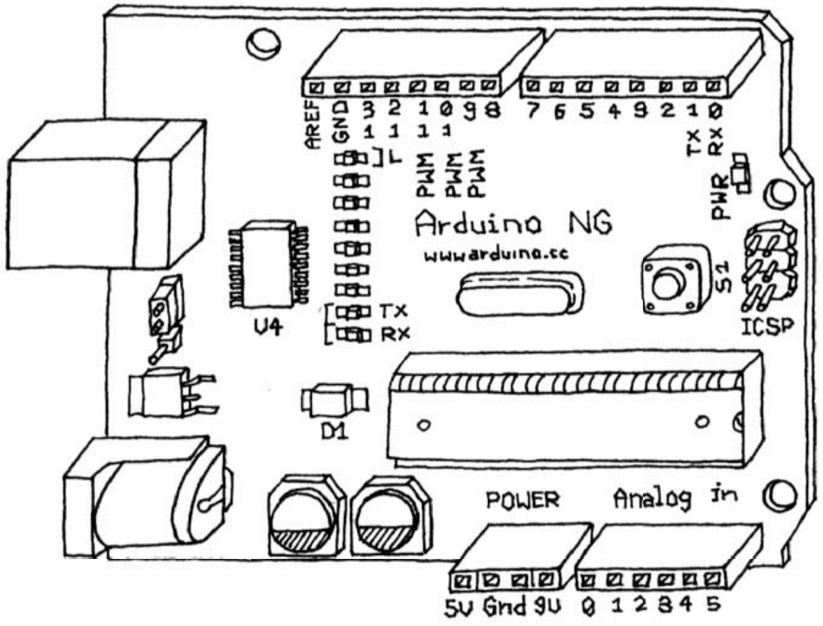


/ collaboration

Collaboration between Arduino users is key. Through the Forum, people from all over the world help each other and share their projects and knowledge while learning about the platform.

The Arduino Team encourages people to collaborate on a local level by helping them to set up user groups in every city they visit.

/the hardware



This is a simple drawing of the Arduino board. Here is an explanation of what every element of the board does:

/ 14 Digital I/O (pins 0 - 13,) can be inputs or outputs as set in software.

/ 6 Analog In (pins 0 - 5) are dedicated analogue input pins. These take analogue values (i.e. voltage readings) and convert them into a number between 0 and 1023.

/3 Analog Out (pins 9, 10, 11) these are actually 3 of the digital pins that can be reassigned to do analog output.

The board can be powered from your USB port or from the power socket. If the jumper marked SV1 is closest to the USB plug then the board is powered from USB. If it is on the 2 pins closest to the DC connector then it is powered through the power socket.

/the software

The last component of Arduino is the software.
This is a special program running on your computer that allows you to write programs for the Arduino board in a simple language (modelled after the Processing language).

The magic happens when you press the button that uploads the program to the board: the code you have written is translated into C language, that is normally quite hard to use for a beginner, and passed on to the avr-gcc compiler (an important piece of open-source software) that makes the ultimate translation into the language understood by the microcontroller.

This last step is where Arduino makes your life simple and hides away the complexities of programming microcontrollers.

Downloading and installing the software

In order to program the Arduino board you need to download the development environment (called IDE for Integrated Development Environment) from here:

/Go to *http://www.arduino.cc/en/Main/Software*

/Choose the right version for your Operating System

/Download the file to your desktop

/Uncompress it

/Install the drivers that allow your computer to talk to your board through the USB port.

Dowload your the drivers on a Macintosh:

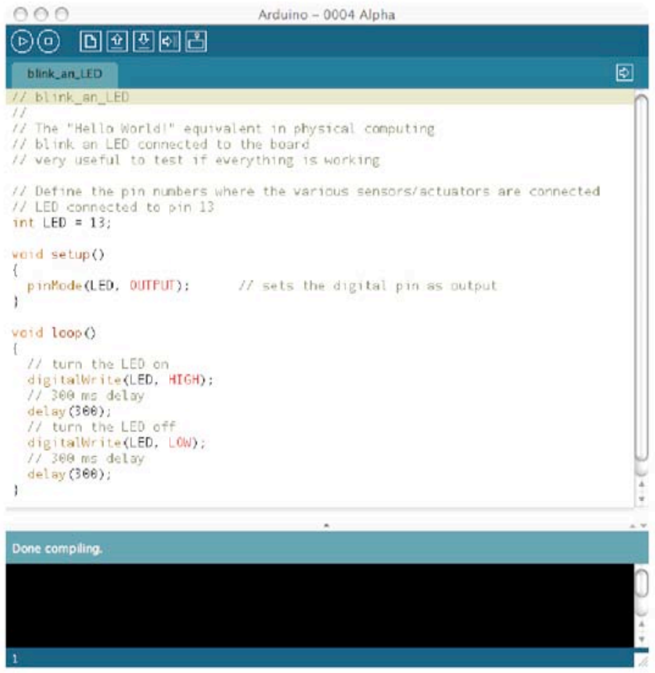
/Look for the “Drivers” folder inside the “arduino-0004” folder
/Double-click on the file called FTDIUSBSerialDriver_v2_0_1.dmg.
/When this has opened, install the software contained in the FTDIUSBSerialDriver.pkg.
/At the end of this process you’ll have to restart your machine to make sure the drivers are properly loaded.
/After the installation is successful you will also need to run the command called “macosx_setup.command”
/Follow the instructions provided by the program and type the password that you use to login into your computer when asked.
/After this program has run successfully you need to turn off your computer. *Don’t just reboot or logout, really turn it off and back on again.*
/When this is done you can plug the board into the computer.

Download the drivers on a Windows machine:

/Unzip the file called [FIXME] contained in the Drivers directory into a directory you can easily find later on.
/Plug the board into the computer and, when the [FIXME] New Device Found [/FIXME] window comes up, specify the location for the install wizard to look for the drivers.
/This will happen twice because the first time the computer installs the low level driver then a piece of code that makes the board look like a serial port.
/Once the drivers are installed we can launch the development environment and start using Arduino.

/the development environment

After the application has come up you will see a window like this one



```
// blink_an_LED
//
// The "Hello World!" equivalent in physical computing
// blink an LED connected to the board
// very useful to test if everything is working

// Define the pin numbers where the various sensors/actuators are connected
// LED connected to pin 13
int LED = 13;

void setup()
{
  pinMode(LED, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  // turn the LED on
  digitalWrite(LED, HIGH);
  // 300 ms delay
  delay(300);
  // turn the LED off
  digitalWrite(LED, LOW);
  // 300 ms delay
  delay(300);
}
```

```
Done compiling.
```

Firstly, you need to find out which port your device is connected to

On a Macintosh:
/From the “Tools” menu select “Serial Port” and select the port that begins with “/dev/cu.usbserial-“.
/The following characters identify which one is the USB port the board is plugged to and change if you plug Arduino into a different port.
[PIC screenshot of the Tools / Serial Port menu showing the list of ports]

On a Windows machine:

/ Opening the "device manger" from:
Start menu -> Control Panel -> System.. -> Hardware -> Device Manager

/ Look for the device in the list under "Ports (COM & LPT)".
Arduino will come up as an "USB Serial Port" and will have a name like COM4.
[PIC screenshot of the device manager showing Arduino]

/ Then in the IDE, select the appropriate port from the
Tools -> Serial Port menu.

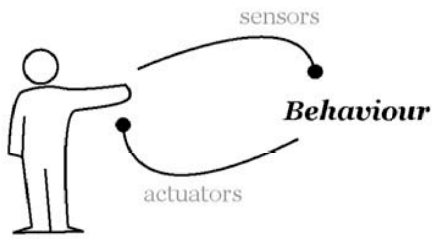
Now the Arduino development environment can talk to the Arduino board and program it.

NB: For some reasons on some windows machine the COM port has a number greater than 9 and this creates some problems when Arduino is trying to communicate with it. Check in the Arduino troubleshooting section of the website how to fix that.

Really getting started

Now that we have introduced you to the Arduino philosophy and its components we are ready to make something happen with your board.
In the following pages we will go through a few examples of the basic things you can do with Arduino. After these exercises you will be ready to experiment on your own.

/ the interactive device
Most of the objects we will build using the Arduino board follow a very simple pattern that we call the "Interactive Device"
It is an electronic circuit that is able to sense the environment using components called "sensors" and processing the information through "behaviour" implemented as software. The device will then be able to interact back with the world using "actuators".



/ sensors and actuators
Sensors and Actuators are electronic components that allow a piece of electronics to interact with the world.
Since the microcontroller is a very simple computer it can only process electric signals (a bit like the electric pulses that are sent between neurons in our brains). In order to enable it to sense light, temperature or other physical quantities it needs something that can convert them into electricity first.
The eye, for example, converts light into signals that get sent to the brain using nerves . In the world of electronics, we could use a simple device called LDR that can measure the amount of light that hits it and reports it back as a signal that can be understood by the processor.
Once the sensors have been read, the device has the information needed to "decide" how to react. This is done through Actuators. These are electronic components that can convert an electric signal into a physical action.Muscles receive electric signals from the brain and convert them into a movement while in the electronic world this functions could be performed by an electric motor.

In the next chapters we will see how to read sensors of different types and control different kinds of actuators.

/basic introduction to programming

If you have porgrammed before you can skip this chapter.

Programming is about translating the behaviour we have in mind for our device into instructions that can be understood by the processor. In reality the processor uses a very detailed and very low-level language so through time "high level" languages have been developed. These are closer to human language.

Let's see how we would go about translating a simple behaviour into a piece of program. A simple program can always be written as sentence:

When it is very dark I want the light to go on and the motor to start turning slowly

We could then rewrite this into "pseudo code" (something that looks more like a program but it's still human language):

```
If light level is less than 50 then
Turn Light on
Turn motor on slow
Loop again
```

We quickly realise that we need two types of programming structures: 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 check for certain conditions and change the course of the program depending on them. So the basic Arduino program looks like this:

```
// This is a comment
// variable declaration
int x;

void setup() {
// put code here
}
void loop () {
// put code here
}
```

At the beginning of the program, we declare variables which are areas of memory used to store data.

Then the void int( ) function is used to setup the program and defines which pins on the processor are inputs and which are outputs.

The last function, void loop( ), will be executed indefinitely until you turn the Arduino board off. This is where all the conditional logic will be stored and it is the core of the program where you can control the flow of the program.

The text beginning with // is a comment which is useful for you to remind yourself what your code does when you re-open it after a while, or for other people who might want to rework your code.

variables
One special thing about programming is that every time you want to store some value you need to use a variable that you have to declare. That is, tell the computer what kind of value to expect. There are a few basic data types that a computer can expect that we will go through here:

int	an integer is a whole number i.e. 1,2,3,5 etc.
byte	an integer number between 0 and 255, this is useful if you need to save memory because it uses only 1 byte of memory. (Remember that the Arduino board has only 1024 bytes of RAM)

For most of the programming you will be doing, these data types will be all you will need.

For more details see: *http://www.arduino.cc/en/Reference/HomePage*

flow control
If [condition] Then
In Arduino, an if statement looks like this:

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

Where the expression could be one (or a combination) 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

(N.B. do not confuse == with the assignment operator = that actually changes the value of the variable to the left of it, this can be very dangerous!)
Statements can be anything you like (including more conditional statements).

for loops

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

A for loop is a way to execute a certain piece of code for a very specific amount of times. In the example shown "statement" is executed 8 times with "i" going from 1 to 8.

delays

```
delay(100) //
```

The delay( ) function is useful for embedded programming to slow down the rate at which the processor updates. This is used to make thing happens at a certain rate, for example if we want to blink and led every second, after we turn it on we can place a delay(1000) which will make the processor sit there and do nothing for a second (1000 milliseconds)
Delay is also useful in debugging and general flow control.

/making an LED blink

This program is the first code to run that tests that your Arduino board is working and configured correctly. Type the following text into your Arduino editor:

```
// Blinking LED -

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

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

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

Now that the code is in your IDE you need to verify it and upload it to the board. Press *"Verify"* and if everything is correct you'll see this message appear at the bottom of the program text:

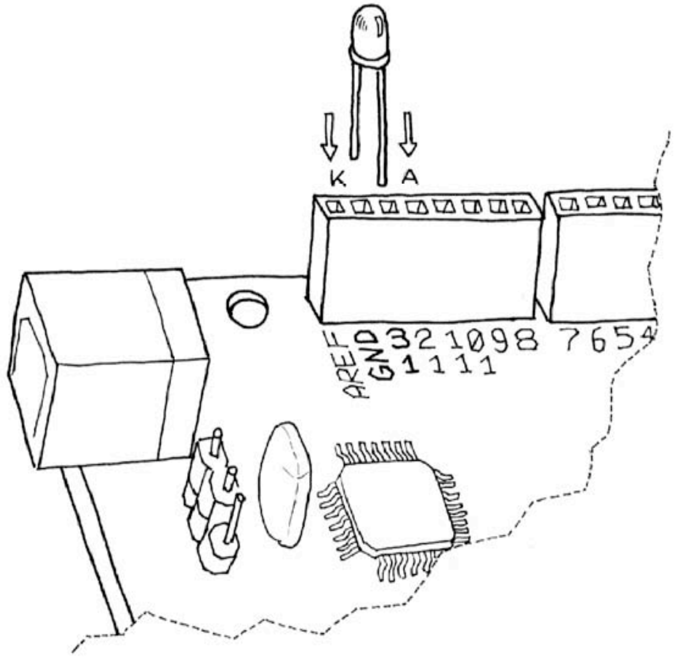
```
Done compiling
```

At this point you can upload it into the board. Press the reset button on the Arduino board, this forces the board to stop what it's doing and listen for instructions coming from the USB port. Now you have about 6 seconds to press the *"Upload to I/O Board"* button.

This sends the current program to the board that will store it in its memory and eventually run it. You will see a few messages appear in the black area at the bottom of the window. These are messages that make it easier to understand if the process has completed correctly.

There are 2 LEDs marked RX and TX on the board, they flash every time a byte is sent or received by the board. During the download process, they keep flickering. If this doesn't happen, it means there are some communication problems or you haven't selected the right port in the "Tools / Serial Port" menu.

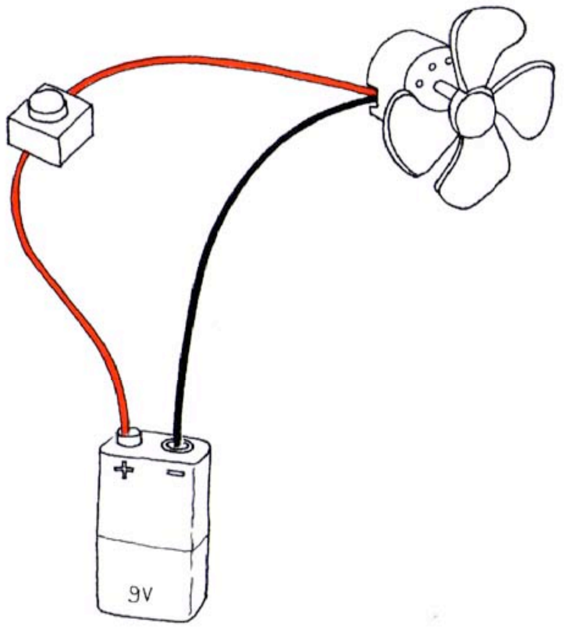
Assuming the program has been uploaded correctly connect an LED to the pins 13 and GND (which means Ground) on the board as seen in this illustration.



/what is electricity?

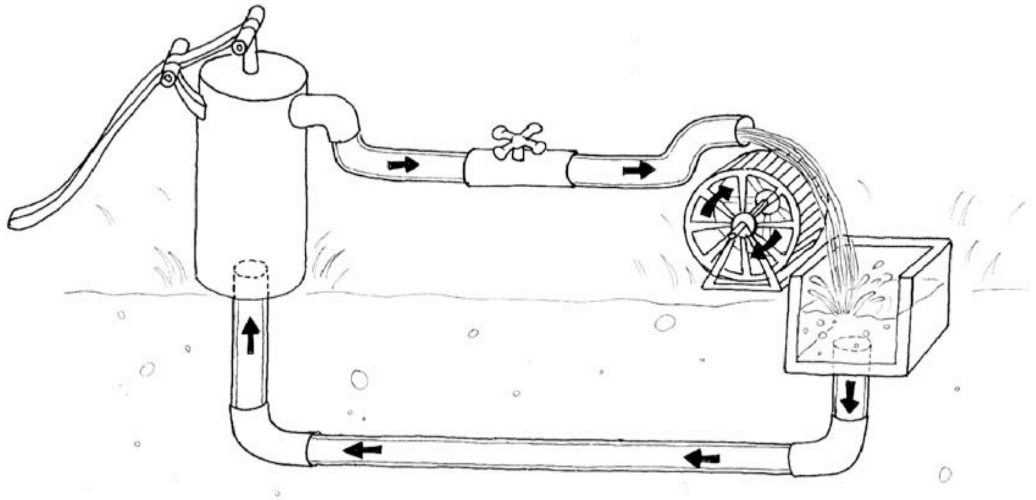
If you've ever done any plumbing at home, understanding electronics won't be a problem for you. In order to understand how electricity and electric circuits work we'll use a mental model called the "water analogy".

To start with, let's look at a simple device like a portable fan.



If you were to take it apart, you'll see that it contains a small battery, a few wires going to an electric motor one of which is interrupted by a switch. Make sure you have a new battery fitted into the device and activate the switch; the motor starts to spin.

How does this work? Imagine that the battery is a water pump and the switch is a tap while the motor is one of those wheels in watermills. When you open the tap, water flows from the pump and pushes the wheel into motion.



In this simple system two parameters are important: the pressure of the water (determined by how powerful the pump is) and the amount of water that can flow in the pipes (determined by the size of the pipes and the resistance that the wheel will oppose to the stream of water hitting it).

If you want the wheel to spin faster, you need to increase the size of the pipes and increase the pressure that the pump can achieve. Increasing the size of the pipes allows more water to flow through them, and by making them bigger we reduce the resistance they oppose to the flow of water. This works until the wheel won't spin any faster because the pressure of the water is not strong enough. This is when we need the pump to be stronger.

In turn, this works until the wheel falls apart because the water flow is too strong and destroys it. You'll notice that as the wheel spins faster, the axle will heat up a little bit. No matter how well we mount the wheel, the friction between the axle and the holes it is mounted in will generate heat.

It's important to understand that in a system like this, not all the energy you pump into the system will be converted into movement. Some of it will be lost in a number of inefficiencies (generally as heat emanating from some parts of the system).

The important pasrts of this system are therefore: the pressure produced by the pump, the resistance that the pipes and wheel oppose to the flow of water and the actual flow of water (represented by the number of litres of water that flow in one second).

Electricity works in a similar fashion: any source of electricity like a battery or a wall plug acts like a kind of pump that pushes electric charges (like "drops" of electricity) down wires (similar to the pipes) where some devices are able to use them to produce heat (your grandma's thermal blanket) light (your bedroom's lamp) sound (your stereo) movement (your fan) and much more.

Furthermore, when you read on a battery "9V" imagine the voltage of the battery like the water pressure that can be potentially produced by this little "pump". This is measured in Volts (from Alessandro Volta, the inventor of the first battery).

The flow of water has an electric equivalent called "current" that is measured in Amperes (from Andre Marie Ampere).

Finally the resistance opposed to the flow of current by any means it travels through is called, "resistance", measured in Ohms (from the German physicist Georg Ohm).

He was also able to demonstrate that in a circuit, the "voltage", the "current" and the "resistance" are all related to each other. The resistance opposed by the circuits determines the amount of current that will flow through it give a certain supply voltage.

Going back to our analogy, given a certain pump and assuming a tap (which we can assimilate to a variable resistor in electricity) the more I close the tap, increasing resistance to water flow, the less water will flow through the pipes. So, if you take a 9V battery and plug it into a simple circuit while measuring current, you will find that the more resistors you add to the circuit the less current will travel through it.

Mr Ohm summarised his law into this formula:

R (resistance) = V (voltage) / I (current)

V = R * I

I = V / R

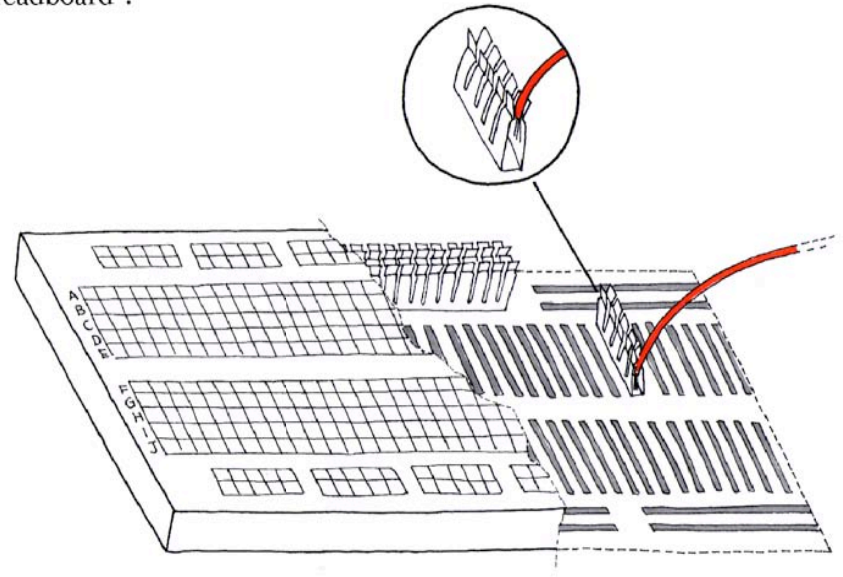
This is the only rule that you really have to memorise and learn to use.

/the breadboard

The process of getting a circuit to work is largely based on making lots of changes to it until it behaves properly; it's a very fast iterative process that could be seen as the electronic equivalent to sketching. The design evolves in your hands as you try different combinations. In order to achieve the best results you need to use a system that will allow you to change the connections between components in the fastest, most practical and non-destructive way.

This requirement rules out soldering, it's a time-consuming process that puts every component under stress every time you heat them up and cool them down.

The answer to this issue is the very practical device called a"Solder-less Breadboard".



It is a small plastic board full of holes, each one of them containing a spring-loaded contact. When you push a component's leg into one of theses holes it establishes an electrical connection with all the other holes in the same vertical column.

Each hole is at a distance of 2.54 mm from the others, since most of the components have their legs, (also known as pins) spaced at that standard distance. This is very practical for chips with multiple legs.

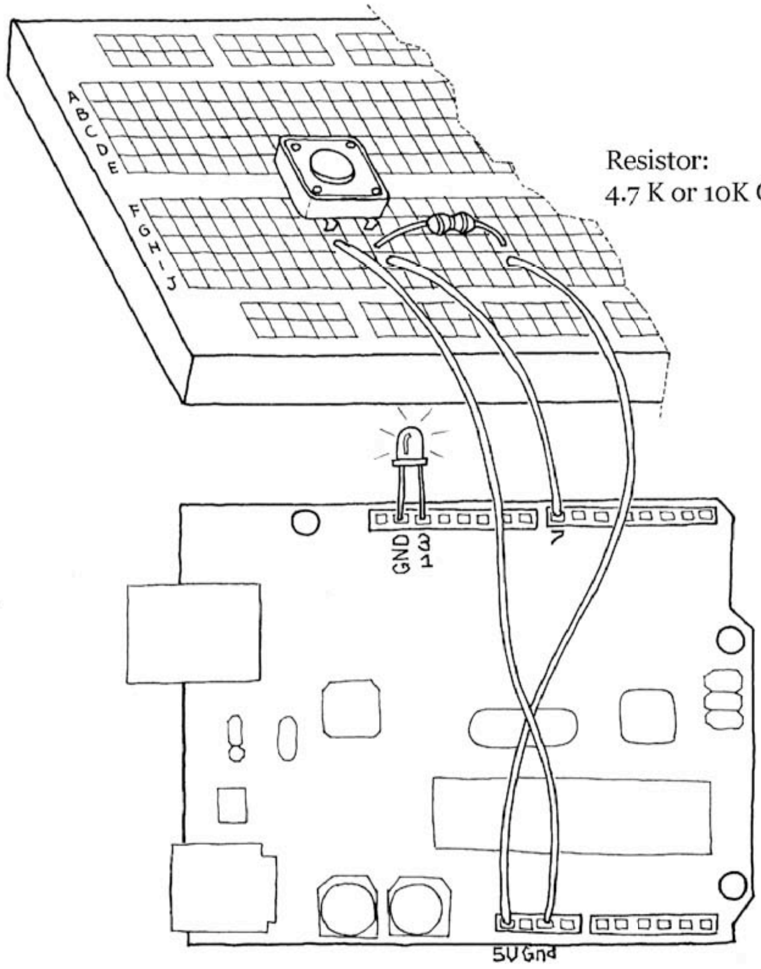
Not all the contacts on a breadboard are created equal. The topmost and bottom row (coloured in red and blue and aptly marked with + and -) are connected horizontally and are used to carry the power across the board so that when we need power or ground we can provide it very quickly with a short jumper (this is not a sweater or a funny insect but a short piece of wire used to connect two points in the circuit).

Additionally, in the middle there is a large gap that is as wide as the size of a small chip. This is to indicate that each vertical line of holes is interrupted in the middle so that when you plug a chip in, you won't short circuit pins that are on the two sides of the chip.

/reading a pushbutton

Now let's see how Arduino receives input from the external world.
This happens with the digitalRead() function which, when given the number of a pin, will tell you if there is a voltage applied to it. If the voltage is 2.5 or more volts, digitalRead() will return HIGH while if there is no voltage it will return LOW.

This code checks the state of the digital pin and turn on the LED if the button is pressed. To get started quickly, build the circuit you see below.



Resistor:
4.7 K or 10K Ohms

```
/* Blink LED when the button is pressed
 * ------------------------
 */

int ledPin = 13; // choose the pin for the LED
int inPin = 7;   // choose the input pin
                 // (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin);     // read input value

  // check if the input is HIGH (button released)

  if (val == HIGH) {
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    // blink the LED and go OFF
    digitalWrite(ledPin, HIGH);
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(1000);
  }
}
```

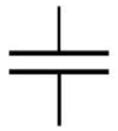
/drawing a schematics diagram

So far, we have used very detailed illustrations to describe how to assemble circuits. However, in the long run, drawings aren't the most efficient way to document your experiments.

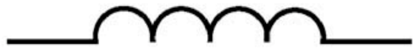
Engineers have developed a quick way to capture the essence of a circuit to be able to document it and rebuild it later on. Schematics diagrams allow you to describe your circuit in a way that is understood by the rest of the community as well.

Individual components are represented by symbols that are a sort of abstration of either the shape of the component or the essence of them.

Here are a few examples:
A capacitor is made of two metal plates separated either by air or plastic therefore its symbol is:



An inductor that is built by winding copper wire around cylindrical shape; consequently the symbol is:

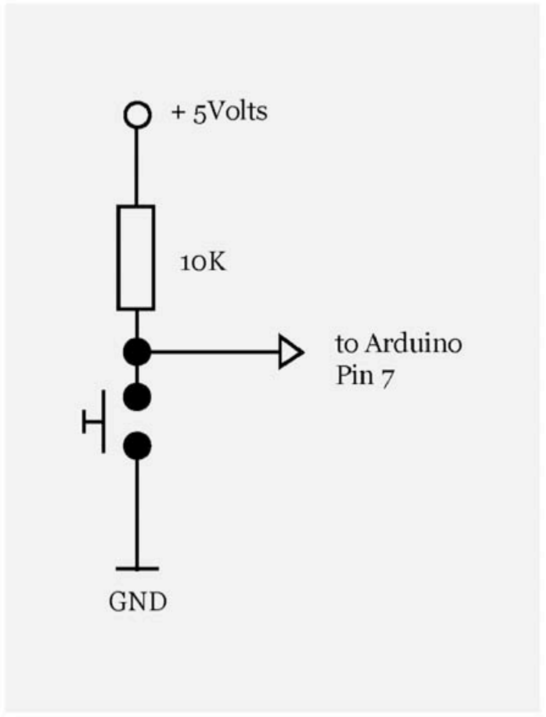


The connections between components are usually made using either wires or tracks on the printed circuit board and they are represented on the diagram as simple lines. When two wires are connected this is represented by placing a big dot where the two lines cross.

This is all you need to understand basic schematics. You will find a comprehensive list of symbols on *http://www.kpsec.freeuk.com/symbol.htm*

The symbols in this booklet are commonly used in Europe. In the USA however, some symbols have a slightly different shape, but are still easily recognisable.

Conventionally, you should always draw diagrams from left to right as shown in this diagram of the pushbutton circuit you have just put together.



/ *other on & off sensors*

Now that you've used the push-button you can replace it with a lot of different sensors that have the same type of output: a contact that closes and opens.

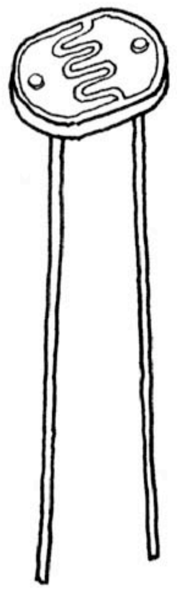
One good example is the tilt sensor. This is a simple electronic component that contains two contacts and a little metal ball.
When the sensor is in its upright position the ball bridges the two contacts and like when you press the push-button. When you tilt this sensor the ball moves and the contact is opened like when you release the push-button. Using this simple component you can implement, for example, gestural interfaces that reacts to when an object is moved or shaken.

Another sensor you might want to try is the infra-red sensor from burglar alarms (also known as PIR sensor)
This small device triggers when a human being is moving in its proximity. It's a simple way to detect the presence of people.
It's good to experiment by looking at all the possible devices that have two contacts that close, like the thermostat that is used in apartments to set the room temperature or just placing two wires next to each others and dropping water onto them to create a connection.

/ *using a light sensor*

This is an LDR or light dependent resistor. In the dark, its resistance is quite high while when it is exposed to light, the resistance quickly drops and it becomes a reasonably good conductor of electricity.

Replace the push-button by the LDR and you will notice that if you cover or uncover it with your hands the LED is turned on and off. You've just built your first sensor-driven LED.



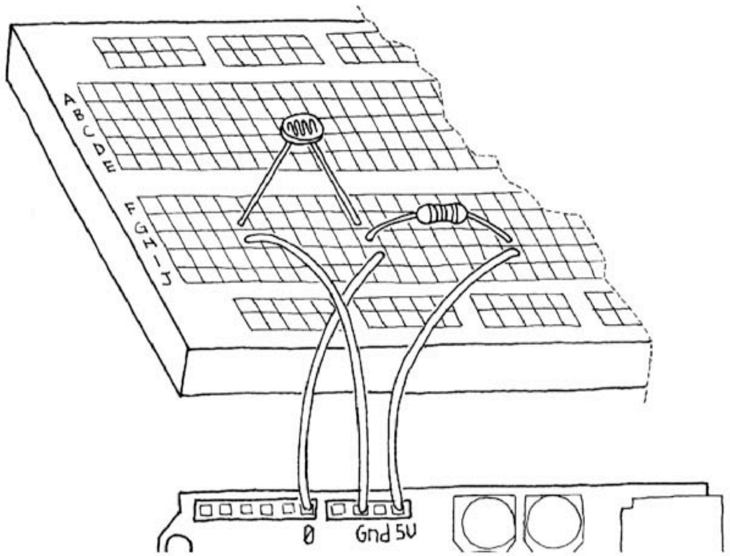
/ analog inputs

As seen in the previous section, Arduino can detect if there is a voltage applied to one of its pins and reports it through the digitalRead function. This works for a lot of applications, but the light sensor that we have just used can tell us not just if there is light or not, but also how much light there is.

This is the difference between an on/off sensor (simply telling us if something is there or not) and an analogue sensor whose value continuously changes. In order to read this type of sensors we need a different type of pin. In the lower-right part of Arduino you'll see 6 pins marked "Analog In", these are special pins that not only can tell us if there is a voltage applied to them or not but also it value.

By using the analogRead() function we can read the voltage applied to one of the pins. This function returns a number between 0 and 1023 representing voltages between 0 and 5 volts. For example if there is a voltage of 2.5 volts applied to pin number 0, analogRead(0) will return 512 etc.

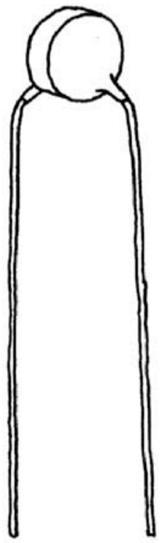
If you now build the circuit that you see in the illustration by using a 10 K resistor and you run the piece of code included you'll see the led blinking at a rate that is dependent on the amount of light that hits the sensor.



```
// Light sensitive LED -

int val = 0;         // variable to store the value
                // coming from the sensor

void setup() {
pinMode(13, OUTPUT);  // ledPin is as an OUTPUT
}

void loop() {
val = analogRead(0);      // read the value from
                     // the sensor
digitalWrite(13, HIGH); // turn the LED on
delay(val);             // stop the program for
                     // some time
digitalWrite(13, LOW);  // turn the LED off
delay(val);             // stop the program for
                     // some time
}
```

/ *other resistive sensors*

Using the same circuit that you have seen in the previous section you can connect a lot of other resistive sensors that work more or less in the same way. For example you could connect a thermistor.



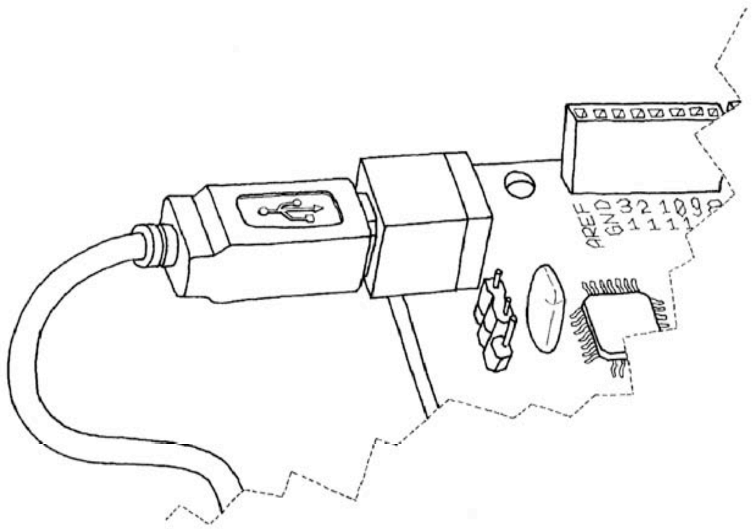
This is a simple device whose resistance changes with the temperature. In the circuit we have shown you, changes in resistance become changes in voltage that can be measured by Arduino.

Be aware that there isn't a direct relationship between the value you read and the actual temperature measured. If you need an exact reading, you should read the numbers that come out of the analogue pin while measuring with a real thermometer. You could put these numbers side by side in a table and work out the relationship between the values.

Until now, we have just used an LED as an output device but how do we read the actual values that Arduino is reading from the sensor? For this type of issue, we can use Serial Communication that is described in the next chapter.

/ serial communication

At the beginning of the booklet, you might have noticed that Arduino has a USB connection that is used by the IDE to upload code to the processor.



Additionally, this connection can also be used by the programs we write in Arduino to send data back to the computer or receive com-mands from it.

For this purpose, we are going to use the Serial object. This object contains all the code we need to send and receive data.
We're now going to use the last circuit we have built using the the thermistor and send the values read back to the computer.

Type this code into a new sketch:

```
int sensorPin = 0;     // select the input pin for the
                       // potentiometer
int val = 0;        // variable to store the value
                    // coming from the sensor
void setup() {
Serial.begin(9600);  // open the serial port to send
                     // data back to the computer at
                     // 9600 bits per second
}
void loop() {
val = analogRead(sensorPin);   // read the value from
                               // the sensor
Serial.println(val);           // print the value to
                               // the serial port
delay(100);
}
```

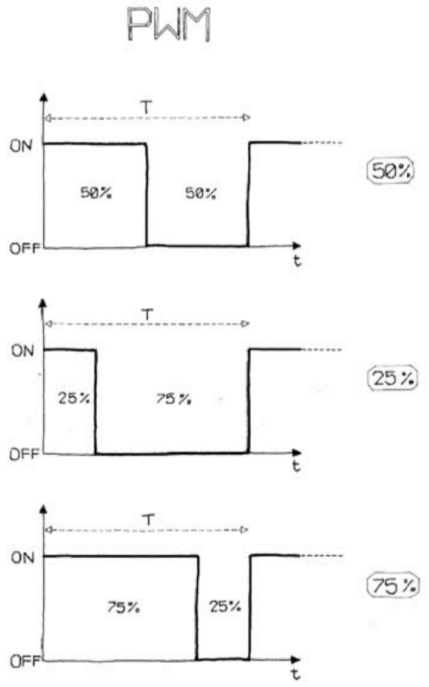
Now that the program is running, you should press the “Serial Monitor” button on the Arduino IDE and you'll see the numbers lining up in the bottom part of the window.
Any software that can read from the serial port can talk to Arduino now.

/ analog outputs

With what you've learnt so far, you could take up the task of building an interactive lamp, a lamp that can be controlled not just with an on-off switch but maybe with a little poetry.
One of the limitations of the blinking LED examples we have seen so far is that you can only turn on and off the light while a fancy interactive lamp needs to be able to dim the light properly. To solve this problem, we can use a trick that is used in TV or cinema: persistance of vision.

In the first LED blinking example you made, change the numbers in the delay function until you can't see the LED blinking anymore.

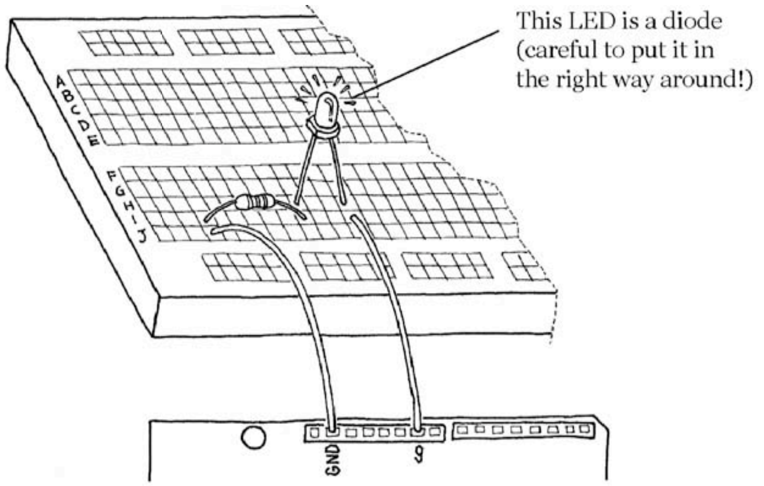
You'll observe that the LED seems to be dimmed at 50% of its normal brightness. Now change the numbers so that the time the LED is on is one forth of the time it's off. Run the program and you'll see the brightness is roughly 25%.
This technique is called Pulse Width Modulation (PWM), which means that that if you blink the LED fast enough you don't see it blink anymore but you can change its brightness by changing the ratio between the on time and the off time. Here is a small diagram showing how this works.



This technique also works with other devices. For example you can change the speed of motor in the exact same way. While experimenting, you can see that blinking the LED by hand is slightly inconvenient because as soon as you want to read a sensor or send data on the serial port, the LED will flicker.
Fortunately, the processor used by the Arduino board has a piece of hardware that can very efficiently blink 3 LEDs while your program does something else. This is implemented by pins 9, 10 and 11 that can be controlled by the analogWrite() instruction.

Writing analogWrite(9,128) will set to 50% the brightness of an LED connected to pin 9. Why 128? because analogWrite expects a number between 0 and 255 as a parameter where 255 means 100%.

Having 3 channels is very good because if you buy red, green and blue LEDs you can make light of any colour you like!


This LED is a diode (careful to put it in the right way around!)
A
B
C
D
E
F
G
H
I
J
GND
9

/ *driving bigger loads*

Each pin on an Arduino board can be used to power devices that use up to 20milliamps which is a very small amount of current, just enough to drive an LED and not much more.

If you try to drive something like a motor the pin will immediately stop working and will potentially burn the whole processor.

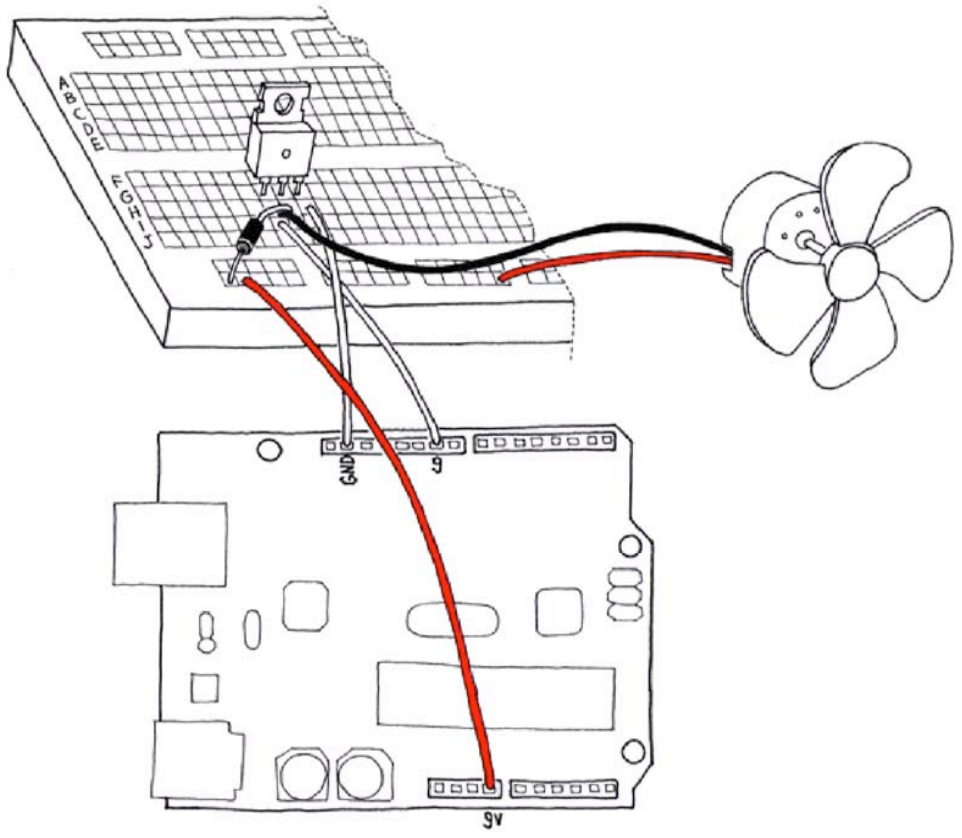
To drive bigger loads like motors or incandescent lamps, we need to use an external component that can switch such things on and off while being driven by an Arduino pin. One of such devices is called a MOSFET Transistor which is an electronic switch that can be driven by applying a voltage to one of its three pins called "gate".

It can be seen as the light switch we use at home where the action that our finger does to turn the light on and off is replaced by a pin on the Arduino board sending voltage to the "gate" of the MOSFET.

In the following picture you see how you would use a MOSFET like the IRF520 to turn on and off a small motor attached to a fan.

You will also notice that the motor takes its power supply from the 9V connector on the Arduino board, which is another benefit of the MOSFET: it allows us to drive devices that have a power supply different from the one used by Arduino.

Since the MOSFET is connected to pin 9 we can also use analogWrite() to control the speed of the motor through PWM.

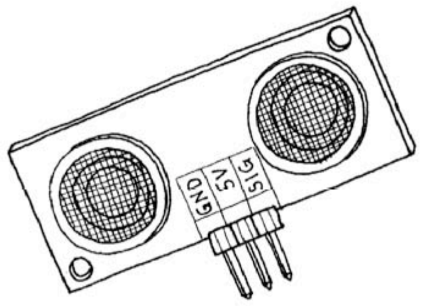


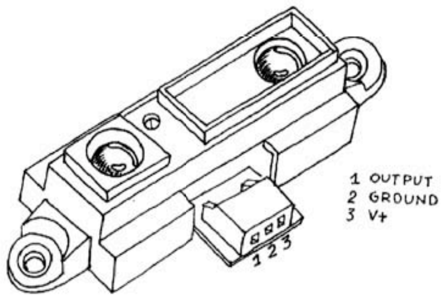
/ *complex sensors*

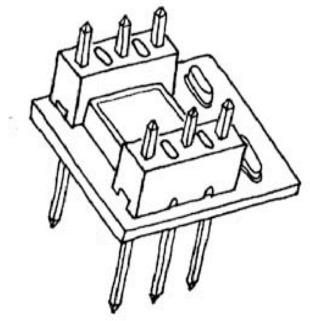
We define "complex sensors" as the ones that produce a type of information that requires more than a digitalRead() or an analogRead() to be used. These are usually small circuits that have already a small microcontroller inside that pre-processes the information.

We've illustrated 3 types of sensors below: an Ultrasonic Ranger, an Infrared Ranger and an Accelerometer.

You can find examples on how to use them at
http://www.arduino.cc/en/Tutorial/HomePage







/ talking to software

In this section we will show you how to use the data sent by an Arduino board through the USB port to a piece of software running in your computer.

```
 /* Virtual Etch A Sketch using 2 Potentiometers
This program reads two analog sensors via serial
and draws their values as X and Y
Processing Code by Christian Nold, 22 Feb 06
*/
import processing.serial.*;
String buff = "";
String temp = "";
float temporary = 0.0;
float screenwidth = 0.0;
float xCoordinate = 0;
float yCoordinate = 0;
int val = 0;
int NEWLINE - 10;
Serial port;
void setup()
{
size(200, 200);
strokeWeight(10); // fat
stroke(255);
smooth();

port = new Serial(this, "/dev/cu.usbserial-A4001qym", 9600);
}
// Change this to the correct serial port

void draw()
{
fill(0,2); // use black with alpha 2
rectMode(CORNER);
rect(0,0,width,height);
```

```
while (port.available() > 0) {
serialEvent(port.read());
}
point(xCoordinate, yCoordinate);
}
void serialEvent(int serial)
{
if(serial != NEWLINE) {
buff += char(serial);
}
else {

if (buff.length() > 1) {
buff = buff.substring(0, buff.length()-1);
int p1 = buff.indexOf(","); // find comma

// put everything before the comma into a temp variable
temp = buff.substring(0, p1);
// turn that string into a floating point number
temporary = float(temp);

// calculate x position
xCoordinate = width/(1024/temporary);
// print it as a debugging
println(xCoordinate);

// do the same with the test after the comma
temp = buff.substring(p1 + 1, buff.length());
temporary = float(temp);


yCoordinate = height/(1024/temporary);
println(yCoordinate);
}

// Clear the value of "buff"
buff = "";
}
}
```

Find out more

Apart from the obvious Arduino site, here are a few sites that might be useful as you get deeper into the world of interaction design and physical computing.

http://todbot.com/blog/

http://www.avrfreaks.net/

http://del.icio.us/popular/arduino

http://www.ladyada.net/make/

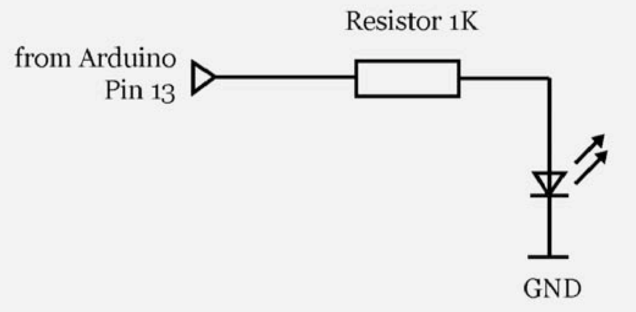
http://www.arduino.cc/en/Main/ArduinoBoardLilyPad

Appendix

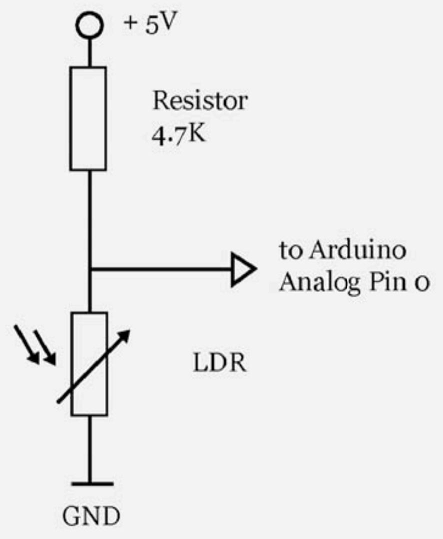
A: schematics diagrams

You'll find here all the schematics diagrams for all the cuircuits built in the booklet.

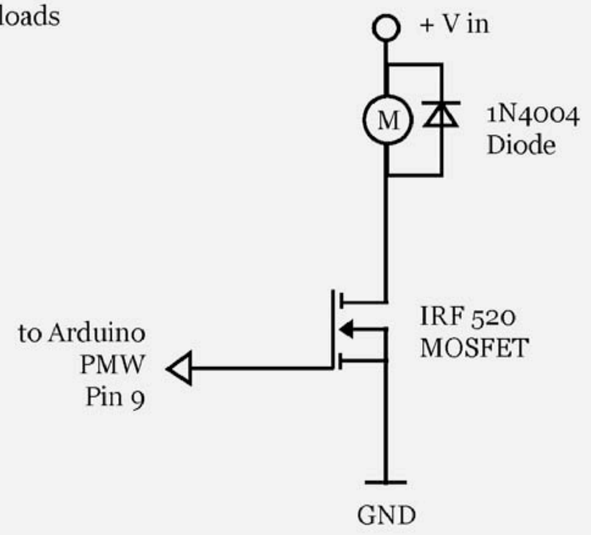
/blinking an LED



/using a light sensor



/driving heavy loads



B: *reading resistors & capacitors*

In order to use electronic parts, you should be able to identify them which can be difficult task for a beginner. Most of the resistors you will find in shops have a body with two legs sticking out and coloured markings all around them.
When the fist commercial resistors were made, there was no way to print numbers small that they would fit on their body so engineers decided that they could just represent the values with strips of coloured paint.

Today's beginners have to figure out a way to interpret these signs.

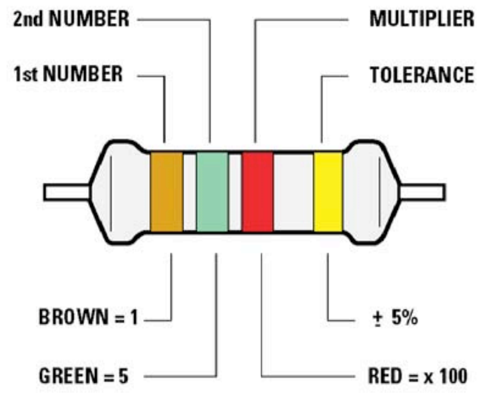
The rule is quite simple, generally you will find 4 strips and each colour number. One of rings is usually "gold" which represents the precision of that place that to the right hand side. read the colours and map them to the numbers.

For example brown, black, orange and gold markings mean 103 ±5%. Easy right? quite, because there is a twist: the third ring represent the number of zeroes in the value therefore 103 is actually 10 plus 3 zeros so the end result is 10 000 Ohms Electronics geeks tend to shorten values by expressing them in Kilo Ohm Ohms) and Mega Ohms (millions of Ohms) so a 10000 ohm resistor is usually to 10k. Please note that, since engineers are fond of opimising everything, on schematic diagram you might find values expressed as 4k7, this means 4.7 kilo 4700.

With the following table, you'll be able to find a translation between the colours and their numeric values.

You can also find a widget to do this for Macs:
http://zambetti.com/projects/resistulator/

	1st NUMBER	2nd NUMBER	MULTIPLIER	TOLERANCE	
BLACK	= = = =	⬛ 0	⬛ x 1	10 % ⬜	SILVER
BROWN	🟫 1	🟫 1	🟫 x 10	5 % 🟨	GOLD
RED	🟥 2	🟥 2	🟥 x 100		
ORANGE	🟧 3	🟧 3	🟧 x 1.000		
YELLOW	🟨 4	🟨 4	🟨 x 10.000		
GREEN	🟩 5	🟩 5	🟩 x 100.000		
BLUE	🟦 6	🟦 6	🟦 x 1.000.000		
PURPLE	🟪 7	🟪 7	🟨 GOLD : 10		
GREY	▨ 8	▨ 8			
WHITE	⬜ 9	⬜ 9			



1 5 00 = 1 500 Ohms ± 5%